1. Obsah

2. Úvod

Na úvod diplomové práce bych chtěl říci, proč vlastně vznikla. Téma této práce vzešlo po konzultaci s firmou ARCDATA PRAHA, s.r.o., kterou jsem oslovil při hledání tématu pro projekt 30, kde jsou popsány topologické operace v geografických informačních systémech, a následnou diplomovou práci. Po vzájemné dohodě jsme si vyšli vstříc a jeden z odborníků firmy, Ing. Štěpán Rybář, zabývající se uplatněním norem OpenGIS (zejména WMS, WFS, a SFS) v praxi, přišel s požadavkem na vyřešení problému převodu topologických pravidel vestavěných v programu ArcGIS 9.0 do jazyka SQL podle normy OpenGIS Simple Features Specifications for SQL, verze 1.1, z roku 1999.

Cílem diplomové práce je pokus o převod všech vestavěných topologických pravidel v ArcGIS 9.0 do Simple Features Specifications for SQL. Následně i vydání příručky v anglickém jazyce s topologickými pravidly definovanými v ArcGIS 9.0 a převedenými do Simple Features Specifications for SQL pod licencí GPL.

O diplomové práci

Diplomová práce je rozdělena do šesti kapitol, z nichž tři jsou stěžejní. Kapitola „Co je to topologie a kde se ji využívá“ uvede do problematiky geografických informačních systémů (někde uváděných pouze jako GIS) a zobrazování geografických objektů v digitálních mapách pomocí základních geografických prvků. Právě ty jsou v této kapitole podrobně popsány a vysvětleny i pomoci názorných obrázků. Jejich definice a některé obrázky jsou převzaty z normy OpenGIS Simple Features Specification for SQL. V samém závěru kapitoly je vysvětlen pojem topologie, její definice a využití v souvislosti s geografickými informačními systémy.

Poté následuje kapitola „Návrh převodu topologických pravidel Simple Features Specification for SQL“. Ta nás v úvodu stručně seznámí se samotným jazykem SQL, s výhodami ukládání dat do relačních databází a s několika praktickými příklady použití.

V následující kapitole „Přehled topologických pravidel a jejich převod do SQL“, je popsáno všech 25 topologických pravidel z ArcGIS 9.0 i s jejich definicemi, praktickým použitím a obrázkem. Dále je uveden samotný převod, ke kterému by nám měla pomoci norma OpenGIS Simple Feature Specification for SQL. Jedná se o rozšíření programovacího jazyka SQL o datové typy a funkce používané v geografických informačních systémech. Kapitola je rozdělena do třech podkapitol, kde jsou topologická pravidla rozdělena pro body, lomené čáry a mnohoúhelníky.

Ve zbývajících třech kapitolách je uveden obsah, úvod a použité zdroje.

3. Co je to topologie a kde se ji využívá

Jako první krok k tomu, abychom pochopili podstatu věci, co je to topologie, si musíme vysvětlit několik důležitých pojmů a hlavně si uvědomit, že základem jakékoliv práce s geografickými informačními systémy je používání mapové databáze. Proto je třeba mít nejprve jasno v tom, z čeho se taková databáze skládá, ale také jaké používá typy mapové informace a typy geografických prvků.

Z tohoto důvodu si následující vysvětlení rozdělíme do několika menších podkapitol.

- dva základní typy mapové informace: prostorová a popisná
- hlavní typy geografických prvků: body, lomené čáry a mnohoúhelníky
- hierarchické rozdělení geografických prvků
- definice jednotlivých geografických prvků
- topologie, její využití a význam

3.1. Dva základní typy mapové informace

Obecně rozlišujeme dva základní typy mapové informace. Jedná se o ***prostorově lokalizační informace*** a o ***popisné informace o geografických prvcích.*** Prostorově lokalizační informace popisují polohu a tvar jednotlivých geografických prvků a jejich prostorové vztahy k dalším geografickým prvkům. Jejich vyjádření je možné rozdělit do třech skupin. Jsou to body, lomené čáry a mnohoúhelníky. Informace vyjádřené mapou jsou prezentovány graficky jako soubor dílčích grafických složek. Prostorová informace je vyjádřena bodově pro prvky typu prameny, televizní vysílače, či sloupy elektrického vedení, liniově pro prvky typu silnice, vodní toky či inženýrské sítě, a plošně pro prvky, typu jezera, kraje nebo pole, či lesy. Na tom je pak založen celý geografický informační systém, který spočívá ve spojení již zmiňovaných prostorově lokalizačních informací a popisných informací o geografických prvcích a v zabezpečení prostorových vztahů mezi mapovými prvky.

3.2. Hlavní typy geografických prvků

Prvek typu ***bod*** je vyjádřen diskrétní polohou definující mapovaný prvek, jehož hranice nebo tvar jsou příliš malé, aby jej bylo možno vyjádřit jako linii nebo plochu. Je to například skalní masív, soutěska, či propast. Nebo se může jednat o prvek, který nemá plochu, např. horský vrchol, pozice GPS, či pramen vodního toku. Poloha bodu je zpravidla vyznačována speciální značkou a popisem.

Prvek typu ***lomená čára*** je uspořádaný soubor souřadnic, které po svém zřetězení vyjadřují tvar lomené čáry takového prvku v mapě, který je příliš úzký, aby mohl být vyjádřen jako plocha. Jedná se například o silnice, železnice, či vodní tok. Nebo se může jednat o prvek, který nemá šířku, například vrstevnice, isotermy, isokvanty.

Prvek typu ***mnohoúhelník*** je uzavřený obrazec, jehož hranice vymezují nějakou homogenní oblast, jako např. stát, okres nebo vodní plocha.

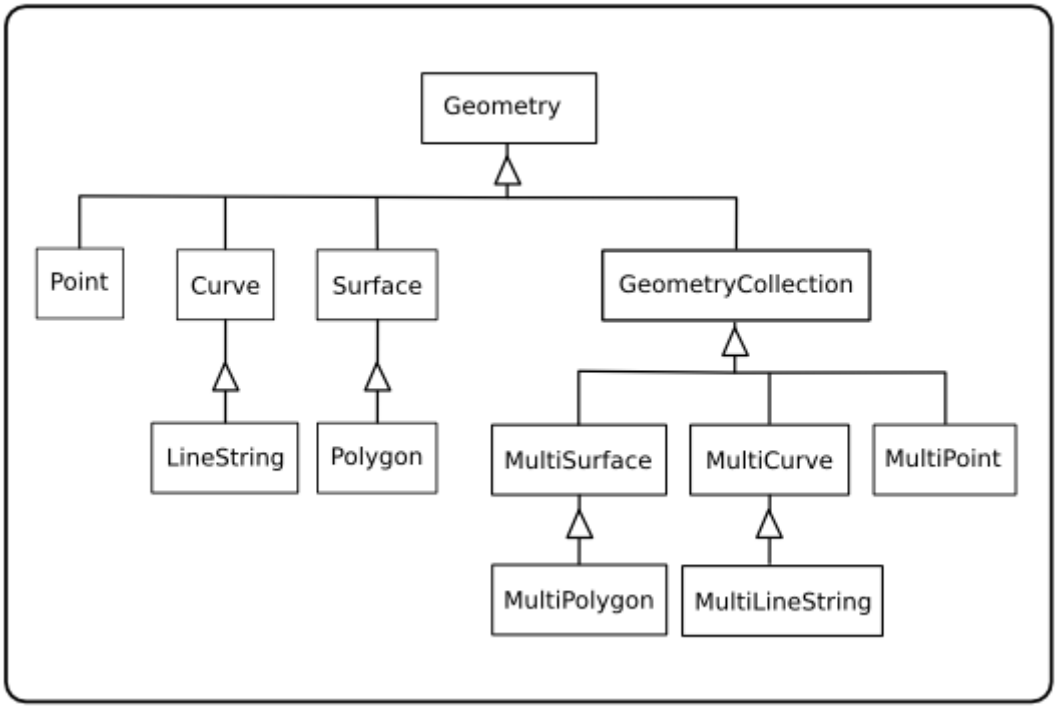
Prostorové vztahy mezi mapovými prvky jsou v mapě vyjádřeny svoji vzájemnou polohou, ale záleží na čtenáři mapy, jak je pochopí. Může nás například zajímat, zda město leží blízko jezera, můžeme hledat silniční vzdálenost mezi dvěma městy, nebo nejkratší možnou trasu mezi nimi. Dále třeba potřebujeme nalézt nejbližší nemocnici a výškovou kótu hladiny jezera na základě výšky vrstevnice, která jej lemuje, atd. Tento druh informací není v mapě obsažen přímo, ale musíme jej odvodit z grafického zákresu v mapě.

Ve smyslu grafického znázornění mapa představuje polohu jednotlivých prvků a jejich charakteristiky tak názorně, že jejich interpretace může být snadnou záležitostí. Charakteristiky mapových prvků (tj. jejich atributy) jsou vyjádřeny grafickými symboly - mapovými značkami. Např. silnice jsou kresleny pomocí čar různé šířky, typu, barvy a popisu tak, aby byly rozlišeny různé typy komunikací, vodní toky jsou kresleny a popisovány svými názvy modrou barvou, poloha škol je znázorňována smluvenou bodovou značkou, jezera mají modrou výplň, lesní plochy jsou zelené, atd.

3.3. Hierarchické rozdělení geografických prvků

Na následujícím obrázku jsou definovány vztahy mezi jednotlivými geometrickými prvky používanými v geografických informačních systémech. Nyní už také všechny geografické prvky budou nazývány geometrickými třídami (zkráceně třídami), nebo geometrickými objekty (objekty) a také bude používán pojem abstraktní a neabstraktní třída.

Abstraktní třída je taková třída, která pouze definuje společné vlastnosti a funkce jí náležících podtříd. Neabstraktní je taková třída, které mohu přidělit souřadnice a mohu z ní udělat objekt.



Obr.1: Základní hierarchické rozdělení.

3.4. Definice jednotlivých geografických tříd a jejich funkcí

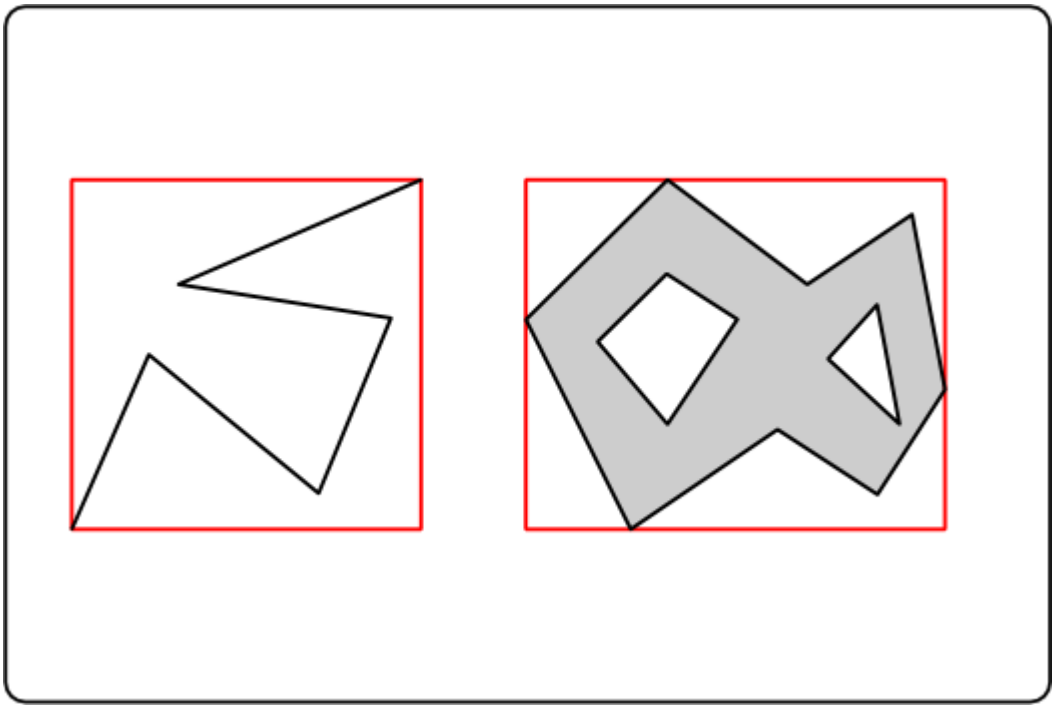
Vzhledem k odborné terminologii v anglickém jazyce nebudeme názvy jednotlivých tříd a jejich funkcí překládat. Je to dáno již zavedeným používáním anglických výrazů v tomto odvětví v celém světě a vyvarujeme se tím možných nedorozumění, zejména špatných pojmenování jednotlivých tříd a jejich funkcí při překladu do českého jazyka. Pro upřesnění a případná nedorozumění je zde uveden slovníček nejpoužívanějších terminologických výrazů.

Pro účely diplomové práce jsou stěžejní definice jednotlivých tříd a jejich funkcí, které jsou převzaty z normy Simple Feature Specification for SQL. Ty jsou podrobně popsány v této kapitole a u neabstraktních tříd jsou vyobrazeny některé jejich příklady. Nyní již ovšem přejdeme k již zmiňovanému slovníčku pojmů.

empty	prázdný
non-empty	neprázdný
simple	jednoduchý
non-simple	nejednoduchý
closed	uzavřený
non-closed	neuzavřený
closed simple	uzavřený jednoduchý
closed non-simple	uzavřený nejednoduchý
Point	bod
MultiPoint	shluk bodů
Curve	křivka
MultiCurve	shluk křivek
Line	úsečka
LineString	lomená čára
LinearRing	uzavřená jednoduchá lomená čára
MultiLineString	shluk lomených čar
Polygon	mnohoúhelník
MultiPolygon	shluk mnohoúhelníků
envelope	min. ohraničující pravoúhelník
boundary	obrys, konkávní hranice

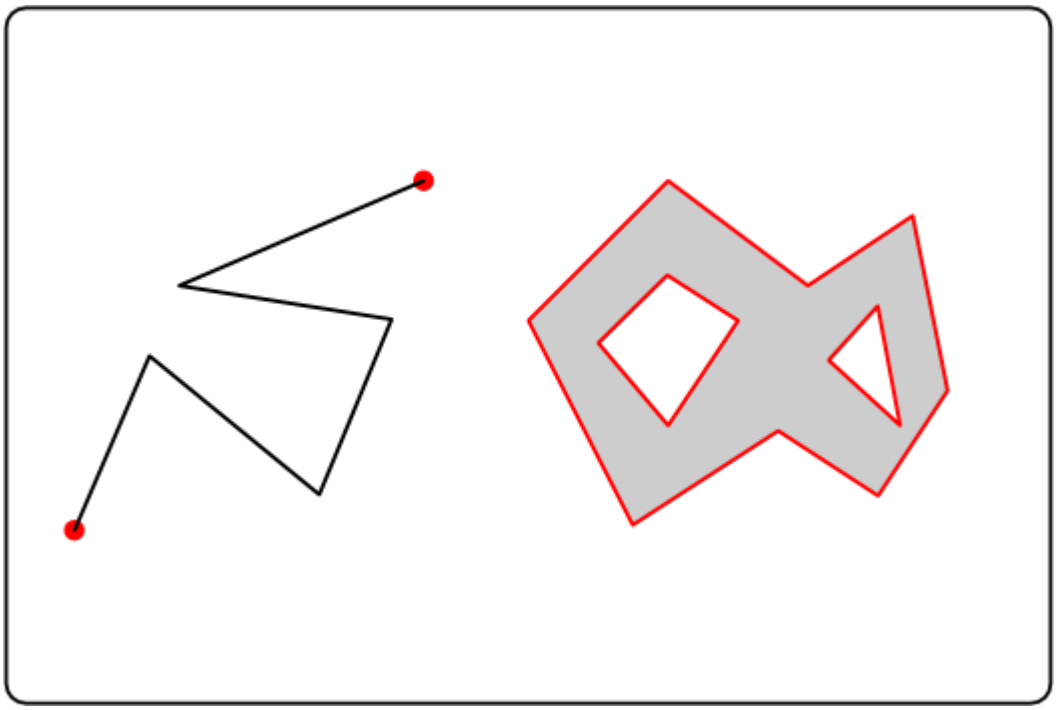
Pro úplnost jsou na následující stránce vysvětleny ještě dva důležité pojmy. Jedná se o envelope a boundary**.**

Envelope je minimální pravoúhelník ohraničující třídu Geometry. Pro lepší pochopení si envelope vysvětlíme na následujícím obrázku.



Obr.2: Grafické vysvětlení pojmu envelope.

Boundary je obrys třídy Geometry. Pro lepší pochopení si boundary opět vysvětlíme na názorném obrázku.



Obr.3: Grafické vysvětlení pojmu boundary.

K jednotlivým třídám, které jsou vyjmenovány v podkapitolách 2.4.1. až 2.4.12, jsou v normě Simple Feature Specification for SQL příslušné specifické funkce. Každá třída má několik svých funkcí, pomocí kterých zjistíme podrobné informace zrovna k té třídě, ke které je příslušná funkce přidružená. Pomocí těchto funkcí například zjistíme, jestli je daná třída simple, jestli je empty, koncový, či počáteční bod objektu typu LineString, atd. U jednotlivých geografických tříd také platí jedna důležitá vlastnost. Podřízená třída dědí funkce třídy nadřazené. Znamená to například, že funkce přiřazené kořenové třídě Geometry dědí všechny její podřazené třídy a k tomu jsou navíc rozšířeny ještě o funkce jim přidělené. Pro názornost si ještě uvedeme jednoduchý způsob, kterým budeme jednotlivé funkce zapisovat.

první řádek	deklarace funkce ve stylu jazyka Java
druhý řádek	příklad použití funkce v jazyce SQL
třetí řádek	stručné vysvětlení funkce

Tab.1: Příklad zápisu funkce.

3.4.1. Geometry

Definice:

Třída Geometry je kořenová geometrická třída celé hierarchie tříd. Jedná se o abstraktní třídu. Všechny podtřídy této třídy jsou rozděleny do nula, jedna a dvourozměrných geometrických tříd.

Funkce:

```
Integer Dimension(Geometry myGeometry)
SELECT Dimension(geometry) FROM sample_geometries
```

vrací celé číslo, které je rozměrem objektu myGeometry

```
String GeometryType(Geometry myGeometry)
SELECT GeometryType(geometry) FROM sample_geometries
```

vrací řetězec, který je názvem třídy objektu myGeometry

```
String AsText(Geometry myGeometry)
SELECT AsText(geometry) FROM sample_geometries
```

vrací řetězec, který obsahuje souřadnice objektu myGeometry ve formátu well-known-text

`Binary AsBinary(Geometry myGeometry)`

`SELECT AsBinary(geometry) FROM sample_geometries`

vrací řetězec, který obsahuje souřadnice objektu myGeometry ve formátu well-known-binary

`Integer SRID(Geometry myGeometry)`

`SELECT SRID(geometry) FROM sample_geometries`

vrací celé číslo, které je identifikátorem souřadnicového systému objektu myGeometry

`Integer IsEmpty(Geometry myGeometry)`

`SELECT IsEmpty(geometry) FROM sample_geometries`

testuje, jestli je objekt myGeometry empty

`Integer IsSimple(Geometry myGeometry)`

`SELECT IsSimple(geometry) FROM sample_geometries`

testuje, jestli je objekt myGeometry simple

`Geometry Boundary(Geometry myGeometry)`

`SELECT Boundary(geometry) FROM sample_geometries`

vrací objekt typu Geometry, který je hranicí myGeometry

`Geometry Envelope(Geometry myGeometry)`

`SELECT Envelope(geometry) FROM sample_geometries`

vrací objekt typu Geometry, který je envelope objektu myGeometry

3.4.2. GeometryCollection

Definice:

Třída GeometryCollection je souborem jednoho, či více geometrických objektů. Všechny objekty musí bý vyjádřeny v souřadnicích stejného souřadnicového systému (mají stejný SRID)

Funkce:

`Integer NumGeometries(GeomCollection myGeomCollection)`

`SELECT NumGeometries(geometry) FROM sample_geometries`

vrací celé číslo, které určuje počet geometrií v objektu myGeomCollection

```
Geometry GeometryN(GeomCollection myGeomCollection, Integer n)
SELECT GeometryN(geometry, 2) FROM sample_geometries
```

vrací objekt typu Geometry, který je n-tou geometrií myGeomCollection

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope`

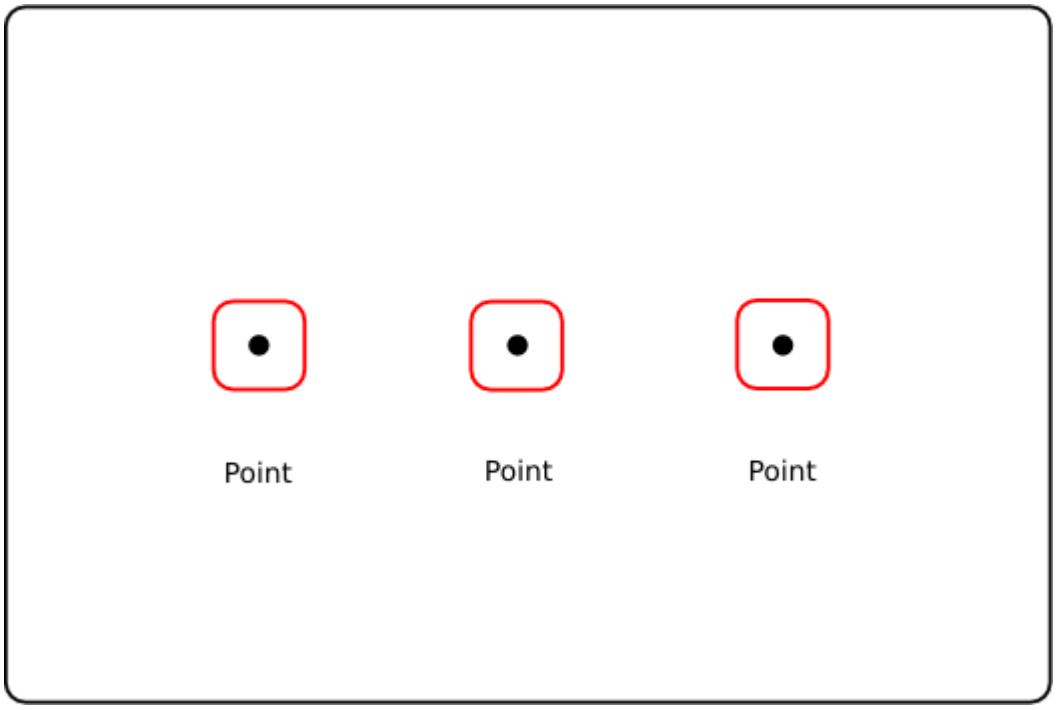
3.4.3. Point

Definice:

Třída Point je nula rozměrná geometrická třída vyjádřena x-ovou a y-ovou souřadnicí. Hranicí Pointu je prázdný soubor.

Příklad:

Názorným příkladem třídy Point může být v praxi horský vrchol, pozice GPS, či pramen vodního toku, což jsou tzv. Pointy „bez plochy“, ale také například skalní masív, soutěska, či propast, což jsou tzv. Pointy „s plochou“.



Obr.4: Příklady geometrické třídy typu Point.

Funkce:

```
Double X(Point myPoint)
SELECT X(geometry) FROM sample_geometries
```

vrací číslo, které je X souřadnicí objektu myPoint

```
Double Y(Point myPoint)
SELECT Y(geometry) FROM sample_geometries
```

vrací číslo, které je Y souřadnici objektu myPoint

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope`

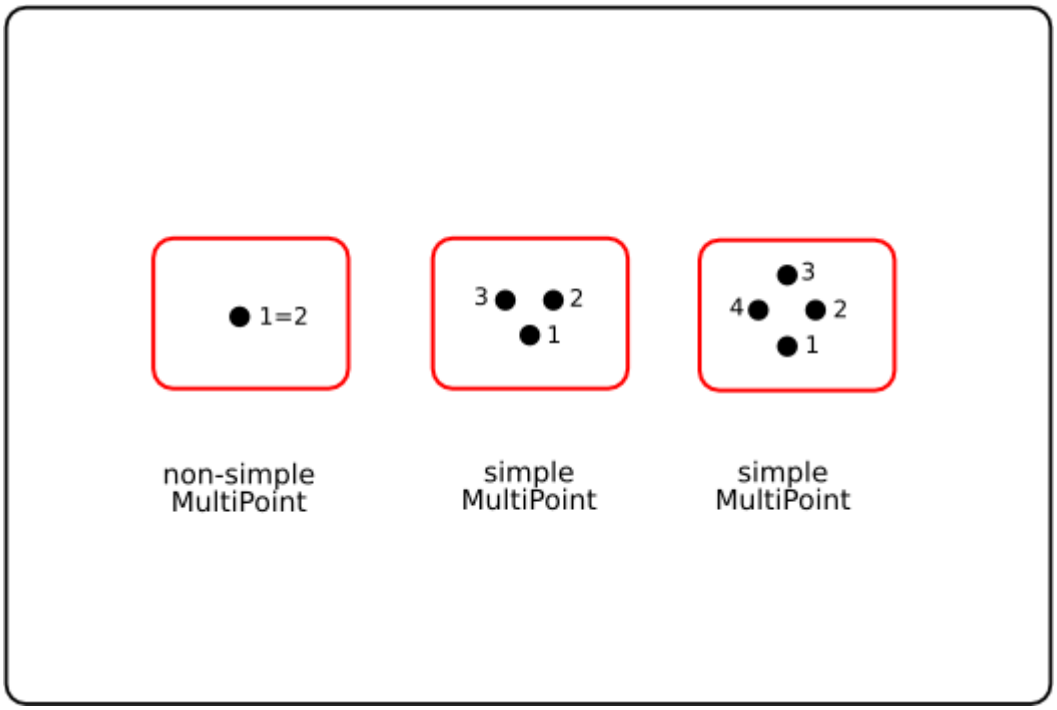
3.4.4. MultiPoint

Definice:

Třída MultiPoint je souborem jednoho, či více objektů typu Point. Části MultiPointu jsou omezeny Pointy. Pointy nejsou spojeny, či uspořádány. MultiPoint je simple, pokud žádné dva Pointy tvořící MultiPoint nejsou shodné (nemají stejné souřadnice). Hranicí MultiPointu je prázdný soubor.

Příklad:

Názorným příkladem třídy MultiPoint můžou být v praxi skutečné body, jako soubor bodů při trasování GPS, či soubor těžebních věží jednoho dolu, ale také příklady bývalých ploch. Je to například místo výskytu stejného druhu rostlin, či stromů, nebo nebezpečná podmořská skaliska.



Obr.5: Příklady geometrické třídy typu MultiPoint.

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope, NumGeometries, GeometryN`

3.4.5.Curve

Definice:

Třída Curve je jednorozměrná abstraktní geometrická třída, většinou ukládaná jako posloupnost Pointů se specifickým typem interpolace mezi Pointy. Tato specifikace definuje pouze jednu podtřídu Curve, a to LineString, který používá lineární interpolace mezi Pointy. Curve je simple, pokud tentýž bod protíná pouze jednou. Curve je closed, pokud její počáteční a koncový bod je tentýž. Hranicí Curve je objekt typu MultiPoint, nebo prázdný soubor.

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope`

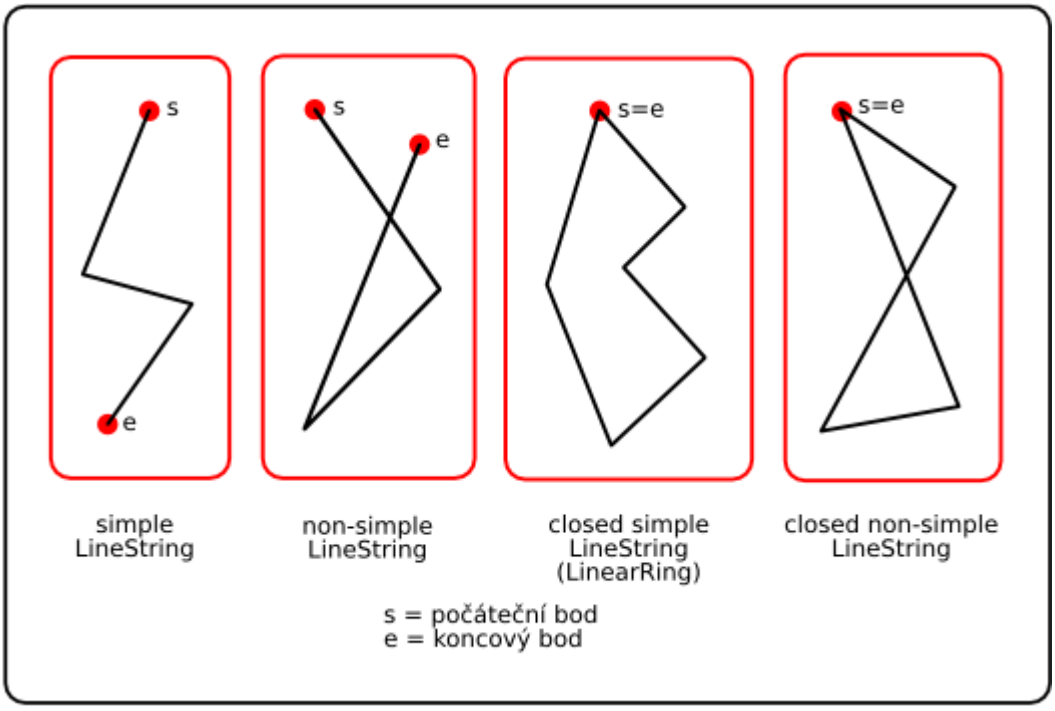
3.4.6. LineString, Line, LinearRing

Definice:

Třída LineString je třída Curve s lineární interpolací mezi body, neboli oba body jsou spojeny přímkou, což znamená, že se jedná o úsečku. Každá následující dvojice bodů definuje úsek jednotlivé Line. LineString může být simple, non-simple, nebo také closed, non-closed. LineString je simple, pokud vícekrát neprotíná tentýž bod, takže se nekříží. Line je LineString právě s dvěma Pointy. LinearRing je closed a zároveň simple LineString. Hranicí LineStringu je objekt typu Point, nebo prázdný soubor.

Příklad:

Názorným příkladem třídy LineString můžou být v praxi skutečné LineStringy, jako vrstevnice, isotermy, isokvanty, ale také příklady bývalých ploch, což jsou například silnice, železnice, či vodní toky.



Obr.6: Z leva doprava se jedná o simple LineString, non-simple LineString, closed simple LineString (neboli LinearRing) a closed non-simple LineString.

Funkce:

```
Point StartPoint(LineString myLineString)
```

```
SELECT StartPoint(geometry) FROM sample_geometries
```

vrací objekt typu Point, který je počátečním bodem objektu myLineString

`Point EndPoint(LineString myLineString)`

`SELECT EndPoint(geometry) FROM sample_geometries`

vrací objekt typu Point, který je koncovým bodem objektu myLineString

`Integer IsClosed(LineString myLineString)`

`SELECT IsClosed(geometry) FROM sample_geometries`

testuje, jestli objekt myLineString je closed

`Integer IsRing(LineString myLineString)`

`SELECT IsRing(geometry) FROM sample_geometries`

testuje, jestli objekt myLineString je LinearRing, tj. simple a closed LinearString

`Double Length(LineString myLineString)`

`SELECT Length(geometry) FROM sample_geometries`

vrací číslo, které je délkou objektu myLineString

`Integer NumPoints(LineString myLineString)`

`SELECT NumPoints(geometry) FROM sample_geometries`

vrací celé číslo, které je počtem Pointů v objektu myLineString

`Point PointN(LineString myLineString, Integer n)`

`SELECT PointN(geometry, 2) FROM sample_geometries`

vrací objekt typu Point, který je n-tým Pointem v objektu myLineString

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope`

3.4.7. MultiCurve

Definice:

Třída MultiCurve je souborem jednoho, či více objektů typu Curve. Části třídy MultiCurve jsou jednotlivé Curve. MultiCurve je simple, pokud všechny její elementy jsou také simple.

Průniky jakýchkoliv dvou Curve se objevují pouze na hranicích těchto dvou elementů. MultiCurve je closed, pokud všechny její elementy jsou taktéž closed. MultiCurve je definována jako topologicky uzavřená. Hranice MultiCurve je objekt typu MultiPoint, nebo prázdný soubor.

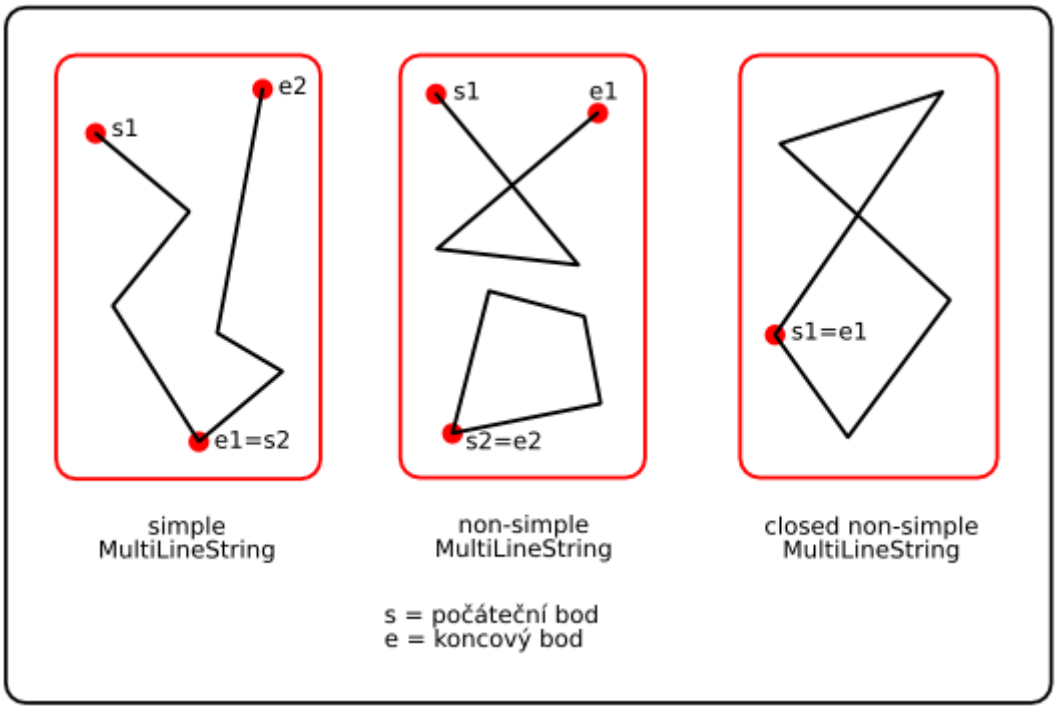
3.4.8. MultiLineString

Definice:

Třída MultiLineString je souborem jedné, či více tříd typu LineString. Pro větší představu jednotlivých druhů třídy MultiLineString nám poslouží následující obrázek.

Příklad:

Názorným příkladem třídy MultiLineString můžou být v praxi skutečné MultiLineStringy, jako úseky kabelového vedení mezi jednotlivými sloupy, či části parovodu mezi bezpečnostními uzávěry, nebo odbočkami, ale také příklady bývalých ploch, což jsou například silnice s křižovatkami, železniční trať s nádražími, či závodní okruh s jednotlivými etapami závodu.



Obr.7: Z leva doprava se pak jedná o simple MultiLineString s dvěma elementy, non-simple MultiLineString s 2 elementy a non-simple closed MultiLineString s 2 elementy.

Funkce:

```
Integer IsClosed(MultiLineString myMultiLineString)
SELECT IsClosed(geometry) FROM sample_geometries
```

testuje, jestli objekt myMultiLineString je closed

```
Double Length(MultiLineString myMultiLineString)
SELECT Length(geometry) FROM sample_geometries
```

vrací číslo, které je součtem délek jednotlivých segmentů objektu myMultiLineString

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope, NumGeometries, GeometryN`

3.4.9. Surface

Definice:

Třída Surface je dvourozměrná geometrická třída. Simple Surface lze definovat jako geometrickou třídu, skládající se z jednotlivých „políček“, kterým je přiřazena vnější a vnitřní hranice. Hranicí simple Surface je sada closed Curve, které odpovídají jejich vnějším a vnitřním hranicím. Hranicí Surface je objekt typu MultiCurve.

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope`

3.4.10. Polygon

Definice:

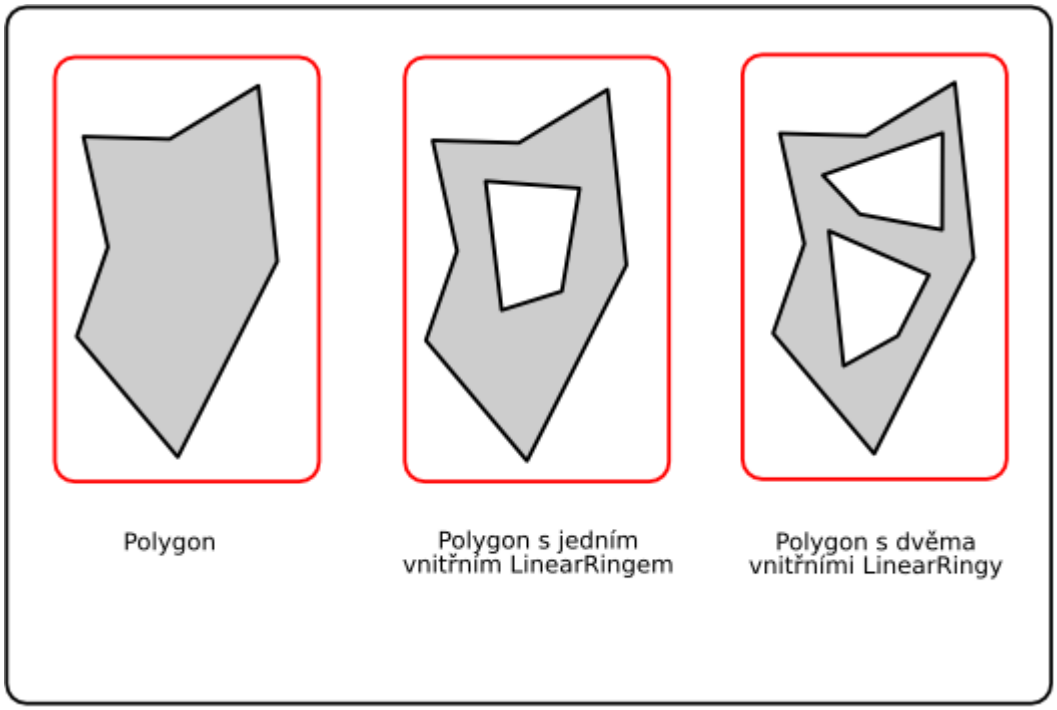
Třída Polygon je Surface definovaná jednou vnější a žádnou, jednou, nebo více vnitřními hranicemi. Každá vnitřní hranice definuje ostrovy v Polygonu. Hranicí Polygonu je objekt typu MultiLineString.

Příklad:

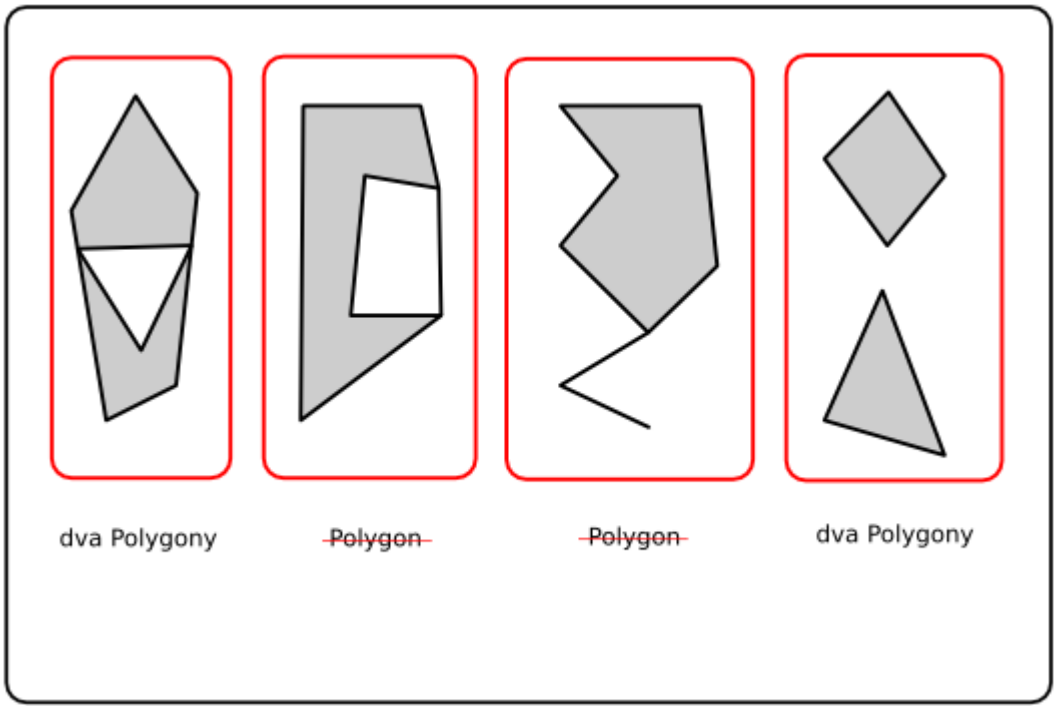
Názorným příkladem této třídy může být v praxi ohraničený pozemek, stavební parcela kde výkopové jámy představují ostrovy v polygonu.

- Polygon je topologicky uzavřen.
- Hranice Polygonu je tvořena sadou LinearRingů, které tvoří jeho vnější a vnitřní hranice.
- Žádné dvě třídy typu LinearRing se nesmějí protínat, třídy LinearRing mohou mít na hranicích třídy Polygon průsečík, ale pouze jako bod, ve kterém se dotýkají
- Vnitřek každého Polygonu je souvislá sada třídy typu Point.
- Vnějšek každého Polygonu s jednou, či více ostrovy není spojený, každý ostrov definuje spojenou část vnějšku.
- Polygon by neměl mít ostré hrany, či hroty.

Pro další představu o třídě typu Polygon nám poslouží názorné obrázky, které jsou uvedeny níže.



Obr.8: Příklady geometrické třídy typu Polygon s žádnou, jednou a dvěma vnitřními třídami typu LinearRing.



Obr.9: Příklady objektů, které by mohly být chybně označovány jako Polygony. Příklady 1 a 4 však mohou být prezentovány jako dva jednotlivé Polygony.

Funkce:

```
Point Centroid(Polygon myPolygon)
```

SELECT Centroid(geometry) FROM sample_geometries

vrací objekt typu Point, který je těžištěm objektu myPolygon

```
Point PointOnSurface(Polygon myPolygon)
```

SELECT PointOnSurface(geometry) FROM sample_geometries

vrací objekt typu Point, který zaručeně leží uvnitř objektu myPolygon

```
Double Area(Polygon myPolygon)
```

SELECT Area(geometry) FROM sample_geometries

vrací číslo, která určuje plochu objektu myPolygon v jednotkách na druhou

```
LineString ExteriorRing(Polygon myPolygon)
```

SELECT ExteriorRing(geometry) FROM sample_geometries

vrací objekt typu LineString, který je vnější hranicí (LinearRing) objektu myPolygon

`Integer NumInteriorRing(Polygon myPolygon)`

`SELECT NumInteriorRing(geometry) FROM sample_geometries`

vrací číslo, která určuje počet vnitřních hranic *(LinearRing)* objektu myPolygon

`LineString InteriorRingN(Polygon myPolygon,Integer n)`

`SELECT InteriorRingN(geometry, 2) FROM sample_geometries`

vrací objekt typu LineString, který je n-tým vnitřním LineRingem objektu myPolygon

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope`

3.4.11. MultiSurface

Definice:

MultiSurface je dvourozměrná geometrická třída, jejíž elementy jsou třídy Surface. Dvě Surface třídy MultiSurface se navzájem nesmějí překrývat. Hranice jakýchkoliv dvou elementů MultiSurface se mohou dotýkat v omezeném počtu Pointů. Podtřídou MultiSurface je MultiPolygon, který je zastoupen souborem Polygonů. Hranicí MultiSurface je objekt typu MultiCurve.

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope, NumGeometries, GeometryN`

3.4.12. MultiPolygon

Definice:

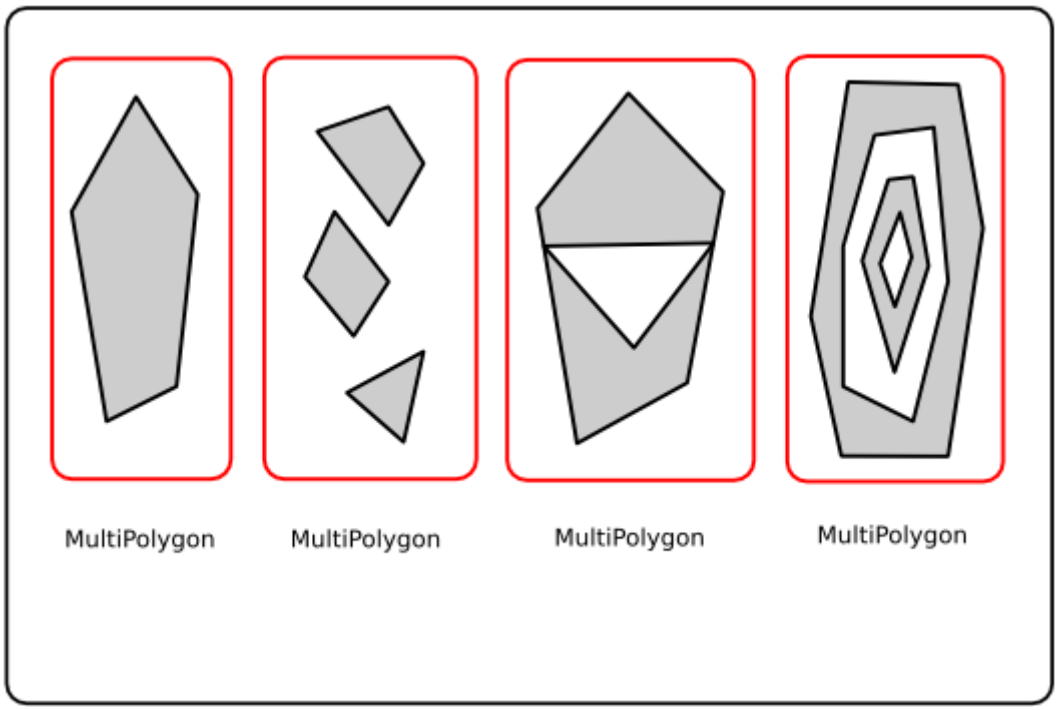
Třída MultiPolygon je MultiSurface, jejíž elementy jsou Polygony. Platí pro ně následující pravidla.

- Vnitřky dvou Polygonů, které jsou částmi MultiPolygonu se nesmí protínat.
- Hranice dvou Polygonů, které jsou částmi MultiPolygonu se nesmí křížit a mohou se dotýkat pouze v omezeném počtu bodů.
- MultiPolygon je definován jako topologicky uzavřený
- MultiPolygon by neměl mít ostré hrany, hroty, či díry
- MultiPolygon je uzavřený soubor bodů

Hranice třídy MultiPolygon je soubor tříd LineString zastoupených hranicemi jejich vlastních Polygonů. Každý LinearRing na hranici MultiPolygonu je na hranici právě jedné části Polygonu a každý LinearRing na hranicích jednotlivých části Polygonu je také na hranici MultiPolygonu.

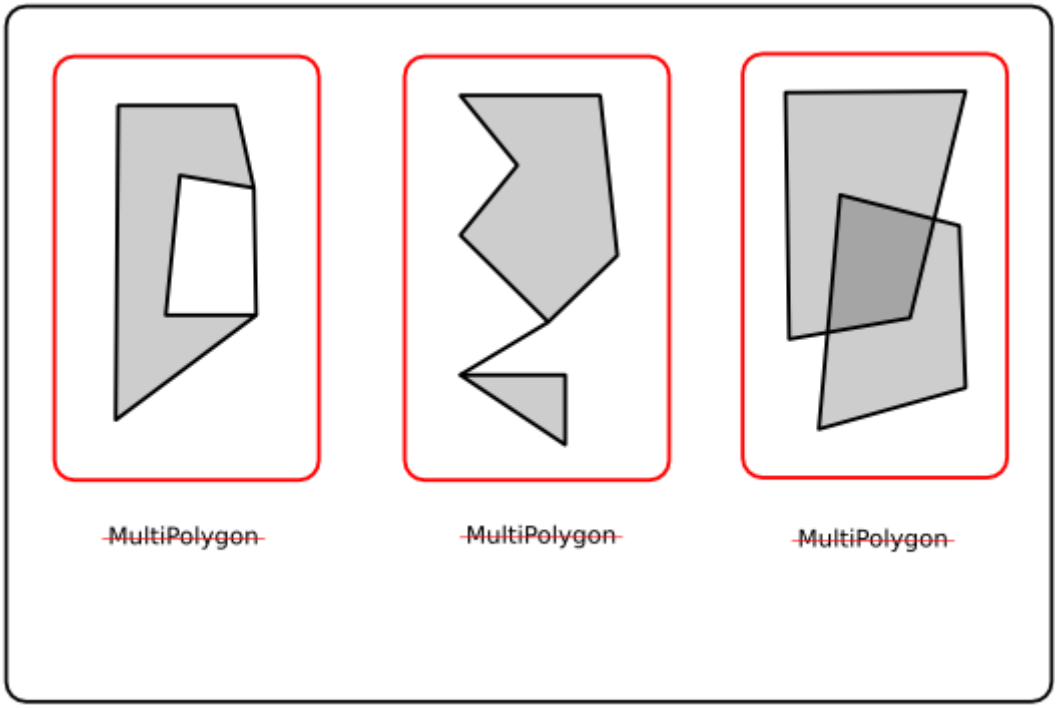
Příklad:
Názorným příkladem této třídy může být v praxi výskyt ledových polí, vrstevnicové pásy, nebo jedno souostroví, skládající se z několika ostrovů, nebo pozemek jednoho vlastníka skládající se z několika parcel.

Pro představu je na obrázku na následující stránce uvedeno několik příkladů platných MultiPolygonů s 1, 3, 2 a 2 Polygony.



Obr.10: Příklady geometrické třídy typu MultiPolygon.

Na obrázku níže si pro zajímavost ukážeme příklady několika objektů, které by mylně mohly být zaměňovány s MultiPolygony.



Obr.11: Příklady neplatné geometrické třídy typu MultiPolygon.

`Funkce:`

`Point Centroid(Polygon myMultiPolygon)`

`SELECT Centroid(geometry) FROM sample_geometries`

vrací objekt typu Point, který je těžištěm objektu myMultiPolygon

`Point PointOnSurface(Polygon myMultiPolygon)`

`SELECT PointOnSurface(geometry) FROM sample_geometries`

vrací objekt typu Point, který zaručeně leží uvnitř objektu myMultiPolygon

`Double Area(Polygon myMultiPolygon)`

`SELECT Area(geometry) FROM sample_geometries`

vrací číslo, které určuje plochu objektu myMultiPolygon v jednotkách na druhou

Zděděné funkce:

`Dimension, GeometryType, AsText, AsBinary, SRID, IsEmpty, IsSimple, Boundary, Envelope, NumGeometries, GeometryN`

3.5. WKT a WKB formát zápisu geometrických tříd v SQL

Simple Features Specification specifikace definuje dva standardní způsoby zápisů prostorových objektů: well-known-text reprezentaci (WKT) a well-known-binary reprezentaci (WKB). Oba dva, WKT i WKB, zahrnují informace o typu objektu a souřadnicích, ze kterých se objekt tvoří. OpenGIS specifikace také požaduje, aby formát prostorového objektu zahrnoval identifikátor souřadnicového systému (SRID). SRID je požadován při vkládání nově vytvořeného prostorového objektu do databáze.

Příklad:

```
INSERT INTO SPATIAL TABLE (GEOMETRIE,JMENO)
VALUES(GeometryFromText('POINT(-126.4 45.32)',312),'A place')
```

Poznámka: Funkce `GeometryFromText` potřebuje `SRID` číslo.

geometrická třída	**formát zápisu WKT**	**komentář**
Point	`'POINT (10 10)'`	Point
LineString	`'LINESTRING (10 10, 20 20, 30 40)'`	LineString s třemi Pointy
Polygon	`'POLYGON ((0 10, 10 0, 20 10, 10 20, 10 10, 0 10),(4 8, 8 4, 8 8, 4 8),(12 5, 17 10, 12 15, 12 5))'`	Polygon s jedním vnějším LinearRingem a s dvěmi vnitřními LinearRingy
MultiPoint	`'MULTIPOINT ((10 10),(20 20))'`	MultiPoint s dvěma Pointy
MultiLineString	`'MULTILINESTRING ((10 10, 20 20), (15 15, 30 15))'`	MultiLineString s dvěma LineStringy
MultiPolygon	`'MULTIPOLYGON ( ((10 10, 10 20, 20 20, 20 15, 10 10)), ((60 60, 70 70, 80 60, 60 60 ) ))'`	MultiPolygon s dvěma polygony
GeomCollection	`'GEOMETRYCOLLECTION (POINT (10 10), POINT (30 30), LINESTRING (15 15, 20 20))'`	GeometryCollection skládající se z dvou Pointů a jednoho LineStringu

Tab.2: Formát zápisu WKT jednotlivých geometrických objektů.

Dosud jsme se naučili, jak mapové prvky mohou být vyjádřeny body, lomenými čarami a mnohoúhelníky. Nicméně, jak jsme se dozvěděli na začátku této kapitoly, při čtení mapy náš mozek vyhodnocuje i informace o prostorových vztazích mezi prvky. Tak např. studujeme trasy na uliční mapě města, abychom nalezli vhodnou cestu z letiště do hotelu, nebo podobně jsme schopni určit dva sousedící pozemky a ulici, u které se nacházejí. Tyto vztahy vyhodnocujeme hledáním na sebe navazujících linií tvořících cestu, resp. Vymezením ploch mezi liniemi a identifikací sousedících oblastí. Dalším takovým příkladem může být výskyt chyb v digitálních mapách, kde se nám překrývají například vodní plocha s parcelami, které jsou k ní přilehlé. Abychom tyto chyby odstranili, musíme tedy najít průsečíky vodní plochy a přilehlých parcel. V digitálních mapách jsou takové prostorové vztahy vyjádřeny pomocí topologie. S tím souvisí i následující dvě kapitoly, kde jsou představeny funkce pro testování prostorových vztahů mezi geometrickými objekty a funkce podporující prostorovou analýzu. Tyto funkce popisují základní topologické vztahy mezi jednotlivými geografickými objekty. Pro následný převod topologických pravidel byly testovány v programu JTS Test Builder verze 1.5, což je program vycházející z normy Simple Features Specification for SQL od OpenGIS konsorcia pro testování prostorových vztahů a funkcí v dvourozměrném souřadnicovém systému.

- JTS conforms to the Simple Features Specification for SQL published by the Open GIS Consortium
- JTS provides a complete, consistent, robust implementation of fundamental 2D spatial algorithms
- JTS is fast enough for production use
- JTS is written in 100% pure Java™
- JTS is open source (under the LGPL license)

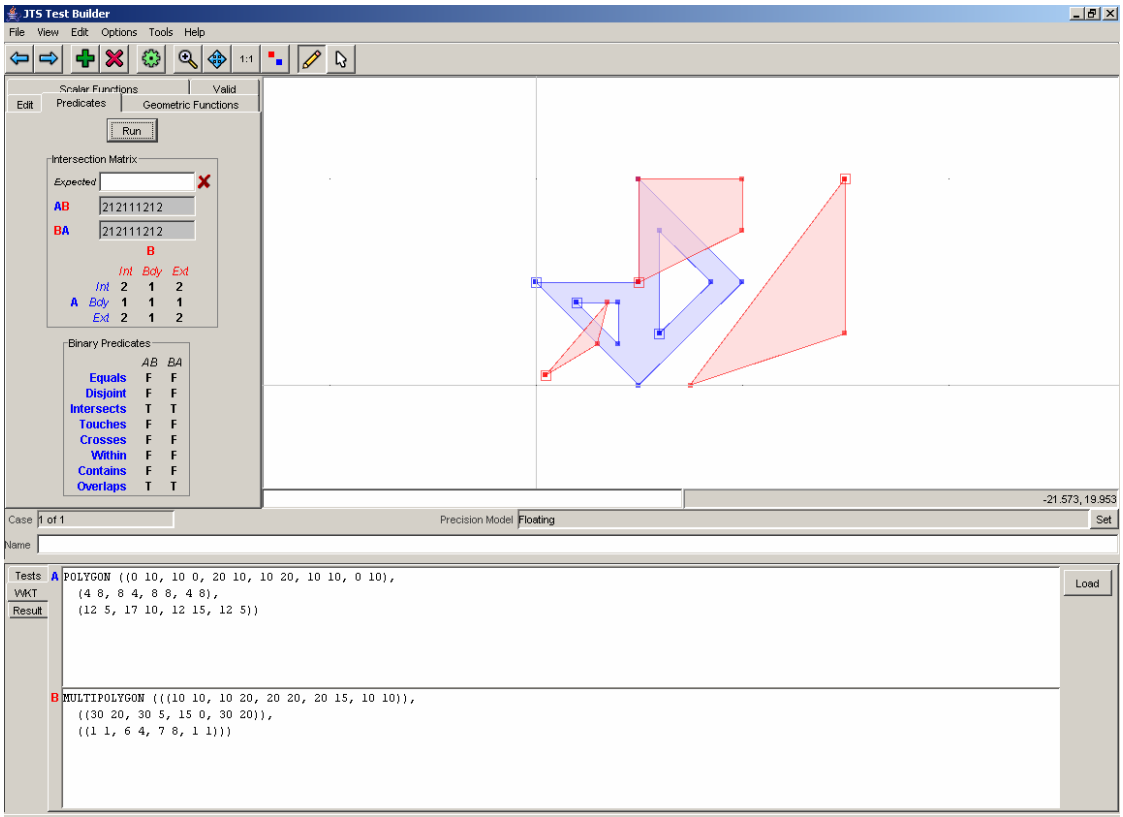
3.6. Funkce pro testování prostorových vztahů mezi geom. objekty

Všechny funkce, které jsou uvedeny v této kapitole, popisují prostorové vztahy, které mohou nastat mezi jednotlivými geometrickými objekty. Jejich definice a obrázky jsou převzaty z normy Simple Features Specification for SQL. V následující tabulce jsou definovány symboly, používané v logických zápisech jednotlivých funkcí.

I(a)	libovolný vnitřní bod objektu *a*
E(a)	libovolný vnější bod objektu *a*
B(a)	libovolný hraniční bod objektu *a*
dim(x)	rozměr objektu v *x*
⇔	právě když
∩	průnik
=	rovná se
≠	nerovná se
!	negace
Ø	prázdná množina
∧	a zároveň

Tab.3: Logické symboly

Všechny funkce, jmenované v této a následující kapitole, tedy kapitole 2.6. a 2.7. jsou testovány v programu JTS Test Builder verze 1.0, z něhož je i následující ukázka vztahu Intersection.



Obr.12: Ukázka vztahu Intersection z JTS Test Builder verze 1.0

3.6.1. Disjoint (nenáleží)

Vztah Disjoint mezi dvěma objekty *a* a *b* je definován takto:

$$a.Disjoint(b) \Leftrightarrow a \cap b = \varnothing$$

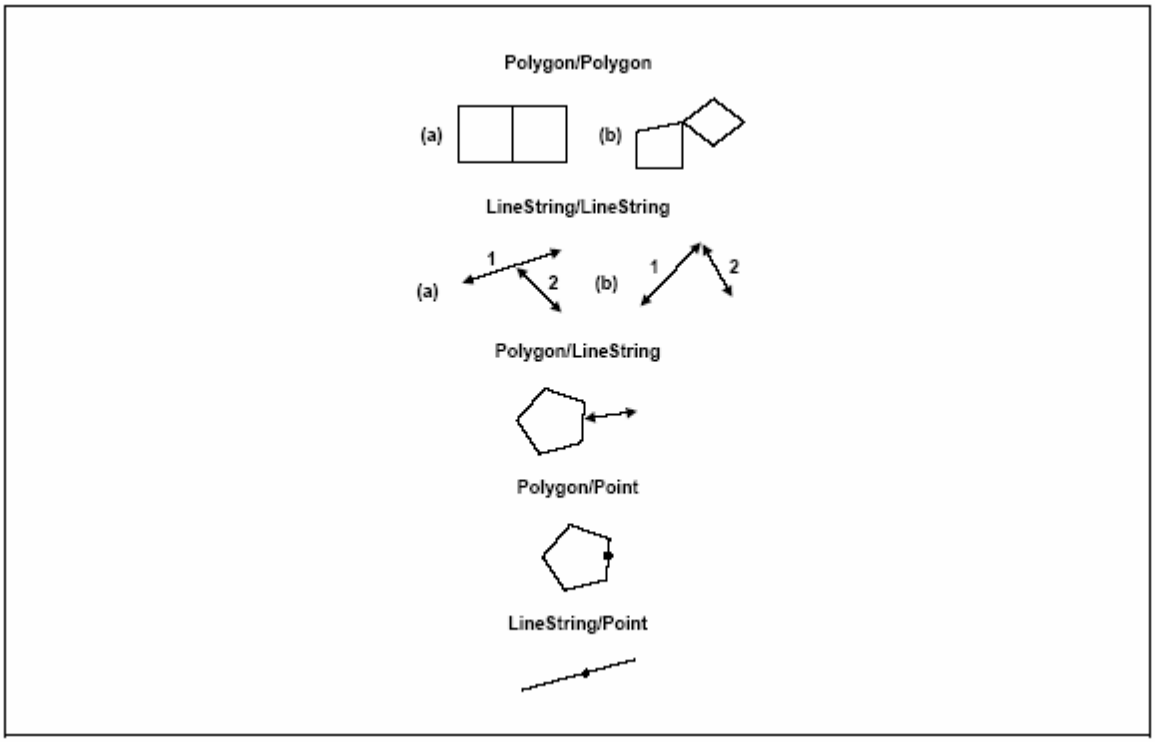
Logický zápis znamená, že průnikem dvou objektů *a* a *b* je prázdná množina.

3.6.2. Touches (dotýká)

Vztah Touches mezi dvěma objekty *a* a *b* je definován takto:

$$a.Touches(b) \Leftrightarrow (I(a) \cap I(b) = \varnothing) \wedge (a \cap b) \neq \varnothing$$

Logický zápis znamená, že neexistuje ani jeden společný vnitřní bod a zároveň existuje alespoň jeden společný bod, tzn. že existuje jeden, nebo více hraničních bodů.



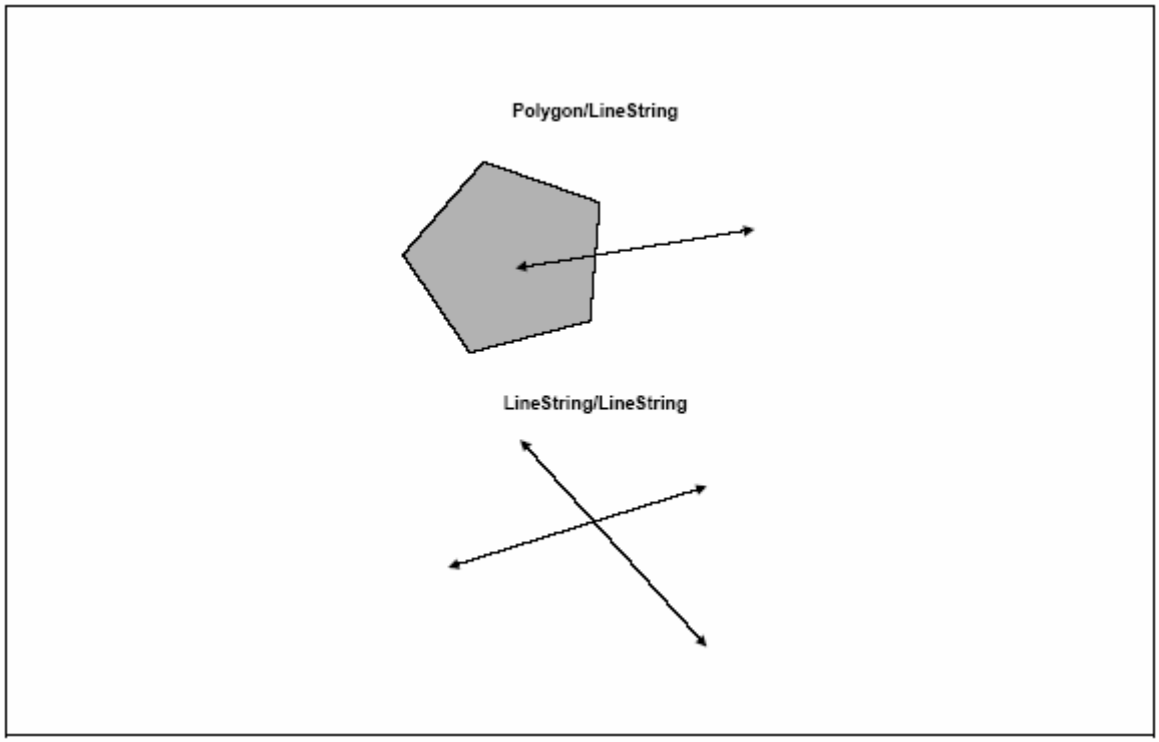
Obr.13: Několik příkladů vztahu Touches.

3.6.3. Crosses (kříží)

Vztah Crosses mezi dvěma objekty *a* a *b* je definován takto:

$$a.Crosses(b) \Leftrightarrow (dim(I(a) \cap I(b)) < max(dim(I(a)), dim(I(b)))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

Logický zápis znamená, že rozměr průniků vnitřních části objektů *a* a *b* je menší než maximum obou těchto rozměrů a zároveň průnikem *a* a *b* není *a* a zároveň průnikem *a* a *b* není *b*.



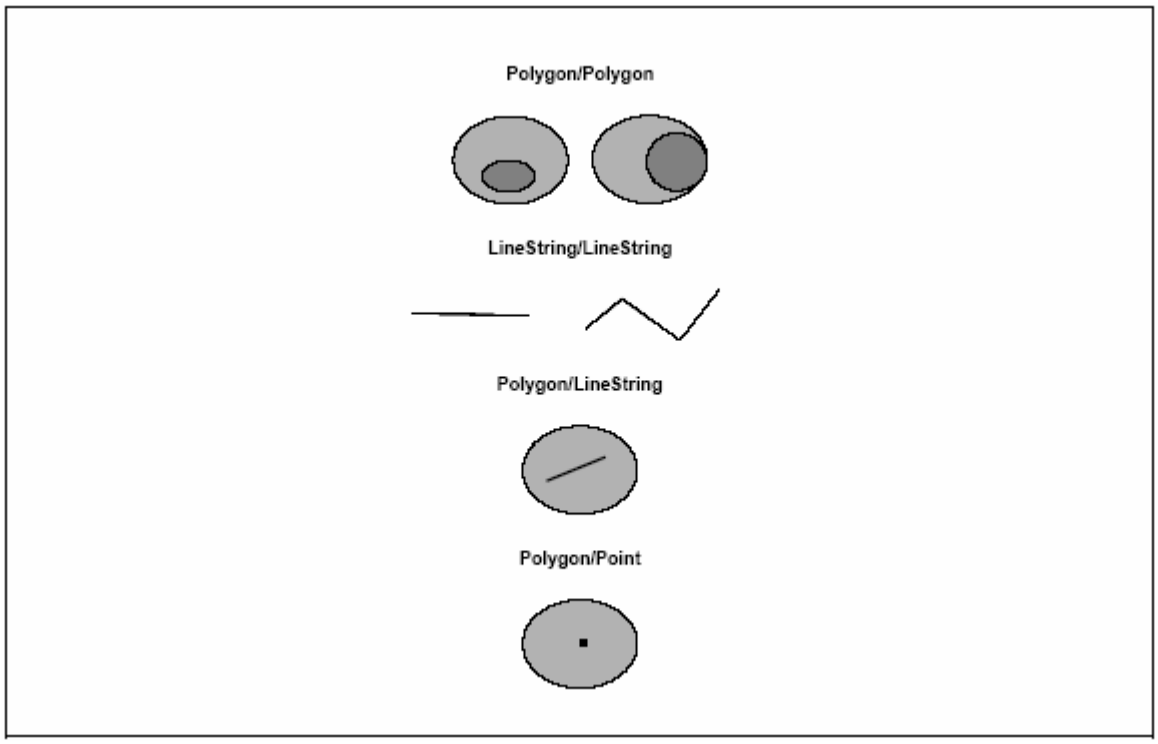
Obr.14: Několik příkladů vztahu Crosses.

3.6.4. Within (leží uvnitř, je obsažen v)

Vztah Within mezi dvěma objekty *a* a *b* je definován takto:

$$a.Within(b) \Leftrightarrow (a \cap b = a) \wedge (I(a) \cap I(b) \neq \varnothing)$$

Logický zápis znamená, že *a* náleží celé *b* a zároveň průnik vnitřních částí *a* a *b* není prázdný.



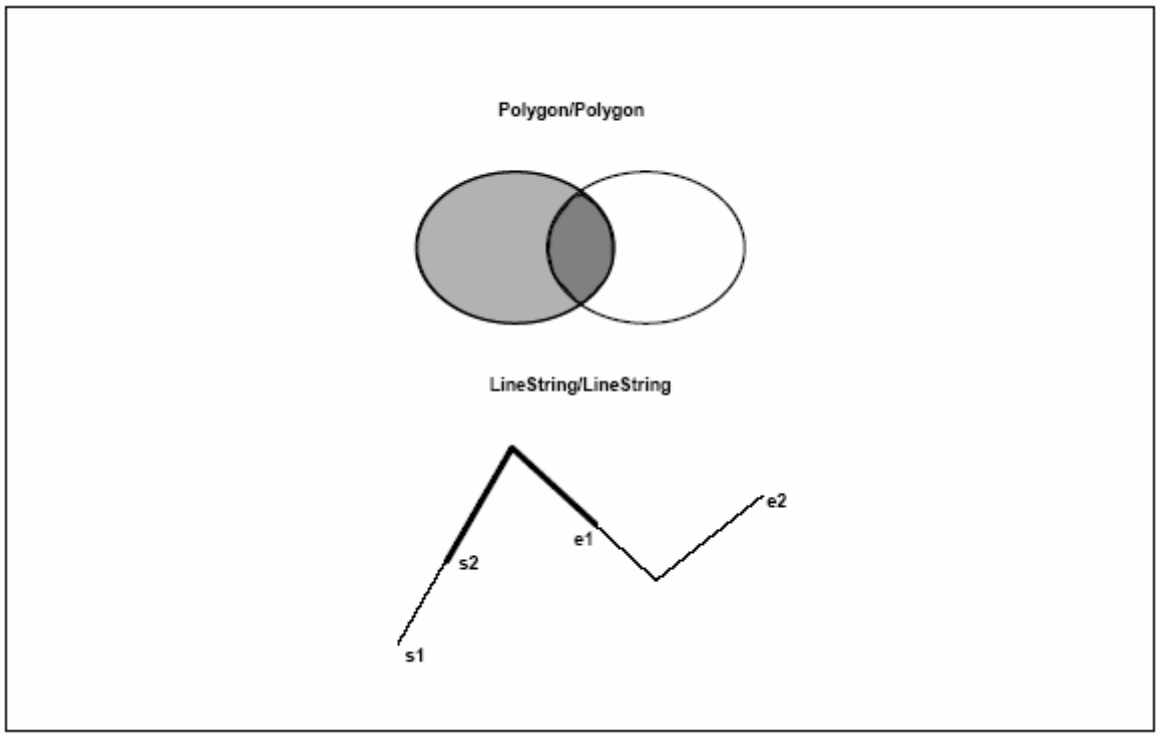
Obr.15: Několik příkladů vztahu Within.

3.6.5. Overlaps (překrývá, přesahuje)

Vztah Overlaps mezi dvěma objekty *a* a *b* je definován takto:

$$a.Overlaps(b) \Leftrightarrow (dim(I(a)) = dim(I(b)) = dim(I(a) \cap I(b))) \wedge (a \cap b \neq a) \wedge (a \cap b \neq b)$$

Logický zápis znamená, že *a*, *b*, i jejich průnik musí být stejný objekt a zároveň *a* neleží celé v *b* a naopak, rozměr *a* je stejný jako rozměr *b* a stejný jako rozměr průniku uvnitř částí *a* a *b*.



Obr.16: Několik příkladů vztahu Overlaps.

3.6.6. Contains (obsahuje)

Vztah Contains mezi objekty *a* a *b* je definován takto.

$$a.Contains(b) \Leftrightarrow b.Within(a)$$

Logický zápis znamená, že *a* obsahuje *b*, právě když *b* leží uvnitř *a*.

3.6.7. Intersects (protíná)

Vztah Intersects mezi objekty *a* a *b* je definován takto.

$$a.Intersects(b) \Leftrightarrow !\, a.Disjoint(b)$$

Logický zápis znamená, že *a* protíná *b*, právě když není pravda, že *a* nenáleží *b*..

3.6.8. Equals (rovná se)

Vztah Equals mezi objekty *a* a *b* je definován takto.

Každý bod obsažený v objektu *a* je roven každému bodu obsaženému v objektu *b*.

3.7. Funkce podporující prostorovou analýzu

Těchto sedm funkcí je využito pro vytváření nových geometrických objektů na základě topologických vztahů mezi již existujícími geometrickými objekty. Jejich aplikací vznikne nová geometrie.

3.7.1. Distance

```
Double Distance(Geometry myGeometryA, Geometry myGeometryB)
```

Vrací číslo, reprezentující nejkratší vzdálenost mezi dvěma body dvou geometrií

3.7.2. Buffer

```
Geometry Buffer(Geometry myGeometry, Double myNumber)
```

Vrací objekt typu Geometry, reprezentující všechny body, jejíž vzdálenost od zadané geometrie je menší, nebo rovna požadované vzdálenosti. Vrací tzv. obalovou plochu.

3.7.3. ConvexHull

```
Geometry ConvexHull(Geometry myGeometry)
```

Vrací objekt typu Geometry, reprezentující nejmenší konvexní obal zadané geometrie.

3.7.4. Intersection

```
Geometry Intersection(Geometry myGeometryA, Geometry
myGeometryB)
```

Vrací objekt typu Geometry, reprezentující průnik jedné geometrie s druhou geometrií.

3.7.5. Union

```
Geometry Union(Geometry myGeometryA, Geometry myGeometryB)
```

Vrací objekt typu Geometry, reprezentující sjednocení jedné geometrie s druhou geometrií.

3.7.6. Difference

```
Geometry Difference(Geometry myGeometryA, Geometry
myGeometryB)
```

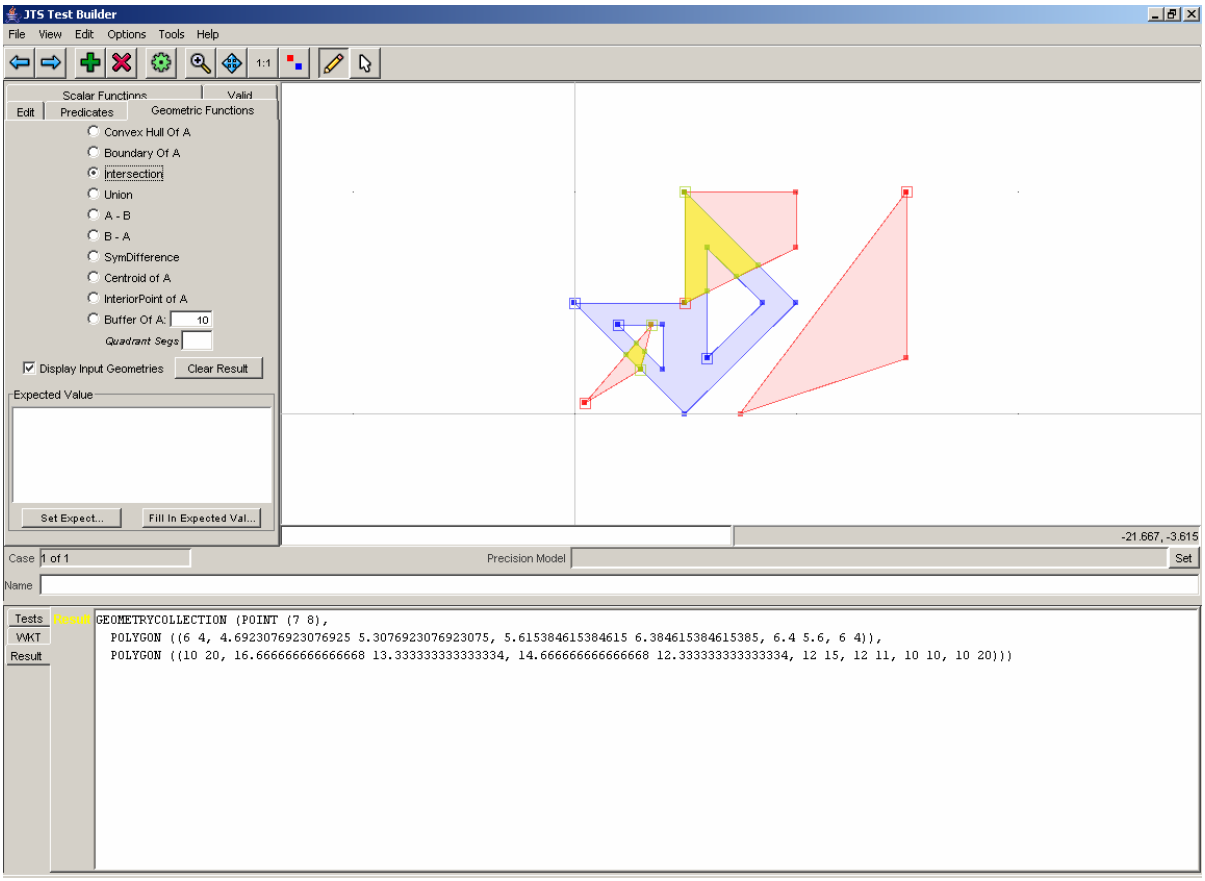
Vrací objekt typu Geometry, reprezentující rozdíl jedné a druhé geometrie.

3.7.7. SymDifference

`Geometry SymDifference(Geometry myGeometryA, Geometry myGeometryB)`

Vrací objekt typu Geometry, reprezentující symetrický rozdíl jedné a druhé geometrie.

Všechny výše jmenované funkce byly opět testovány v programu JTS Test Builder, verze 1.5. Na následujícím obrázku je příklad funkce Intersection.



Obr.17: Ukázka vztahu Intersection z JTS Test Builder verze 1.5

3.8. Topologie, kde se jí využívá a proč je důležitá

Nyní si již konečně můžeme vysvětlit, co tento stěžejní termín znamená. Jedna z mnoha definic říká, že topologie je matematický postup založený na explicitním definování prostorových vztahů. To prakticky znamená, že v mapách definuje topologie prostorové vztahy mezi geografickými prvky a je nezbytná pro zajištění kvalitních dat. Identifikuje přilehlé polygony a může definovat jeden prvek jako soubor jiných prvků (např. plochu jako soubor lomených čar, které ji ohraničují). Umožňuje pak také pokročilé prostorové analýzy a hraje klíčovou roli pro zajištění kvalitní databáze geografického informačního systému (GIS).

S topologií se asi nejvíce setkáme právě v geografických informačních systémech. Umožňuje jim, aby odpověděly na otázky týkající se přilehlosti, sousedství a totožnosti. Například v ArcGIS nabízí topologie uživatelům přínosný a flexibilní nástroj pro specifikaci pravidel sloužících k vytvoření kvalitních a celistvých prostorových dat. Můžeme tak například zjistit, že všechny parcely tvoří uzavřený prstenec, nepřekrývají se a nejsou mezi nimi mezery. Topologii můžeme také použít k ověření prostorových vztahů mezi třídami prvků. Jedním z názorných použití například je, když hranice bloků parcel v datovém modelu parcel musí být totožné s hranicemi parcel. Topologické vztahy mohou být považovány za omezující podmínky pro prostorová data. ArcGIS aplikuje tyto podmínky a upozorní nás, pokud jsou porušeny. Aby to fungovalo, musí být ArcGIS ovšem vybaven nástroji pro nalezení míst, kde jsou tato pravidla porušena, a nástroji na opravu chyb.

Vytváření a ukládání topologických vztahů poskytuje řadu výhod. Jak se později sami přesvědčíme, vede použití topologie k efektivnějšímu uložení dat. Proto je zpracování dat rychlejší a lze zpracovávat rozsáhlejší soubory. Jakmile existují topologické vztahy, lze provádět analytické funkce. To jsou např. toky v sítích, slučování přilehlých polygonů s obdobnými charakteristikami a překrývání geografických prvků.

4. Návrh převodu topologických pravidel Simple Feature Specification for SQL

V této kapitole bychom se měli zamyslet nad možným převodem již vyjmenovaných topologických pravidel do SQL (Structured Query Language). Nejdříve si proto stručně řekneme, co to SQL je. Jedná se o strukturovaný jazyk sloužící k definici objektů relačních databází a vkládání, mazání, změnu a především výběr dat z relační databáze.

V poslední době se jako úložiště dat pro aplikace geografických informačních systémů stále více používají relační databáze. Oproti klasickému ukládání dat do souboru má tento způsob řadu výhod. Mezi hlavními lze jmenovat především snadnější vyhledávání dat v relační databázi pomocí standardních SQL příkazů, dále nejsme limitováni velikostí datového souboru, jelikož se může jednat o replikovaný databázový server, který jako datový sklad používá například diskové pole, snadné sdílení dat mezi klientskými aplikacemi atd. Relačních databází vhodných jako datasource pro aplikace geografických informačních systémů je mezi komerčním software hned několik, například Oracle, DB2 a Informix, ze světa OpenSource lze jmenovat především databázi PosgtreSQL a MySQL. Pro většinu z těchto databází jsou vyvinuty nebo se vyvíjejí knihovny a programy, které umožňují dodržet standardní strukturu tabulek a funkcí pro práci s GIS daty. Tyto standardy byly definovány OpenGIS konsorciem, sdružujících řadu vývojářů a firem zabývajících se vývojem GIS aplikací. Z nejznámějších lze jmenovat například ESRI, Oracle, Informix, atd. Pro použití databází se využívá specifikace ”Simple Features Specification for SQL” která přesně popisuje struktury databázových tabulek a funkcí určených pro práci s daty. Současně platné specifikace datových typů jsou platné od roku 1999.

Možná by někdo i nadále mohl namítnout otázku proč zrovna ukládat do SQL databáze rozšířené o gisovské datové typy, když bychom taky jednoduše mohli použít například Coverage, či již zmiňovaný ShapeFile. Vede nás k tomu minimálně šest následujících důvodů, z nichž některé již byly zmiňovány v předchozím odstavci.

- Atributová složka a prostorová složka jsou na jednom místě. Názorným příkladem je následující tabulka, kde geometry, attribute 1, attribute 2 představují atributovou složku a gid, xMin, yMin, xMax, yMax představují prostorovou část.

gid	xMin	yMin	xMax	yMax	geometry	attribute 1	attribute 2
1	0	0	30	30	< WKBGeometry>	black	0
2	30	0	60	30	< WKBGeometry >	red	1
3	0	30	30	60	< WKBGeometry >	yellow	1
4	30	30	60	60	< WKBGeometry >	green	0

Tab 4: Příklad tabulky s WKB zápisem geometrické třídy typu Polygon

- Můžeme použít indexy na prostorovou i atributovou část dat, což nám podstatně urychlí práci. Je to patrné z tabulky 4, kde xMin, yMin, xMax, yMax určují envelope geometrického objektu a jsou základem prostorového indexu, jehož nerovnice je uvedena níže.

```
SELECT a.gid, b.gid FROM a, b WHERE (a.xMax >= b.x.Min)
AND (b.xMax >= a.xMin) AND (a.yMax >= b.yMin) AND (b.yMax
>= a.yMin)
```

- Spolu s XML je to dnes ukládání do SQL databáze rozšířené o gisovské datové typy nejpoužívanější způsob ukládání dat.

- Díky tomu, že budeme používat GIS funkce, můžeme celou práci, která doposud příslušela klientovi převést na server. Názornou ukázkou je několik příkladů z praxe

  Vyber všechny obce, které celé leží v kraji s názvem Liberecký!

```
SELECT nazev FROM obce WHERE Within(geometry,(SELECT
geometry FROM kraje WHERE nazev = 'Liberecký'))
```

  Ve kterém okrese se nachází letiště s názvem Kunovice?

```
SELECT nazev FROM okresy WHERE Within((SELECT geometry
FROM letiste WHERE nazev = 'Kunovice'), geometry)
```

  Vyber všechny obce, kterými prochází silnice číslo 266, přičemž silnic s číslem 266 může být více!

```
SELECT nazev FROM obce WHERE Intersects(geometry, (SELECT
Collect(geometry) FROM silnice WHERE cislo_sil = '266'))
```

Vyber všechny obce, které leží do 5000 metrů od silnice číslo 266, přičemž silnic s číslem 266 může být více!

```
SELECT nazev FROM obce WHERE Intersects(geometry,(SELECT
Buffer(Collect(geometry),5000) FROM silnice WHERE
cislo_sil = '266'))
```

- Lehká a centralizovaná správa dat, zálohování.

- Sdílení dat pro všechny uživatele.

K našemu záměru, tedy převodu topologických pravidel do SQL, by nám měla pomoci následující norma. Jedná se o OpenGIS Simple Features Specification For SQL, Revision 1.1.

Tato norma je takovým rozšířením SQL databáze o práci s datovými typy GISů. Jedná se o implementaci následujících neabstraktních datových typů: Point, MultiPoint, LineString, MultiLineString, Polygon a MultiPolygon a o abstraktní datové typy Geometry, Curve, MultiCurve, Surface a MultiSurface, které ovšem nemohou být implementovány. K těmto datovým typům náleží i příslušné funkce, pomocí kterých s nimi můžeme pracovat. Ty byly představeny ve druhé kapitole. V následující čtvrté kapitole je hlavní náplní pokus o převedení jednotlivých topologických pravidel z grafické podoby do jazyka SQL tak, abychom dostali jasné a srozumitelné algoritmy, pomocí kterých budeme moci jednotlivá pravidla definovat.

5. Přehled topologických pravidel a jejich převod do SQL

Ve všech níže uvedených příkladech předpokládejme, že tabulka s danou či testovanou vrstvou (testedLayer, testedLayerA, testedLayerB) má následující strukturu:

gid (unsigned long number)
xMin (signed long number)
yMin (signed long number)
xMax (signed long number)
yMin (signed long number)
geometry (Geometry)

Dále předpokládejme, že všechny geometrické objekty vcházející do níže uvedených funkci mají stejný souřadnicový systém. Pro sjednocení všech objektů testované vrstvy použijeme funkci Collect, která slouží právě k takovému účelu a narozdíl od funkce Union definované v Simple Features Specification for SQL má za své argumenty nula, jeden či více objektů typu Geometry. Tudíž zápis

```
Collect(SELECT geometry FROM testedLayer WHERE
Intersects(geometry, testedGeometry))
```

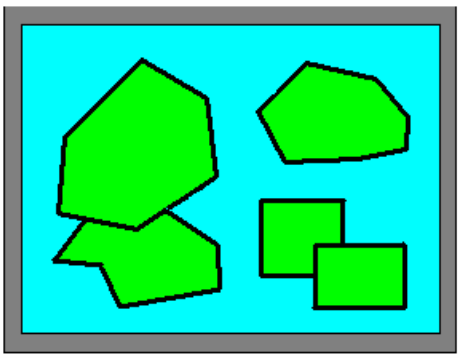
znamená: "Vyber všechny geometrické objekty z testované vrstvy, které mají neprázdný průnik s testovaným geometrickým objektem, a udělej z nich vícenásobným sjednocením jeden geometrický objekt typu GeometryCollection." Takto vzniklý objekt se dále používá v testovacích topologických funkcích, které mají jeden, či dva argumenty.

Topologická pravidla mohou být definována pro prvky ze třídy prvků nebo mezi dvěma či více třídami prvků. Např. polygony se nesmí překrývat, linie nesmí mít volné konce, body musí být na hranici ploch, třída polygonů nesmí mít mezery, linie se nesmí křížit a body musí být umístěny na koncovém bodě. Topologická pravidla mohou být nadefinována také pro subtypy ze třídy prvků. V této práci se budeme zabývat topologickými pravidly v ArcGIS. Jedná se o 25 pravidel, která jsou rozdělena na pravidla pro mnohoúhelníky (9), pravidla pro lomené čáry (12) a pravidla pro body (4).

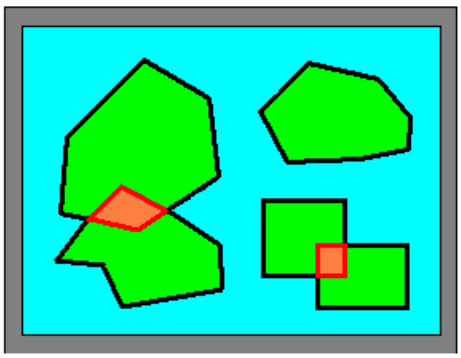
Všechna topologická pravidla budou prezentována v následující formě zápisu:

5.X.X. Název topologického pravidla

Obecný popis příkladu použití daného pravidla a příklad aplikace tohoto pravidla v praxi.



Popis topologického pravidla a příklad jeho splnění.



Popis a příklad chyb generovaných při porušení topologického pravidla. Na obrázku je chyba vyznačena červeně.

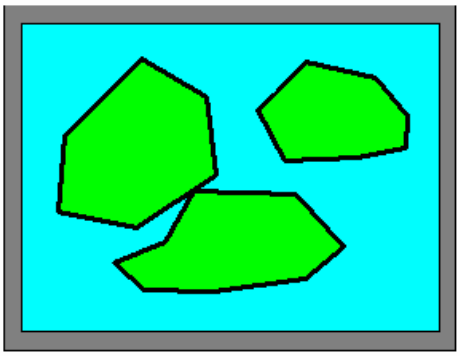
Teoretická definice převodu daného topologického pravidla do jazyka SQL.

Zápis daného topologického pravidla v jazyce SQL.

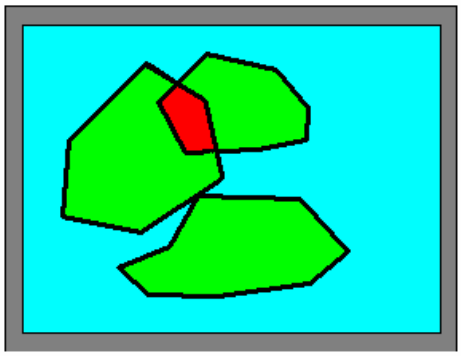
5.1. Pravidla pro mnohoúhelníky

5.1.1. Nesmí přesahovat

Toto pravidlo použijeme, chceme-li se ujistit, že v rámci jedné třídy prvků, nebo podtypu nepřesahuje žádný z mnohoúhelníků jiný mnohoúhelník. V praxi si to například můžeme ukázat na mapě volebních okrsků, kdy nesmí žádný z nich přesahovat svým rozsahem ostatní.



V rámci třídy prvků, nebo podtypu nesmí mnohoúhelníky přesahovat. Mohou být izolované, nebo se vzájemně dotýkat bodem, či hranou.



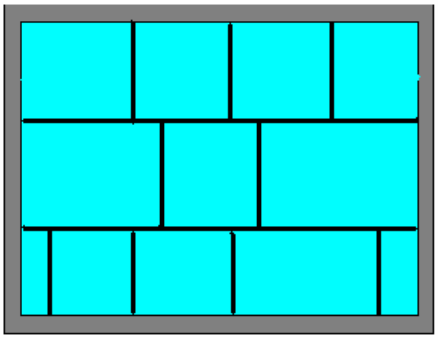
Mnohoúhelníkové chyby jsou vytvořeny v místech, kde se mnohoúhelníky vzájemně přesahují.

Mějme testovaný mnohoúhelník (testedPolygon), o kterém máme rozhodnout, zda přesahuje libovolný mnohoúhelník v dané vrstvě (testedLayer). Pomoci funkce Overlaps testujme, zda-li Overlaps(polygonFromTestedLayer, testedPolygon) vrací true. Níže uvedeny SQL příkaz v případě chyby vrací false hodnotu.

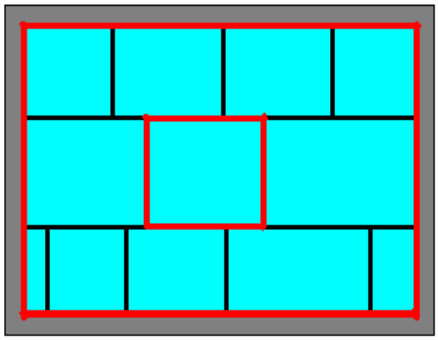
```
SELECT Overlaps(testedPolygon, Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPolygon)))
```

5.1.2. Nesmí obsahovat mezery

Toto pravidlo použijeme, pokud mají všechny mnohoúhelníky tvořit souvislý povrch bez jakýchkoli prázdných míst, nebo mezer. V praxi jsou to například mnohoúhelníky geologické mapy, které nesmí obsahovat žádné mezery ani prázdná místa – musí souvisle pokrývat území.



V rámci jedné třídy prvků, nebo podtypu nesmí být mezi mnohoúhelníky žádný prázdný prostor.

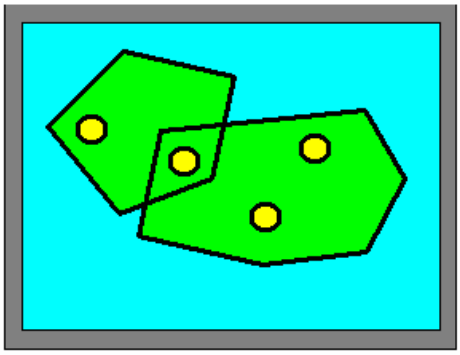


Chyby tvoří linie vnějších okrajů prázdné oblasti uvnitř mnohoúhelníku, nebo mezi . Chybami jsou i nesdílené hranice mnohoúhelníků.

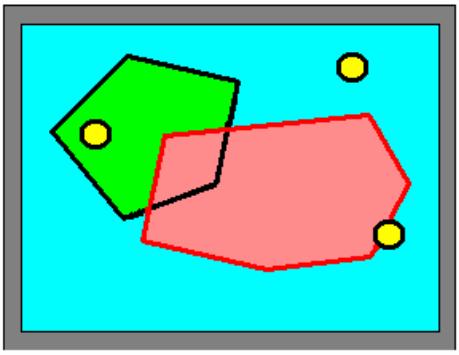
MĚJME TESTOVANÝ POLYGON (TESTED POLYGON) O KTERÉM MÁME ROZHODNOUT, ZDA MÁ OSTROVY, ČI MEZERY MEZI JINÝMI MNOHOÚHELNÍKY V DANÉ VRSTVĚ (TESTEDLAYER). POMOCÍ FUNKCE TOUCHES TESTUJEME, ZDA SE VŠECHNY POLYGONY DOTÝKAJÍ.

5.1.3. Musí obsahovat bod

Toto pravidlo použijeme, chceme-li se ujistit, že všechny mnohoúhelníky obsahují uvnitř svých hranic nejméně jeden bod. Přesahující mnohoúhelníky mohou body uvnitř průniku sdílet. Z běžného života známe jeden názorný příklad, kdy parcely musí obsahovat jeden adresní bod.



Každý mnohoúhelník první třídy prvků, nebo podtypu musí obsahovat nejméně jeden bod ze druhé třídy prvků, nebo podtypu.



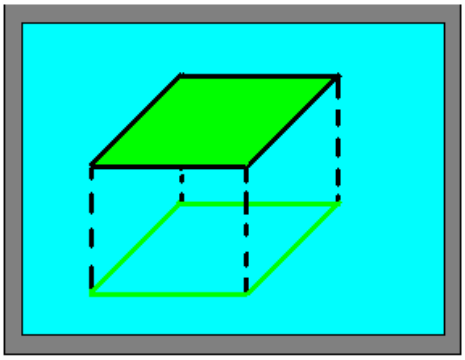
Chyby jsou tvořeny mnohoúhelníky, které neobsahují žádný bod. Body ležící na hranici mnohoúhelníku se nepovažují za body uvnitř mnohoúhelníku.

Mějme testovaný mnohoúhelník (testedPolygon), o kterém máme rozhodnout, zda obsahuje libovolný bod v dané vrstvě (testedLayer). Pomoci funkce Contains testujme, zda-li Contains(testedPolygon, pointFromTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby false hodnotu.

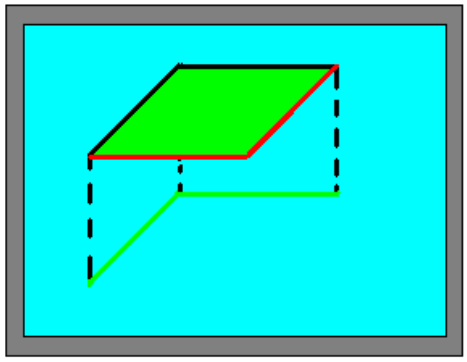
```
SELECT Contains(testedPolygon, Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPolygon)))
```

5.1.4. Hranice musí být pokryty liniemi

Toto pravidlo použijeme, pokud mají být hranice mnohoúhelníků sdíleny s další liniovou třídou prvků, nebo podtypem. Jsou to například ulice ve čtvrti rodinných domků, kdy všechny hranice parcel musí být pokryty lomenými čarami, ale všechny lomené čáry nemusí ležet na hranici parcel.



Hranice mnohoúhelníků jedné třídy prvků, nebo podtypu musí být pokryty lomenými čarami jiné třídy prvků, nebo podtypu.



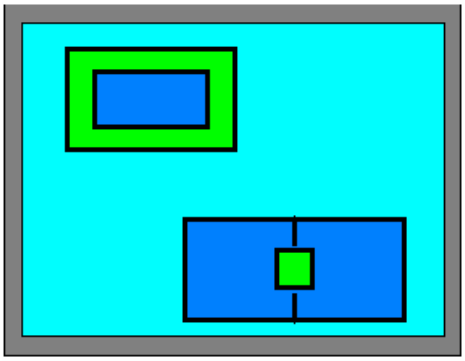
Chyby tvoří lomené čáry-hranice mnohoúhelníků, které nejsou pokryty lomenými čarami z jiné třídy prvků, nebo podtypu

Mějme testovaný mnohoúhelník (testedPolygon), o kterém máme rozhodnout, zda jeho hranice (objekt typu MultiLineString) leží uvnitř objektu typu GeometryCollection, který je sjednocením všech objektů testované vrstvy objektu typu LineString (testedLayer). Abychom urychlili testování, do testu zahrneme pouze ty objekty z testedLayer, které mají neprázdný průnik s testedPolygon. Pomoci funkce Within testujeme, zda-li Within(boundaryOfTestedPolygon, unionOfTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby hodnotu false.

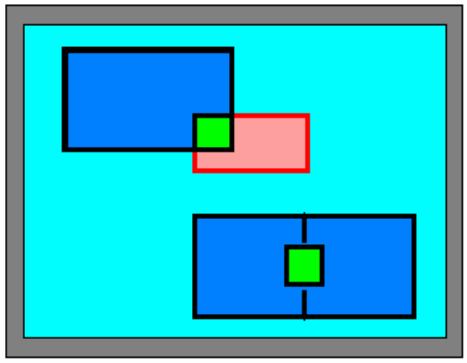
```
SELECT Within(Boundary(testedPolygon), Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPolygon)))
```

5.1.5. Musí být pokryty třídou prvků

Toto pravidlo použijeme v případech, kdy by každý mnohoúhelník jedné třídy prvků, nebo podtypu měl být zcela pokryt mnohoúhelníky druhé třídy prvků, nebo podtypu. Praktickým příkladem jsou například státy různých zemí, které jsou zcela pokryty jejich kraji, na které jsou rozděleny.



Mnohoúhelník v první třídě prvků, nebo podtypu musí být plně pokryt mnohoúhelníky ze druhé třídy prvků, nebo podtypu.



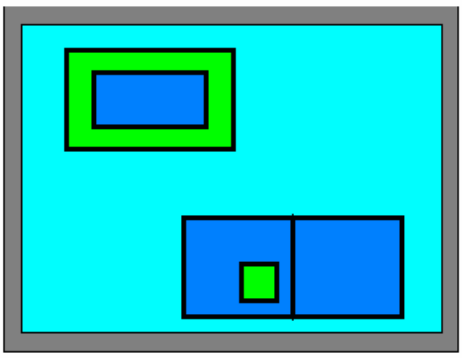
Mnohoúhelníkové chyby jsou vytvořeny z nepokrytých oblastí mnohoúhelníků z první třídy prvků

Mějme testovaný mnohoúhelník (testedPolygon), o kterém máme rozhodnout, zda leží uvnitř **?NEBO NA HRANICÍCH?** objektu typu GeometryCollection, který je sjednocením všech objektů testované vrstvy objektu typu Polygon (testedLayer). Abychom urychlili testování, do testu zahrneme pouze ty objekty z testedLayer, které mají neprázdný průnik s testedPolygon. Pomoci funkce Within testujeme, zda-li Within(testedPolygon, unionOfTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby hodnotu false.

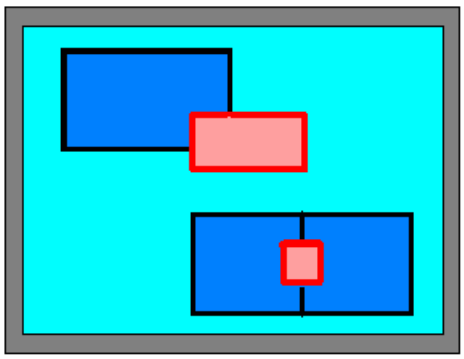
```
SELECT Within(testedPolygon, Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPolygon)))
```

5.1.6. Musí být pokryty polygonem

Pravidlo použijeme, pokud chceme, aby mnohoúhelníky v jedné třídě byly uvnitř mnohoúhelníků ve druhé třídě. Setkáváme se s tím například, když kraje musí ležet uvnitř států.



Každý mnohoúhelník v jedné třídě prvků, nebo podtypu musí být uvnitř jediného mnohoúhelníku ve druhé třídě prvků, nebo podtypu.



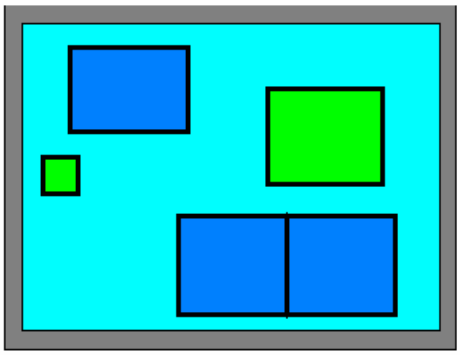
Chyby tvoří mnohoúhelníky 1. třídy prvků, nebo podtypu, které nejsou pokryty jedním mnohoúhelníkem 2. třídy prvků (podtypu)

Mějme testovaný mnohoúhelník (testedPolygon), o kterém máme rozhodnout, zda leží uvnitř libovolného mnohoúhelníku v dané vrstvě (testedLayer). Pomoci funkce Within testujme, zda-li Within(testedPolygon, polygonFromTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby nulovou hodnotu.

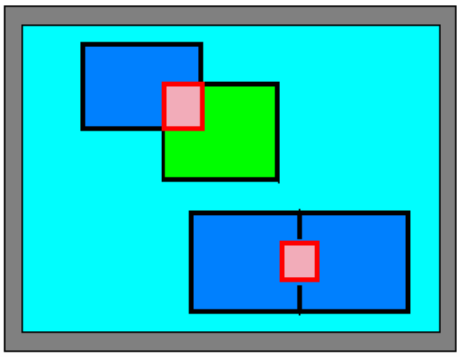
```
SELECT COUNT(gid)
FROM testedLayer
WHERE Within(testedPolygon, geometry)
```

5.1.7. Nesmí přesahovat (vztah dvou tříd)

Toto pravidlo použijeme v případech, kdy by mnohoúhelníky z jedné třídy prvků, nebo podtypu neměly přesahovat mnohoúhelníky jiné třídy prvků, nebo podtypu. V praxi to jsou například vodní plochy a parcely ze dvou různých tříd prvků, které se nesmí přesahovat.



Mnohoúhelníky z první třídy prvků, nebo podtypu nesmí přesahovat mnohoúhelníky z druhé třídy prvků, nebo podtypu.



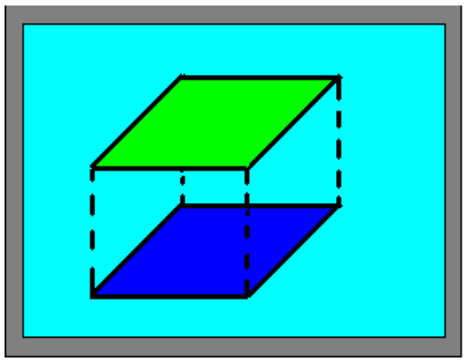
Mnohoúhelníkové chyby jsou vytvořeny v místech, kde mnohoúhelníky ze dvou tříd prvků, nebo podtypů přesahují.

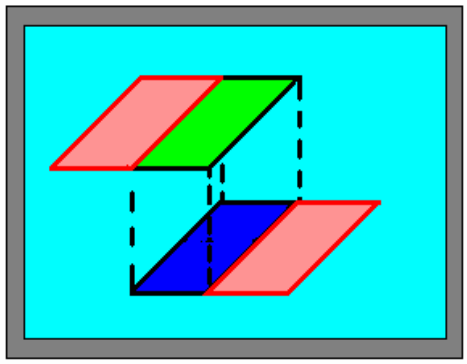
Toto pravidlo je stejné jako pravidlo „4.1.1. Nesmí přesahovat“, až na to, že zde je vztah dvou tříd a ne jedné, což ale pro SQL je jedno a to samé. Níže uvedený SQL příkaz vrací v případě chyby hodnotu true.

```
SELECT Overlaps(testedPolygon, Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPolygon)))
```

5.1.8. Musí být vzájemně pokryty

Pravidlo použijeme tehdy, pokud chceme, aby mnohoúhelníky ze dvou tříd prvků, nebo podtypů pokrývaly stejnou oblast. Jsou to například mnohoúhelníky vegetace a typů půd, které musí pokrývat stejné území.





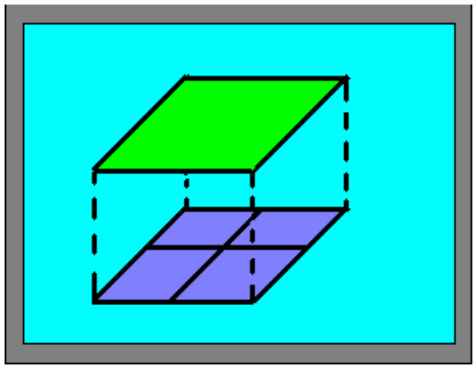
Všechny mnohoúhelníky z 1. třídy prvků a všechny mnohoúhelníky ze 2. třídy prvků se musí navzájem pokrývat. Jedna třída prvků je pokryta druhou a naopak.

Chyby tvoří mnohoúhelníky, nebo jejich části, které nejsou pokryty jedním, nebo více mnohoúhelníky ze druhé třídy prvků (podtypu).

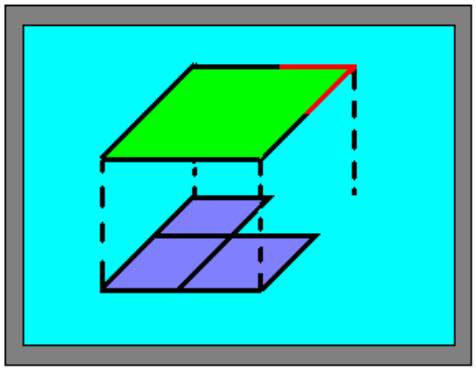
TOTO PRAVIDLO SE BOHUŽEL NEPODAŘILO PŘEVÉST DO JAZYKA SQL, I KDYŽ PATŘÍ MEZI NA PRVNÍ POHLED JEDNODUCHÁ PRAVIDLA, NEBYLA NALEZENA ŽÁDNÁ PŘÍSLUŠNÁ FUNKCE, POMOCI KTERÉ BY SE NÁSLEDNÝ PŘEVOD MOHL USKUTEČNIT.

5.1.9. Hranice musí jít po hranicích polygonů

Toto poslední pravidlo pro mnohoúhelníky použijeme v případech, kdy by měly hranice mnohoúhelníků jedné třídy prvků souhlasit s hranicemi mnohoúhelníků druhé třídy prvků, nebo podtypu. Ukázkou toho je názorný příklad, kdy hranice obytných zón jsou tvořeny hranicemi parcel, ale obytné zóny nemusí nepokrývat všechny parcely.



Hranice mnohoúhelníků jedné třídy prvků, nebo podtypu musí být pokryty hranicemi mnohoúhelníků v jiné třídě prvků, nebo podtypu.



Chyby lomených čar tvoří části hranic mnohoúhelníků z 1. třídy prvků (podtypu) nepokrytých hranicemi mnohoúhelníků 2. třídy prvků (podtypu).

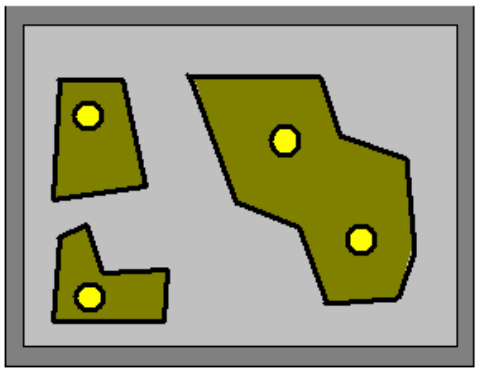
Mějme testovaný mnohoúhelník (testedPolygon), o kterém máme rozhodnout, zda jeho hranice (objekt typu MultiLineString) lezí uvnitř objektu typu GeometryCollection, který je sjednocením všech hranic objektu testované vrstvy objektu typu Polygon (testedLayer). Abychom urychlili testování, do testu zahrneme pouze ty objekty z testedLayer, které mají neprázdný průnik s testedPolygon. Pomoci funkce Within testujeme, zda-li Within(boundaryOfTestedPolygon, unionOfTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby hodnotu false.

```
SELECT Within Boundary(testedPolygon), Collect
     (SELECT Boundary(geometry)
     FROM testedLayer
     WHERE Intersects(geometry, testedPolygon)))
```

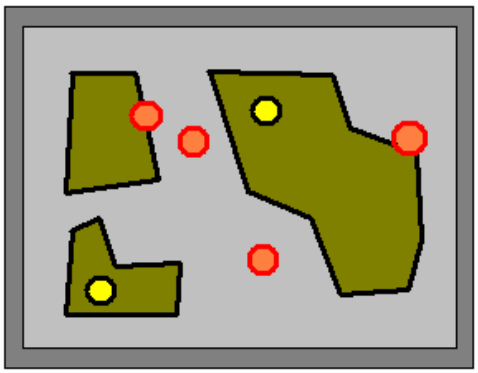
5.2. Pravidla pro body

5.2.1. Musí být uvnitř polygonu

Pravidlo použijeme tehdy, pokud chceme, aby všechny body byly umístěny uvnitř mnohoúhelníků. To například znamená, že hlavní města států musí ležet uvnitř jednotlivých států.



Body v jedné třídě prvků, nebo podtypu musí ležet uvnitř mnohoúhelníků z druhé třídy prvků, nebo podtypu.



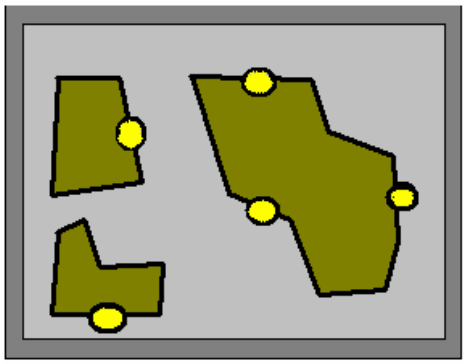
Chybami jsou body, které jsou buď mimo mnohoúhelníky, nebo leží na jejich hranicích.

Mějme testovaný bod (testedPoint), o kterém máme rozhodnout, zda leží uvnitř libovolného mnohoúhelníku v dané vrstvě (testedLayer). Pomoci funkce Contains testujeme, zda-li Contains(testedPoint, polygonFromTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby nulovou hodnotu.

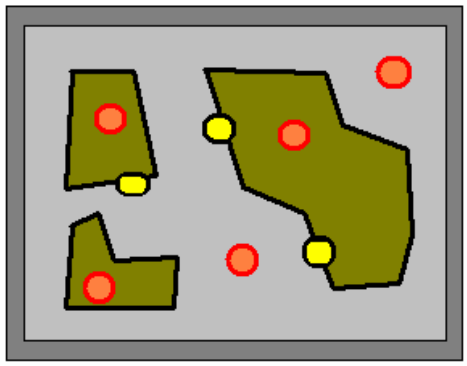
```
SELECT Contains(testedPoint, Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPoint)))
```

5.2.2. Musí ležet na hranicích polygonů

Toto pravidlo použijeme v případě, že chceme, aby všechny body ležely na hranicích mnohoúhelníků. Setkáme se s tím například u přípojek inženýrských sítí, kdy můžeme požadovat, aby ležely na hranicích parcel.



Body v jedné třídě prvků, nebo podtypu musí ležet na hranicích mnohoúhelníků z druhé třídy prvků, nebo podtypu.



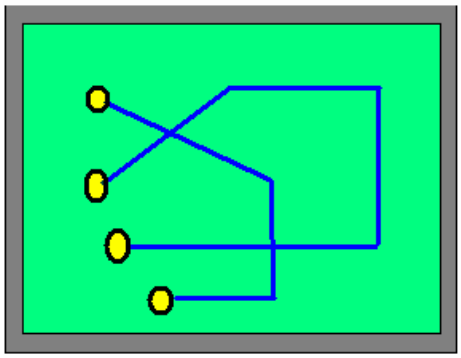
Chybami jsou body, které neleží na hranicích mnohoúhelníků.

Mějme testovaný bod (testedPoint), o kterém máme rozhodnout, zda leží na hranici libovolného mnohoúhelníku v dané vrstvě (testedLayer). Pomoci funkce Touches testujeme, zda-li Touches(testedPoint, polygonFromTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby nulovou hodnotu.

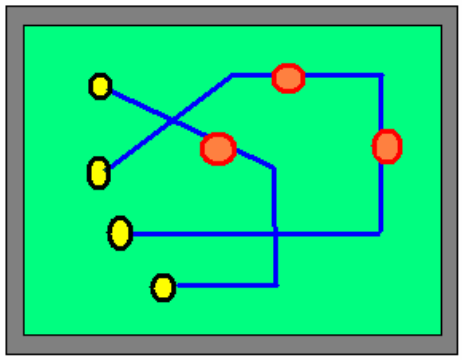
```
SELECT Touches(testedPoint, Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPoint)))
```

5.2.3. Musí být pokryty koncovými body

Toto pravidlo použijeme, pokud chceme, aby všechny body v dané třídě prvků souhlasily s konci lomených čar. Jsou to například body, reprezentující křižovatky, které musí být pokryty koncovými body středových ulic.



Body v jedné třídě prvků, nebo podtypu musí být pokryty koncovými body lomených čar jiné třídy prvků, nebo podtypu.

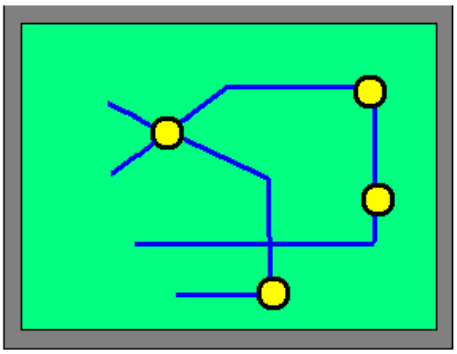


Chybami jsou body, které nejsou pokryty konci lomených čar.

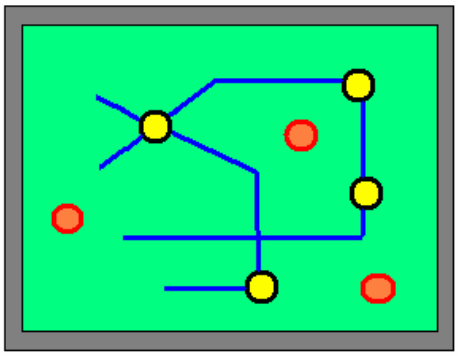
TOTO PRAVIDLO SE NEPODAŘILO PŘEVÉST.

5.2.4. Body musí ležet na liniích

Poslední pravidlo pro body použijeme tehdy, pokud chceme, aby všechny prvky v dané třídě prvků ležely na lomených čarách. V běžném životě se s tím například setkáváme v případě, že čističky odpadních vod musí ležet na vodních tocích.



Body v jedné třídě prvků, nebo podtypu musí ležet na lomených čarách jiné třídy prvků, nebo podtypu.



Chybami jsou body, které leží mimo lomené čáry.

TEN PŘEVOD JE POUZE MÁ DOMNĚNKA!!!!!!

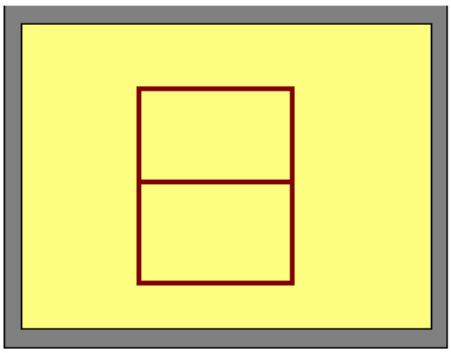
Toto pravidlo je stejné jako „4.2.2. Musí ležet na hranicích polygonů“, akorát že testujeme body a lomené čáry. Mějme testovaný bod (testedPoint), o kterém máme rozhodnout, zda leží na hranici libovolné lomené čáry v dané vrstvě (testedLayer). Pomoci funkce Touches testujeme, zda-li Touches(testedPoint, polygonFromTestedLayer) vrací true. Níže uvedený SQL příkaz vrací v případě chyby nulovou hodnotu.

```
SELECT Touches(testedPoint, Collect
     (SELECT geometry
     FROM testedLayer
     WHERE Intersects(geometry, testedPoint)))
```

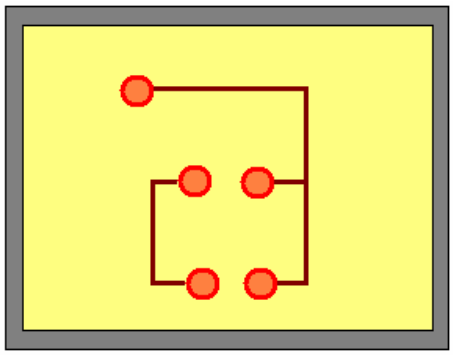
5.3. Pravidla pro lomené čáry

5.3.1. Nesmí mít volné konce

Pravidlo použijeme, pokud chceme, aby byly všechny lomené čáry ve třídě prvků, nebo podtypu vzájemně propojené. Je to například silniční síť, která je tvořena propojenými segmenty. Případy slepých ulic je možné při editaci ošetřit výjimkou.



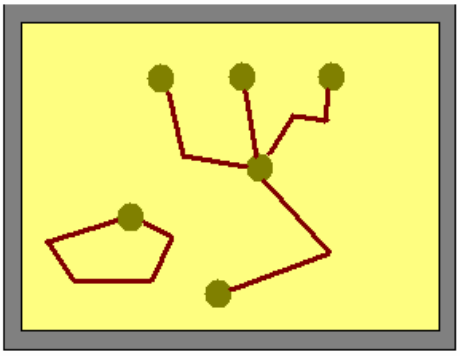
Konec každé lomené čáry v rámci jedné třídy prvků, nebo podtypu se musí dotýkat buď jiné lomené čáry, nebo sebe samé.

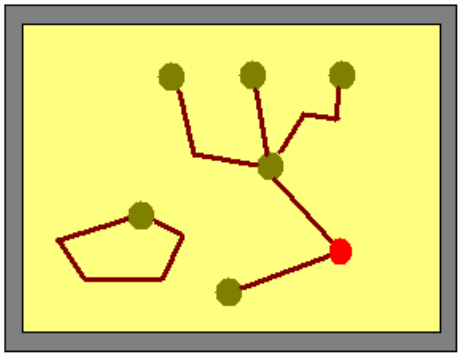


Chyby tvoří koncové body lomených čar, které se nedotýkají žádné další lomené čáry, nebo sebe samé.

5.3.2. Nesmí mít pseudonódy

Toto pravidlo použijeme, když chceme vyčistit data obsahující nevhodně rozdělené lomené čáry. Používá se to například pro hydrologickou analýzu, kdy je možné nařídit, aby segmenty lomených čar říční sítě měly nódy pouze v koncových bodech, nebo uzlech.



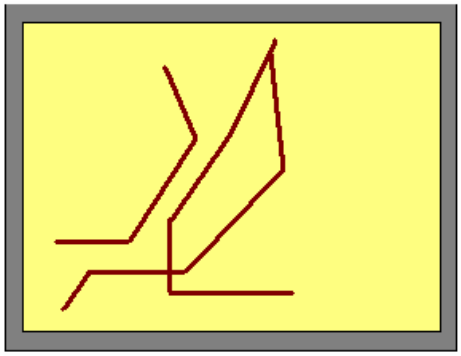


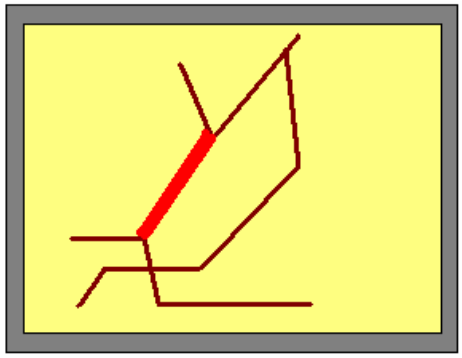
Žádná lomená čára v rámci jedné třídy prvků, nebo podtypu nesmí být ve svém koncovém bodě spojena právě s jednou jinou lomenou čarou. Lomená čára se může svým koncovým bodem dotýkat sama sebe.

Bodové chyby jsou vytvořeny v místech, kde je lomená čára spojena pouze s jednou další lomenou čarou.

5.3.3. Nesmí se překrývat

Pravidlo použijeme na lomené čáry, které by neměly mít stejný průběh s ostatními lomenými čarami. Jsou to například lomené čáry reprezentující úseky silnic, které se mohou protínat a dotýkat, ale nesmí se vzájemně překrývat.



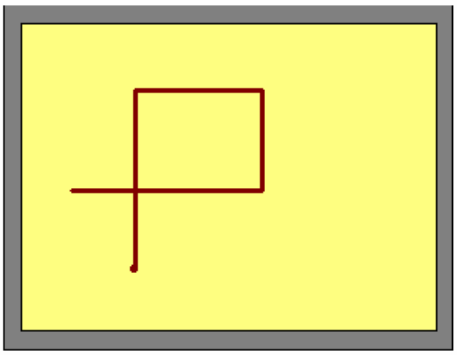


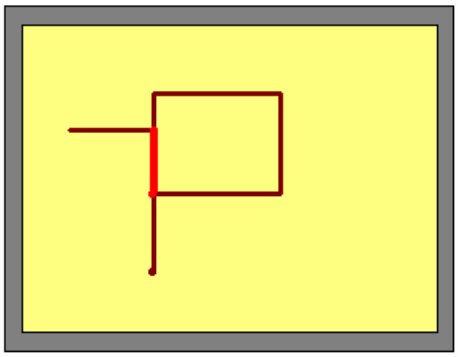
Lomené čáry se nesmí překrývat žádnou svojí částí s jinou lomenou čarou v téže třídě prvků, nebo podtypu. Lomené čáry se mohou křížit, dotýkat koncovým bodem, nebo překrývat samy sebe.

Chyby lomených čar jsou vytvořeny v místech, kde se lomené čáry překrývají.

5.3.4. Nesmí překrývat samy sebe

Následující pravidlo použijeme u lomených čar, jejichž segmenty by nikdy neměly mít stejný průběh s jinými segmenty téže lomené čáry. Použijeme to například pro dopravní analýzu, kdy by se neměly segmenty lomených čar v rámci jednoho prvku překrývat.



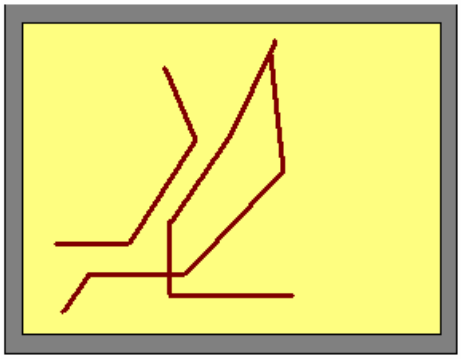


Lomené čáry se v rámci jedné třídy prvků, nebo podtypu nesmí překrývat. Mohou se dotýkat, protínat a překrývat s lomenými čarami jiné třídy prvků (podtypu).

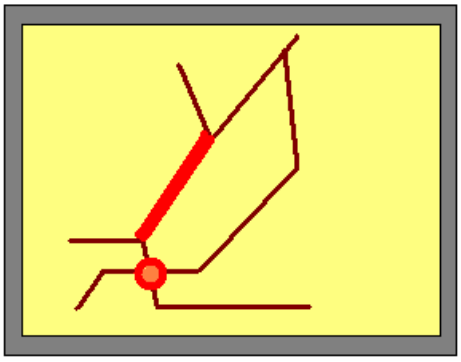
Chyby lomených čar jsou vytvořeny v místech, kde lomené čáry překrývají samy sebe.

5.3.5. Nesmí se překrývat ani protínat

Použijeme toto pravidlo na lomené čáry, jejíž segmenty by se nikdy neměly protínat, nebo mít stejný průběh s ostatními lomenými čarami. Jsou to například lomené čáry vodních toků, které se nesmí protínat ani překrývat. Koncové body se však mohou dotýkat jiného segmentu.



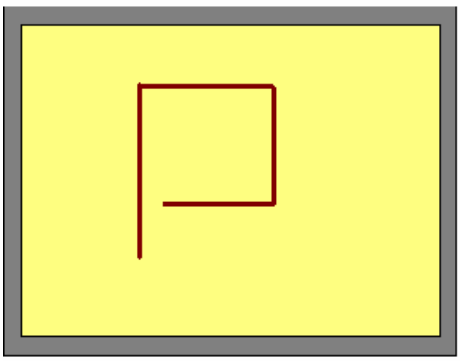
Všechny lomené čáry v rámci jedné třídy prvků, nebo podtypu se nesmí překrývat, ani se vzájemně protínat.

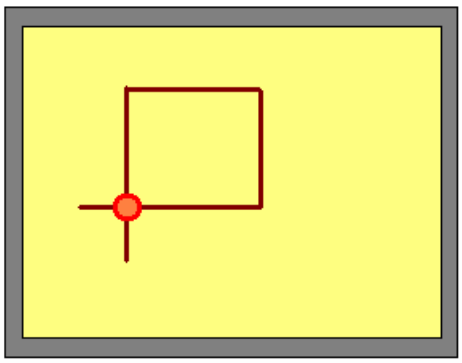


Chyby jsou tvořeny lomenými čarami v místech, kde se lomené čáry překrývají, a body tam, kde se protínají.

5.3.6. Nesmí protínat samy sebe

Toto pravidlo použijeme u lomených čar, které b se měly dotýkat pouze svými konci bez toho, aby se protínaly, nebo přesahovaly. Názorným příkladem toho jsou třeba vrstevnice, které se nesmí protínat.



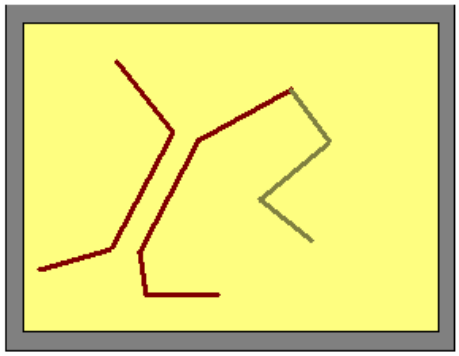


Lomené čáry v rámci jedné třídy prvků, nebo podtypu nesmí samy sebe protínat, ani překrývat. Smí se dotýkat samy sebe i jiných lomených čar a protínat a překrývat se s jinými lomenými čarami.

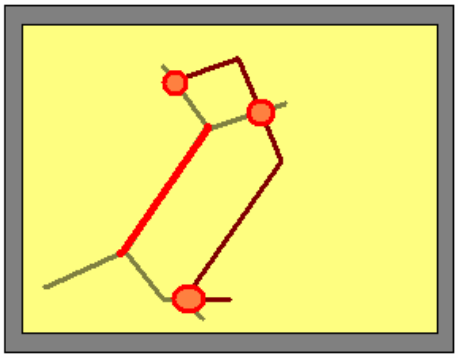
Chyby lomených čar jsou vytvořeny v místech, kde lomené čáry překrývají samy sebe, a bodové chyby tam, kde lomené čáry protínají samy sebe.

5.3.7. Nesmí se překrývat, protínat ani dotýkat (mimo konců)

Pravidlo použijeme na lomené čáry, pokud chceme, aby se vzájemně dotýkaly pouze svými konci, ale neprotínaly se ani se nepřekrývaly. Jsou to například lomené čáry hranic parcel, které se nesmí vzájemně překrývat, dotýkat ani protínat. Jsou vzájemně propojeny pouze svými koncovými body.



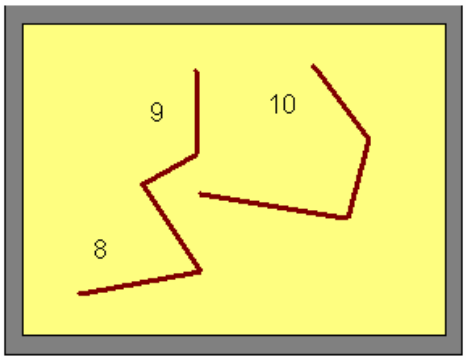
Všechny lomené čáry v rámci jedné třídy prvků, nebo podtypu se smí dotýkat jen svými konci a nesmí se překrývat ani protínat.



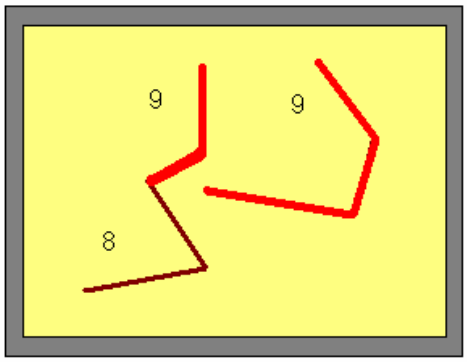
Chyby jsou tvořeny lomenými čarami v místech, kde se lomené čáry překrývají, a body, kde se protínají nebo dotýkají (kromě konců).

5.3.8. Musí mít jedinou část

Toto pravidlo použijeme, když chceme, aby každá lomená čára byla složena pouze z jedné řady navzájem propojených segmentů. Jedná se například o silniční systém, který je vytvořen z jednotlivých prvků, přičemž žádný z nich nemá více než jednu část.



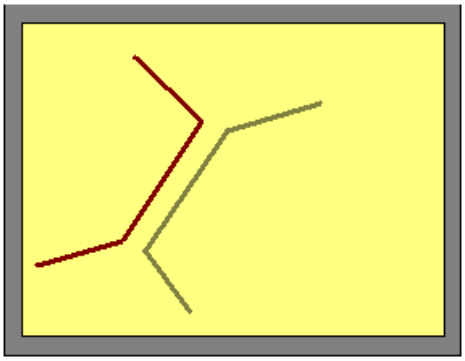
Lomené čáry v rámci jedné třídy prvků, nebo podtypu musí být tvořeny pouze jednou částí.



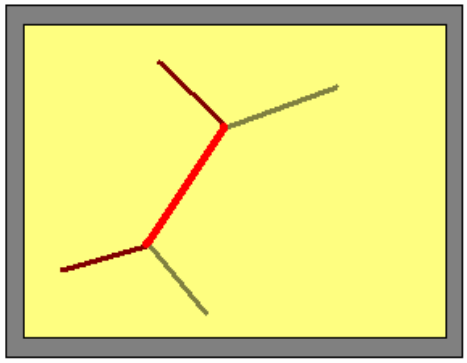
Chyby jsou tvořeny vícenásobnými lomenými čarami v místech, kde lomené čáry mají více než jednu část.

5.3.9. Nesmí se překrývat (vztah dvou tříd)

Toto pravidlo použijeme na lomené čáry, které by nikdy neměly mít shodný průběh jako lomené čáry z jiné třídy prvků, nebo podtypu. Setkáváme se s tím například u silnic, které mohou protínat řeky, nebo se k nim přibližovat, ale nesmí se s nimi vzájemně překrývat.



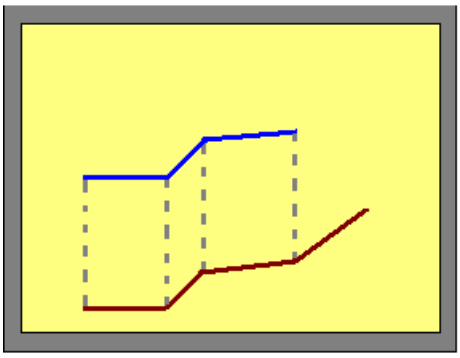
Všechny lomené čáry v jedné třídě prvků nebo podtypu nesmí překrývat žádnou část lomené čáry ze druhé třídy prvků, nebo podtypu.



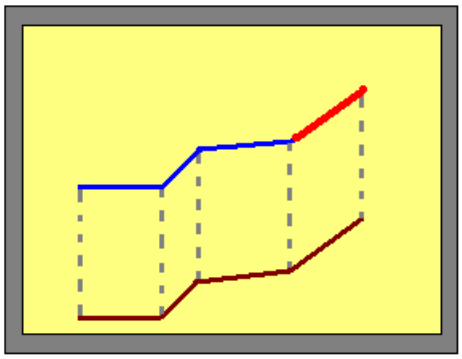
Chyby lomených čar jsou vytvořeny v místech, kde se lomené čáry ze dvou tříd prvků, nebo podtypu překrývají.

5.3.10. Musí být pokryty třídou prvků

Pravidlo použijeme, pokud máme více skupin lomených čar, které mají mít stejný průběh. Jsou to například lomené čáry vyznačující trasy městských autobusů, a ty musí vést po lomených čarách uličních sítí.



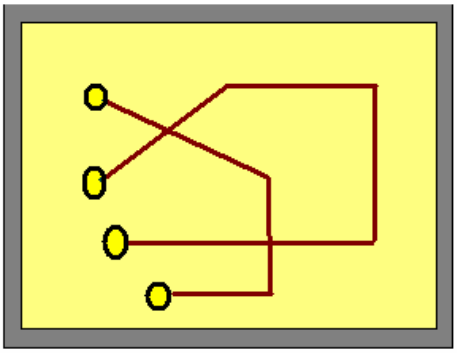
Lomené čáry v jedné třídě prvků, nebo podtypu musí být pokryty lomenými čarami v jiné třídě prvků, nebo podtypu.



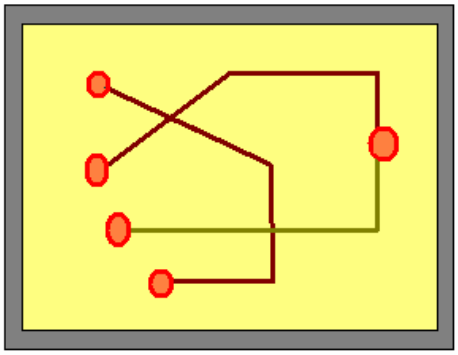
Chyby jsou tvořeny vícenásobnými lomenými čarami v místech, kde lomené čáry mají více než jednu část.

5.3.11. Koncové body musí být pokryty

Toto pravidlo použijeme, pokud chceme modelovat konce lomených čar z jedné třídy prvků, nebo podtypu, které se kryjí s bodovými prvky jiné třídy prvků. Vyskytuje se to například v energetice. V koncových bodech lomené čáry přípojky elektrického vedení musí být buď transformátor, nebo elektroměr.



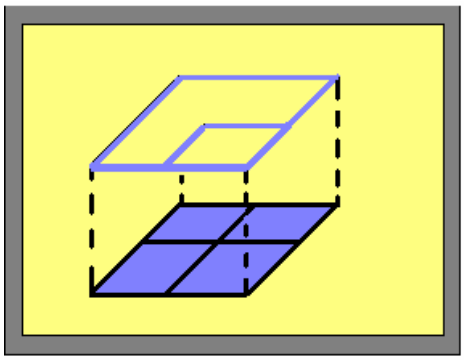
Konce lomených čar jedné třídy prvků, nebo podtypu musí být pokryty body z jiné třídy prvků, nebo podtypu.



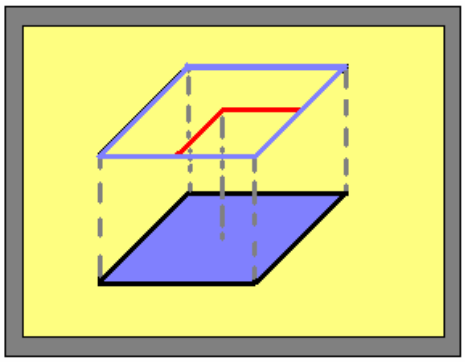
Bodové chyby jsou vytvořeny na koncích lomených čar, které nejsou pokryty bodem.

5.3.12. Musí ležet na hranicích polygonů

Toto poslední pravidlo pro lomené čáry použijeme tehdy, pokud mají být totožné s hranicemi mnohoúhelníků. Názorným příkladem pro toto pravidlo je například sčítání hlasovacích lístků při volbách. Hranice sčítacích obvodů jsou zčásti tvořeny lomenými čarami hlavních silnic. (Všechny hlavní silnice jsou zároveň hranicemi sčítacích obvodů).



Lomené čáry v jedné třídě prvků, nebo podtypu musí být pokryty hranicemi mnohoúhelníků jiné třídy prvků, nebo podtypu.



Chyby jsou tvořeny lomenými čarami, které nejsou pokryty hranicemi mnohoúhelníků

6. Použité zdroje:

6.1. Literatura

[1] OpenGIS: *Simple Features Specification For SQL, Revision 1.1*, OpenGIS, 1999.

[2] Časopis ArcRevue 4/2003

[3] Časopis ArcRevue 3/2002

[3] Davis E. David: *Publikace GIS for Everyone,* 1999

[4] ARCDATA PRAHA s.r.o.: *Seznamte se s GIS, firemní manuál,* ARCDATA PRAHA s.r.o., 1993

[5] ARCDATA PRAHA s.r.o.: *Manuál PostGIS 0.9.0*, ARCDATA PRAHA s.r.o.2004

[6] ESRI.: *ArcGIS 9, začínáme s ArcGIS,* ESRI, 2004

[7] Rybář Štěpán Ing.: *Spatial Functions in MySQL 4.1,* 2004

[8] Benák Karel Ing.: *Použití relačních databází pro uložení dat GIS programů a programu TOPOS*, 2002

[9] Plakát: *Topologická pravidla v geodatabázi ArcGIS,* ESRI, 2002

6.2. Internetové stránky

[10] klobouk.fsv.cvut.cz/~beny/skola/pj10/postgis.pdf

[11] http://postgis.refractions.net

[12] http://www.opengis.org

[13] http://www.vividsolutions.com/jts/JTShome.htm

6.3. Data

[14] ARC ČR 2.0 pro Jihočeský kraj

6.4. Programové vybavení

[15] ArcGIS Desktop, verze 9.0.

[16] JTS (J Topology Suite), verze 1.5

[17] PostGIS, verze 0.9.0

[18] Jump (J Unified Mapping Platform), verze 1.1.2