V Š B – T e c h n i c k á u n i v e r z i t a O s t r a v a
F a k u l t a e l e k t r o t e c h n i k y a i n f o r m a t i k y
K a t e d r a i n f o r m a t i k y

Nástroj pro generování testovacího vytížení běžící na Oracle

Tool for a Database Workload Generation

2012 **Petr Kloza**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Petr Kloza**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Nástroj pro generování testovacího vytížení databáze běžící na Oracle
Tool for a Database Workload Generation

Zásady pro vypracování:

V některých případech je vhodné otestovat fyzický návrh databáze, aby se odhadlo chování databáze při určitém zatížení blízkém ostrému nasazení. K tomuto existuje několik komerčních nástrojů a také jednoduché nástroje zdarma. Cílem této práce je vytvořit nástroj, jenž by umožňoval pro existující schéma databáze vytvořit takovéto vytížení.

Výsledný nástroj bude splňovat následující body:
1. Bude schopen generovat data pro základní datové typy: řetězec, číslo, datum.
2. Bude k dispozici několik různých domén, ze kterých bude možné vybrat pro řetězcové hodnoty: jméno, příjmení, město, ulice, zaměstnání, atd. Celkový počet domén bude přibližně dvacet.
3. Bude možné definovat vlastní doménu.
4. Jednotlivé domény bude možné přiřadit testovacím dotazům.
5. Bude možné nastavit frekvenci spouštění dotazů.
6. Bude možné spustit i uložené procedury a funkce.

Při řešení bude student postupovat v následujících krocích:
1. Analýza podobných nástrojů a jejich možností.
2. Návrh a implementace aplikace.
3. Důkladné otestování a vyhodnocení výsledků.

Seznam doporučené odborné literatury:

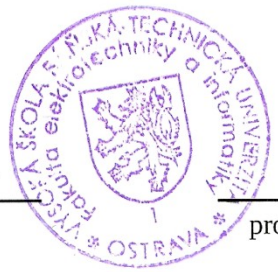
Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Radim Bača, Ph.D.**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

„Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.“

Datum podepsání: 4. 5. 2012 **Podpis:** .....

Abstrakt:

Cílem práce je navrhnout a vytvořit aplikaci schopnou generovat testovací vytížení databáze.
Aplikace bude obsahovat následující prvky:
Generování dat pro základní datové typy (řetězec, číslo, datum) podle přiřazených domén, vytváření vlastních domén, přiřazování domén testovacím dotazům, regulování frekvence spouštění dotazů, možnost spouštění uložených procedur a funkcí na Database.
V první části práce je analyzováno několik podobných již existujících komerčních nástrojů dostupných na internetu.
Druhá část práce je zaměřena na návrh a implementaci funkcí aplikace. Hlavní funkce jsou:
Generování dat do uživatelských tabulek a generování testovacího vytížení.
Třetí část je zaměřena na testování funkčnosti a výkonu aplikace.

Klíčová slova:

generování testovacího vytížení databáze, propustnost, testování výkonu databázového serveru, fyzický návrh databáze, škálovatelnost DB, škálovatelnost HW databázového serveru, uživatelské vytížení, TPS (Transaction per Second), generování dat, hromadné vkládání dat do databáze

Abstract:

The goal is to design and create an application capable of generating load test database.
The application will contain the following elements:
Generating data for basic data types (string, number, date) in accordance with assigned domains, creating custom domains, assigning domain to testing queries, seting up the frequency of running queries, ability to run stored procedures and database functions.
The first part is analyzed several similar existing commercial tools available from the Internet.
The second part focuses on the design and implementation of application functions. The main functions are: Generating data into user tables and generating a test workload.
The third part is focused on the testing of the functionality and performance of the application.

Keywords:

generating the database workload test, throughput, performance analysis of the database server, physical database design, scalability of DB, database server hardware scalability, user load, TPS (Transaction per Second), generating data, bulk loading data into database

Seznam použitých zkratek, značek a definovaných pojmů

ANSI	American National Standards Institute (instituce vyvíjející benchmarky)
AS3AP	Benchmark od ANSI, testuje výpočetní výkon.
BF	Benchmark Factory (nástroj pro generování testovacího vytížení databáze)
C#	Programovací jazyk pro prostředí .NET Framework
CPU	Central Processing Unit (procesor)
DB	Databáze
DML	Data Manipulation Language (patří pod SQL… updates, deletes and inserts data)
FK	Foreign Key (cizí klíč tabulky – umožňuje referenci na jinou tabulku)
GB	Gigabyte (jednotka kapacity, 1 GB = 10^9 bitů)
GUI	Graphical User Interface (grafické uživatelské rozhraní)
HW	Hardware (fyzická (hmotná) složka počítače)
I/O	Input/Output (Vstup/Výstup)
kB	kilobyte (jednotka kapacity, 1 kB = 1000 bitů)
MB	Megabyte (jednotka kapacity, 1 MB = 10^6 bitů)
min	Podle kontextu: min – minuta (jednotka času, 1 min = 60 s) nebo min – minimum
MS	Microsoft (společnost zabývající se vývojem softwaru)
MVC	Model View Control (návrhový vzor oddělení dat, GUI a uživatelského vstupu)
ns	nanosekunda (jednotka času, 1 ns = 10^{-9} s)
OLTP	Online Transaction Processing (metodika zpracování transakcí pro aplikace)
OS	Operační systém
pdf	Portable Document Format (formát dokumentu)
PFK	Primary Foreign Key (primární a zároveň cizí klíč tabulky)
PK	Primary Key (primární klíč tabulky, podle PK jsou vyhledávána data tabulky)
RAC	Real Application Clusters (SW pro klastrování databáze a zvýšení dosažitelnosti)
RAM	Random Access Memory (paměť s náhodným přístupem – operační paměti)
s	sekunda (základní jednotka času)
SQL	Structured Query Language (programovací jazyk využíván DB systémy)
SŘBD	Systém řízení báze dat
SW	Software (programová (nehmotná) část počítače)
TMP	Transaction per Minute (provedené transakce za minutu)
TPC	Transaction Processing Council (viz 2. Analýza vybraných komerčních nástrojů..)
TPS	Transaction per Second (provedené transakce za sekundu)
txt	Text File
vpn	virtual private network (virtuální privátní síť)
XML (xml)	eXtensible Markup Language (meta-značkový jazyk)

Značky v tabulkách a definované pojmy

●	ano (daný element splňuje (popř. obsahuje) daný parametr)
—	ne (daný element nesplňuje (popř. neobsahuje) daný parametr)
○	částečně daný element splňuje (popř. obsahuje) daný parametr

Unique Member: Označení atributu podílejícího se na unikátnosti klíče. Kombinace dat těchto atributů jednoho záznamu musí být jedinečná v rámci tabulky, ostatní atributy PK mohou mít náhodná data.

Unique Members: Pole atributů podílejících se na unikátnosti klíče.

Obsah

1. Úvod

V praxi se lze setkat s informačním systémem, který nepracuje výkonnostně dle možností databázového serveru. Důvodů, proč například vykonávání transakcí trvá delší dobu, může být hned několik. Jedním z důvodů může být špatný fyzický návrh databáze. Takovým problémům je vhodné předcházet. Nejlepším způsobem, jak otestovat databázový server, zda pracuje správně je simulovat vysoké uživatelské vytížení blízké ostrému nasazení různými způsoby (benchmarky).
Benchmark obecně označuje počítačový program (nebo jeho část) sloužící k dosažení nějakých charakteristik výkonu testovaného objektu. [5]
Benchmarky slouží například k testům propustnosti databázového serveru, škálovatelnosti atd. Výsledky testů poskytují programátorovi (vývojáři) informace potřebné k optimalizaci fyzického návrhu databáze, optimalizaci SQL dotazů informačního systému, případně posílení hardwaru databázového serveru atd.
Fyzický návrh databáze se týká například použitých typů tabulek (typ tabulky specifikuje způsob uložení dat), indexů, datových typů atd. [1]

1.1. Stručný přehled jednotlivých kapitol

Stručný popis kapitol obeznamuje čtenáře se strukturou práce a obsahem jednotlivých kapitol.

1 Úvod
Úvod se snaží přiblížit téma práce a objasnit řešenou problematiku.

2 Analýza vybraných komerčních nástrojů pro generování testovacího vytížení databáze
Tato kapitola se zaměří na analýzu nástrojů generujících testovací vytížení, čtenář se obeznámí s jednotlivými nástroji s jejich přednostmi i chybami.

3 Analýza aplikace
Kapitola obsahuje stručný přehled funkcí aplikace, návrh knihoven a návrh tříd TableStructure a Profile.

4 Technické prvky ke generování dat
Význam domén a dataSetu je přiblížen v této kapitole.

5 Generování náhodných dat
Generování náhodných dat je prováděno pomocí dvou způsobů, které jsou v této kapitole popsány. Jedná se o generování náhodných dat pomocí regulárních výrazů a generování náhodných dat z textových souborů.

6 Generování a hromadné vkládání dat do databáze
Kapitola rozebírá funkcionalitu generování dat a následného vkládání do databáze.

7 Generování testovacího vytížení
Kapitola věnuje pozornost generování testovacího vytížení (zejména provádění testů propustnosti).

8 Testování aplikace
Testování se zaměřuje na funkcionalitu a výkon aplikace.

9 Závěr
Závěr obsahuje stručné zhodnocení vytvořené aplikace, zhodnocení dosažených výsledků, porovnání s ostatními komerčními nástroji a návrhy pro případný další vývoj.

2. Analýza vybraných komerčních nástrojů pro generování testovacího vytížení databáze

Před započetím samotného vývoje vlastního nástroje pro generování testovacího vytížení databáze nad uživatelskými tabulkami byl zhotoven přehled o již existujících podobných nástrojích dostupných prostřednictvím internetu. Tyto nástroje byly podrobeny analýze.
Analýza každého nástroje zahrnuje rozbor funkcionality nástroje (spektrum jeho využití), přehled benchmarků (pokud nástroj nějaké má) se zaměřením především na vlastní benchmarky (pro generování testovacího vytížení nad vlastními tabulkami). Dále tato kapitola obsahuje kritiku a srovnání nástrojů, základní informace o nástrojích (cena licence, podporované databáze, podporované OS a tak podobně).
TPC: Transaction Processing Council. Jedná se o neziskovou společenost, založenou za účelem definovat zpracování transakcí a benchmarků, o kterých šíří objektivní a ověřitelná data pro průmysl. [6]

Standardizované TPC Benchmarky:

TPC-B: Provádí zátěžový test databáze. Generuje vysoké vytížení databáze (čtení, zápis, mazání, aktualizace). [7]

TPC-C: Pět souběžných konkurenčních transakcí se snaží pracovat s daty nad nízkým počtem tabulek. Tyto transakce zahrnují zadávání a realizaci zakázek, evidenci plateb, kontrolu stavu objednávek a sledování úrovně zásob na skladech. Počet tabulek pro benchmark TPC-C je stanoven na devět. [8]

TPC-D: Spouští složité SQL dotazy nad rozsáhlou datovou strukturou. [9]

TPC-E: TPC-E benchmark používá databázový model makléřské firmy se zákazníky, kteří vytvářejí transakce týkající se obchodů, informace o účtech, a průzkum trhu. Makléřská firma spolupracuje s finančními trhy, za účelem provádět příkazy jménem zákazníků a aktualizovat příslušné informace o účtu. Zaměření benchmarku je na centrální databázi, která provádí transakce vztahující se k podniku zákaznických účtů. I když základní obchodní model TPC-E je makléřské firmy, tak schéma databáze, data populace, transakce, a prováděcí pravidla jsou navržena tak, aby mohla reprezentovat široké spektrum moderních systémů OLTP. [10]

TPC-H: Benchmark používaný pro business logiku. [11]

2.1. Quest Software – Benchmark Factory 6.6.1

Benchmark Factory vytváří obrovské vytížení databázového systému, kterého je obvykle obtížné dosáhnout ve standardním testovacím prostředí (nízké nároky na HW počítače). Systém se zpravidla zhroutí při extrémním zatížení. Benchmark Factory dopomáhá k vývoji databázového prostředí stanovením systémového výkonu, systémové kapacity (myšleno počtem zpracování transakcí v čase) a slabých míst. [3]
Nástroj zahrnuje testování: TPC benchmarky, škálovatelnost DB, škálovatelnost SQL, škálovatelnost a maximální vytížení HW databázového serveru, škálovatelnost RAC Oracle databází, atd. Díky těmto

testům se vývojáři informačních systémů a jiných aplikací využívajících databáze dozví, zda je databáze dobře škálovatelná (např. při zvyšujícím se počtu připojení), jakou má propustnost atd. Nástroj dále nabízí možnost načíst a pracovat s Trace Files databáze, vytvoření vlastního benchmarku s vlastními SQL transakcemi, možnost náhodného generování dat pro vkládání či aktualizace. [3] Zkušební verzi nástroje lze stáhnout na adrese: http://www.quest.com/benchmark-factory/.

2.1.1. Krátký přehled nástroje

Systém	Windows 2000, Windows 2003, Windows 2008
Podporované databáze	Oracle: 8, 8i, 9i, 9.2, 10g, 11g; SQL Server: 7, 2000,2005, 2008; Sybase: 12.5, 15; DB2 LUW: od 8.1.5 do 9; MySQL: 4.5, 5.x
Vyžadovaný Software:	Microsoft .NET Framework 2.0
Licence	Shareware (3742,55 €)

2.1.2. Popis funkcionality

- Určení propustnosti systému a kapacity databázových systémů. [3]
- Simulace tisíců souběžných uživatelů s minimálním množstvím hardwaru. [3]
- Určí, zda aplikace je dobře škálovatelná, při zvyšujícím se počtu uživatelů. [3]
- Hledání bodu zlomu, slabých nebo úzkých míst systému. [3]
- Kvantifikování aplikace nebo výkonu serveru. [3]

2.1.3. Přehled standardních benchmarků

Standardní benchmarky využívané analyzovanými nástroji jsou benchmarky z rodiny ANSI a benchmarky z rodiny TPC.

Benchmarky z rodiny ANSI

AS3AP
Tento Benchmark je od American National Standards Institute (ANSI). [12]
AS3AP Testuje výpočetní výkon.
Je použitelný pro širokou škálu databázových systémů.

Scalable Hardware Benchmark
Tento benchmark je podmnožinou AS3AP. Je určen pro relační databázové systémy. Testuje škálovatelnost CPU, disku, sítě a jejich vzájemné kombinace.

Benchmarky z rodiny TPC
TPC-B, TPC-C, TPC-D, TPC-E, TPC-H (viz 2. Analýza vybraných komerčních nástrojů)

Vlastní Benchmark

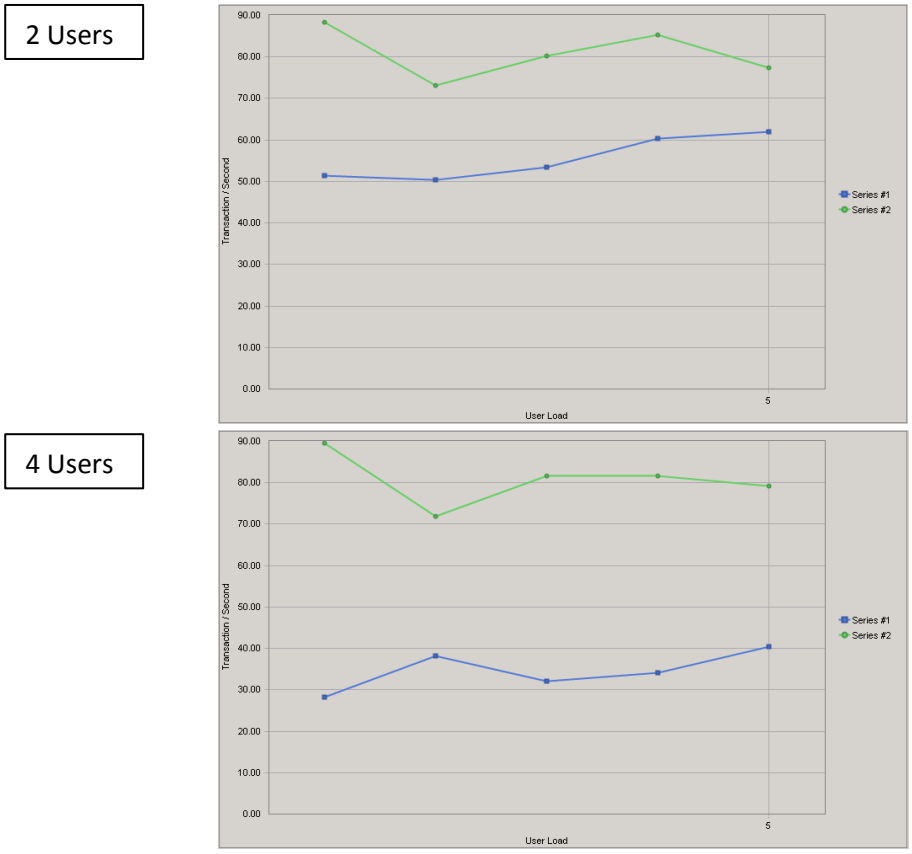
Zahrnuje nastavení počtu virtuálních uživatelů a jejich transakcí. Transakcím lze přiřadit generovaná data (jména, města, emaily…) a nastavit klidovou dobu uživatele po vykonání této transakce. Nelze však nastavit poměr vytížení transakcí (na rozdíl od Swingbench).

2.1.4. Celkové zhodnocení nástroje

+ Široké spektrum testování, široké spektrum podporovaných databází, velký důraz na TPC benchmarky včetně nejnovějších TPC-E a TPC-H, široké spektrum testování škálovatelnosti SŘBD i HW serveru, propustnosti, detekce úzkých míst a opožďování vykonávání transakcí, testování RAC, práce s Trace Files, možnost vytváření vlastních benchmarků s vlastními SQL transakcemi, generování dat do SQL transakcí, relativně nízké požadavky na frekvenci CPU a kapacitu RAM, dlouhodobý vývoj.
– Vysoká cena licence, nemožnost regulovat poměr vytížení transakcí u vlastního benchmarku na úrovni uživatele, nutnost instalace Statspack (bez nainstalovaného Statspack nemůže běžný uživatel provádět sebemenší testy nad Oracle databázemi).

2.1.5. Ukázka TPS grafů testování nad vlastním schématem

Obr. 1 TPS grafy Benchmark Factory



Obr. 1 znázorňuje grafy testů propustnosti provedených na školní výukové databázi VŠB-TUO (dbedu.cs.vsb.cz) v čase mezi 20:10 až 20:30 dne 12. 12. 2011 přes připojení k virtuálnímu serveru vsrva1028qedu-017.vsb.cz. Jelikož v této době nebyl zamezen veškerý uživatelský přístup do databáze, tak tyto testy nevypovídají o skutečných charakteristikách databáze ani nemohou být

vzájemně srovnávány. Stejné testy byly toho dne provedeny vícekrát s dosti rozdílnými výstupy i při stejném počtu výchozích záznamů.
Příčinou těchto rozdílů je především uživatelská aktivita.
Grafy slouží pouze pro demonstraci jedné z výstupních charakteristik vlastního benchmarku. Testy byly provedeny nad tabulkami s nízkým počtem záznamů, průměrně bylo naměřeno 80 transakcí za sekundu.

2.2. Dominic Giles – Swingbench 2.3

Swingbench je bezplatná aplikace, kterou lze demonstrovat a otestovat tyto technologie: Real Application Clusters (RAC), Online table processing (OLTP), Standby databáze (kopie databáze, kterou lze synchronizovat s primární databází), atd. [13] Další produkty jako Datagenerator, Trace Analyzer apod. najdete na http://www.dominicgiles.com.
Uživatelsky přívětivé prostředí, 4 různé benchmarky, možnost vytvoření vlastního benchmarku s vlastními SQL transakcemi (lze editovat předpřipravené transakce přímo na databázi, anebo zakomponovat vlastní transakce v jazyce Java), konfigurační soubory k benchmarkům psány v xml – snadná editace, dostačující spektrum využití testování, grafické rozhraní zobrazuje průběh testu v reálném čase.

2.2.1. Krátký přehled nástroje

Systém	Linux (Ubuntu), Windows
Podporované databáze	Oracle 10g, Oracle 11g
Vyžadovaný SW	Java
Licence	Freeware

2.2.2. Popis funkcionality nástroje

Veškerá základní nastavení pro Swingbench: Connection data, počet virtuálních uživatelů, dobu čekání mezi transakcemi, délkou testování, dále informace o transakcích, jejich klíč, lokaci, load ratio (poměr vytížení transakcí), aktivitu, jméno souboru výstupních statistik atd. Veškerá základní nastavení lze přímo v grafickém uživatelském rozhraní aplikace upravovat a následně uložit do XML souboru. Vhodné je například uložit všechny benchmarky s vlastními hodnotami pro připojení do databáze.

2.2.2.1. Srovnání funkcionality s Benchmark Factory

Aplikace Swingbench umožňuje snadnou editaci benchmarku. Vznikne tak vlastně nový benchmark, který může být nápomocen k novému testování z jiného úhlu pohledu. Editací lze upravit i poměr vytížení transakcí.
V Benchmark Factory nelze upravovat poměr vytížení transakcí u vlastních benchmarků. Swingbench nabízí základní otestování databázového serveru a vyhodnocení statistik TPM, počtu DML operací za minutu, škálovatelnosti RAC systémů pomocí Cluster Overview, apod.
Aplikace Cluster Overview je součástí balíčku aplikace Swingbench.
Benchmark Factory nabízí přece jen širší spektrum využití, více benchmarků, větší zaměření na škálovatelnost SŘBD, hledání slabých míst SŘBD, kde může docházet ke globálním chybám nebo ke zpomalování transakcí.

Je třeba zvážit, zda investovat 3742,55 € do Benchmark Factory nebo se spokojit s volně dostupnou aplikací Swingbench.

2.2.3. Benchmarky

Nástroj obsahuje 4 benchmarky. Bližší zkoumání odhalí, zda nástroj obsahuje aspoň základní benchmarky a kterým z rodiny TCP se podobají.
Čísla v závorkách u jednotlivých benchmarků vyjadřují procentuální poměr čtení a zápisu.

OrderEntry (60/40)
Podobný benchmarku TPC-C („like TPC-C“). Pět souběžných konkurenčních transakcí se snaží pracovat s daty nad nízkým počtem tabulek. Transakce často přistupují k datům stejné tabulky. Benchmark je navržen tak, aby se zatěžovala jednotlivá připojení a operační paměť. Před spuštěním je nutné připravit schéma na databázi (tabulkový prostor, tabulky, indexy, data atd.). Aplikace Oewizard spustí automaticky SQL skripty k vytvoření schématu. [4]

Calling Circle (70/30)
Umožňuje maximální zatížení CPU a paměti bez potřeby silného systému. Před každým novým testem je potřeba generovat nová data pro schéma, aby se testy daly srovnávat.
Aplikace umožňuje vytvořit nové tabulkové schéma nebo jen generovat nová data pro další test, popř. smazat tabulkové schéma po ukončení práce. Tento benchmark se podobá benchmarkům typickým pro OLTP aplikace. [4]

Stress tests (50/50)
Jedná se o testy, které generují velké vytížení databáze (čtení, zápis, mazání, aktualizace) s hraničním zatížením operační paměti, disku… Cílem je co největší zatížení OS databázového serveru, jaký nátlak I/O operací systém vydrží, aniž by se zhroutil apod. Tento benchmark nepotřebuje žádné schéma, vše je vytvořeno za chodu aplikace – výborné pro rychlé testy. [4]

DSS Benchmark (100/0)
Read only benchmark (pouze čtení dat z DB). Testuje SQL dotazy typu select nad velkými tabulkami (s velkým počtem atributů a záznamů). [4]

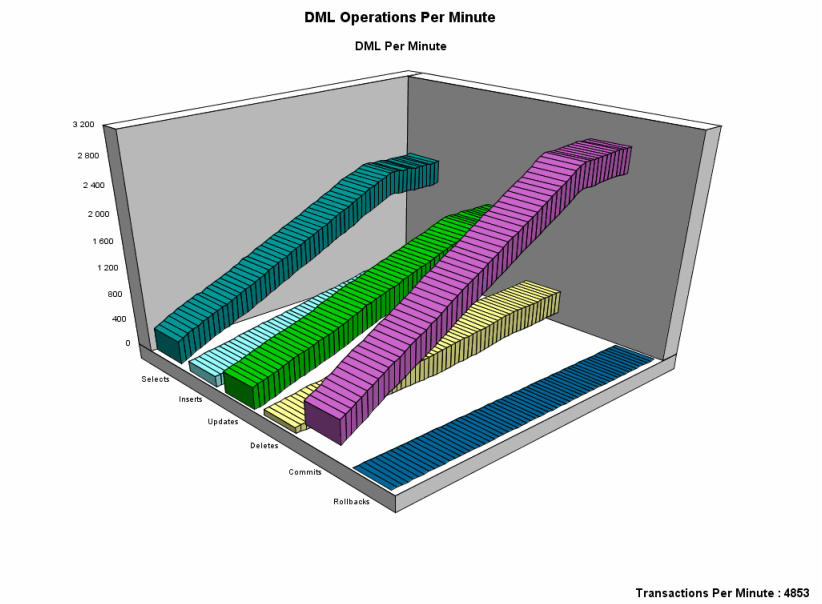
Blank Benchmark
Blank Benchmark využívá uložené procedury a funkce na databázi, a tudíž je pro tuto práci nejpodstatnější. Nejdříve je zapotřebí vytvořit funkce na databázi pomocí skriptu (sql/storedprocedures.sql). Tyto funkce lze pak upravit přímo na databázi například prostřednictvím aplikace Sql Developer. Uživatel tak může provádět operace insert, select, update, delete apod. i nad vlastním schématem. Uživatel může napsat až 6 různých funkcí. [14]
Jediné, co zbývá před zahájením testování, je vygenerovat data do uživatelského schématu. Na to je třeba použít nějaký generátor dat. Je-li maximum šesti transakcí pro uživatele nedostačující, tak je zapotřebí vytvořit (podle předlohy ostatních procedur) vlastní procedury v jazyce Java, které budou spouštět uložené funkce (procedury) na databázi, nastavit cestu k příslušnému jar souboru a ke knihovně obsahující dané procedury. [14]

2.2.4. Celkové zhodnocení nástroje

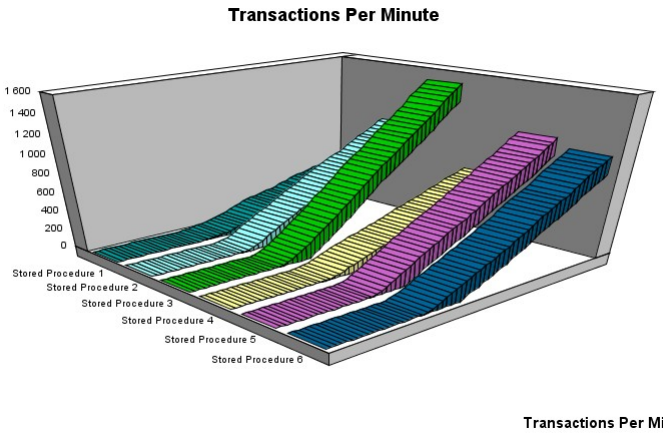
+ Bezplatná licence, podpora pro Windows i pro Linux, nevyžadován další software kromě Javy, uživatelsky přívětivé prostředí, dostačující spektrum využití pro testování, možnost testovat škálovatelnost a propustnost RAC systémů pomocí Cluster Overview, možnost vytvoření vlastního benchmarku s vlastními transakcemi, zobrazování průběhu testu v reálném čase, přívětivý průvodce aplikací (stáhnutelný v pdf na stránkách nástroje), srozumitelná doprovodná videa, relativně jednoduchá příprava jednotlivých benchmarků.
– Podpora pouze pro databáze Oracle (10g, 11g), jistá uživatelská omezení při vytváření „blank“ benchmarku pro vlastní transakce, neobsahuje žádné standardní benchmarky (pouze „like TPC-C“)

2.2.5. Ukázka výstupních charakteristik

Obr. 2 Swingbench – DML operace za minutu



Obr. 3 Swingbench – TPM pro uložené procedury



Průměrný počet transakcí za minutu se v rámci několika testů pohyboval zhruba kolem hodnoty 5300 (což je 88 transakcí za sekundu). Testy propustnosti byly provedeny nad nízkým počtem záznamů a odpovídají zhruba provedeným testům propustnosti pomocí aplikace Benchmark Factory.

2.3. Andreas Frost – DB Load Generator 1.0

Jedná se o velmi jednoduchý nástroj určený pro spouštění vlastních SQL transakcí pro Microsoft SQL Server databáze. Nástroj se hodí k pohotovému, rychlému a snadnému otestování vlastních transakcí s využitím simulace primitivního uživatelského zatížení. http://www.dbloadgenerator.com/
Aplikace nevyžaduje instalaci, spouštění vlastních transakcí.

2.3.1. Krátký přehled nástroje

Systém	Windows XP, Windows Vista, Windows 7
Podporované databáze	Microsoft SQL Server
Vyžadovaný SW	Microsoft .NET 4.0
Licence	Shareware (€10)

2.3.2. Popis funkcionality nástroje

DB Load Generator 1.0 obsahuje pouze vlastní benchmark. Ten umožňuje vytvořit až 8 vlastních SQL příkazů, které jsou paralelně vybírány a opakovaně spouštěny pomocí vláken, které simulují virtuální uživatele.
Aplikace dále nabízí možnost zvolit počet iterací a délku klidové doby vlákna mezi transakcemi. Iterace udává, kolikrát má každý virtuální uživatel transakci vykonat. Nevýhodou je, že nastavená iterace je pro všechny transakce (kteréhokoliv uživatele) stejná, stejně tak klidová doba mezi transakcemi. Poměr vytížení transakcí nelze ani nijak jinak nastavit.
Dále nelze nastavit pro jednotlivé uživatele, které transakce vynechat atp. Vlastní benchmark je tedy hodně strohý v porovnání s konkurencí.

2.3.3. Celkové zhodnocení nástroje

+ Není nutná instalace, jednoduché uživatelské prostředí, spouštění vlastních transakcí.
– Podpora pouze pro SQL Server databáze, bezplatná licence k vyzkoušení omezena na jednu transakci a jednoho uživatele, nemožnost regulovat poměr vytížení transakcí, neobsaženy žádné benchmarky, bez podpory výstupní charakteristiky v podobě grafu, bez oficiální dokumentace.

2.4. Srovnání jednotlivých nástrojů podle vybraných parametrů

Následující tabulka obsahuje krátké shrnutí a závěrečné porovnání analyzovaných nástrojů dle sledovaných parametrů touto prací.

Tabulka:

	Benchmark Factory		Swingbench		DB Load Generator	
Licence (Typ free verze)	Shareware (Trial)	€ 3 742,55	Freeware	€ 0,00	Shareware (Crippleware)	€ 10,00
OS	Windows 2000, 2003, 2008, XP, Vista, 7		Windows, Linux		Windows XP, Vista, 7	
Podporované databázové systémy	Oracle: 8, 8i, 9i, 9.2, 10g, 11g ; SQL Server: 7, 2000, 2005, 2008 ; Sybase: 12.5, 15 ; DB2 LUW: od 8.1.5 do 9 ; MySQL: 4.5, 5.x		Oracle 10g, Oracle 11g		SQL Server	
CPU(GHz)/RAM(GB)	1,0/0,512		n.a./n.a.		n.a./n.a.	
Vyžadovaný SW	MS .NET Framework 2.0		Java		MS .NET Framework 4.0	
Počet benchmarků	7 + vlastní		4 + vlastní		0 + vlastní	
Podpora TPC	●	B,C,D,E,H	●	"C like"	—	
Podpora ANSI	●	AS3AP, Scalable HW	—		—	
Škálovatelnost DB	●		●	RAC systémy	—	
Generování dat do vlastních tabulek	—		●	Dominic Giles - DataGenerator	—	
Generování dat do parametrů testovacích dotazů	●		○	txt soubory, nelze je však propojit přímo s dotazy	—	
Vlastní benchmark	○	nastavit lze pouze delay pro transakce	○	pouze 6 procedur pro všechny vlastní benchmarky	○	bez regulace poměru vytížení transakcí
Uživatelské procedury	●		●		●	
Něco navíc	Záruka dlouhodobého vývoje a kvality		Uživatelsky přívětivé prostředí		Nevyžaduje instalaci, uživatelsky přívětivé prostředí	

2.5. Závěr analýzy

Tato analýza se snažila vystihnout pozitiva i nedostatky jednotlivých nástrojů. Z již vyplynulých skutečností je zřejmé, že většina nejznámějších komerčních nástrojů pro generování testovacího vytížení databáze se nezaměřuje tolik na vlastní uživatelské schéma (viz tabulka). Většina nejznámějších nástrojů se zaměřuje na testování nad tabulkami, které jsou předem definované. Na těch tabulkách jsou pak prováděny testy: škálovatelnost DB, škálovatelnost HW serveru, propustnost serveru – TPS (popř. TPM)… Tyto poznatky byly použity při tvorbě vlastní aplikace.

3. Analýza aplikace

Analýza se soustředí na stručný přehled funkcí aplikace, návrh knihoven a návrh nejpodstatnějších tříd.
Dále je do návrhu začleněn i návrh kompatibility s Oracle a SQL Server databázemi. Kompatibilita není zcela dokončena, aplikace podporuje jen Oracle databáze (dle zadání práce), ale pro pozdější případný vývoj je zhotoven návrhový vzor pro podporu SQL Server databází.

3.1. Návrh funkcí aplikace

Klíčové funkce aplikace:

1. **Generování náhodných dat**
   Tato funkce zprostředkovává generování dat, které je využito v dalších funkcích.
2. **Generování dat a následné hromadné vkládání do databáze**
   Tato funkce se zaměřuje na generování dat do dataSetu [15], který obsahuje tabulky, jejichž struktura odpovídá struktuře tabulek načtených z databáze (ze systémového katalogu), v následující fázi se daný dataSet mapuje na databázi.
3. **Generování testovacího vytížení**
   Tato funkce spouští samotné vytížení, test vytížení měří propustnost databázového serveru (TPM) a také škálovatelnost propustnosti se zvyšujícím se počtem připojení.

3.1.1. Generování náhodných dat

Generování náhodných dat generuje náhodná data pro hromadné vkládání do databáze a také generuje data pro vázané proměnné v SQL příkazech (procedurách) účastnících se testovacího vytížení.

Funkce využívá:

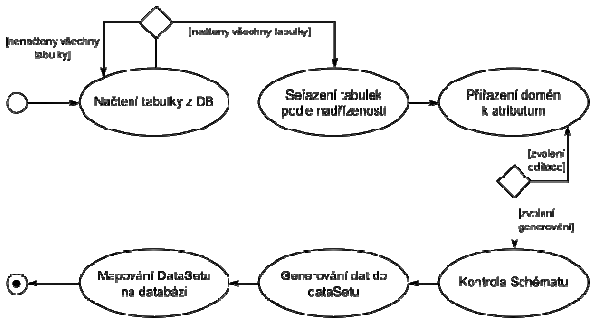
1) Textové soubory
Textové soubory obsahují textové řetězce (jeden textový řetězec na řádek). Z těchto textových řetězců se při generování dat vybere jeden textový řetězec, který je pak převeden na příslušný typ definovaný příslušnou doménou (viz 4.1.1.).

2) Regulární výrazy
Regulární výrazy slouží pro generování textových řetězců (viz 6.1.). Generátor regulárních výrazů vygeneruje náhodně jeden textový řetězec a převede jej na typ definovaný příslušnou doménou.

3.1.2. Generování dat a následné hromadné vkládání do databáze

V první fázi, před započetím samotného generování testovacího vytížení, je nutné vygenerovat data do tabulek na databázi, aby mělo testování nějaký smysl.

Obr. 5 Diagram aktivit generování dat a následného hromadného vkládání do databáze:

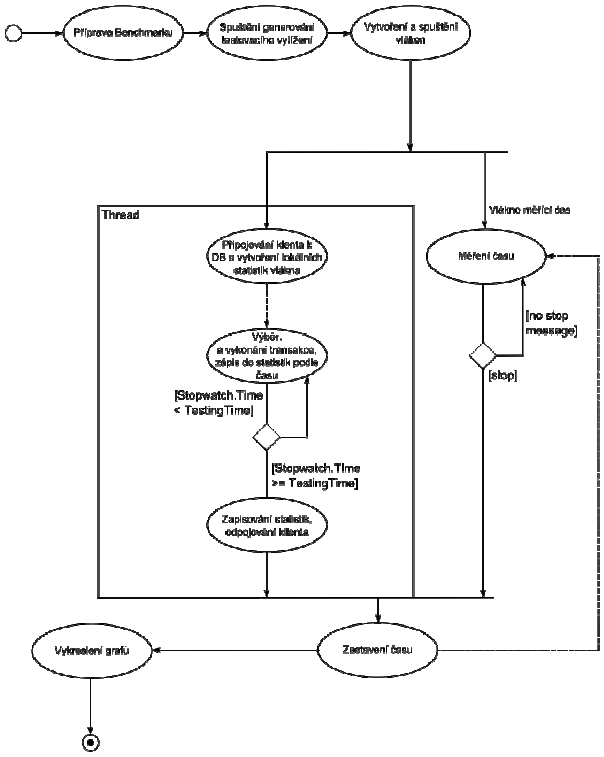


- Načtení tabulky z DB: Načtení jména tabulky, atributů tabulky, primárních a cizích klíčů do interního seznamu zpracovávaných tabulek.
- Seřazení tabulek podle nadřízenosti: Tabulky v seznamu jsou seřazeny podle relací: nadřízená tabulka (rodičovská tabulka) je umístěna v seznamu výše než tabulka, která na ni referuje. Aplikace kontroluje, zda neexistuje cyklus vzájemných referencí. Data jsou pak generována do tabulek v tomto pořadí. Tento postup zajistí generování platných cizích klíčů.
- Přiřazení domén k atributům: V této fázi uživatel přiřadí základní domény atributům tabulek (včetně atributů cizího a primárního klíče). Dále uživatel zvolí počet řádků k vygenerování do jednotlivých tabulek.
- Kontrola schématu: Uživatel zvolí „Generování dat“ a provede se kontrola schématu: Kontrola schématu zahrnuje kontrolu přiřazených domén atributům primárního a cizího klíče.
- Generování dat do dataSetu: Data jsou generována do objektu typu `DataSet` (viz 4.2.), jehož tabulková struktura je vytvořena podle seznamu načtených tabulek.
- Mapování dataSetu na databázi: Mapování dataSetu na databázi je poslední fáze generování dat. Data se zapíší do databáze.

3.1.3. Generování testovacího vytížení

Na základě definovaného vytížení jsou vytvořeni virtuální klienti, kteří pak spouští náhodně vybrané transakce. Na obrázku 6 je uveden diagram aktivit testovacího vytížení tak, jak jednotlivé aktivity postupují.

Obr. 6 Diagram aktivit generování testovacího vytížení



Detailnější popis generování testovacího vytížení (viz 7.).

3.2. Schéma knihoven

Schéma knihoven bylo vytvořeno následujícím způsobem:

V Knihovně `Models` jsou jen třídy, obsahující převážně pouze data, se kterými aplikace pracuje a které jsou ukládány do projektových souborů.

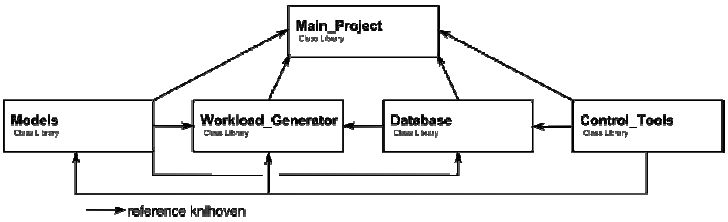
Knihovna `Workload_Generator` obsahuje převážně jen statické třídy s metodami, které s těmito daty pracují.

Knihovna `Control_Tools` obsahuje pomocná data a metody, které mohou používat všechny ostatní knihovny.

V knihovně `Database` jsou třídy používané ke komunikaci s databází – připojení a vykonávání operací na databázi.

Knihovna `Main_Project` je startovním projektem (Startup Project). Obsahuje `Windows Forms`. Řeší jednak grafické zpracování aplikace (zobrazení dat získaných z modelů) a jednak všechny události způsobené uživatelskou aktivitou – klikání myší, vstup z klávesnice apod. Reference mezi těmito knihovnami jsou zobrazeny v Obrázku 7.

Obr. 7 Schéma knihoven



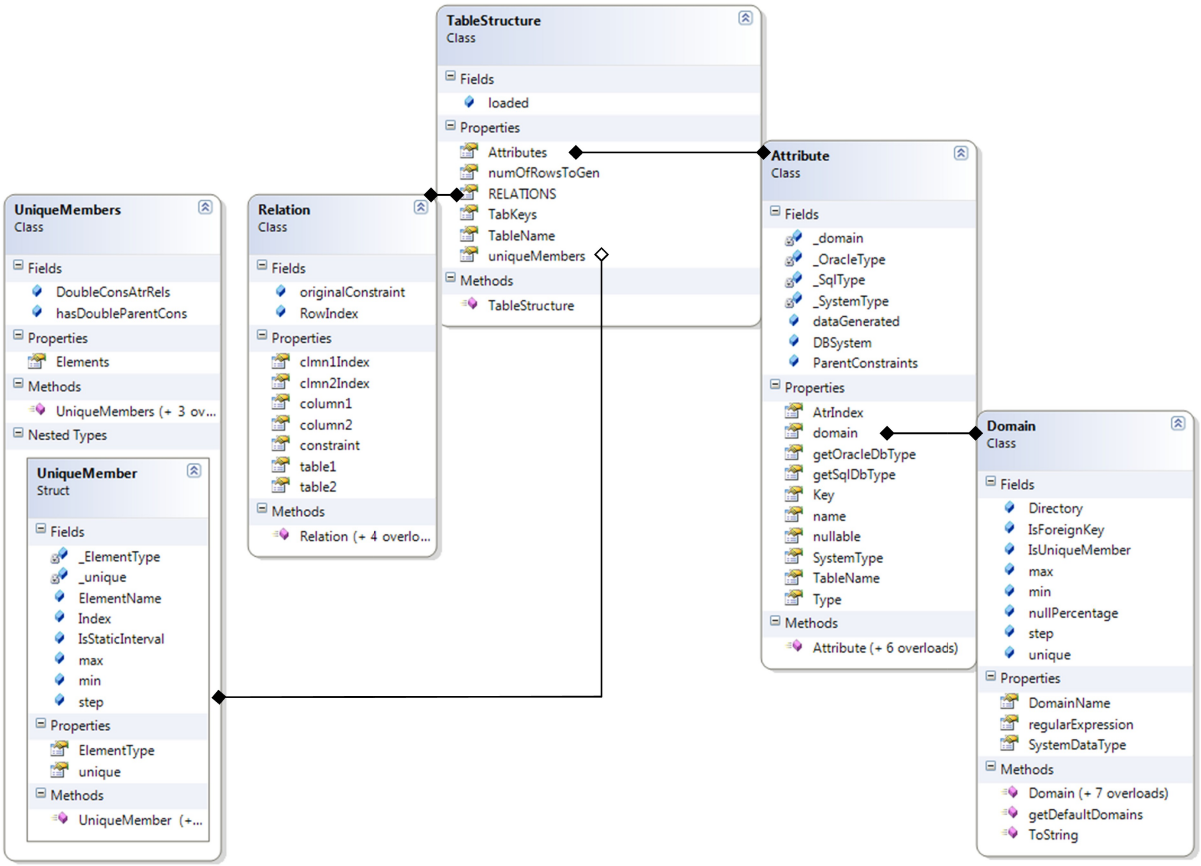
3.3. Návrh tříd

Návrh tříd se soustředí na třídu `TableStructure`, která představuje objektovou reprezentaci databázové tabulky a na třídu `Profile`, která obsahuje veškerá data daného profilu.
Důležité je navrhnout třídy jednotlivých knihoven a utřídit jednotlivá data a metody do těchto tříd podle logických souvislostí.

3.3.1. TableStructure (knihovna Models)

`TableStructure` je třída reprezentující konkrétní databázovou tabulku.
Třídní diagram `TableStructure` spolu s jednotlivými komponentami je zobrazena v třídním diagramu (Obr. 8).

Obr. 8 Třídní diagram `TableStructure` s jednotlivými komponentami



3.3.2. Ostatní třídy z knihovny Models

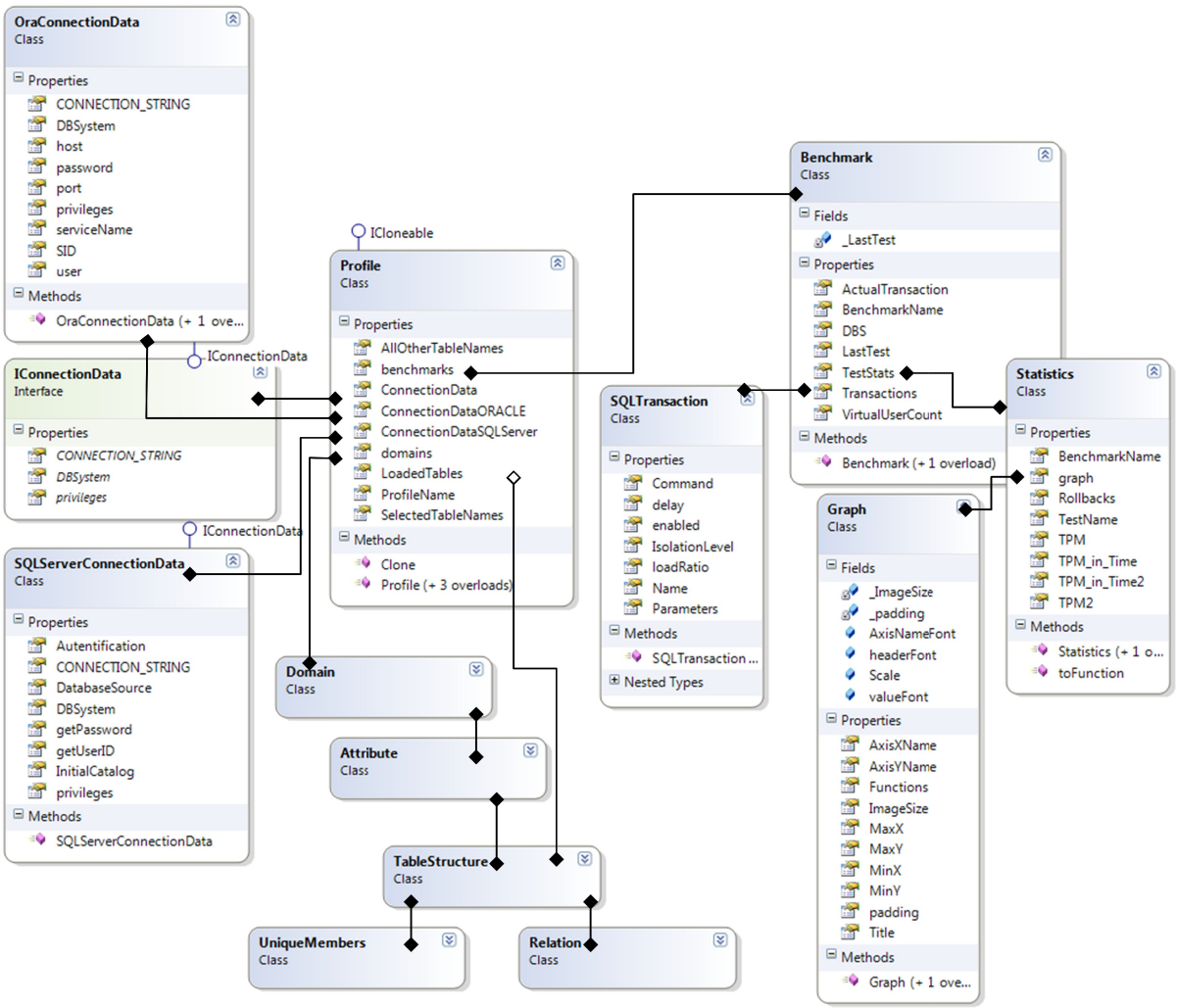
Třída `Profile` obsahuje data potřebná pro připojení se k databázi, seznam tabulek (seznam objektů typu `TableStructure`) a data o jednotlivých benchmarcích. Seznam profilů tvoří projekt. Všechny komponenty třídy `Profile` musí být serializovatelné (uložitelné).

Třída `Benchmark` obsahuje počet virtuálních uživatelů, seznam transakcí a testovací statistiky. Pro snazší přístup k jednotlivým transakcím a benchmarkům jsou využity property.

Třída `SQLTransaction` reprezentuje transakci a její atributy.

V následujícím třídním diagramu jsou zobrazeny všechny třídy knihovny `Models`.

Obr. 9 Třídní diagram knihovny `Models`



3.3.3. Třídy ostatních knihoven

Knihovna `Workload_Generator` obsahuje především statické třídy pracující s daty tříd knihovny `Models`. Nejdůležitějšími třídami jsou `DataGenerator`, který obstarává generování dat do databáze, `LoadGenerator`, který obstarává generování testovacího vytížení a `RegexGenerator`, který umožňuje generování textových řetězců podle regulárního výrazu.

Knihovna `Control_Tools` obsahuje: Vlastní výjimky (`WGException`, `ExceptionData`). Dále obsahuje třídu `AppSupports`, která definuje podporované databázové datové typy, jejich převody na systémové datové typy a maximální možný počet záznamů ke generování do dataSetu.
Knihovna `Database` obsahuje třídy zprostředkovávající komunikaci s databází: `DbClient`, `OracleClient` a `SQLServerClient`. Dále obsahuje třídu `Rollback`, která se používá pro zápis chyby v prováděné transakci při generování testovacího vytížení.

3.4. Kompatibilita s Oracle a SQL Server databázemi

Kompatibilita využívá knihovnu Oracle.DataAccess.Client.
http://www.oracle.com/technetwork/database/windows/downloads/index-101290.html
Kompatibilita není zcela dokončena. Aplikace dle zadání bakalářské práce přednostně podporuje databázový systém Oracle, ale pro pozdější případný vývoj je komunikace s databází zobecněna.
Kompatibilita dále zahrnuje seznam podporovaných datových typů pro Oracle i pro SQL server a zohledňuje různý formát vázaných proměnných. Zbývá už jen vyřešit zejména odlišnosti v systémových katalozích.

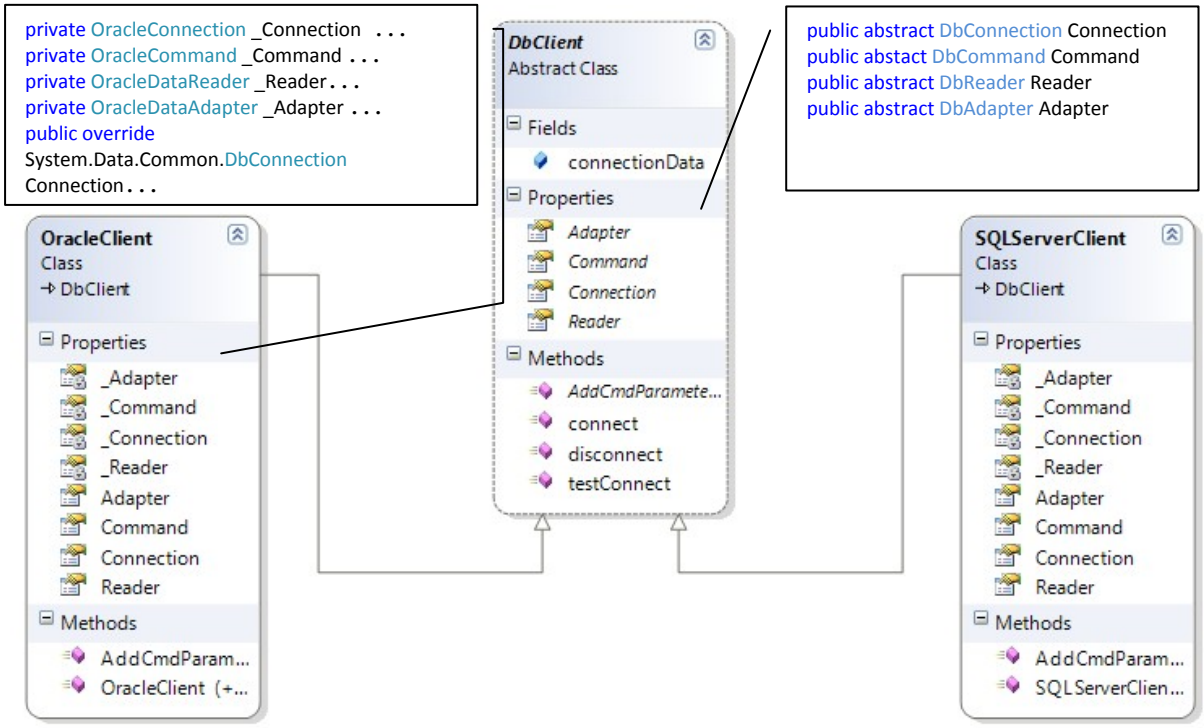
3.4.1. Propojení s databázemi Oracle a SQL Server

Kompatibilita vychází z toho, že objekty pro práci s DB (`Adapter`, `Command`, `Connection` a `Reader`) pro Oracle i pro SQL Server jsou potomci stejných abstraktních tříd, které obsahují metody a proměnné, ze kterých tito potomci dědí.
Byla vytvořena abstraktní třída `DbClient`, která deklaruje všechny objekty pro práci s databází najednou. Všechny třídy aplikace pracující s databází využívají tuto abstraktní třídu, která může skrývat objekt typu `OracleClient`, anebo `SQLServerClient` – z abstraktní třídy ani z rozhraní nelze vytvořit instanci, proto informace o datovém typu zůstane zachována. Stejná metoda dvou objektů (různého typu), které byly přetypovány na `DbClient` může vykonávat různé operace, pokud původní datové typy jsou odlišné.
Tento jev je nazýván v informatice nazýván polymorfismus. [16]

3.4.1.1. Abstraktní třída DbClient

DbClient je abstraktní třída, která zprostředkovává komunikaci s různými databázemi. Třída aplikaci poskytuje jednotné připojení k databázi.

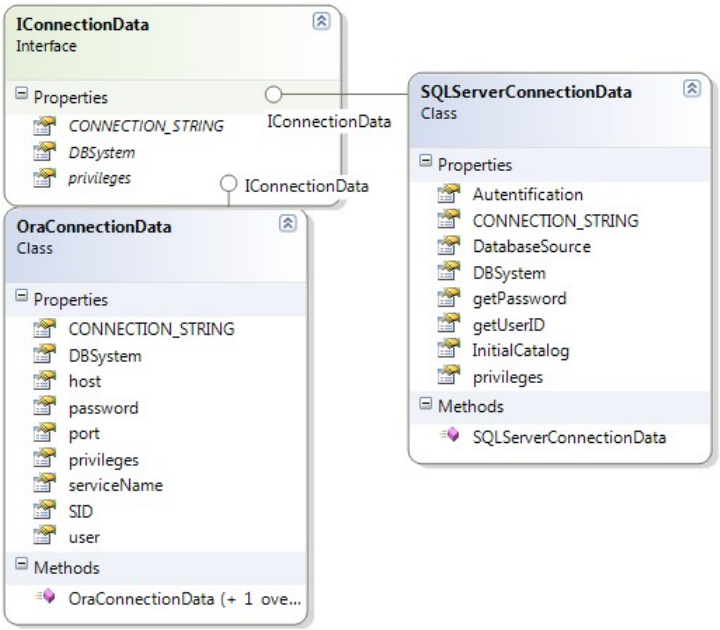
Obr. 10 Třídní diagram DbClient s potomky této třídy



3.4.1.2. Rozhraní IConnectionData

Abstraktní třída `DbClient` využívá rozhraní typu `IConnectionData`. Tomuto rozhraní se přiřadí datový typ buď `OracleConnectionData`, anebo `SQLServerConnectionData`. Rozhraní `IConnectionData` deklaruje properties, které využívají třídy implementující toto rozhraní – přihlašovací údaje (CONNECTION_STRING – datový typ: string), název databázového systému (DBSystem – datový typ: string) a uživatelská práva (privileges – datový typ: string).

Obr. 11 Třídní diagram (`IConnectionData` a třídy implementující toto rozhraní)



3.4.2. Seznam podporovaných datových typů

Seznamy podporovaných datových typů:

- Seznam podporovaných datových typů pro Oracle
- Seznam podporovaných datových typů pro SQL Server
- Seznam podporovaných datových typů primárních klíčů pro Oracle
- Seznam podporovaných datových typů primárních klíčů pro SQL Server.

Každý z těchto seznamů slouží také pro převod databázového datového typu na systémový.
Seznamy jsou statické a obsahují KeyValuePair<*TKey*, *TValue*> elementy. *TKey* představuje datový typ mapující klíč na hodnotu. *TValue* představuje systémový datový typ (obecně System.Type).
Tyto seznamy jsou obsaženy ve třídě `AppSupports`, která je využívá ve svých metodách, které slouží především k ověření, zda dané SŘBD podporuje daný databázový datový typ a jakému systémovému datovému typu odpovídá. Dále slouží ke zjištění toho, zda datový typ primárního klíče je podporován či nikoliv pro dané SŘBD. Tyto metody jsou využity při načítání tabulek a při přiřazování domén atributům.

Podporované datové typy pro Oracle a SQLServer:

OracleDbType	Systémový typ	Použitelný jako klíč
Varchar2	System.String	●
NVarchar2	System.String	●
Char	System.String	●
NChar	System.String	●
Byte	System.Byte	—
Int16	System.Int16	—
Int32	System.Int32	●
Int64	System.Int64	●
BinaryFloat	System.Single	—
BinaryDouble	System.Double	—
TimeStamp	System.DateTime	●
SqlDbType	**Systémový typ**	**Použitelný jako klíč**
Varchar	System.String	●
NVarchar	System.String	●
Char	System.String	●
NChar	System.String	●
SmallInt	System.Int16	—
Int	System.Int32	●
BigInt	System.Int64	●
Float	System.Single	—
Time	System.TimeSpan	—
DateTime	System.DateTime	●
DateTime2	System.DateTime	●

3.4.3. Vázané proměnné

Odlišný formát vázaných proměnných je vyřešen pomocí funkce `if`, která se používá téměř vždy při změně textu příkazu.
Pravdivostní výraz: IConnectionData.DBSystem == „ORACLE“

Formát vázaných proměnných v SQL příkazu
Oracle: :par
SQL Server: @par

4. Technické prvky ke generování dat

V jednotlivých funkcích jsou využity základní technické prvky pro generování dat, jmenovitě: Doména, DataSet.

4.1. Domény

Domény jsou klíčovým prvkem určujícím generovaná data, mají svůj předpis a datový typ.

4.1.1. Systémový datový typ domény

Systémový datový typ domény určuje typ generovaných dat.
Musí souhlasit se systémovým datovým typem atributu, ke kterému je doména přiřazena, aby bylo možné generovat a hromadně vkládat data do databáze.
Aplikace nedovolí přiřadit atributu doménu s odlišným systémovým datovým typem kromě domény `Default`. Doména `Default` má generický datový typ. Tuto doménu není dovoleno mazat ani editovat. U domén přiřazených testovacím dotazům není kontrolován typ domény.

4.1.2. Předpis domény

Předpis domény:

- Doména určená pro generování primárních klíčů (viz 6.1.)
  - Předpis pro generování unikátních dat
  - Předpis pro *UniqueMember* atribut
  - Předpis pro generování unikátních dat pro PFK atributy
  - Předpis pro *UniqueMember* PFK atributy
- Doména určena pro generování cizích klíčů
  - Předpis pro FK atributy
- Doména určená pro generování náhodných dat (viz 5.)
  - Předpis: Regulární výraz
  - Předpis: Textový soubor

Aplikace umožňuje uživateli nadefinovat doménu s vlastním předpisem..
Uměle vytvořené constrainty nesou název tohoto formátu: <TableName>_FK<index>.

4.2. Objekt DataSet

`DataSet` je objekt v programovacím jazyce C# z knihovny `System.Data`, který reprezentuje tabulky a jejich data. Vytvořenou instanci typu `DataSet` pak lze mapovat na databázi. [16]
Generování velkého počtu dat pro následné hromadné vkládání do databáze je poměrně časově náročná záležitost, proto aplikace ukládá `DataSet`, který vytvořila při generování do příslušného adresáře „DataSets“ pro pozdější opětovné použití.
`DataSet` je ukládán ve formátu XML.

Obr. 12 `DataSet` s tabulkou BEACH_PLAYER zapsaný do XML souboru

```
<?xml version="1.0"?>
<DataSet>
  <xs:schema id="ds" xmlns="" xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
    <xs:element name="ds" msdata:IsDataSet="true" msdata:UseCurrentLocale="true">
      <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
          <xs:element name="BEACH_PLAYER">
            <xs:complexType>
              <xs:sequence>
                <xs:element name="ID" type="xs:string" minOccurs="0" />
                <xs:element name="JMENO" type="xs:string" minOccurs="0" />
                <xs:element name="PRIJMENI" type="xs:string" minOccurs="0" />
                <xs:element name="POHLAVI" type="xs:string" minOccurs="0" />
                <xs:element name="DATUM_REGISTRACE" type="xs:dateTime" minOccurs="0" />
                <xs:element name="TOUR_POINTS" type="xs:int" minOccurs="0" />
                <xs:element name="EMAIL" type="xs:string" minOccurs="0" />
                <xs:element name="DATUM_AKTUALIZACE" type="xs:dateTime" minOccurs="0" />
              </xs:sequence>
            </xs:complexType>
          </xs:element>
        </xs:choice>
      </xs:complexType>
    </xs:element>
  </xs:schema>
  <diffgr:diffgram xmlns:msdata="urn:schemas-microsoft-com:xml-msdata" xmlns:diffgr="urn:schemas-
microsoft-com:xml-diffgram-v1">
    <ds>
      <BEACH_PLAYER diffgr:id="BEACH_PLAYER1" msdata:rowOrder="0" diffgr:hasChanges="inserted">
        <ID>ID0</ID>
        <JMENO>Kgt</JMENO>
        <PRIJMENI>Aye</PRIJMENI>
        <POHLAVI>žena</POHLAVI>
        <DATUM_REGISTRACE>2004-04-07T12:03:20+02:00</DATUM_REGISTRACE>
        <TOUR_POINTS>765</TOUR_POINTS>
        <EMAIL>lywiiijiv.ulpady@wuiuptafigyf.hsy</EMAIL>
        <DATUM_AKTUALIZACE>2008-02-02T21:46:40+01:00</DATUM_AKTUALIZACE>
      </BEACH_PLAYER>
      <BEACH_PLAYER diffgr:id="BEACH_PLAYER2" msdata:rowOrder="1" diffgr:hasChanges="inserted">
        <ID>ID1</ID>
        <JMENO>Medmqv</JMENO>
        <PRIJMENI>Nwpfa</PRIJMENI>
```

5. Generování náhodných dat

Generování náhodných je využíváno pro generování dat a následné hromadné vkládání do databáze a pro testovácí SQL dotazy použité v rámci testovacího vytížení.
Generování náhodných dat zajišťuje třída `DataGenerator`, která na základě předpisu domény, kterým může být regulární výraz nebo textový soubor, umožňuje generovat data.
Takto generovaná data nejsou unikátní.

5.1. Generování dat z textových souborů

Generování dat z textových souborů je ta nejjednodušší cesta jak generovat data. Prostřednictvím příslušného průvodce obsaženého v aplikaci lze přiřadit doméně uživatelem zvolený textový soubor (dostupný na disku) pro generování dat.
Soubor musí být formátu „txt“ a jednotlivé hodnoty oddělené řádkem. `DataGenerator` přečte soubor, oddělí jednotlivé řádky, které uloží do pole řetězců, ze kterého vybere náhodně vždy jednu hodnotu. V případě potřeby generovat hodnoty pro SQL dotaz, který má odkazovat na již existující záznamy, aplikace umožňuje provést `select` dat referovaného atributu do textového souboru.

5.2. Regulární výrazy

Aplikace má vlastní generátor regulárních výrazů, který využívá jazyk .NET Frameworku pro regulární výrazy. [20]
Aplikace si definuje i dvě vlastní pravidla (viz 5.2.1. Podporovaný jazyk pro regulární výrazy). Dostupné zdroje řeší spíše opačný problém, než je generování řetězců podle regulárního výrazu, a to přiřazení daného řetězce k regulárnímu výrazu, proto bylo zapotřebí vytvořit vlastní generátor regulárních výrazů.

5.2.1. Podporovaný jazyk pro regulární výrazy

Aplikace podporuje jen zlomek pravidel, které definuje oficiální jazyk regulárních výrazů pro prostředí .NET Frameworku.

Přehled všech podporovaných elementů jazyka pro regulární výrazy

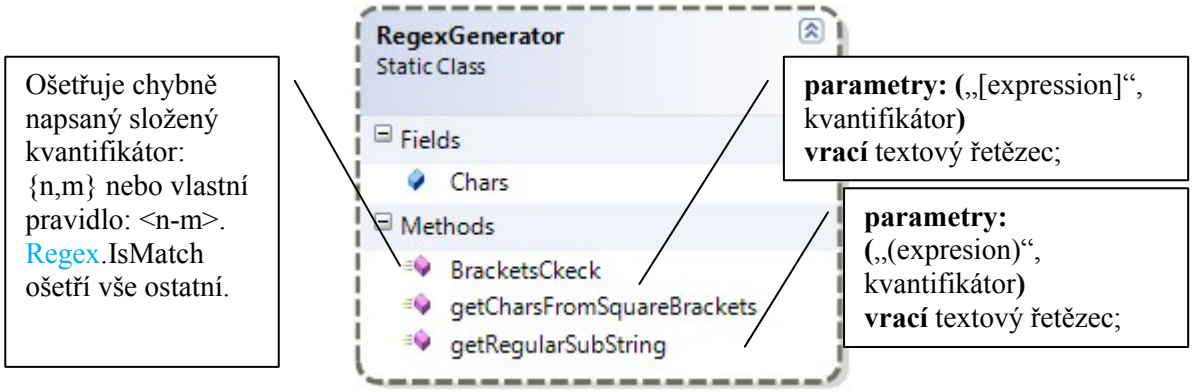
Konstrukce	
\D	- Jedná se o vlastní konstrukci aplikace. Převede hodnotu čísla formátu `Int32` na `DateTime`, který pak převede na `String`.
(subexp)	- Konstrukce umožňuje do regulárního výrazu vnořit jiný regulární výraz (rekurzivním voláním funkce).
[xyzA-Za-z0-9]	- Tato konstrukce vybere vždy jen jeden znak z výčtu znaků (popř. z rozsahu znaků).
<num-num>	- Jedná se o vlastní pravidlo, které vrací číslo z daného intervalu.
(exp1|exp2|exp3)	- Vrátí jeden z podvýrazů oddělených logickým OR.
\	- Následující znak je brán jako obyčejný znak (s výjimkou ’D’).
Kvantifikátory	
{n}	Předchozí element se vyskytne n-krát ve výsledku.

{n,}	Předchozí element se vyskytne nejméně n-krát ve výsledku. Limit: (n; n+10 ⟩.
{n,m}	Předchozí element se vyskytne n-krát, maximálně však m-krát.
*	Předchozí element se může vyskytnout vícekrát ve výsledku, anebo se nemusí vyskytnout vůbec. Limit aplikace: (0; 10 ⟩
?	Předchozí element bude zopakován jednou, anebo ani jednou.
+	Předchozí element se vyskytne ve výsledku aspoň jednou. Limit: (1; 10 ⟩

5.2.2. Generování textových řetězců podle regulárního výrazu

Generování regulárních výrazů má na starosti statická třída RegexGenerator (viz Obr. 13).

Obr. 13 Třída RegexGenerator



Statické pole Chars: Toto pole obsahuje rozsahy číselných a abecedních znaků se zvýhodněním samohlásek pro zvýšení šance výběru samohlásky (výskyt samohlásek je u velkých písmen dvojnásobný a u malých písmen trojnásobný s výjimkou „y“ – výskyt „y“ je jen dvojnásobný).
Písmena i číslice jsou v poli seřazeny přirozeně chronologicky. Pole je využíváno jen tehdy, když aplikace narazí na nějaký rozsah.

Metoda BracketsCheck: Tato metoda slouží ke kontrole validity výrazu ve složených nebo v úhlových závorkách, která nelze provést pomocí odchycení výjimky v metodě `Regex.IsMatch`.

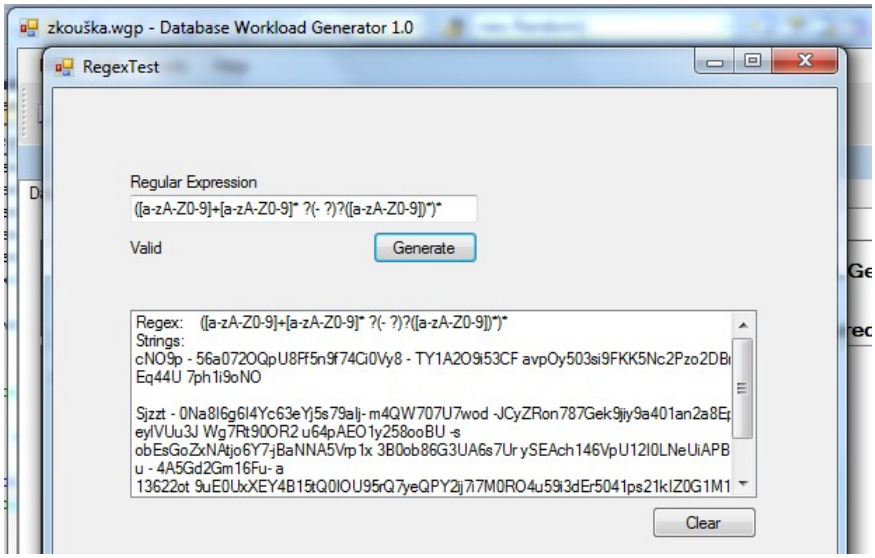
Metoda getCharsFromSquareBrackets:
Tato metoda vrací jeden znak z výrazu uvnitř hranatých závorek. Veškeré vnořené hranaté závorky, okrouhlé závorky, úhlové závorky a všechny ostatní znaky kromě pomlčky jsou považovány za normální znaky.
Pomlčka má význam rozsahu znaků.
Náhodně vybraný znak z rozsahu se uloží do pole možností, stejně jako výčet znaků.
Metodě je předáván i kvantifikátor, který tahle metoda sama rozpozná a podle něj určí počet iterací. Na konci každé iterace do výsledného pole přibude další náhodně vybraný znak.

Metoda getRegularString:

Tato metoda pomocí rekurzivního sestupu vrací textový řetězec, který je vygenerován podle daného regulárního výrazu.
Jednotlivé kroky metody:

- Rozpoznání kvantifikátoru
- Určení počtu iterací na základě kvantifikátoru
- Vrácení textového řetězce vygenerovaného pomocí regulárního výrazu v okrouhlých závorkách.

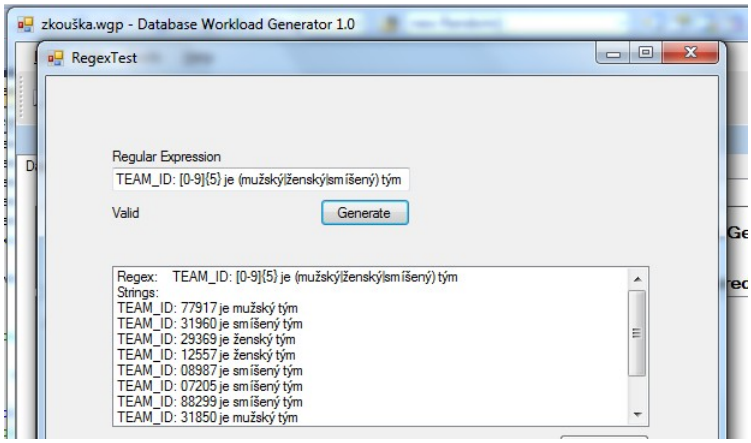
Obr. 14 Generování dat podle regulárního výrazu



5.2.3. Regulární výraz s alternativním výběrem možností

Některý z atributů na databázi, může mít omezení v tom, že může nabývat jen určitých hodnot. Například pohlaví (muž nebo žena), kategorie (mužská, ženská nebo smíšená) a tak dále. V tomto případě je nezbytně nutné použít regulární výraz formátu (exp1|exp2|exp3) – viz 5.2.1. Podporovaný jazyk pro regulární výrazy. Narazí-li metoda `getRegularSubString` na logický OR, tak dosavadně vytvářený textový řetězec, se zapíše do pole možností a pro každý další podvýraz je zavolána rekurzivně metoda `getRegularSubString`.

Obr. 15 Generování kategorie týmu



6. Generování a hromadné vkládání dat do databáze

Hromadné generování a následné vkládání dat do databáze je první krok pro generování testovacího vytížení databáze. Jak nastavit jednotlivé komponenty pro generování dat je popsáno v předchozích třech kapitolách. V případě, že je vše nastaveno správně, začíná samotný proces generování dat a následně se vygenerovaná data mapují na databázi.

6.1. Generování primárních klíčů

Pro každou tabulku obsahující primární klíč je nutné zajistit, aby byl primární klíč (PK) každého ze záznamů jedinečný. Podle PK je totiž záznam vyhledáván.
Generování primárních klíčů, popř. primárních a zároveň cizích klíčů, závisí na způsobu přiřazení domén s určitými předpisy primárním klíčům (viz 4.1.2.).
Generování primárních klíčů by se dalo vyřešit úplně jednoduše tím způsobem, že by každá tabulka obsahující jakkoliv složitý klíč měla vždy jen jeden atribut, kterému by byla přiřazena doména určená ke generování unikátních dat. Tím pádem by se vždy jen jeden atribut podílel na unikátnosti klíče. Taková datová struktura by však nemusela být zajímavá pro některé SQL dotazy využívající několika násobné spojení tabulek na základě cizích klíčů, které jsou zároveň primární. V případě, že by v dotazu konfiguroval takový „PFK“ atribut mající unikátní doménu, pak by byly už ve druhé fázi spojovány tabulky o kvantitě 1 a n. Aplikace proto poskytuje propracovanější generátor primárních klíčů, který bere ohled i na složené primární klíče.
Pokud je zapotřebí k již vygenerovaným datům, vygenerovat data nová, tak je třeba změnit předpis domény nebo přiřadit doménu novou s jiným předpisem, aby nedošlo ke vkládání duplicitního klíče (viz 4.1.2.).

6.1.1. Rozbor technického řešení

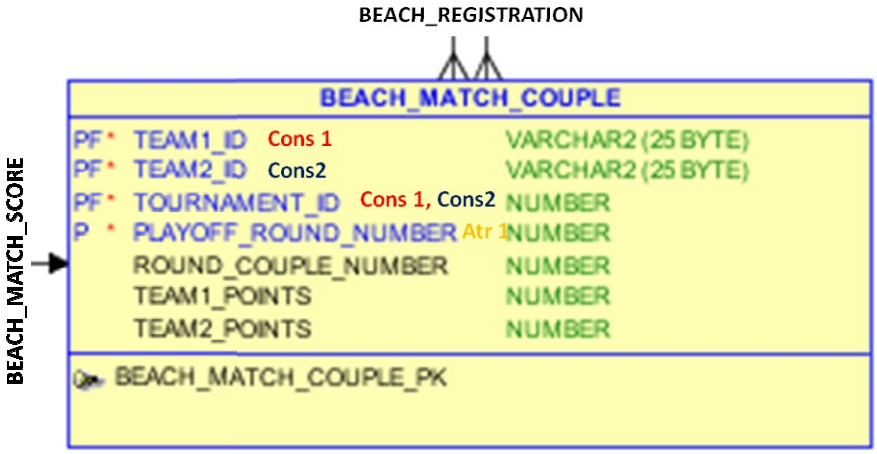
Technické řešení bylo navrženo tak, aby se minimalizovaly nároky na operační paměť a na časovou složitost.
Aplikace si uchovává data o použitých primárních klíčích v paměti. Tato data jsou mimo dataSet a navíc v operační paměti se uchovávají jen data o použitých klíčích pro aktuálně zpracovávanou tabulku – objekt typu `DataTable`.
Způsob uchovávání použitých klíčů v paměti: Pro uchování použitých klíčů si aplikace eviduje jen seznam s názvem *UsedKeys*, který obsahuje pole `Int32` o délce 3. Pro každý záznam právě zpracovávané tabulky si aplikace uchovává pouze 12-ti bytovou informaci o použitém klíči.

Jednotlivé elementy seznamu *UsedKeys* mohou reprezentovat:

a) 3 hodnoty náležící až třem intervalům domén přiřazených příslušným atributům podílejících se na unikátnosti klíče (přičemž každá z těchto hodnot daného atributu se může opakovat, ale kombinace dvojic popřípadě trojic je unikátní).
b) 3 hodnoty mohou náležet až 3 indexům záznamů referovaných tabulek, na které odkazují referenční „constrainty“ (constrainty pro cizí klíče). [19] Každý z indexů daného „constraintu“ se může opakovat, ale kombinace dvojic (popř. trojic) je unikátní),
c) Kombinaci hodnot intervalů domén atributů PK a indexů „constraintů“.

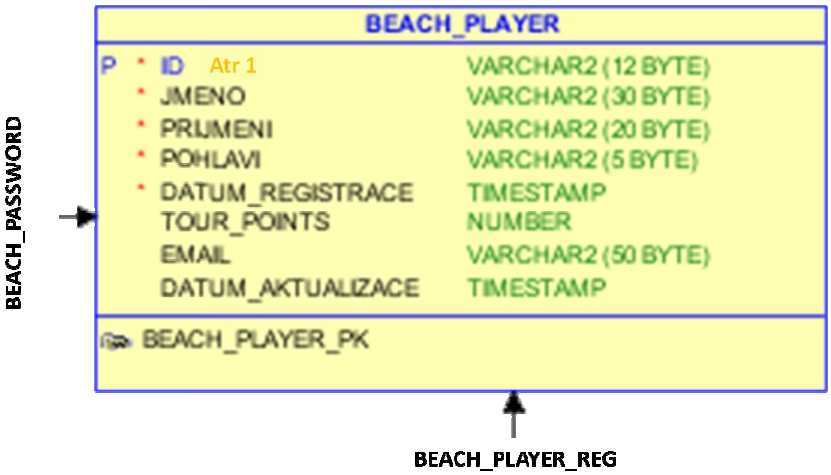
Tímto postupem nemůže dojít k přetečení paměti vlivem uchovávání primárních klíčů, pokud uživatel na vlastní riziko nenastaví v možnostech aplikace vyšší počet záznamů než je maximální podporovaný počet (1 500 000 záznamů).
Například *UsedKeys* testované tabulky BEACH_PLAYER s 1 000 000 záznamů zabírá pouhých 25,7 MB – počet tabulek ani složitost primárního klíče tuhle hodnotu neovlivní!
V případě, že tabulka obsahuje složitější klíč, tak jsou některá data pro tento klíč generována náhodně (viz Obr. 16)

Obr. 16 Uchovávání primárních klíčů



3 atributy PFK patří dvěma constraintům ⇒ 2 paměťová místa v poli o velikosti 3 budou indexy referovaných záznamů. ⇒ Zbylo 1 místo pro hodnotu intervalu PK pro poslední atribut PK. ⇒ Všechny 4 atributy primárního klíče mohou být v *UniqueMembers*.
Element ze seznamu *UniqueMembers* := pole: {**index1**, **index2**, **rand (interval)**}.

Obr. 17 Uchovávání primárních klíčů



Tento atribut tvoří sám primární klíč, to znamená, že musí mít přiřazenou unikátní doménu.
Do seznamu *UsedKeys* se vkládají jen hodnoty sekvence, definované doménou.
Element ze seznamu *UniqueMembers* := pole: {**sekvence**, **-1,-1**}.

6.2. Maximální počet záznamů dataSetu

Jak již bylo zmíněno v kapitole Generování primárních klíčů, vytvořené dataSety zabírají při generování dat velkou část paměti, proto byl omezen maximální počet všech záznamů v dataSetu na 1 500 000.

Bylo otestováno, kolik místa v paměti zabere dataSet s tabulkou `BEACH_PLAYER` (viz Obr. 17) s miliónem záznamů. DataSet s touto tabulkou uložený v XML souboru zabírá až 477 MB. DataSet s tabulkou `BEACH_TEAM` (viz Příloha 1: Testovací databáze) obsahující méně atributů může zabrat s miliónem záznamů až 347 MB.

Pro celkový počet 1 500 000 záznamů by měla vystačit kapacita operační paměti 2 GB (záleží na systému).

V možnostech aplikace (Tools/Options) je možné nastavit i vyšší počet záznamů než 1 500 000, ale hrozí zhroucení aplikace – paměť může přetéct a aplikace tak zkolabovat uprostřed generování dat.

6.3. Prováděné kontroly před zahájením generování dat

Hromadné vkládání dat je časově náročná operace, a proto je vhodné výjimky, které je možné odhalit ještě před generováním dat, odchytit. Některé výjimky jsou odchyceny až při vytváření dataSetu, který se pak mapuje na databázi.

6.3.1. Kontroly prováděné před vytvářením dataSetu

Metoda `LoadTables.KeysCheck` kontroluje pro každou tabulku, zda domény pro *UniqueMember* atributy jsou správně přiřazeny.

Metoda `LoadTables.MaxRowsCheck` kontroluje, zda je možné vygenerovat zvolený počet záznamů do tabulky (viz Příloha 3: Ostatní technické detaily). Dále kontroluje, zda zvolený počet záznamů k vygenerování není vyšší než maximální podporovaný počet záznamů.

Metoda `DataGenerator.DomainCheck` volá všechny již zmiňované metody a navíc kontroluje přiřazené textové soubory, zda obsahují textové řetězce, které jsou převoditelné na daný datový typ.

6.3.2. Výjimky vyhozené při vytváření dataSetu

Některé výjimky mohou být odchyceny až při vytváření dataSetu.

Například může být odchycena výjimka, která je vyhozena v případě, že aplikace nenašla shodu hodnot atributů klíče dvou různých tabulek, na které odkazuje jeden atribut.

Příklad výskytu atributu, který referuje dvakrát na jednu či více tabulek: `BEACH_MATCH_COUPLE` (viz Příloha 1: Testovací databáze)

Další výjimka, která může být vyhozena je systémová výjimka `OutMemoryException`. Tato výjimka je vyhozena v případě, že operační paměť nebude schopna pojmout dataSet s vysokým počtem záznamů.

Další skupina výjimek může nastat při mapování, při komunikaci s databází apod.

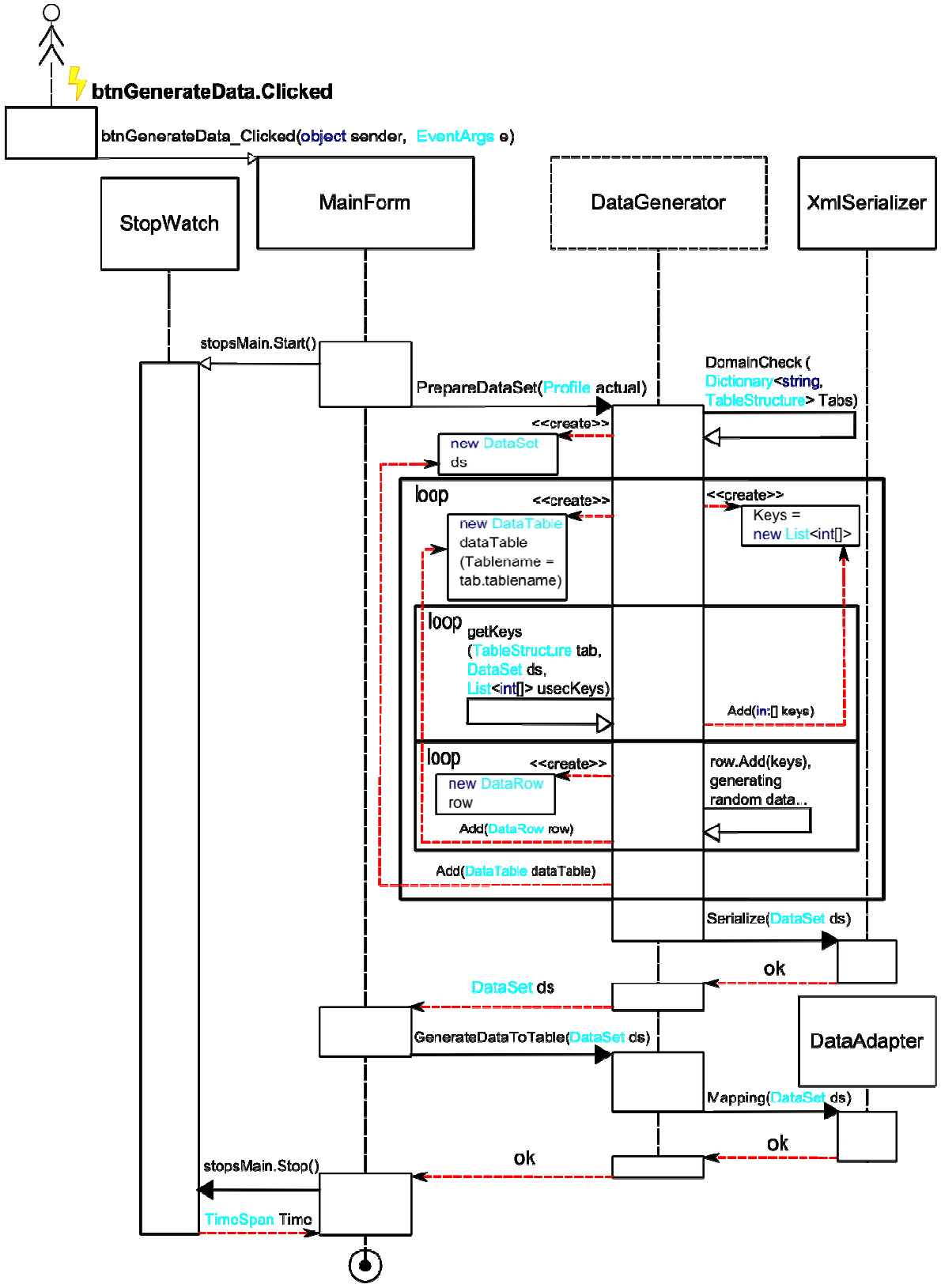
6.4. Statická třída DataGenerator

Statická třída `DataGenerator` obstarává metody pro vytváření dataSetu, mapování dataSetu na databázi, generování regulárních výrazů atp.

6.4.1. Sekvenční diagram třídy DataGenerator

Tento sekvenční diagram zachycuje ideu technického řešení generování a následného hromadného vkládání dat do databáze.

Obr. 18 Sekvenční diagram generování a hromadného vkládání dat do databáze



- **Kontrola přiřazených domén**

Jedná se o první krok po spuštění generování dat. Vše, co v tomto kroku DataGenerator kontroluje, je popsáno v kapitole 6.3.1. Kontroly prováděné před vytvářením dataSetu.

- **Příprava dataSetu**

V této fázi se pro každou z načtených tabulek (LoadedTables) vytváří objekt DataTable, který se na konci každé iterace vloží do dataSetu.
V paměti se uchovávají jen indexy právě vytvářené tabulky.
Příprava dataSetu při testování se jevila jako časově náročná operace (záleží na struktuře tabulek v dataSetu).

Postup: Vytváření dataSetu sleduje následující kroky:

- Vytvoření nového objektu typu `DataSet` reprezentující tabulkové schéma.
- Cyklus naplňování dataSetu.
  - Určení *UniqueMembers* tabulky.
  - Generování primárních klíčů (viz 6.1. Generování primárních klíčů).
  - Vytvoření nové instance typu `DataTable` [17] reprezentující načtenou tabulku ze seznamu tabulek (typu TableStructure – viz 3.3.1.).
  - Cyklus vkládání záznamů do dataTable.
    - Vytvoření nového objektu `DataRow` [18] reprezentující záznam tabulky.
    - Zápis primárních klíčů do dataRow. Podle pořadí iterace je vybrán element z *Uniquemembers*, podle kterého jsou vytvořena data pro atributy primárního klíče.
    - Zápis cizích klíčů, které nejsou zároveň primárními, do dataRow.
    - Zápis náhodných dat
    - Vložení dataRow do dataTable.
  - Zápis dataTable do dataSetu (konkrétní instance).
- Navrácení hotového dataSetu.

- **Mapování dataSetu na uživatelské tabulky na databázi**

Mapování je prováděno pomocí objektu `DbDataAdapter`.

6.5. Generování dat pro testování škálovatelnosti propustnosti při zvyšujícím se počtu záznamů

Úkolem testování škálovatelnosti propustnosti při zvyšujícím se počtu záznamů je obdržet výsledky testů, které přímo odpovídají tomu, jak se databáze chová nejdříve s nízkým počtem záznamů a pak s vysokým počtem záznamů.
Nejdříve je třeba vygenerovat nízký počet dat (například 1000), poté provést testy TPM, smazat data z databáze, vygenerovat několika násobně vyšší počet dat (například 100 000) a znovu provést testy.

6.5.1. Způsoby generování dat

První způsob: Využít předpřipravené dataSety vygenerované při předchozích generováních dat.
Druhý způsob: Vygenerovat nízký počet dat, provést testy, změnit předpis domén a znovu provést testy.

7. Generování testovacího vytížení

Po vygenerování a následném hromadném vložení dat do databáze následuje fáze generování testovacího vytížení. V této fázi se testuje propustnost – testování TPM a popř. škálovatelnost propustnosti TPM nebo TPM v čase (t) se zvyšujícím se počtem připojení nebo se zvyšujícím se počtem záznamů (viz 7.5. Generování dat pro testování škálovatelnosti TPM z hlediska zvyšujícího se počtu záznamů).

7.1. Technické řešení poměru vytížení transakcí

Transaction Load Ratio (poměr vytížení transakce), jak již bylo řečeno, určuje poměrově, jak často bude transakce vykonávána v průběhu testu vůči ostatním aktivním transakcím.
Procentuálně lze poměr vyjádřit vzorcem:

$Percentage\ Load\ Ratio = \frac{loadRatio \cdot 100}{n \sum_{i=0}^{n} loadRatio_{\ i}}(\%)$; kde i je index seznamu aktivních transakcí.

To znamená, že pokud máme 3 transakce s poměry vytížení 20,70 a 10, k nim je přidána další transakce s poměrem vytížení například 40, tak procentuální poměr vytížení například první transakce bude následující: 20*100/(20+70+10+40) = 2000/140 = 14,3 (%). Stejná metoda nastavení vytížení transakcí se využívá i v aplikaci Swingbench (viz 2.2.).

Dokumentace Swingbench bez překladu

What does the transaction load ratio mean?
The load ratio is the ratio in comparison to other transactions. ie.
T1 load ratio 10 = typically executes 16% of the time
T2 load ratio 20 = typically executes 33% of the time
T3 load ratio 30 = typically executes 50% of the time
Load ratios allow more precise control of the transactions. You change the ratios by modify the values within the config file or by changing them with the swingbench UI as shown below.
[4]

7.1.1. Technické řešení

Je třeba si uvědomit, že aplikace by se neměla při testování zatěžovat složitými výpočty, kterou transakci si vybrat, proto je vhodné navrhnout jak zkonstruovat vybírání transakcí za předpokladu dodržení poměru vytížení.

a) Řešení č. 1: Vytvořit slovník, kde klíč by byl poměrový nebo procentuální rozsah jednotlivých aktuálních transakcí a hodnota by byla indexem transakce ze seznamu transakcí. Tuhle možnost je možné předem vyloučit kvůli příliš složitého vyhledávání klíče podle náhodně zvoleného čísla (muselo by se porovnávat pro všechny klíče, zda číslo je z daného časového rozpětí, dokud by se nenašel ten správný klíč, což by mohlo být časově náročné).
b) Řešení č. 2: Před testováním převést všechny poměry vytížení aktuálních transakcí na procenta a vytvořit obyčejný *seznam indexů transakcí*, do kterého by se index transakce ze seznamu transakcí vkládal v cyklu s počtem iterací, který by byl roven procentuální hodnotě vytížení transakce.

c) Zvolené řešení: Bylo zvoleno podobné řešení jako v případě *b)*. Zvolené řešení však zohledňuje transakce se zanedbatelným poměrem vytížení (load ratio) tím, že index transakce ze seznamu transakcí je vložen do *seznamu indexů transakcí* desetinásobně vícekrát než v případě *b)*.

Pseudokód 1: Vytvoření a naplnění seznamu indexů

```
List<NUMBER> Indexes (new);
foreach (transaction in Transactions)
{
    NUMBER iterations = transaction.loadRatio*1000/SUM(loadRatio of all enabled transactions);
    for(i ; i < iterations; i++)
    {
        Indexes.Add(Transactions.IndexOf(transaction));
    }
}
```

Pseudokód 2: Vyběr transakce

```
Transactions[random.Next(0, Indexes.Count)];  //rychlý přístup k transakci podle indexu v seznamu
```

7.2. Definování vytížení

Před generováním testovacího vytížení je nutné nastavit „parametry“ benchmarku:

a) **Počet virtuálních uživatelů:** Udává počet spouštěných vláken, která budou provádět transakce.
b) **Editace transakcí:**
   i) Nastavení názvu
   ii) Zdrojový kód transakce může být reprezentován pomocí anonymní procedury nebo pomocí SQL dotazu nejlépe s využitím vázaných proměnných (pokud jsou zapotřebí). Vázaným proměnným (parametrům) lze přiřadit domény, které slouží jako předpis pro generování náhodných dat do testovacího dotazu.
   iii) Nastavení úrovně izolace
   iv) Poměr vytížení transakce
   v) Prodleva po vykonání transakce
   vi) Aktivita (aktivita určuje, zda se transakce zúčastní testovacího vytížení v závislosti na své hodnotě poměru vytížení).
c) **Škálovatelnost podle počtu připojení:** Zvolí-li uživatel volbu Test škálovatelnosti DB podle počtu připojení, generování testovacího vytížení bude generovat vytížení dvakrát: Jednou se zvoleným počtem virtuálních uživatelů a pak při dvojnásobném počtu virtuálních uživatelů.

7.3. Průběh generování testovacího vytížení

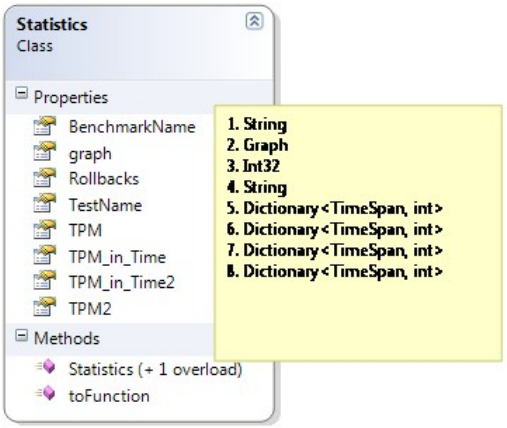
Jak již bylo zmíněno v návrhu, generování testovacího vytížení (již připraveného benchmarku) postupuje v těchto krocích:

a) **Spuštění:** Vyvoláním události spuštění (uživatelským vstupem), začne celý proces.
b) **Vytvoření a spuštění vláken:** Nejdříve je třeba v cyklu vytvořit a spustit tolik vláken, kolik je virtuálních uživatelů (každé jedno vlákno představuje jednoho virtuálního uživatele).
c) **Operace vlákna:** Každé z těchto vytvořených vláken provádí stejné operace definované stejnou metodou.

i) V první fázi si vlákno vytvoří lokální proměnné – novou kolekci TPM_in_Time, nového klienta pro komunikaci s databází (přihlašovací údaje jsou získány z aktuálního profilu).
ii) Každé vlákno si vybere transakci (viz Pseudokód 2: Výběr transakce).
iii) Provede transakci.
iv) V poslední fázi se zapíší do lokální kolekce TPM_in_Time data v závislosti na čase, který se měří ve speciálním vlákně pomocí stopek (System.Dianostics.Stopwatch) Pokud je při transakci vyhozena výjimka, vše je zapsáno do příslušné tabulky Rollbacks. Před Odhlašováním se zapíší všechny hodnoty lokálních statistik do globálních s využitím uzamykání objektů ve společném paměťovém prostoru vláken (viz 7.3.2.).

d) **Zastavení času:** Vlákno pro měření času se zastaví až po odpojení posledního klienta.
e) **Vykreslení grafů:** Na základě sloučených kolekcí TPM, TPM_inTime a popř. i TPM2 a TPM_inTime2 jsou vykresleny grafy do bitmapového obrázku.

Obr. 19: Třída Statistics



- Kolekce statistik jsou:
  - TPM (transakce za minutu): Jedná se o slovník, kde klíčem je čas T a hodnotou je celkový počet provedených transakcí od začátku testu v čase t.
  - TPM_inTime (transakce za minutu v čase): Jedná se o slovník, kde klíčem je čas T a hodnotou je celkový počet provedených transakcí od času T do času T + 1 min.
  - Další dvě kolekce TPM2 a TPM_inTime2 jsou využívany pro testování škálovatelnosti propustnosti při zvyšujícím se počtu uživatelů.
  - Poslední kolekce (Rollbacks) je obyčejný seznam, který zaznamenává počet operací Rollback.

7.3.1. Naplňování kolekcí statistik

Kolekce TPM, TPM2, TPM_inTime a TPM_inTime2 jsou slovníky. Klíčem těchto slovníků je čas a přidělená hodnota počet transakcí. Do slovníků jsou již po vytvoření vygenerovány dvojice klíč – hodnota (čas – počet transakcí), kde sousední časy (klíče) se liší o jednu minutu a počet transakcí (referované hodnoty) na které klíče odkazují, jsou rovny nule.
Po ukončení každé transakce se provádí aktualizace seznamu TPM_in_Time – podle aktuálního času se vybere hodnota ze slovníku, vybraná hodnota je inkrementována o jedna.
Pokud bylo zvoleno Testování škálovatelnosti propustnosti při zvyšujícím se počtu záznamů, tak jsou provedeny dvě fáze testu. První fáze testu je test se zvoleným počtem uživatelů a druhá fáze je test

s dvojnásobně vyšším počtem uživatelů. Ve druhé fázi je využívána lokalní kolekce vlákna TPM_in_Time2.
V konečné fázi po ukončení vytížení se přepíší slovníky TPM_in_Time (popř. TPM_in_Time2) do statistik slovníků v objektu Statistics. počtu připojení a první testování je u konce, pak se spustí druhé testování, ve kterém po ukončení transakce se obdobně naplňují kolekce TPM2 a TPM_inTime2.

7.3.2. Synchronizace vláken

Synchronizace v této aplikaci využívá zamykání proměnných ve společném paměťovém prostoru vláken (což jsou globální objekty třídy).
Zamykání objektů ve společném paměťovém prostoru znamená, že k proměnné může přistupovat současně jen jedno vlákno. Vlákna ovšem pracují především s lokálními objekty (deklarovanými v metodě) z toho důvodu, aby se nemusely vzájemně blokovat. V průběhu testu jsou globální objekty třídy většinou pouze čteny (s výjimkou celkového aktuálního počtu provedených transakcí).
Zápis lokálních statistik získaných vláknem se do globálních statistik uskutečňuje až po skončení testování. Při tomto zápisu je využit zámek na objekt představující právě zapisovanou statistiku.
Synchronizace vláken: [2].

8. Testování aplikace

Na závěr je třeba ještě otestovat hotovou aplikaci na propustnost a výkonnost.

8.1. Testování propustnosti

Generování testovacího vytížení bylo otestováno na propustnost. Tyto testy pak byly porovnány s testy komerčních nástrojů.

Testování propustnosti bez potvrzování transakcí

TPS v rozsahu: **95 – 99 (transakcí za sekundu)**, což je více, než bylo naměřeno pomocí aplikací Benchmark Factory (popř. Swingbench).
Při zapnutí Autocommit na databázi (automatického potvrzování transakcí) byly naměřeny o něco nižší hodnoty: **90 – 95 (transakcí za sekundu)**.
Tyto testy však prováděny s již připojenými virtuálními uživateli (popř. s již vytvořeným prostředím pro komunikaci s databází). Testy, které se hodnotou TPS blížili nejvíce testům již zmiňovaných aplikací, spočívaly v tom, že v okamžiku spuštění časomíry se uživatelé teprve začínali připojovat a potvrzování transakcí bylo aktivní pouze na databázi (Autocommit). Při těchto testech byly naměřeny tyto hodnoty TPS: **80 – 87 (transakcí za sekundu)**.

Testování souběhu transakcí s potvrzováním transakcí

V první fázi testování souběhu transakcí s potvrzováním transakcí v rámci aplikace je nutné vypnout automatické potvrzování transakcí na databázi (Autocommit), aby se transakce zbytečně nepotvrzovaly dvakrát.
Naměřené hodnoty při testování souběhu transakcí s potvrzováním transakcí se zvolenou izolační úrovní READCOMMITED: **49 – 52 (transakcí za sekundu)**.
Naměřené hodnoty při testování souběhu transakcí s potvrzováním transakcí se zvolenou izolační úrovní SERIALIZABLE: **31 – 35 (transakcí za sekundu)**.

Testování souběhu transakcí v interní školní síti

Testování souběhu transakcí ve školní síti na školní výukové databázi je naprosto nesrovnatelné s testováními provedenými stejné databázi připojením přes vpn klienta. Testy v interní školní síti dosahovaly podstatně vyšší propustnosti. TPS bez potvrzování transakcí: až 639 (transakcí za sekundu). TPS s potvrzováním transakcí: až 357 (transakcí za sekundu). Testy v interní školní síti byly prováděny pouze s vlastní aplikací, z důvodu vypršení zkušební doby zkušební verze aplikace Benchmark Factory.

Shrnutí testování souběhu transakcí

Za předpokladu, že vypustit potvrzování transakcí v reálném provozu nepřichází v úvahu, je z vyplynulých skutečností zřejmé, že nejlepším nastavením generování testovacího vytížení je vypnuté potvrzování transakcí v rámci aplikace a zapnuté automatické potvrzování transakcí na DB (Autocommit). Automatické potvrzování transakcí přímo na databázi nesnižuje tolik propustnost, jak potvrzování transakcí v rámci aplikace.

8.2. Testování výkonu aplikace

Testování výkonu aplikace odhaluje slabá místa programu s vysokou časovou složitostí prováděných operací apod.

8.2.1. Optimalizace vytváření dataSetu

Při testování bylo analyzováno několik slabých míst při vytváření dataSetu.

Optimalizace:

- Minimalizace operací v cyklu
- Generátor regulárních výrazů
- Generování primárních klíčů

Testovací tabulka

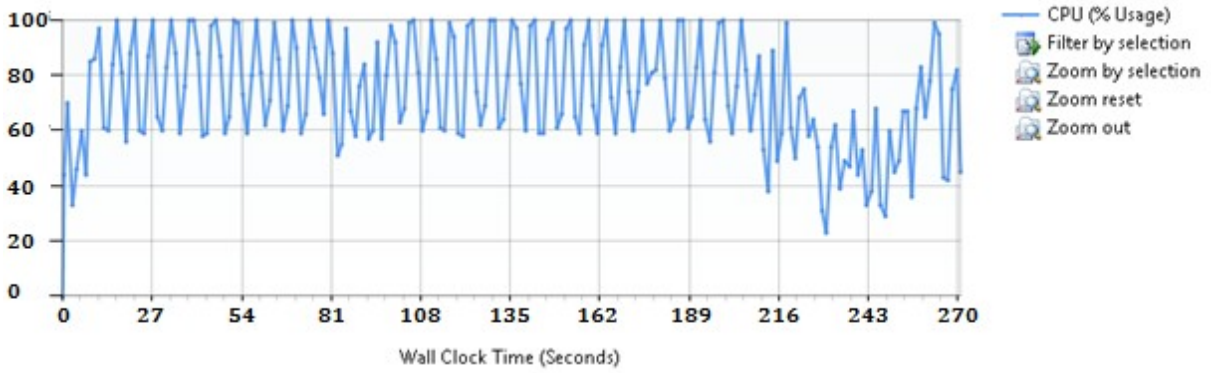
Úprava kódu	Tabulky testovaného dataSetu	Počet záznamů	Čas generování	
			Před (s)	Po (s)
Minimalizace operací v cyklu	BEACH_PLAYER	1 000 000	845	764
Generátor řetězců podle Regex*	BEACH_TEAM	1 000 000	**> 3 600**	**457**
Generátor složených PK	BEACH_GROUP, BEACH_TOURNAMENT	1 400 000	**> 18 000**	**767**

*Regulární výraz

8.2.2. Analýza výkonu nástroje

Testování výkonu bylo provedeno pro základní funkce: Generování dat do dataSetu a generování testovacího vytížení.

Obr. 20 Generování dat do dataSetu s tabulkami BEACH_GROUP a BEACH_TOURNAMENT



Průměrné využití CPU: 82,48%
Využitá kapacita RAM: 754,87 MB

9. Závěr

Nejprve byly analyzovány možnosti vybraných komerčních nástrojů pro generování testovacího vytížení databáze.
Z analýzy vyplynulo, že většina těchto nástrojů se zaměřuje především na testy nad předem definovanými schématy tabulek. Na těch tabulkách jsou pak prováděny testy: škálovatelnost DB, škálovatelnost HW serveru, propustnost serveru.
Komerční nástroje řeší vlastní schéma tabulek jen okrajově.
Tento a další poznatky z analýzy pomohly při návrhu a implementace vlastní aplikace.

Aplikace je schopná do vlastních uživatelských tabulek generovat náhodná data základních datových typů (řetězec, číslo, datum). Při generování těchto dat řeší problematiku primárních a cizích klíčů. Pro uživatele je k dispozici několik předdefinovaných typů domén. Aplikace také umožňuje uživateli vytvářet domény vlastní. Zdrojem možných hodnot každé domény může být seznam hodnot v textovém souboru nebo regulární výraz. Aplikace proto obsahuje vlastní generátor řetězců, které odpovídají zadanému regulárnímu výrazu.

Při generování testovacího vytížení lze definovat jednotlivé benchmarky. V rámci benchmarku může uživatel zadat libovolné SQL dotazy, uložené procedury nebo funkce, k těmto příkazům definovat pravděpodobnost jejich spuštění. U jednotlivých příkazů je opět možné použít domény pro vygenerování náhodných dat. Dále je možné zvolit počet virtuálních uživatelů, kteří budou paralelně do databáze přistupovat a simulovat tak reálné zatížení.
Po spuštění testovacího vytížení aplikace zobrazí graf se získanými statistikami, který poskytne uživateli představu o výkonu databáze.

Testování ukázalo, že aplikace je funkční a splňuje zadání bakalářské práce. Aplikace se zaměřuje na funkcionalitu, která je u analyzovaných komerčních nástrojů spíše opomíjena, ale funkcionalitu zejména standardizovaných testů TPC neřeší. Přímé srovnání s těmito aplikacemi tedy nedává úplně smysl.
V budoucnu by bylo možné aplikaci rozšířit například o podporu dalších datových zdrojů (MS SQL Server, MySQL, Aplikace umožňuje testovat propustnost databáze. Generuje data pro základní datové typy (řetězec, číslo, datum) a má k dispozici několik různých domén. Umožňuje také definovat vlastní doménu a jednotlivé domény je možné přiřadit testovacím dotazům.
Je možné nastavit frekvenci spouštění dotazů a spustit uložené procedury a funkce.
Návrhem pro další vývoj by mohlo být testování škálovatelnosti.

Literatura

Tradiční tištěné dokumenty

[1] Thomas Kyte: Oracle Database Architecture
[2] Simon Robin: C# Programujeme profesionálně
[3] Benchmark Factory 6.6.1 – nápověda. USA: Quest Software
[4] Dokumentace k aplikaci Swingbench. USA: Dominic Giles

Elektronické zdroje

[5] Benchmark – definice:
URL: <http://en.wikipedia.org/wiki/Benchmark_%28computing%29> [cit. 2012-02-05]

[6] TPC – Transaction Processing Council
URL: < http://www.tpc.org/> [cit. 2011-12-10]

[7] TPC-B
URL: <http://www.tpc.org/tpcb/default.asp> [cit. 2011-12-10]

[8] TPC-C
URL: <http://www.tpc.org/tpcc/default.asp> [cit. 2011-12-10]

[9] TPC-D
URL: <http://www.tpc.org/tpcd/default.asp> [cit. 2011-12-10]

[10] TPC-E
URL: <http://www.tpc.org/tpce/default.asp> [cit. 2011-12-10]

[11] TPC-H
URL: <http://www.tpc.org/tpch/default.asp> [cit. 2011-12-10]

[12] ANSI – American North Standards Institute
URL: <http://www.ansi.org/default.aspx> [cit. 2011-12-10]

[13] Dominic Giles - Swingbench
URL: <http://www.dominicgiles.com/swingbench.html> [cit. 2011-12-10]

[14] Swingbench – Videa
URL: <http://www.dominicgiles.com/screencasts.html> [cit. 2012-01-05]

[15] VŠB Katedra informatiky – Polymorfismus
URL: <http://www.cs.vsb.cz/benes/vyuka/upr/texty/objekty/ch01s03s03.html> [cit. 2012-01-15]

[16] msdn – DataSet Class

URL: <http://msdn.microsoft.com/en-us/library/system.data.dataset.aspx> [cit. 2012-04-29]

[17] msdn – DataTable Class
URL: <http://msdn.microsoft.com/en-us/library/system.data.datatable.aspx> [cit. 2012-04-29]

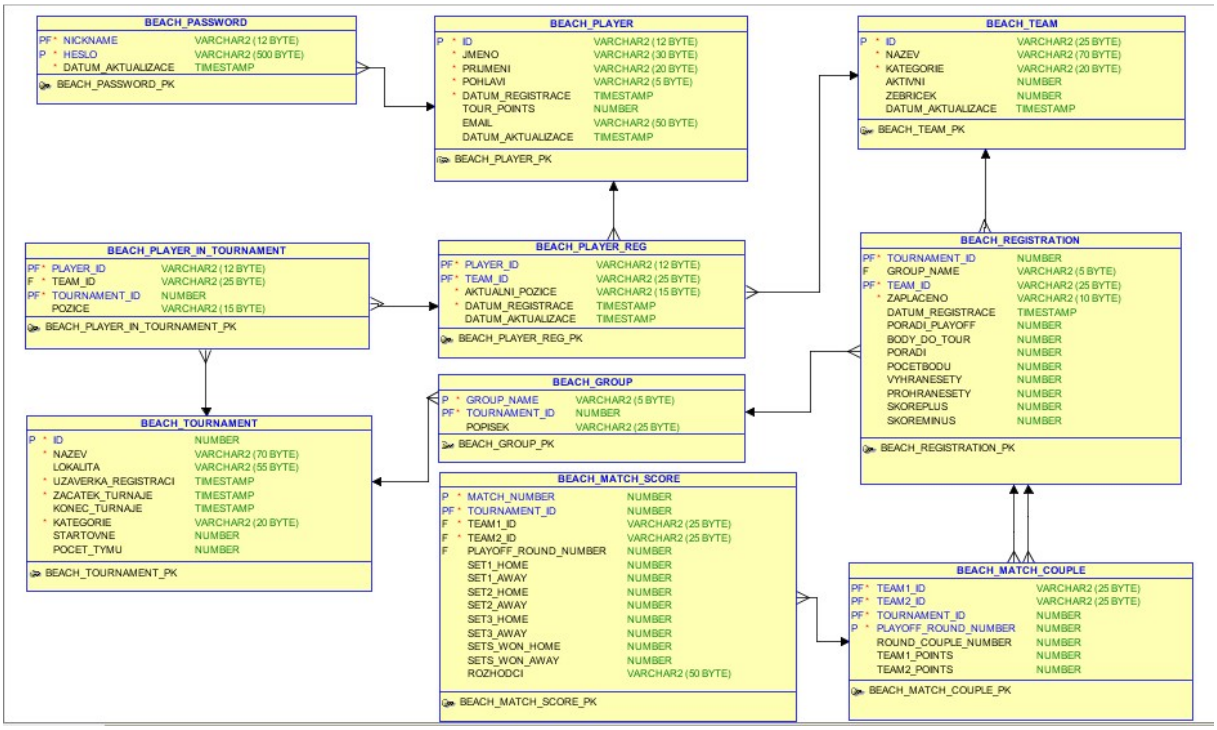
[18] msdn – DataRow Class
URL: <http://msdn.microsoft.com/en-us/library/system.data.datarow.aspx> [cit. 2012-04-29]

[19] Oracle Dokumentace – Constraint Clause
URL: <http://docs.oracle.com/javadb/10.6.1.0/ref/rrefsqlj13590.html> [cit. 2012-04-29]

[20] msdn – jazyk pro regulární výrazy
URL: <http://msdn.microsoft.com/en-us/library/az24scfc.aspx> [cit. 2012-04-30]

Příloha 1: Testovací databáze

E-R diagram:



Příloha 2: Technická dokumentace ke generování PK

Nastavení předpisu domén u primárních klíčů

Aplikace definuje 3 typy domén určených pro primární klíče, každá z těchto domén má odlišný předpis. Typy domén: domény pro generování unikátních dat, domény pro generování dat podílejících se na unikátnosti klíče a domény pro atributy cizího klíče, které se podílejí na unikátnosti klíče.

Důvody změn předpisů domén pro primární klíče

Měnit předpis domén u primárních klíčů nebo přiřazovat pro primární klíče domény nové, které nezasahují do intervalu minimální a maximální hodnoty sekvence domény staré, je nutné provádět tehdy, pokud je zapotřebí k již vygenerovaným datům na databázi podle starých domén vygenerovat data nová, **aby nedošlo ke vkládání duplicitního klíče**. Lepší způsob však je v xml editoru upravit vytvořený dataSet (jak již bylo zmíněno v kapitole 2.2 Editace PK v dataSetu pro znásobení počtu dat) a pro generování dat použít nově vzniklý dataSet. **Změnit předpis domény nebo přiřadit novou doménu s jiným předpisem je nutné před zahájením testování pro testovací dotazy typu „insert“** (popř. „update“, pokud se aktualizace týká primárního klíče), protože při testování jsou hodnoty generovány bez ohledu na to, jaké hodnoty byly generovány při hromadném vkládání.

Domény pro generování unikátních dat

Tyto domény mají označení „unique“ a používají se zpravidla pro atributy, které samy tvoří primární klíč. Atributům, které nejsou součástí primárního klíče, není povoleno tuto doménu přiřadit. Tyto domény využívají **jen omezení minimální hodnoty sekvence**, každá další hodnota se vypočítá tak, že se k aktuální hodnotě přičte krok. Na začátku se přičte krok k minimální hodnotě.

Pokud je zapotřebí znovu generovat unikátní data pro atribut, pro který již byla generována data stejné unikátní domény, tak je zapotřebí změnit minimální hodnotu sekvence, anebo změnit krok – **nikoliv však obojí!** V případě změny minimální hodnoty sekvence, kterou by bylo možné použít, se dá použít následující rovnice:

$$\mathbf{min_{new}} = \boldsymbol{min_{old}} + (\boldsymbol{n_{old}} \cdot \boldsymbol{step_{old}}) + \mathbf{1}\,;\, \boldsymbol{kde\ n_{old}\ je\ počet\ vygenerovaných\ dat\ dle\ staré\ domény}$$

Tento vzorec pro výpočet minimální hodnoty sekvence je možné ignorovat a obejít ho následujícím způsobem: Stačí jen nastavit krok na jinou hodnotu než 1, vygenerovat data a pak stačí změnit jen krok nikoliv však minimální hodnotu sekvence. **V případě, že uživatel nezmění minimální hodnotu sekvence, tak krok nové domény nesmí být násobkem kroku staré domény, (popř. nesmí být dělitelný krokem staré domény).**

Domény pro generování dat podílejících se na unikátnosti klíče

Tyto domény se používají pro atributy, které netvoří samy primární klíč, ale podílejí se na jeho unikátnosti, v této práci se takový atribut označuje jako *UniqueMember*.

Ne všechny atributy primárního klíče musí být v *UniqueMembers*, pokud jeden atribut má doménu určenou pro generování unikátních dat, tak dokonce všem ostatním atributům primárního klíče stejné tabulky musí být přiřazena doména pro generování náhodných dat. Pokud mají alespoň dva atributu doménu pro generování dat podílejících se na unikátnosti klíče, pak kombinace těchto dvou dat se nebude opakovat u vygenerovaných záznamů a to stačí k tomu, aby byl celý klíč obsahující libovolný počet atributů unikátní.

Atributům, které nejsou součástí primárního klíče, není povoleno tuto doménu přiřadit. Tyto domény používají minimum i maximum pro omezení intervalu, ze kterého se náhodně vybere hodnota pro klíč (třída DataGenerator zajistí, že bude kombinace takovýchto hodnot unikátní). Minimální hodnota intervalu, kterou lze použít pro další generování se řídí následujícím vzorcem:
Vzorec: $\min_{new} = max_{old} + 1$;
Maximální hodnotu je nutné volit tak, aby součin rozdílu maxima a minima všech *UniqueMembers* dané tabulky byl minimálně roven počtu záznamů, které se mají vygenerovat pro tu tabulku. Tyto domény nemohou být přiřazeny atributu primárního klíče, který je zároveň součást cizího klíče.

Domény pro atributy cizího klíče, které se podílejí na unikátnosti klíče

Tyto domény se používají výhradně pro atributy, které jsou součástí jak primárního tak cizího klíče. Základní rozdíl oproti předchozím dvěma typům domén je v tom, že tato doména nepoužívá ani sekvenci pro generování dat ani interval hodnot. Data generuje tak, že vybere náhodný index z již existujících záznamů referované tabulky a podle toho, kterému atributu doména patří, vybere pro tento atribut příslušná data z atributu, na který daný ten daný atribut referuje. Pokud doména přiřazená atributu, je určená pro atribut podílející se na unikátnosti klíče (*UniqueMember*), pak kombinace vybraného indexu a dalších hodnot pole musí být v rámci seznamu *UsedKeys* unikátní. Pokud je navíc tato doména unikátní, pak samotné vybrané indexy se nesmí opakovat (respektive kombinace tří hodnot, kdy 2 z těchto hodnot jsou vždy -1 a ta třetí je ten daný index).

Editace PK v dataSetu pro znásobení počtu dat

Jak již bylo zmíněno dataSety jsou ukládány do xml editoru pro snadnou editaci. Někdy může být zapotřebí vygenerovat k již vygenerovaným datům další data (například v případě, že operační paměť není dostačující pro generování vysokého počtu záznamů najednou).
Zjednoho dataSetu lze snadno udělat dva vytvořením kopie souboru a pak v xml editoru změnou hodnoty u aspoň jednoho z atributů primárního klíče typu řetězec (viz příklad). To samé je třeba udělat u všech tabulek pro hodnoty atributu cizího klíče, který odkazuje na ten daný atribut. **Je nutné mít pro tuto operaci v každé tabulce aspoň jeden atribut PK formátu řetězec a navíc regulární výrazy nastavit tak, aby u všech takových atributů PK kterékoliv tabulky bylo zřejmé, které tabulce patří**, protože pro ostatní formáty (Int32,Int64, DateTime) to nelze tak jednoduše udělat. Obyčejný xml editor však může mít problém s načítáním rozsáhlého dataSetu, proto je vhodné uvážit, kolik záznamů generovat najednou. Pokud uživatel generuje nízký počet záznamů najednou, je zbytečné přiřazovat velké rozsahy minima a maxima domén pro PK. S připravenými dataSety lze pořád dokola vkládat záznamy (samozřejmě za předpokladu smazání starých dat) a provádět testy. Testovacím dotazům pro vázané proměnné spojené s primárními klíči je nutné opět přiřadit jiné domény než ty, které byly použity při generování.

Příklad:

Regulární výraz pro atribut ID tabulky BEACH_PLAYER byl nastaven na P_ID… u cizího klíče například tabulky BEACH_PASSWORD by byl pro atribut PLAYER_ID například výraz: PSPID.
dataSet.xml:

```
…
<BEACH_PLAYER diffgr:id="BEACH_PLAYER1" msdata:rowOrder="0" …>
    <ID>P_ID0</ID>
…
```

Následuje příkaz pro xml editor: najít všechna P_ID a nahradit P_IDa … Element „<ID>P_ID0</ID>“ se změní na „<ID> P_IDa0</ID>“. Poté je třeba zjistit, které tabulky odkazují na BEACH_PLAYER (BEACH_PLAYER_REG a BEACH_PASSWORD) a změnit obdobně hodnoty atributu cizího klíče, který odkazuje na atribut ID tabulky BEACH_PLAYER. **Není vhodné příliš prodlužovat délku generovaného řetězce při nízké hodnotě omezení počtu znaků Varchar2 na databázi**
Pro **ID číselného formátu**, popř. pro ID ve formátu kalendářního data, něco podobného udělat nelze.

Uchovávání primárních klíčů v operační paměti

Jedním ze základních problémů hromadného generování dat do databáze, se kterým se musí počítat, je kapacita operační paměti. DataSet neobsahující žádná data o primárních ani cizích klíčích, obsahující jen jednu běžnou tabulku s devíti atributy (převážně datového typu VARCHAR2), která má tisíc záznamů, může zabírat v paměti kolem 485 kB. Stejný dataSet se stejnou tabulkou obsahující milión záznamů může zabírat v paměti až 477 MB (489 159 kB)…otestováno s tabulkou z vlastní databáze – BEACH_PLAYER. Z toho vyplývá, že počet záznamů tabulky dataSetu je přímoúměrný paměti, kterou tabulka v dataSetu zabere, a to se jednalo o dataSet, který neměl v sobě uchovanou žádnou informaci o klíčích. Efektivním řešením z hlediska šetření místa v operační paměti by bylo ukládat informace o klíčích přímo do dataSetu. Mělo by to však jednu nevýhodu: V cyklu by se muselo zkoušet vkládat nepoužité klíče, což by mohlo nějakou chvíli trvat.

Podporované datové typy pro primární klíče

Aplikace podporuje pro primární klíče víceméně všechny datové typy převeditelné do systémových datových typů: System.String, System.Int32, System.Int64 a System.DateTime (s omezením viz DateTime). Tabulka může mít jako součást primárního klíče i atributy s jinými datovými typy (např. „float“, „double“, „int16“), ale takové atributy se smí podílet na unikátnosti klíče (*UniqueMembers*) jen v případě, kdy se jedná o PFK, jehož „constraint“ obsahuje aspoň nějaké atributy s podporovanými datovými typy. Atributy nepodporovaných datových typů pro primární klíče mohou nabývat jedině náhodných hodnot podle regulárního výrazu nebo podle připojeného textového souboru.

System.String

Pro databázové datové typy převeditelné do systémového datového typu System.String aplikace generuje textové řetězce ve formátu: **domain.regularExpression + sekvence || *rand (interval)***.
Oracle: Varchar2, NVarchar2, Char, NChar
SQL Server: Varchar, NVarchar, Char, NChar

System.Int32 a System.Int64

Pro databázové datové typy převeditelné do systémového datového typu System.Int32 nebo System.Int64 aplikace generuje čísla ve formátu: **(Int32)sekvence || *rand (interval)***.
Oracle: Int32 (Number), Int64
SQL Sever: Int, BigInt

System.DateTime

Pro databázové datové typy převeditelné do systémového datového typu Sytem.DateTime aplikace generuje datum, hodiny, minuty a sekundy. Negeneruje však jen datum nebo jen čas. DateTime ukládá všechny informace o letech, měsících, dnech, hodinách, minutách, sekundách atd. do „ticks“, což je jedno číslo formátu Int64.

Aplikace si však o PK uchovává ale jen 3 čísla formátu Int32. Vše je vyřešeno následujícím způsobem: 1tick = 100 ns $\Rightarrow$ 1tick * 10^7 = 1 s.
Pro generování aktuálních kalendářních dat je nutné vynásobit nejvyšší číslo, které pojme typ Int32 (2147483647) *hodnotu* tak, aby součin odpovídal datu, které převyšuje aktuální o několik let:

a) *hodnota* = 10^7; datum = 69/1/19 3:14:07
b) *hodnota* = 10^8; datum = 681/7/6 8:21:10
c) *hodnota* = 10^9; datum = 6806/2/8 11: 31: 40

Je zřejmé, že je třeba zvolit *hodnotu* 10^9 pro generování aktuálních kalendářních dat.
Formát generovaného data: **new DateTime((Int64) sekvence*10^9 || *rand (interval)**10^9).**
Oracle: TimeStamp
SQL Server: DateTime, DateTime2 (nikoliv však Time!!)

rand(interval): Jedná se o náhodně vybrané číslo z intervalu, který specifikuje doména.

Příloha 3: Ostatní technické detaily

Kontrola maximálního počtu generovatelných dat

Tato kontrola je velmi důležitá a je úzce spjatá s generováním primárních klíčů. Maximálním možným počtem generovatelných klíčů je totiž omezen také maximální počet dat, která uživatel bude moct generovat do dané tabulky. Uživatel volí počet záznamů ručně, a tudíž je nutné tuto kontrolu provádět vždy po změně počtu záznamů v tabulce.
Nejdříve se vytvoří pole všech referovaných tabulek, to se v cyklu prochází a aktuální hodnota maximálního počtu generovatelných referencí (dále jen „maxRefs“) se násobí hodnotou maximálního počtu generovatelných dat referované tabulky (v první iteraci se tato hodnota přičítá). Pokud tabulka obsahuje atribut přiřazený ke dvěma „constraintům“, tak se ten druhý „constraint“ uloží do pole a pak pokud nějaký další atribut má na prvním místě „constraint“, který je uložen v poli „constraintů“, tak se nevloží další záznam do pole referovaných tabulek.
Vzorec pro určení maximálního počtu referencí použitý v aplikaci (v přepisu):

$$maxRefs = \prod_{j=0}^{n} numOfRowsTogenerate_j \; ; \; kde \; j \; je \; index \; pole \; ref.tabulek$$

Pole referovaných tabulek neobsahuje tabulky, na které odkazují atributy, kterým jsou přiřazeny 2 referenční constrainty.
Nelze využít vzorec pro kombinaci bez opakování, protože klasická kombinace bez opakování vybírá určitý počet prvků pouze z jedné množiny, tak aby vybraný prvek nebyl vybrán ještě jednou, ale v tomto případě vybíráme z více množin (tabulek) pokaždé jen jeden prvek (záznam).

Upozornění: V případě, že počet záznamů, které se mají generovat pro referovanou tabulku, na kterou odkazuje cizí klíč, který je v *UniqueMembers*, je roven 0, tak je vyhozena výjimka.

Vzorce pro výpočet maximálního počtu generovatelných dat do tabulky

1) V případě, že tabulka obsahuje atribut s doménou nesoucí informaci o tom, že generovaná data mají být unikátní, pak platí tento vzorec:
$\boldsymbol{maxRows = \frac{Int32.MaxValue - PK.domain.\min}{PK.domain.step+1}}$; *(PK.domain.step+1 – jednička se připočítává kvůli zaokrouhlování Integerů po dělení)*

2) V případě, že tabulka obsahuje atributy primárního klíče, jejichž domény je řadí do *UniqueMembers* a zároveň žádný z těchto atributů není cizím klíčem a nemá ani doménu, pro generování unikátních dat, pak platí tento vzorec:
$\boldsymbol{maxRows = (PK.domain.\max - PK.domain.\min)} \cdot \ldots$; *kde PK je atribut s doménou nesoucí informaci o tom, že generovaná data mají být v UniqueMembers.*

3) V případě, že tabulka obsahuje atributy primárního klíče, jejichž domény je řadí do *UniqueMembers* a zároveň některý z těchto atributů je i cizím klíčem a zároveň některý z těchto atributů není cizím klíčem, pak platí tento vzorec:

$\boldsymbol{maxRows} = \max \boldsymbol{Refs} \cdot (\boldsymbol{PK.domain}.\max - \boldsymbol{PK.domain}.\min) \cdot \ldots$; kde PK je atribut s doménou nesoucí informaci o tom, že generovaná data mají být pro *UniqueMember* atribut.

4) V případě, že primární klíč tabulky obsahuje atributy, které jsou všechny zároveň i cizími klíči, pak platí tento vzorec:
$\boldsymbol{maxRows} = \boldsymbol{maxRefs}$; $(maxRows\ je\ maximální\ počet\ generovatelných\ dat)$

5) V případě, že celkový součet všech záznamů, které se mají generovat najednou je vyšší, než maximum podporované aplikací, tak je vyhozena výjimka. Standardní maximální počet záznamů je 1 500 000. Toto maximum se však dá změnit (Tools/Options). Viz též 7.1. Kapacita operační paměti.