Térbeli indexelési megoldások a Microsoft SQL Server adatbázis-kezelőben

Ismeretes, hogy az indexelés célja elsősorban az, hogy bizonyos információk megtalálását segítse: a könyvekben a tárgymutatók, a telefonkönyvekben, szótárakban az ábécé szerinti rendezés is egyfajta indexelésnek tekinthető.

Az adatbázis-kezelő rendszerekben a kezdetektől fogva létezik ilyen mechanizmus, ha más indexek nem is, de valamilyen elsőleges kulcs szerinti fizikai rendezés formájában. Mára magától értetődő az is, hogy az elemi adattípusokon, mint a számszerű típusok, szöveges típusok és dátumok, további indexeket lehet létrehozni, amelyek a rekordok fizikai elrendezését nem befolyásolják, csupán a kívánt rekordok megtalálását segítik, gyorsítják. Ezt két féleképpen teszik: egyrészt, ha a lekérdezés valamely tábla olyan oszlopaira vonatkozik, melyek mindegyikére létezik egy közös index, akkor az adatok kinyerhetők közvetlenül az indexből, a táblát nem szükséges beolvasni, ami teljesítmény-javulással jár, hiszen többnyire bizonyos oszlopok tartalmát nem kell beolvasni, valamint a konkurens írás-olvasás műveletek is kevésbé várakoztatják egymást (pl. ha egy író művelet a táblát épp módosítja, de az még nincs jóváhagyva – commit-olva -, akkor az ezzel egyidejűleg futó olvasó műveletnek a még módosítatlan adatokat kell visszaadnia, azokat pedig kinyerheti az indexből). Másrészt, maguk az indexek mutatókat tartalmaznak azokra a rekordokra, amelyek az index egyes bejegyzéseiben szereplő értéket/értékeket tartalmazzák. Mindezt úgy, hogy azok rendezetten szerepelnek az indexben, így rendezettségen alapuló algoritmusokkal lehet a megfelelő rekordokat megkeresni, amelyek hatékonyabbak azoknál, amelyek nem feltételeznek rendezettséget.

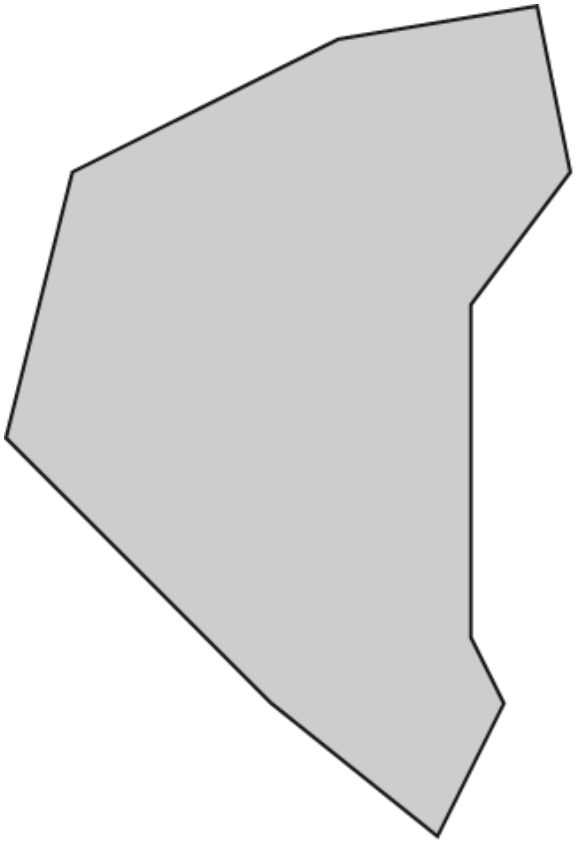
Látszik ugyanakkor, hogy mindez azért tud működni, mert az egyszerű adattípusokon magától értetődő módon állapítjuk meg a rekordok egymáshoz viszonyított sorrendiségét: a számszerű adatokat a legkisebbtől növekedve a legnagyobb felé, a szöveges adatokat ábécé szerint (kiegészítve valamilyen konvencióval a számokra és a speciális karakterekre), a dátumokat pedig kronologikus sorrendben rendezhetjük. A térbeli adatokkal más a helyzet, ott nem egyértelmű, hogy mi szerint történhetne rendezés, hiszen még ha ki is tűntetünk egy origót, akkor is legkevesebb két paramétertől (x, y) függ az, hogy mi hol van ehhez az origóhoz képest. Mit lehetne akkor csinálni, ha két pontnak a (pl. euklideszi) távolsága megegyezik, de eltérő x, y értékek szerint?

Ezt a kérdést jobban meggondolva rájöhetünk, hogy térbeli adatokkal dolgozva valójában többnyire nem is arra vagyunk kíváncsiak, hogy hogyan lehetne azokat sorba rendezni, sőt, nem is arra, hogy hogyan lehetne őket egy origó köré besorolni. Sokkal inkább arra, hogy mi ezeknek az egymáshoz való viszonya: ha a GPS-emet nézem, és egy 480*800 pixel felbontású kijelzője van a készülékemnek, a nagyítás szintje N, a pozícióm pedig (X,Y,Z), akkor milyen objektumok esnek a képernyő által megjelenített területet reprezentáló téglalapba? Vagy Perth mely szállodái esnek az óceántól 300 m-nél nem messzebb, melyekben egy vendégéjszaka nem drágább, mint AU$ 200? Nyilvánvaló, hogy pl. ha az adatbázisunk a Föld – vagy akár csak a kontinens – összes szállodáját tartalmazza (vagy legalábbis jelentős hányadát), akkor a rekordoknak elhanyagolható hányada lesz csak benne az eredményhalmazban, a többire nincs szükség.

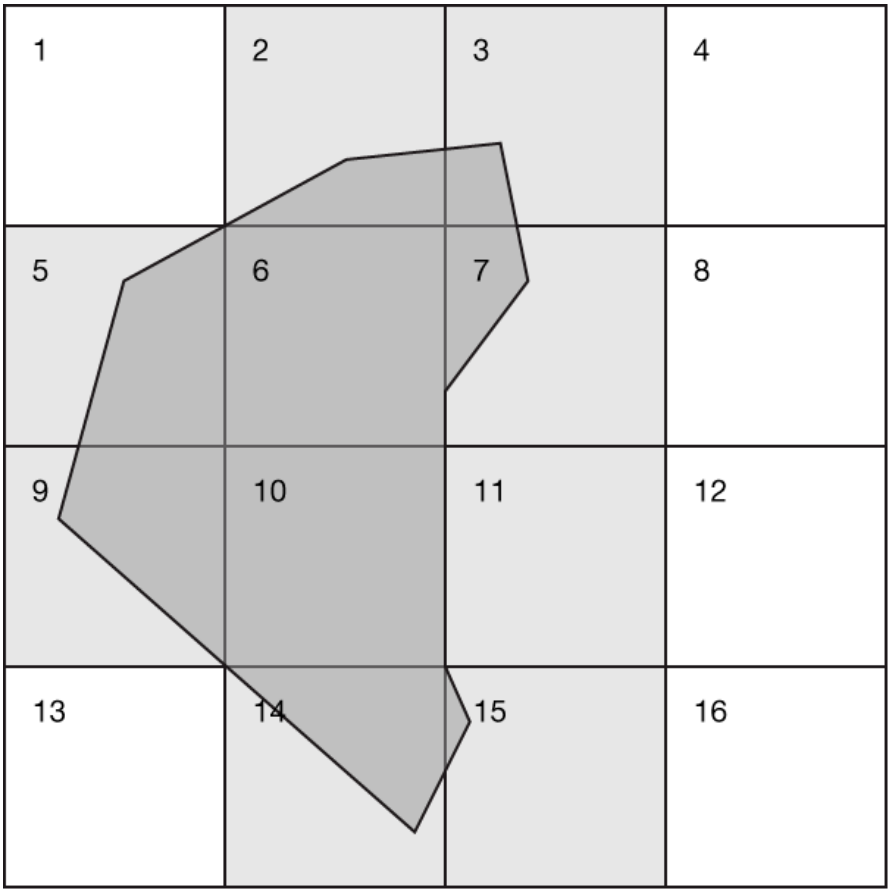
Tehát inkább pontok, vonalak, poligonok egymáshoz való viszonyát vizsgáljuk: átfedik/metszik-e egymást, tartalmazza-e egyik a másikat, érintik-e egymást, vagy teljesen külön állnak egymástól? Mekkora a távolságuk? – ezek a jellemző kérdések, és sok segítséget nyújtana egy olyan megoldás, amellyel – az előbbi példa alapján – eldobálhatjuk azokat a rekordokat, amelyekről nyilvánvaló, hogy nem lesznek benne a kimeneti adathalmazban.

A Microsoft SQL Server így a térbeli lekérdezések elvégzése során két fázist hajt végre. Az első fázis, az elsődleges szűrés (primary filter) egy „előválogatás”, aminek a célja éppen az, hogy a nyilvánvalóan nem megfelelő rekordokat eldobálja. Ez a fázis eldobálja azokat a rekordokat, amelyek biztosan nem lesznek benne az eredményhalmazban, azonban jellemzően nem mindet. Ugyanakkor biztosan megtart minden rekordot, amelynek benne kell lennie az eredményhalmazban. A második fázis, a másodlagos szűrés (secondary filter), az eredményhalmaz tényleges meghatározása, azaz lényegében a térbeli lekérdezésben szereplő számítások elvégzése, ami az első fázisnak köszönhetően várhatóan lényegesen kisebb adathalmazzal dolgozik, mint az első fázis lefutása nélkül.

Az érdekes tehát az első fázis. A MS SQL Server a térbeli adatokat úgy indexeli, hogy azokat egy adott méretű rácshálózatba sorolja be. A valódi implementáció a Hilbert-görbéken alapul, a szemléletesség kedvéért az ábrák azonban sorfolytonos számozást használnak. A következő poligon szemlélteti a továbbiakban leírtakat:

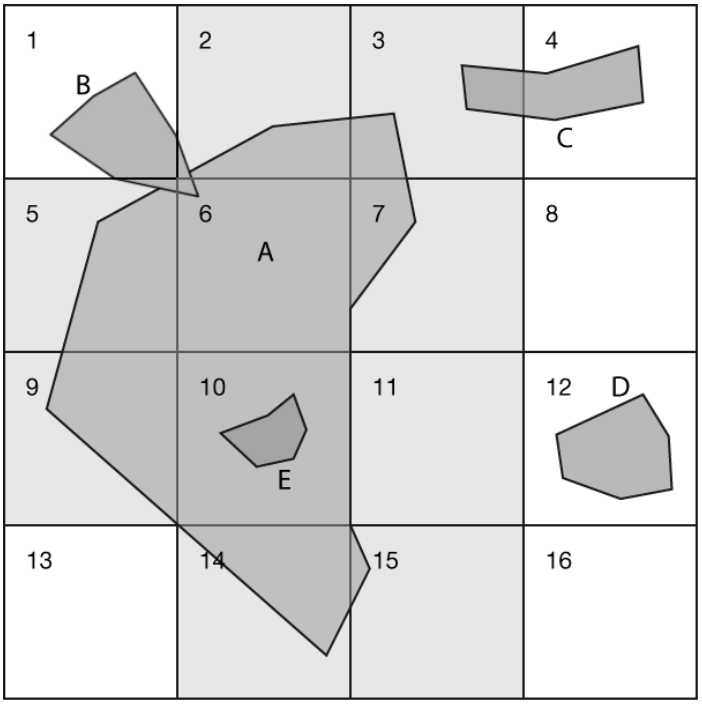


A lenti ábra ennek a poligonnak egy 4x4-es rácshálózaton való elhelyezését mutatja:



Eszerint a térbeli index úgy épül fel, hogy az egyes térbeli elemek (pontok, vonalak, poligonok, továbbiakban *alakzatok*) és a rácshálózat cellái közti viszonyokat írja le. Például a fenti ábrán a rácshálózat (továbbiakban

grid) 2, 3, 7, 9, 14, 15 számú celláit metszi a poligon, érinti a 11-es cellát, és teljesen lefedi a 6-ost és a 10-est. A többi cellával nem áll kapcsolatban. Ez meglehetősen sok információval szolgál. Nézzük például a következő ábrát:

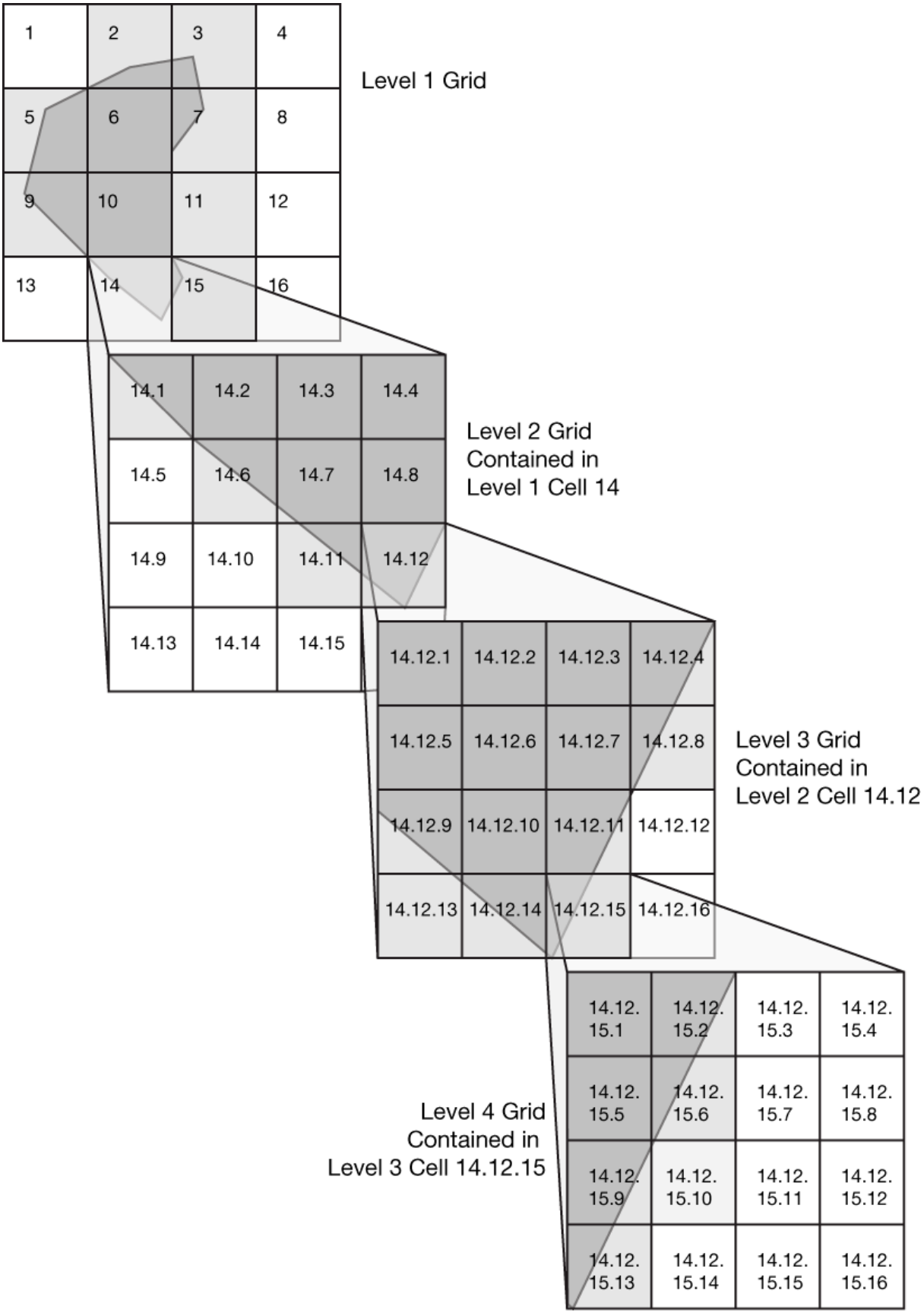


Pusztán annak az információnak a birtokában, hogy az A, B, C, D, E alakzatok mely cellákkal milyen kapcsolatban állnak, számos következtetést levonhatunk, ha azt vizsgáljuk, hogy az A alakzat mely egyéb alakzatokat metszi:

- A B alakzatnak egy része a 6-os cellába esik, amelyet teljesen lefed az A alakzat. Így az A és B alakzatok biztosan metszik egymást.
- A C alakzat részben a 3-as és részben a 4-es cellában helyezkedik el, de egyiket sem fedi le teljesen. Az A alakzatnak szintén esik egy része a 3-as cellába, de ugyancsak nem foglalja el egészben, így az A és C alakzatok közti viszonyt a második fázisnak kell pontosan meghatároznia.
- A D alakzatnak a 12-es cellán kívül nincs kapcsolata semmivel, amivel azonban az A alakzatnak nincs kapcsolata, így az A és D alakzatok biztosan nem érintkeznek.
- Az E alakzat a 10-es cellába esik, más cellával nem áll kapcsolatban, a 10-est azonban az A teljesen lefedi, így az A alakzat tartalmazza az E-t.

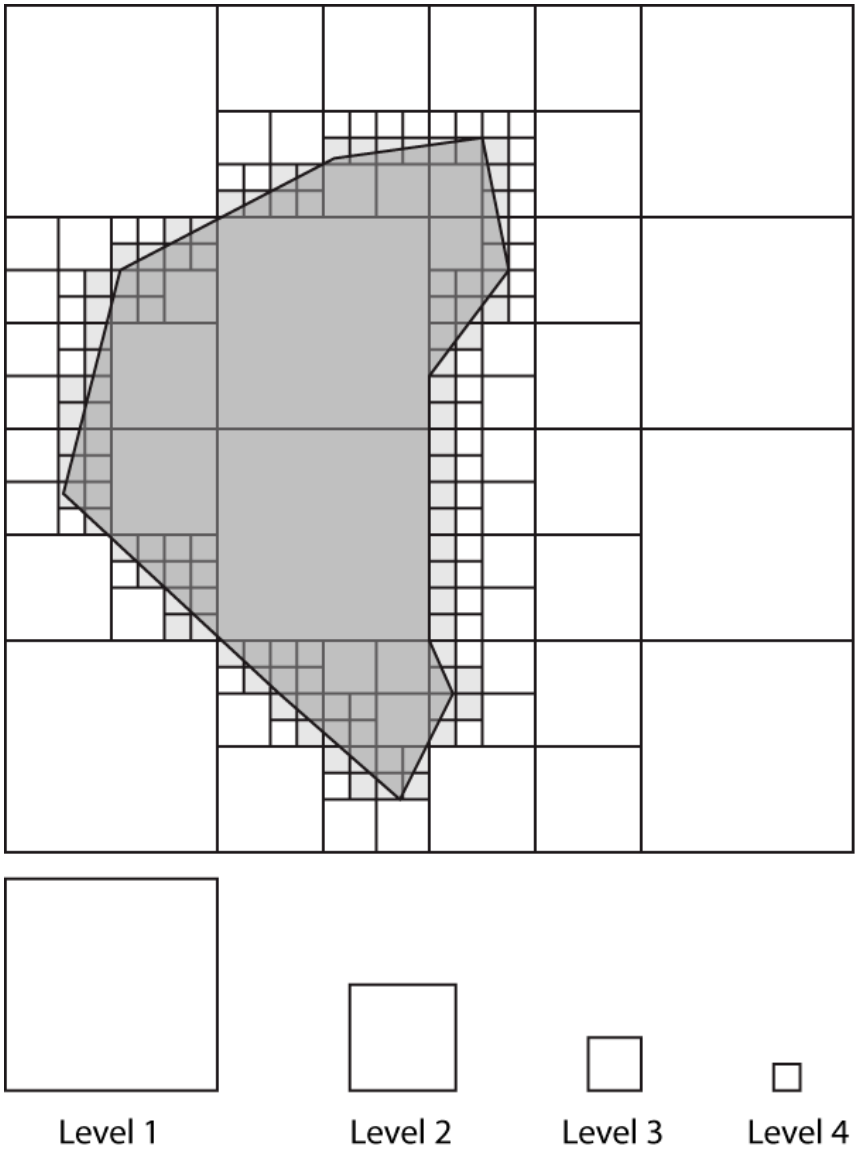
Ezek alapján a D kizárható, a B és E egyértelműen eleme az eredményhalmaznak, a C-vel kapcsolatban pedig precíz számítások szükségesek. Visszatérve a szállodás példához, ha a Perth-ben, Ausztráliában keresünk szállodákat, és az adatbázisunk a Föld összes szállodáját tartalmazza – vagy legalábbis tartalmaz sok szállodát a Föld minden lakott területéről – akkor nagyon könnyen megszabadulhatunk a rekordok nagy részétől. Nyilvánvaló ugyanis, hogy még egy ilyen alacsony felbontású grid használatával sem fog egyetlen szálloda sem egy cellába esni egyetlen európai, ázsiai vagy afrikai szállodával sem, így esetleg pár dél-amerikai, és egyéb, Ausztrália-Óceánia térségbe eső szállodával kell csak tovább dolgoznunk.

Az előbbi példából látszik, hogy sokszor már ez a megoldás is nagy segítséget nyújt, de még pontosabbá lehet tenni az előszűrést. Az MS SQL Server erre még két finomítási módszert kínál. Az egyik, hogy valójában nem egy, hanem 4 grid réteg áll rendelkezésre, a másik, hogy minden réteg felbontása külön szabályozható az alacsony (4x4), közepes (8x8), nagy (16x16) felbontások közül. Ezek felett van lehetőség egy „auto grid”-nek nevezett opció használatára, amelynek kiválasztása esetén nem 4, hanem 8 rétegbe vannak sorolva az adatok, ahol az első réteg 16x16-os felbontású, a többi pedig 4x4-es. A rétegezés minden esetben azt jelenti, hogy az i+1-edik réteg az i-edik réteg *egyetlen celláját* osztja tovább, nem pedig azt, hogy az egész teret egy finomabb felbontásban. Eszerint, ha mind a négy réteg nagy felbontású, akkor végül 256^4 darab cellába sorolhatjuk az elemeket ami ≈4,294 milliárd cellát jelent. A rétegezési szerkezetet a következőképpen lehet elképzelni egy minden szinten 4x4-es grid esetén:



A korábbiakban említésre került, hogy pusztán abból, hogy mely alakzatok mely cellákkal állnak kapcsolatban, le lehet vonni következtetéseket a különböző alakzatok közti viszonyokról, ugyanakkor le lehet vonni következtetéseket az egyes grid szintek és az alakzatok kapcsolatáról is. Ilyen következtetéseken alapul két optimalizációs szabály, amely tárhely- és műveletigény-optimalizációt hivatottak segíteni. Ezek a „lefedési szabály” (covering rule), illetve a „legmélyebbi cella szabály” (deepest cell rule). Ezenkívül még egy optimalizációs szabály létezik, ám ez pusztán egy korlát, ez a „maximális cellaszám objektumonként” szabály (cells per object rule).

A lefedési szabály azt az egyszerű tényt használja ki, hogy amennyiben valamely szinten egy alakzat egy cellát teljesen lefed, akkor egyben lefedi ugyanannak a cellának az összes, lentebbi szinteken elhelyezkedő felosztását is (amiből nyilván több van, hacsak nem a legalsó szintről van szó), így fölösleges a lentebbi szinteken lefedett cellákat is nyilvántartani, az implicit módon adódik. Ezt a szabályt szemlélteti a következő ábra:



A legmélyebbi cella szabály azt mondja ki, hogy ha egy grid cella és egy alakzat között részleges fedési kapcsolat van, akkor elegendő a legalsó olyan grid cellákat feljegyezni, ahol a részleges átfedés áll fenn, a fentebbi szinteken lévő cellákat nem szükséges feljegyezni. Ez azért lehetséges, mert az i+1. szint bármely cellájáról egyértelműen tudjuk, hogy az az i. szinten mely cellának az eleme. A rétegezést példázó ábrát felhasználva, ha pl. a 14.12.15.13-as celláról már tudjuk, hogy részleges átfedés áll fönn a poligonnal, akkor tudjuk, hogy a fentebbi szint megfelelő cellájában, azaz a 14.12.15-ös cellában is ez a helyzet, és ennek megfelelően a 14.12-es cellában, és végül a 14-es cellában is.

Végül, a maximális cellaszám objektumonként szabály egy felső határt szab arra, hogy egy alakzat esetén legfeljebb hány cellát tartunk nyilván az indexben. E szabály egyetlen esetben azonban figyelmen kívül hagyásra kerül: ha már a legfelső szinten is meghaladná a nyilvántartandó cellák száma ezt a számot, akkor nincs mit tenni, túl kell lépni ezt a korlátot. Ennek a paraméternek 1 és 8192 között lehet az értéke, és az index létrehozásakor kell beállítani. Az alapértelmezett érték 16. Ha az érték a grid továbbosztásával túl lenne lépve, akkor ez a szabály felülbírálja a legmélyebbi cella szabályt. Ennek megfelelően, ha egy, nem a legalsó szinten részleges átfedés van, akkor a lentebbi szintekre vonatkozó információk nem lesznek eltárolva.

Mindezeken felül a térbeli indexeket még egy paraméter szabályozza. Ez a bounding box paraméter. Ez egy téglalapot határoz meg, amely az első szintű grid kiterjedése. Fontos kiemelni, hogy eshetnek értékek e téglalapon kívül is, csak azok nem lesznek indexelve. Ilyenkor az MS SQL úgy viselkedik, hogy ha a grid celláin alapuló heurisztikákkal nem tudja megállapítani, hogy egy, a bounding box-on kívül eső alakzattal mi a viszonya

a lekérdezés tárgyának, akkor azt egyszerűen átadja a második fázisnak precíz számítások elvégzésére. Az, hogy a bounding box-on kívül eső adatok is előfordulhatnak, valójában egy hasznos tulajdonság, hiszen sok esetben nem tudjuk előre az adatok eloszlását, így a lehető legkisebb és legnagyobb koordinátákat sem. Sőt, ha tudjuk is, előfordulhat olyan adathalmaz, melyek csak egy részét célszerű indexelni. Például, ha az adatok minden koordinátája a [0, 255] tartományból kerülhet ki, és tudjuk, hogy eloszlása olyan, hogy a (128, 128) körül erősen csomósodik, akkor érdemes szűken e csomósodási pont köré feszíteni az indexelt tartományt. Ily módon az adatok nagy részét finomabban osztjuk szét a cellák között még egy alacsonyabb felbontású grid használatával is, és az adatoknak csak egy kis része kerül az indexelt tartományon kívülre. A bounding box mérete csak geometry adattípus indexelése esetén állítható be, geography esetén nem. Ez utóbbi esetben a bounding box implicit módon a Földet lefedő koordináta-rendszer.

Felhasznált irodalom:

Alastair Aitchison – Pro Spatial with SQL Server 2012, Apress, 2012