UNIVERZITA PARDUBICE
Fakulta elektrotechniky a informatiky

Optimalizace vybraných vyhledávacích operací v rámci multidimenzionálních dat nad databází Oracle Spatial

Jan Merta

Bakalářská práce
2013

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2012/2013**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jan Merta**
Osobní číslo:	**I10137**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Optimalizace vybraných vyhledávacích operací v rámci multidimenzionálních dat nad databází Oracle Spatial**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

V teoretické části práce bude popsána technologie Oracle Spatial z pohledu uchovávání multidimenzionálních dat.
V praktické části se student zaměří na návrh optimalizaci vyhledávacích operací v rámci detekce pozice pohybujících se objektů v modelu železniční sítě.
Navržená řešení budou otestována v demonstrační aplikaci.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

1. MURRAY, Chuck, et al. Oracle? Spatial : User's Guide and Reference 10g Release 2 (10.2) [online]. [s.l.] : Oracle, 2006 Dostupné z WWW: http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14256.pdf
2. MURRAY, Chuck, et al. Oracle? Spatial : Topology and Network Data Models 10g Release 2 (10.2) [online]. [s.l.] : Oracle, 2005. Dostupné z WWW: http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14256.pdf
3. MOLNÁR, J.: Porovnání intervalového vyhledávání nad vhodnou databází, Univerzita Pardubice, 2012

Vedoucí bakalářské práce: **Ing. Jan Fikejz**
Katedra softwarových technologií

Datum zadání bakalářské práce: **21. prosince 2012**
Termín odevzdání bakalářské práce: **10. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 29. března 2013

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 16. 5. 2013 Jan Merta

Poděkování

Na tomto místě bych rád poděkoval vedoucímu mé bakalářské práce Ing. Janu Fikejzovi za odbornou pomoc, cenné rady a připomínky, které mi pomohly při zpracování bakalářské práce. Dále bych rád poděkoval své rodině a přátelům nejen za morální podporu během studia.

Anotace

V teoretické části práce je popsána technologie Oracle Spatial z pohledu uchovávání multidimenzionálních dat. Praktická část je zaměřena na návrh optimalizací vyhledávacích operací v rámci detekce pozice pohybujících se kolejových vozidel modelu železniční sítě. Navržená řešení jsou otestována v demonstrační aplikaci.

Klíčová slova

Oracle Spatial, detekce, poloha, železnice, optimalizace, vyhledávání, databáze, vícerozměrná data, prostorová data

Title

Optimization of selected search operations within the multidimensional data over database Oracle Spatial

Annotation

The theoretical part describes Oracle Spatial technology from the viewpoint of multidimensional data storage. The practical part is focused on the design of optimization of search operations within detection of position of rolling stock objects in the railway network models. Proposed solutions are tested in demonstration application.

Keywords

Oracle Spatial, detection, position, railway, optimization, search, database, multidimensional data, spatial data

Obsah

Seznam zkratek

BP	Bakalářská práce
ČR	Česká republika
SQL	Structured Query Language
PL/SQL	Procedural Language/Structured Query Language
XML	Extensible Markup Language
JDBC	Java Database Connectivity
DML	Data Manipulation Language
DDL	Data Definition Language
GPS	Global Positioning System
OOP	Objektově orientované programování
GUI	Grafické uživatelské rozhraní (Graphical User Interface)
CSV	Comma-Separated Values
GIF	Graphics Interchange Format
URL	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

Úvod

Cílem této práce je navrhnout a otestovat optimalizaci vybraných vyhledávacích operací v databázi Oracle se zaměřením na vícerozměrná data s použitím rozšiřující technologie Oracle Spatial.

První část bakalářské práce popisuje způsob uložení prostorových dat v databázi Oracle a základní operace při práci s vícerozměrnými daty. Představuje doplňkovou technologii Oracle Spatial, která je velmi užitečnou nadstavbou obsahující předdefinované datové typy, mnoho potřebných funkcí a procedur, které jsou odladěné pro efektivní a rychlou práci s vícerozměrnými údaji a komplexní analýzu prostorových dat. Popisuje základní rysy Oracle Spatial, použité datové typy a v neposlední řadě uvádí oficiální postupy pro vytváření tabulek obsahujících prostorová data včetně indexů a metadat.

Tato část pojednává též o propojení databázového stroje s programovacím jazykem Java, teoreticky srovnává různé postupy při tvorbě dotazů volaných z počítačových aplikací a shrnuje předpoklady ohledně optimalizace databázových dotazů.

Další část je zaměřena na konkrétní návrh optimalizace vyhledávacích operací v rámci detekce pozice kolejových vozidel v modelu železniční sítě vhodným typem dotazů a použitím dynamických databázových pohledů omezením zdrojových dat. Stanovuje hypotézy, které plánuje ověřit pomocí demonstrační aplikace.

Čtvrtá část zahrnuje zejména návrh, implementaci a dokumentaci demonstrační aplikace s jádrem diskrétní simulace a animací zobrazující pohyb kolejových vozidel. Stanovuje si požadavky na aplikaci, které se pak snaží dodržet. V neposlední řadě rozebírá znalosti, techniky a technologie použité při tvorbě celého programu. Definuje strukturu několika databázových tabulek, které se využívají pro uložení prostorových dat. Dále popisuje důležité části projektu a stěžejní třídy. Uvádí souvislosti mezi nimi a snaží se čtenáři přiblížit jejich funkcionalitu a vysvětlit řešení jednotlivých problémů.

Poslední část práce se zabývá testováním původních předpokladů a realizovaných řešení. Srovnává v ní různé velikosti tabulek s multidimenzionálními daty, způsoby pro tvorbu dotazů a hledá optimální velikost dynamického pohledu pro omezení zdrojových dat v rámci urychlení detekce přesné pozice kolejových vozidel v rámci železniční sítě. Závěrem nechybí ani nezbytné shrnutí dosažených poznatků a doporučení vhodného postupu při optimalizaci vyhledávacích operací nad vícerozměrnými daty.

1 Databáze Oracle

Pro uchování vícerozměrných dat a práci s nimi byl zvolen databázový stroj Oracle Database 11g ve verzi Enterprise edition s doplňkem Oracle Spatial.

V současné době podporuje například vlastní implementaci standardního dotazovacího jazyka SQL, XML a programovací jazyk PL/SQL pro vytváření procedur, funkcí a dalších objektů. Samotný databázový stroj obsahuje optimalizátor pro provádění jednotlivých příkazů.

1.1 Instalace databáze

Instalace databáze pomocí instalačního průvodce pod operačním systémem Windows se skládá z několika triviálních kroků. Jediné, na co je potřeba dávat pozor, je uživatelem zadané umístění aplikace na pevném disku, jméno databáze a heslo k databázi pro pozdější přihlášení pod správcovským účtem. Při zapomenutí hesla nebo v případě nainstalování půlky aplikace na externí harddisk je pravděpodobné, že nebude možné s databázovým strojem plnohodnotně pracovat. Současně snaha zbavit se původní instalace nebude snadnou a samozřejmou záležitostí.

1.2 Oracle Spatial

Oracle Spatial je nadstavbovou technologií obsahující předdefinované datové typy, potřebné funkce a procedury, které jsou odladěné pro efektivní a rychlou práci s vícerozměrnými daty. V databázi lze uchovávat nejen prostorové body, úsečky ale i složitější geometrické útvary jako jsou například polygony, oblouky, kruhy atd.

Pro zvýšení výkonu využívá různých optimalizačních mechanismů včetně indexace obsahu pomocí R-strom či Quad-strom indexu. V nejnovější verzi se doporučuje používat výhradně R-strom index, protože se na něj tvůrci zaměřili v poslední době nejvíce a v Oracle Spatial se prozatím jedná o nejlepší možnou volbu. Balík disponuje celou řadou pokročilých a šikovných operací, ale v rámci bakalářské práce bylo využito v podstatě pouze ukládání dvojrozměrných bodů reprezentujících železniční síť a vestavěné funkce pro nalezení nejbližšího souseda *SDO_NN*.

1.3 Práce s vícerozměrnými daty v Oracle Spatial

Pro práci s prostorovými daty používá Oracle sloupce s datovým typem *SDO_GEOMETRY*. Při vytváření tabulek obsahujících tento datový typ je potřeba dodržet následující postup skládající se z několika jednoduchých kroků:

1. Vytvoření tabulek pro prostorová data
2. Aktualizování prostorových metadat
3. Naplnění prostorových tabulek daty
4. Vytvoření prostorového indexu

1.3.1 Vytvoření tabulek pro prostorová data

Vytvoření tabulky pro prostorová data lze předvést na příkladu tabulky uchovávající body železniční tratě. Jednoduchý SQL příkaz vytvoří tabulku *BODY_ZELEZNICE* se sloupci *id_uzlu*, *jmeno* a *umisteni* s datovým typem *SDO_GEOMETRY* pro prostorová data.

```
CREATE TABLE BODY_ZELEZNICE (
      id_uzlu NUMBER PRIMARY KEY,
      jmeno VARCHAR2(10),
      umisteni SDO_GEOMETRY
);
```

1.3.2 Aktualizování prostorových metadat

Aktualizace metadat se pak provádí pomocí SQL příkazu *INSERT* do pohledu *USER_SDO_GEOM_METADATA*.

```
INSERT INTO USER_SDO_GEOM_METADATA
VALUES (
      'BODY_ZELEZNICE',
      'UMISTENI',
      SDO_DIM_ARRAY(
            SDO_DIM_ELEMENT('X', 0, 20, 0.00005),
            SDO_DIM_ELEMENT('Y', 0, 60, 0.00005)
      ),
      8307
);
```

Předchozí příklad přidá informace o tabulce *BODY_ZELEZNICE* a prostorovém sloupci *UMISTENI*. Dimenzi *X* nastaví na interval 0-20, dimenzi *Y* na interval 0-60 a přesnost na 0.00005. Číslo 8307 definuje používaný souřadnicový systém.

1.3.3 Vložení záznamu do tabulky s prostorovými daty

Data se do tabulky se přidávají pomocí příkazu *INSERT* a na prostorový sloupec je nutné použít konstruktor datového typu *SDO_GEOMETRY*.

```
MDSYS.SDO_GEOMETRY(2001,8307,MDSYS.SDO_POINT_TYPE(16.44,49.89,null),
  null,null);
```

První parametr deklaruje, o jaký typ prostorových dat jde. Číslo 2001 ukazuje, že jde o bod o dvou dimenzích. Druhý parametr s hodnotou 8307 představuje vybraný souřadnicový systém a vnořený konstruktor *SDO_POINT_TYPE* pomocí parametrů předá hodnoty 16.44 pro souřadnici *X*, 49.89 pro souřadnici *Y* a *NULL* pro souřadnici *Z*. Protože se jedná o bod, poslední dva parametry konstruktoru mají hodnotu *NULL*.

Celý *INSERT* lze tedy napsat následujícím způsobem.

```
INSERT INTO BODY_ZELEZNICE (ID,JMENO,UMISTENI)
VALUES (
      433,
      'N433',
      MDSYS.SDO_GEOMETRY(
            2001,
            8307,
            MDSYS.SDO_POINT_TYPE(
                  16.44,
                  49.89,
                  null
            ),
            null,
            null
      )
);
```

1.3.4 Vytvoření prostorového indexu

Poté již zbývá vytvořit index nad sloupcem s prostorovými daty.

```
CREATE INDEX BODY_ZEL_IDX ON BODY_ZELEZNICE UMISTENI)
  INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Příkaz z příkladu vytvoří index se jménem *BODY_ZEL_IDX* nad tabulkou *BODY_ZELEZNICE* a sloupcem *UMISTENI*. Od této chvíle lze s tabulkou a daty plnohodnotně pracovat.

1.3.5 Dotaz na nejbližšího souseda

Dotaz na nejbližšího souseda zadaného bodu by v našem případě mohl vypadat takto:

```
SELECT id, SDO_NN_DISTANCE(1) as vzdalenost FROM BODY_ZELEZNICE
where SDO_NN(
      BODY_ZELEZNICE.UMISTENI,
      MDSYS.SDO_GEOMETRY(
            2001,
            8307,
            MDSYS.SDO_POINT_TYPE(
                  16.50,
                  50.03,
                  NULL
            ),
            NULL,
            NULL
      ),
      'SDO_NUM_RES = 1',
      1
) = 'TRUE';
```

Uvedený příklad umožňuje vyhledat ve sloupci *UMISTENI* z tabulky *BODY_ZELEZNICE* nejbližší bod s jeho *id* a vzdálenost v metrech od bodu o souřadnicích [16.50, 50.03] vytvořeného pomocí konstruktoru *SDO_GEOMETRY*.

'SDO_NUM_RES = 1' nám říká, že chceme 1 výsledek. Pokud bychom chtěli nejbližší sousedy třeba 3, napíšeme *'SDO_NUM_RES = 3'*. Poslední parametr funkce se používá, pokud dotaz obsahuje funkci *SDO_NN_DISTANCE*. Při jejím volání je totiž nutné jí jako parametr předat stejné číslo, které bylo zvoleno v příslušné funkci *SDO_NN*. V tomto případě to je tedy číslo 1, tedy *SDO_NN_DISTANCE(1)*.

1.4 Propojení databáze Oracle s programovacím jazykem Java

Pro propojení programovacího jazyka Java a databázového stroje Oracle lze použít standardní balík tříd JDBC. JDBC v Javě definuje jednotné rozhraní pro propojení a práci s různými relačními databázemi.

1.4.1 Ovladače

Pro jednotlivé databáze jsou dodávány ovladače, pomocí kterých klientský program přes JDBC ovládá danou databázi. Oracle není samozřejmě výjimkou a příslušný ovladač OJDBC lze získat z oficiálních stránek výrobce nebo přímo z instalačního adresáře databázového stroje v adresáři *C:\app\uživatel\product\11.2.0\dbhome_1\jdbc\lib*. Tyto knihovny je poté nutné naimportovat do projektu vyvíjené aplikace a dodávat je s ní v případě, že nebudou na cílovém systému k dispozici.

Před tvorbou připojení k databázi Oracle je potřeba načíst její JDBC ovladač. Jednou z možností je použití metody *Class.forname()*.

```
String ovladac = "oracle.jdbc.driver.OracleDriver";
Class.forName(ovladac); // načtení ovladače pro DB
```

1.4.2 Objekt Connection

Objekt typu Connection reprezentuje spojení s databází a poskytuje metody pro získávání informací o databázovém serveru, nastavování transakcí nebo vytváření objektů typu Statement apod.

Po načtení ovladače do programu stačí již jen k vybrané databázi vytvořit spojení s daným databázovým serverem na určeném portu. Jednoduchý příklad ukazuje spojení typu THIN s databází Oracle *orcl1* na adrese *localhost* (127.0.0.1) a portu 1521.

```
String url = "jdbc:oracle:thin:@127.0.0.1:1521:orcl1;
Connection spojeni =
      DriverManager.getConnection(url, uzivatelskeJmeno, heslo);
```

Připojení je realizováno pomocí metody *DriverManager.getConnection()*, které je předána složená adresa URL ve tvaru:

```
jdbc:oracle:thin:@adresa_serveru:číslo_portu:jméno_databáze
```

Dalšími parametry metody jsou uživatelské jméno a heslo pro přístup k danému databázovému účtu.

1.5 Techniky dotazů

Funkce pro nalezení nejbližšího souseda je stěžejním prvkem pro hledání přesné pozice objektu, který se pohybuje po železniční trati. Důležité je zjistit, jaký typ dotazu nám nejrychleji vrátí výsledek této operace. Následující část teoreticky porovnává veškerá pro a proti jednotlivých technik používaných pro tvorbu dynamických dotazů.

1.5.1 Objekt Statement

Hlavním úkolem instancí třídy Statement je provádění databázových příkazů. Jejich použití umožňuje:

- vracet výsledky dotazů,
- vytvářet tabulky,
- vkládat, aktualizovat a mazat řádky v tabulkách,
- vytvářet a upravovat pohledy a ostatní databázové objekty,
- práci s procedurami jazyka PL/SQL
- a obecně umí vykonávat jakékoli DDL či DML operace.

Instance třídy Statement se vytváří pomocí tovární metody objektu Connection.

```
Statement stmt = spojeni.createStatement();
ResultSet vysledek = stmt.executeQuery("SELECT * FROM TABULKA1");
```

První řádek vytvoří objekt *stmt* typu *Statement* a druhý řádek pak pomocí metody *executeQuery()* zavolá dotaz a jako výsledek vrátí instanci třídy *ResultSet*. Ta lze pak zpracovat pomocí cyklu *while*, metody *next()* a příslušných metod *getX(int cisloSloupce)* či *getX(String jmenoSloupce)*.

Následující příklad do proměnné *pomocnaDouble* postupně ukládá hodnoty jednotlivých řádků z 1. sloupce, které vždy hned vypíše na standardní výstup.

```
while (vysledek.next()) {
      double pomocnaDouble = vysledek.getDouble(1);
      System.out.print(pomocnaDouble);
}
```

Pro získání hodnoty z vybraného sloupce a aktuálního řádku existují pro různé datové typy dvojice metod getX() jako:

- getString(), getClob()
- getInt(), getDouble(), getLong(),
- getDate(), getBlob().

Pomocí objektu Statement lze sestavit i dotaz na nejbližšího souseda, který ve sloupci železničních bodů vyhledá bod s nejmenší vzdáleností od bodu, který je funkci *SDO_NN* předán jako parametr. V následujícím příkladu je ukázán dynamicky složený dotaz, který pro bod [16,24; 49,87] najde nejbližšího souseda ve sloupci *UMISTENI* tabulky *UZLY_CR*.

```
Double x = 16.24;
Double y = 49.87;
String tabulka = "BODY_ZELEZNICE";
String dotazNejblizsi =
  "select * from " + tabulka + " s "
  + "where SDO_NN(s.UMISTENI,MDSYS.SDO_GEOMETRY(2001, 8307, "
  + "MDSYS.SDO_POINT_TYPE(" + x + ", " + y + ", NULL), NULL, NULL), "
  + "'SDO_NUM_RES = 1', 1) = 'TRUE'";
Statement stmt = spojeni.createStatement();
ResultSet vysledek = stmt.executeQuery(dotazNejblizsi);
```

1.5.2 Objekt PreparedStatement

Vytváření dotazů (a obecně SQL příkazů) pomocí skládání řetězců a metod třídy *Statement* může sice na první pohled vypadat jednoduše, ale určitě se nejedná o nejelegantnější řešení. V rámci vyšší bezpečnosti, komfortu a vlastně i optimalizace nám Java nabízí řešení v podobě služeb třídy *PreparedStatemen*t s vázanými proměnnými.

Třída totiž používá specializované metody *setX(poradiParametru, hodnota)* na skládání SQL příkazů, které jsou schopny vybrat a doplnit do výrazu správný datový typ se všemi jeho náležitostmi, takže se například v případě řetězce a metody *setString(1,"řetězec")* nemusíme starat o to, zda jde o *CHAR* či *VARCHAR2*. Zároveň dojde doplnění hodnoty *"řetězec"* na místo prvního zástupného znaku *"?"* uvnitř SQL řetězce a kolem dosazené hodnoty se automaticky doplní jednoduché uvozovky, takže odpadá i problém se zápisem složitých složených řetězců.

Celý princip spočívá v použití základního SQL příkazu a vázaných proměnných. Databázový stroj tak dostává stále stejný příkaz SQL lišící se pouze jinými hodnotami parametrů a tím šetří práci optimalizátoru. Protože se využívá již jednou vytvořený exekuční plán, který je stále uložen v paměti, dochází ke zrychlení práce s databází.

```
for (int i = 0; i < 100; i++) {
    String tabulka = "BODY_ZELEZNICE";
    PreparedStatement nejblizsiStmt;
    String dotazNejblizsi =
        "select * from " + tabulka + " s "
        + "where SDO_NN(s.UMISTENI,MDSYS.SDO_GEOMETRY(2001, 8307, "
        + "MDSYS.SDO_POINT_TYPE(?, ?, NULL), NULL, NULL),
        + "'SDO_NUM_RES = 1', 1) = 'TRUE'";
    nejblizsiStmt = vlak.getSpojeni().prepareStatement(dotazNejblizsi);
    // nastavení prvního ? na reálné číslo 3*i/2*i
    nejblizsiStmt.setDouble(1, 3*i/2*i);
    // nastavení druhého ? na reálné číslo 2*i/3*i
    nejblizsiStmt.setDouble(2, 2*i/3*i);
    ResultSet vysledek = nejblizsiStmt.executeQuery();
}
```

Pokud si objekt *PreparedStatement* připravíme dokonce jen jednou a dále voláme příkaz použitím stejného objektu pouze s přenastavením parametrů pomocí příslušných metod *setX()* a *execute()/executeQuery()* dochází k dalšímu zrychlení.

```
String tabulka = "BODY_ZELEZNICE";
PreparedStatement nejblizsiStmt;
String dotazNejblizsi =
    "select * from " + tabulka + " s "
    + "where SDO_NN(s.UMISTENI,MDSYS.SDO_GEOMETRY(2001, 8307, "
    + "MDSYS.SDO_POINT_TYPE(?, ?, NULL), NULL, NULL),
    + "'SDO_NUM_RES = 1', 1) = 'TRUE'";
nejblizsiStmt = vlak.getSpojeni().prepareStatement(dotazNejblizsi);
for (int i = 0; i < 100; i++) {
    // nastavení prvního ? na reálné číslo 16.24
    nejblizsiStmt.setDouble(1, 3*i/2*i);
    // nastavení druhého ? na reálné číslo 49.87
    nejblizsiStmt.setDouble(2, 2*i/3*i);
    ResultSet vysledek = nejblizsiStmt.executeQuery();
}
```

Stále však musíme skládat řetězce kvůli dynamickému doplnění názvu tabulky, protože navázání proměnných funguje jen na ty části SQL příkazu, které lze v SQL standardně parametrizovat.

1.5.3 Volání funkce uložené v databázi

Další možností je využití jakékoli funkce vytvořené přímo v databázi. Proměnné uvnitř procedur a parametry vstupující do nich jsou totiž vždy vázané. Příkaz SQL napsaný uvnitř funkce využívající vypočtené či předané hodnoty se tak automaticky optimalizuje na připravený příkaz s proměnnou sadou parametrů.

Mějme jednoduchou funkci *FUNKCE_ORACLE* obalující dotaz na nejbližšího souseda s parametry x a y pro předání souřadnic porovnávaného bodu.

```
CREATE OR REPLACE
FUNCTION FUNKCE_ORACLE(x NUMBER,y NUMBER) RETURN NUMBER AS
id_n number;
BEGIN
  /* dotaz uvnitř funkce musí předávat alespoň jednu hodnotu ven,
     jinak dojde k chybě při kompilaci */
  SELECT s.node_id INTO vzdal FROM BODY_ZELEZNICE s
  WHERE SDO_NN(s.umisteni,MDSYS.SDO_GEOMETRY(2001, 8307,
    MDSYS.SDO_POINT_TYPE(x,y, NULL), NULL, NULL),
    'SDO_NUM_RES = 1', 1) = 'TRUE';
  RETURN id_n;
END;
```

Tuto funkci lze v Javě zavolat jednoduchým způsobem. Zajímavé však je, že pro dosazení parametrů funkce nelze použít konstrukci *PreparedStatement* a vázanné proměnné.

```
Double x = 16.24;
Double y = 49.87;
String tabulka = "BODY_ZELEZNICE";
Statement stmt = spojeni.createStatement();
dotazNejblizsi = "select FUNKCE_ORACLE(" + x + "," + y + ") from dual";
ResultSet rs = stmt.executeQuery(dotazNejblizsi);
```

Tento způsob dotazu by měl mít podobnou rychlost provádění jako správně zvolený *PreparedStatement*, otázkou však je, kolik času spotřebuje volání obalové funkce a získání výsledků.

1.6 Optimalizace pomocí pohledů

Se zvětšujícími se tabulkami vyvstává otázka, zda objem zdrojových dat ovlivňuje rychlost vyhledávání konkrétních záznamů i přes existenci pokročilých indexovacích algoritmů. Přestože se lze určitě spolehnout na to, že tvůrci databázového stroje Oracle dospěli k odladěným a optimalizovaným řešením, mohou být tyto prostředky příliš obecné.

V databázi poskytnuté pro účely BP je téměř 100 000 záznamů reprezentujících železničních body České republiky. Dá se předpokládat, že funkce *SDO_NN* pro hledání nejbližšího souseda zadaného bodu bude ovlivněna mohutností množiny bodů, které bude muset prozkoumat, aby našla správný výsledek. Počítá se však i s tím, že samotná implementace funkce obsahuje mechanismy, které prohledávanou oblast zmenšují a zkracují tak dobu hledání.

Pokud bude aplikace v jednu chvíli potřebovat pracovat jen s omezeným okruhem dat a získávání výsledků bude na tomto zúženém prostoru rychlejší a pokud se zároveň počítá s tím, že se interval tohoto okruhu jistý čas nemění a po tomto čase se změní podle jistého pravidla, nabízí se stanovení hypotézy, která předpokládá, že lze databázovému stroji pomoci nějakým vlastním inteligentním řešením.

Pokud vlak jedoucí po trati ověřuje svou přesnou polohu vyhledáváním v databázi, není nutné prohledávat oblast ve velikosti celé České republiky. Stačí si stanovit jistou oblast v okolí vlaku. Tento okruh by bylo možné definovat jako dynamicky vytvořený databázový pohled, který zdrojová data dočasně omezí pomocí podmínky na polohu bodu.

```
CREATE OR REPLACE VIEW POHLED_DEMO_BODY
AS
  FROM BODY_ZELEZNICE s
  WHERE s.umisteni.sdo_point.x >= 15.66686319334385
  AND s.umisteni.sdo_point.x   <= 15.835496806656149
  AND s.umisteni.sdo_point.y   >= 50.06320795107337
  AND s.umisteni.sdo_point.y   <= 50.11715204892661;
```

Hledání lze poté provádět přímo na pohledu a ne na celé tabulce. Když bude tato oblast dostatečně velká, pohybující se vlak použije k ověření své pozice stejný pohled před jeho přepočtením vícenásobně. Díky tomu by stroj mohl využívat již jednou vytvořeného exekučního plánu a tím ulehčit databázovému stroji. K ověření této hypotézy je zapotřebí navrhnout, implementovat a otestovat demonstrační aplikaci.

2 Demonstrační aplikace

Hlavním cílem praktické části BP bylo navrhnout aplikaci pro detekci pozice kolejových vozidel v modelu železniční sítě, která ověří stanovené optimalizační předpoklady z teoretické části. Bylo nutné vytvořit spojení s databází Oracle, navrhnout a implementovat třídy pro tvorbu dynamických dotazů, pohledů a funkcí, připravit pomocné moduly pro uložení a výpočet souřadnic GPS. Důležité bylo též implementovat jednoduché jádro diskrétní simulace pro napodobení skutečného železničního provozu.

2.1 Požadavky na aplikaci

Aby byla aplikace dostatečně demonstrační, měla by umožňovat:

- načítání trati z databáze,
- vykreslení trati a vlaků s jejich pohledy na plátno,
- přidání vlaku na trať (zvolení velikosti obdélníkového pohledu a barvy)
- simulaci pohybu vlaků po železnici,
- načtení průjezdu vlaku ze souboru,
- provádění testů různých optimalizačních řešení,
- zobrazení výsledků testů a statistik,
- uložení výsledků pro pozdější zpracování.

2.2 Použité technologie

Jako programovací jazyk byla vybrána Java ve verzi 6 zejména z hlediska rychlosti vývoje na této platformě a také pro její jednoduché a efektivní spojení s databází Oracle. Toto spojení je realizováno pomocí balíku tříd JDBC a příslušného ovladače JDBC.

Mezi další použité nástroje patří:

- integrované vývojové prostředí NetBeans 7.2.1,
- Microsoft Excel,
- databáze Oracle 11g (SQL a PL/SQL),
- a nadstavbová technologie Oracle Spatial pro práci s prostorovými daty.

2.3 Metodika

Celá aplikace se snaží respektovat základní principy a koncepty OOP a zbytečného neopakování se při psaní kódu. Simulační jádro diskrétní simulace běží ve vlastním vláknu, aby neblokovalo GUI a zároveň nemělo problém s výkonem. Další samostatná vlákna se

spouštějí při načítání trati a při běhu testů. Při použití vláken je synchronizován přístup ke sdíleným prostředkům.

2.4 Programovací techniky

Při programování demonstrační aplikace bylo využito znalostí a programovacích technik z předmětů Databázové systémy I, Databázové systémy II, Datové struktury, Modelování a simulace, Objektově orientované programování, Počítačová grafika a Programovací techniky v jazyce Java.

Jedná se zejména o:

- práci s kolekcemi, iterátory a komparátory,
- vícevláknové programování,
- generické programování,
- práci s databází, použití SQL a PL/SQL,
- implementaci návrhových vzorů,
- diskrétní simulaci a její implementaci,
- dynamické vykreslování pomocí třídy Graphics2D.

2.5 Použité datové struktury

Aplikace využívá již připravených datových struktur, které jsou v podobě kolekcí, implementovány přímo v Javě. Jedná se o optimalizované nativní implementace od tvůrců jazyka. Mezi použité kolekce a datové struktury patří:

- pole s proměnlivou délkou ArrayList (načítání procesů),
- spojový seznam LinkedList (seznam vykreslovaných objektů na plátně),
- prioritní fronta PriorityBlockingQueue (kalendář událostí v simulaci).

Všechny tyto kolekce podporují genericitu a při deklaraci jsou tedy parametrizovány příslušným datovým typem.

2.6 Návrhové vzory

Aplikace používá následující návrhové vzory a postupy:

- Jednoduchá tovární metoda,
- Jedináček (Singleton),
- Výčtový typ,
- Knihovní třída.

2.6.1 Jednoduchá tovární metoda

Jedná se o statickou metodu, která vrací referenci na instanci své třídy a oproti použití operátoru *new* má několik výhod. Návrhový vzor *Jednoduchá tovární metoda* je v BP využíván jako součást použití návrhového vzoru *Jedináček* ve třídě *SimulacniJadro,* kde zajišťuje, že dojde k vytvoření jen jedné instance simulačního jádra.

2.6.2 Jedináček (Singleton)

Jedináček se používá v případě, že potřebujeme třídu, která bude mít jen jednu instanci. Taková instance je poté globálně dostupná přes jméno třídy a příslušnou tovární metodu *getInstance()*.

Implementace tohoto vzoru má dvě vlastnosti:

- soukromý konstruktor (zamezí vytvoření instance pomocí operátoru *new*),
- *Jednoduchá tovární metoda* (vždy vrátí odkaz na stejnou instanci).

Důležité je ošetřit *Jedináčka* při použití vláken. V tomto případě je nutné prohlásit atribut instance za *volatile*. Tím naznačíme kompilátoru, že nemá provádět optimalizaci pomocí lokální kopie proměnné, protože přístup k ní může být proveden z více míst. Dále je potřeba tvorbu nového jedináčka uzavřít do bloku *synchronized*, ve kterém se ještě jednou ověří, zda již mezitím nebyl vytvořen.

Tento vzor a postup používá třída *SimulacniJadro*. Bylo by totiž nevhodné mít v aplikaci více než jednu instanci jádra diskrétní simulace a zároveň lze k jádru přistupovat z více vláken. V případě třídy *Databaze* bylo nad tímto návrhovým vzorem uvažováno též, později se však ukázalo, že toto řešení by neumožňovalo připojit se k více databázím současně.

2.6.3 Výčtový typ

Výčtový typ považovat za zobecnění jedináčka. Umožňuje vytvoření omezeného počtu instancí definovaných výčtem. V BP jsou takové třídy dvě a to:

- eAzimut (definující azimut pro 4 základní směry),
- eTypDotazu (definující výčet různých technik dotazů).

2.6.4 Knihovní třída

Třída *MojeMat* s prozatím jedinou metodou *zaokrouhli()* je naprogramována jako třída knihovní s veřejně přístupnými statickými metodami a konstantami. Třída má soukromý konstruktor, nikdy se nevytváří její instance a nelze od ní vytvářet potomky. Ke statickým metodám či konstantám se přistupuje pomocí jména třídy a tečky.

2.7 Využití simulace

Pro namodelování provozu po železnici bylo v aplikaci nutné implementovat jádro diskrétní simulace. Celé řešení zahrnuje i vizualizaci zkoumaných objektů v podobě

jednoduché animace, která zachycuje pohyb vlaků po trati. Následující část pojednává o základních pojmech a rysech simulace, přičemž vždy jako příklad uvádí konkrétní využití v rámci detekce polohy vlaků na železnici.

2.7.1 Základní pojmy

Simulace se společně s modelováním zabývá studiem zkoumaných objektů reálného světa. Jedná se o objekty, které buď existují, nebo by existovat mohly. V rámci BP je modelován pohyb vlaku po existující železniční síti. Obvykle není potřeba popsat tyto objekty v jejich celé složitosti, ale stačí zavést abstrakce (neboli systémy) s popisem vlastností, které souvisí se záměrem prováděného zkoumání. Vytvořená simulace sloužící k demonstraci detekce pohybujících se objektů nijak neřeší například cestující, náklad či mechanismus výhybek na trati. Objekty, které existují mimo zkoumaný systém, ale přesto je třeba počítat s jejich existencí, jsou nazývány okolím systému.

Podle významu času se systémy dělí na:

- statické systémy (čas se zanedbává),
- dynamické systémy (čas se nezanedbává).

Dynamický systém poté existuje pouze v okamžicích, kdy se vykonává nějaká akce. V rámci detekce vlaků se sleduje systém v okamžicích, kdy vlak pošle informaci o změně své polohy. V tu chvíli se provádějí akce pro výpočet přesné pozice. Mezi těmito okamžiky systém vlastně neexistuje a čas se skokově mění. Rychlost plynutí simulačního času ovlivňuje délka simulačního kroku, která určuje pauzu mezi časovými skoky.

Aktivity představují jednotlivé činnosti vykonávané během simulace. Mohou být spojité (mění stav systému v průběhu celé své existence) a diskrétní (existuje a mění stav systému pouze ve chvíli svého ukončení). Událost lze charakterizovat jako ukončení diskrétní aktivity včetně provedení příslušných změn. Posloupnost na sebe navazujících aktivit tvořících jeden logický celek se nazývá proces. Proces pro změnu pozice vlaku se poté skládá z přijetí signálu o jeho nepřesné pozici, výpočtu jeho přesné polohy na trati a přesunu vlaku na tuto vypočtenou pozici.

2.7.2 Algoritmizace simulačního modelu

Pro provádění simulačních experimentů je často nutné implementovat vlastní jádro simulace. Algoritmizace simulačního modelu obsahuje několik kroků, které zajistí její přesnost a efektivitu. Jedná se hlavně o:

- návrh datových struktur,
- plynutí simulačního času,
- zajištění synchronizace stavových změn.

Výběrem vhodné datové struktury se zajistí, že práce s událostmi měnících stav systému bude rychlý a efektivní. Pokud se jedná o dynamický systém, je nutné realizovat nějakým

způsobem plynutí simulačního času a zajistit správnou závislost událostí na plynoucím čase. Pro synchronizaci simulačního výpočtu existuje několik postupů. Pro potřeby jednoduché diskrétní simulace je vhodné použít jednu z nejrozšířenějších metod.

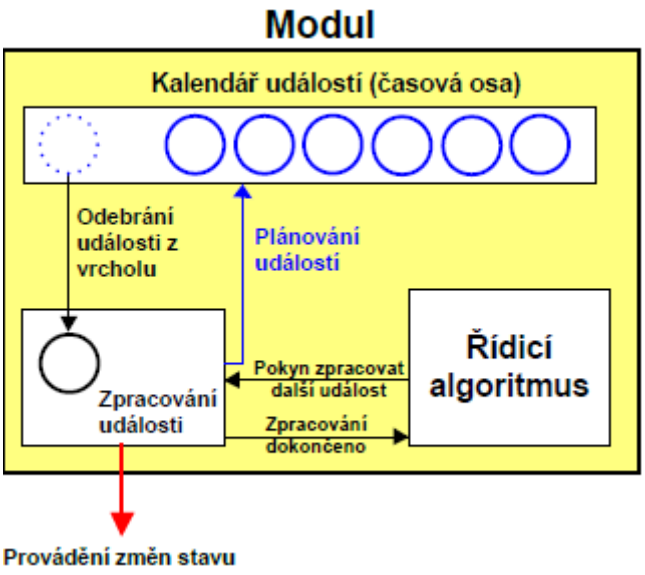
2.7.3 Metoda plánování událostí

Pro potřeby jednoduché diskrétní simulace je vhodné použít tuto metodu, která je založena na plánování výskytů událostí do budoucnosti. Diskrétní simulace se skládá pouze z diskrétních aktivit, jejichž doba a tudíž i čas ukončení (změn stavu) je předem znám. Jednotlivé události jsou uchovávány v kalendáři událostí, což je datová struktura postavená na datové struktuře prioritní fronta.

Kalendář obsahuje události, které mají v sobě *časové razítko* reprezentující výskyt konce události v budoucnosti, a metodu *Proveď*, která v okamžiku výskytu události provede příslušné změny stavu systému. Dále může poskytovat metodu *Plánuj*, která je schopna dynamicky naplánovat další události a vložit je do *Kalendáře*.

Průběh simulace je poté řízen algoritmem, který cyklicky odebírá z „vrcholu“ *Kalendáře* jednotlivé události a provádí příslušné změny stavu systému (viz Obrázek 1).

0. Inicializace času t_s na hodnotu 0
1. Má simulace pokračovat? (ukončení pokud je *Kalendář* prázdný nebo vypršel *čas*)
2. Odeber událost *U* s nejmenším časovým razítkem t_u z kalendáře
3. Aktualizace času $t_s = t_u$
4. Vykonej akce spojené s výskytem události *U*
5. Návrat na krok 1



Obrázek 1 - Algoritmus metody plánování událostí (zdroj (1))

Všechny tyto postupy jsou implementovány v samostatném modulu, který se nazývá simulační jádro. Simulační jádro poté umožňuje monitorovat simulaci a aktivně do ní zasahovat.

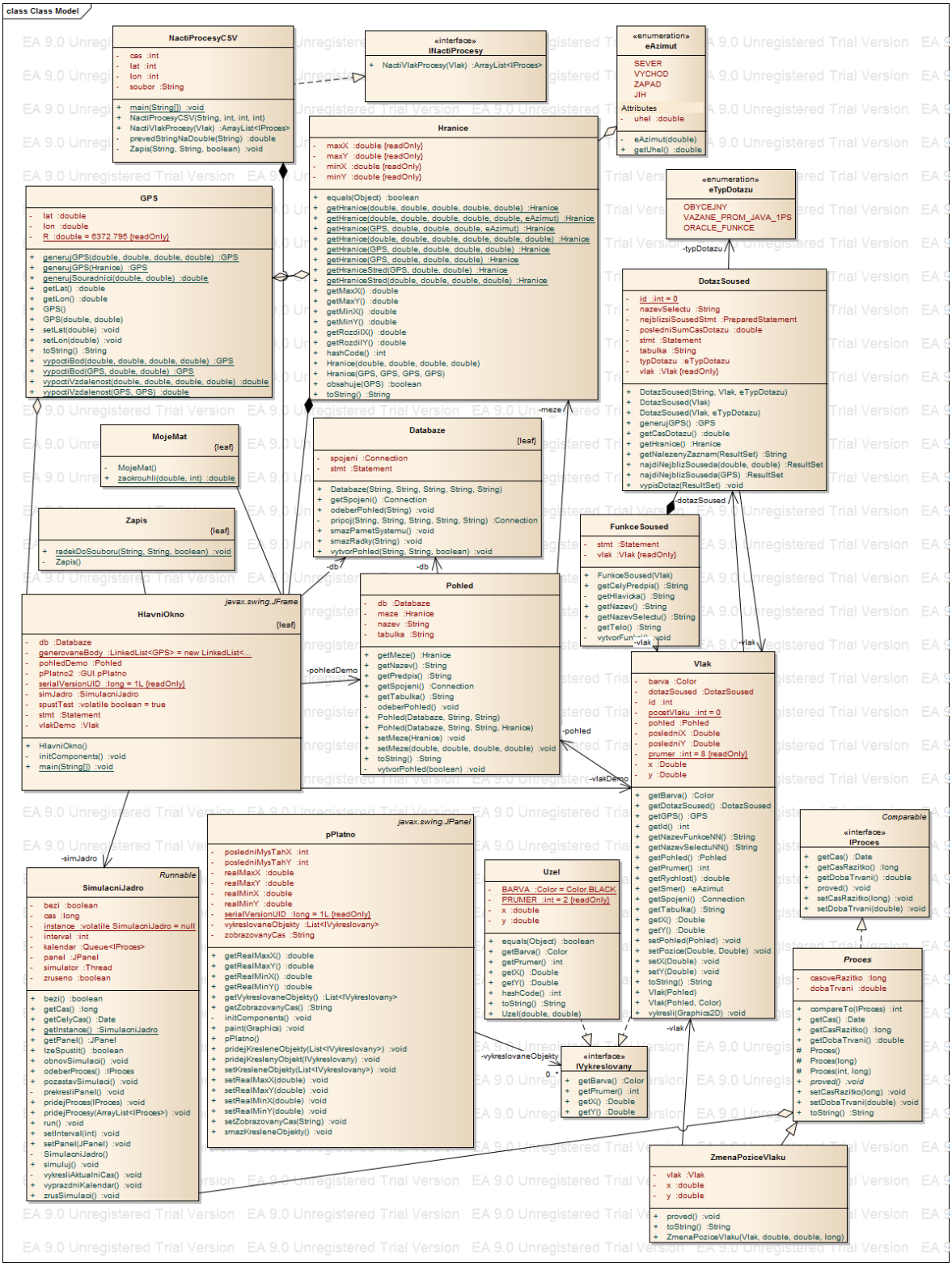
2.8 Koncepce aplikace

Projekt demonstrační aplikace se skládá z 5 logických balíčků

- uživatelské rozhraní (GUI),
- databáze (DB),
- vstupy a výstupy (IO),
- simulace (sim)
- a ostatní pomocné třídy (Ostatní).

Balíček *GUI* obsahuje hlavní okno aplikace a realizuje celé uživatelské rozhraní. V balíčku *DB* jsou uloženy třídy pro práci s databází, vytváření dynamických databázových pohledů, dotazů a funkcí. Sekce *IO* se zabývá souborovým načítáním a ukládáním, *sim* obsahuje implementaci simulačního jádra a procesů a v balíčku *Ostatni* se nacházejí pomocné modul pro práci s GPS souřadnicemi a třída týkající výpočtů mezí pro tvorbu dynamických pohledů.

2.8.1 Diagram tříd



Obrázek 2 - UML diagram návrhu demonstrační aplikace

2.8.2 Model databáze

Aplikace využívá databázový stroj Oracle 11g a v ní databázi jménem *orcl2*. Připojuje se k ní jako správce *SYS*. Model databáze není nijak rozsáhlý či složitý. V databázi byly vytvořeny dvě tabulky, které se následně naplnily poskytnutými daty (sql, xls). První z nich reprezentuje oblast Pardubického a Královehradeckého kraje (*RAILWAY_NODE$*) a druhá obsahuje železniční body v rozsahu celé ČR (*UZLY_CR*).

```
CREATE TABLE RAILWAY_NODE$ (
  NODE_ID NUMBER PRIMARY KEY,
  NODE_NAME VARCHAR2(10),
  NODE_TYPE VARCHAR2(10),
  ACTIVE CHAR(1),
  PARTITION_ID NUMBER(2,0),
  GEOMETRY SDO_GEOMETRY,
  ROUTE VARCHAR2(15),
  TUDU VARCHAR2(10),
  KM number,
  KV CHAR(1)
);
```

Jednotlivé záznamy pak evidují:

- identifikační číslo uzlu,
- jméno uzlu,
- umístění uzlu vyjádřené GPS souřadnicemi,
- traťový definiční úsek TUDU atd.

Tyto tabulky jsou využívány v demonstrační aplikaci pro vykreslení trati a zjišťování přesné pozice vlaku. Třetí tabulka *CASY_SELECTU* slouží k uchování časů dotazů a vzdáleností nejbližšího souseda z posledního proběhlého testu.

Dále se v databázi dynamicky vytváří pohledy a funkce pro hledání nejbližšího souseda, které klientská aplikace využívá v rámci optimalizace detekce objektů pohybujících se po železniční trati.

2.9 Implementace

Následující část se zabývá popisem stěžejních tříd aplikace, které byly implementovány podle požadavků na aplikaci a koncepčního návrhu aplikace.

2.9.1 Třída Databaze

Tato třída se stará o spojení aplikace s databází Oracle a při existenci více instancí by měla umožňovat práci s více databázemi najednou. Spojení je implementováno soukromou metodou *pripoj()*, která je volána přímo v konstruktoru. Tomu je nutné předat adresu serveru, jméno databáze, číslo naslouchajícího portu a uživatelské jméno s heslem.

Vytvořený objekt se poté o spojení postará a uchová ho pro další použití. V případě, že dojde k nějaké chybě, vyvrhne výjimku typu *SQLException*.

Dále třída obsahuje metody pro vytváření či odebírání pohledů, smazání řádků dané tabulky a vyčištění systémové paměti databázového stroje.

2.9.2 Třída Vlak

Objekty typu Vlak v sobě uchovávají:

- identifikátor vlaku,
- aktuální pozici vlaku,
- barvu pro vykreslení,
- pohled pro dotaz na nejbližšího souseda
- a objekt *dotazSoused* typu *DotazSoused*.

Atribut *dotazSoused* slouží k ověřování přesné pozice kolejového vozidla na železniční síti a proto mu v konstruktoru vlak předá sám sebe. Metody této třídy většinou jen zjišťují či nastavují hodnoty soukromých atributů. Za zmínku stojí hlavně metody *setPozice(Double x, Double y)* sloužící pro přesun vlaku, *getGPS()* vracející aktuální souřadnice vlaku a metoda *getDotazSoused()*, která vrací přidružený objekt typu *DotazSoused* pro ověření pozice. Každý vlak obsahuje metodu pro vygenerování názvu tohoto dotazu a případné funkce v databázi.

Pro účely testování a měření je potřeba dotaz pojmenovat, aby šel poté snáze najít v tabulce *v$sql*, kde lze v neposlední řadě zjistit dobu jeho trvání. Jméno dotazu je složeno z řetězce VLAK a unikátního identifikačního čísla vlaku. Název obalové funkce se použije, pokud zvolíme dotaz pomocí vygenerované funkce uložené v databázi. Tu je nutné dynamicky vytvořit za běhu aplikace a její jméno má tvar *FCN_VLAK_id_vlaku*.

Vlak implementuje ještě rozhraní *IVykreslovany*, které zajišťuje, že obsahuje metody nutné pro vykreslení vlaku na plátno v GUI.

2.9.3 Třída Pohled

Modul *Pohled* slouží zejména k vytváření dynamických pohledů, které se pak využívají při snaze optimalizovat vyhledávací operace v prostorových tabulkách pomocí omezení okruhu vyhledávání na menší ohraničenou oblast. Objekt typu pohled si pamatuje:

- k jaké databázi se váže,
- své jméno,
- zdrojovou tabulku
- a meze pro souřadnice *X* a *Y*.

Tyto informace je nutné zadat do konstruktoru. Pro *Pohled* existují dva konstruktory. Pokud není zadán parametr *meze*, budou meze považovány za nulové a data v pohledu budou totožná s celým obsahem zdrojové tabulky.

Dále následuje ukázkový předpis dotazu dynamicky vytvořeného v databázi.

```
CREATE OR REPLACE VIEW POHLED_DEMO
AS
  SELECT * FROM RAILWAY_NODE$ s
  WHERE s.geometry.sdo_point.x >= 15.675891402130938
  AND s.geometry.sdo_point.x   <= 15.843834311250458
  AND s.geometry.sdo_point.y   >= 50.001327951073385
  AND s.geometry.sdo_point.y   <= 50.055272048926604;
```

2.9.4 Třída FunkceSoused

Tato třída slouží k dynamickému generování předpisu funkce obalující dotaz na přesnou polohu vlaku a k jejímu vytvoření v databázi. Při vytváření instance této třídy je potřeba zadat referenci na vlak, pro který má být generovaná funkce určena, a zbytek informací si metody dohledají přímo z této předané reference na instanci vlaku.

Funkce FCN_VLAK1 je funkce vytvořená vlakem s identifikačním číslem 1. Vstupními parametry jsou souřadnice *X* a *Y*. Funkce vrací řetězec typu *CHAR*, který pomocí středníku od sebe odděluje

- id uzlu,
- přesnou souřadnici *X* a *Y*,
- vzdálenost hledaného bodu od nejbližšího souseda.

Pro přístup k jednotlivým položkám je nutné výsledek v aplikaci „rozparsovat" metodou *split()*. Pomocí komentáře a znaménka plus je vnitřní dotaz na nejbližšího soused pojmenován, pro lepší hledání v tabulce *v$sql*. Bez umístění znaku + hned za otevřením komentáře /* k pojmenování příkazu *SELECT* nedojde. Na vině je nejspíše optimalizace kódu funkce při kompilaci, která se tímto postupem obejde.

2.9.5 Třída DotazSoused

Třída *DotazSoused* je stěžejní třídou starající se o výpočet přesné pozice vlaku (hledání nejbližšího souseda od polohy vlaku v databázi železničních bodů) a analýzu efektivity různých typů dotazů včetně optimalizace pomocí dynamických pohledů. Poskytuje dynamické vytvoření dotazů s ohledem na vybranou techniku dotazu a vyhledání přesné polohy objektu na železnici.

Nabízené typy dotazů jsou:

- obyčejný dotaz (celý vytvořený pouze skládáním řetězců),
- jeden dotaz s vázanými proměnnými *PreparedStatement* (stále stejná instance),
- funkce obsahující dotaz na polohu uložená přímo v databázi Oracle.

Třída dále obsahuje metodu, která umí zjistit dobu trvání jednotlivých dotazů, čímž pomáhá porovnat různé přístupy mezi sebou. Instance třídy *DotazSoused* potřebuje při vytváření v parametrech předat pouze referenci na:

- *Vlak*, který bude chtít zjišťovat svou přesnou pozici,
- a typ dotazu, který se má při tomto výpočtu použít.

Pokud není zadán typ dotazu, použije se jednoparametrický konstruktor, který vytvoří instanci používající techniku s připraveným dotazem v Javě. Veřejná metoda *najdiNejblizSouseda()* najde přesnou pozici na trati, pokud jí jsou předány nepřesné souřadnice *x* a *y*.

2.9.6 Třída NactiProcesyCSV

Modul *NactiProcesyCSV* implementuje obecné rozhraní *INactiProcesy*. Implementovaná metoda *NactiVlakProcesy(Vlak vlak)* umí vybrat potřebná data pro průjezd vlaku ze souboru CSV a vytvořit tak kolekci procesů *ZmenaPoziceVlaku*, kterou poté naimportuje do kalendáře simulačního jádra.

Konstruktor pro objekty třídy *NactiProcesyCSV* potřebuje předat řetězec s cestou k zdrojovému souboru ve formátu CSV a poté postupně čísla sloupců, v kterých se nachází údaj o zeměpisné šířce, zeměpisné délce a čas, ve kterém se vlak na této pozici nachází.

2.9.7 Třída Proces

Jedná se o abstraktní třídu, která implementuje metody z rozhraní *IProces*. Nelze od ní vytvořit instanci. Obsahuje metody (společné pro všechny potomky) pro přístup k privátním atributům *doba trvání procesu* a k *časovému razítku*, které reprezentuje čas, kdy je proces ukončen. Dále předepisuje důležitou metodu *proved()*, kterou musí implementovat každý odvozený potomek. Tuto metodu pak využívá simulační jádro pro provedení akcí, které daný proces obsahuje.

2.9.8 Třída SimulacniJadro

Simulační jádro skrývá atributy uchovávající:

- informace o aktuálním čase simulace,
- interval definující čekání mezi jednotlivými kroky simulace
- a kalendář událostí.

Do tohoto kalendáře, lze přidat jakýkoli proces implementující abstraktní třídu *Proces* a její stěžejní metodu *proved()*. Třída používá návrhový vzor *Jedináček* a implementuje rozhraní *Runnable* s předepsanou metodou *run()*, která obsahuje kód reprezentující algoritmus diskrétní simulace. Tuto metodu využívá vlákno *simulator* a po startu tohoto vlákna je zahájena samotná diskrétní simulace.

Simulace v metodě *run()* postupně odebírá procesy s nejnižším časovým razítkem z kalendáře událostí, provádí akce a změny definované aktuálním procesem (předepsané v metodě *proved()* z aktuálního procesu) a pokud se mění čas simulace, vlákno se uspí metodou *sleep()* na počet milisekund, který je definován pomocí hodnoty atributu *interval*. Další metody umožňují simulaci pozastavit, obnovit a úplně vypnout a využívají k tomu proměnných typu *boolean* a samozřejmě metodu pro překreslení plátna, kde se předanému *JPanelu* zavolá jeho metoda *repaint()*.

2.9.9 Třída ZmenaPoziceVlaku

Tato třída implementuje abstraktní třídu proces a její abstraktní metodu *proved()*. Účelem instancí této třídy je:

- převzít novou nepřesnou polohu vlaku,
- vypočítat jeho přesné souřadnice
- a přemístit ho na jeho nově vypočtenou pozici.

Kromě nových souřadnic zvládne zjistit, jakou chybu měly původní souřadnice a jak dlouho výpočet přesných souřadnic trval. Instance zároveň obsluhuje optimalizační mechanismus týkající se dynamických pohledů. Když vlak opustí meze původního pohledu, postará se o inteligentní přepočtení mezí nových.

2.9.10 Třída GPS

Původně sloužila pouze k uchování dvojice hodnot reprezentující souřadnice, ale během vývoje aplikace se ukázalo, že je výhodné připsat do ní statické metody pro generování náhodných souřadnic GPS, výpočty vzdálenosti mezi body a pro hledání posunutých bodů.

2.9.11 Třída Hranice (Meze)

Objekty typu Hranice reprezentují dvojrozměrné rovnoběžné meze na osách X a Y, které vlastně definují obdélníkovou oblast v dvojrozměrném prostoru. Instance této třídy lze vytvořit buď přímým předáním horizontálních (levá, pravá) a vertikálních (dolní, horní) mezí, nebo pomocí čtveřice bodů. Pokud se nejedná o body reprezentující vrcholy obdélníka, konstruktor vytvoří obalový obdélník z krajních hodnot všech souřadnic X a Y.

Objekty této třídy sice uchovávají oblast definovanou jejími hranicemi, ale v současné době obsahují i několik užitečných metod pro

- výpočet délky stran této obdélníkové oblasti (rozdíl *X* a *Y*),
- metodu, která zkoumá, zda zadaný GPS bod v této oblasti leží,

- a přístupové metody pro členské atributy *minX*, *maxX*, *minY*, *maxY*.

Přidružené statické metody *getHranice()* a *getHraniceStred()* slouží k výpočtu a vygenerování obdélníkových hranic okolo konkrétně zadaného bodu. Tyto metody jsou poté používány při generování a přepočítávání dynamických pohledů pro jedoucí vlaky a zároveň i pro účely testování aplikace, porovnávání dotazových technik a ověřování stanovených hypotéz z teoretické části BP.

2.9.12 Třída HlavniOkno

Třída HlavniOkno je hlavním oknem celé aplikace a poskytuje ovládací prvky pro volání potřebných metod a vykonávání akcí důležitých pro běh programu. Dále obsahuje *JPanel,* který slouží jako plátno a realizuje vykreslování trati a animaci pohybujících se objektů. V konstruktoru dojde k inicializaci potřebných komponent, vytvoření demonstračního vlaku a jeho ukázkového pohledu.

Aplikace se při startu automaticky připojí na databázi Oracle na serveru *localhost* (127.0.0.1) k databázi *orcl2* naslouchající na portu 1522 pod správcovským účtem a heslem. Port 1522 není oproti portu 1521 standardní, ale jelikož bylo nutné nainstalovat databázi podruhé, port se u druhého databázového stroje nastavil automaticky na číslo o hodnotu vyšší.

Načtení samotné trati funguje tak, že se nejdříve načtou všechny body z příslušné databázové tabulky, které se poté jako kolekce předají plátnu a vykreslí se pomocí zavolání metody *repaint()*. Během provádění tohoto postupu se zviditelní animovaný *progress bar* ve formátu GIF a po načtení celé trati zase zmizí. Celá operace probíhá ve vlastním vlákně, aby šlo i nadále pracovat s aplikací a aby se zobrazil již zmiňovaný *progress bar* signalizující načítání tratě.

Vizualizaci trati (a ostatních objektů) na plátno obstarává vložený *JPanel*. Ten si pomocí kolekce *vykreslovaneObjekty* udržuje v paměti, jaké objekty má vykreslovat. Vykreslení těchto objektů probíhá v překryté metodě *paint()*. Pomocí iterátoru se postupně projdou všechny body a vykreslí se. Jelikož mají jednotlivé objekty souřadnice z reálného světa a plátno má omezené rozměry, je nutné přepočítat souřadnice objektů do jiného měřítka. Dále je nutné počítat s tím, že počátek počítačových souřadnic leží v levém horním rohu panelu.

Pokud jsou přepočtené souřadnice vykreslovaného bodu mimo plátno, nedojde k vykreslení tohoto objektu. Tím se zajistí zrychlení celého vykreslení. Pokud je vykreslovaným objektem vlak, vykreslí se i obdélník zobrazující hranice jeho dynamického pohledu a nakonec se do vrchní vrstvy vykreslí panel s časem simulace.

Samotné plátno poté dokáže reagovat na akce myši. Při tažení myši se stlačeným tlačítkem dochází k posouvání mapy a při pohybu kolečkem myši se mění měřítko vykreslení mapy, což působí jako efekt zoom.

Za zmínku stojí ještě implementace testovací části aplikace. Pomocí komponent dojde k načtení vstupních dat, vybere se typ dotazu, pohled a vytvoří se pomocný vlak s tímto pohledem. Vlak se pak použije jako parametr pro vytvoření instance třídy *DotazSoused,* která bude provádět testování.

Testování probíhá v samostatném vlákně, aby šla aplikace během testů nadále ovládat a docházelo k průběžnému výpisu dat. Pro generování náhodného bodu *GPS* se použije metoda objektu *DotazSoused*, která zavolá metodu pro generování bodu *GPS* z třídy *Hranice*. Tento výsledek se předá metodě s dotazem na vyhledání nejbližšího souseda z bodů skutečné trati a provede se dotaz a výpočet.

Poté se spočítá čas tohoto dotazu společně se vzdáleností mezi generovaným bodem a jeho nejbližším sousedem na trati. Tento čas se vypíše do komponenty *Textarea*, uloží se do pomocné databázové tabulky *CASY_SELECTU* (časy jednotlivých dotazů) a zároveň se společně se zmiňovanou vzdáleností připíše na konec souboru *CASY_POHLED_TYP_DOTAZU.csv* pro pozdější zpracování. Po dokončení testů dojde k výpočtu statistik týkajících se souboru časů a dojde znovu k výpisu výsledků na komponentu *Textarea*.

Aplikace je kompletní a umožňuje otestovat jednotlivé optimalizační techniky a stanovené hypotézy z teoretické části.

3 Testování

Testování navržené optimalizace vybraných vyhledávacích operací pomocí demonstrační aplikace by mělo prokázat, zda byly uvažované předpoklady správné. Testovány byly jednotlivé techniky vyhledávací operací nad tabulkou čítající přibližně stotisíc, respektive třináct tisíc záznamů, a nad dynamickými pohledy, které bázi prohledávaných dat výrazně redukovaly.

Jako soubor testovaných dat bylo vygenerováno 1000 GPS bodů z předem dané oblasti. Na těchto bodech byly prováděny dotazy na nejbližšího souseda z bodů příslušné železniční sítě. Časy těchto dotazů byly v databázi měřeny a ukládány pro pozdější statistické zpracování. Pro druhotné porovnání slouží změřené celkové časy jednotlivých testů v rámci aplikace, kde se projeví i režie ve formě obsluhy databáze, spojení s databází a zpracování výsledků v jazyce Java.

Pro změření časů jednotlivých dotazů bylo využito záznamů v systémové tabulce databázového stroje Oracle *v$sql* a hodnot ze sloupce *ELAPSED_TIME*. Tyto hodnoty byly převedeny z mikrosekund na milisekundy a následně byly zpracovány aplikací. V případě dotazů s vázanými proměnnými bylo nutné upravit získávání časů. Díky optimalizaci se dotaz připraví pouze jednou a poté jsou mu jen předávány vázané proměnné. Takovýto dotaz je v tabulce *v$sql* uložen jako jeden záznam, kterému se při každém dalším volání zvýší počet provedení a čas provádění se navýší o dobu trvání posledního provedeného dotazu. Pro výpočet času posledního dotazu tedy bylo nutné si vždy pamatovat původní sumu časů a od nově aktualizované sumy ji odečíst.

3.1 Počet záznamů v tabulce

První část testů zkoumá, jak počet záznamů v dané tabulce ovlivňuje rychlost vyhledávání v této tabulce. Testování probíhalo na tabulkách *RAILWAY_NODE$* (13 474 záznamů) a *UZLY_CR* (95 679 záznamů). Dotazy byly prováděny pomocí obyčejného dotazu a objektu typu *Statement*.

Tabulka 1 – Porovnání časů dotazů mezi velkou a malou tabulkou v sekundách

Rozsah zdrojových dat	UZLY_CR	RAILWAY_NODE$
Průměr	0,152424742	0,159892272
Min	0,060082	0,063586
Max	1,263402	1,087492
Medián	0,154051	0,1586725
Modus	0,1047	0,129895
Odchylka	0,064793217	0,055391611
Celkový čas	152,424742	159,892272
Celkový čas v Javě	244,96869837	265,559209837

Je podivuhodné, že hledání v tabulce s menším počtem záznamů je dokonce o trochu pomalejší než hledání v tabulce větší. Rozdíl však není tak velký a tak lze konstatovat, že při použití objektu *Statement* na těchto množstvích dat nemá menší velikost tabulky očekávaný vliv na rychlost hledání.

3.2 Techniky dotazů

V této části byly otestovány techniky, pomocí nichž lze sestavit dotaz na nalezení nejbližšího souseda v rámci výpočtu přesné pozice kolejového vozidla. Jedná se o volání obyčejného dotazu pomocí skládání SQL z řetězců objektem *Statement* v jazyce Java, volání jednou připraveného dotazu pomocí objektu *PreparedStatement* s vázanými proměnnými a volání dotazu v databázové funkci uložené přímo v databázi Oracle.

Následující tabulka srovnává jednotlivé techniky dotazu na celé tabulce *UZLY_CR*.

Tabulka 2 – Porovnání časů různých technik dotazů v sekundách

Technika dotazu	Statement	PreparedStatement	Oracle funkce
Průměr	0,120897181	0,009185022	0,012439586
Min	0,059907	0,003553000	0,004768000
Max	3,28629	1,640714	1,607601
Medián	0,0994355	0,006620500	0,009731500
Modus	0,066694	0,004035000	0,010057
Odchylka	0,115109118	0,052007543	0,051450222
Celkový čas	120,897181	9,185022	12,439586
Celkový čas v Javě	195,717063164	53,824611413	60,935994212

Z výsledků lze vyčíst, že nejrychlejší technikou je použití objektu *PreparedStatement*. Následuje použití funkce uložené přímo v databázi Oracle a daleko za nimi zaostává obyčejný dotaz pomocí metod třídy Statement. Pomocí použití vhodné techniky dotazu lze snížit průměrný čas až 13 krát.

3.3 Optimalizace pomocí dynamických pohledů

Pokud omezíme zdrojovou oblast dat na okolí vyhledávaného bodu, mělo by dojít ke zrychlení vyhledávání. První pokus byl proveden s technikou PreparedStatement na pohledech o velikosti 10 na 10 km, 40 na 20 km a v poslední řadě čtvercovým pohledem o straně 50 km odvozených od tabulky *UZLY_CR*.

Tabulka 3 – Časy dotazů PreparedStatement nad pohledy různých velikostí v sekundách

Velikost pohledu	10x10	40x20	50x50
Průměr	0,007647564	0,007537129	0,007978
Min	0,002743	0,002815	0,003319
Max	1,454968	1,157581	1,261327
Medián	0,0049415	0,005347	0,006121
Modus	0,004397	0,003494	0,004287
Odchylka	0,046213794	0,03691007	0,039899127
Celkový čas	7,647564	7,537129	7,977629
Celkový čas v Javě	39,759844446	39,70151928	40,368371538

Srovnání různých velikostí pohledů napovídá, že všechny tyto pohledy optimalizují už tak rychlý připravený dotaz na přesnou pozici. Nejrychlejší je pohled o velikosti 40x20 km. Pomocí této optimalizace byl snížen čas z původního času 0,0092 sekund o dalších pár tisícin sekundy. Minimální čas se již pohybuje pod hodnotou 0,003 sekundy. Do konečných výsledků by ještě mohla zasáhnout podobná optimalizace pro funkci uloženou v databázi Oracle.

Tabulka 4 – Časy dotazů v Oracle funkci nad různě velkými pohledy v sekundách

Velikost pohledu	10x10	40x20	50x50
Průměr	0,007312738	0,007474627	0,007914767
Min	0,003082	0,003242	0,003108
Max	1,370451	1,354342	1,568563
Medián	0,0052415	0,005647	0,005894
Modus	0,003895	0,005491	0,004772
Odchylka	0,043302936	0,0427506	0,049460025
Celkový čas	7,312738	7,474627	7,914767
Celkový čas v Javě	51,351476774	54,835111842	59,058458159

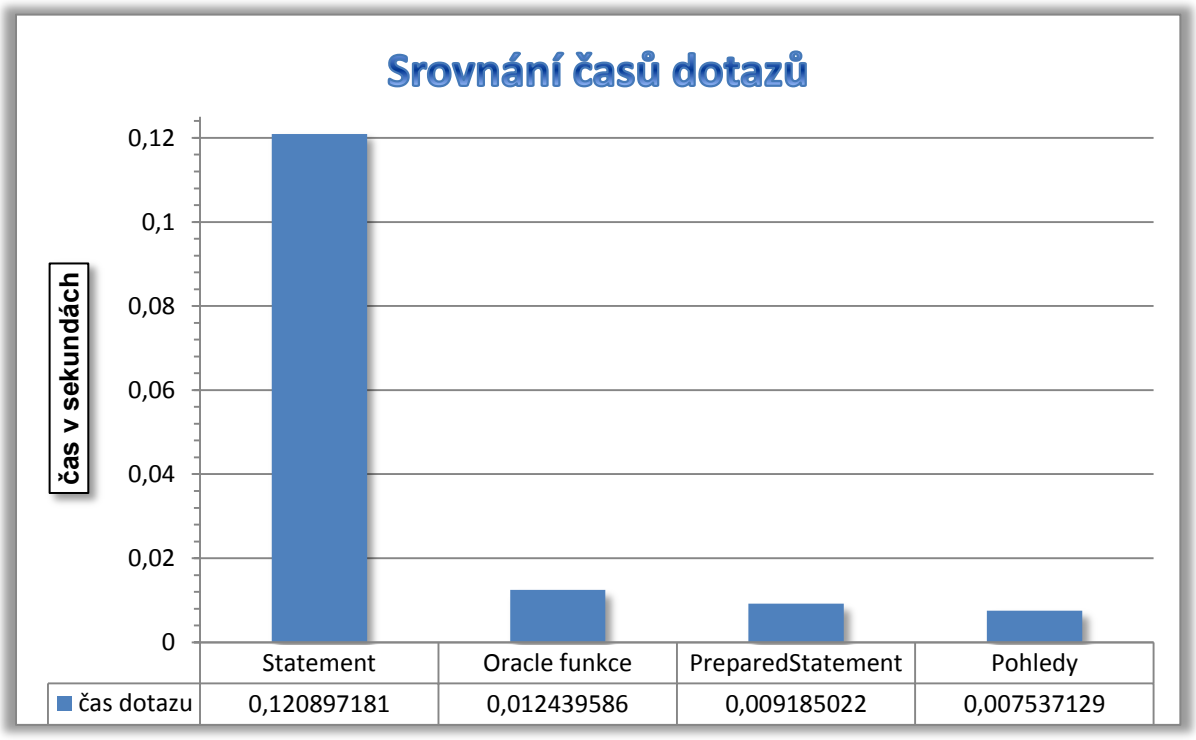
Zde je nejrychlejší dotaz z dynamického pohledu 10x10 km. Zrychlení původního volání dotazu uvnitř funkce a znatelnější než zrychlení u metody využívající objekt *PreparedStatement*. Dokonce došlo k tomu, že pomocí pohledů 10x10 a 40x20 se dotaz uvnitř funkce zrychlil natolik, že překonal čas objektu *PreparedStatement* na celé tabulce a srovnal se i s jeho pohledovou optimalizací 40x20 (vlastně může být i rychlejší než optimalizovaný *PreparedStatement*).

Tabulka 5 – Časy různých technik dotazů nad pohledem 40x20 km v sekundách

Metoda – pohled	Oracle funkce – 40x20	PreparedStatement – 40x20
Průměr	0,007475	0,007537
Min	0,003242	0,002815
Max	1,354342	1,157581
Medián	0,005647	0,005347
Modus	0,005491	0,003494
Odchylka	0,0427506	0,036910
Celkový čas	7,4746270	7,537129
Celkový čas v Javě	54,835112	39,70152

Celkové časy v Javě při použití funkce v databázi Oracle jsou však oproti připravenému dotazu *PreparedStatement* znatelně vyšší. Přestože je vnitřní dotaz velmi rychlý, režie vynaložená na volání a převzetí výsledků funkce je o dost pomalejší než u optimalizovaného dotazu *PreparedStatement*. Proto i přes zlepšení času není vhodné upřednostnit funkci v databázi Oracle před připraveným dotazem.

Následující graf ukazuje postupné zrychlování dotazů pomocí dílčích optimalizací.



Obrázek 3 – Graf srovnání jednotlivých optimalizací

3.3.1 Shrnutí testů

Testy prokázaly, že předpoklady stanovené v teoretické části byly správné. Časový rozdíl mezi vyhledáváním v celé tabulce o 13 000 řádcích a v celé tabulce o 95 000 řádcích není

znatelný. Dále bylo jasně prokázáno, že je důležité používat dotazy s vázanými proměnnými, protože časová úspora je skutečně velká. Řešení pomocí třídy *PreparedStatement* se jeví jako lepší nejen díky vyšší rychlosti dotazů, ale i díky nižší režii jak z hlediska databázového stroje, tak z hlediska nároků na programátora. Optimalizace pomocí dynamických pohledů přinesla další zrychlení. Je však nutné podotknout, že toto zrychlení není tak markantní a vyplatí se zejména v případě většího množství aktivních vlaků (řekněme více než 500) s malou periodou ověřování jejich skutečné polohy (do 10 sekund). V tomto případě se doporučuje kombinace techniky připraveného dotazu *PreparedStatement* a dynamického pohledu 40x20 km, případně 50x50 km. Takto dojde ke zrychlení celé aplikace o celé sekundy. Vytvoření dynamického pohled však trvá většinou přibližně 0,1-0,3 sekundy, takže pokud bude v systému malé množství vlaků, které se budou dotazovat svou pozici v časovém intervalu například 30 sekund, nepřinese tato optimalizace v podstatě žádné výhody.

Závěr

Všechny cíle bakalářské práce byly splněny v plném rozsahu. V teoretické části byla popsána databáze Oracle s nadstavbou Oracle Spatial. Dalším tématem bylo spojení databáze Oracle s programovacím jazykem Java pomocí ovladačů JDBC a využití obecného rozhraní a balíku tříd pro práci s databázemi. Veškeré příklady jsou funkční a slouží k základnímu seznámení se s rozhraním pro práci databází v jazyce Java.

Z hlediska Oracle Spatial byla popsána práce zejména s tabulkami obsahující dvojrozměrné body a případným zájemcům o technologii poskytuje základní informace o tvorbě tabulek s prostorovými daty. Dále byl rozebrán pouze nezbytný zlomek funkcionality celého doplňku se zaměřením na funkci pro hledání nejbližšího souseda *SDO_NN*, která je v projektu využívána pro výpočet přesné pozice kolejového vozidla v železniční síti. Tuto látku bylo nutné samostatně nastudovat.

V praktické části bakalářské práce byla navržena a implementována demonstrační aplikace pro ověření teoretických předpokladů. Došlo k teoretickému zopakování a praktickému procvičení několika programovacích technik, návrhových vzorů a dalších znalostí získaných během vysokoškolského studia. Výsledná aplikace umí načíst trať z databáze, dynamicky přidávat vlaky včetně jejich pohledů, načítat průjezdy kolejových vozidel z externích souborů a provést vizualizovanou simulaci těchto průjezdů v rámci železniční sítě. Aplikace též umožňuje změřit vliv různých optimalizačních postupů na záložce *Testy* a provést analýzu výsledků.

Nad rámec zadání bylo implementováno jednoduché jádro diskrétní simulace a animace pro vizualizaci pohybujících se objektů po železniční trati. Při implementaci aplikace bylo využito pokročilých programovacích technik včetně práce se soubory, grafikou, databází, kolekcemi a vlákny.

Nakonec byly provedeny testy jednotlivých technik dotazů nad tabulkami různých velikostí. Ukázalo se, že je důležité používat dotazy s vázanými proměnnými z důvodu velké časové úspory. Druhá fáze testů se zabývala testováním navržené optimalizace vybraných vyhledávacích operací z hlediska redukce báze prohledávaných dat pomocí dynamicky vytvářených pohledů. Tato část testů potvrdila většinu teoretických předpokladů a dokázala, že navržená optimalizace vyhledávacích operací pomocí dynamických pohledů má své opodstatnění. Ukázalo se však, že nasazení této optimalizace není úplně samozřejmé a vyplatí se pouze v systémech, kde je velká frekvence dotazování se na přesnou polohu kolejových vozidel pohybujících se v rámci železniční sítě. Závěrem tedy nechybí doporučená technika dotazu a velikost případného dynamického pohledu.

Dalším rozšířením aplikace by mohla být implementace modelu železniční sítě pomocí grafové reprezentace a doladění některých funkcionalit jako je automatické odstraňování dynamických pohledů či funkcí, které v současné chvíli zůstávají v databázi i v době, kdy už nejsou zapotřebí.

Literatura

1. **KAVIČKA, Antonín.** *Simulace.* Pardubice : Elektronické sylaby přednášek k předmětu Modelování a simulace, 2010.

2. **KISZKA, Bogdan.** *1001 tipů a triků pro jazyk Java.* Brno : Computer Press, 2009. 978-80-251-2467-3.

3. **PECINOVSKÝ, Rudolf.** *OOP: naučte se myslet a programovat objektově.* Brno : Computer Press, 2010. 978-80-251-2126-9.

4. **KOTHURI, Ravi, GODFRIND, Albert a BEINAT, Euro.** *Pro Oracle Spatial for Oracle Database 11g.* New York, NY : Distributed to the book trade worldwideby Springer-Verlag New York, 2007. 978-1-59059-899-3.

5. **MURRAY, Chuck.** Oracle® Spatial : User's Guide and Reference 10g Release 2 (10.2). [Online] 2005. [Citace: 18. 4 2013.] http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14256.pdf.

6. **MOLNÁR, Jan.** *Porovnání intervalového vyhledávání nad vhodnou databází a vybranými datovými strukturami.* Pardubice : Univerzita Pardubice, 2012.

7. **ŘEZANINA, Emil.** *Návrh modelu železniční sítě s využitím technologie Oracle Spatial Topology and Network Data Models.* Pardubice : Univerzita Pardubice, 2012.

8. **ORACLE.** Oracle Database Documentation Library. [Online] 2013. [Citace: 12. 3 2013.] http://www.oracle.com/pls/db112/portal.all_books.

Příloha A – Uživatelská příručka k demonstrační aplikaci

Tato příloha popisuje základní ovládací prvky demonstrační aplikace. Ta umí vykreslit mapu z bodů uložených v databázi Oracle Database 11g, vytvořit vlak, načíst pro něj průjezd a spustit provádění načtených průjezdů vlaků pomocí jádra diskrétní simulace. Celý průjezd je vizualizován krokovou animací. Dále je možné provádět testy dotazů na přesnou pozici vlaku nad různými tabulkami, pohledy a také pomocí různých technik.

Pro spuštění demonstrační aplikace je nutné mít na počítači nainstalovánu Javu nejméně ve verzi 6. Potřebné knihovny by měly být k programu přiloženy.

Uživatelské rozhraní

Uživatelské rozhraní je rozděleno na dvě záložky, které obsluhují základní funkcionalitu aplikace. První částí je záložka *Simulace* a vedle ní se nachází záložka *Testy*.

Záložka Simulace

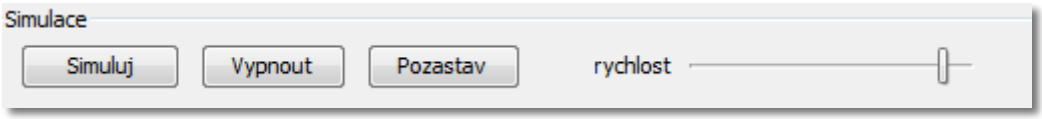
Stěžejní částí simulační části je plátno, na které se vykresluje mapa trati a animace pohybujících se vlaků. Po výběru trati stačí kliknout na tlačítko *Načti trať* a daná trať se načte z příslušné databázové tabulky. Na výběr jsou tabulky *RAILWAY_NODE$* (Královéhradecký a Pardubický kraj) a *UZLY_CR* (celá ČR). Během načítání trati se zobrazuje animovaný *progress bar*. Když je trať načtena, vykreslí se na plátno (viz Obrázek 5).

Pro demonstrační účely je v programu již vytvořen jeden ukázkový vlak. Dále je možné vytvořit vlak nový a to tím, že mu budou zadány rozměry jeho obdélníkového dynamického pohledu (*a* a *b*). Po kliknutí na tlačítko *Přidej vlak* se zobrazí dialogové okno s výběrem barvy, která bude použita pro vykreslení vlaku a jeho pohledu.

Poté je nutné pro vybraný vlak načíst jeho průjezd ze souboru CSV. Po kliknutí na tlačítko *Načti průjezd* lze v dalším dialogovém oknu vybrat zdrojový soubor s průjezdem a potvrzením dojde vytvoření příslušných událostí v kalendáři událostí simulačního jádra.

Se samotným plátnem lze provádět posuny všemi čtyřmi směry pomocí tlačítek se šipkami nebo tahem myši. Tlačítky se znaky plus a minus lze měnit měřítko a tím provádět *zoom*. Stejnou funkci má pohyb kolečkem myši nahoru a dolů. Při detailním přiblížení je vidět, že trať je vykreslená pomocí jednotlivých bodů.

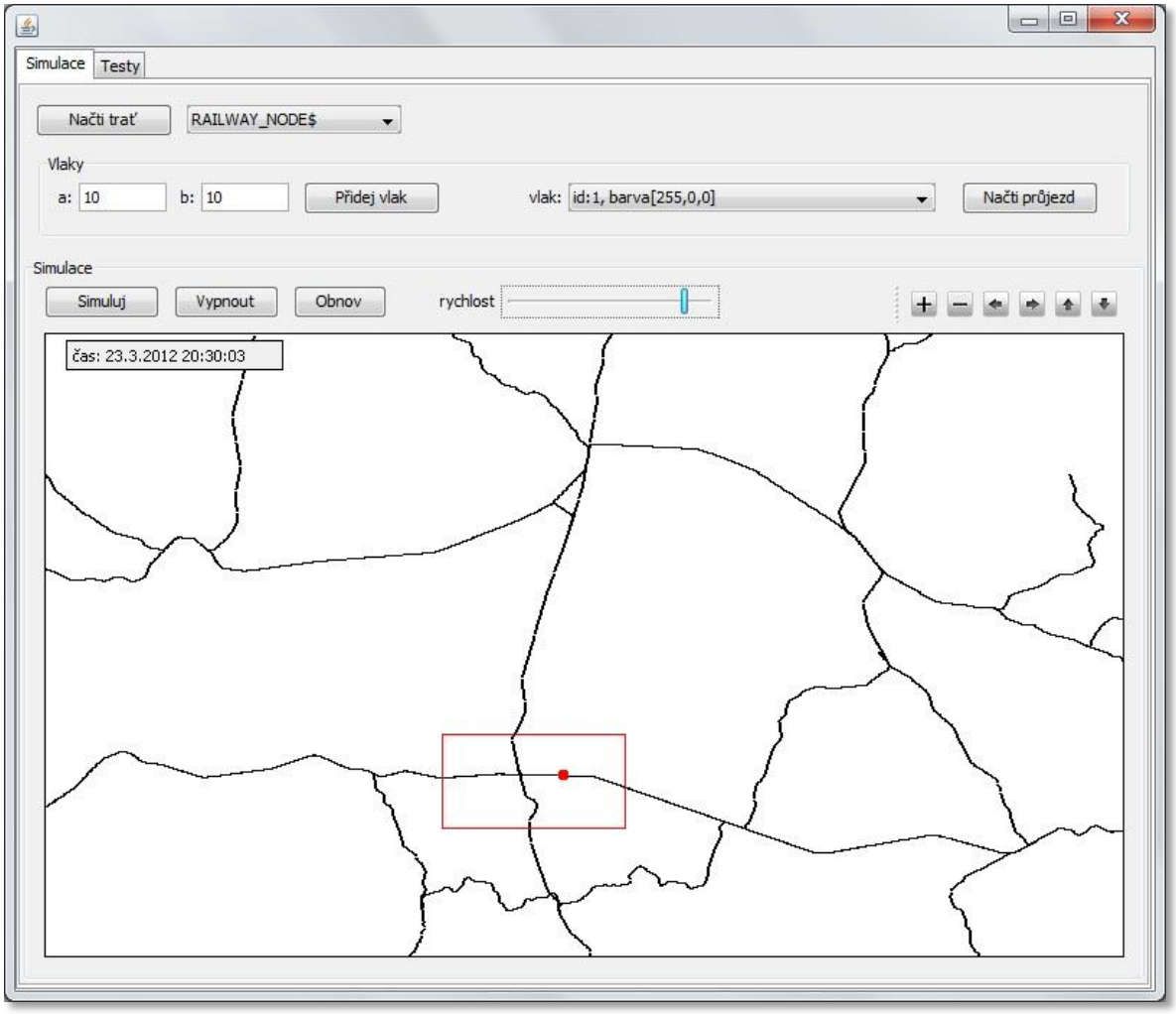
Nejdůležitějším prvkem je ovládání simulace pohybu vlaků a detekce jejich polohy. Tlačítkem *Simuluj* dojde ke spuštění simulačního běhu, začne se skokově měnit simulační čas, který je zobrazován v levém horním rohu vykreslovacího plátna. Pokud se v aktuálním čase vlak pohybuje, dojde ke změně jeho polohy.



Obrázek 4 – Náhled ovládacího panelu pro simulaci

Simulaci lze pozastavit a znovu rozběhnout nebo úplně zrušit tlačítkem *Vypnout*. Pomocí posuvníku může uživatel měnit simulační krok a tím i rychlost simulace od velmi pomalého k poměrně rychlému průběhu (viz Obrázek 4). Poté je nutné načíst veškeré průjezdy znovu, vlaky zůstanou v paměti uloženy.

Během průjezdu je možné sledovat, jak vlak, který opustí svůj dosavadní dynamický pohled, inteligentně přepočítá nové meze a vytvoří pohled nový.

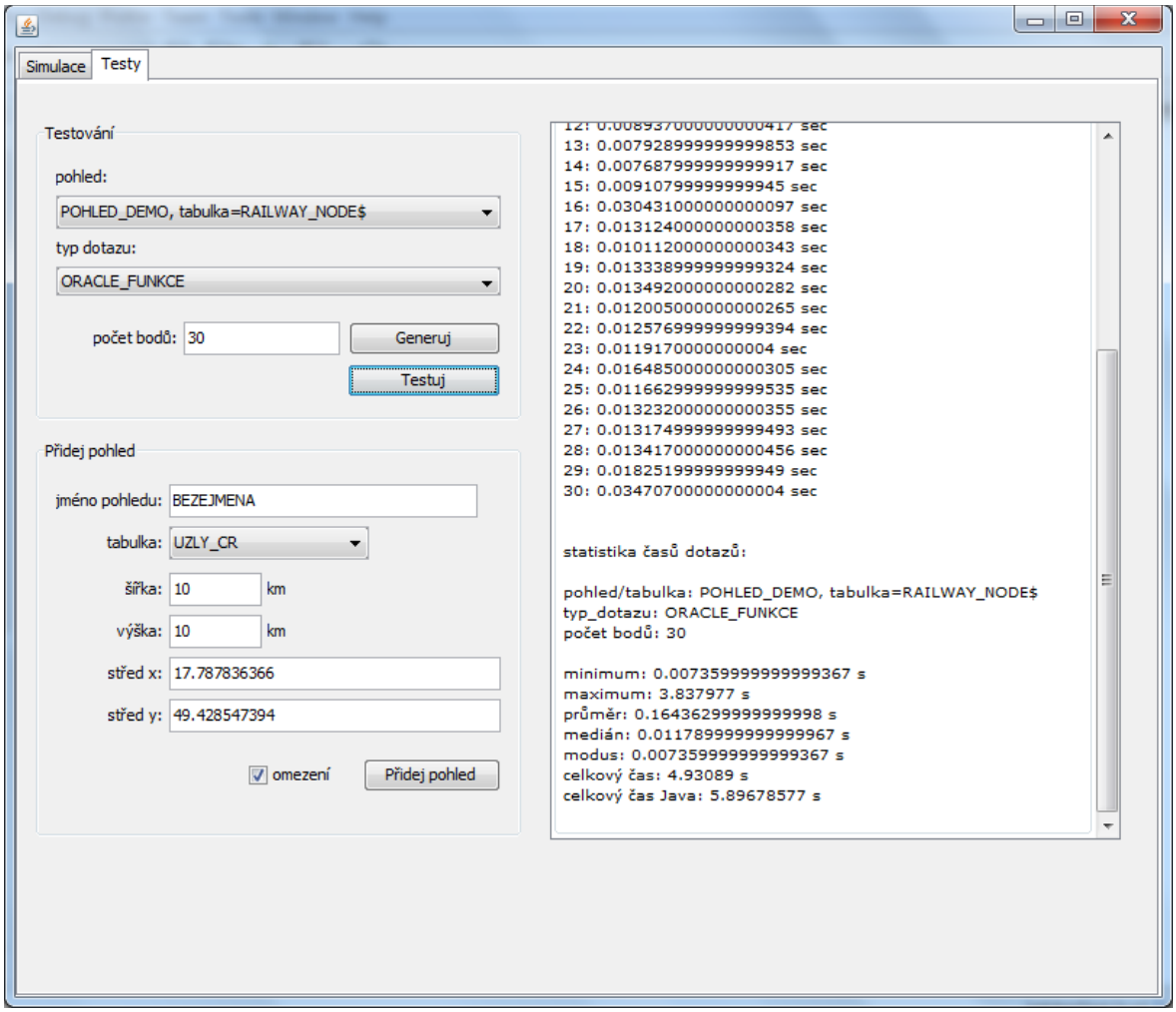


Obrázek 5 – Náhled rozhraní simulační části aplikace

Záložka Testy

V části *Testy* je možné vytvořit dynamický pohled, který chceme otestovat. Je nutné zadat mu jméno, zdrojovou tabulku s prostorovými daty a volitelně šířku, výšku a středové souřadnice jeho omezujícího pohledu. Odškrtnutím volby *omezení* se vybere vytvoření pohledu v rozsahu celé tabulky.

Sekce *Testování* slouží k ovládání testů. Je možné si vybrat zdrojovou tabulku či pohled a typ dotazu s příslušnou technikou. Dále je nutné nastavit počet náhodných bodů a vygenerovat je tlačítkem *Generuj*. Tyto body se vygenerují z oblasti definované vybranou tabulkou/pohledem. Pokud je další testování spuštěno bez generování nových bodů, probíhají výpočty s původními body. Pokud je potřebné použít pokaždé jinou bázi náhodných dat, je nutné před každým testováním provést nové generování bodů manuálně kliknutím na tlačítko *Generuj* (viz Obrázek 6).



Obrázek 6 – Náhled rozhraní testovací části aplikace

Tlačítkem *Testuj* se spustí test, který pro každý vygenerovaný bod provede nalezení nejbližšího souseda na trati a změří trvání tohoto dotazu v sekundách. Tento čas poté

vypíše do sekce *Výpis*. Když je algoritmus proveden na všech bodech, provede se výpočet závěrečných statistik a i ty se vypíší na konec *Výpisu*. Časy jednotlivých dotazů jsou zároveň ukládány do příslušného souboru CSV pojmenovaného podle použité tabulky/pohledu a použité techniky dotazu pro další zpracování.