VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

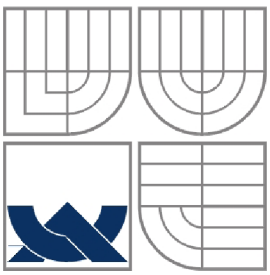
INDEXOVÁNÍ DAT POHYBUJÍCÍCH SE OBJEKTŮ

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

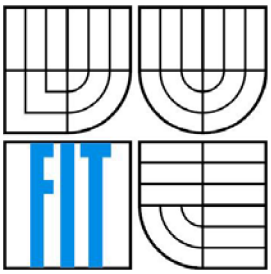
AUTOR PRÁCE
AUTHOR

Bc. MARTINA KŘÍŽOVÁ

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INDEXOVÁNÍ DAT POHYBUJÍCÍCH SE OBJEKTŮ

MOVING OBJECTS INDEXING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. MARTINA KŘÍŽOVÁ
AUTHOR

VEDOUCÍ PRÁCE — Doc. Ing. JAROSLAV ZENDULKA, CSc.
SUPERVISOR

BRNO 2010

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav informačních systémů Akademický rok 2009/2010

Zadání diplomové práce

Řešitel: **Křížová Martina, Bc.**
Obor: Informační systémy
Téma: **Indexování dat pohybujících se objektů**
Moving Objects Indexing
Kategorie: Databáze

Pokyny:

1. Seznamte se se způsoby indexování časoprostorových dat reprezentujících pohybující se objekty.
2. Seznamte se s podporou pro časoprostorová data v prostředí databázového serveru Oracle10g a pro možné způsoby indexace časoprostorových dat v tomto prostředí.
3. V návaznosti na výsledky diplomové práce J. Vetešníka navrhněte experimenty pro ověření vlastností různých způsobů uložení časoprostorových dat v databázi a jejich indexace.
4. Navržené experimenty realizujte.
5. Zhodnoťe dosažené výsledky z pohledu nejvhodnějšího způsobu uložení a indexace dat reprezentujících pohybující se objekty.

Literatura:

- Mallet, D., J.: Relational Database Support for Spatio-Temporal Data. Technical Report TR 04-21. University of Alberta. 2004. 67 p. Dostupné na http://www.cs.ualberta.ca/TechReports/2004/TR04-21/TR04-21.pdf.
- Dokumentace Oracle dostupná na http://www.oracle.com/pls/db102/homepage.
- Vetešník, J.: Indexování pohybujících se objektů. Diplomová práce. FIT VUT v Brně. 2008
- další dle pokynů vedoucího

Při obhajobě semestrální části diplomového projektu je požadováno:

- 1 až 3.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Zendulka Jaroslav, doc. Ing., CSc.**, UIFS FIT VUT
Datum zadání: 21. září 2009
Datum odevzdání: 26. května 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav informačních systémů
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Dušan Kolář
vedoucí ústavu

Abstrakt

Tato práce se zabývá problematikou indexování pohybujících se objektů. Popisuje existující přístupy k indexování časoprostorových dat a podporu pro jejich indexování v databázovém systém Oracle 11g. Cílem této práce je navrhnout struktury databází pro uložení časoprostorových dat nad databázovým systémem Oracle 11g a pro tyto databáze navrhnout experimenty. Dle těchto experimentů jsou zhodnoceny jednotlivé způsoby uložení časoprostorových dat z pohledu časové náročnosti dotazů a vhodnosti použití dostupných indexačních struktur a prostorových operátorů.

Abstract

This thesis deals with indexing of spatio-temporal data. It describes existing approaches to indexing data and support for indexing in Oracle Database 11g. The aim of this work is to design structures of databases for storing spatio-temporal data over Oracle Database 11g to propose experiments for these databases. Ways of spatio-temporal data storage are evaluated according to these experiments in terms of time demands of queries and appropriateness of using available indexing structure and spatial operators.

Klíčová slova

R-strom, 3D R-strom, B-strom, Quad-strom, časoprostorová data, indexační struktury, LRS, Oracle, prostorový operátor

Keywords

R-tree, 3D R-tree, B-tree, Quad-tree, spatio-temporal data, indexing structure, LRS, Oracle, spatial operator

Citace

Křížová Martina: Indexování dat pohybujících se objektů, diplomová práce, Brno, FIT VUT v Brně, 2010

Indexování dat pohybujících se objektů

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracovala samostatně pod vedením doc. Ing. Jaroslava Zendulky, CSc.
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

……
Bc. Martina Křížová
26.05.2010

Poděkování

Tímto bych chtěla poděkovat doc. Ing. Jaroslavu Zendulkovi, CSc., vedoucímu mé diplomové práce, za podnětné nápady, konzultace a odborné vedení při tvorbě této práce.

Obsah

1 Úvod 3

2 Časoprostorová data 4

2.1 Generování časoprostorových dat 4

2.2 Modelování časoprostorových dat 4

2.3 Časoprostorové dotazy 5

3 Základní indexační struktury 6

3.1 Rodina B-stromu 6

3.1.1 B-strom 6

3.1.2 B^+-strom 7

3.2 Rodina R-stromu 8

3.2.1 R-strom 8

3.2.2 R^+-strom 9

3.2.3 R^*-strom 9

3.3 Quad-strom 10

4 Indexování pohybujících se objektů 11

4.1 Indexování současného pohybu 12

4.2 Indexování současného a budoucího pohybu 12

4.2.1 Transformační metody 12

4.2.2 Parametrické prostorové přístupové metody 13

4.3 Indexování historie pohybu 14

4.3.1 Časová dimenze 14

4.3.2 Časová informace obsažena v uzlu 14

4.3.3 Překrývající se a mnohočetné struktury 14

4.3.4 Přístupové metody orientované na trajektorii 15

5 Databázový systém Oracle 16

5.1 Oracle Spatial 11g 16

5.2 Typy geometrie 16

5.3 Prostorové operátory 17

5.4 Indexování 18

5.4.1 Model dotazu 18

5.4.2 Indexy 19

5.4.3 Indexování časoprostorových dat 21

6 Příprava pro experimenty 24

6.1 Použité nástroje 24

6.2 Generování dat 24

6.3 Objekty v SQL 26

6.4 Struktura databáze 27

6.4.1 Databáze s 2D geometrií 27

6.4.2 Databáze využívající LRS 29

6.4.3 Databáze s 3D geometrií 30

6.4.4 Databáze bez geometrie 30

6.5 Postup příprav 31
7 Experimenty 33
7.1 Vlastnosti pro testování 33
7.2 Úvodní statistiky 34
7.3 Optimalizátor 36
7.4 Databáze s 2D geometrií 38
7.4.1 Porovnání operátorů SDO_RELATE, SDO_FILTER 38
7.4.2 Quad-strom: Fixní indexování 40
7.4.3 Quad-strom: Hybridní indexování 41
7.4.4 Porovnání indexačních struktur R-strom, Quad-strom 43
7.5 Databáze využívající LRS 44
7.5.1 Nalezení trajektorií v části regionu 44
7.5.2 Porovnání databází s operátorem SDO_RELATE 46
7.6 Porovnání všech struktur databází 47
7.6.1 Nalezení trajektorií v části regionu 47
7.6.2 Nalezení nejbližší trajektorie 50
8 Zhodnocení experimentů 53
Závěr 55
Literatura 56
Seznam příloh 58
Příloha A. Tabulky k úvodním statistikám 59
Příloha B. Explain plan SQL dotazu 60
Příloha C. Tabulka k experimentu 7.4.3 61
Příloha D. Procedura NN_LRS 62
Příloha E. Procedura NN_3D 63
Příloha F. Procedura NN_Full 64
Příloha G. Procedura NN_Full_G 65
Příloha H. Obsah přiloženého CD 66

1 Úvod

Během posledních let vznikají nové aplikace, které jsou zaměřené na multimédia, medicínu, bioinformatiku, počasí, astrofyziku. Takové aplikace jsou založeny na stále složitějších datech a algoritmech. Rozvíjejí se technologie, které přesně určují pozice objektů, jako např. GPS. Takové informace lze výhodně využít ke zjišťování informací o nejbližších objektech a dalších interakcí s okolními objekty, k predikování budoucího pohybu, či k analýze minulého pohybu.

Data, která jsou ukládána do databáze nebo přijímána z GPS, mají diskrétní charakter, i když pohyb objektu je spojitý. Proto se diskrétní data, která v průběhu času mění svoji pozici, musí aproximovat ve spojitou křivku. Tato křivka nám určí přibližný pohyb objektu. Samotná diskretizace dat probíhá především z důvodu snížení objemu databáze.

Ale i při diskretizaci obsahují databáze, které ukládají pohybující se objekty, obrovské množství dat. Proto v posledních letech s nárůstem nových technologií a aplikací vznikají nové přístupové metody k indexovaní těchto dat. Vytvořené indexy zvyšují výkon dotazu, tedy snižují dobu, která je nutná na vyhledání a vyhodnocení odpovídajících dat. Indexační struktury pro pohybující se objekty vznikaly především z již existujících indexačních struktur vhodných pro prostorová data jako je R-strom, Quad-strom atd.

Indexační struktury pro tato data můžeme rozdělit na tři typy dle typu dotazu. Typy dotazů mohou být historické (zaměřené na minulost pohybu), současné a budoucí, kde dochází k predikci pohybu. Každá indexační struktura je vhodná pro jiný typ dotazu.

V následujících kapitolách je popsána problematika pohybujících se objektů a možnost jejich indexace, včetně možnosti indexování takových objektů v databázovém systému Oracle. Druhá kapitola poskytuje širší úvod k pohybujícím se objektům, jejich generování a modelování. Ve 3. kapitole jsou popsány základní indexační struktury vhodné pro indexování prostorových dat. Pro indexování pohybujících se objektů je nutné tyto struktury upravit a rozšířit. Takové indexační struktury jsou popsány v kapitole 4. V 5. kapitole je popsán databázový systém Oracle z pohledu indexování dat a možnosti vytváření indexů nad pohybujícími se objekty.

Ze získaných znalostí v rámci teoretické části jsem navrhla experimenty, které zjišťují časovou složitost jednotlivých indexačních struktur dle navržených struktur databází. Experimenty jsou založeny na indexování pohybujících se objektů v minulosti. V kapitole 6. je uveden postup k realizaci experimentů a struktury jednotlivých typů databází. Samotný návrh a výsledky experimentů jsou v kapitole 7. V 8. kapitole je zhodnocení jednotlivých struktur databází a přístupů k provádění dotazů.

V této práci navazuji na diplomovou práci [13], jejímž cílem bylo testování pohybujících se objektů. Cílem mé práce je ověření experimentů nad strukturou databáze, která nevyužívá prostorový datový typ pro ukládání prostorových dat, a zhodnocení s dalšími mnou navrženými způsoby uložení dat. V této práci je převzata struktura databáze a případně upravena dle požadavků na způsob uložení dat. V rámci semestrálního projektu byly prostudovány informace k pohybujícím se objektům a jejich uložení v databázi, odpovídají tomu kapitoly 1 až 5.

2 Časoprostorová data

Časoprostorová data, neboli informace o pohybujících se objektech (jejich pozice v určitý čas), můžeme získat z různých zdrojů, např. pohybující se lidé, zvířata, letadla, počasí. Existuje několik druhů pohybů: neomezený pohyb (plavidlo na moři), omezený pohyb (lidé) a pohyb v sítích (vlaky). Časoprostorová data mohou být následně využita při sledování lodní přepravy, časoprostorového dolování dat a nalezení vzorů ve velkých databázích.

Ke sběru dat může být využito velké množství technologií, jako je např. GPS (Global Positioning System). Technologie GPS přesně určí pozici uživatele. Ale lze ji zjistit i bez GPS. Principiálně to funguje tak, že určení pozice mobilního telefonu v síti je založeno na měření vzdálenosti mezi překrývajícími se dosahy signálů. V následujících kapitolách jsem čerpala převážně z [2].

2.1 Generování časoprostorových dat

Jelikož většina reálných časoprostorových dat je soukromého či majetkového charakteru, je velice obtížné získat data pro experimenty. Proto vznikly generátory časoprostorových dat. Jedny ze zajímavých generátorů jsou "Generate SpatioTemporal Data" (GSTD), "Network-based Generator of Moving objects" a "City Simulator by IBM".

GSTD generuje data na základě statistického rozložení. Jsou podporována 2 rozložení: Gaussovské a asymetrické. Pro generování bodových dat GSTD požaduje: počáteční rozložení (definuje, kde objekty začínají), časové rozložení (kontroluje časová razítka, když pozice objektu je aktualizována) a střed rozložení (kontroluje pohyb bodu v prostoru). Pokročilé funkce podporované GSTD podporují "rámce" – překážky, objekty, kam nelze vstoupit. Nicméně i s použitím rámců se lze pohybovat kamkoliv, podél cesty, kde překážky nejsou definovány.

"Network-based Generator of Moving objects" generuje datovou množinu, kde objekty se mohou pohybovat pouze po určité síti např. po silnici. Hlavní výhodou tohoto generátoru je, že data se pohybují po určité topologii (síti).

"City Simulator by IBM" vytváří časoprostorová data, která představují pohyb lidí v 3D prostoru. Generátor je schopen vytvořit obrovské množství dat zahrnující lidi a infrastrukturu města (parky, budovy, silnice).

2.2 Modelování časoprostorových dat

Reálná a vygenerovaná data lze modelovat různými způsoby v závislosti na typu dotazu a sémantice aplikace. Data lze modelovat jako 2D nebo 3D prostorové objekty, kde časová složka bude mít další novou dimenzi. Pohyb prostorových objektů (i když je spojitý) se skládá z diskrétních vzorků.

Takto diskrétně uložená data můžeme interpretovat pomocí abstrakce trajektorie objektu. V tomto případě se předpokládá lineární trajektorie, tedy objekt se pohybuje po přímce konstantní rychlostí. Takto vymodelovaná trajektorie je užitečná především v oblastech, kde se data aktualizují zřídka a dotazy mohou spadat do intervalu mezi dvěma aktualizacemi.

Druhá varianta, jak můžeme data modelovat, je použití parametrizovaných modelů. Nezaznamenávají se jednotlivé body pohybu, jak tomu bylo v předešlé variantě, ale časoprostorová databáze ukládá parametrizovanou funkci rychlosti pro každý pohybující se objekt. Tento přístup nám

snižuje požadavky na ukládání jednotlivých bodů. Ale velkou nevýhodou je, že model předpokládá, že se objekt bude pohybovat dle naplánované cesty. Takový předpoklad může vést k velké nepřesnosti.

Další varianta je omezení pohybu objektů. Data jsou modelována jako např. body na 1D řadě silnic. Tento model se liší od výše uvedených. Body zůstávají v určité pozici (na jedné silnici) po daný časový interval (tedy máme uloženou geometrii a časový interval).

2.3 Časoprostorové dotazy

Rozlišujeme dva typy časoprostorových dotazů dle toho, co nás zajímá. Pokud nás zajímá historie pohybu objektu, budeme se zajímat o historické dotazy. Ovšem pokud se zajímáme o současný/budoucí pohyb a jeho predikci, jedná se o současné/budoucí dotazy.

Aktuální/prediktivní časoprostorové přístupové metody, které podporují dotazy na budoucí/současný pohyb, předpovídají dle aktuální rychlosti, kde se daný objekt bude nacházet. Nicméně taková odpověď nemusí být přesná (pohyb objektů se může kdykoliv změnit), je pouze orientační. Řešením tohoto problému je průběžné dotazování.

Historické časoprostorové přístupové metody podporující dotazy na historii pohybu můžeme rozdělit na dva typy:

- dotazy založené na souřadnicích: týkají se především samotných bodů a jejich okolí (nejbližší soused, rozsah,…), slouží pro získávání objektů ve stanovené oblasti v daném časovém intervalu.
- dotazy založené na trajektorii: zaměřují se na topologickou a navigační informaci. Topologický dotaz se ptá, zda trajektorie vstupuje, opouští časoprostorový rozsah. Navigační dotaz posuzuje odvozené informace jako je rychlost, procestovaná vzdálenost, atd.

3 Základní indexační struktury

Hlavním úkolem indexování – indexačních struktur je zajistit rychlý přístup k jednomu nebo více záznamům v databázi na základě vyhledávacího klíče a minimalizovat tak sekvenční prohledávání. Pro tyto účely indexační struktury poskytují specifické operace k podpoře speciálního typu dotazu: dotaz na přesnou shodu, dotaz na rozsah, a operace k podpoře sekvenčního přístupu. Pro standardní alfa-numerická data, je navrženo několik indexačních struktur. Např. struktury zahrnující index-sekvenční metodu (ISAM), stromové indexační struktury (např. B-tree, B^+-tree, B^*-tree) a indexační struktury založené na hašování [1]. V následujících podkapitolách jsou popsány základní stromové indexační struktury, které jsou implementovány v databázích, nebo díky modifikaci těchto indexačních struktur můžeme dostat indexační struktury vhodné pro indexování časoprostorových dat.

3.1 Rodina B-stromu

Stromové datové struktury rodiny B-stromu jsou takové struktury, které jsou odvozeny od B-stromu. Blíže je popsána indexační struktura B-strom a B^+-strom. Do rodiny B-stromu patří také B*-strom.

3.1.1 B-strom

B-strom je stromová datová struktura, ve které jsou uložena setříděná data. B-strom umožňuje vyhledávání, vkládání a mazání dat. Jelikož se jedná o stromovou strukturu, veškeré operace probíhají v logaritmickém čase. B-strom vznikl zobecněním binárního stromu. Na rozdíl od binárního stromu je schopen ukládat velký objem dat, proto nachází uplatnění v databázových systémech (např. Oracle). Každý uzel B-stromu je mapován na diskovou stránku, kde jsou uloženy jednotlivé záznamy odkazované z daného uzlu.

Vlastnosti B-stromu stupně n

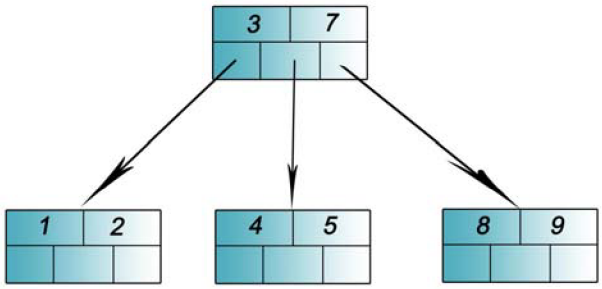
- Kořen má nejméně dva potomky, pokud není listem
- Každý uzel kromě kořene a listu má nejméně $\lceil n/2 \rceil$ a nejvýše n potomků/ukazatelů na potomky[1]
- Každý uzel kromě kořene má nejméně $\lceil n/2 \rceil - 1$ a nejvýše $n-1$ položek[1]
- Všechny cesty od kořene k listům jsou stejně dlouhé
- Data v uzlu jsou organizována tak, aby logicky vytvářela posloupnost $p_0, k_1, p_1, k_2, p_2, ..$, kde k označuje klíče ukládaných dat a p_i označují odkazy na podstromy. Pro k lze zavést relaci uspořádání " < " : $\forall k_i, k_j$, kde $i < j$, platí $k_i < k_j$
- $\forall k$ v podstromě, které odpovídá odkazu p_{i-1}, platí $k < k_i$
- $\forall k$ v podstromě, které odpovídá odkazu p_i, platí $k > k_i$

Princip uložení dat

Data jsou ve stromě uložena jako setříděné hodnoty. Samotné záznamy jsou přístupné z odkazů z uzlů a z listů. V každém uzlu se nachází klíče a ukazatelé na podstromy, viz obr. 3.1. Pokud uzel obsahuje dva klíče k_1 a k_2, potom platí:

[1] Výraz $\lceil x \rceil$ značí nejmenší celé číslo větší nebo rovno x zaokrouhlené nahoru.

- $\forall k : k < k_1$, k je uloženo v levém podstromě
- $\forall k : k > k_1 \wedge k < k_2$, k je uloženo v prostředním podstromě
- $\forall k : k > k_2$, k je uloženo v pravém podstromě



obr. 3.1: B-strom

3.1.2 B$^+$-strom

B$^+$-strom je stromová datová struktura vycházející z B-stromu. Umožňuje rychlé vkládání, vyhledávání a mazání dat. Záznamy jsou přístupné pomocí odkazů až z konce B$^+$-stromu - z jeho listů. Tím se B$^+$-strom liší od B-stromu. Na obr. 3.2 lze vidět ukázku B$^+$-stromu.

Vlastnosti B$^+$-stromu

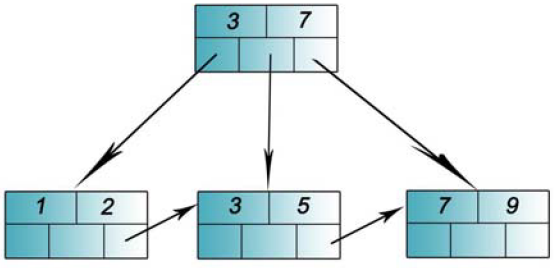
Vlastnosti B$^+$-stromu jsou stejné jako vlastnosti B-stromu s výjimkou dvou posledních bodů, tedy platí:

- $\forall k$ v podstromě, které odpovídá odkazu p_{i-1}, platí $k < k_i$
- $\forall k$ v podstromě, které odpovídá odkazu p_i, platí $k \geq k_i$
- Veškerá data jsou přístupná z listů
- Každý list obsahuje odkaz na následující list

Princip uložení dat

Data jsou uložena podobně jako v B-stromu, ale musíme zohlednit podmínku přístupu k datům pouze z listů. Mějme uzel, který obsahuje dva klíče k_1 a k_2, potom platí:

- $\forall k : k < k_1$, k je uloženo v levém podstromě
- $\forall k : k \geq k_1 \wedge k < k_2$, k je uloženo v prostředním podstromě
- $\forall k : k \geq k_2$, k je uloženo v pravém podstromě



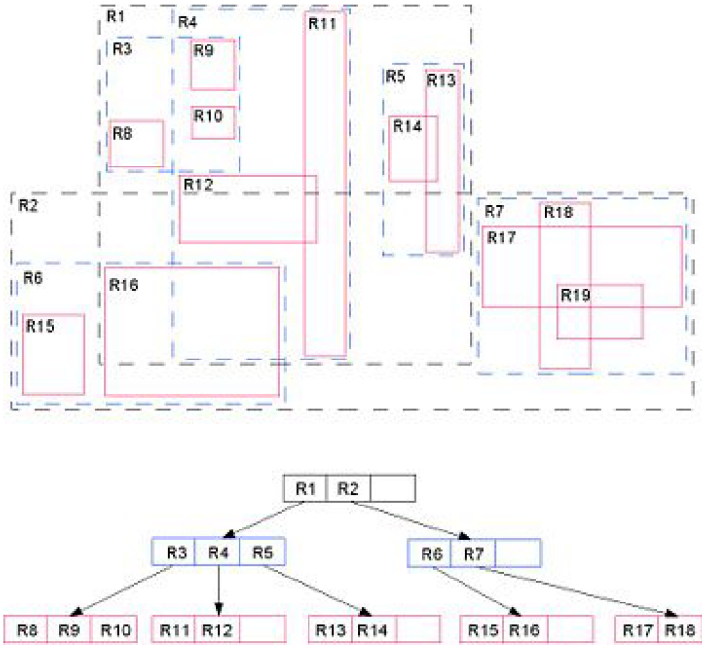
obr. 3.2: B$^+$-strom

3.2 Rodina R-stromu

R-strom je jeden z prvních nejdůležitějších a nejpopulárnějších indexačních struktur. R-strom a jeho pozdější varianty R^*-strom a R^+-strom jsou výškově vyvážené stromy. Vznikly rozšířením B-stromu pro *k* dimenzí. Prostorové objekty jsou založeny na MBB(minimum bounding box – minimální ohraničující boxy), také jsou známy jako MBR(minimum bounding rectangles – minimální ohraničující obdélníky) ve 2D prostoru. Stejně jako u B-stromu, každý uzel R-stromu je mapován na diskovou stránku. Ale zatímco B-stromy jsou postaveny na klíčích, kde je pevně stanovené pořadí, R-stromy organizují MBB dle topologických vztahů. Každý uzel je spojen s k-rozměrným boxem (MBB), ten představuje všechny MBB následujícího uzlu. V této kapitole a podkapitolách jsem čerpala z [1].

3.2.1 R-strom

R-strom se používá především pro prostorové přístupové metody pro indexování vícerozměrné informace. Veškerá data jsou uložena v listech. R-strom ukládá (2D,3D) objekty způsobem, že danou skupinu objektů (pokud se objekty překrývají, bývají v jedné skupině) obalí MBB. MBB obaluje vždy celé objekty. Každý uzel R-stromu má proměnlivý počet záznamů, nicméně je dáno minimum a maximum počtu záznamů z důvodu zachování vyváženosti stromu. Každý záznam v uzlech uchovává další dvě informace: způsob identifikace následujícího uzlu a MBB, který ohraničuje veškeré MBB všech záznamů v následujícím uzlu. Jelikož dotazovaný MBB (při vyhledávání objektů) může protínat několik MBB uložených ve stromě, musíme R-strom procházet několikrát. Na obr. 3.3 lze vidět ukázku R-stromu.



obr. 3.3: R-strom [19]

Vlastnosti R-stromu

- Každý uzel obsahuje minimálně *m* a maximálně *M* záznamů, kde $0 < m \leq (M / 2)$
- Pro každou položku*(mbb, nodeid)* uzlu, *mbb* je MBB, který prostorově obsahuje všechny MBB v jeho následujících uzlech, které jsou odkazovány z *nodeid*
- Pro každou položku*(mbb, oid)* listu, *mbb* je MBB, který prostorově obsahuje k-dimenzionální objekt odkazovaný *oid*
- Kořenový uzel má nejméně 2 záznamy, pokud se nejedná o list
- Všechny listy jsou na stejné úrovni

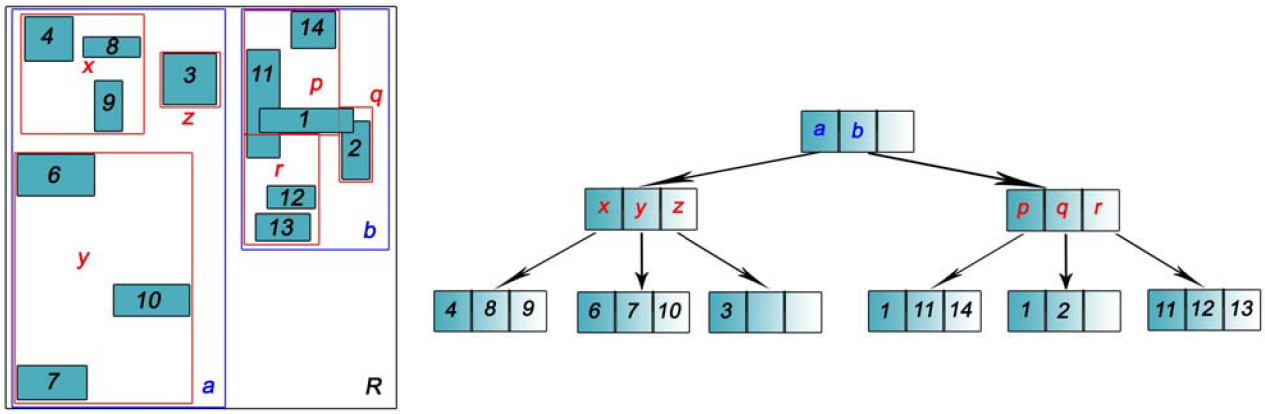
3.2.2 R$^+$-strom

U R$^+$-stromu se MBB na dané úrovni nepřekrývají, ale jsou disjunktní. To nám přináší výhodu při dotazu na bod: existuje pouze jedna unikátní cesta od kořene k listu. K dosažení nepřekrývajících se MBB ploch prostorových objektů, může být MBB rozdělen několika MBB, které budou představovat uzly stromu.

Vlastnosti

- MBB každého uzlu obsahuje všechny MBB jeho podstromu
- R$^+$-strom je založen na nepřekrývajících se MBB, pokud se však některé MBB překrývají, tyto MBB jsou přiřazeny do více listů, či uzlů, které příslušný MBB obsahují
- MBB dvou uzlů na stejné úrovni se nepřekrývají
- Kořenový uzel má nejméně dva záznamy, pokud není listem
- Všechny listy jsou na stejné úrovni

Obr. 3.4 zobrazuje R$^+$-strom. Můžeme vidět, že jednotlivé MBB uzlů se nepřekrývají. MBB *1* je pokryt (rozdělen mezi) MBB *p* a *q*. Stejně tak MBB *11* je rozdělen mezi MBB *p* a *r*. Takové úpravy mohou zvýšit výšku stromu, ale prohledávání bude prováděno efektivněji.



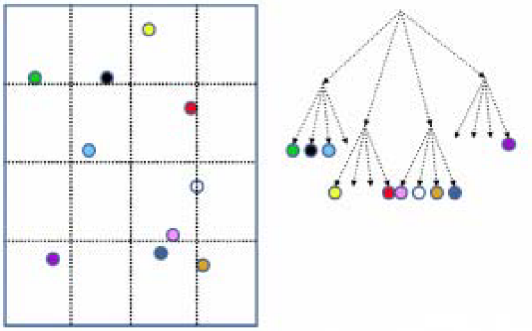
obr. 3.4: R$^+$-strom

3.2.3 R*-strom

R*-strom je variantou R-stromu. Pokouší se minimalizovat pokrytí a překrytí MBB, kde minimalizace překrytí je mnohem důležitější než minimalizace pokrytí. Ve 2D prostoru to závisí na minimalizaci parametrů pro optimalizaci překrytí uzlů a parametru MBB uzlu (obsahuje tvar MBB). Neexistují žádné dobré techniky, které minimalizují všechny parametry.

3.3 Quad-strom

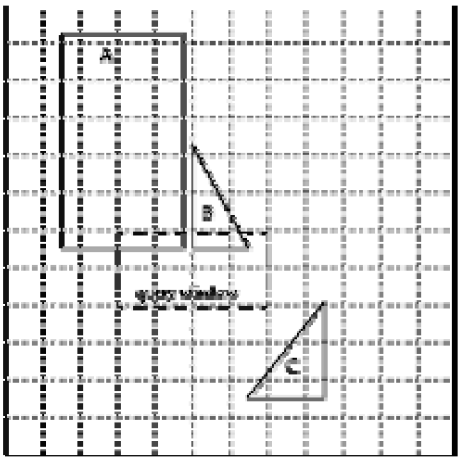
Quad-strom se nejčastěji používá k rozdělení nD prostoru na stejné části, kde n je počet dimenzí. Např. jak je tomu v databázovém systému Oracle, prostor je postupně dělen na 4^i částí pro $i \in \{1,2,..,m\}$ [10]. Toto dělení pokračuje, dokud nejsou splněna zadaná kritéria, jako je např. maximální počet částí prostoru, velikost jednotlivých částí prostoru, atd. Každý interní uzel má až 4 následníky, viz obr. 3.5.



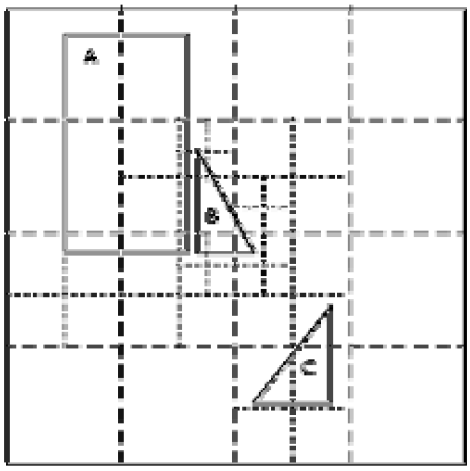
obr. 3.5: Quad-strom [10]

Quad-strom implementovaný v Oracle pracuje na podobném principu jako výše zmíněný. Dále je čerpáno z [4]. Dle nastavení dělení (na fixní velikost jednotlivých částí nebo proměnnou velikost) jsou dvě možnosti, jak prostor rozdělit:

- Fixní velikost: na obrázku 3.6 můžeme vidět, jak lze rozdělit prostor s geometriemi a dotazovacím oknem na části s fixní velikostí. Tento způsob je doporučován z důvodu snadného dotazování na shodnost jednotlivých částí (každá část je stejně velká). Při tomto způsobu dotazování se zvyšuje výkonnost při vyhledávání. Velkou nevýhodou tohoto přístupu je, že musíme zvolit správnou velikost částí, na které se má prostor rozdělit.
- Proměnlivá velikost: v tomto případě je prostor nejdříve dělen na stejně velké části, jako tomu bylo u fixního indexování. Části, které obsahují geometrii, jsou dále opět děleny na podrobnější (menší), než části prostoru, kde geometrie není. Tento typ indexování je vhodný např. pokud chceme indexovat důležité geografické prvky, jako jsou hranice, atd. Ukázku dělení prostoru s proměnlivou velikostí částí lze vidět na obr. 3.7.



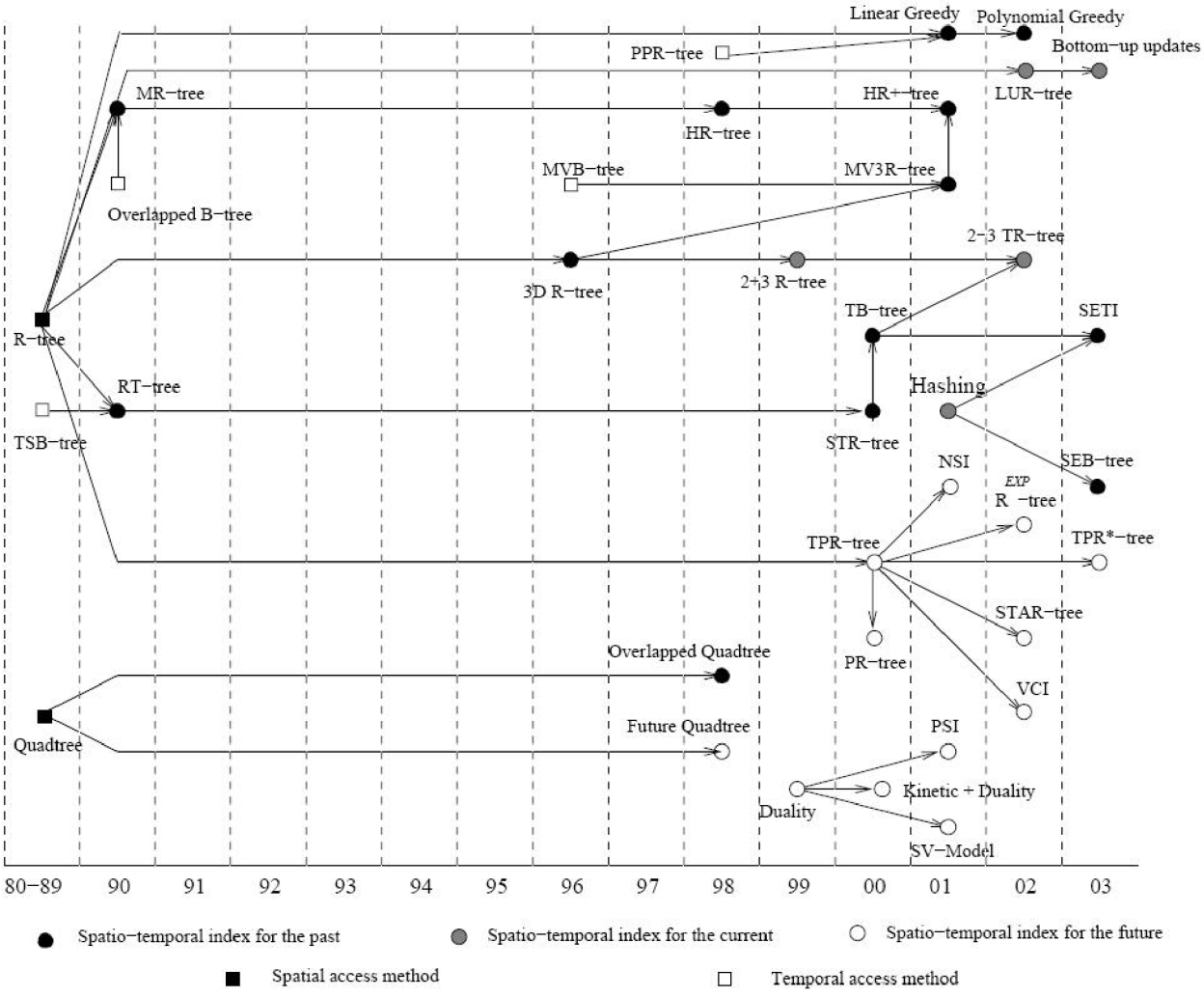
Obr. 3.6: Fixní velikost částí prostoru [4]



obr. 3.7: Proměnlivá velikost částí prostoru [4]

4 Indexování pohybujících se objektů

Časoprostorové indexační struktury vznikly rozšířením již existujících struktur/metod jako je R-strom, Quad-strom. Tyto struktury, které byly určeny pro indexování prostorových dat, nezahrnovaly časovou složku. Stejně jako metody pro indexování časových dat nezahrnovaly prostorovou složku. Proto bylo nutné vytvořit nové metody pro indexování časoprostorových dat. Nicméně každá metoda je vhodná pro jiný typ dotazu. Dotazy můžeme primárně rozdělit na současné, budoucí a současné, a na dotazy vztahující se k minulosti. Na obrázku 4.1 je znázorněn časový vývoj indexů a přístupů v letech 1980-2003 vhodných pro časoprostorová data. Jsou zde rozlišeny přístupy, které jsou vhodné pro jednotlivé typy dotazů. V následujících kapitolách jsou popsány některé přístupy a indexy. Následující podkapitoly byly, pokud není uvedeno jinak, převzaty z [6].



obr. 4.1: Časový vývoj indexů [6]

4.1 Indexování současného pohybu

Následující přístupy jsou vhodné především pro zodpovězení dotazů, které se vztahují k aktuálnímu času. Tedy chceme vědět, co se děje s objekty právě teď. Zástupci této metody jsou 2+3 R-strom, 2-3 TR-strom, LUR-strom a hašování. Dále je podrobně popsán pouze LUR-strom.

LUR-strom (Lazy Update R-tree)

LUR-strom je indexační metoda, která uchovává pouze aktuální data. Je založena na R-stromu. Veškeré algoritmy a indexové struktury jsou podobné těm, které jsou použity v R-stromě, pouze je přidán algoritmus pro aktualizaci (update). LUR-strom využívá sekundární indexační strukturu. V této kapitole bylo čerpáno z [11].

Popis problému u R-stromu

LUR-strom se zaměřuje na snížení počtu aktualizačních operací u časoprostorových dat. Pokud budeme předpokládat několik pohybujících se objektů, za každý časový interval je do databáze odeslána dvojice (*oid,* p_{new}), kde *oid* je identifikátor objektu a p_{new} je nová pozice objektu. Jelikož databáze uchovává pouze aktuální data, veškeré objekty musí být aktualizovány. R-strom nedovoluje objekty aktualizovat, v tomto případě se aktualizace řeší smazáním a vložením objektu. To má za následek velký počet přístupů na disk, zvláště v případě, že se uzly R-stromu musí spojovat a rozdělovat. Z tohoto důvodu LUR-strom má nepatrně pozměněnou indexační strukturu a obsahuje novou metodu pro aktualizaci pozice objektu.

Aktualizace objektu

Při aktualizaci pozice objektu má příslušná funkce k dispozici dvojici (*oid,* p_{new}). Ke struktuře R-stromu je přidána další indexační struktura nazvaná DirectLink. Prvky v této struktuře mohou být přístupné přes B-strom nebo přes metodu hašování. DirectLink obsahuje *oid* a ukazatel na objekt v listu R-stromu, jehož *id* je právě *oid*. Každý index má ukazatel na odpovídající záznam v listu. Samotná funkce update funguje tak, že pokud se objekt nepohybuje mimo MBB, objekt je stále v rámci MBB, je pouze aktualizovaná informace o pozici. Pokud se objekt pohybuje mimo MBB, existuje několik přístupů: objekt je smazán a znovu vložen, MBB je rozšířen (pokud se objekt nepohybuje příliš daleko).

4.2 Indexování současného a budoucího pohybu

K předpovědi budoucího pohybu potřebujeme další informace jako rychlost, vzdálenost, atd. Pohyb objektu v d-dimenzionálním prostoru je modelován referenční pozicí:

$\vec{x}_{ref} = (x_1, x_2, .., x_d)$ v referenčním čase t_{ref} a vektorem rychlosti $\vec{v} = (v_1, v_2, .., v_d)$.

Předpovídaná pozice $\vec{x}_t$ pohybujícího se objektu v čase $t > t_{ref}$ lze spočítat:

$$\vec{x}_t = \vec{x}_{ref} + \vec{v}(t - t_{ref})$$

4.2.1 Transformační metody

Transformuje časoprostorovou doménu (kde jedna osa je určena pro čas, druhá pro pozici) do jiného prostoru. Cílem této transformace je jednodušší reprezentace a dotazování se na data. Mezi tyto

techniky patří duální transformace, duální transformace s kinetickou datovou složkou, SV-model a PSI. PSI bude v této kapitole blíže představeno.

PSI (Parametric Space Index Technique)

Jedná se o indexační techniku, která používá R-strom k indexování (*2d+1*) – dimenzionálního prostoru, kde *d* rozměry odpovídají referenční pozici x_{ref}, *d* rozměry odpovídají rychlosti *v* a jeden rozměr odpovídá času (proto je nutné indexovat (d+d+1)-dimenzionální prostor). Pohyb objektu je modelován (*2d+1*)-dimenzionální trajektorií, která je reprezentována úsečkami o koncových bodech: $(\vec{x}_{ref}, \vec{v}, t_s)$ a $(\vec{x}_t, \vec{v}, t_e)$, kde (t_s,t_e) je časový interval, ve kterém se objekt pohyboval. Trajektorie jsou obaleny MBB. Každý MBB požaduje uložení *2(2d+1)* hodnot bodů. Hlavní myšlenkou je uložení časového intervalu (t_s,t_e), ve kterém se daný objekt pohybuje, do indexu. To znamená, že neexistuje žádný globální čas odkazovaný z jednotlivých objektů. Informace o PSI byly čerpány z [6,15].

4.2.2 Parametrické prostorové přístupové metody

Parametrické prostorové přístupové metody indexují původní časoprostor s parametrickými obdélníky. Cílem tohoto přístupu je, aby parametrické obdélníky byly funkcí času, takže pohybující se objekt bude ve stejném obdélníku. Stromy založené na této metodě jsou např. TPR-strom, TPR*-strom a PR-strom.

TPR*-strom (Time-Parameterized R*-tree)

TPR*-strom je indexační technika, která rozšiřuje R*-strom a využívá lineárních funkcí pro indexování pohybujících se objektů. Využívá se pro indexování (a s ní související predikci) současného a budoucího pohybu. Lze indexovat objekty v jedno, dvou nebo tří-dimenzionálním prostoru. TPR*-strom indexuje data ve svém prostoru, nevyužívá replikaci a nepožaduje pravidelnou obnovu indexu. Informace o TPR*-stromu byly čerpány z [1].

Popis problému

R*-strom nám neumožňuje predikci budoucího pohybu objektu, to má za následek časté aktualizace (mazání a vkládání) objektu. Cílem TPR*-stromu je zachytit spojitý pohyb tak, aby bylo vyžadováno minimum aktualizací.

Struktura TPR*-stromu

Listy obsahují pozici pohybujícího se objektu a ukazatel na objekt v databázi. Uzly obsahují ukazatel na podstrom a MBB, ohraničující všechny objekty v příslušném podstromu.

TPR*-strom pro indexování objektů používá lineární funkce, které mají jako parametry referenční pozici objektu x_{tref} (pozice objektu v čase *tref*) a vektor rychlosti $\vec{v}$. Platí:

$$x_t = x_{tref} + \vec{v}(t - tref), \text{ kde } x_t=(x_{1t}, x_{2t}, x_{3t} \ldots)$$

Pozice pohybujícího se objektu je poté reprezentována referenční pozicí a vektorem rychlosti.

K ohraničení skupiny d-dimenzionálních pohybujících se objektů se používají d-dimenzionální ohraničující boxy, které jsou funkcí času. Základním úkolem je najít rovnováhu mezi přesností, s jakou MBB ohraničují pohybující se objekty, a náklady na uložení.

K tomu TPR*-strom využívá tzv. konzervativní ohraničující boxy, které v některých počátečních časech jsou minimální. Spodní hranice tohoto obdélníku je nastavena na pohyb s minimální rychlostí uzavřených bodů. Horní hranice konzervativního boxu je nastavena na pohyb s maximální rychlostí uzavřených bodů. Z těchto omezení je garantováno, že obdélník vždy obsahuje uzavřenou množinu pohybujících se objektů [6].

Výhodou tohoto přístupu a samotného TPR*-stromu je částečné vyřešení častých aktualizací pomocí předpovědi budoucí pozice objektu.

4.3 Indexování historie pohybu

Tato kapitola se zaměřuje na historická data. Jelikož množství dat s časem roste, můžeme využít dva přístupy, jak minimalizovat celkovou velikost historických dat.

První přístup je vzorkování. Tok dat je vzorkován na určité pozice vzhledem k času. Z těchto diskrétních dat se vytváří pomocí lineární interpolace trajektorie dat. Cílem je se dotazovat a indexovat spojitě se pohybující objekty (ne diskrétně se měnící pozice objektů) a minimalizovat velikost časoprostorových dat.

Druhým přístupem je aktualizovat časoprostorová data pouze při jejich změně. Pohybující se objekt pošle informaci o změně pouze při změně rychlosti, směru atd. Časoprostorové metody/indexační schémata můžeme rozdělit do 4 kategorií:

- Metody, které mají čas jako samostatnou dimenzi
- Metody, které včleňují časovou informaci do uzlu
- Metody, které používají překrývající se indexové struktury pro reprezentaci stavu databáze v odlišném čase
- Metody orientované na trajektorii

4.3.1 Časová dimenze

Tato kategorie metod se především zabývá prostorovou stránkou časoprostorových dat a jejich správnou indexací. Časová složka je v tomto případě druhořadou záležitostí. Jeden z indexů, který je založen na tomto přístupu, je 3D R-strom.

3D R-strom

3D R-strom modeluje časovou složku jako nový rozměr k prostorovým rozměrům. Hlavní myšlenka tohoto přístupu je nezvýhodňovat časovou nebo prostorovou složku dat. 3D R-strom podporuje časové i prostorové dotazy, nicméně je to za cenu snížení výkonnosti.

4.3.2 Časová informace obsažena v uzlu

Tato kategorie metod včleňuje časovou informaci jako časový interval do struktury jednotlivých uzlů. Příkladem této struktury je RT-strom.

RT-strom

RT-strom kombinuje prostorové přístupové metody z R-stromu a časové přístupové metody. Záznam v RT-stromu sestává ze čtveřice: (id,MBR,t_s,t_e), kde *id* - identifikátor objektu, MBR – minimální ohraničující obdélník objektu, *ts* a *te* udává časový interval, ve kterém daný objekt platí. Efektivnost prostorových dotazů je stejná jako u R-stromu, ale časové dotazy mohou být v rozsahu celého stromu.

4.3.3 Překrývající se a mnohočetné struktury

Tento typ struktur odděluje časovou dimenzi od prostorových dimenzí. Cílem je zachovat všechna data, která se nachází v jednom čase, v jedné indexační struktuře a následně vybudovat samostatný R-strom pro každý čas. Nevýhodou tohoto přístupu je nadměrné ukládání. Jako příklady lze uvést MR-strom, HR-strom, HR^+-strom a MV3R-strom. V této kapitole jsou blíže popsány struktury HR-strom a HR^+-strom.

HR-strom (Historical R-tree)

HR-strom je založen na technice překrývání. HR-strom obsahuje R-strom pro každý čas zavedený do databáze, ale společné větve po sobě následujících R-stromů jsou ukládány pouze jedenkrát, což ušetří místo. Tyto společné větve jsou sdíleny 2 a více R-stromy. To znamená, že na uzel, který obsahuje stejná data pro nějaký časový interval, se odkazují uzly z více R-stromů.

Pokud se dotazujeme na pozici objektu v určitém čase, je nejdříve vyhledán R-strom s daným časem a dále prohledán za nalezením daného objektu. Pro rychlejší vyhledávání jsou implementovány tzv. pozitivní a negativní ukazatelé, které rozlišují, zda jde o uzel sdílený nebo samostatný.

Nevýhodou tohoto přístupu je redundance dat a vysoký požadavek na velikost prostoru pro uložení indexu. Je to dáno tím, že pokud pouze jeden objekt změní svojí pozici, celá větev, která tento objekt obsahuje (obsahuje i jiné objekty, které pozici nemusely změnit), je následně duplikována a uložena do nového R-stromu. Čímž vzniká obrovské množství R-stromů o relativně velké velikosti. Informace o HR-stromu byly čerpány z [14].

HR^+-strom

HR^+-strom je navržen k zabránění zbytečné replikace položek v HR-stromě. HR^+-strom povoluje, aby údaje s různým časovým razítkem byly v jednom uzlu, ale rodičovský uzel má přístup pouze k údajům následujícího uzlu, které mají stejné časové razítko. Uzel může mít více rodičovských uzlů, ale každý rodičovský uzel má přístup pouze do dané části následujících uzlů.

4.3.4 Přístupové metody orientované na trajektorii

Tato kategorie se zaměřuje na metody, které jsou orientovány na trajektorii. Zástupci jsou SETI, SEB-strom, STR-strom a TB-strom. Poslední dvě jmenované metody jsou popsány podrobněji.

STR-strom (Spatio-Temporal R-tree)

STR-strom rozšiřuje 3D R-strom a podporuje efektivní zpracování dotazů na trajektorie pohybujících se objektů. Souřadnicový systém je tří-dimenzionální (2 dimenze pro určení polohy, 1 dimenze pro určení času). Hlavní rozdíly mezi 3D R-stromem a STR-stromem je struktura listů a přístupové metody pro vložení a rozdělení uzlů. Cílem této metody je držet segmenty patřící do stejné trajektorie při sobě a vkládat nový řádek segmentu co nejblíže k jeho předchůdci v trajektorii. Listové uzly obsahují záznam: (*id, t_{id}, MBR, o*), kde *id* je identifikátor objektu, *MBB* – minimální ohraničující kvádr (box), t_{id} – identifikátor trajektorie, *o* – orientace trajektorie v MBB. Informace pro tuto podkapitolu byly čerpány z [1].

TB-strom (Trajectory-Bundle R-tree)

TB-strom je založen na struktuře R-stromu. Listové uzly mohou obsahovat pouze segmenty, které patří do stejné trajektorie. Nevýhodou tohoto přístupu je, že úsečky z různých trajektorií, které leží blízko sebe, budou obsaženy v různých listech TB-stromu. Struktura ukládaných záznamu do listů stromu je podobná jako u STR-stromu. Ale jelikož TB-strom nepovoluje vkládat do listu segmenty různých trajektorií, t_{id} – identifikátor trajektorie je přiřazen uzlu a list obsahuje záznam: (*id,MBR, o*). Informace pro tuto podkapitolu byly čerpány z [1].

5 Databázový systém Oracle

Podpora pro prostorová data se nachází v Oracle Spatial 11g jako možnost Oracle Database 11g Enterprise Edition. Databáze poskytuje vyspělejší prostorové schopnosti pro podporu geoprostorových aplikací a služeb založených na pozici.

5.1 Oracle Spatial 11g

Oracle Spatial 11g přináší nové výhody a vylepšené rysy. Dále jsou uvedeny jedny z nejzajímavějších vlastností. V této kapitole bylo čerpáno z [3].

Podpora pro 3D datový typ

Oracle Spatial 11g poskytuje nativní ukládání, dotazování a vyhledávání pro 3-dimenzionální (3D) data včetně bodů, čar, ploch a trojúhelníkové nepravidelné sítě. 3-dimenzionální data jsou také podporována indexační strukturou R-stromu. Dále jsou poskytovány některé SQL operátory a analytické funkce pro 3D data.

Podpora pro segmentaci prostorových indexů

Architektura Oracle databáze zahrnuje segmentování, kde jedna logická tabulka a její indexy jsou rozděleny do jedné nebo více fyzických tabulek. Každá fyzická tabulka má svůj vlastní index. Prostorové indexy spojené se segmentovanými tabulkami mohou být také segmentovány. Segmentace přináší vyšší výkon, škálovatelnost a další výhody jako snížení doby odezvy na dlouho běžící dotazy, souběžné dotazy a snadnější údržbu indexů.

Paralelní vytvoření prostorového indexu

Prostorové indexy a indexační tabulky mohou být vytvořeny paralelně. Vytvoření R-stromu indexu může být rozděleno na menší úlohy, které mohou být prováděny paralelně s pomocí nevyužitého CPU. Takové vytvoření indexu podstatně šetří čas.

Paralelní prostorové dotazy

Prostorové dotazy mohou nyní běžet paralelně na segmentovaných prostorových indexech. Paralelní dotazy zlepšují výkon dotazů, které se týkají vzdálenosti, nejbližších sousedů a vztahových dotazů.

Neviditelný index

Neviditelný (invisible) index je index, který je spravován databází, ale ignorován optimalizátorem. Ignorování optimalizátorem lze zakázat nastavením příslušného parametru [5].

5.2 Typy geometrie

Oracle Spatial poskytuje SQL schéma a funkce, které usnadňují ukládání, vyhledávání a aktualizaci kolekce prostorových rysů v databázi. Oracle poskytuje MDSYS schéma, které popisuje uložení, syntaxi a sémantiku podporovaných geometrických datových typů. Dále poskytuje objektově-relační model pro reprezentaci geometrie. Takový model ukládá veškerou geometrii jako prostorový datový typ: `SDO_GEOMETRY`. Geometrický popis prostorových objektů je uložen ve sloupci objektového typu `SDO_GEOMETRY`. Tabulky, které obsahují sloupec typu `SDO_GEOMETRY`, se označují jako prostorové. V této kapitole bylo čerpáno z [4].

Oracle Spatial definuje objektový typ SDO_GEOMETRY jako:

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

Jednotlivé sloupce tohoto typu jsou:

- SDO_GTYPE – označuje typ geometrie, jedná se o číslo ve formátu DLTT
    - D – počet dimenzí
    - L – míra dimenze pro 3D linear referencing system (LRS) geometrie
    - TT – typ geometrie: 00 - 09 (bod, úsečka, polygon, těleso, atd.)
- SDO_SRID – k určení souřadnicového systému, který má být spjat s geometrií, pokud je null, žádný souřadnicový systém není s geometrií spjat
- SDO_POINT – je definován pomocí SDO_POINT_TYPE (s atributy X,Y,Z – souřadnice bodu), pokud jsou hodnoty SDO_ELEM_INFO a SDO_ORDINATES null a atribut SDO_POINT není null, potom SDO_POINT_TYPE s vlastnostmi X,Y,Z jsou souřadnicemi bodu, v opačném případě je SDO_POINT ignorován
- SDO_ELEM_INFO – určuje, jak interpretovat souřadnice. Každá trojice množiny čísel je interpretována:
    - SDO_STARTING_OFFSET – offset v poli SDO_ORDINATES, kde je uložena první souřadnice pro daný prvek, jedná se o číslo v rozsahu *1-n*, kde *n* je n-té číslo v SDO_ORDINATES
    - SDO_ETYPE – typ prvku (bod, lomená úsečka, lomený oblouk, polygon, obdélník, kruh, atd.)
    - SDO_INTERPRETATION – interpretace SDO_ETYPE, pokud SDO_ETYPE určuje prvek, který je složen z více typů prvků (např. kombinace lomené úsečky a oblouku), SDO_INTERPRETATION určuje, kolik následujících trojic prvku je součástí složeného prvku
- SDO_ORDINATES – množina čísel tvořící hranici prostorového objektu ve formátu: (P1,P2,…), kde P1,P2 jsou n-tice čísel, kde n je počet dimenzí.

5.3 Prostorové operátory

V této kapitole jsou popsány operátory, které se mohou použít pro práci s prostorovými daty. Budou vysvětleny: SDO_JOIN, SDO_NN, SDO_NN_DISTANCE, SDO_FILTER, SDO_REALTE, SDO_WITHIN_DISTANCE. Prostorové operátory SDO_FILTER a SDO_RELATE poskytují optimální výkon, protože využívají primárního nebo i sekundárního filtru viz 5.4.1 Model dotazu. Pokud vstupní parametry (geometrie hledaných objektů a dotazovaného objektu) nemají stejný souřadnicový systém, provede se implicitní transformace souřadnicového systému. Při použití prostorového operátoru je nutné mít alespoň jednu geometrii indexovanou.

Při dotazování nad prostorovými daty se mohou také využít procedury a funkce. Ty jsou poskytovány jako podprogramy v jazyce PL/SQL. Nevyžadují definování prostorového indexu a ani jej nevyužívají, když je již vytvořen. Obě vstupní hodnoty (geometrie) musí mít stejný souřadnicový systém. V této kapitole bylo čerpáno z [4].

SDO_JOIN

Provádí prostorové spojení na základě jednoho nebo více topologických vztahů, které jsou zadány jako parametr tohoto operátoru. `SDO_JOIN` po technické stránce není operátor ale tabulková funkce. Nicméně bývá uveden mezi operátory, protože má podobné použití.

SDO_NN

Používá prostorový index k identifikaci nejbližších sousedů. Chování tohoto operátoru určuje parametr operátoru, ve kterém se může specifikovat vzdálenost, počet řádků k vyhodnocení (v případě, že `SDO_NN` výraz musí být vyhodnocen vícekrát, `SDO_NN` může být použit pouze při použití R-stromu), atd. Tento operátor je nedostupný, pokud neexistuje prostorový index.

SDO_NN_DISTANCE

Vrátí vzdálenost objektu, který byl vyhodnocen pomoci `SDO_NN`.

SDO_FILTER

Nalezne/určí geometrie, které spolu interagují – nejsou disjunktní. Tuto operaci vykonává pouze primární filtr.

SDO_RELATE

Nalezne/určí geometrie, které spolu interagují způsobem, který je určen ve vstupním parametru operátoru. Tuto operaci vykonává primární i sekundární filtr.

SDO_WITHIN_DISTANCE

Nalezne objekty, které jsou uvnitř zadané vzdálenosti od daného objektu. Nastavením `querytype` u 3. parametru operátoru můžeme určit, zda vzdálenost bude aproximována (nastavením `querytype=FILTER`) nebo přesně určena. U aproximované vzdálenosti se použije pouze primární filtr, ale u přesně určené se použije i sekundární filtr.

5.4 Indexování

V této kapitole jsou uvedeny možnosti indexování v databázovém serveru Oracle, které lze využít pro indexování časoprostorových dat, a model dotazu. Model dotazu popisuje, jak jsou data získávána z databáze při dotazu na ně.

5.4.1 Model dotazu

Prostorová databáze používá dvouvrstvé modely dotazů pro řešení prostorových dotazů. Dvouvrstvý model dotazu znamená, že vykonávání dotazu probíhá ve dvou fázích. Nejdříve je dotaz zpracován první fází a výsledky z první fáze jsou vstupními daty do druhé fáze. Výstup z druhé fáze přináší přesnou množinu výsledků. V první fázi se používá primární filtr a ve druhé sekundární:

- Primární filtr – umožňuje rychlý výběr z kandidátních záznamů k následnému poslání sekundárnímu filtru. Primární filtr porovnává aproximace geometrií za účelem redukování výpočetní složitosti. Proto je výpočetně méně náročný. Jelikož primární filtr porovnává aproximace geometrií, vrací množinu s přibližnými výsledky.
- Sekundární filtr – aplikuje přesné výpočty na geometrie, které jsou výstupem primárního filtru. Sekundární filtr dává přesnou odpověď na dotazy. Operace sekundárního filtru jsou výpočetně náročné, ale jelikož se operace sekundárního filtru aplikují na výsledky z primárního filtru, ne na celou datovou množinu, náročnost výpočtu se snižuje.

V této kapitole bylo čerpáno z [4].

5.4.2 Indexy

V databázovém systému Oracle lze data indexovat větším počtem indexů. Oracle má v sobě některé indexy zabudované jako je R-strom, Quad-strom a B-strom. Nicméně je možné vytvářet své vlastní indexy, které jsou založeny na funkci nebo doménové indexy. Dále lze vytvářet segmentované a neviditelné indexy.

Vestavěné indexy

Oracle má vestavěné indexy: B-strom, Quad-strom, R-strom. Pro indexování prostorových dat se především používají indexy Quad-strom a R-strom. Princip těchto indexačních struktur byl popsán v kapitole 3.2 a 3.3. Proto si v této kapitole popíšeme R-strom a Quad-strom z pohledu samotného uložení a fungování v Oracle. V této kapitole bylo čerpáno z [9].

Quad-strom a R-strom jsou implementovány v Oracle za použití rozšiřitelného indexačního frameworku. Vytvořené indexy jsou uloženy v tabulce, kde u R-stromu jsou především uloženy uzly a u Quad-stromu jsou uloženy části rozděleného prostoru, jejichž velikost je dána parametrem SDO_LEVEL a NUMTILES (při použití hybridního indexování). Tyto parametry jsou blíže popsány u experimentů v kapitole 7. Při vytváření prostorového indexu, jsou definovány nové predikáty a operátory. Operátory mají shodnou sémantiku bez ohledu na to, zda jde o Quad-strom nebo R-strom.

Zpracování dotazu s využitím Quad-stromu a R-stromu

V první fázi se používá primární filtr, který využívá prostorový index k filtrování kandidátských geometrií. V této fázi jsou s pomocí externí aproximace nalezeny geometrie, které mohou splňovat kritéria dotazu. U Quad-stromu se k externí aproximaci používají části rozděleného prostoru, v případě R-stromu se k externí aproximaci používají MBR.

Ve střední části jsou kandidátské geometrie porovnávány s dotazovanou geometrií za použití interní aproximace (popsána níže). Některé kandidátské geometrie jsou následně odstraněny nebo přijaty na základě zadaných kritérií. Zbytek dat, jejichž interakce nebyla stanovena ve střední části zpracování dotazu, je následně poslána do sekundárního filtru (poslední fáze).

Ve 3. fázi, kde vstupují geometrie z prostřední fáze, jsou určeny přesné výsledky a vráceny uživateli. Sekundární filtr používá na rozdíl od obou předešlých, které používaly aproximaci dotazu a geometrii, přesné výpočetní algoritmy.

Primární filtr Quad-strom

Dle zvoleného stupně (tím se určí i maximální velikost jednotlivých částí) pro dělení prostoru se dělí prostor i geometrie na jednotlivé části. Každá část je identifikována kódem. Při dělení prostoru na části je každá část rozdělena do jedné ze dvou skupin:

- Interní: část rozděleného prostoru je zcela uvnitř geometrie
- Hraniční: část rozděleného prostoru je protnuta geometrií (např. obsahuje hranu obdélníku)

Do tabulky indexů jsou následně vloženy pouze části prostoru, které kryjí geometrii, ve formě: (*tile_code, s, id*), kde *tile_code* je kód části, *s* je status (interní/hraniční část), *id* je identifikátor geometrie. Pro urychlení dotazů je následně konstruován B-strom nad *tile_code* a *id*.

Primární filtr R-strom

R-strom spravuje logickou stromovou strukturu, ale fyzicky je uložen v tabulce. Každý uzel R-stromu odpovídá řádku tabulky a ukazatel v R-stromu odpovídá *id* řádku v tabulce. Ukazatel na kořen je uložen v metadatech pro index. Řádky, které představují listové uzly, ukládají MBR a identifikátor geometrie.

Prostřední filtr

V primárním filtru index používá externí aproximaci a určuje, zda geometrie spolu interagují. Pokud spolu interagují, dotaz, geometrie a interní aproximace jsou přeneseny do prostředního filtru. Prostřední filtr porovnává dotaz a geometrie. Pokud vyhodnotí, že spolu interagují, vrací je jako výsledná data, pokud vyhodnotí, že spolu neinteragují, jsou vyloučena. A pokud prostřední filtr nic nevyhodnotí, data dále pokračují do sekundárního filtru. Prostřední filtr používá k vyhodnocování interní aproximaci. V případě Quad-stromu jsou hraniční a interní části dotazu porovnávány s interními a hraničními částmi geometrie. V případě R-stromu je použit pouze vnitřek dotazované geometrie. Vnitřky ostatních geometrií nejsou počítány.

Interní aproximace – Quad-strom

Při vytváření indexu jsou části prostoru děleny na hraniční a interní. Proto při dotazu se pouze porovnávají jednotlivé části prostoru dotazu s uloženými částmi prostoru geometrie.

Interní aproximace – R-strom

K získání interní aproximace se používá dvoubodový přístup:

- U konvexní geometrie se geometrie rozdělí na 4 části dle osy *x* nebo *y* a vypočítá se největší vepsaný obdélník pro každou z částí.
- U konkávní geometrie se nejdříve 4x rekurzivně rozdělí MBR na kvadranty, takové dělení se provádí u kandidátských geometrií i u dotazované geometrie. Následně jsou jednotlivé části porovnávány.

Tento přístup dosahuje značného zlepšení výkonu.

Segmentované indexy

Segmentovaný prostorový index můžeme vytvořit na segmentované tabulce. Segmentování tabulky (rozdělení na menší oddíly) se provádí u velmi velkých tabulek. Každý takový oddíl tabulky nebo indexu musí mít stejné logické atributy: názvy sloupců, datové typy, omezení. Jednotlivé menší části jsou potom lépe upravovatelné, zlepšuje se ovladatelnost, dostupnost a výkonnost. Největší výhody segmentovaného indexu jsou:

- Snížení doby odezvy na dlouhotrvající/souběžné dotazy
- Snadnější údržba indexů
- Paralelní dotazy na více oddílů

Při vytváření indexu na sloupec s prostorovými údaji se musí zadat klíčové slovo `LOCAL`:

```
CREATE INDEX counties_idx ON counties(geometry)
    INDEXTYPE IS MDSYS.SPATIAL_INDEX LOCAL;
```

V tomto příkladu je vytvořený segmentovaný index závislý na segmentované tabulce. Index je vytvořen pro každý oddíl tabulky. V této kapitole bylo čerpáno z [4].

Indexy založené na funkci

Indexy založené na funkci vypočítají hodnotu funkce (příslušných výrazů) a tato hodnota je uložena do indexu. Index založen na funkci lze vytvářet i nad typem `SDO_GEOMETRY`. Takové indexy zvyšují rozmanitost v přístupu k datům. Funkce použita pro vytvoření indexu musí být deklarována jako `DETERMINISTIC`. Pokud vytváříme funkci vracející `SDO_GEOMETRY`, musíme následně aktualizovat tabulku `USER_SDO_GEOM_METADATA`, která obsahuje metadata (informace) o prostorových tabulkách (sloupcích). V této kapitole bylo čerpáno z [4].

Neviditelné indexy

Jak již bylo uvedeno výše, neviditelný index je index, který je ignorován optimalizátorem, dokud se nenastaví parametr `OPTIMIZER_USE_INVISIBLE_INDEXES` na `TRUE`. Velkou výhodou tohoto

indexu je možnost testovat index bez ovlivnění všech ostatních dotazů nebo je ho možno využít jako dočasný index. Neviditelný index se vytváří přidáním klausule `INVISIBLE`. V této kapitole bylo čerpáno z [5].

Doménové indexy

Doménové indexy se vytváří za použití rozšiřitelného indexačního frameworku. Umožňují realizovat indexování, které není v databázovém systému vestavěno. Účelem doménových indexů je jako u ostatních indexů rychlejší a efektivnější vyhledávání a získávání dat, ale je potřeba zvolit nejvhodnější typ indexování. Žádná indexační metoda není výhodná pro všechny typy dat a dotazů. V této kapitole bylo čerpáno z [4].

Doménové indexy nám poskytují možnost vytvořit si vlastní metody indexace a tím zvýšit výkonnost a snadnost použití. Doménové indexy mohou vyřešit problémy jako zavádění nových vyhledávacích operátorů, indexování nestrukturovaných dat a indexování atributů sloupců.

Rozšiřitelný indexační framework se skládá z následujících komponent:

Indextypes: objekt indextype definuje postupy, které řídí definici a údržbu operací, které jsou volány při vytváření indexu. Např. se definuje funkce, která je volána při vytvoření indexu – zadáním příkazu `create index ..`

Domain indexes: jedná se o aplikačně-specifický index vytvořený na základě indextypes, doménový index je instancí indexu, který je vytvořen, spravován a řízen z daného indextype

Operators: Aplikačně-specifické operátory mohou být využity dotazy a příkazy pro manipulaci s daty, uživatelem definované operátory jsou vázány na funkce

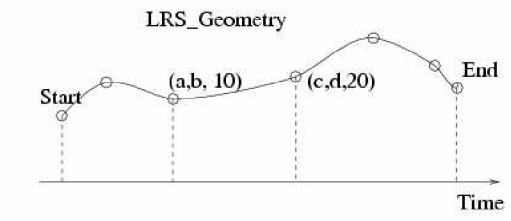
Index-Organized Tables: V tomto případě je indexační struktura modelována jako tabulka (Index-Organized Table), kde každý řádek představuje index.

5.4.3 Indexování časoprostorových dat

Jelikož Oracle obsahuje pouze indexy pro prostorová data, časoprostorová data se musí přepracovat tak, aby mohla využívat podpory pro prostorová data. Asi nejjednodušším řešením je modelovat čas jako další prostorovou dimenzi. Nicméně to má za následek nižší výkon. V následujících kapitolách budou popsány možnosti indexování časoprostorových dat v databázovém systému Oracle. V této kapitole a podkapitolách bylo čerpáno z [2].

5.4.3.1 Linear Referencing Systém (LRS)

LRS se používá k indexování třetí (časové) dimenze časoprostorových objektů. Výhodou LRS je, že časová dimenze se může indexovat odděleně od prostorových dat bez potřeby vytvoření úložiště (sloupce) pro časovou dimenzi. LRS geometrie na obrázku 5.1 ukazuje trajektorii pohybujícího se objektu. Bod (a, b, 10) představuje objekt se souřadnicemi (a, b) v čase 10. Počáteční a koncová doba je indexována pomocí funkcí obsažených v balíčku LRS. Prostorová data jsou následně indexována pomocí R-stromu.



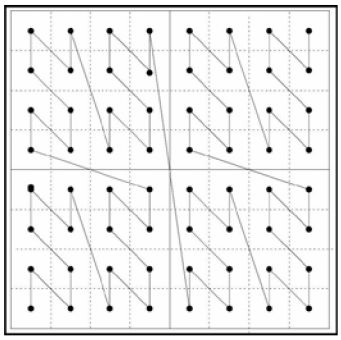
obr. 5.1: LRS geometrie [2]

Při vkládání LRS geometrie je nutné zadat odpovídající parametr `G_TYPE`. Do `SDO_ORDINATES` je možné vložit celou trajektorii, pokud počet vkládaných hodnot nepřesahuje limit pole `SDO_ORDINATES`. LRS geometrie může vypadat následovně:

```
MDSYS.SDO_GEOMETRY(
        3302, NULL, NULL,
        MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1),
        SDO_ORDINATE_ARRAY(x1,y1,t1, x2,y2,t2, x3,y3,t3)
)
```

5.4.3.2 Z-křivka + B-strom

Další možností, jak indexovat časoprostorová data, jsou křivky vyplňující prostor. Prostor je rozdělen mřížkou do částí. Křivka prochází jednotlivými částmi právě jednou a části, kterými křivka prošla, jsou zaznamenány a uloženy v lineárním pořadí. K indexaci takového přístupu lze použít B^+-strom (uložené hodnoty jsou v B^+-stromě seřazeny dle hodnoty). Objekty mohou být reprezentovány čísly, která přesně specifikují, o kterou část prostoru se jedná. Obr. 5.2 zobrazuje příklad vyplnění prostoru křivkou.



obr. 5.2: Z-křivka [2]

V ideálním případě chceme, pokud jsou dva body blízko v Euklidovském prostoru, aby byly blízko v uspořádání, které je definováno křivkou. Kompromis mezi cenou pro vytvoření křivky a dobrými vlastnostmi nám poskytuje Z-křivka, viz obrázek 5.2. Podpora pro časoprostorová data se vytvoří kombinací Z-křivky a B-stromu.

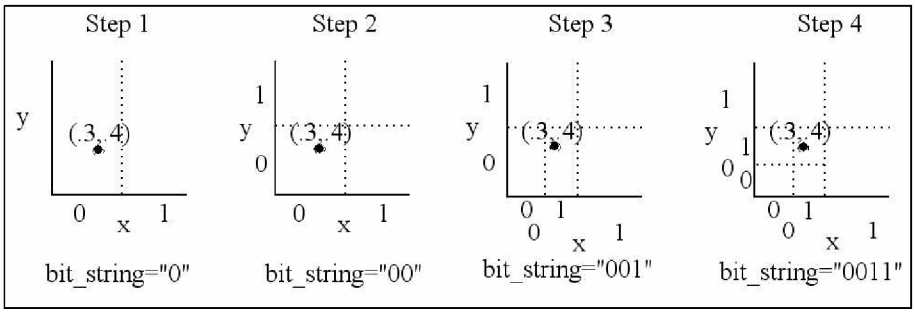
U tohoto přístupu je prostorová složka aproximována použitím Z-hodnoty a dále je vytvořen kombinovaný B-strom se Z-hodnotami a časovou dimenzí záznamů. Kombinovaný B-strom obsahuje jako klíče Z-hodnoty a časovou informaci.

Výpočet Z-hodnoty

Prostor lze rekurzivně dělit na dvě části s dělicí čarou rovnoběžnou s osou *x* nebo s osou *y*. Po každém dělení přímkou rovnoběžkou s osou *x* nebo *y* se zaznamená bitová hodnota (*0/1*):

- Pokud bod leží v levé nebo spodní polovině od dělící přímky, je zaznamenána hodnota *0*
- Pokud bod leží v pravé nebo horní polovině od dělící přímky, je zaznamenána hodnota *1*

Stávající Z-hodnota je bitově posunuta doleva a pomocí bitového *or* s novou zaznamenanou hodnotou vytvořena nová Z-hodnota. Konečná Z-hodnota je následně převedena na číslo v číselné soustavě 10. Na obrázku 5.3 můžeme vidět postup výpočtu Z-hodnoty.



obr. 5.3: Výpočet Z-hodnoty [2]

5.4.3.3 R-strom (prostorová složka) + B-strom (časová složka)

Další možnost pro indexování časoprostorových dat, je vytvořit index R-strom nad prostorovými daty a B-strom nad časovými daty (sloupce, které určují platnost geometrických dat). Poté při dotazu se pro vyhledání geometrie použije R-strom a pro vyhledání správného časového údaje B-strom.

6 Příprava pro experimenty

Pro realizaci experimentů je nutné získat odpovídající časoprostorová data a navrhnout způsob jejich uložení. Proto se tato kapitola zabývá především generováním dat za využití generátoru "Network-Based Generator of Moving Object" a návrhem jednotlivých struktur databází pro uložení dat. Jelikož databáze jsou implementovány pomocí typů objektů a tabulek objektů, popis vytváření typů objektů a tabulek objektů je uveden v kapitole 6.3 Objekty v SQL.

6.1 Použité nástroje

Při přípravě dat a realizaci samotných experimentů jsem využila následujících nástrojů:

Generátor Network-Based Generator of Moving Object

Jedná se o volně dostupný generátor pohybujících se objektů po dané síti. Je blíže popsán v kapitole 6.2 Generování dat.

JDeveloper

JDeveloper obsahuje grafický nástroj vhodný pro vývoj databáze. Obsahuje přehledný seznam tabulek, funkcí, procedur, indexů a dalších objektů databáze. Umožňuje uživateli provádět skripty, procedury a SQL dotazy, jejichž výsledek se zobrazí v podokně "Query result". Nejdůležitější funkcí, kvůli které jsem tento nástroj použila, je možnost volby "Explain Plan". Tato volba nám zobrazí plán vykonávání SQL dotazu. Zobrazuje strukturu (postup), jak bude SQL dotaz vyhodnocen, které tabulky a indexy budou použity. Dle tohoto plánu můžeme zjistit, zda jsou používány vytvořené indexy, nebo zda optimalizátor vyhodnotil jako výhodnější řešení projití celé tabulky. SQL Developer neobsahuje možnost pro získání času potřebného na vykonání dotazu.

SQL*Plus

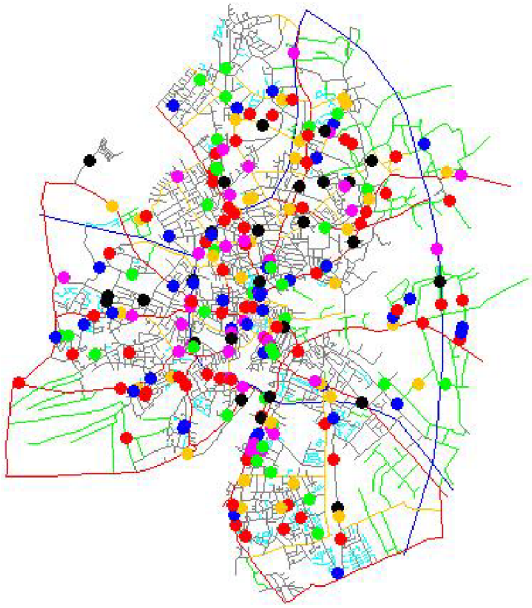
SQL*Plus je volně dostupná konzole pro zadávání příkazů, skriptů do připojené databáze. Lze zde nastavovat mnoho parametrů pro zobrazení výstupních dat, ale i možnost (timing) pro získání času stráveného vykonáním dotazu.

PHP

PHP je skriptovací jazyk prováděný na straně serveru, který je vhodný pro rychlou tvorbu skriptů. Proto jsem ho využila k převodu souboru s výstupní množinou dat získanou z generátoru "Network-Based Generator of Moving Object" do souboru ve formátu procedur pro vložení dat.

6.2 Generování dat

V návaznosti na diplomovou práci [13], kde byl použit generátor „Network-Based Generator of Moving Object", jsem zvolila tentýž generátor časoprostorových dat. Jedná se o volně dostupný generátor z [16] napsaný v jazyce Java. Informace o tomto generátoru jsou získané z [16]. Jak již z názvu napovídá, generuje pohybující se objekty po určité síti. Tedy pohyb těchto objektů je omezen pouze na danou síť, např. síť silnic. Síť lze generovat synteticky nebo lze zvolit již existující síť, např. plány měst. Jelikož generování sítě by bylo příliš náročné a použitím již existující sítě se přiblížíme reálné situaci, zvolila jsem již existující síť (plán města Oldenburg). Grafický výstup generátoru je zobrazen na obr. 6.1.



obr. 6.1: Grafický výstup generátoru “Network-Based Generator of Moving Object“

Nastavení generátoru se provádí v konfiguračním souboru, kde se nastavuje především vzhled, cesta k souboru s daty sítě, výstupní soubor a další, nebo přímo v programu. V programu se nastavuje časová perioda (počet jednotek času, za kterých se objekty pohybují po síti), počet počátečních objektů a počet objektů, které nově vzniknou za jednotku času.

Jak již bylo zmíněno, objekty se mohou pohybovat v síti po danou periodu T (v programu označeno jako “maximum time“). Pohyb objektů je diskrétní. Změna pozice objektů je vždy generována diskrétně - vždy za jednotku času. Proto i do výstupního souboru jsou zaznamenány veškeré objekty v každé jednotce času. Ukládání objektů do souboru nezávisí na rychlosti daného objektu.

Při generování objektů lze využít 2 způsoby generování:

- Přístup orientovaný na datový prostor (DSO)
- Přístup založený na síti (NB)

Přístup orientovaný na datový prostor (DSO) generuje souřadnice objektu (x, y) na základě uniformního rozložení pravděpodobnosti v rámci celého prostoru sítě. Jelikož souřadnice objektu mohou být generovány mimo síť, je dále vypočítáván nejbližší uzel sítě od vygenerovaných souřadnic. Síť se skládá, mimo jiné, z jednorozměrného pole uzlů. Tento uzel se stává konečnou pozicí nově vygenerovaného objektu. Nevýhodou tohoto přístupu je kumulace objektů v prostorách s menší hustotou sítě.

Přístup založený na síti (NB) generuje číslo opět na základě uniformního rozložení pravděpodobnosti. Toto číslo je dále použito jako index do pole uzlů dané sítě. V tomto případě, každý uzel sítě se stane se stejnou pravděpodobností startovacím bodem objektu.

Vygenerované časoprostorové objekty lze uložit do souboru textového nebo binárního dle zadané přípony výstupního souboru. Každý řádek v souboru obsahuje údaje, které jsou odděleny tabulátorem:

- **type**: “newpoint/point/disappearpoint“
    - “newpoint“ - první bod nové trajektorie pohybujícího se objektu
    - “point“ – další bod trajektorie pohybujícího se objektu
    - “disappearpoint“ - poslední bod trajektorie pohybujícího se objektu

- **id**: identifikátor trajektorie
- **sn**: číslo sekvence – pořadí záznamu v rámci jedné trajektorie objektu
- **io**: identifikátor třídy objektu – pod třídou objektu si můžeme představit objekty se stejnými vlastnostmi např. auto
- **ts**: časové razítko zaznamenání objektu
- **x**: x-ová souřadnice objektu
- **y**: y-ová souřadnice objektu
- **v**: aktuální rychlost objektu
- **xu**: x-ová souřadnice bodu, kterým objekt projde
- **yu**: y-ová souřadnice bodu, kterým objekt projde

Příklad výstupní soubor:

type	id	sn	io	ts	x	y	v	xu	yu
newpoint	3	1	5	0	5255.0	19436.0	32	5324	19389
newpoint	4	1	1	0	18100.0	17157.0	132	17965	17263
point	4	2	1	1	17996.17929882	17238.518476478	132	17965	17263
point	3	2	5	1	5281.05996869	19418.249006833	32	5324	19389

6.3 Objekty v SQL

Databáze Oracle 11g podporuje řadu rysů zavedených standardy SQL:1999 a pozdějšími. Týká se to zejména objektových rozšíření v podobě strukturovaných uživatelem definovaných typů (u Oracle od verze Oracle 8i). V terminologii Oracle jsou označovány jako typy objektů nebo též objektové typy (object type). Ty lze potom používat v PL/SQL k objektově-orientovanému programování. Lze je chápat jako třídy v jazycích Java, C++. Objektové typy lze v databázi použít jednak k definici typu sloupce relační tabulky, jednak k vytvoření tabulky objektů. Sloupce tabulek objektů odpovídají atributům typů objektů. V této kapitole bylo čerpáno z [17].

Typy objektů

Typy objektů se skládají z atributů a metod. Metody představují procedury a funkce. Atributy popisují objekt, jeho vlastnosti. Typy objektů se skládají ze záhlaví a těla. Záhlaví je veřejná deklarace atributů a metod a tělo obsahuje implementaci metod, tedy procedur a funkcí.

Následující ukázky zobrazují vytváření záhlaví a těla typu objektu, ale i náznak vytvoření objektu, který je použit při návrhu struktury databáze.

Ukázka vytvoření záhlaví typu objektu

```
CREATE OR REPLACE TYPE TSegment_typ AS OBJECT (
 -- atributy
id_tsegment NUMBER;
mpoint REF MPoint_typ;
region REF Region_typ;
 -- metody
STATIC PROCEDURE insert_tsegment(mpoint_num NUMBER,..)
);
```

Typ objektu *TSegment_typ* obsahuje atributy *id_tsegment*, *mpoint* a *region*. Poslední dva jmenované se odkazují na objekt *Mpoint_typ* případně *Region_typ*. Tento objekt dále obsahuje metodu, v tomto případě se jedná o statickou proceduru *insert_tsegment* pro vkládání nového segmentu do databáze.

Ukázka vytvoření těla typu objektu

```
CREATE OR REPLACE TYPE BODY TSegment_typ
AS
     STATIC PROCEDURE insert_tsegment(mpoint_num NUMBER,...)
     IS
        -- lokální proměnné
     BEGIN
        -- tělo procedury
     END insert_tsegment;
END;
```

Statická procedura se následně volá: `Tsegment_typ.insert_tsegment( …)`

Tabulky objektů

Tabulky objektů se tvoří z typů objektů a sloupce těchto tabulek odpovídají atributům typů objektů. V následující ukázce kódu je znázorněné vytváření tabulky objektů.

Ukázka vytvoření tabulky objektů

```
CREATE TABLE TSegment_tab OF TSegment_typ (
      PRIMARY KEY (id_tsegment),
      FOREIGN KEY (mpoint) REFERENCES MPoint_tab ON DELETE CASCADE,
      FOREIGN KEY (region) REFERENCES Region_tab ON DELETE CASCADE)
      OBJECT IDENTIFIER IS PRIMARY KEY;  -- OID je primární klíč
```

Tabulky objektů jsou jednoznačně identifikovány pomocí identifikátoru objektu OID přiděleného systémem, který lze přepsat zadaným primárním klíčem (viz příklad výše).

6.4 Struktura databáze

Struktury databází vychází ze struktury databáze uvedené v diplomové práci [13]. Navrhla jsem čtyři způsoby uložení časoprostorových dat. První způsob uchovává čas v tabulce, na kterou se odkazuje tabulka s 2D geometrií. Druhý způsob využívá LRS. V tomto případě, jak již bylo uvedeno v kapitole 5.4.3, je čas uložen spolu s trajektorií objektů. Třetí způsob ukládá čas spolu s geometrií, kde čas je modelován jako nová dimenze, vzniká 3D geometrie. Čtvrtý způsob ukládá pozice objektů do jednoduchých datových typů představujících hodnoty na x-ové a y-ové souřadnici ve 2D prostoru.

Struktura databáze je vytvořena na základě reálné situace, kde prostor může být snímán několika kamerami. Každá kamera snímá právě jeden disjunktní region, kterým může procházet pohybující se objekt. Pohybující se objekt je vždy v diskrétním čase snímán a uložen do databáze. Propojením jednotlivých pozic objektu v rámci časového sledu získáme dráhu objektu neboli trajektorii. Trajektorie může procházet více regiony. Pro tento kamerový systém lze využít data uměle vygenerovaná generátorem "Network-Based Generator of Moving Object". V následujících kapitolách jsou popsány jednotlivé struktury databází. Jednotlivé tabulky v databázích jsou implementovány jako tabulky objektů z typů objektů.

6.4.1 Databáze s 2D geometrií

Struktura databáze s 2D geometrií uchovává čas v samostatné tabulce. V této kapitole je popsán objektově-orientovaný návrh databáze a její implementace.

6.4.1.1 Návrh

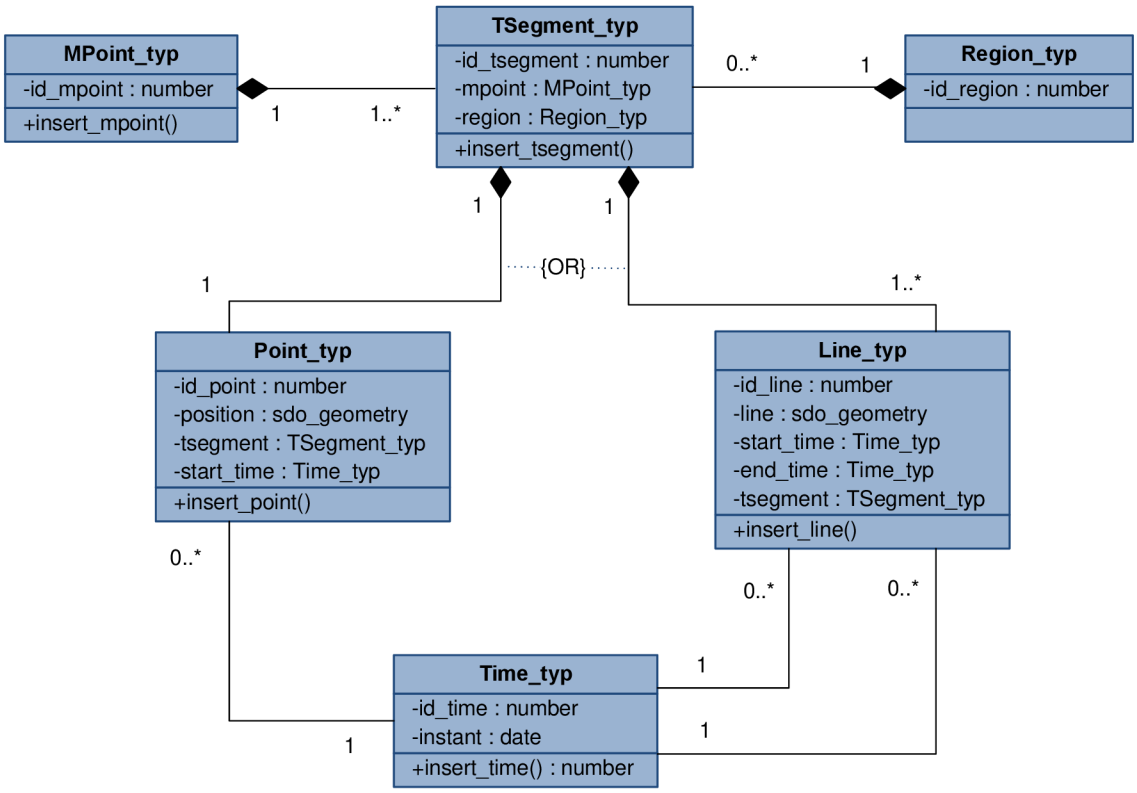
Návrh struktury databáze musí splňovat výše uvedené podmínky a další omezení vzhledem k danému typu uložení dat. Struktura databáze s 2D geometrií uchovává informaci o čase v samostatné tabulce objektů, na kterou se odkazují objekty tabulky uchovávající geometrie. Samotná trajektorie objektu v rámci regionu je uchována v tabulce *TSegment_tab*.

Trajektorie se může skládat ze samostatného bodu. V realitě si pod trajektorií vyjádřenou bodem můžeme představit nový, příchozí objekt v rámci regionu. Pokud se tento objekt začne pohybovat nebo jsou do databáze zasílány veškeré objekty, i nepohybující se, trajektorie se již neskládá z bodu, ale z úseček, které spojují dva body – aktuální pozici a předchozí.

Na obr. 6.2 je znázorněn návrh diagramu typů objektů, ze kterých se vytváří tabulky objektů. Diagram se skládá z typů objektů:

- **Region_typ -** představuje disjunktní region, který je snímán kamerou
- **TSegment_typ -** představuje úplnou trajektorii pohybujícího se objektu v rámci regionu
- **MPoint_typ** – představuje pohybující se objekt
- **Line_typ –** představuje úsečku (část trajektorie) a odkazuje se na čas, ve kterém byly body úsečky zaznamenány
- **Point_typ** – představuje bod trajektorie, pokud tento bod existuje, trajektorie se skládá pouze z tohoto bodu
- **Time_typ** – představuje čas, ve kterém byly objekty zaznamenány

Na základě vztahu kompozice a omezení {OR}, *TSegment_typ* se může skládat buď z jednoho *Point_typ* nebo z více *Line_typ*. *Point_typ* a *Line_typ* nemohou existovat samostatně bez *TSegment_typ*. Vztah kompozice a omezení existence je také mezi typy objektů *MPoint_typ*, *TSegment_typ* a *Region_typ*, *TSegment_typ*, viz obr. 6.2. Hodnota *instant* u *Time_typ* je unikátní.



obr. 6.2: Diagram typů objektů databáze s 2D geometrií

6.4.1.2 Implementace

K implementaci databáze jsem zvolila (z důvodu návaznosti na diplomovou práci [13]) objektově-orientovaný přístup. Vytvářím typy objektů se statickými třídami pro vložení dat. Jelikož generovaná data jsou pouze v rámci jednoho regionu, typ *Region_typ* nemá žádnou metodu.

Přístup zvolený v diplomové práci [13] jsem upravila na stávající podobu, kde vkládám do databáze vždy jeden bod s časovým údajem a identifikátorem objektu. Pokud je vkládán první bod trajektorie (newpoint), je volána metoda *MPoint_typ.insert_mpoint()* pro vložení identifikátoru objektu a *TSegment_typ.insert_tsegment()* pro vytvoření nového segmentu a vložení prvního bodu do tabulky *Point_tab*.

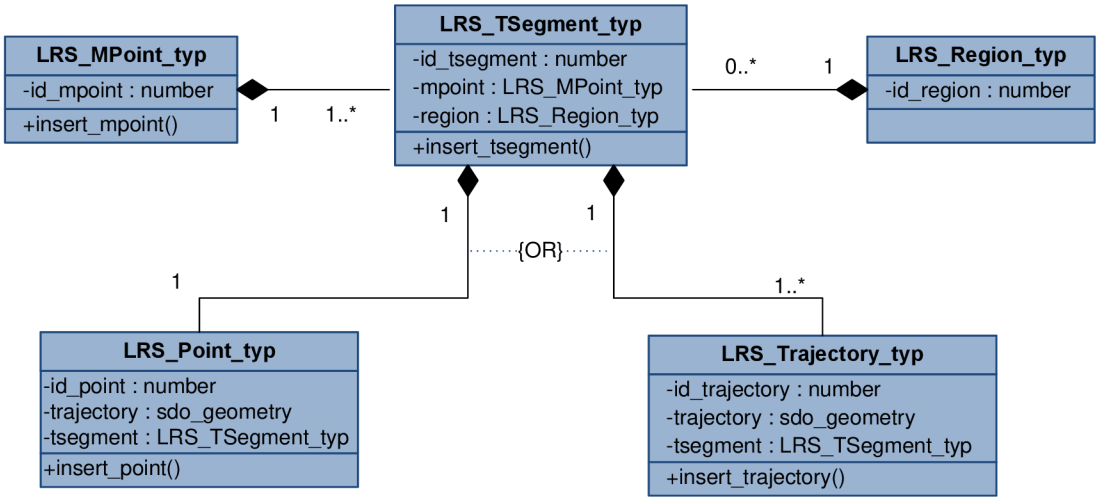
Pokud je vkládán další bod trajektorie (point, dissappearpoint), je volána metoda *Line_typ.insert_line()*. Tato metoda získá poslední bod trajektorie, z tohoto a nového bodu vytvoří novou geometrii (úsečku). Pokud se poslední bod trajektorie nachází v tabulce *Point_tab*, záznam z tabulky *Point_tab* je smazán. Tabulka *Time_tab* vytvořena z *Time_typ* obsahuje unikátní sloupec *instant*, kde je uložen čas. Čas se vždy ukládá s nově ukládaným bodem nebo úsečkou, pokud již v tabulce není uložen.

6.4.2 Databáze využívající LRS

V této kapitole je popsán návrh a implementace struktury databáze využívající LRS, která využívá LRS pro uchování prostorové i časové složky v geometrii.

6.4.2.1 Návrh

Struktura databáze s využitím LRS je podobná struktuře databáze s 2D geometrií. Liší se způsobem uložení informace o čase. V tomto případě je čas uložen v rámci geometrie v *LRS_Trajectory_tab* nebo v *LRS_Point_tab*. Návrh diagramu typů objektů je na obr. 6.3, z těchto typů se vytváří tabulky objektů. *LRS_Trajectory_tab* uchovává maximální možnou trajektorii objektu, kterou lze uložit. Jelikož součást `sdo_geometry`: `sdo_ordinates` (souřadnice bodů, které vytváří příslušnou geometrii) je definováno jako pole čísel, které má omezenou velikost, *LRS_Trajectory_tab* nemusí obsahovat celou trajektorii objektu v rámci regionu. Celou trajektorii opět uchovává *TSegment_tab*.



obr. 6.3: Diagram typů objektů databáze využívající LRS

6.4.2.2 Implementace

Implementace databáze je podobná implementaci databáze s 2D geometrií, založena na typech objektů se statickými přetíženými funkcemi. Lze vkládat trajektorie po jednotlivých bodech, což je časově velice náročné při velkém objemu dat, nebo lze vkládat celé trajektorie nebo body. Druhá možnost se využije především při inicializačním nahrávání. Vkládání je v tomto případě velice rychlé. Při vkládání souřadnic objektů je nutné zohlednit omezenou velikost pole `SDO_ORDINATE_ARRAY`. Pokud počet souřadnic bodů v trajektorii přesáhne limit tohoto pole, je vytvořena nová trajektorie.

6.4.3 Databáze s 3D geometrií

Tato struktura databáze uchovává časovou složku spolu s prostorovou a tím pádem vytváří tří-dimenzionální prostor pro pohybující se objekty.

6.4.3.1 Návrh

Diagram typů objektů je podobný diagramu pro databázi využívající LRS viz obr. 6.3. Rozdíl mezi databází s 3D geometrií a využívající LRS je v uchovávání jednotlivých úseček. Databáze využívající LRS uchovává (maximálně možnou) část trajektorie v *LRS_Trajectory_tab*. Kdežto tabulka *Line_tab3D* odpovídající dle diagramu *LRS_Trajectory_tab* uchovává pouze úsečky. Celá trajektorie je v tabulce *TSegment_tab3D*.

6.4.3.2 Implementace

Implementace databáze je obdobná implementaci databáze s 2D geometrií. Pokud se vkládá nový objekt do databáze, je vložen do geometrie spolu s prostorovými souřadnicemi i čas (do tabulky *Point_tab3D*). Pokud se vkládají nové informace o již uloženém objektu, informace uložené v *Point_tab3D* jsou smazány a vloženy spolu s novými informacemi do *Line_tab3D* jako 3D úsečka.

6.4.4 Databáze bez geometrie

Struktura databáze bez geometrie nevyužívá datový typ `SDO_GEOMETRY` pro uložení geometrie. Souřadnice jsou uloženy v jednoduchých datových typech.

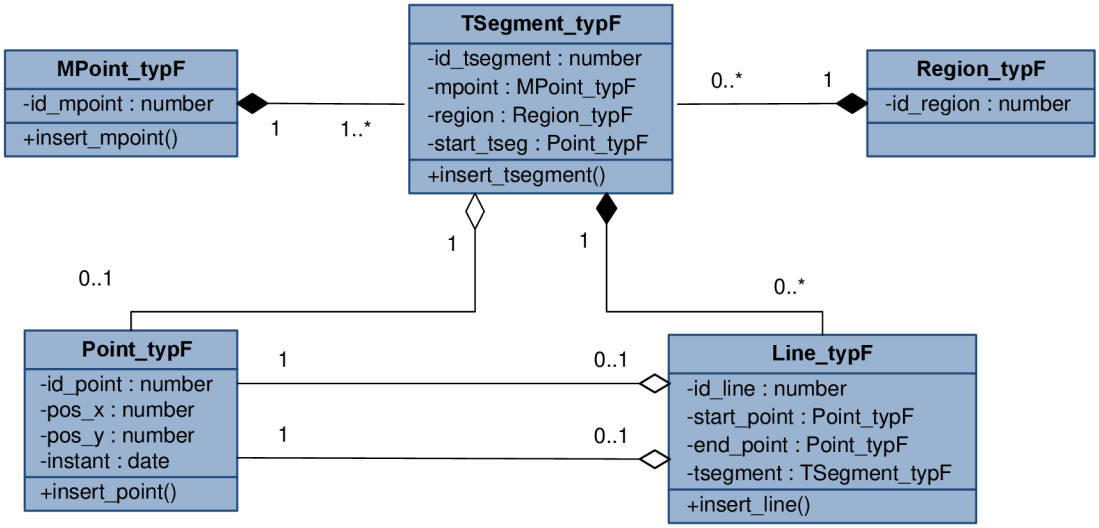
6.4.4.1 Návrh

Návrh databáze je podobný návrhu databáze s 2D geometrií, nicméně jsou zde menší rozdíly. Segment trajektorie uchovává počáteční bod trajektorie i jednotlivé úsečky. Úsečky jsou uchovávány v tabulce *Line_tabF*, odkud existují odkazy na tabulku *Point_tabF*. V tabulce *Point_tabF* jsou uloženy jednotlivé koncové body úseček a čas, kdy se objekty v těchto bodech nacházely. Diagram typů objektů je uveden na obr. 6.4.

6.4.4.2 Implementace

Struktura této databáze je implementována tak, že při přidání nového objektu se volá metoda *MPoint_typF.insert_mpoint()*, která uloží identifikátor trajektorie, a metoda *TSegment_typF.insert_tsegment()*, která uloží nový segment a volá metodu *Point_typF.insert_point()* pro uložení nového bodu. Při přidání dalšího bodu již existujícímu objektu (změně pozici objektu) se volá metoda *Line_typF.insert_line()*, která volá metodu *Point_typF.insert_point()* pro vložení nového

bodu a zároveň uloží nový záznam do tabulky *Line_tabF* s referencí na poslední bod trajektorie a nově uložený.



obr. 6.4: Diagram typů objektů databáze bez geometrie

6.5 Postup příprav

Pro provedení experimentů je nejdříve nutné implementovat výše uvedené struktury databází s příslušnými funkcemi pro vkládání dat. Data je nutné vygenerovat. Pro generování dat byl použit generátor „Network-Based Generator of Moving Object“. Parametry pro generátor byly nastaveny tak, aby se generovala data s 2016 časovými jednotkami, 60 objekty na počátku generování a 7 objekty, které nově vzniknou za časovou jednotku.

Další fází je převedení vygenerovaných dat pomocí PHP skriptu na soubor volání metod pro vložení. Vložená data jsou v časovém rozmezí jednoho týdne (1.1.2010 00:00:00 – 8.1.2010 00:00:00), kde rozdíl mezi dvěma po sobě jdoucími časovými jednotkami je 5 minut. U databáze s 3D geometrií a databáze využívající LRS je čas vyjádřen v minutách s počáteční hodnotou 0, která odpovídá času 1.1.2010 00:00:00. Tedy čas 15 v databázi s 3D geometrií nebo v databázi využívající LRS odpovídá času 1.1.2010 00:15:00 v databázi s 2D geometrií (nebo bez geometrie).

Pokud vkládáme nový bod (počáteční bod trajektorie, v původních datech z generátoru označeno jako newpoint), jsou volány následující metody, které jsou vygenerovány PHP skriptem:

```
execute MPoint_typ.insert_mpoint();
execute TSegment_typ.insert_tsegment(1, 1, 12649, 18013,
        '1.1.2010 00:00:00');
```

První metoda vkládá *id* nové trajektorie. Druhá metoda vkládá nový segment spolu s uložením příslušného bodu. Pokud vkládáme další bod již existující trajektorie (v původních datech z generátoru označeno jako point nebo disappearpoint), volá se následující metoda vygenerována PHP skriptem:

```
execute Line_typ.insert_line(48, 1, 4142, 22280, '1.1.2010 00:05:00');
```

Metoda uloží příslušnou geometrii (úsečku v databázích s 2D, 3D geometrii, trajektorii u databáze s LRS při inicializačním nahrávání).

Poslední fází je samotné nahrání dat do databáze. U většiny struktur databáze bylo nahrávání časově náročné. Pouze u inicializačního nahrávání dat do databáze, která využívá LRS, bylo nahrávání velice rychlé, nicméně toto zrychlení vykompenzovalo velké zpomalení při převodu dat PHP skriptem na soubor s voláním metod.

7 Experimenty

V této kapitole jsou uvedeny návrhy pro experimenty, popsána jejich realizace, výsledky a dále možnosti, které nám skýtají jednotlivé struktury databáze a indexační přístupy. V kapitole 7.2 Úvodní statistiky je uvedena časová a prostorová náročnost jednotlivých struktur databází a indexů.

7.1 Vlastnosti pro testování

Navržené experimenty zkoumají možnosti indexačních struktur, jejich výkonnost a vhodnost použití. Nicméně při volbě dané indexační struktury jsme značně omezeni způsobem uložení dat v databázi. Nelze využít veškeré přístupy pro indexování na všechny navržené struktury databází. Stejně tak nelze využít veškeré prostorové operátory, což je v případě struktury s 3D geometrií. U 3D R-stromu, který je použit k indexování 3D geometrie, lze použít pouze operátory: `SDO_FILTER`, `SDO_ANYINTERACT`, `SDO_INSIDE`, `SDO_NN`, `SDO_WITHIN_DISTANCE` [4].

Navržené experimenty lze rozdělit do několika kategorií dle způsobu uložení dat, typu indexování a typu dotazu. Lze předpokládat, že dotazy, které vyhledávají větší množství dat, trvají delší dobu. Proto experimenty, které zkoumají časovou závislost, budou vyjádřeny jako funkce závislosti času potřebného pro vyhledání dat na velikosti vyhledaných dat (počtu vrácených geometrií - úseček). Jednotlivé kategorie pro experimenty jsou:

Způsob uložení dat

- Databáze s 2D geometrií
- Databáze využívající LRS
- Databáze s 3D geometrií
- Databáze bez geometrie

Typ indexování

- R-strom (pro prostorová data)
- B-strom (pro časová data)
- Quad-strom (pro prostorová data): fixní indexování
- Quad-strom (pro prostorová data): hybridní indexování
- 3D R-strom

Typ použitého prostorového operátoru

- `SDO_RELATE`
- `SDO_FILTER`
- `SDO_NN`, `SDO_NN_DISTANCE`

Dva možné způsoby, kterými lze zkoumat výkonnost indexů a operátorů, jsou dotazy, které buď vrací vždy stejný počet geometrií, nebo které se zaměřují vždy na stejně velkou část regionu pro porovnání. Způsob dotazu, který jsem zvolila pro experimenty, vrací vždy stejný počet geometrií (úseček). Experimenty tedy zkoumají závislost času na procentuálním vyjádření vyhledaných, podobných úsečkách (části trajektorií). Pokud bych zvolila druhý způsob, kde bych zkoumala závislost času na procentuálním poměru prohledávané plochy, výsledky by byly zkreslené. Při zkušebních testech se diametrálně lišily časy, výkony dotazů, které vyhledávají trajektorie na různých částech regionu (města Oldenburg), ale které mají stejný obsah plochy. Je to z důvodu různé hustoty trajektorií vzhledem k celému regionu.

U většiny experimentů je prováděno 5 typů dotazů. Tyto typy dotazů se liší počtem vrácených geometrií. Tedy první typ dotazu vrátí 1000 geometrií, druhý typ 6000, třetí typ 12000, čtvrtý typ 18000 a pátý typ 24000. Každý typ dotazu obsahuje 10 variant dotazů. Varianty dotazů v rámci jednoho typu dotazu jsou rozdílné v geometrii (umístění a velikost plochy obdélníků), která je srovnávána s uloženými trajektoriemi. Každá varianta dotazu se provede 8x. Z výsledků v rámci jedné varianty dotazu je vypočítán průměr. Ze všech 10 průměrů v rámci typu dotazu je následně vypočítán průměr a směrodatná odchylka[2]. Pokud se u některého experimentu bude lišit průběh provádění, bude to uvedeno u jeho návrhu.

Jak již bylo zmíněno, každý typ dotazu se skládá z 10 různých dotazů, které se liší velikostí a umístěním geometrie pro porovnávání. Tato geometrie je náhodně generována tak, aby na daný dotaz vrátila vždy požadovaný počet geometrií, které s generovanou geometrií interagují.

7.2 Úvodní statistiky

V této kapitole jsou uvedeny souhrnné informace o době potřebné pro vytvoření indexů a informace o velikosti všech tabulek (dle typu databáze) a indexů použitých pro experimenty. Časoprostorová data se uchovávají ve čtyřech typech databází: databáze s 2D geometrií, databáze využívající LRS, databáze s 3D geometrií a databáze bez geometrie.

Nad databází s 2D geometrií lze vytvořit R-strom a Quad-strom nad prostorovou složkou a B-strom nad časovou složkou. Nad databází využívající LRS lze vytvořit pouze R-strom nad prostorovou složkou. Nad databází s 3D geometrií lze vytvořit pouze 3D R-strom (Quad-strom stejně jako u databáze s LRS nelze vytvořit nad 3D daty). Databáze bez geometrie neobsahuje žádný sloupec typu `SDO_GEOMETRY`, proto R-strom a Quad-strom se nevytváří. Databáze obsahuje pouze časovou složku, nad kterou lze vytvořit B-strom. Níže jsou uvedeny příklady pro vytvoření jednotlivých indexů.

Příklad vytvoření indexační struktury R-strom nad sloupcem *Line_tab.line*:

```
CREATE INDEX Line_idx ON Line_tab(line) INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Příklad vytvoření indexační struktury Quad-strom (fixní indexování) nad sloupcem *Line_tab.line*:

```
CREATE INDEX Line_idx ON Line_tab(line) INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('SDO_LEVEL=8');
```

Příklad vytvoření indexační struktury Quad-strom (hybridní indexování) nad sloupcem *Line_tab.line*:

```
CREATE INDEX Line_idx ON Line_tab(line) INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('SDO_LEVEL=4 NUMTILES=6');
```

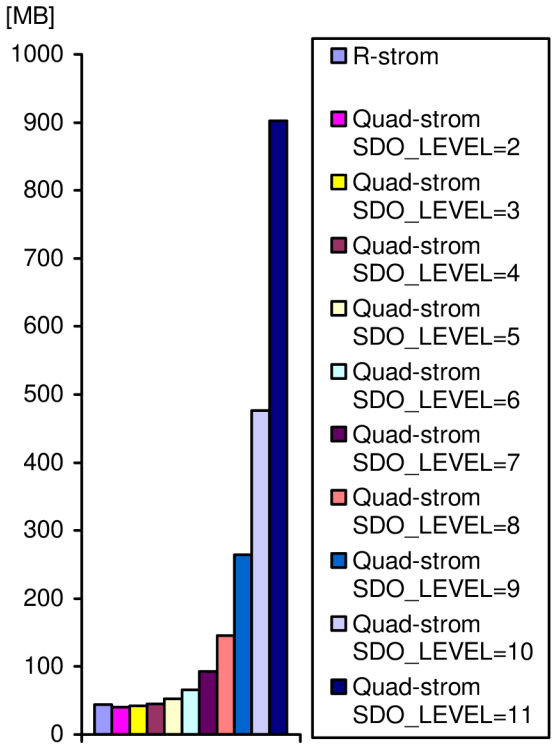
Příklad vytvoření indexační struktury 3D R-strom nad sloupcem *Line_tab.line*:

```
CREATE INDEX Line_idx ON Line_tab(line) INDEXTYPE IS MDSYS.SPATIAL_INDEX
PARAMETERS('sdo_indx_dims=3');
```

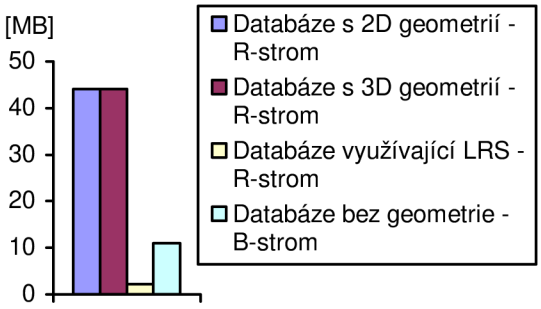
Jak lze vidět z grafu 7.2 prostorově nejnáročnější jsou indexační struktury nad 2D nebo 3D geometriemi. V [4] je uvedeno, že indexační struktura R-strom požaduje minimálně `100*n B` volného místa pro samotné uložení indexu (kde `n` je počet záznamů s geometrií), pokud počet

[2] Směrodatná odchylka: míra proměnlivosti, rozptýlení, nízká hodnota znamená, že hodnoty se nachází kolem průměru, vysoká odchylka značí, že data jsou více rozptýlena

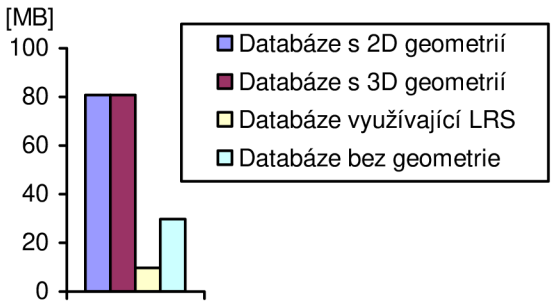
záznamů přesahuje milion. Jelikož databáze s 2D geometrií obsahuje stejný počet řádků s geometrií jako databáze s 3D geometrií, velikost výsledných indexů je téměř stejná. Databáze obsahují 487392 řádků s geometriemi. Tedy teoreticky je potřeba 46,48 MB volného místa pro R-strom. Reálně bylo potřeba 44,13 MB volného místa.



Graf 7.1: Velikost indexačních struktur nad databází s 2D geometrií



Graf 7.2: Porovnání velikosti indexačních struktur u jednotlivých databází



Graf 7.3: Velikosti jednotlivých databází

Objekt	Velikost[MB]
Databáze s 2D geometrií	80,75
Databáze s 3D geometrií	80,69
Databáze využívající LRS	9,69
Databáze bez geometrie	29,69

Tabulka 7.1: Velikosti jednotlivých databází

V grafu 7.3 a tabulce 7.1 je uvedeno potřebné místo pro jednotlivé struktury databází. Jak lze vidět prostorově nejnáročnější jsou databáze s 2D a 3D geometrií.

Prostorovou náročnost jednotlivých indexačních struktur, které lze vytvořit nad databází s 2D geometrií, můžeme vidět na grafu 7.1 Jak je patrné, prostorová náročnost Quad-strom indexu při zvětšování parametru `SDO_LEVEL` roste exponenciálně. Kompletní tabulky ke grafům jsou uvedeny v Příloze A, kde jsou i údaje a grafy s časovou náročností při vytváření jednotlivých indexů.

7.3 Optimalizátor

Zpracování SQL dotazu používá následující komponenty k provedení SQL dotazu:

- Parser: kontroluje syntax a sémantiku dotazu
- Optimalizátor: hledá optimální plán, jak vykonat SQL dotaz, používá metody pro ohodnocení nebo interní pravidla
- "Row Source Generator": na vstupu dostane optimální plán z optimalizátoru a generuje plán pro vykonání SQL dotazu
- Provádění SQL: realizuje plán pro vykonání SQL dotazu

Komponenta, která ovlivňuje výkonnost dotazu (jeho dobu provádění), je optimalizátor. Určuje nejefektivnější způsob, jak vykonat SQL dotaz s ohledem na faktory spojené s odkazovanými objekty a podmínkami uvedenými v SQL dotazu. Vykonání SQL dotazu lze ovlivnit použitím tzv. "hints", což je nápověda pro optimalizátor pro vytvoření požadovaného plánu vykonávání na základě jednotlivých typů "hints":

- Přístup k datům
- Transformace dotazu
- Operace join
- Paralelní vykonávání

Plán pro vykonání dotazu můžeme zkoumat za použití příkazu `EXPLAIN PLAN FOR`. V JDeveloper pro tuto možnost existuje tlačítko "Explain plan". V tomto plánu lze také ověřit, zda se používají vytvořené indexy. Ukázka struktury plánu pro vykonání je na obr. 7.1. V následujícím textu jsou popsány jednotlivé typy "hints", které jsou zajímavé pro následné vysvětlení problému. V této kapitole jsem čerpala z [18].

Přístup k datům

Jedná se o skupinu "hints" popisující, jak jsou data načtena z databáze. Příkladem může být:

Full table scans

Jedná se o typ skenování, který prochází veškeré řádky tabulky a filtruje ty, které odpovídají daným kritériím pro výběr. Tento přístup je rychlejší pro výběr velkého množství dat. Použití: /*+ *FULL(table_name)* */

Index scans

Index scans načte řádek pomocí indexovaného sloupce. Při provádění tohoto způsobu vykonávání Oracle hledá index pro hodnotu indexovaného sloupce. Pokud se dotazuje pouze na hodnotu uloženou přímo v indexu, Oracle přečte indexovanou hodnotu přímo z indexu. Index obsahuje hodnotu sloupce a identifikátory řádků v tabulce s indexovaným sloupcem. Použití: /*+ *INDEX(table_name index)* */

Transformace dotazu

Jedná se o skupinu příkazů, které převedou SQL dotaz na jiný dotaz, který vyhledá stejná data, ale jeho struktura bude odlišná. Tento typ příkazů je nutné provádět pouze tehdy, pokud to zvýší výkonnost dotazu.

Concatenation

Tento typ transformace převede `or` podmínky ve `where` klauzuli na složený dotaz použitím operátoru `UNION ALL`. Použití: /*+ *USE_CONCAT* */

Operace join

Join (dále spojení) dvou tabulek se děje na základě charakteristiky v klauzuli `from` a definovaných vztahů v klauzuli `where`. Jako příklady spojení jsou:

Nested loop joins

Tento typ spojení se použije, pokud je potřeba připojit malou množinu dat jedné tabulky k druhé tabulce. Skládají se ze dvou cyklů:

```
NESTED LOOPS
      outer_loop
      inner_loop
```

Spojuje řádky dvou tabulek. Tabulka, která je optimalizátorem určena jako řídící, je označena jako vnější tabulka. Druhá tabulka je označena jako vnitřní tabulka. Tento způsob připojení probíhá následovně: U každého řádku ve vnější tabulce Oracle přistoupí ke všem řádkům ve vnitřní tabulce. `Outer_loop` odpovídá vnější tabulce a `inner_loop` odpovídá vnitřní tabulce. Je důležité zajistit, aby vnitřní tabulka byla závislá na vnější tabulce. Pokud je vnitřní tabulka nezávislá na vnější, stejné řádky vnitřní tabulky jsou vyvolány při každé iteraci, což znatelně snižuje výkon. V tomto případě metoda Hash join spojí dvě nezávislé tabulky rychleji. Použití: */*+ USE_NL(table_names) */*

Hash join

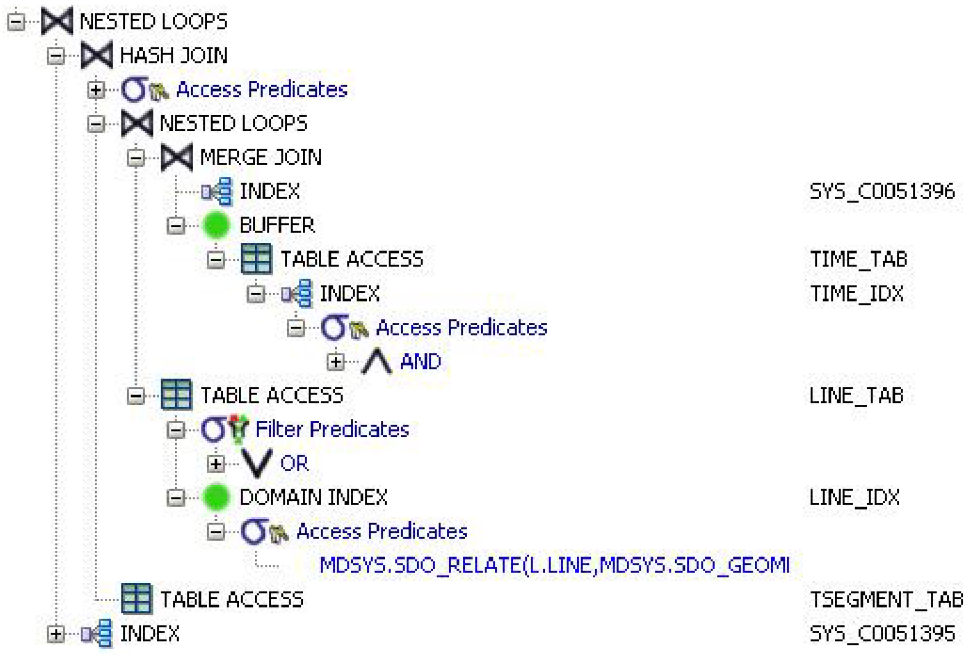
Tento typ spojení se používá při spojování velkých souborů dat. Optimalizátor použije menší ze dvou tabulek pro vytvoření hašovací tabulky. Poté je procházena větší tabulka a zkoumána hašovací tabulka k nalezení řádků k připojení. Použití: */*+ USE_HASH(table_names) */*

Při testování dotazů jsem narazila na neobvyklou časovou náročnost dotazu, který vyhledává v databázi s 2D geometrií celkový počet geometrií (úseček), které interagují s geometrií *geom* a alespoň jeden koncový bod geometrie (úsečky) je v časovém intervalu <3.1.2010 6:00:00; 3.1.2010 19:00:00>. Dotaz je následující:

```
SELECT count (l.id_line)
     FROM time_tab i, tsegment_tab t, line_tab l, mpoint_tab mp,
          region_tab r
     WHERE sdo_relate(l.line, geom),
               'mask=anyinteract querytype=window') = 'TRUE'
     AND l.tsegment=REF(t) AND (l.start_time=REF(i) OR
      l.end_time=REF(i)) AND t.mpoint=REF(mp) AND
       i.instant BETWEEN '3.1.2010 6:00:00' AND '3.1.2010 19:00:00';
```

Stejný dotaz, který vyhledává geometrie (úsečky), jejichž (pouze) počáteční bod je v daném časovém intervalu, se provedl 144x rychleji. Pro ilustraci, výše uvedený dotaz (vyhledávající na základě počátečního i koncového času) se provedl za 64,93 s. Dotaz, který vyhledával geometrie dle počátečního času, se provedl za 0,45 s (nalezl přesně polovinu geometrií co výše uvedený dotaz). Jelikož rozdíl mezi dotazy je v přidání predikátu `OR` a možnosti vyhledání geometrií dle koncového času, předpokládala bych, že výsledná doba provádění bude přibližně dvojnásobná.

Na obrázku 7.1 lze vidět plán pro vykonávání SQL dotazu nad geometriemi dle počátečního i koncového času. Markantní zpomalení vykonávání dotazu, je především díky druhému spojení metodou nested loops. Provádění tohoto procesu je založeno na přístupu k tabulce *line_tab* pro každý řádek v tabulce *time_tab*, což při zvážení velikosti tabulky *line_tab* a náročnosti operace `SDO_RELATE` je značně neefektivní.



Obr. 7.1: Explain plan pro SQL dotaz

Řešením chybně vyhodnoceného dotazu (vytvořeného plánu provádění) je použití "hints" (nápovědy pro vytváření plánu provádění) v dotazu. Na základě zadaných "hints" se vytvoří požadovaný plán provádění. Výše uvedenému příkazu byl přidán příkaz ("hints") ovlivňující strukturu plánu provádění:

```
SELECT /*+  USE_CONCAT  USE_HASH(t) */ count (l.id_line)
    FROM time_tab i, tsegment_tab t, ..
```

Plán vykonávání tohoto příkazu je uveden v Příloze B. Po této úpravě je dotaz proveden přibližně za 0,85 s, což je optimální doba (dotaz vyhledávající pouze dle počátečního času se provede přibližně za polovinu doby).

7.4 Databáze s 2D geometrií

V této kapitole jsou uvedeny experimenty nad časoprostorovými daty, která jsou uložena v databázi s 2D geometrií. Jelikož nad dvou-dimenzionální geometrií lze vytvářet R-strom i Quad-strom, je zde uvedeno srovnání těchto dvou indexačních struktur.

7.4.1 Porovnání operátorů SDO_RELATE, SDO_FILTER

Tento experiment, jak již je patrné z názvů, srovnává dva operátory pro porovnávání časoprostorových dat. Zjišťuje, zda geometrie, zadané jako parametry operátorů, spolu nějakým způsobem interagují. K tomuto experimentu je využita indexační struktura R-strom.

7.4.1.1 Návrh

Index R-strom je vytvořen nad sloupcem *line* tabulky *line_tab*. Experiment porovnává dva prostorové operátory `SDO_RELATE` a `SDO_FILTER`. Jelikož `SDO_FILTER` používá pouze primární filtr při

vyhledávání podobných geometrií a SDO_RELATE používá primární i sekundární filtr, lze očekávat, že dotaz s operátorem SDO_FILTER bude rychlejší.

Operátor SDO_FILTER vyhledává geometrie, které nejsou disjunktní. SDO_RELATE vyhledává geometrie na základě zadané masky. Abychom mohli porovnávat operátory SDO_FILTER a SDO_RELATE, musíme SDO_RELATE zadat masku anyinteract. S touto maskou operátor SDO_RELATE vyhledá geometrie, které nejsou disjunktní.

Struktura dotazu s operátorem SDO_FILTER:

```
SELECT count (l.id_line)
FROM   time_tab i, tsegment_tab t, line_tab l, mpoint_tab mp
     WHERE  sdo_filter(l.line, geom) = 'TRUE'
AND l.tsegment=REF(t) AND l.start_time=REF(i) AND t.mpoint=REF(mp) AND
      i.instant BETWEEN '3.1.2010 6:00:00' AND '3.1.2010 19:00:00';
```

Struktura dotazu s operátorem SDO_RELATE:

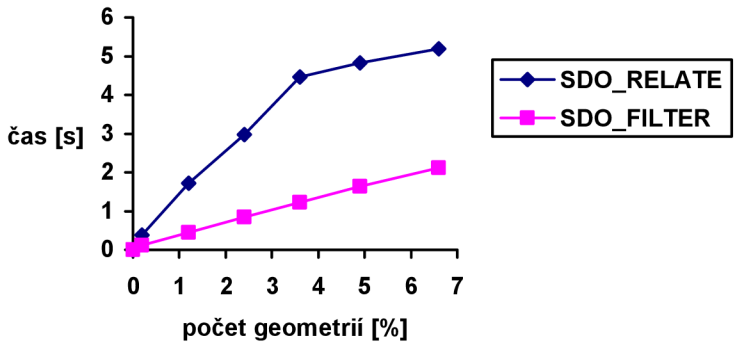
```
SELECT count (l.id_line)
FROM   time_tab i, tsegment_tab t, line_tab l, mpoint_tab mp
WHERE  sdo_relate(l.line, geom, 'mask=anyinteract querytype=window') =
          'TRUE'
AND l.tsegment=REF(t) AND l.start_time=REF(i) AND t.mpoint=REF(mp) AND
 i.instant BETWEEN '3.1.2010 6:00:00' AND '3.1.2010 19:00:00';
```

kde *geom* je porovnávaná geometrie.

7.4.1.2 Výsledky

Jak jsem předpokládala, operátor SDO_FILTER je při vyhledávání podobných geometrií rychlejší než operátor SDO_RELATE. Nicméně v některých případech dotazy s jednotlivými operátory nalezly odlišný počet podobných geometrií. Dotaz s operátorem SDO_FILTER nalezl průměrně o 0,013% geometrií více. Po bližším prozkoumání jsem zjistila, že dotaz s operátorem SDO_FILTER vyhodnotil některé geometrie, které jsou disjunktní, jako interagující. Proto z hlediska rychlosti je výhodnější použít operátor SDO_FILTER, ale z hlediska přesnosti je SDO_RELATE lepší.

V grafu 7.4 je znázorněna závislost času potřebného pro vyhledání na počtu vrácených geometrií, který je vyjádřen procentuálně vůči celkovému počtu geometrií. Tabulka 7.2 udává čas se směrodatnou odchylkou, který byl potřeba pro vyhledání daného počtu geometrií (v závorce je procentuální vyjádření vůči celkovému počtu geometrií) pomocí daných operátorů.



Graf 7.4: Závislost času na počtu vyhledaných geometrií dotazy s SDO_FILTER a SDO_RELATE

operátor / počet geometrií	1000 (0,2 %)	6000 (1,2 %)	12000 (2,4 %)	18000 (3,6 %)	24000 (4,9 %)
SDO_RELATE	(0,38 ± 0,09) s	(1,72 ± 0,34) s	(2,98 ± 0,34) s	(4.46 ± 0,68) s	(4,83 ± 0,44) s
SDO_FILTER	(0,11 ± 0,01) s	(0,44 ± 0,04) s	(0,84 ± 0,30) s	(1,23 ± 0,04) s	(1,64 ± 0,04) s

Tabulka 7.2: Časová náročnost dotazů s operátory SDO_RELATE, SDO_FILTER

7.4.2 Quad-strom: Fixní indexování

V této kapitole je popsán experiment zkoumající vliv parametru SDO_LEVEL na výkon dotazu. Parametr SDO_LEVEL se zadává při vytváření indexu využívajícího indexační strukturu Quad-strom.

7.4.2.1 Návrh

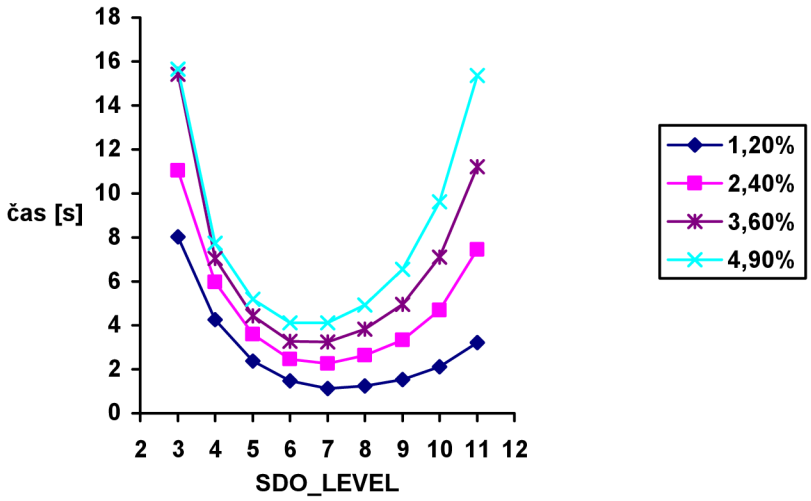
V tomto experimentu se srovnávají různé hodnoty SDO_LEVEL. SDO_LEVEL je parametr udávající stupeň rozčlenění při vytváření Quad-stromu. Při vysoké hodnotě SDO_LEVEL (vysoký stupeň rozčlenění) se bude vytvářet mnoho pokrývajících rozčleňujících ploch, které jsou malé. Poté geometrie může být pokryta např. tisíci rozčleňujícími plochami, což má za následek, že doba zpracování dotazu se prodlužuje.

Pokud jsou plochy příliš velké (hodnota SDO_LEVEL je nízká), primární filtr vybere obrovské množství geometrií, které následně musí zpracovávat sekundární filtr, jehož výkonnost je nižší. Tedy ve výsledku, pokud zvolíme hodnotu SDO_LEVEL příliš nízkou, doba zpracování dotazu se opět prodlouží. Velikost těchto ploch by měla mít vliv na výkon dotazu, proto je vždy nutné zvolit správnou velikost rozčleňujících ploch (SDO_LEVEL).

Tento experiment zkoumá závislost doby zpracování dotazu na velikosti hodnoty SDO_LEVEL vždy pro různý počet vrácených geometrií. Parametr SDO_LEVEL jsem testovala v intervalu celých hodnot <2;11> a nad dotazy, které vrací 6000, 12000, 18000 a 24000 geometrií za použití operátoru SDO_RELATE (struktura dotazu je znázorněna v experimentu 7.4.1 Porovnání operátorů SDO_RELATE a SDO_FILTER). Každý dotaz je v tomto případě opakován 3x. Z těchto opakování je spočítán průměr a ze všech průměrů u jednotlivých typů dotazů je spočítán průměr a směrodatná odchylka.

7.4.2.2 Výsledky

Z grafu 7.5 a tabulky 7.3 lze vyčíst, že časová náročnost dotazu, jak se předpokládalo, je nejvyšší při nízkých a vysokých hodnotách SDO_LEVEL. Veškeré dotazy, při různých hodnotách SDO_LEVEL, vždy vracely správný počet geometrií. Proto lze říci, že pro danou strukturu databáze, je z časového hlediska nejvýhodnější použít Quad-strom s hodnotou SDO_LEVEL 7. V grafu 7.5 je počet vrácených geometrií uveden v procentuálním vyjádření k celkovému počtu geometrií. Popis prostorové náročnosti jednotlivých indexů je uveden v kapitole 7.2 Úvodní statistiky.



Graf 7.5: Závislost času (potřebného pro dotaz) na hodnotách parametru `SDO_LEVEL`

SDO_LEVEL / počet geometrií	6000: 1,2 %	12000: 2,4 %	18000: 3,6 %	24000: 4,9 %
2	(13,56 ± 3,70) s	(18,20 ± 3,33) s	(21,13 ± 1,73) s	(20,31 ± 1,44) s
3	(8,02 ± 0,89) s	(11,05 ± 1,83) s	(12,43 ± 1,24) s	(12,64 ± 1,33) s
4	(4,25 ± 0,54) s	(5,97 ± 1,19) s	(7,03 ± 0,79) s	(7,73 ± 0,49) s
5	(2,38 ± 0,40) s	(3,60 ± 0,61) s	(4.43 ± 0,56) s	(5,20 ± 0,40) s
6	(1,49 ± 0,24) s	(2,47 ± 0,27) s	(3,28 ± 0,33) s	(4,12 ± 0,23) s
7	**(1,14 ± 0,15) s**	**(2,27 ± 0,29) s**	**(3,26 ± 0,48) s**	**(4,11 ± 0,24) s**
8	(1,25 ± 0,18) s	(2,65 ± 0,29) s	(3,84 ± 0,49) s	(4,94 ± 0,20) s
9	(1,54 ± 0,21) s	(3,33 ± 0,29) s	(4,95 ± 0,51) s	(6,56 ± 0,26) s
10	(2,12 ± 0,30) s	(4,69 ± 0,27) s	(7,09 ± 0,59) s	(9,63 ± 0,25) s
11	(3,23 ± 0,49) s	(7,44 ± 0,33) s	(11,21 ± 0,80) s	(15,35 ± 0,43) s

Tabulka 7.3: Časová náročnost dotazu dle hodnot parametru `SDO_LEVEL`

7.4.3 Quad-strom: Hybridní indexování

Hybridní indexování u Quad-stromu znamená, že prostor je rozčleněn jako u fixního indexování na stejně velké plochy. A navíc plochy, které pokrývají geometrii, jsou dále rozčleňovány. V tomto experimentu jsou porovnávány parametry `NUMTILES`, které určují počet ploch použitých na objekt při rozčleňování prostoru.

7.4.3.1 Návrh

Při indexování dat Quad-stromem pomocí hybridního indexování je nutné zadat parametry `SDO_LEVEL` a `NUMTILES`. `SDO_LEVEL` udává stupeň rozčlenění, je popsán blíže u experimentu 7.4.2, `NUMTILES` udává počet rozčleňujících ploch, kterými je pokryta daná geometrie.

Pokud hodnota `NUMTILES` je nízká, výsledná geometrie bude nepřesně aproximována. Naopak při větší hodnotě parametru `NUMTILES`, indexování geometrie bude přesnější a zvýší se selektivita primárního filtru, čímž zvýšíme výkonnost dotazu. Ale při velmi velkém počtu ploch vzniká příliš mnoho položek a tím se výkonnost dotazu snižuje.

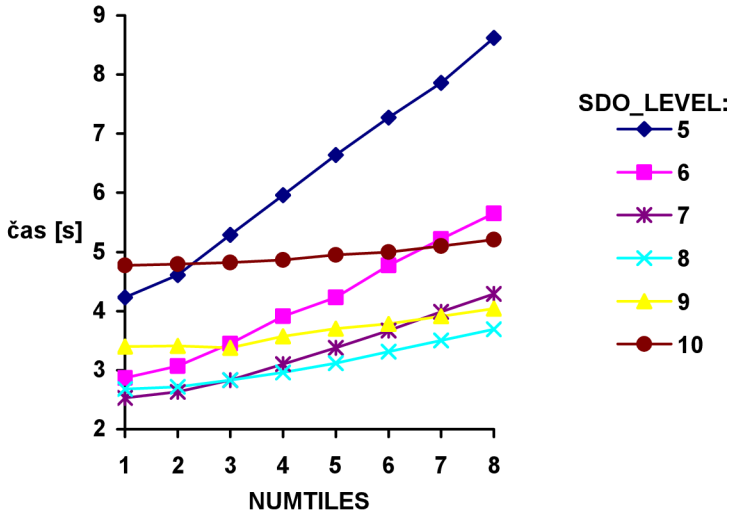
Testování proběhlo pouze nad jedním typem dotazu (dotazy, které vrací 12000 geometrií) s parametrem SDO_LEVEL v intervalu celých hodnot <2;10> a NUMTILES v intervalu celých hodnot <1;8>. Každý dotaz byl opakován 3x. Z těchto výsledků byl spočítán průměr. A pro každou dvojici (SDO_LEVEL, NUMTILES) byl spočítán průměr a směrodatná odchylka (z průměrů nad varianty dotazů).

7.4.3.2 Výsledky

Z návrhu experimentu jsem předpokládala, že časová závislost na hodnotě parametru NUMTILES bude obdobná časové závislosti na hodnotě parametru SDO_LEVEL u experiment 7.4.2. Nicméně, jak lze vyčíst z grafu 7.6 a z tabulky 7.4, s rostoucí hodnotou NUMTILES roste i čas potřebný pro vykonání dotazu dle jednotlivých hodnot parametru SDO_LEVEL.

Z [12] vyplývá, že hybridní indexování není doporučováno. V ojedinělých případech ovšem může přinést zvýšení výkonnosti dotazu oproti použití fixního indexování. Je to především při použití parametru querytype=join[3] u operátoru SDO_RELATE a v případě, že rozdíl optimální hodnoty SDO_LEVEL jednotlivých sloupců je 4 a více. Jelikož porovnávám geometrie z jednoho sloupce s pouze jednou geometrií, namísto parametr querytype=join používám parametr querytype=window. Proto v tomto případě hybridní indexování pomocí Quad-stromu není vhodné.

V grafu 7.6 a v tabulce 7.4 jsou uvedeny výsledky pouze pro SDO_LEVEL v intervalu <5;10> a pro NUMTILES v intervalu <1;5>. Kompletní tabulka s výsledky je uvedena v Příloze C.



Graf 7.6: Závislost času na hodnotách parametru NUMTILES

SDO_LEVEL / NUMTILES	1	2	3	4	5
5	(4,23 ± 0,75) s	(4,61 ± 0,82) s	(5,29 ± 0,93) s	(5,96 ± 1,03) s	(6,64 ± 1,16) s

[3] *querytype=join*: jedná se o parametr operátoru SDO_RELATE, hodnota *join* se používá, pokud chceme porovnávat všechny geometrie jednoho sloupce se všemi geometriemi jiného sloupce

6	(2,87 ± 0,34) s	(3,07 ± 0,37) s	(3,45 ± 0,42) s	(3,91 ± 0,48) s	(4,23 ± 0,75) s
7	(2,53 ± 0,22) s	(2,63 ± 0,23) s	(2,83 ± 0,25) s	(3,10 ± 0,27) s	(3,38 ± 0,30) s
8	(2,68 ± 0,30) s	(2,72 ± 0,29) s	(2,83 ± 0,30) s	(2,96 ± 0,30) s	(3,12 ± 0,30) s
9	(3,40 ± 0,29) s	(3,41 ± 0,29) s	(3,38 ± 0,49) s	(3,57 ± 0,3) s	(3,70 ± 0,31) s
10	(4,77 ± 0,29) s	(4,79 ± 0,29) s	(4,82 ± 0,30) s	(4,86 ± 0,30) s	(4,95 ± 0,33) s

Tabulka 7.4: Časová náročnost dotazu dle hodnot parametru `NUMTILES`

7.4.4 Porovnání indexačních struktur R-strom, Quad-strom

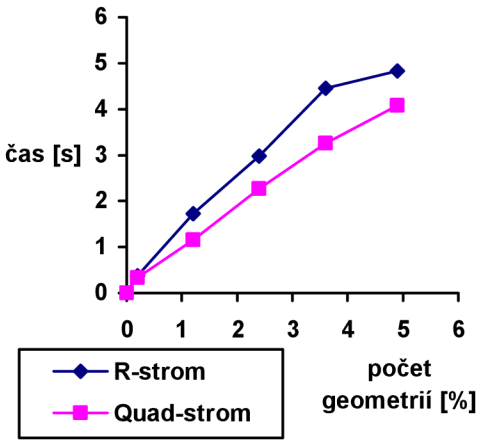
Pro indexování prostorových dat v Oracle lze využít dvě indexační struktury: R-strom a Quad-strom. Tento experiment se snaží zjistit, která indexační struktura je lepší z hlediska výkonnosti.

7.4.4.1 Návrh

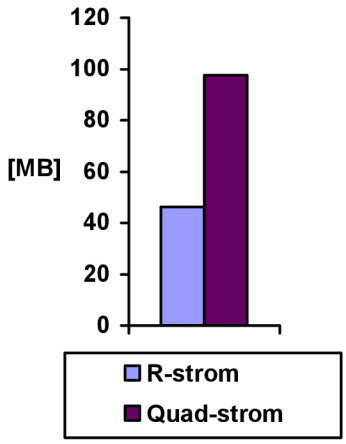
Tento experiment zkoumá výkonnost dotazu s indexační strukturou R-strom a Quad-strom. Jelikož dle experimentu 7.4.2 vyšla optimální hodnota parametru `SDO_LEVEL` 7 pro fixní indexování Quad-stromem, tuto hodnotu jsem zvolila i v tomto případě. V obou případech je nad časovou složkou vytvořen B-strom. Prostorový operátor, který je použit pro vyhledání podobných geometrií, jsem použila `SDO_RELATE`. Struktura dotazu je stejná jako u experimentu 7.4.1. Experiment se opět skládá z 5 typů dotazů vracející 1000, 6000, 12000, 18000, 24000 geometrií.

7.4.4.2 Výsledky

Z grafu 7.7 a z tabulky 7.5, které nám ukazují časovou závislost dotazů na počtu vrácených geometrií (v grafu je tento počet uveden v procentuálním vyjádření vůči celkovému počtu geometrií) jednotlivých indexačních struktur za použití operátoru `SDO_RELATE`, můžeme vyčíst, že Quad-strom je z hlediska výkonnosti rychlejší. Nicméně prostorová náročnost je u Quad-stromu o 100% vyšší než u R-stromu viz graf 7.8.



Graf 7.7: Porovnání indexačních struktur - časová náročnost dotazu



Graf 7.8: Porovnání indexačních struktur – velikost indexů

index / počet trajektorií	1000 (0,2 %)	6000 (1,2 %)	12000 (2,4 %)	18000 (3.6 %)	24000 (4,9 %)

R-strom	(0,38 ± 0,09) s	(1,72 ± 0,34) s	(2,98 ± 0,34) s	(4,46 ± 0,68) s	(4,83 ± 0,44) s
Quad-strom	(0,34 ± 0,09) s	(1,15 ± 0,16) s	(2,27 ± 0,31) s	(3,26 ± 0,47) s	(4,08 ± 0,23) s

Tabulka 7.5: Časová náročnost dotazu jednotlivých indexačních struktur

7.5 Databáze využívající LRS

Geometrii LRS, která obsahuje 3D data (2D pozici + čas), nelze indexovat pomocí Quad-stromu. Pomocí Quad-stromu lze indexovat pouze 2D data, v tomto případě by byly indexovány dimenze x, y, ale třetí dimenze, v našem případě čas, by indexována nebyla [10]. Proto v této kapitole jsou experimenty, které srovnávají možnosti využití operátoru a výkonnosti indexu (R-stromu) mezi databází využívající LRS a databází s 2D geometrií.

7.5.1 Nalezení trajektorií v části regionu

V této kapitole je popsán experiment, který se zabývá vyhledáním trajektorií (geometrií) v části regionu a v daném intervalu. Experiment byl prováděn nad databází využívající LRS a nad databází s 2D geometrií.

7.5.1.1 Návrh

Pro otestování databáze s LRS systémem jsem zvažovala dotaz, který by omezoval trajektorie jak z prostorového, tak z časového hlediska a zároveň by byl využit index (R-strom). Proto jsem zvolila jako prostorový operátor `SDO_RELATE`. Nicméně při testech na zkušební databázi jsem zjistila, že nelze porovnávat dvě geometrie využívající LRS z hlediska 3. dimenze (času). Čas se v těchto případech nebral v úvahu. Porovnání prvních dvou dimenzí proběhlo v pořádku. Proto lze použít operátor `SDO_RELATE` pouze ve `WHERE` klauzuli pro omezení počtu výsledných geometrií, které se dále zpracovávají.

Balíček funkcí `SDO_LRS` obsahuje funkci `LRS_INTERSECTION`, která by měla vrátit geometrie, jež jsou logickým součinem (operace AND). Výhodou této funkce je, že vrací geometrie správně oříznuté v časové dimenzi. Ovšem velkou nevýhodou, která mi zamezila tuto funkci použít, je její chybná implementace. Vrací LRS segmenty, které nejsou validní, a vyvolá chybu. Proto jsem se rozhodla použít funkci `SDO_LRS.CLIP_GEOM_SEGMENT` (pro oříznutí trajektorie dle daného časového intervalu) a funkci `SDO_GEOM.SDO_INTERSECTION` (pro zjištění, zda spolu geometrie interagují).

Struktura dotazu je následující:

```
SELECT count(id) FROM(
```

```
      SELECT l.id_trajectory id,
             SDO_LRS.CLIP_GEOM_SEGMENT(l.trajectory,s,e) res
      FROM lrs_trajectory_tab l, lrs_tsegment_tab t, lrs_mpoint_tab m
      WHERE SDO_LRS.CLIP_GEOM_SEGMENT(l.trajectory,s,e) IS NOT NULL
            AND SDO_RELATE(l.trajectory,  geom, 'mask=anyinteract
                           querytype=window') = 'TRUE'
            AND l.tsegment=REF(t)
            AND t.mpoint=REF(m)
) WHERE
    SDO_GEOM.SDO_INTERSECTION(res,geom,1) IS NOT NULL;
```

kde *s* je počáteční čas, *e* koncový čas, *geom* porovnávaná geometrie.
Dotaz nejdříve ořízne geometrie, které interagují se zadanou geometrií *geom*, dle zadaného časového intervalu <*s; e*> a z těchto oříznutých geometrií vybere ty, které interagují s geometrií *geom*.

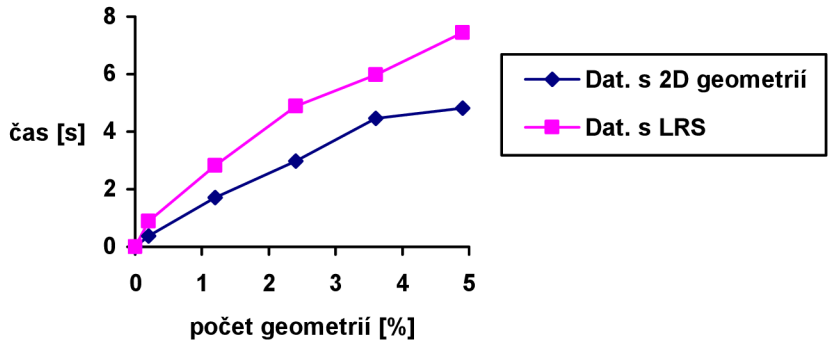
Dotaz vyhledávající trajektorie v databázi s LRS porovnávám s dotazem, který vyhledává podobné geometrie v databázi s 2D geometrií. Princip obou dotazů je shodný. Struktura dotazu pro databázi s 2D geometrií:

```
SELECT count (l.id_line)
FROM   time_tab i, tsegment_tab t, line_tab l, mpoint_tab mp
WHERE  sdo_relate(l.line, geom, 'mask=anyinteract querytype=window') =
          'TRUE'
AND l.tsegment=REF(t) AND l.start_time=REF(i) AND t.mpoint=REF(mp) AND
 i.instant BETWEEN '3.1.2010 6:00:00' AND '3.1.2010 19:00:00';
```

Experiment je založen opět na 5 typech dotazů dle počtu vrácených trajektorií. Každý typ dotazu obsahuje 10 dotazů, které jsou opakovány 8x.

7.5.1.2 Výsledky

Záporem u databáze s LRS jsou samotné funkce a operátory, které neumí dostatečně zacházet s 3. dimenzí (measure), v našem případě s časem. A je nutné vytvářet složité dotazy, které prodlouží jejich samotné vykonávání. Dle grafu 7.9 a tabulky 7.6 můžeme vidět, že dotazy nad databází s 2D geometrií jsou rychlejší. Index byl využit u obou typů dotazů.



Graf 7.9: Porovnání databází s 2D geometrií a využívající LRS

operátor / počet geometrií	1000 (0,2 %)	6000 (1,2 %)	12000 (2,4 %)	18000 (3,6 %)	24000 (4,9 %)
Dat. bez LRS	(0,38 ± 0,09) s	(1,72 ± 0,34) s	(2,98 ± 0,34) s	(4.46 ± 0,68) s	(4,83 ± 0,44) s

Dat. s LRS	(0,88 ± 0,23) s	(2,83 ± 0,49) s	(4,89 ± 0,20) s	(5,97 ± 0,27) s	(7,45 ± 0,29) s

Tabulka 7.6: Časová náročnost dotazů na databázích s 2D geometrií a využívající LRS

7.5.2 Porovnání databází s operátorem SDO_RELATE

Tento experiment porovnává databáze dle rychlosti zpracování dotazu s `SDO_RELATE` operátorem. Časovou složku v tomto případě nebere v úvahu.

7.5.2.1 Návrh

V tomto experimentu se srovnávají dvě struktury databáze: databáze využívající LRS a databáze s 2D geometrií. U každé databáze nad sloupci s geometrií je vytvořen index R-strom. Časová složka se v tomto případě neuvažuje, tudíž index nad touto složkou nemusíme vytvářet. Dotaz vyhledá geometrie, které interagují se zadanou geometrií pomocí prostorového operátoru `SDO_RELATE`. Struktura dotazu pro databázi využívající LRS je uvedena níže:

```
SELECT count (l.id_trajectory)
   FROM LRS_tsegment_tab t, LRS_line_tab l, LRS_mpoint_tab mp
   WHERE  sdo_relate(
            l.trajectory, geom, 'mask=anyinteract querytype=window'
          ) = 'TRUE'
     AND l.tsegment=REF(t) AND t.mpoint=REF(mp);
```

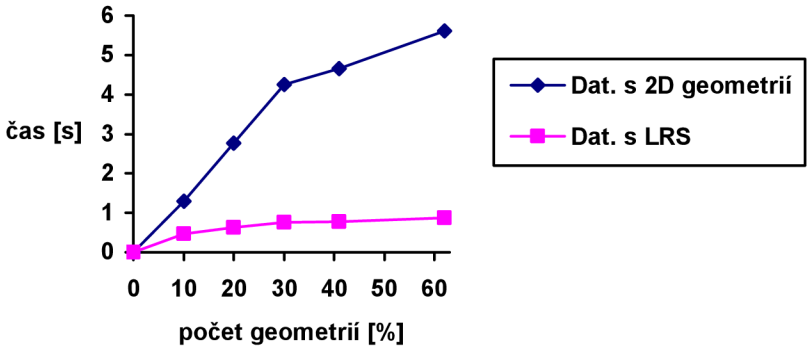
Experiment se skládá z 5 typů dotazů, které se v rámci obou databází neliší (liší se pouze názvy tabulek, které jsou však vzhledem k logické struktuře databází stejné). Dotazy jsou rozděleny na typy podle počtu vrácených geometrií (u databáze s 2D geometrií), stejná geometrie, která je použita pro vyhledání interagujících geometrií v zadané části regionu, je použita u databáze s LRS. Jednotlivé typy dotazů vyhledají přibližně 50000, 100000, 150000, 200000, 300000 geometrií. Každý typ dotazu se skládá z 10 variant dotazů, které jsou opakovány 8x. Následně je spočítán průměr a směrodatná odchylka.

7.5.2.2 Výsledky

Z grafu 7.10 a tabulky 7.7 lze vyčíst, že dotaz, který obsahuje `SDO_RELATE` a který je zaměřen pouze na prostorovou složku, je rychlejší nad strukturou databáze využívající LRS než nad strukturou databáze s 2D geometrií. Počet interagujících geometrií je v grafu 7. uveden jako procentuální poměr vůči celkovému počtu geometrií u databáze s 2D geometrií. U databáze využívající LRS nelze zjistit počet nalezených interagujících úseček, ale pouze počet interagujících trajektorií, proto procentuální vyjádření počtu experimentů odpovídá velikosti části regionu, který byl testován u obou databází.

operátor / počet geometrií	50000 (10 %)	100000 (20 %)	150000 (30 %)	200000 (41 %)	300000 (62 %)
Dat. bez LRS	(1,30 ± 0,15) s	(2,77 ± 0,33) s	(4,26 ± 0,66) s	(4,65 ± 0,51) s	(5,61 ± 0,64) s
Dat. s LRS	(0,47 ± 0,09) s	(0,63 ± 0,07) s	(0,76 ± 0,08) s	(0,77 ± 0,12) s	(0,87 ± 0,08) s

Tabulka 7.7: Časová náročnost dotazů jednotlivých typů struktur databází



Graf 7.10: Časová závislost na počtu vrácených geometrií

7.6 Porovnání všech struktur databází

V této kapitole jsou uvedeny experimenty, které se snaží porovnat všechny struktury databází na dvou typech dotazů. Účelem této kapitoly je zhodnotit vhodnost použití jednotlivých struktur databází.

7.6.1 Nalezení trajektorií v části regionu

V této kapitole je popsán experiment, který se zabývá vyhledáním trajektorií (geometrií) v části regionu a v daném časovém intervalu. Experiment je proveden na všech strukturách databází.

7.6.1.1 Návrh

Cílem tohoto experimentu je zjistit, která struktura je nejvýhodnější z pohledu časové náročnosti dotazu, který vyhledává geometrie v části regionu v daném časovém úseku. To znamená, že dotaz bude zohledňovat jak prostorovou, tak časovou složku dat. Jelikož při vytvořeném indexu 3D R-strom, který je vytvořen nad geometrií v databázi s 3D geometrií, nelze použít operátor `SDO_RELATE`, bude v tomto experimentu použit v dotazech nad databázemi, které obsahují geometrii, operátor `SDO_FILTER`. U databáze bez geometrie je nutné porovnávanou část regionu vyjádřit jiným způsobem. Experiment se skládá z 5 typů dotazů dle počtu nalezených geometrií případně velikosti porovnávací plochy. V tomto návrhu jsou dále popsány návrhy experimentů pro jednotlivé struktury.

Databáze s 2D geometrií

U této struktury databáze je vytvořen R-strom nad *line_tab.line* a B-strom nad *time_tab.instant*. Tento typ dotazu byl již použit u dřívějších experimentů.

Struktura dotazu s operátorem `SDO_FILTER`:

```
SELECT count (l.id_line)
FROM   time_tab i, tsegment_tab t, line_tab l, mpoint_tab mp
WHERE  sdo_filter(l.line, geom) = 'TRUE'
AND l.tsegment=REF(t) AND l.start_time=REF(i) AND t.mpoint=REF(mp) AND
 i.instant BETWEEN '3.1.2010 6:00:00' AND '3.1.2010 19:00:00';
```

kde *geom* je geometrie (část regionu tvaru obdélník), se kterou jsou porovnávány úsečky.

Databáze využívající LRS

U databáze využívající LRS je vyhledání časoprostorových dat, omezených jak prostorově tak časově, značně problematické. Bližší popis problému byl uveden u experimentu 7.5.1. Dotaz, který je

uveden níže, má následující strukturu: nejdříve vyfiltruje geometrie, které interagují se zadanou geometrií, tyto geometrie ořízne dle zadaného časového intervalu a nakonec se provede opět porovnání se zadanou geometrií, zda část geometrie (oříznutá geometrie dle časového intervalu) interaguje se zadanou geometrií.
Struktura dotazu je následující:

```
SELECT count(id) FROM(
     SELECT l.id_trajectory id,
            SDO_LRS.CLIP_GEOM_SEGMENT(l.trajectory,s,e) res
     FROM lrs_trajectory_tab l, lrs_tsegment_tab t, lrs_mpoint_tab m
     WHERE SDO_LRS.CLIP_GEOM_SEGMENT(l.trajectory,s,e) IS NOT NULL
           AND SDO_FILTER(l.trajectory,  geom) = 'TRUE'
           AND l.tsegment=REF(t)
           AND t.mpoint=REF(m)
) WHERE
    SDO_GEOM.SDO_INTERSECTION(res,geom,1) IS NOT NULL;
```

kde *s* je počáteční čas, *e* koncový čas, *geom* porovnávaná geometrie.

Databáze s 3D geometrií

Databáze s 3D geometrií obsahuje geometrii, která se skládá ze tří dimenzí: x-ová souřadnice, y-ová souřadnice a čas. Proto v tomto případě není nutno se zabývat časovou složkou. Je obsažena v geometrii, kde spolu s ní je indexována pomocí R-stromu. Quad-strom v tomto případě nelze vytvořit (V Oracle lze vytvářet Quad-strom max. nad 2D geometriemi). Pokud se v tomto případě chceme dotazovat na část regionu, kde se geometrie vyskytovaly v určitém čase, je nutné vytvořit dotaz, který jako porovnávací geometrii bude mít kvádr. Struktura dotazu je:

```
SELECT count (l.id_line)
   FROM tsegment_tab3D t, line_tab3D l, mpoint_tab3D mp
   WHERE SDO_FILTER(
         l.line,
         MDSYS.SDO_GEOMETRY(
         3007,
         NULL,
         NULL,
         MDSYS.SDO_ELEM_INFO_ARRAY(
         1,1003,1, 16,1003,1, 31,1003,1, 46,1003,1, 61,1003,1, 76,1003,1),
         MDSYS.SDO_ORDINATE_ARRAY(cube_ordinates))
         ) = 'TRUE'
    AND l.tsegment=REF(t) AND t.mpoint=REF(mp);
```

kde *cube_ordinates* jsou souřadnice kvádru (kvádr je nutno vyjádřit jako 6 obdélníků).

Databáze bez geometrie

U databáze bez geometrie nelze použít prostorové operátory (neexistuje žádná indexovaná geometrie). První způsob, kterým lze zjistit, zda body (v tomto případě se neporovnávají úsečky jako geometrie) patří do části regionu, je pomocí operátoru >= a <=. Nicméně lze v tomto případě očekávat, že se nenaleznou všechny úsečky (v tabulce *line*) tvořené hledanými body.

Struktura dotazu je následující:

```
SELECT count (l.id_line)
   FROM tsegment_tabF t, line_tabF l, point_tabF p, point_tabF pp,
        mpoint_tabF mp
   WHERE ((p.pos_x<=x1 AND p.pos_x>=x2 AND p.pos_y<=y1 AND p.pos_y>=y2)
     OR (pp.pos_x<=x1 AND pp.pos_x>=x2 AND pp.pos_y<=y1 AND pp.pos_y>=y2))
     AND l.tsegment=REF(t) AND l.start_point=REF(p)
     AND l.end_point=REF(pp) AND t.mpoint=REF(mp)
     AND p.instant BETWEEN '3.1.2010 6:00:00' AND '3.1.2010 19:00:00';
```

kde *x1, y1, x2, y2* jsou souřadnice porovnávaného obdélníku (levý horní a pravý spodní bod).
Druhý způsob pro nalezení již přesného počtu interagujících úseček je vyjádření počátečního a koncového bodu úsečky (v tabulce *line*) pomocí geometrie a následné porovnání funkcí `SDO_GEOM.RELATE` s maskou `anyinteract` (z hlediska počtu nalezených geometrií je tato funkce stejná jako operátor `SDO_RELATE`). Struktura dotazu je následující:

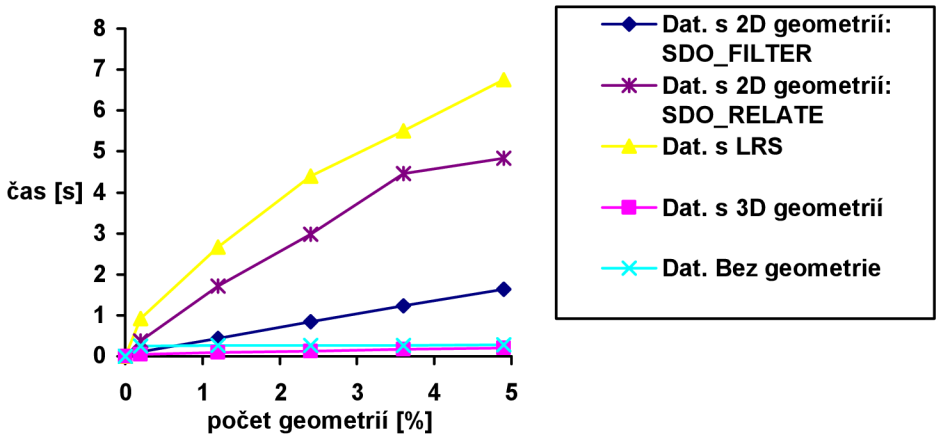
```
SELECT count (l.id_line)
   FROM tsegment_tabF t, line_tabF l, point_tabF p, point_tabF pp,
        mpoint_tabF mp
   WHERE  sdo_geom.relate(
             geom_line(p.pos_x,p.pos_y,pp.pos_x,pp.pos_y),
             'anyinteract', geom, 0.005
         ) = 'TRUE'
     AND l.tsegment=REF(t) AND l.start_point=REF(p) AND
     l.end_point=REF(pp) AND t.mpoint=REF(mp) AND
     p.instant between '3.1.2010 6:00:00' and '3.1.2010 19:00:00';
```

kde *geom_line(p.pos_x, p.pos_y, pp.pos_x, pp.pos_y)* je geometrie úsečky a *geom* je porovnávaná geometrie.
Experiment opět zohledňuje prostorovou i časovou složku. Časová složka je indexována pomocí B-stromu.

7.6.1.2 Výsledky

V grafu 7.11 a v tabulce 7.8 lze vidět výsledky jednotlivých experimentů. V grafu není uveden dotaz Dat. bez geom: `SDO_GEOM.RELATE` uvedený v tabulce. Jelikož dotazy nad databází s LRS a bez geometrie simulují spíše dotaz vyhledávající geometrie s operátorem `SDO_RELATE`, pro porovnání jsem přidala hodnoty výsledků dotazů nad databází s 2D geometrií (s operátorem `SDO_RELATE`). Nejmenší časovou náročnost měly dotazy nad databází s 3D geometrií a nad databází bez geometrie. Nicméně dotaz nad databází bez geometrie nevyhledal veškeré úsečky interagující se zadanou geometrií. 0,014% úseček nebylo vyhodnoceno jako interagující. Řešením tohoto problému je druhý dotaz pro databázi bez geometrie využívající funkci `SDO_GEOM.RELATE`[4]. Tento dotaz nalezl veškeré interagující úsečky, ale jeho časová složitost je příliš vysoká (hodnoty jsou uvedeny pouze v tabulce). Prostorový index R-strom byl využit u dotazů nad databází s 2D, 3D geometrií a nad databází využívající LRS.

[4] Při testování funkce SDO_GEOM.RELATE na databázi s 2D geometrií a bez geometrie (v případě databáze s 2D geometrií byl dotaz obdobný, testovala se geometrie l.line) byl dotaz nad databází s 2D geometrií asi 14,3x pomalejší. Toto zpomalení je dáno vyhodnocením dotazu optimalizátorem. U databáze s 2D geometrií se vyhodnocovaly veškeré úsečky. U databáze bez geometrie se vyhodnocovaly úsečky, které patřily do daného časového intervalu.



Graf 7.11: Závislost času na počtu nalezených geometrií dle typu dotazu, databáze

operátor / počet geometrií	1000 (0,2 %)	6000 (1,2 %)	12000 (2,4 %)	18000 (3,6 %)	24000 (4,9 %)
Dat. s 2D geom. SDO_FILTER	(0,11 ± 0,01) s	(0,44 ± 0,04) s	(0,84 ± 0,30) s	(1,23 ± 0,04) s	(1,64 ± 0,04) s
Dat. s 2D geom. SDO_RELATE	(0,38 ± 0,09) s	(1,72 ± 0,34) s	(2,98 ± 0,34) s	(4.46 ± 0,68) s	(4,83 ± 0,44) s
Dat. s LRS	(0,92 ± 0,29) s	(2,67 ± 0,50) s	(4,40 ± 0,17) s	(5,51 ± 0,20) s	(6,75 ± 0,22) s
Dat. 3D	(0,05 ± 0,01) s	(0,09 ± 0,00) s	(0,13 ± 0,00) s	(0,17 ± 0,00) s	(0,21 ± 0,00) s
Dat. bez geom.	(0,25 ± 0,00) s	(0,26 ± 0,00) s	(0,26 ± 0,01) s	(0,27 ± 0,01) s	(0,28 ± 0,00) s
Dat. bez geom.: SDO_GEOM.RELATE	(11,32 ± 0,10) s	(12,39 ± 0,08) s	(13,35 ± 0,04) s	(14,25 ± 0,07) s	(15,08 ± 0,08) s

Tabulka 7.8: Časová náročnost jednotlivých dotazů

7.6.2 Nalezení nejbližší trajektorie

Tento experiment zkoumá možnosti jednotlivých struktur databází pro vyhledání nejbližší trajektorie od zadaného bodu a zjištění vzdálenosti od tohoto bodu.

7.6.2.1 Návrh

Prostorový operátor, který využívá prostorový index pro nalezení nejbližší geometrie od zadaného bodu a určení vzdálenosti, je `SDO_NN` a `SDO_NN_DISTANCE`. Experiment je založen na nalezení nejbližší geometrie od zadaného bodu v daném časovém intervalu. Níže jsou popsány realizace dotazů u jednotlivých struktur databází.

Databáze s 2D geometrií

U struktury databáze s 2D geometrií pro nalezení nejbližší trajektorie lze použít prostorový operátor `SDO_NN`. Nad prostorovou složkou (*line_tab.line*) je vytvořen R-strom, nad časovou složkou je vytvořen B-strom. Dotaz vyhledá pouze jednu nejbližší geometrii v zadaném časovém intervalu.

Struktura dotazu je:

```
SELECT t.id_tsegment, SDO_NN_DISTANCE(1) dist
 FROM tsegment_tab t, line_tab l, mpoint_tab mp, region_tab r,
    time_tab i
 WHERE l.tsegment=REF(t) AND l.start_time=REF(i) AND
    t.mpoint=REF(mp) AND t.region=REF(r) AND
    SDO_NN(l.line, geom_point, 'sdo_batch_size=10', 1) = 'TRUE' AND
    ROWNUM <=1 AND
    i.instant BETWEEN '3.1.2010 6:00:00' AND '3.1.2010 19:00:00';
```

kde *geom_point* je geometrie bodu.
Bod, ke kterému se vyhledávají nejbližší trajektorie, jsem zvolila o souřadnicích [10000,10000].

Databáze využívající LRS

U tohoto typu databáze je nutné simulovat časový interval. Jelikož prostorový operátor `SDO_NN` lze použít pouze na indexovaná data, nebylo možné vytvořit jednotný (`select`) dotaz. K nalezení nejbližší trajektorie používám uloženou proceduru:

```
NN_LRS(x IN NUMBER, y IN NUMBER, start_time IN NUMBER,
                    end_time IN NUMBER)
```

kde *x*, *y* jsou souřadnice bodu, od kterého hledám nejbližší trajektorii, *start_time*, *end_time* je počáteční a koncový čas. Základní myšlenkou pro nalezení nejbližší trajektorie je oříznutí geometrií dle časového intervalu. A následné procházení jednotlivých oříznutých geometrií a počítání vzdálenosti od zadaného bodu pomocí funkce `SDO_GEOM.SDO_DISTANCE` a konečného vyhodnocení nejbližší trajektorie. Celé tělo procedury je uvedeno v Příloze D.
Lze ji volat následujícím způsobem:

```
Call NN_LRS(10000,10000,3240,4020);
```

kde 3240 odpovídá datu 3.1.2010 6:00:00 a 4020 odpovídá datu 3.1.2010 19:00:00.

Je nutné mít zapnutý výpis na výstup pomocí:

```
set serveroutput on;
```

Databáze s 3D geometrií

Nad 3D geometrií lze použít prostorový operátor `SDO_NN`, nicméně s použitím tohoto operátoru nemusíme vždy dostat správné výsledky. Ty bychom dostali za předpokladu, že všechny objekty by se pohybovaly stejnou konstantní rychlostí a časová souřadnice 3D prostoru by byla optimálně nastavena. Jelikož tomu tak není, pro nalezení nejbližší trajektorie v daném časovém intervalu využívám obdobný postup jako u nalezení nejbližší trajektorie u databáze využívající LRS. Nejdříve vyberu (oříznu) geometrie, které jsou v daném časovém intervalu, a poté zjistím vzdálenost od zadaného bodu a vyhodnotím nejbližší geometrii. K tomu je implementována procedura:

```
NN_3D(x IN NUMBER, y IN NUMBER, start_time IN NUMBER,
                   end_time IN NUMBER)
```

a je uvedena v Příloze E.

Databáze bez geometrie

Pro tento typ databáze jsem navrhla 2 dotazy. První, který vyhledá nejbližší bod od zadaného bodu, a druhý, který vyhledá nejbližší úsečku od zadaného bodu. První typ dotazu je implementován procedurou:

```
NN_FULL(x IN NUMBER, y IN NUMBER, start_time
        IN DATE, end_time IN DATE)
```

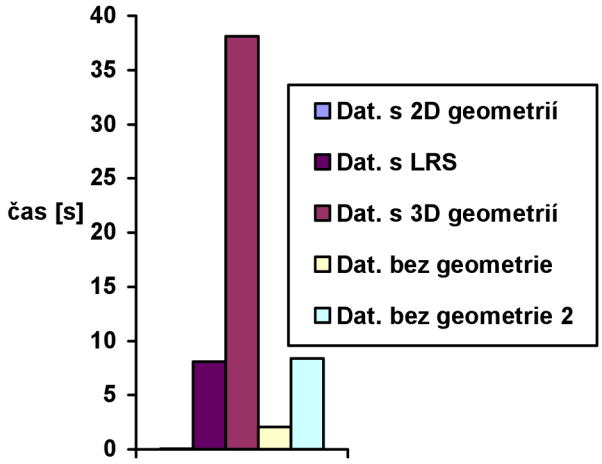
kde *start_time*, *end_time* se zadává ve formátu: '`DD.MM.YYYY HH:MM:SS`', např. '3.1.2010 6:00:00'. Procedura nejdříve vybere body, které se nachází v daném časovém intervalu, a následně za využití euklidovské vzdálenosti spočítá vzdálenost mezi tímto a zadaným bodem. Vyhodnocením jednotlivých vzdáleností získáme nejbližší bod od zadaného bodu. Tělo procedury je uvedeno v Příloze F. Druhý způsob, který vyhledá nejbližší úsečku od zadaného bodu, je implementován procedurou:

```
NN_FULL_G(x IN NUMBER, y IN NUMBER, start_time
          IN DATE, end_time IN DATE)
```

Procedura vybere úsečky (2 body tvořící úsečku) nacházející se v daném intervalu (*start_time, end_time*) a za pomocí funkce `SDO_DISTANCE` určí vzdálenost mezi zadaným bodem a úsečkou. Následně jsou výsledky vyhodnoceny. Tělo procedury je uvedeno v Příloze G.

7.6.2.2 Výsledky

Výsledky jednotlivých měření lze vidět na grafu 7.12. Pro nalezení nejbližší trajektorie vyšla z časového hlediska nejlépe databáze s 2D geometrií. Dobrý čas zpracování dotazu měla i databáze bez geometrie s dotazem[5] vyhodnocující vzdálenost 2 bodů za použití euklidovské vzdálenosti, v grafu označeno jako Dat. bez geometrie. Nicméně výsledná vzdálenost nebyla stejná jako u databáze s 2D geometrií, kde se počítala vzdálenost od úsečky. Dotaz nad databází bez geometrie (v grafu označeno jako Dat. bez geometrie 2) počítající vzdálenost mezi zadaným bodem a vytvořenou úsečkou dával přesné výsledky, ale za cenu vyšší časové náročnosti. Prostorový index R-strom využívají pouze databáze s 2D a 3D geometrií. Velká časová náročnost u dotazu nad databází s 3D geometrií je zapříčiněna pomalým vyhodnocováním vzdálenosti funkcí `SDO_GEOM.SDO_DISTANCE` s 3D geometrií. V tabulce 7.9 jsou uvedeny přesné hodnoty výsledků experimentů.



Graf 7.12: Časová náročnost dotazů dle jednotlivých databází

	Dat. s 2D geometrií	Dat. s LRS	Dat. s 3D geometrií	Dat. bez geometrie	Dat. bez geometrie 2
Čas[s]	(0,07 ± 0,00) s	(8,10 ± 0,10) s	(38,10 ± 0,72) s	(2,08 ± 0,04) s	(8,36 ± 0,81) s

Tabulka 7.9: Časová náročnost dotazů dle jednotlivých databází

[5] V dotazu nad databází bez geometrie (vzdálenost dvou bodů) jsem vzdálenost počítala dvěma způsoby: použití vektorového počtu a použití funkce sdo_geom.sdo_distance. S použitím funkce sdo_geom.sdo_distance byla časová náročnost 3,5x větší (7,26 s).

8 Zhodnocení experimentů

Experimenty, které jsem v této práci provedla, se zaměřovaly na časovou složitost dotazů nad jednotlivými strukturami databází v závislosti na použité indexační struktuře a prostorovém operátoru. U všech struktur, kde časová složka je oddělena od prostorové, je časová složka indexována pomocí B-stromu.

Jednotlivé struktury databází se liší uložením časové složky, ale i možností použití indexačních struktur a prostorových operátorů. Indexační strukturu Quad-strom v databázovém systému Oracle lze použít pouze na 2D data, ale R-strom lze použít i na 3D data. Při použití prostorového operátoru nad databází využívající LRS byla časová složka ignorována. V tomto případě byly prostorové operátory využívány okrajově. Bylo nutné použít funkce, které oříznou trajektorii dle časového intervalu, a následně se provedou další funkce v závislosti na tom, co chceme zjistit. U databáze s 3D geometrií lze použít pouze omezený počet prostorových operátorů a to [3]:

- `SDO_FILTER`
- `SDO_ANYINTERACT`
- `SDO_INSIDE`
- `SDO_NN`
- `SDO_WITHIN_DISTANCE`

Nicméně v případě ukládání časoprostorových dat je nalezení nejbližší trajektorie pomocí operátoru `SDO_NN` nevyhovující, opět bylo nutné (jako v případě databáze využívající LRS) využít prostorové funkce.

Databáze bez geometrie neobsahuje žádný sloupec typu `SDO_GEOMETRY`, proto nelze využít indexačních struktur R-strom a Quad-strom a ani prostorových operátorů. Ale lze použít prostorové funkce, kde koncové body úseček budou vytvářet geometrii.

Databáze s 2D geometrií může využívat veškeré prostorové operátory a indexační struktury R-strom a Quad-strom nad prostorovou složkou.

Pokud bych chtěla srovnat výkonnost dotazů nad jednotlivými typy databází, je nutné zohlednit typ dotazu a případnou náročnost porovnávané geometrie. Jestliže se dotaz skládal z jednoduchého SQL dotazu obsahující prostorový operátor (což bylo pouze v případě databáze s 2D geometrií v obou dotazech a 3D geometrií v dotazu pro nalezení trajektorií v části regionu), doba zpracování dotazu byla krátká. Dotazy nad databází bez geometrie byly zpracovávány velice rychle ale pouze v případech, kdy se koncové body úseček nepřeváděly na geometrie. Toto mělo za důsledek, že např. nebyly nalezeny veškeré trajektorie v prohledávané části regionu nebo byla vrácena chybná vzdálenost k nejbližší trajektorii, či samotná trajektorie od zadaného bodu. Souhrn srovnání jednotlivých databází je uveden v tabulce 8.1.

Další zajímavé experimenty, které jsou uvedeny v této práci, porovnávají indexační struktury R-strom a Quad-strom použitelné nad databází s 2D geometrií. Pokud u Quad-stromu zadáme správnou hodnotu pro parametr `SDO_LEVEL`, dotazy nad databází s Quad-stromem jsou v tomto případě rychlejší než dotazy nad databází s R-stromem. Při porovnání operátoru `SDO_FILTER` a `SDO_RELATE` v dotazech nad databází s 2D geometrií, byl dotaz s operátorem `SDO_FILTER` rychlejší, ale vyhodnotil geometrie jako interagující i takové, které byly s porovnávanou geometrií disjunktní.

	Databáze s 2D geometrií	Databáze využívající LRS	Databáze s 3D geometrií	Databáze bez geometrie
Možnost vytvoření R-stromu nad `SDO_GEOMETRY`	Ano	Ano	Ano	Ne
Možnost vytvoření Quad-stromu nad `SDO_GEOMETRY`	Ano	Ne	Ne	Ne
Možnost použití operátoru `SDO_RELATE`	Ano	Ano[1]	Ne	Ne
Možnost použití operátoru `SDO_FILTER`	Ano	Ano[2]	Ano	Ne
Možnost použití operátoru `SDO_NN`	Ano	Ne[3]	Ne[4]	Ne
Vyhledání trajektorií v části tvaru obdélník	Ano	Ano	Ano	Ano
Vyhledání trajektorií v části se složitým tvarem	Ano	Ano	Ano	Ne

[1] Operátor `SDO_RELATE` je u databáze využívající LRS funkčně omezen pouze na prostorovou složku

[2] Operátor `SDO_FILTER` je u databáze využívající LRS funkčně omezen pouze na prostorovou složku

[3] Operátor `SDO_NN` je u databáze využívající LRS funkčně omezen na prostorovou složku. Pro nalezení nejbližší trajektorie v daném intervalu je tento operátor nepoužitelný

[4] Operátor `SDO_NN` nelze použít k nalezení přesných výsledků u databáze s 3D geometrií, kde jedna dimenze představuje čas (lze použít, pokud by se objekty pohybovaly stejnou konstantní rychlostí).

Tabulka 8.1: Přehled vlastností jednotlivých struktur databází

Závěr

S nárůstem nových technologií a aplikací, které pracují s pohybujícími se objekty a uchovávají je v databázi, databáze musí pracovat s obrovským množstvím dat. Rychlejší vyhledávání v těchto datech nám usnadní index, který je vytvořen nad časoprostorovými daty. Proto se tato práce zabývá indexačními strukturami a dalšími přístupy k indexování časoprostorových dat a možnostmi indexování těchto dat v databázovém systému Oracle 11g.

Cílem bylo navrhnout struktury databází pro uložení časoprostorových dat a zhodnotit jejich vhodnost pro uložení těchto dat. Navrhla jsem čtyři struktury lišící se v uložení prostorových a časových složek. Jednotlivé databáze jsem dále otestovala. Testy se zaměřovaly na dotazy, které obsahovaly prostorový operátor. Tento operátor využívá prostorový index, tudíž pokud mnou navržená databáze je schopna tento operátor využít, dochází k velkému zrychlení doby zpracování dotazu. Nicméně výkonnost dotazu je značně ovlivněna optimalizátorem, který vytváří tzv. plán vykonávání SQL dotazu. Optimalizátor by měl vyhodnotit nejefektivnější přístup, jak dotaz provést, ale bohužel ne vždy tomu tak bylo.

Výsledky jednotlivých experimentů jsou popsány u jednotlivých experimentů a shrnuty v kapitole 8. Z pohledu využití indexačních struktur, které databázový systém Oracle poskytuje, a z prostorových operátorů vyšla nejlépe databáze s 2D geometrií, která ukládá zvlášť prostorovou a časovou složku. Tyto složky mohou být indexovány B-stromem a R-stromem nebo Quad-stromem. Doba zpracování dotazu nad touto geometrií byla dobrá. Z pohledu výkonnosti byly nejrychlejší dotazy nad databází bez geometrie. Nevýhodou tohoto přístupu byly nepřesné či neúplné výsledky. Ty lze dosáhnout převodem koncových bodů úseček na typ `SDO_GEOMETRY`, ale je to za cenu rapidního zpomalení zpracování dotazu (v tomto případě se tuto strukturu nevyplatí používat).

Další dvě struktury: databáze využívající LRS a databáze s 3D geometrií jsou velmi omezeny z pohledu využití prostorových operátorů. Databáze s 3D geometrií může používat pouze některé prostorové operátory. U databáze využívající LRS při použití prostorových operátorů byla časová složka ignorována, proto v tomto případě lze prostorové operátory používat pouze okrajově a využít další funkce, což zpracování dotazu zpomaluje.

V průběhu vytváření této práce mě zaujaly dvě věci, které by mohly na tuto práci volně navázat. Jednou z nich je obrovské množství indexačních struktur, které jsou vhodné pro časoprostorová data. Jelikož Oracle neposkytuje žádnou indexační strukturu vhodnou přímo pro časoprostorová data, tyto nové struktury mohou poskytnout zrychlení zpracování dotazu, ale třeba i některé nevýhody. Dále mě zaujal optimalizátor Oracle a jeho obrovský vliv na dobu zpracování dotazu. Jelikož při vyhledávání nad obrovským množstvím dat, která jsou navíc prostorová, je doba zpracování vysoká, je nutné, aby optimalizátor vyhodnotil plán provádění opravdu efektivně.

Literatura

[1] Güting R.H, Schneider M.: *Moving Object Databases*, Hardbound, ISBN-13: 978-0-12-088799-6, ISBN-10: 0-12-088799-1, Morgan Kaufman, Elsevier, 2005, [cit. 2010-01-02], s. 261-316.

[2] Mallet, D., J.: *Relational Database Support for Spatio-Temporal Data*, Department of computing Science University of Alberta Edmonton, Alberta, Canada, 2004, [cit. 2010-01-02].

[3] Ihm J., Lopez X., Ravada S.: *Oracle Spatial 11g: Advanced Spatial Data Management For Enterprise Applications*, Oracle Corporation, 2007, [cit. 2010-01-02], dokument dostupný na: <http://www.oracle.com/technology/products/spatial/pdf/11g_collateral/spatial11g_advdatamgmt_twp.pdf >

[4] Murray C.: *Oracle® Spatial Developer's Guide 11g Release 1 (11.1),* Oracle Corporation, 2009, [cit. 2010-01-02], dokument dostupný na: <http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28400/toc.htm>

[5] Garmany J., Karam S., Hartmann L., Jain V. J. Carr B., *Oracle 11g New Index Features*, 2008, [cit. 2010-01-02],
článek dostupný na: <http://www.dba-oracle.com/t_11g_new_index_features.htm>

[6] Mokbel M. F., Ghanem T. M., Aref W. G.: *Spatio-Temporal Access Method*, Department of Computer Sciences, Purdue University, West Lafayette, [cit. 2010-01-02], dokument dostupný na: <ftp://ftp.research.microsoft.com/pub/debull/A03june/arefF.ps>

[7] Tao Y., Papadias D.: *The MV3R-Tree: A Spatio-Temporal Access Metod for Timestamp and Interval Queries*, Department of Computer Science, Hong Kong Univerzity of Science and Technology, [cit. 2010-01-02], dokument dostupný na: <http://www.vldb.org/conf/2001/P431.pdf>

[8] Tao Y., Papadias D.: *The TPR*-Tree: An Optimized Spatio-Temporal Access Metod for Predictive Queries,* Department of Computer Science, Hong Kong Univerzity of Science and Technology, [cit. 2010-01-02], dokument dostupný na <http://www.cs.ust.hk/~dimitris/PAPERS/VLDB03-TPR.pdf>

[9] Kothuri, R. K., Ravada, S., Abugov, D.: Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data. V *Proceedings of the 2002 ACM SIGMOD international Conference on Management of Data* (Madison, Wisconsin, 2002). SIGMOD '02. ACM, New York, NY, 546-557, [cit. 2010-01-02], článek dostupný na <http://doi.acm.org/10.1145/564691.564755>

[10] Abugov D., Blowney J., Geringer D., Ravada S.: *Oracle Spatial Quadtree indexing*, 10g Release 1 (10.1) Oracle Corporation, 2003, [cit. 2010-05-14], dokument dostupný na <http://www.oracle.com/technology/products/spatial/pdf/qt.pdf>

[11] Dongseop K., Sangjun L., Sukho L.: *Indexing the Current Posititons of Moving Objects Using the Lazy Update R-tree*, mdm, pp.113, Third International Conference on Mobile Data Management, 2002, [cit. 2010-05-14], dokument dostupný na: <http://doi.ieeecomputersociety.org/10.1109/MDM.2002.994387>

[12] *Hybrid Indexing*, Oracle Spatial User's Guide and References, Release 9.2, 2002, [cit. 2010-05-14], článek dostupný na: <http://download.oracle.com/docs/cd/B10501_01/appdev.920/a96630/sdo_hybrid.htm#g632325>

[13] Vetešník J., *Indexování pohybujících se objektů*, diplomová práce, Brno, FIT VUT v Brně, 2008, [cit. 2010-05-14]

[14] Yufei T., Dimitris P.: *Efficient Historical R-Trees*, ssdbm, pp.0223, Thiteenth International Conference on Scientific and Statistical Database Management, 2001, [cit. 2010-05-14], dokument dostupný na: <http://doi.ieeecomputersociety.org/10.1109/SSDM.2001.938554>

[15] Porkaew K., Lazaridis I., Mehrotra S.: *Querying Mobile Objects in Spatio-Temporal Databases*, University of Technology at Thonburi, Thailand, University of California, Irvine, USA, [cit. 2010-05-14], s. 69-70,dokument dostupný na <http://www.springerlink.com/content/9h2d0lbjpq4a9k5f/fulltext.pdf>

[16] Brinkhoff T.: *Network-based Generator of Moving Objects*, 2005, [cit. 2010-05-14], stránka dostupná na <http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/>

[17] Urman S., Hardman R., McLaughlin M.: *Oracle Programování v PL/SQL*, ISBN: 978-80-251-1870-2, Computer Press, a. s., 2004, [cit. 2010-05-14]

[18] *The Query Optimizer*, Oracle Database Performance Tuning Guide 11g Release 1 (11.1), 2008, [cit. 2010-05-14], článek dostupný na: <http://download.oracle.com/docs/cd/B28359_01/server.111/b28274/optimops.htm#i21299>

[19] *R-tree*, [cit. 2010-05-14], článek dostupný na: <http://en.wikipedia.org/wiki/R-tree>

Seznam příloh

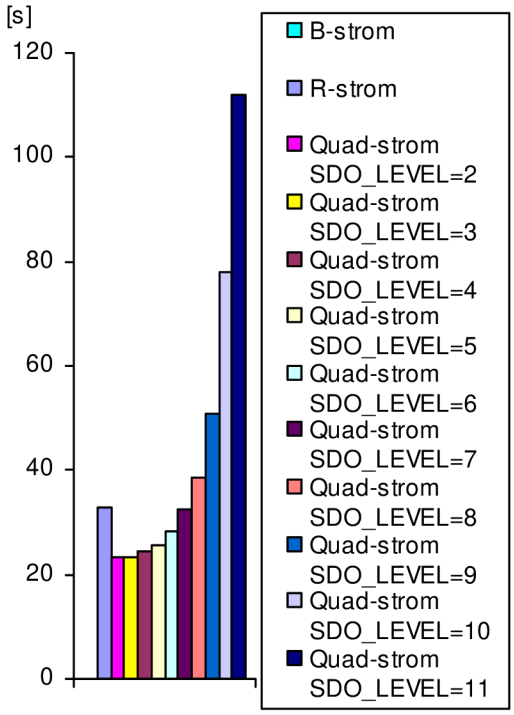
Příloha A. Tabulky k úvodním statistikám
Příloha B. Explain plan SQL dotazu
Příloha C. Tabulka k experimentu 7.4.3
Příloha D. Procedura NN_LRS
Příloha E. Procedura NN_3D
Příloha F. Procedura NN_Full
Příloha G. Procedura NN_Full_D
Příloha H. Obsah přiloženého CD

Příloha A. Tabulky k úvodním statistikám

Databáze s 2D geometrií

Tabulka a graf s časem vytvoření indexů nad Line_tab.line(R-strom, Quad-strom) a nad Time_tab.instant (B-strom):

		Čas[s]	Velikost[MB]
B-strom		(0,05 ± 0,02)	0,13
R-strom		(32,82 ± 2,08)	44,13
Quad-strom nad line_tab.line SDO_LEVEL:	**2**	(23,17 ± 0,95)	40,00
	3	(23,25 ± 0,41)	42,00
	4	(24,31 ± 0,91)	45,00
	5	(25,73 ± 0,62)	53,00
	6	(28,33 ± 0,88)	66,00
	7	(32,42 ± 0,78)	93,00
	8	(38,69 ± 1,20)	145,00
	9	(50,76 ± 1,15)	264,19
	10	(1:18,15 ± 1,95)	476,13
	11	(1:51,99 ± 3,25)	902,00



Databáze s využitím LRS

Tabulka s časem vytvoření R-strom a jeho velikostí nad LRS_Trajetory_tab.trajectory:

	Čas[s]	Velikost[MB]
R-strom	(1,67 ± 0,61)	2,13

Databáze s 3D geometrií

Tabulka s časem vytvoření 3D R-strom a jeho velikostí nad Line_tab3D.line:

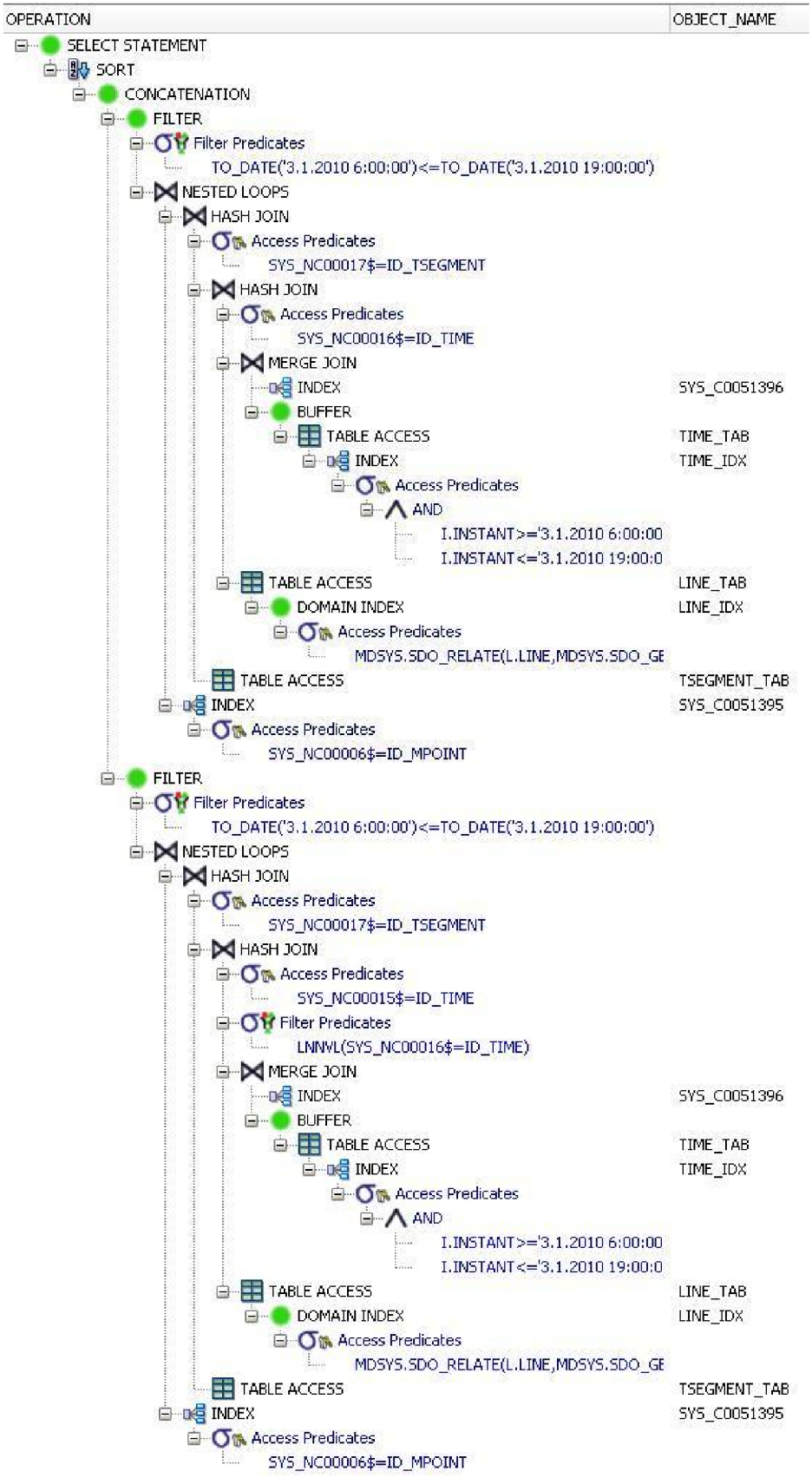
	Čas[s]	Velikost[MB]
R-strom	(40,19 ± 1,46)	44,13

Databáze bez geometrií

Tabulka s časem vytvoření B-strom a jeho velikostí nad Point_tabF.instant:

	Čas[s]	Velikost[MB]
B-strom	(0,60 ± 0,05)	11,00

Příloha B. Explain plan SQL dotazu



Příloha C. Tabulka k experimentu 7.4.3

SDO_LEVEL	NUMTILES							
	1	2	3	4	5	6	7	8
2	(20,6 ± 3,33) s	(23,49 ± 4,20) s	(27,37 ± 5,41) s	(31,39 ± 6,76) s	(35,26 ± 8,06) s	(39,21 ± 9,42) s	(42,71 ± 10,59) s	(46,26 ± 11,77) s
3	(12,95 ± 2,21) s	(14,48 ± 2,48) s	(16,45 ± 2,81) s	(18,58 ± 3,21) s	(20,50 ± 3,53) s	(22,57 ± 3,89) s	(24,37 ± 4,22) s	(26,43 ± 4,57) s
4	(7,01 ± 1,43) s	(7,79 ± 1,59) s	(8,91 ± 7,82) s	(10,10 ± 2,07) s	(11,2 ± 2,29) s	(12,22 ± 2,51) s	(13,41 ± 2,67) s	(14,75 ± 3,04) s
5	(4,23 ± 0,75) s	(4,61 ± 0,82) s	(5,29 ± 0,93) s	(5,96 ± 1,03) s	(6,64 ± 1,16) s	(7,27 ± 1,27) s	(7,86 ± 1,34) s	(8,62 ± 1,48) s
6	(2,87 ± 0,34) s	(3,07 ± 0,37) s	(3,45 ± 0,42) s	(3,91 ± 0,48) s	(4,23 ± 0,75) s	(4,77 ± 0,58) s	(5,22 ± 0,64) s	(5,65 ± 0,68) s
7	(2,53 ± 0,22) s	(2,63 ± 0,23) s	(2,83 ± 0,25) s	(3,10 ± 0,27) s	(3,38 ± 0,30) s	(3,67 ± 0,32) s	(3,99 ± 0,37) s	(4,29 ± 0,42) s
8	(2,68 ± 0,30) s	(2,72 ± 0,29) s	(2,83 ± 0,30) s	(2,96 ± 0,30) s	(3,12 ± 0,30) s	(3,31 ± 0,32) s	(3,50 ± 0,32) s	(3,69 ± 0,33) s
9	(3,40 ± 0,29) s	(3,41 ± 0,29) s	(3,38 ± 0,49) s	(3,57 ± 0,3) s	(3,70 ± 0,31) s	(3,78 ± 0,31) s	(3,91 ± 0,32) s	(4,04 ± 0,32) s
10	(4,77 ± 0,29) s	(4,79 ± 0,29) s	(4,82 ± 0,30) s	(4,86 ± 0,30) s	(4,95 ± 0,33) s	(5,00 ± 0,29) s	(5,10 ± 0,30) s	(5,21 ± 0,31) s

Příloha D. Procedura NN_LRS

```
CREATE OR REPLACE PROCEDURE NN_LRS(x IN NUMBER, y IN NUMBER, start_time IN
                                          NUMBER, end_time IN NUMBER)
AS
  idx number;
  count_row number:=0;
  v_geom SDO_GEOMETRY;
  v_distance number;
  min_distance number;
  min_idx number;
  isFirst boolean:=true;
  CURSOR geom
      IS
         select l.id_trajectory, SDO_LRS.CLIP_GEOM_SEGMENT( l.trajectory,
               start_time, end_time)
         from lrs_trajectory_tab l, lrs_tsegment_tab t, lrs_region_tab r,
              lrs_mpoint_tab m
         where l.tsegment=REF(t) and t.mpoint=REF(m) and t.region=REF(r)
         and SDO_LRS.CLIP_GEOM_SEGMENT(l.trajectory, start_time, end_time)
              IS NOT NULL;
BEGIN
    OPEN geom;

  LOOP

      fetch geom into idx,v_geom;
      exit when geom%NOTFOUND;
      v_distance := null;

      v_distance := SDO_GEOM.SDO_DISTANCE(v_geom,MDSYS.SDO_GEOMETRY(
                2001, NULL, sdo_point_type(x,y,NULL), NULL, NULL),0.005);
      if isFirst = true then
           min_distance := v_distance;
           isFirst := false;
      end if;
      if v_distance < min_distance then
            min_distance := v_distance;
            min_idx := idx;
      end if;

  END LOOP;

  close geom;
  dbms_output.put_line('nejblizsi trajektorie: '||min_idx);
  dbms_output.put_line('nejmensi vzdalenost: '||min_distance);

END;
```

Příloha E. Procedura NN_3D

```
CREATE OR REPLACE PROCEDURE NN_3D(x IN NUMBER, y IN NUMBER, start_time IN
                        NUMBER, end_time IN NUMBER)
AS
  idx number;
  count_row number:=0;
  v_geom SDO_GEOMETRY;
  v_distance number;
  min_distance number;
  min_idx number;
  isFirst boolean:=true;


  CURSOR geom
      IS
       SELECT t.id_tsegment, SDO_GEOM.SDO_INTERSECTION(
              l.line, SDO_GEOMETRY(3007, NULL, NULL,
              SDO_ELEM_INFO_ARRAY(1,1003,1, 16,1003,1, 31,1003,1,
                 46,1003,1, 61,1003,1, 76,1003,1),
              SDO_ORDINATE_ARRAY(cube_ordinate(start_time, end_time))),1)
       FROM Line_tab3D l, tsegment_tab3D t, mpoint_tab3D mp, region_tab3D r
       WHERE l.tsegment=REF(t) AND t.mpoint=REF(mp) AND t.region=REF(r) AND
              sdo_filter(l.line,
                   MDSYS.SDO_GEOMETRY(3007, NULL, NULL,
                   MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1, 16,1003,1,
                     31,1003,1, 46,1003,1, 61,1003,1, 76,1003,1),
                   MDSYS.SDO_ORDINATE_ARRAY(cube_ordinate(start_time,
                       end_time)))) = 'TRUE';
BEGIN
  OPEN geom;
  LOOP
      fetch geom into idx,v_geom;
      exit when geom%NOTFOUND;
      v_distance := null;

      v_distance := SDO_GEOM.SDO_DISTANCE(v_geom,MDSYS.SDO_GEOMETRY(
                 2001, NULL, sdo_point_type(x,y,NULL), NULL, NULL),0.005);

      if isFirst = true then
                   min_distance := v_distance;
                   isFirst := false;
      end if;


      if v_distance < min_distance then
                   min_distance := v_distance;
                   min_idx := idx;
      end if;


  end LOOP;


  close geom;
      dbms_output.put_line('nejblizsi trajektorie: '||min_idx);
      dbms_output.put_line('nejmensi vzdalenost: '||min_distance);


end;
```

Příloha F. Procedura NN_Full

```
CREATE OR REPLACE PROCEDURE NN_FULL(x IN NUMBER, y IN NUMBER, start_time
                                    IN DATE, end_time IN DATE)
AS
  id_t number;
  posX number;
  posY number;
  min_id_t number;
  v_distance number;
  min_distance number;
  isFirst boolean:=true;
  CURSOR geom
      IS
         select t.id_tsegment, p.pos_x, p.pos_y
         from tsegment_tabF t, line_tabF l, point_tabF p, mpoint_tabF m,
              region_tabF r
         where l.tsegment=REF(t) and t.mpoint=REF(m) and t.region=REF(r)
               and (l.start_point=REF(p) or l.end_point=ref(p))
               and p.instant between start_time and end_time;

BEGIN
   OPEN geom;

  LOOP
      fetch geom into id_t, posX, posY;
      exit when geom%NOTFOUND;
      v_distance := null;
      v_distance := sqrt(power((x-posX),2) + power((y-posY),2));
      if isFirst = true then
          min_distance := v_distance;
          isFirst := false;
      end if;
      if v_distance < min_distance then
          min_distance := v_distance;
          min_id_t := id_t;
      end if;

  end LOOP;

  close geom;
  dbms_output.put_line('nejblizsi trajektorie: '||min_id_t);
  dbms_output.put_line('nejmensi vzdalenost: '||min_distance);

end;
```

Příloha G. Procedura NN_Full_G

```
CREATE OR REPLACE PROCEDURE NN_FULL_G(x IN NUMBER, y IN NUMBER, start_time
                                      IN DATE, end_time IN DATE)
AS
  id_t number;
  posX number;
  posY number;
  posX_end number;
  posY_end number;
  min_id_t number;
  v_distance number;
  min_distance number;
  isFirst boolean:=true;
  CURSOR geom
      IS
         select t.id_tsegment, p.pos_x, p.pos_y, pp.pos_x, pp.pos_y
         from tsegment_tabF t, line_tabF l, point_tabF p, mpoint_tabF m,
              region_tabF r, point_tabF pp
         where l.tsegment=REF(t) and t.mpoint=REF(m) and t.region=REF(r)
               and l.start_point=REF(p) and l.end_point=ref(pp)
               and p.instant between start_time and end_time;
BEGIN
    OPEN geom;
  LOOP
      fetch geom into id_t, posX, posY, posX_end, posY_end;
      exit when geom%NOTFOUND;
      v_distance := null;
      v_distance := SDO_GEOM.SDO_DISTANCE(
            MDSYS.SDO_GEOMETRY(2002, NULL, NULL,
                MDSYS.SDO_ELEM_INFO_ARRAY(1,2,1),
                MDSYS.SDO_ORDINATE_ARRAY( posX,posY,posX_end,posY_end)),
            MDSYS.SDO_GEOMETRY(2001, NULL, sdo_point_type(x,y,NULL),
                NULL, NULL),0.005);

      if isFirst = true then
          min_distance := v_distance;
          isFirst := false;
      end if;
      if v_distance < min_distance then
           min_distance := v_distance;
           min_id_t := id_t;
      end if;
  END LOOP;
  close geom;
  dbms_output.put_line('nejblizsi trajektorie: '||min_id_t);
  dbms_output.put_line('nejmensi vzdalenost: '||min_distance);

END;
```

Příloha H. Obsah přiloženého CD

Adresářová struktura přiloženého CD:

- **generátor**
  - **aplikace** – aplikace "Network-Based Generator of Moving Object"
  - **dokumentace** – dokumentace k aplikaci
  - **testovana_mnozina_dat** – soubor s testovanými daty
- **skripty**
  - **create** – kódy pro vytvoření všech databází
  - **insert** – skripty s daty pro vložení do databáze
  - **PHP_skripty** – skripty pro převod testované množiny dat soubor příkazů pro vložení
  - **PL_SQL_skripty** – skript pro generování obdélníku, jež vrací zadaný počet geometrií
  - **testy** – kódy použité při testování
    - **statistiky** – soubory s příkazy pro kapitolu 7.2 Úvodní statistiky
    - **2D_1** – kódy pro experiment 7.4.1
    - **2D_2** – kódy pro experiment 7.4.2
    - **2D_3** – kódy pro experiment 7.4.3
    - **2D_4** – kódy pro experiment 7.4.4
    - **LRS_1** – kódy pro experiment 7.5.1
    - **LRS_2** – kódy pro experiment 7.5.2
    - **All1** – kódy pro experiment 7.6.1
    - **All2** – kódy pro experiment 7.6.2
- **zprava** – text diplomové práce