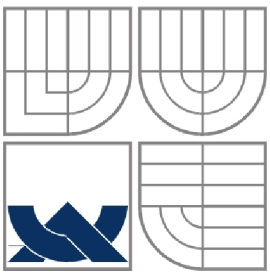
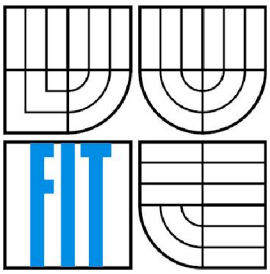


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYNCHRONIZACE DATABÁZÍ MYSQL

MYSQL DATABASE SYNCHRONIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. ONDŘEJ DLUHOŠ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. PETR JAŠA

BRNO 2010

Abstrakt

Tato diplomová práce se zabývá synchronizací databází MySQL. Cílem této práce bylo seznámení se se synchronizací databází v širším kontextu, zvolit vhodné nástroje pro nasazení v praxi, tyto nástroje implementovat, zhodnotit a analyzovat. Z použitých nástrojů byla vybrána technika replikace od MySQL, která nejlépe řeší úlohu synchronizace distribuovaného databázového systému evidence implantabilních zdravotnických prostředků. Replikace byla na tomto databázovém systému realizována a po otestování byla nasazena ve firmě Timplant s.r.o.

Abstract

This thesis deals with the MySQL database synchronization. The goal of this work was to get acquainted with database synchronization in a broader context, to choose the appropriate tools for real usage, and them to implement, evaluate and analyze these tools. From these techniques was selected MySQL replication technology that solves the synchronization task of distributed evidence database system of health implantable devices in the best way. The replication was implemented on this database system and after the testing was used in the company Timplant Ltd.

Klíčová slova

MySQL, synchronizace, platforma, SQL, databáze, server, distribuce dat, konzistence dat

Keywords

MySQL, synchronization, platform, SQL, database, server, data distribution, data consistency

Citace

Dluhoš Ondřej: Synchronizace databází MySQL, diplomová práce, Brno, FIT VUT v Brně, 2010

Synchronizace databází MySQL

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Petra Jaši. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Ondřej Dluhoš
24. 5. 2010

Poděkování

Chtěl bych na tomto místě poděkovat panu Ing. Petru Jašovi za odborné vedení a konzultace, které mi poskytl v teoretické a praktické části diplomové práce.

Obsah

1 Úvod

Počítače zpracovávají data od počátku své historie a nezbytným úkolem aplikací je tyto data ukládat, modifikovat či mazat. Tento úkol se může zdát někdy samozřejmostí, ale tyto data musí být nějakým způsobem ukládány a to většinou do nějakých databází. S databázemi se samozřejmě setkáváme denně. V nejširším slova smyslu je databáze i seznam věcí, které se mají koupit k obědu, výpis z účtu nebo třeba seznam uskutečněných telefonních hovorů. V počítačovém slova smyslu se pod výrazem "databáze" obvykle rozumí software, který spravuje nějaká data a umožňuje uživatelům tyto data rozumně měnit a spravovat.

Se zavedením databází se vytváří všemožné prostředky a nástroje, které usnadní uživateli či programátorovi s těmito daty pracovat. Mezi nemalou podporu pro snadnější práci lze zařadit i synchronizační prostředky, které dosáhly značného vývoje až v posledních letech.

Tato diplomová práce se zabývá synchronizací databází MySQL. Synchronizace databází je obecně proces, kdy se vyměňují data z jedné databáze do druhé či naopak a to tak, aby v obou databázích byla stejná data. Tato problematika nemusí být pro programátora vždy jednoduchá, a proto se tato práce snaží odpovědět na některé otázky i nejasnosti a také proč se synchronizace databází využívá.

Text této práce je rozdělen do několika navazujících kapitol. První kapitola má název Úvod. Druhá kapitola obsahuje úvod do systému řízení báze dat MySQL, čtenáře seznamuje s jejími vlastnostmi a zodpoví otázku, proč je pro synchronizaci vybrána databáze MySQL.

Třetí kapitola se v širším kontextu zabývá synchronizací. Je zde popsáno, proč se synchronizace využívá, kde je vhodné synchronizaci použít a jaké výhody může přinést jak uživateli, tak i programátorovi. Závěr kapitoly obsahuje hlavní prostředky a nástroje pro synchronizaci databází MySQL, které se nejvíce používají.

V další části jsou vybrány tři nástroje pro synchronizaci, SQLyog, Microsoft Sync Framework a MySQL replikace, které v posledních letech dosáhly značného rozvoje. Je zde rozebrána jejich funkčnost a použitelnost.

Pátá kapitola se zaměřuje na návrh synchronizačního procesu vzhledem k databázovému systému evidence implantabilních zdravotnických prostředků, který se použije pro realizaci. Jsou zde rozebrány požadavky synchronizace a potřebná konfigurační topologie.

Další kapitola obsahuje implementaci vybraných nástrojů na databázový systém. Implementace představuje vyzkoušení těchto nástrojů a v závěru kapitoly jsou tyto prostředky porovnány s ohledem na nasazení v reálném systému.

Předposlední kapitola souvisí s praktickou realizací synchronizace na databázovém systému. Pro realizaci je vybrána replikace od MySQL, která byla z použitých technik nejlepší pro nasazení v praxi. V této kapitole jsou rozebrány hlavní kroky celého synchronizačního procesu, problémy, které mohou při používání nastat, monitorování replikace a potřebné úpravy klientského programu. Zhodnocení dosažených výsledků je pak diskutováno v závěrečné kapitole.

2 Úvod do MySQL

MySQL je databázový Open Source systém, vytvořený švédskou firmou MySQL AB. Jeho hlavními autory jsou Michael „Monty“ Widenius a David Axmark. V roce 2008 společnost Sun Microsystems koupila společnost MySQL AB a tak se MySQL stává dalším Open Source softwarem od firmy Sun Microsystems. MySQL je podle [1] považován za úspěšného průkopníka dvojího licencování, tudíž je k dispozici jak pod bezplatnou licencí GPL, tak pod komerční placenou licencí.

MySQL je multiplatformní databáze. Komunikace s ní probíhá pomocí jazyka SQL. Podobně jako u ostatních SQL databází se jedná o dialekt tohoto jazyka s některými rozšířeními. Pro svou snadnou implementovatelnost (lze jej instalovat na Linux, MS Windows, ale i další operační systémy), výkon a především díky tomu, že se jedná o volně šiřitelný software, má podle [2] vysoký podíl na v současné době používaných databázích. Velmi oblíbená a často nasazovaná je kombinace MySQL, PHP a Apache jako základní software webového serveru.

MySQL bylo od počátku optimalizováno především na rychlost a to i za cenu některých zjednodušení: má jen jednoduché způsoby zálohování a až donedávna nepodporovalo pohledy, triggery, uložené procedury a synchronizaci [1]. Tyto vlastnosti jsou doplňovány teprve v posledních letech, kdy začaly nejčastějším uživatelům produktu poněkud scházet.

2.1 Proč MySQL ?

MySQL je velmi populární databáze a podle mnoha zdrojů je rovněž velmi rychlá. Nemá však tolik funkcí a možností jako některé konkurenční systémy. Vybrat si vhodnou databázi je tedy klasický kompromis mezi rychlostí softwaru a jeho schopnostmi. MySQL má samozřejmě své zastánce a odpůrce. Odpůrci například často tvrdí, že MySQL je tak rozšířená proto, že webhostingové společnosti ji často nabízejí pro hostované weby jako součást portfolia svých služeb. Kdyby se webhostingové společnosti rozhodly upřednostnit jiný software, popularita MySQL by podle jejich odpůrců podstatně klesla.

Naproti tomu webhostingové společnosti nabízejí MySQL proto, že je dobrá, a kdyby existovala lepší databáze než MySQL, byla by nabízena. Sun Microsystems tvrdí, že trh si MySQL vybral - a to podle nich jasně poukazuje na její kvality.

Asi největším soupeřem MySQL v oblasti volně šiřitelných databázových systémů je databáze PostgreSQL, která umí běžet na Windows NT (2000, XP) a na celé řadě UNIX platforem a má podstatně více možností pro uživatele než MySQL. Podle [2] je však o něco pomalejší.

V poslední době však dobývá tento segment trhu poměrně výrazným způsobem databáze Firebird. Firebird je rovněž open-source systém řízení báze dat, může běžet na systémech Windows, Linux, Mac OS X, HP-UX a Solaris a podle Petra Zajíce [2] má rovněž více možností než MySQL, ovšem je také o něco pomalejší.

2.2 Vlastnosti MySQL

Tato diplomová práce se zabývá synchronizací databází MySQL. Pro přehled nejdůležitějších vlastností a nástrojů, které mohou být jistými stavebními bloky při realizaci synchronizací databází, slouží následující seznam. Více lze nalézt v odborné literatuře, například v [3], [4], ze kterých definice některých vlastností vycházejí.

Architektura klient/server – MySQL je systém klient/server. Systém sestává z databázového serveru (MySQL) a libovolného množství klientů, kteří komunikují se serverem. Klienti se dotazují na data, ukládají změny a tak dále. Klienti mohou běžet na stejném počítači jako server, nebo mohou být na síti, třeba kdekoli na Internetu.

Kompatibilita s SQL – MySQL používá jako svůj databázový jazyk SQL (*Structured Query Language*). SQL je standardní jazyk pro dotazy a aktualizaci dat, a také se používá pro správu databáze. MySQL dodržuje současný standard SQL, avšak s některými omezeními a velkým množstvím rozšíření. Některé příkazy standardu SQL nepodporuje a definuje na místo toho své vlastní nebo je využívá jiným způsobem, jako jsou např. SUBSTRING, LIMIT, TIMESTAMP, LOCAL TIMESTAMP, atd.

Pohledy – pohled je fiktivní tabulka vytvořená dotazem SQL, umožňuje pohled na část databáze.

Uložené procedury – jsou programy v SQL uložené v databázi. Uložené procedury se všeobecně používají pro zjednodušení určitých kroků, jako například vkládání a mazání záznamů.

Triggery – jsou příkazy SQL, které server automaticky spouští při určitých operacích (např. *INSERT, UPDATE, DELETE*).

Replikace – umožňuje kopírování (replikaci) databáze na více počítačů. Toto se používá v praxi ze dvou důvodů: zvýšení odolnosti proti výpadkům serverů, a také ke zlepšení rychlosti zpracování dotazů. Podrobně bude tato problematika probrána v dalších kapitolách.

Transakce – transakce znamená v kontextu databází provedení několika databázových operací jako bloku. Databázový systém zaručuje provedení všech operací v bloku, nebo žádné z nich.

GIS funkce – MySQL podporuje od verze 4.1 ukládání dvoudimenzionálních geografických dat.

ODBC – MySQL podporuje ODBC rozhraní Connector/ODBC. Tuto vlastnost umožňuje v prostředí Microsoft Windows využití všech populárních programovacích jazyků. Rozhraní ODBC může být také implementováno do Unixu, což je ale zřídka kdy nutné.

Nezávislost na platformě – nejen aplikace klienta mohou běžet v různých operačních systémech. Server MySQL může být provozován pod mnoha operačními systémy. Nejvýznamnější jsou Apple Macintosh OS X, Linux, Microsoft Windows a mnoho dalších, jako AIX, BSDI, FreeBSD, HP-UX, OpenBSD, Net BSD, SGI Iris a Sun Solaris.

3 Synchronizace databází MySQL

Synchronizace databází je podle [4] obecně proces, kterým se stanoví soulad mezi daty ze zdroje i daty do cíle a naopak, včetně průběžné harmonizace dat v čase. V tomto procesu si obě databáze vyměňují všechny aktualizované záznamy a objekty. Jakmile jsou změny v jedné databázi použité v druhé a naopak, pak se může říci, že jsou obě databáze synchronizovány (mají stejná data). Tyto databáze musí být nějakým způsobem propojeny, nejčastěji se však jedná o spojení TCP/IP. Synchronizace databází může probíhat i v rámci jednoho serveru, což není nic neobvyklého.

Synchronizace databází se používá zejména pro kopírování dat mezi dvěma a více odloučenými pracovišti, kde všechny pracoviště potřebují pracovat nad nejnovějšími daty. Pracovištěm může například být samostatný notebook, PDA, mobil, počítač v kanceláři nebo server na centrále či pobočce. Podle [4] se také často synchronizace využívá pro vysoce výkonné aplikace, kde počet dotazů na server za jednu sekundu dosahuje v řádu stovek dotazů. Proto jsou data z databáze distribuovány na další servery a tím se rozdělí zátěž a zamezí se spadnutí či přetížení serveru [5]. Velice často se synchronizace používá také pro zálohování, ale s tím, že se většinou záloha provádí v předem určený čas, a ne při každé změně databáze. Pokud ovšem nemůže být dovoleno nějaká data ztratit, tak lze data zálohovat při každé změně v databázi, vše záleží na konkrétních požadavcích. Nezřídka se také synchronizace používá pro testovací či výukové účely.

Důvodem pro synchronizaci, jak už bylo naznačeno, může být výkon, spolehlivost, aktuálnost dat na více místech nebo snadné zálohování. Synchronizace může podle [5] poskytovat odolnost vůči selhání, v případě, že se zhroutí hlavní server (master), podřízený server (slave) lze použít jako připravenou zálohu a okamžitě ho proměnit v server hlavní (master).

3.1 Proč synchronizovat?

Existují tři dobré důvody, proč synchronizovat databázové systémy: *bezpečnost*, *rychlost* a *distribuce dat*.

3.1.1 Bezpečnost

Díky synchronizaci dat je databáze dostupná na několika počítačích. Pokud pobočný počítač přejde do stavu offline, může celý systém dál běžet, aniž by došlo k jakémukoli přerušení. Nově přidaný pobočný systém se může později synchronizovat sám.

Jestliže do stavu offline přejde hlavní počítač, jsou data uchována na pobočném počítači. V případě potřeby lze pro tyto výpadky celý systém nastavit tak, aby pobočný počítač přebral roli počítače hlavního.

Pokud by měla být synchronizace databází použita jen z bezpečnostních důvodů, tak by měla být podle Michaela Koflera [3] zvážena jiná možnost – nasazení systému RAID, který zajišťuje synchronizaci obsahu dvou (či více) pevných disků. Systém RAID však chrání jen před havárií pevných disků, nikoli před selháním operačního systému, výpadkem proudu či jinými podobnými chybami.

Synchronizaci lze použít také jako náhradu záloh. Díky tomu může být záloha vždy v aktuálním stavu, vše záleží na nastavení.

3.1.2 Rychlost

Jestliže je rychlost databázového systému limitována především mnoha dotazy, která data jen čtou (k zápisu dat dochází jen zřídka), poté synchronizované databáze mohou podle [5] ušetřit značné množství času: náročné dotazy mohou být rozděleny mezi několik pobočných systémů, zatímco hlavní systém je používán výhradně nebo hlavně pro aktualizace dat. (Určitá část z teoretického procenta zvýšení rychlosti se samozřejmě ztratí díky zvýšené míře režie, nutné pro vzájemnou komunikaci.)

Je třeba podotknout, že rychlost lze získat jen v případě, kdy je kód klienta kompatibilní s databázovým systémem. Klientské programy musí dotazy rozdělit podle pravidel rozložení zátěže (nebo prostě náhodným způsobem) mezi všechny dostupné pobočné systémy. Jsou li databázové dotazy převážně poměrně jednoduché, účinně může zátěž rozprostřít i jednoduchý rotační algoritmus na vyvažování zátěže. Pokud ovšem některé dotazy vyžadují rozsáhlé zpracování, bude se pravděpodobně muset najít důmyslnější vyvažování zátěže na úrovni aplikace.Podle [5] MySQL samo o sobě žádný takový mechanismus nenabízí.

Synchronizace může zaujmout především vyšším možným výkonem systému, ale měly by být také podle Petra Zajíce [2] zváženy i jiné prostředky pro zvýšení výkonu, především pak lepší hardware.

3.1.3 Distribuce dat

Synchronizace MySQL serveru obvykle nezatěžuje příliš šířku pásma a může být podle chuti zastavována a spuštěna. Synchronizace několika serverů je užitečná pro udržování kopie dat v zeměpisně vzdálených lokalitách, jako jsou různá datová centra. Vzdálená distribuce dat může v některých případech pracovat s připojením, které je přerušované (neúmyslně nebo jinak). Pokud ovšem bude potřeba, aby synchronizace dat měla velmi malé zpoždění, bode potřeba stabilní spoj s nízkou latencí [4].

3.2 Přehled dostupných nástrojů pro synchronizaci

Databázový Open Source systém MySQL byl vytvořen v roce 1995 jako jednoúčelová databáze pro snadné ukládání a především čtení textových dat v internetových aplikacích. Od té doby bylo vydáno přes pět nových verzí a MySQL se dočkalo nemalých změn. Jedná se zejména o nové vlastnosti a nástroje, které do jisté míry přispěly k tak velké popularitě. Pro tento databázový systém je vyvíjeno nemalé množství doplňků a podpory, což přináší správcům i klientům databází velké výhody.

Synchronizace databází avšak není tak jednoduchá, jak by se na první pohled mohlo zdát. Vše totiž záleží na počtu databází, které mají být synchronizovány, na topologii návrhu, na požadavcích a také zejména na prostředcích, které máme k dispozici. Může se stát, že byť jen z banálního návrhu topologie může vzniknout obtížně řešitelný problém.

Co se týká nástrojů pro synchronizaci databází MySQL, tak za poslední roky bylo vyvíjeno mnoho prostředků pro distribuování dat na další servery, což umožnilo správcům odpoutat se od vlastního zdlouhavého řešení a použít již hotový nástroj. Tyto prostředky jsem si dovolil rozdělit do tří kategorií a to: *Programy s GUI*, *různé platformy* nebo *řešení od MySQL*. Jsou to tři různé kategorie a nedá se říci, že jen jeden z nich se bude hodit pro všechny možné situace. Každý z těchto nástrojů má své výhody i nevýhody a jeho použití musí být prozkoumáno při konkrétní situaci. Pokud se některý z těchto nástrojů použije, je třeba také vzít v úvahu, že ve většině případů bude nutný zásah do klientských programů, které s databází komunikují.

3.2.1 Programy s GUI

Tyto nástroje jsou význačné tím, že mají grafické uživatelské prostředí a většinou nejsou zdarma, ovšem nabízejí verze, které je možno omezeným způsobem vyzkoušet. Jsou zde uvedeny jen ty programy, které se hodí pro MySQL databáze.

SQL Data Examiner

SQL Data Examiner je nástroj, který porovnává a synchronizuje obsah databáze všech verzí SQL Serveru, a také podporuje Oracle, MySQL a MS Access. Program se vyskytuje ve dvou verzích a to standard za 160,00€ nebo professional za 240, 00€, která má více funkcí. Pro základní synchronizační úlohy by měla stačit verze standard. Výhodou tohoto programu je možnost použití přes příkazový řádek a možnost uložení nastavení pro porovnání či synchronizaci. V případě potřeby provést synchronizaci z externího programu, skriptu atd. Tento program má velmi intuitivní ovládání, ale nenabízí mnoho možností pro nastavení. Více informací lze nalézt na stránkách výrobce: http://www.sqlaccessories.com/.

SQLyog

Tento program je navrhnut pouze pro databáze MySQL. SQLyog používá SQL dotazy na servery, porovnává data pomocí algoritmů a při změně provádí SQL dotazy INSERT, UPDATE a DELETE tak, aby byly databáze synchronizovány. Při použití MySQL databáze není potřeba žádného zvláštního nastavení k využívání SQLyog, pro synchronizaci stačí pouze jednoduché dotazy a k nim příslušné oprávnění. Pro využití SQLyog není zapotřebí žádného specializovaného hardware a umožňuje použít příkazový řádek. Tento program se nemusí instalovat na všechny počítače, stačí pouze jedna instalace v jedné topologii. Program lze koupit ve třech variantách a to professional za 99$, enterprise za 139$ a ultimate za 179$, to vše pro jednoho uživatele. Více informací na stránkách výrobce: http://www.webyog.com/en/.

Sync Database

Program je určen pouze pro databáze MySQL. Podporuje používání přes příkazový řádek. Může být použit pro synchronizaci sloupců, indexů, pohledů, uložených procedur a funkcí. Podporuje

synchronizaci mezi různými servery. Umožňuje ukládání a generování SQL dotazů do souborů a dokáže rozpoznat změny ve struktuře tabulky. Cena tohoto produktu je 149$ na jednu licenci. Více informací lze nalézt na stránkách výrobce: http://www.spectralcore.com/syncdatabase/.

Každý s těchto programů má své výhody a nevýhody, nedá se tedy obecně vyvodit závěry, který je lepší či horší. Vše záleží na konkrétních požadavcích pro synchronizaci. Pokud je potřeba silný nástroj, který umí hodně věcí a je za rozumnou cenu, pak je vítězem bezesporu SQLyog, který se zdá být ideálním nástrojem pro mnoho požadavků. Podrobný popis tohoto programu bude v následujících kapitolách.

3.2.2 Platformy

Hlavním zástupcem je bezesporu komplexní synchronizační platforma od společnosti Microsoft s názvem *Microsoft Sync Framework*. Tato platforma umožňuje spolupráci a offline přístup pro aplikace, služby a zařízení pro jakýkoli typ dat, ukládání dat, podporuje různé přenosové protokoly a různé topologie sítě. Zkráceně řečeno se jedná o sadu objektů pro synchronizaci různých zdrojů. Bližší popis této synchronizační platformy bude v následující kapitole.

3.2.3 Řešení od MySQL

MySQL disponuje speciálním vybavením přímo pro synchronizaci databází, které se nazývá *Replikace*. Replikace je jednou z náročnějších funkcí, jimž je databázový systém MySQL vybaven. S její pomocí lze uchovávat stejná data na více serverech. Zabudované vybavení, které má MySQL pro potřeby replikací, tvoří podle [4] základy pro budování rozsáhlých, vysoce výkonných aplikací nad MySQL. Replikace je podle Roberta Shneidera [6] k dispozici už od MySQL 3.23 a od první verze se dočkala výrazného vylepšení. Značnou výhodou replikace je přímá podpora od MySQL, tudíž je vypracována pořádná dokumentace a existuje mnoho knížek či článků pro nasazení synchronizace v praxi, což se o ostatních nástrojích říct nedá. Tento nástroj se velmi často používá, a proto bude replikace probrána v následující kapitole.

Alternativou by mohlo být použití SQL dotazů spolu se základními funkcemi MySQL ve spolupráci s nějakým aplikačním jazykem a celý synchronizační proces si naimplementovat sám. Pro inspiraci výměny dat mezi databázemi by zajisté mohl posloužit princip již zmíněného frameworku od Microsoftu, kde základní podstata synchronizace databází je v tom, že všechny změny (INSERT, UPDATE, DELETE) jsou zaznamenávány do databáze v podobě metadat. Tyto metadata jsou následně porovnávány a jsou prováděny požadované akce. Toto řešení by zajisté našlo uplatnění tam, kde nelze využít žádný zmiňovaný nástroj nebo není pro synchronizaci databází vhodný. Implementace vlastního řešení by zajisté byla velmi náročná a při složitější topologii serverů by bylo nutné použít již hotové nástroje.

4 Popis nástrojů pro synchronizaci

Z předchozí kapitoly lze vidět, že nástrojů pro synchronizaci neexistuje příliš mnoho, ale každý z nich je svým způsobem specifický a jeho konkrétní výběr závisí na daném problému. Následující kapitola se zaměří na tři různé zástupce již zmíněných synchronizačních technik z kapitoly 3.2. Všechny tyto nástroje by měly dosahovat stejného výsledku, přestože mají k synchronizaci odlišný přístup.

Jako první z vybraných nástrojů je SQLyog, což je placený program s grafickým uživatelským prostředím. Druhou možností je Microsoft Sync Framework, který je zcela odlišný od SQLyogu, ale nachází velké uplatnění v platformě .NET. Poslední popsaný nástroj je MySQL replikace, která umožňuje nastavení synchronizace do posledního detailu. Na závěr kapitoly budou tyto synchronizační nástroje porovnány.

4.1 SQLyog

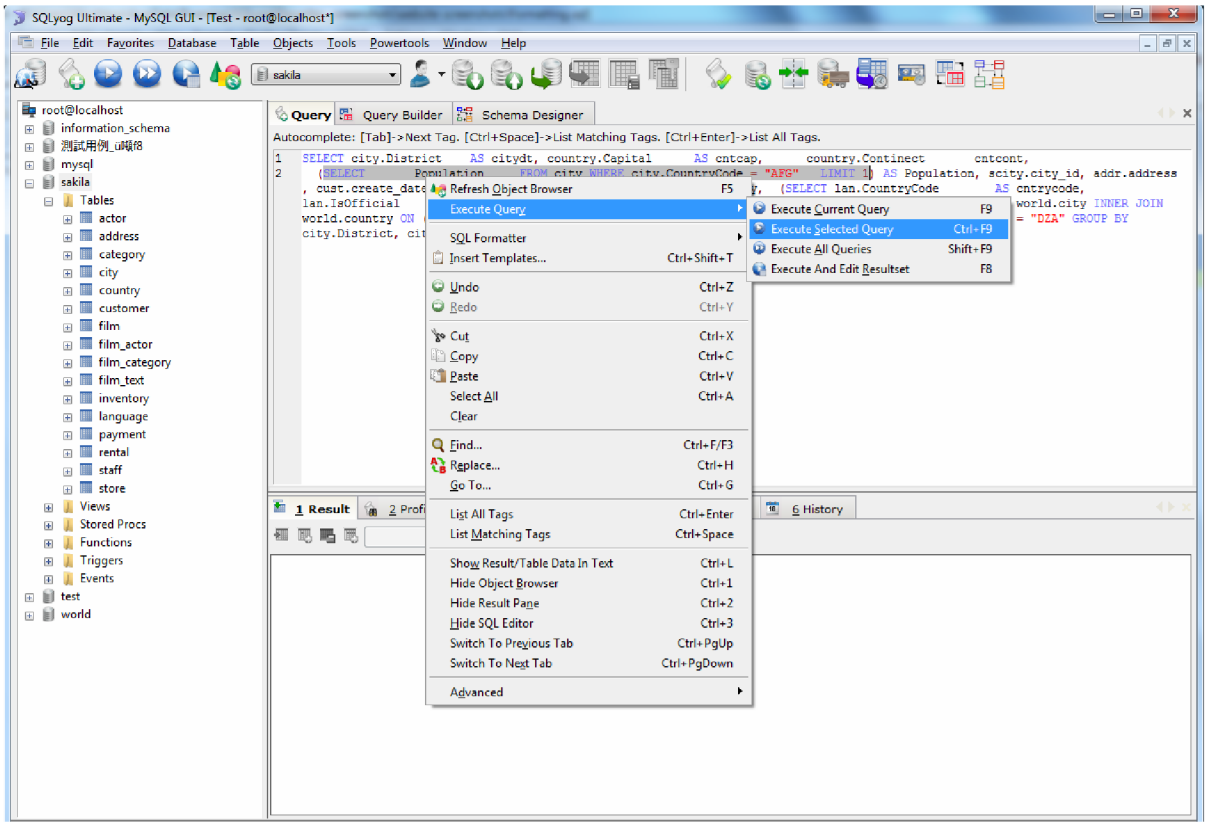
SQLyog je vysoce výkonný manažer a administrační nástroj, který kombinuje hlavní rysy prohlížečů MySQL databází, administračních programů a dalších MySQL GUI nástrojů do jediného rozhraní s intuitivním ovládáním. SQLyog je nástroj pro RDBMS MySQL. Tento software je vytvořený společností Webyog. SQLyog verze 0.9 byla pro veřejnost poprvé uvolněna v roce 2001 jako volně šiřitelný software po 8 měsících vývoje. SQLyog byl volně šiřitelný software až do verze 3.0, kdy se z něho stala komerční verze. Nejlevnější licence na jeden rok pro jednoho uživatele, která umožňuje synchronizaci, je verze *enterprise* za 139$. Nabízí také 30-denní trial verzi pro vyzkoušení, ve které lze synchronizaci vyzkoušet. Seznam podporovaných funkcí lze naleznout na adrese: http://webyog.com/en/sqlyog_feature_matrix.php. Aktuální verze je 8.2.

Hlavní rysy programu:

- Návrh databáze dle vizuálního schéma
- Podpora databáze MySQL včetně verze 6.x
- Mnoho funkcí pro správu databáze, provádění dotazů, testování atd.
- Obsahuje dotazový (Query) editor a výsledkový (Result) editor
- Umožňuje velké množství exportů (dump, XML, HTML, CSV) a importů dat
- Inteligentní dokončování kódu
- Podporuje připojení k databázi pomocí SSH a HTTP
- Podpora SSL spojení
- Řízená migrace dat
- Synchronizace dat či struktury

SQLyog funguje na platformě systému Windows od Windows 2000 až po Windows Vista a Server 2008 (podpora Windows 9x/ME/NT4 byla odstraněna ve verzi 5.0 kvůli nedostatku podpory

Unicode v těchto prvních verzích Windows). SQLyog byl také navržen pro práci pod Linuxem a různých UNIX (včetně Mac OS X) operačních systémů pomocí Wine prostředí. Další podmnožiny funkcí SQLyog Enterprise jsou k dispozici zdarma SJA(SQLyog Job Agent) pro Linux jako nativní utilita Linuxu. Díky tomu je možné specifikovat a testovat 'plánované úlohy' na prostředí Windows a přenášet nastavení do prostředí Linux. Na obrázku 4.1 lze vidět GUI programu. Více informací lze nalézt na adrese: http://www.webyog.com/en/.



Obr. 4.1 Program SQLyog

4.2 Microsoft Sync Framework

Microsoft Sync Framework je platforma pro synchronizaci dat od společnosti Microsoft, která lze použít pro synchronizaci dat mezi různými datovými sklady. Podle [7] Sync Framework umožňuje nejen synchronizaci souborů, databází, RSS a Atom systémů, ale také napsat vlastní providery a použít tak libovolné datové úložiště. Po této stránce je Sync Framework velmi silný nástroj a poskytuje flexibilní přístup k datovým úložištím. Podle [8] umožňuje integraci do různých protokolů, včetně bezdrátového přenosu a embedded zařízení.

Architektura Sync Frameworku je založena na tom, že se uživatel nemusí o přenos dat přes dané rozhraní starat, ovšem musí pomocí již hotových tříd nadefinovat způsob výměny dat. Tato transportní architektura je založena na komponentě ADO.NET API, což je komponenta („cesta“), která může být využita pro přístup k datům v různých datových zdrojích. Základní princip synchronizace spočívá v tom, že se vytvářejí a ukládají informace (metadata) o synchronizovaných

datech přímo u synchronizovaných dat. Metadata ukládají všechny potřebné informace o datech, které podléhají synchronizaci a jedná se většinou o informace o vložení, úpravě či smazání těchto dat. Na základě těchto metadat jsou pomocí tříd Sync Frameworku prováděny změny mezi synchronizovanými daty.

Sync Framework pro databáze funguje na principu metadat, která jsou zaznamenávána přímo do databáze. V případě nějakých změn v datech jsou pomocí triggerů ukládány změny do těchto metadat. Pro jednu tabulku je obecně nutné vytvořit 2 triggery, první pro aktualizaci metadat při příkazu UPDATE a druhý při příkazu DELETE. Při vykonání příkazu INSERT jsou metadata automaticky nastavena na výchozí hodnotu.

Pro ukládání metadat je nutný zásah do struktury databáze a také každá tabulka musí mít primární klíč. Jedinou výjimkou je případ, kdy chceme na klienta stáhnout všechna data bez ohledu na předchozí synchronizace, potom není žádná změna nutná. Zásahy do struktury nejsou striktně definovány, avšak existují jisté pravidla, jak strukturu tabulek upravit tak, aby uživateli při implementaci synchronizace usnadnila co nejvíce práci. Obecně lze podle [8] říct, že v každé tabulce je zapotřebí přidat 4 sloupce pro metadata a vytvořit pro každou tabulku další tabulku se třemi sloupci. Této tabulce se říká tabulka „náhrobků“ (tombstones) a obsahuje informace o smazaných řádcích. Nutné změny ve struktuře databáze jsou podle [8] rozebrány v následující tabulce.

	PK	Update	Insert	Delete	ID klienta (updaty)	ID klienta (inserty)	ID klienta (mazání)
Stažení inkrementálních insertů a updatů na klienta	Ano	Ano	Ano[1]	Ne	Ne	Ne	Ne
Stažení inkrementálních insertů, updatů a mazání na klienta	Ano	Ano	Ano[1]	Ano	Ne	Ne	Ne
Nahrání insertů na server	Ano	Ne	Ne	Ne	Ne	Ne[2]	Ne
Nahrání insertů a updatů na server	Ano	Ne	Ne	Ne	Ne[2]	Ne[2]	Ne
Nahrání insertů, updatů, a mazání na server	Ano	Ne	Ne	Ne	Ne[2]	Ne[2]	Ne
Obousměrné inserty a updaty s detekcí konfliktů	Ano	Ano	Ano[1]	Ne	Ano[3]	Ano[3]	Ne
Obousměrné inserty, updaty a mazání s detekcí konfliktů	Ano	Ano	Ano[1]	Ano	Ano[3]	Ano[3]	Ano[3]

Tab. 4.2 Nutné změny v tabulkách při synchronizaci [8]

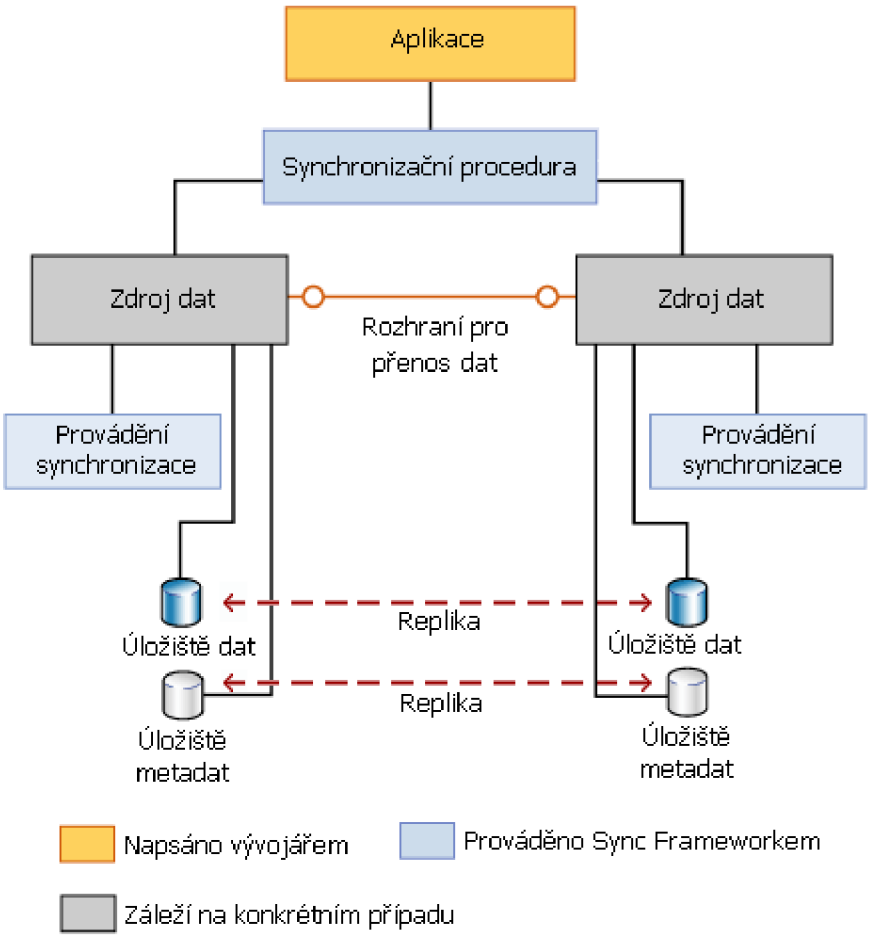
[1] Vyžadováno pokud potřebujeme rozlišit inserty a updaty.

[2] Vyžadováno pokud více klientů může změnit řádek a chceme tyto změny rozlišit.

[3] Vyžadováno pokud nechceme změny propagovat zpět na klienta, který je provedl.

Přestože je Sync Framework poměrně silný nástroj a nabízí řadu funkcí a různých přístupů pro různé situace, neobejdeme se vždy bez vlastního zásahu (řešení) a je nutné si napsat tzv. providery. Provider je objekt, který definuje připojení k databázi a definuje jak se budou vykonávat změny v databázi, které se mají provést po porovnání se synchronizovanou databází. Samotnou synchronizaci už provádí tzv. synchronizační agent. Napsání vlastního providera spočívá v definování funkcí, které vyhledávají změny, provádí změny dat a příslušných metadat za podpory databázových procedur, které musí být také implementovány. Napsání vlastního providera je více rozebráno v kapitole 6.2. Provider je základem celé synchronizace a vyžaduje to napsání značného kusu kódu.

Tento framework zahrnuje kompaktní a účinný model metadat, který umožňuje synchronizaci prakticky pro kteréhokoli účastníka. Architekturu celého synchronizačního procesu pro dva servery lze vidět na obrázku 4.3.



Obr. 4.3 Architektura Microsoft Sync Framework [9]

Sync Framework podporuje podle [10] topologie klient-server, klient-klient, a smíšené topologie. V topologii klient-server se všichni klienti synchronizují s centrálním serverem. V topologii klient-klient může každý klient synchronizovat data s dalším klientem, aniž by změny musely projít centrálním serverem. Smíšená topologie se skládá z kombinace klient-klient a klient-

server topologií. Sync framework také podporuje rozšířitelný model, což umožňuje integrovat více datových zdrojů do synchronizace.

Platforma .NET nepředepisuje použití žádného programovacího jazyka. Bez ohledu na to, v čem byla aplikace původně napsána, se vždy přeloží do mezijazyka Common Intermediate Language(CIL). Nejpoužívanější programovací jazyky pro vývoj .NET aplikací jsou C#, Visual Basic a Delphi. K dispozici je také řada dalších programovacích jazyků, například: Managed C++, IronPython, F# (funkcionální programovací jazyk), J# (jazyk velmi podobný Javě). Sync Framework se nejvíce používá spolu s Microsoft SQL Serverem a k synchronizaci lze zajisté nalézt hodně dokumentace přímo na stánkách Microsoftu.

Nespornou výhodou toho frameworku je možnost použití různých systémů řízení báze dat (SŘBD) pro synchronizaci, ovšem v případě databází je nutný zásah do struktury databáze a implementace vlastních providerů, což může být docela obtížné. Tato technika byla vyzkoušena a implementace je probrána v kapitole 6.2.

4.3 Replikace

MySQL má užitečný prvek označovaný jako *replikace*. Replikace podle [6] funguje tak, že k jedinému „pánovi“, master serveru, nebo zkráceně masteru, se může připojit několik „otroků“, slave serverů, zkráceně replik, a replika zase může naopak pracovat jako master. Mastery a repliky mohou být uspořádány v mnoha různých topologiích. Mohou se replikovat celé servery, jen některé databáze, nebo dokonce určovat, které tabulky a data se mají replikovat.

MySQL podporuje dva druhy replikace: příkazovou a řádkovou. Příkazová (neboli „logická“) replikace je k dispozici už od MySQL 3.23 a v ostrém provozu ji používá většina lidí. Podle [4] příkazová replikace funguje tak, že zaznamenává dotaz, který změnil data na masteru. Když pak replika přečte událost z relay logu (popsaný níže) a spustí ho, znovu se vykoná skutečný dotaz SQL, ten, který se vykonal na masteru. Na replice se tedy vykonávají přesně ty dotazy, které se vykonávaly na masteru.

Řádková replikace je novinkou v MySQL 5.1 a funguje tak, že se do binárního logu zaznamenávají skutečné změny v datech, takže se podobá tomu, jak implementuje replikaci Sync Framework [4].

Protože žádný z obou formátů replikace není vhodný pro každou situaci, MySQL se dynamicky přepíná mezi příkazovou a řádkovou replikací [5]. Oba druhy replikace pracují tak, že zaznamenávají změny dat v binárním logu mastera a přehrávají tento log na repliku. Obě jsou asynchronní – tj. u kopie dat není zaručeno, že se v každém okamžiku bude jednat o „nejčerstvější data“. Podle [4] není zde ani žádná garance toho, jak dlouhá může být latence na replice. Rozsáhlé dotazy mohou způsobovat, že budou repliky zpožděny v řádu sekund, minut, nebo dokonce i hodin za mastrem.

Replikace pro mastera obecně neznamená mnoho režie navíc. Požaduje sice, aby na masteru bylo zapnuto zaznamenávání do binárního logu, ale režie není velká. Kromě binárního logu přidává během normálního fungování něco režie mastera i každá připojená replika (většinou se jedná o síťový I/O).

Replikace je poměrně dobrá pro škálování čtení, protože je lze směrovat na repliku, ale není už tak dobrá pro škálování zápisů, pokud není navrhnuta opravdu dobře [4]. Když se k masteru připojí hodně replik, jednoduše to způsobí, že zápisy se budou muset udělat mnohokrát, na každé replice jednou. Celý systém pak bude omezen počtem zápisů, které umí vykonat nejslabší část systému.

Replikace se běžně používá při řešení následujících problémů:

- Distribuce dat – výměna dat mezi vzdálenými servery, většinou v zeměpisně vzdálených lokalitách
- Rozložení zátěže (load balancing) – může pomoci s distribucí čtecích dotazů přes několik serverů, funguje velmi dobře u aplikací, které intenzivně čtou.
- Zálohování – replikace může vypomoci se zálohováním, avšak nelze chápat repliku ani jako zálohu, ani jako náhradu záloh.
- Vysoká dostupnost a rychlá náhrada vypadlého serveru – replikace může zamezit, aby se MySQL stal tzv. kritickým místem výpadku, neboli aby kvůli němu padla celá aplikace.
- Testování při upgradu MySQL – podle [4] je běžnou praxí připravit repliku s upgradovanou verzí MySQL a nějakou dobu ji používat, aby jsme se přesvědčili, že dotazy pořád pracují tak, jak očekáváte. Teprve poté upgradovat všechny ostatní instance.

Na vysoké úrovni abstrakce je replikace jednoduchý postup složený ze tří kroků:

1. Master zaznamená změny ve svých datech do svého binárního logu. (Těmto záznamům se říká události binárního logu, binary log events.)
2. Replika zkopíruje události binárního logu masteru do svého logu, říká se mu relay log.
3. Replika přehraje události nacházející se v relay logu a aplikuje změny na svá vlastní data.

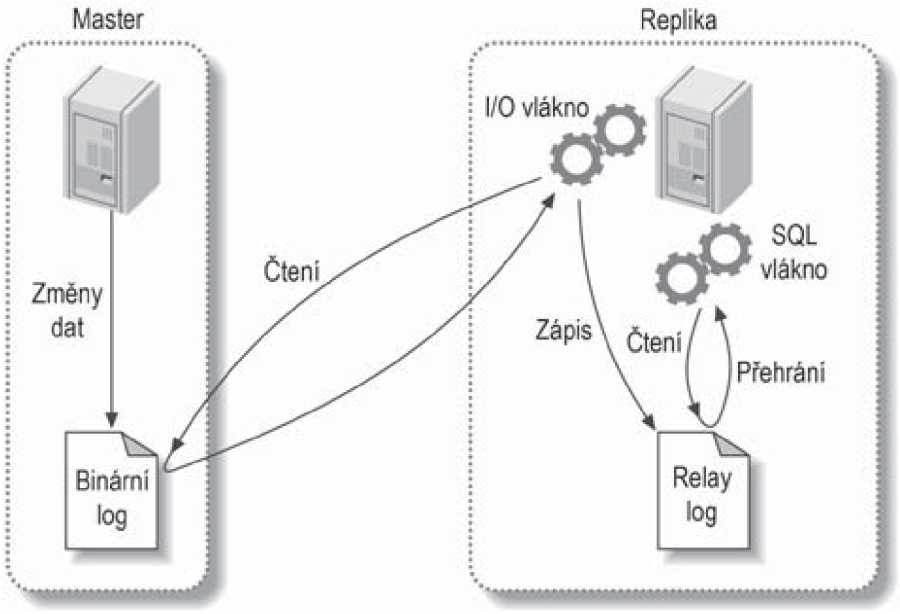
Toto je pouze přehled, ve skutečnosti je každý z těchto kroků dost složitý. Podrobněji je replikace znázorněna na obrázku 4.4.

První částí postupu je zaznamenávání do binárního logu na masteru. Těsně předtím, než se na masteru dokončí jakákoliv transakce, master zaznamená její změny do svého binárního logu. MySQL zapisuje jednotlivé transakce do binárního logu jednu po druhé, i když se příkazy v transakcích během vykonávání střídaly. Poté, co master zapsal události do binárního logu, sdělí úložným enginům, aby transakce potvrdily [5].

V dalším kroku zkopíruje replika binární log mastera na svůj vlastní pevný disk, do relay logu. Začne tím, že nastartuje zpracovatelské vlákno, kterému se říká I/O vlákno repliky (I/O slave thread). Toto I/O vlákno otevře obyčejné klientské připojení k masteru a následně nastartuje speciální proces binlog dump (není k němu odpovídající příkaz SQL). Proces binlog dump čte události z binárního logu masteru. Nedotazuje se na události. Jakmile dostihne mastera, jde spát a čeká na signál od mastera, že jsou k dispozici nové události. I/O vlákno zapíše události do relay logu repliky.

Poslední část celého postupu zpracovává SQL vlákno repliky (SQL slave thread). Toto vlákno čte a přehrává události z relay logu, což znamená, že aktualizuje data repliky tak, aby se shodovala s daty masteru. Dokud toto vlákno udrží krok s I/O vláknem, tak relay log obvykle setrvává v cache operačního systému, a tudíž relay logy mají velmi nízkou režii [4]. Události, které vykonává SQL vlákno repliky, mohou volitelně jít do vlastního binárního logu repliky, což se hodí zejména

v případech, kde daná replika slouží jako master pro další repliku a události jsou tedy šířeny dále po synchronizační topologii.



Obr. 4.4. Jak funguje MySQL replikace [4]

Na obrázku 4.4 jsou znázorněna pouze dvě replikační vlákna, která běží na replice, nicméně ještě existuje jedno vlákno na masteru: podobně jako jakékoliv jiné připojení k MySQL serveru, to připojení, jež replika otevře k masteru, nastartuje vlákno na masteru.

Tato replikační architektura odděluje na replice procesy získávání a přehrávání událostí, takže mohou být asynchronní. Jinak řečeno: I/O vlákno může pracovat nezávisle na SQL vláknu. Vnáší také do replikačního postupu jistá omezení - nejdůležitější z nich je to, že na replice je replikace serializována. To znamená, že aktualizace, které možná na masteru probíhaly paralelně, v různých vláknech, nemohou být na replice paralelizovány, a právě tohle podle [4] tvoří úzké hrdlo výkonu pro mnohé pracovní zátěže.

4.4 Porovnání synchronizačních nástrojů

V předcházející kapitole byly vypsány tři nástroje, které lze využít pro synchronizaci databází MySQL. Každý z těchto nástrojů má své výhody a nevýhody. Nedá se obecně říci, že se jeden hodí na všechny možné synchronizační topologie. Také záleží na programátorovi, jaké má zkušenosti, v jakém jazyce je napsaná aplikace pro databázi, zda lze aplikaci upravit nebo synchronizační proces musí být zcela oddělený od aplikace, jestli je možnost si zaplatit plnohodnotný program pro správu databáze, atd.

Podle mého názoru je pro synchronizaci MySQL databází nejlepší použít MySQL replikaci, pokud to verze MySQL serveru podporuje. Některé zásadní zlepšení replikace přinesla až verze 5.0.x,

také nejnovější verze 5.1.x se dočkala pár změn v replikaci. Pokud je to možné, tak je nejlepší upgradovat server na nejnovější verzi a přečíst si v manuálu podrobné a aktuální informace o replikaci.

Jeden z několika důvodů, proč zrovna použít replikaci je zajisté ten, že je zde podpora přímo od MySQL, nabízí mnoho topologií synchronizace a její použití většinou nepotřebuje žádný zásah do struktury databáze nebo byť jen minimální. Navíc je zdarma a velmi se v praxi využívá. Také umožňuje replikaci master-master, což bude topologie pro databázový systém evidence implantabilních zdravotnických prostředků.

Využití platformy Microsoft Sync Framework by v tomto konkrétním případě znamenalo velký zásah do struktury databáze a navíc tuto platformu lze využít pouze programovacími jazyky podporující platformu .NET, což Java přímo není. Databázový systém evidence implantabilních zdravotnických prostředků je napsán v jazyce Java a použití Sync Frameworku by znamenalo oddělení klientského programu a procesu synchronizace a s tím spjaté značné nevýhody. Použití Sync Framworku se serverem MySQL lze doporučit při použití klientského programu napsaném např. v C#, kde lze zajisté nalézt podporu přímo od Microsoftu a proces synchronizace může být zcela pod kontrolou přímo v aplikaci. Tato platforma nabízí nemalé možnosti synchronizace, ale hodí se zejména pro programy napsané pro platformu .NET.

Poslední zmiňovaný nástroj je SQLyog. Tento program se jeví jako velice vyvedený, nabízí mnoho funkcí a to nejen ve směru synchronizace. Každého určitě zaujme intuitivním ovládáním a pěkným zpracováním programu. Nevýhodou je ovšem to, že není zcela zdarma a pro dlouhodobější využívání je třeba si tento program zakoupit. Podmínkou pro správnou synchronizaci je pouze použití primárního klíče u všech tabulek. Tento program zajisté najde uplatnění tam, kde nemůže být upravována struktura databáze a hledáme již hotový synchronizační produkt, který nemusí na 100% splňovat naše požadavky, a jsme ochotni za tento produkt zaplatit. Pak SQLyog je to pravé řešení. Jelikož pro implementaci v databázovém systému implantabilních zdravotnických prostředků je tlačeno na použití nekomerčních prostředků, tak tento produkt bohužel není pro realizaci vhodný, ale bude vyzkoušen a porovnám s ostatními nástroji.

5 Návrh synchronizačního procesu

V této kapitole budou vypsány požadavky na synchronizaci databázového systému evidence implantabilních zdravotnických prostředků (dále jen *TiSoft* – Timplant software). Stručně se seznámíme s *TiSoftem,* topologií synchronizačního procesu a potřebných nástrojů k implementaci.

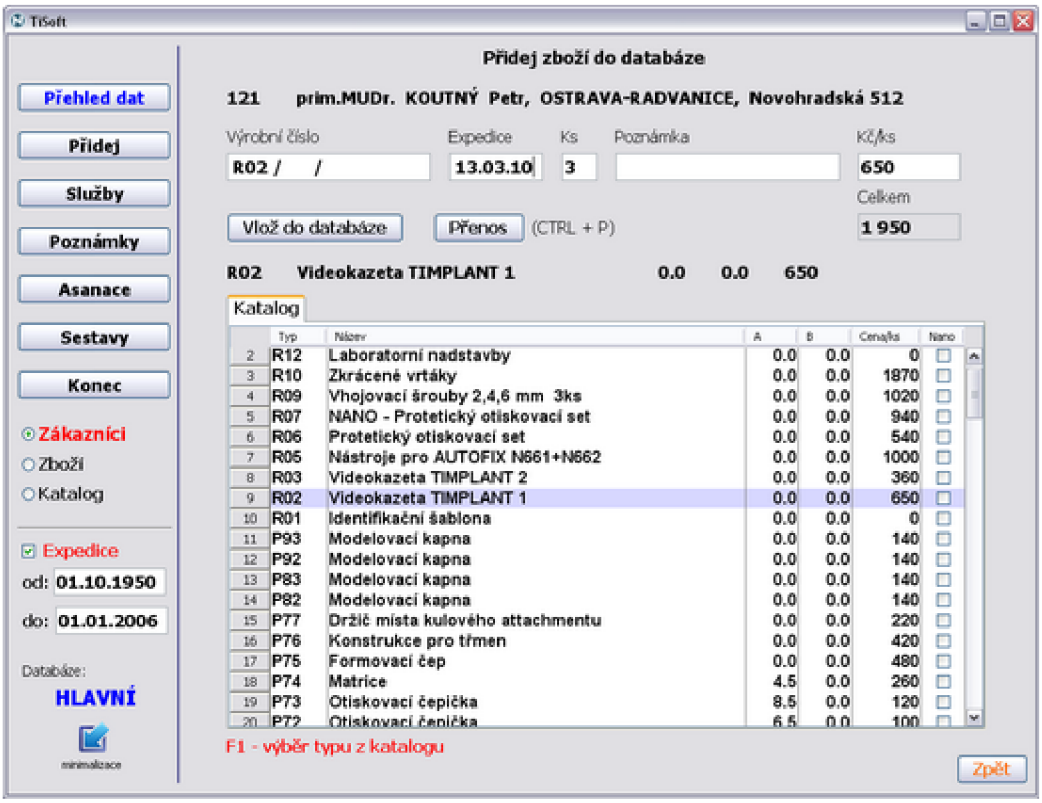
5.1 TiSoft

TiSoft je databázový neboli uživatelský software, který byl realizován v rámci bakalářské práce v roce 2008. Tento software byl „na míru“ vytvořen dle požadavků firmy Timplant s.r.o. Firma se zabývá vývojem, výrobou, expedicí zubních implantátů a jejich příslušenství. Všichni zákazníci, vyrobené a expedované zboží a další údaje potřebné pro splnění podmínek ISO norem jsou ukládány do databáze MySQL

Pro správný chod firmy je vkládání, vyhledávání a úprava dat velmi důležitá. Tyto data také musí být dle nařízení vlády archivována po dobu nejméně 10 let od data expedice k odběrateli a v případě potřeby (kontroly nebo nežádoucí příhody) tyto informace použít. Data jsou také používána pro kontrolu v účetnictví. Z toho vyplývá, že je nutné s těmito daty velmi pečlivě nakládat a spravovat je. V procesu synchronizace na to musí být brán velký zřetel a je nezbytné, aby data nebyla nijak poškozena, ztracena nebo nesprávně zaznamenána.

TiSoft je vytvořen pomocí objektově orientovaného programovacího jazyka Java, který vyvinula firma Sun Microsystems spolu s databázovým systémem MySQL. Jeho nasazení proběhlo v roce 2008 a neustále je tento software upravován dle požadavků firmy. Aktuální podobu lze vidět na obrázku 5.1. Program je nyní nainstalován ve dvou kancelářích a uvažuje se do budoucna o nainstalování do třetí nevybavené kanceláře. Jeho časté úpravy a vylepšení vedly k myšlence synchronizovat data mezi dvěma oddělenými kancelářemi tak, aby data v obou kancelářích byla vždy aktuální.

Databázový systém MySQL používá jednu databázi *timplant*, která obsahuje několik tabulek. Některé tabulky obsahují až tisíce evidenčních záznamů a všechny jsou typu MyISAM, což je nejpoužívanější, platformě neutrální a zároveň výchozí formát úložiště dat (storage engine) v databázovém systému MySQL. *TiSoft* používá databázový server jako localhost a MySQL server je tedy nainstalován na každém počítači a připojení k němu je na základě localhostu. Toto „nevšední“ řešení se na první pohled může zdát nepochopitelné, avšak má své odůvodnění. Tato implementace byla použita zejména kvůli bezpečnosti dat, ušetření finančních nákladů, uchování dat v kanceláři, plnou kontrolou nad daty a jejich zabránění proti zneužití. Některé osobní data zákazníků jsou velice citlivá, a kdyby se dostala do nesprávných rukou, tak by mohla způsobit nemalé problémy. V případě více počítačů má každý počítač nainstalovaný MySQL server, kde se k němu připojuje přes localhost.

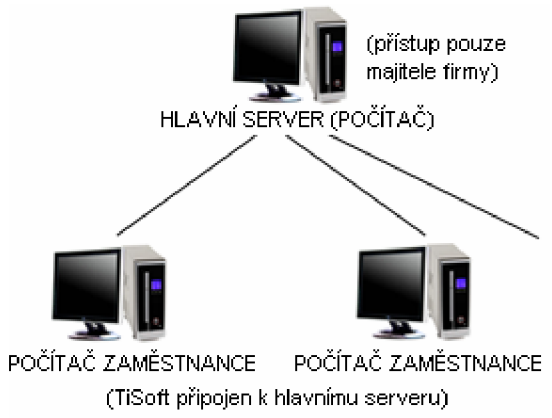


Obrázek 5.1 Ukázka programu TiSoft

Mít více počítačů ve firmě a používat k *TiSoftu* lokální server znamenalo pomalé, ale jisté zadělávání si na problém a to zejména v distribuci nových dat v ostatních počítačích. Ze začátku to byl podřadný problém a hlavní důraz se kladl na funkčnost, správnost dat a odladění všech možných komplikací. Ovšem po nějaké době byl software upraven a z původních budoucích problémů se staly problémy aktuální a to zejména s přenášením čerstvých dat na okolní počítače. Tento nemalý zádrhel se začal řešit a prozatimním řešením bylo data upravovat jen na jednom (hlavním) počítači a aktuální data přenášet na ostatní počítače pomocí zálohovacího systému. Metoda „vytvoř zálohu“, „nahrej na flešku“, „zkopíruj na jiný počítač“ a „obnov data ze zálohy“ nebyla vůbec efektivní, byť fungující, a proto se hledal jiný způsob propagace čerstvých dat.

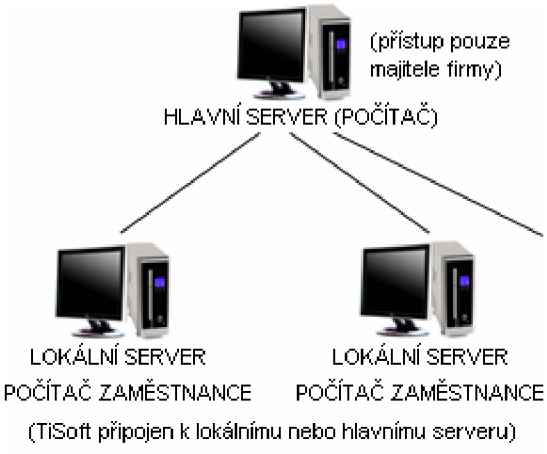
První myšlenkou bylo používání pouze jednoho serveru (hlavního serveru), kde se ostatní uživatelé musí připojit. Prozatímní dva počítače jsou síťově propojeny a budoucí nové počítače by do sítě byly také přidány. Návrh připojení k hlavní databázi demonstruje obrázek 5.2. Síťové propojení všech počítačů zajišťuje trvalé spojení všech počítačů a umožňuje vzájemnou komunikaci. Tato realizace není nijak obtížná a ve zdrojovém kódu programu se pro připojení k databázovému serveru zvolí pouze IP adresa a potřebné parametry vzdáleného serveru. Toto řešení se na první pohled zdálo být rozumné, všechny počítače se v síti mohly připojit na hlavní server a pomocí TCP/IP data spravovat z *TiSoftu*. Avšak nastal problém s tím, že (hlavní) server musel být vždy spuštěný a s tím souviselo zapnutí (hlavního) počítače. Trvalé spuštění serveru je ovšem omezující podmínka, která nemůže být vždy splněna, protože každý počítač je zvlášť v kanceláři a přístup k hlavnímu počítači má jen sám majitel firmy a tedy spuštění (hlavního) serveru zaměstnancem je v tomto případě vyloučeno. Čtení, zápis a úprava dat by v tomto případě mohla být provedena ostatními zaměstnanci,

pouze pokud by měl sám majitel zapnutý počítač. Tato podmínka je velmi omezující a proto se hledalo lepší řešení pro přenos dat mezi počítači.



Obr. 5.2 Použití pouze jednoho MySQL serveru

Dalším nápadem pro zlepšení situace s distribucí dat bylo využít nainstalované MySQL servery na každém počítači a v případě, že hlavní server není zapnutý, tak se připojit na lokální server a pracovat s daty přes localhost, viz obrázek 5.3. Následující návrh se sice nezbavil problému s neustálým zapnutím hlavního serveru, ale bylo umožněno data číst a pracovat s nimi (nikoliv modifikovat) bez ohledu na připojení k hlavnímu serveru. Myšlenka tohoto návrhu byla taková, že pokud je hlavní server zapnutý, tak se ostatní uživatelé mohou připojit, libovolně modifikovat data a nejnovější data budou ukládána pouze na hlavní server. Ale v případě, že je hlavní server vypnutý, tak může každý počítač pracovat nad lokální databází a číst data pro svou potřebu. Lokální databáze ovšem není nijak spojena s hlavní databází a tak čerstvé data si zase zaměstnanec musí přenášet ručně na lokální databázi. Tento návrh byl dokonce implementován a uživatel si mohl vybrat, ke kterému serveru se chce připojit. Výhoda této topologie je pouze v tom, že zaměstnanec může číst data bez ohledu na zapnutí hlavního serveru, ale modifikovat musí až po připojení k serveru hlavnímu. Nedostatek lze jasně vidět. Řešení toho problému nebylo nijak jednoduché, proto se musely využít jiné techniky pro distribuci dat po síti a to synchronizace databází.



Obr. 5.3 Použití lokálního a hlavního serveru

5.2 Požadavky na synchronizaci dat

Jak už bylo zmíněno, tak všechny firemní počítače mají nainstalovaný MySQL server a tudíž pracují se svou lokální databází. Požadavek na propojení těchto dat, byl takový, že všechny databáze budou spolu nějakým způsobem propojeny tak, aby umožňovaly zaměstnancům kdykoli upravovat data ze svého počítače bez ohledu na spuštění ostatních počítačů. Tento požadavek se většinou řeší tím, že je zabudován speciální server a ten je neustále v provozu, ovšem takovými prostředky firma nedisponuje a jediným řešením je zvolit nějaký synchronizační nástroj.

Dalším požadavkem bylo zanechání stávajících funkcí programu TiSoft a synchronizaci dat ponechat zcela automatickou, bez zásahu uživatele. Tudíž je třeba vybrat nástroj, který synchronizaci databází nastaví a pak už není nutný žádný zásah uživatele a programátora. V krajní situaci je zásah programátora jen minimální.

Třetím bodem byla bezpečnost dat a to z hlediska útoku na synchronizační proces a znemožnění druhé osobě získat firemní data. Tento aspekt je velmi důležitý pro návrh a musí být brán v potaz při realizaci celého synchronizačního procesu. Prozrazení firemních dat by znamenalo velký problém pro firmu.

Posledním požadavkem byla možnost rozšíření synchronizačního procesu na ostatní počítače, které mohou být v budoucnu zakoupeny a připojeny do sítě. Nyní firma disponuje pouze dvěma počítači, ale zanedlouho zvažuje o zakoupení dalšího a zapojení do sítě.

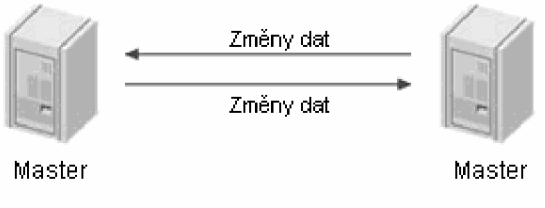
5.3 Návrh synchronizačního procesu

Synchronizační topologie, která se musí použít v tomto případě, není na první pohled zcela zřejmá. Některé synchronizační topologie pro MySQL replikaci jsou pro přehled v příloze této práce, avšak zajisté by se tyto topologie daly implementovat v rámci SQLyogu či za použití Sync Frameworku. Připomeňme, že data musí být přenášena obousměrně a to nezávisle na běhu ostatních serverů (počítačů). Základem návrhu bude použití již nainstalovaných MySQL serverů na každém počítači. Po bližším prozkoumání problému a různých synchronizačních struktur se zde nabízí topologie zvaná *master-master*.

5.3.1 Topologie master-master

Tato topologie se skládá ze dvou serverů, které si navzájem vyměňují data. První server posílá změny v databázi serveru druhému a druhý server posílá změny serveru prvnímu a to zcela nezávisle. Topologie master-master se také někdy nazývá duální master. Tato topologie se vyznačuje tím, že oba servery mohou současně přidávat, upravovat či mazat data a přenášet tyto změny mezi sebou.

Tuto konfiguraci lze vidět na obrázku 5.4, na první pohled se jedná o velice jednoduchou topologii, avšak přináší sebou mnoho problémů. Většinou se master-master konfigurace používá pro speciální účely a mnoho oborníků tuto topologii nedoporučuje používat a radí, ať se programátoři snaží najít jinou konfiguraci a tím se vyhnout nemalým problémům, které mohou při používání nastat a zajisté některé nastanou.



Obr. 5.4 Topologie master-master

Největší problémy s takovouto konfigurací představují konfliktní změny. Seznam možných potíží zapříčiněných tím, že máme dva rovnocenné mastery s právem zapisovat není malý. Potíže se obvykle objeví tehdy, když nějaký dotaz změní řádek simultánně na obou serverech, nebo když vkládá současně na obou serverech do nějaké tabulky, která má sloupec s atributem AUTO_INCREMENT. Podle [4] bylo do verze MySQL 5.0 bylo přidáno několik funkcionalit, které činí tuto topologii poněkud bezpečnějším. Jedná se o nastavení konfiguračního souboru serveru a proměnné `auto_increment_increment` a `auto_increment_offset`. Tato nastavení umožňují serveru automaticky generovat nekonfliktní hodnoty pro dotazy INSERT. Tímto nastavením se lze vyrovnat několika problémům, ale i přesto je pořád nebezpečné povolit zápis oběma masterům. Aktualizace, které se na těchto dvou serverech vykonávají v různém pořadí mohou způsobovat, že se data tiše „rozhodí“ a přestanou být synchronizovaná. Příkladem může být situace, kdy je tabulka o jediném sloupci a s jediným řádkem, který obsahuje hodnotu 1. Nyní předpokládejme, že se vykonají tyto dva příkazy a provede se synchronizace dat:

Na prvním masteru:

```
mysql > UPDATE my_table SET my_column=my_column + 1;
```

A na druhém masteru:

```
mysql > UPDATE my_table SET my_column=my_column * 2;
```

Výsledek? Jeden server bude mít hodnotu 4, druhý hodnotu 3. A to z toho důvodu, že na prvním masteru se nejdříve přičítá hodnota 1 (výsledek 2) a pak se při synchronizaci násobí (výsledek 4). Na druhém serveru se nejdříve násobí (výsledek 2) a pak se při synchronizaci přičítá hodnota 1 (výsledek 3).

Při implementaci musí být brán ohled na problémy, které můžou při provádění změn v databázi nastat. Synchronizační prostředky, které budou implementovány, se snaží s touto problematikou vypořádat každý po svém, ale ve většině případů se to neobejde bez zásahu programátora a implementace vlastních ošetření konfliktních situací. Výsledkem synchronizačního procesu musí být vždy stejná data na obou masterech a v případě konfliktu dat musí být tato situace ošetřena, tak, aby obě databáze uchovávaly stejná data.

6 Implementace synchronizačních technik

Pro implementaci topologie master-master se servery MySQL budou použity již zmiňované prostředky Microsoft Sync Framework, MySQL replikace a program SQLyog. Cílem implementace nebude úplné a odladěné řešení, ale porozumět těmto technikám, uvědomit si problémy při implementaci, prozkoumat nutné zásahy do struktury databáze a vyzkoušet si realizovat tyto nástroje pro reálné využití v databázovém systému evidence implantabilních zdravotnických prostředků.

Dalším záměrem bude získat určitý nadhled nad synchronizací databází a vyzkoušet si různé nástroje a přístupy pro přenos dat mezi MySQL servery. Budou zde probrány problémy při implementaci a návrh či řešení těchto problémů v konkrétním nástroji. V závěru kapitoly budou vyzkoušené techniky porovnány s ohledem na nasazení v praxi a bude vybrán nejlepší nástroj pro realizaci a ten bude dále upraven tak, aby mohl být reálně nasazen v *TiSoftu*.

Při implementaci těchto technik se budu zejména soustředit na distribuci dat mezi databázemi, se kterou úzce souvisí bezpečnost dat dle kapitoly 3.1. U každého nástroje bude také probrána problematika spouštění a hardwarové či softwarové nároky.

Všechny použité techniky budou implementovány na operačním systému Windows XP, protože TiSoft byl také vyvinut pro tento operační systém.

6.1 Synchronizace s využitím SQLyog

SQLyog je první s vybraných nástrojů, který umožňuje synchronizaci. Je to nástroj od společnosti Webyog, která vytváří inovativní programy pro správu dat pro tisíce zákazníků ve více než 100 zemích, od podniků přes malé firmy až po domácí uživatele. SQLyog podporuje komunikaci se servery MySQL od verze 3.23 do 6.x a tudíž umožňuje spravovat od nestarších až po nejnovější vydání MySQL. Tento nástroj neumožňuje pouze synchronizaci databází, ale nabízí nemalou řadu funkcí pro komplexní správu databází, které byly vypsány v kapitole 3.2.1.

SQLyog umožňuje dva synchronizační nástroje a to synchronizace struktury databáze a synchronizace dat. Pomocí záložky **Powertools -> Schéma Synchronization Tool** se lze dostat do synchronizační rutiny, která umožňuje porovnat strukturu dvou databází a případné změny upravit tak, aby se schéma obou databází rovnalo. Toto je velmi cenný nástroj, protože dvě databáze nemusí mít vždy stejnou strukturu, a proto je vždy lepší tyto struktury porovnat a případně tyto struktury změnit. Tento nástroj je velice jednoduchý na použití a pouze pomocí tlačítka **Compare** lze vygenerovat příkazy, které zobrazí SQL dotazy pro změnu struktury. Dále nám program umožňuje tyto příkazy vykonat a tím synchronizovat obě struktury, a nebo uložit SQL dotazy pro pozdější využití.

Stejná struktura obou databází, ve kterých mají být synchronizovaná data, je dobrým předpokladem pro správný chod synchronizačních nástrojů i mimo SQLyog. Některé synchronizační

rutiny se sice dokážou vypořádat s malými odlišnostmi ve struktuře, ale vždy je lepší strukturu obou databází překontrolovat a případně upravit. Pro zjištění schématu tabulky se v MySQL může využít příkaz `DESCRIBE <název_tabulky>` nebo `EXPLAIN <název_tabulky>`.

Druhý nástroj pro synchronizaci samotných dat lze vyvolat pomocí záložky **Powertools -> Database Synchronization Wizzard**, kde lze vytvořit synchronizační úlohu (job) a tu si také uložit pro pozdější využití. SQLyog má pouze jednu omezující podmínku pro využití synchronizace a to nutnost mít v každé tabulce primární klíč, jinak se data nemusí synchronizovat správně. V případě databáze pro TiSoft to znamená přidání primárního klíče s názvem *id* do tabulky *zbozi* a nastavit mu například parametry *auto_increment*. Samozřejmě se musí všechny hodnoty *id* v tabulce nastavit tak, aby nevznikl konflikt v databázi. Znamená to jen malou úpravu v databázi a minimální úpravou aplikace TiSoft. Sloupec *id* by byl využit pouze pro synchronizační účely.

Pro vytvoření úlohy je třeba nastavit několik parametrů, kde prvním z nich je topologie synchronizace. SQLyog nazývá synchronizační topologii, která je v tomto případě potřeba, jako *obousměrnou* (two-way). Databáze, které se mají synchronizovat, se obecně nazývají zdrojová a cílová. V případě konfliktů, které mohou nastat s primárními klíči, jsou cílová data nahrazena daty ze zdroje. Dále je třeba vybrat tabulky, které se mají synchronizovat, a nastavit, zda se má vygenerovat skript, který byl použit pro změnu dat v obou databázích. Tento skript obsahuje SQL příkazy a lze tedy přečíst, jaké změny se obě databáze dočkaly. Lze také nastavit zpracování chyb v průběhu synchronizace a umožňuje tedy přerušit proces v případě chyby nebo oznámení na e-mail. Upozornění na e-mail je velice dobrá funkce a administrátor se tak může v mžiku dozvědět o vzniklých problémech. Poslední nastavení umožňují provést synchronizaci, uložit úlohu pro pozdější využití a naplánovat spouštění této úlohy pomocí nástroje *naplánované úlohy* ve Windows. Pokud je provedena synchronizace, tak se objeví výsledek synchronizace, kde pro každou tabulku databáze je vypsán počet řádků dat a pak kolik řádků bylo ve zdrojové nebo cílové databázi vloženo, upraveno či smazáno. Uživatel tedy může ihned vidět kolik řádků bylo synchronizováno, viz. tabulka 6.1. V případě nějaké chyby je také vypsáno chybové hlášení.

SQLyog umožňuje také spuštění již připravené úlohy z příkazového řádku s parametry:

`sja [job_file`[4]`] -l[log_file`[5]`] -l[session_file`[6]`]`

Table	SrcRows	TgtRows	Inserted	Updated	Deleted
`zakaznik` (zdrojová)	348	342	6	0	0
`zakaznik` (cílová)	348	348	0	0	0
`zbozi` (cílová)	2590	2598	0	0	0
`zbozi` (target)	2598	2590	8	0	0
`katalog` (zdrojová)	240	240	0	3	0
`katalog` (cílová)	240	240	0	0	0

Tab. 6.1 Výsledek synchronizačního procesu

[4] job_file – uložená synchronizační úloha (např. jobfile.xml)

[5] log_file – soubor, do kterého je vypsána tabulka a případné chyby (např. log.txt)

[6] session_file – soubor, který se automaticky vytvoří s úlohou (např. sjasession.xml)

Spustit úlohu lze tedy pomocí příkazu, což je velmi výhodné, protože synchronizace může být spuštěna přímo s TiSoftu např. pomocí `Runtime.getRuntime().exec("sja...")`. Problém ovšem nastane v tom, kdy synchronizaci spouštět a nejspíš by se synchronizace musela spouštět po každé změně v databázi, což sebou přináší určité výpočetní nároky. Podrobněji budou příslušné problémy popsány v kapitole 6.4.

Princip synchronizace spočívá v tom, že používá SQL dotazy na servery, porovnává data za použití kontrolních součtů tabulek (checksum[7]) a provádí jednoduché příkazy INSERT, UPDATE a DELETE, které databáze synchronizují. Je to velice jednoduchá metoda, avšak také přináší některé úskalí. Sun Microsystems na svých stránkách zmiňuje, že pokud jsou kontrolní součty pro dvě tabulky různé, pak se v něčem liší. Nicméně skutečnost, že dvě tabulky mají stejný kontrolní součet, nemusí vždy znamenat, že jsou totožné. Z toho vyplývá, že tato metoda nemusí být vždy stoprocentní a proto je zapotřebí tuto techniku řádně otestovat před nasazením do provozu.

Testování celé synchronizace nepřineslo velké úspěchy a v některých případech synchronizace nefungovala vůbec. Pokud bylo použito vkládání nových řádků (INSERT), tak to vždy proběhlo v pořádku na obou databázích. V případě mazání záznamů (DELETE) synchronizace fungovala přesně naopak, než měla, a místo smazání záznamů na cílové databázi byly záznamy znova přidány tam, kde byly smazány. SQLyog tedy nerozpoznal DELETE příkazy a označil je jako INSERT. Poslední příkaz, který dělal problémy je UPDATE. Tento příkaz fungoval, pouze pokud byla data upravována na zdrojové databázi, v případě úpravy cílové databáze data nebyla distribuována a namísto toho byla nahrazena zpět na data původní. Pokud byla zdrojová databáze vyměněna za cílovou, tak se chování tohoto programu nijak nezměnilo. Tato synchronizace dat připomíná topologii master-slave, kde jsou data upravována pouze na masteru a změny jsou distribuovány na slave. Pouze příkaz DELETE této topologii neodpovídá.

Z výsledných testů lze jasně vidět, že synchronizace není vůbec uspokojivá a SQLyog má s touto partií zajisté co dohánět. Porovnání této techniky s ostatními bude rozebráno v kapitole 6.4.

---

[7] MySQL příkaz `CHECKSUM TABLE <název_tabulky>` umožňuje zobrazit kontrolní součet tabulky. CHECKSUM závisí na datech v tabulce, pokud se změní, tak se také změní CHECKSUM. Lze použít pouze pro tabulky typu MyISAM. Používá se většinou pro porovnání dvou tabulek, zda jsou shodné nebo nedošlo ke změnám.

6.2 Synchronizace pomocí Microsoft Sync Framework

Druhou vybranou technikou pro synchronizaci je Microsoft Sync Framework. Je to sadu objektů a tříd, které umožňují zajišťovat synchronizaci různých obsahů. Microsoft Sync Framework umožňuje synchronizovat různé typy databází a také synchronizovat data mezi jinými systémy řízení báze dat (SŘBD). Sync Framework poskytuje velkou podporu zejména pro servery od firmy Microsoft. Nabízí celou řadu objektů a tříd, které spolupracují s Microsoft servery a velice usnadňují nakonfigurování synchronizace. V případě podpory jiných serverů je situace jiná, tato platforma již nenabízí komplexní a jednoduché řešení a implementace synchronizace je o mnoho komplikovanější. Sync Framework lze použít pro dvě databáze, které jsou nainstalovány na jednom serveru, což je velká výhoda zejména při testovacích účelech oproti MySQL replikaci.

Vyzkoušení této platformy proběhlo ve vývojovém prostředí Visual Studio 2008 s použitím programovacího jazyka C#, se kterým jsem neměl zatím žádné zkušenosti. Po dohodě s vedoucím této práce byla tato platforma nejdříve vyzkoušena na serverech od Microsoftu (SQL SERVER 2008) a to zejména z důvodu přímé podpory od Microsoftu. Vzhledem k tomu, že Sync Framework není na trhu příliš dlouho, tak informace o této platformě není snadné najít a ve většině případů jsem byl nucen použít pouze manuál nebo stránky od Microsoftu[8]. Microsoft na svých stránkách uvádí značné množství informací, avšak v případě synchronizací databází jsou informace o synchronizaci poněkud strohé, nejasné a v některých případech nedostačující. Realizace celé synchronizace je ukázána na poměrně složitém příkladu a porozumění celé struktury je obtížnější, než se na první pohled zdálo.

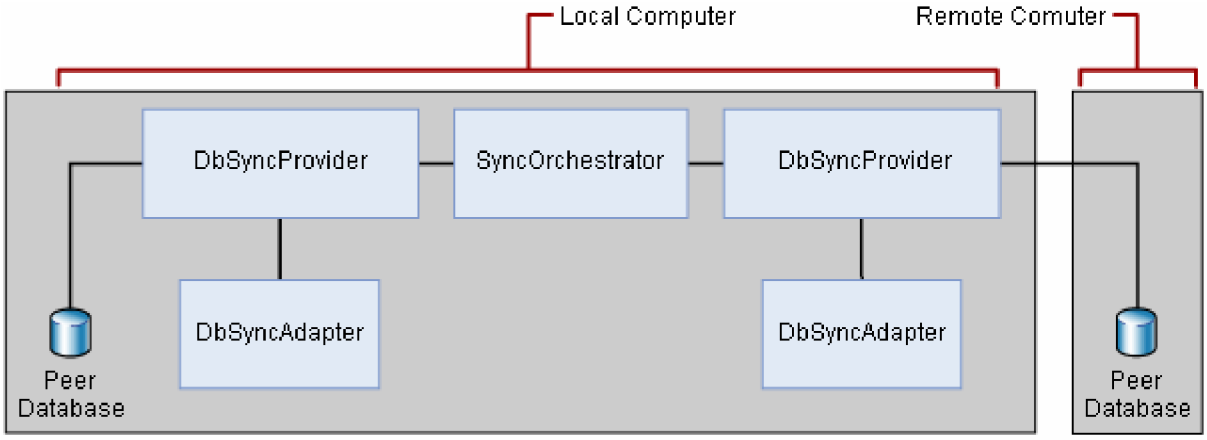
Sync Framework nazývá synchronizaci mezi vzdálenými databázemi jako *peer-to-peer synchronizaci*. Základní podstata synchronizace databází je v tom, že všechny změny (INSERT, UPDATE, DELETE) jsou zaznamenávány do databáze v podobě metadat, ty jsou potom porovnávány a jsou prováděny požadované akce. Nutností je tedy zásah do struktury databáze a v případě TiSoftu to znamená přidání čtyř sloupců v každé tabulce a přidání jedné tabulky pro každou tabulku. Každá synchronizovaná tabulka musí mít také primární klíč. Přidané tabulky se obvykle nazývají „tombstones“ a slouží k udržování informací o smazaných položkách. Sloupce těchto tabulek obsahují ID smazané položky, ID databáze[9] a tzv. „stamp“ což je časové razítko, kdy byla položka smazána. Pokud je nějaká položka z databáze smazána, tak je zaznamenána do této tabulky a při synchronizačním procesu jsou informace o smazaných položkách brány přímo z těchto tabulek. Sloupce, které byly přidány do existujících tabulek, jsou využívány pro informaci o příkazech INSERT a UPDATE. Pro každý z těchto příkazů slouží dva sloupce s údaji o ID databáze[9] a časové razítko o změně položky. Aktualizaci metadat zajišťuje trigger pro příkazy UPDATE a trigger pro příkazy DELETE. V případě příkazu INSERT jsou potřebná metadata nastavena na výchozí hodnoty.

---

[8] http://msdn.microsoft.com/en-us/sync/default.aspx

[9] Používá se pro odlišení, která z databází provedla změny.

Pro implementaci celé synchronizace je zapotřebí využít několik tříd. Pro názornou ukázku slouží obrázek 6.2.



Obr. 6.2 Architektura peer-to-peer synchronizace [11]

Základem této architektury je nastavení objektu `DbSyncAdapter`, který slouží jako most mezi objektem `DbSyncProvider` a databází. `DbSyncAdapter` musí být definovaný pro každou tabulku v databázi, která je synchronizována. Tento adaptér určuje konkrétní příkazy, které jsou nutné pro interakci s databází. Jsou to příkazy: `SelectIncrementalChangesCommand`, `InsertCommand`, `UpdateCommand`, `DeleteCommand`, `SelectRowCommand`, `InsertMetadataCommand`, `UpdateMetadataCommand`, `DeleteMetadataCommand`. Každý konkrétní příkaz zahrnuje spuštění procedury v databázi, která musí být pro tyto účely specifikována. Výsledkem nastavení těchto příkazů je definování struktury a dat, které mají být přenášena a zaznamenána. Pro každou tabulku jsou tedy známy změny v datech. Implementace všech příkazů a procedur pro každou tabulku je velice obtížná a proto v tomto případě bylo použití Sync frameworku pouze vyzkoušeno a otestováno.

Objekt `DbSyncProvider` umožňuje připojení k dané databázi. V tomto objektu jsou uloženy informace o tabulkách, které jsou povoleny pro synchronizaci. Umožňuje také aplikaci načítat změny, které nastaly v databázi od poslední synchronizace. Tento objekt také detekuje možné konfliktní změny. `DbSyncProvider` se vytváří pouze jeden pro každou databázi. Provádění všech synchronizačních změn se nastavuje v objektu `DbSyncAdapter`.

`SyncOrchestrator` zastřešuje celou synchronizační rutinu. Specifikuje tzv. lokálního a vzdáleného providera, který komunikuje s databází. Dále také určuje typ synchronizace, což je v tomto případě `UploadAndDownload`. Tento objekt také vykonává synchronizační rutinu zavoláním funkce `Synchronize()`.

Vyzkoušení tohoto frameworku proběhlo po konzultaci s vedoucím práce na příkladu, kde byl použit SQL SERVER 2008 a databáze s jednou tabulkou. Tento příklad byl stáhnut ze stránek Microsoftu[10] a byl příslušně upraven tak, aby se mohla synchronizace pomocí toho frameworku vyzkoušet a otestovat. Databáze byla samozřejmě upravena do odpovídající podoby a byla přidána

[10] http://msdn.microsoft.com/en-us/library/cc807286.aspx

další tabulka pro záznam metadat. Pro tuto tabulku byly sestaveny 2 triggery a 9 procedur, které byly zapotřebí pro objekt `DbSyncAdapter`. Pro statistiku vykonaných změn byl použit objekt `SyncOperationStatistics`, který zaznamenává změny vykonané v databázi.

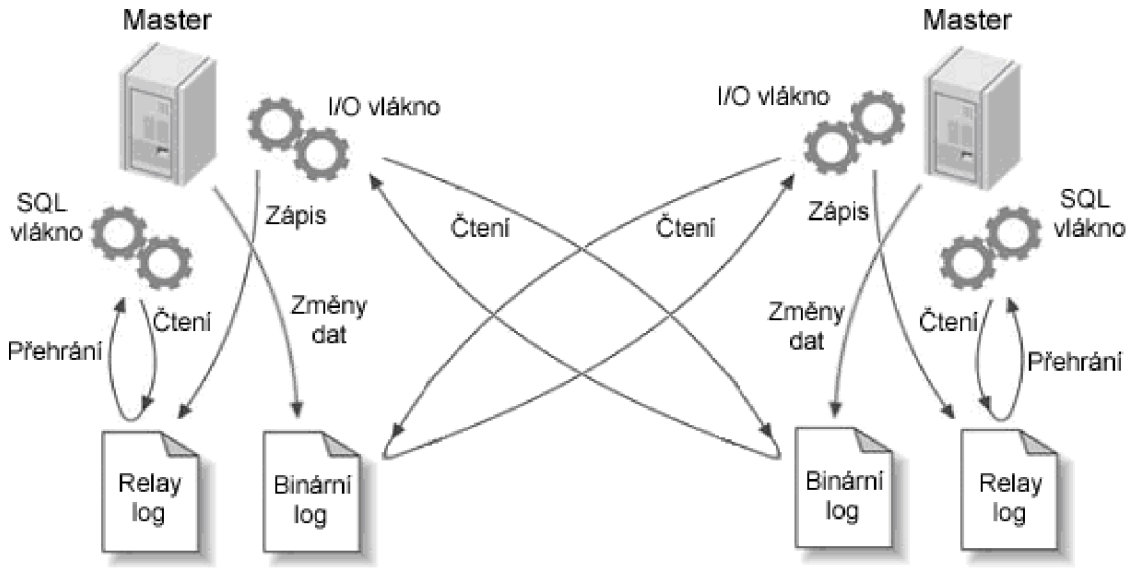
Testování toho frameworku proběhlo na jednom serveru s různými databázemi, ale stejnou strukturou. Výhoda tohoto frameworku je zajisté v tom, že umožňuje synchronizaci databází i v rámci jednoho serveru. Vyzkoušeny byly změny v databázi, které mohou způsobit příkazy INSERT, UPDATE a DELETE. V případě příkazů INSERT a DELETE byly po synchronizaci data v obou databázích správné a tedy Sync framework vykazoval dobré výsledky. Pokud byly provedeny změny pomocí příkazu UPDATE, tak synchronizační rutina data také správě upravila. Pokud byly změny provedeny v obou databázích na stejném místě, tak synchronizační proces vzal jako nejnovější hodnotu takovou, která byla upravena nejpozději. Tímto nevznikl žádný konflikt v databázi, avšak změny, které byly dříve prováděny, v druhé databázi byly zahozeny. Sync Framework při testech dosahoval velmi dobrých výsledků a pro obousměrnou synchronizaci databází lze zajisté použít.

Implementace synchronizace pomocí tohoto frameworku v TiSoftu by znamenala velký zásah do struktury databáze, použití několika triggerů, nemálo procedur a také by bylo zapotřebí velmi dobré zkušenosti s tímto frameworkem. Vzhledem k těmto okolnostem bylo domluveno s vedoucím této práce, že tento framework nebude z těchto důvodů na TiSoft implementován, protože je tato cesta velmi náročná, přestože nabízí schopné řešení.

6.3 Synchronizace pomocí replikace

Poslední použitou technikou je MySQL replikace. Narozdíl od předchozích je tento nástroj zabudován přímo v MySQL serveru a pro použití není zapotřebí žádné dodatečné instalace. Replikaci lze využít od verze 3.23, která byla vydána roku 2001. Od této doby se replikace dočkala nemalých změn a v dnešní době je velice cenný nástroj pro synchronizaci, který se využívá nejvíce z již zmiňovaných nástrojů. Replikaci lze také použít na několik serverů, které si mezi sebou vyměňují „čerstvá data“ a umožňuje několik synchronizačních topologií, ze kterých lze zajisté vybrat tu nejlepší pro konkrétní případ. Pro vyzkoušení replikace je zapotřebí minimálně dvou serverů, protože v rámci serveru replikace neumožňuje výměnu dat mezi databázemi, což ostatní nástroje umožňují.

Pro realizaci byla vybrána topologie *master-master*, kterou MySQL přímo nepodporuje, ale lze ji vytvořit pomocí dvou topologií *master-slave* a nastavit příslušné parametry, zabezpečit konflikty dat a upravit tak, aby vyhovovala přímo požadavkům. Replikace *master-master* se skládá ze dvou serverů. Oba jsou nakonfigurovány jako *master* i *replika* toho druhého, neboli dvojice *spolu-masterů*. Tuto konfiguraci lze názorně vidět na obrázku 6.3. Základem implementace jsou dva servery, které spolu komunikují pomocí TCP/IP. Servery se jmenují *Server 1* (IP adresa 192.168.123.28) a *Server 2* (IP adresa 192.168.123.27). Servery jsou připojeny k lokální síti a nemají veřejnou IP adresu, takže připojení k nim lze pouze z této sítě. Spojení mezi nimi může být přerušeno třeba z důvodu vypnutí serveru (počítače) nebo přerušení spojení z důvodu výpadku sítě. Verze obou serverů je 5.0.51, které byly nainstalovány při zavádění databázového systému. Předpokládá se, že obě databáze mají před synchronizací stejná data v obou databázích a také stejnou strukturu tabulek.



Obr. 6.3 Master-master replikace

V následujících podkapitolách budou vypsány nutné příkazy pro zprovoznění synchronizace *master-master*. Podle těchto kroků lze vytvořit základní koncept replikace, avšak pro použití v TiSoftu je třeba ještě dalších nastavení a ošetření konfliktů dat, které budou probrány v kapitole 7. Implementace replikace se skládá z těchto kroků:

1. Na každém se serverů se připraví replikační účet
2. Na obou serverech se nakonfiguruje zároveň master a replika
3. Na každém serveru se vydá pokyn, aby se replika připojila k masteru.

6.3.1 Vytvoření replikačních účtů

MySQL disponuje několika speciálními oprávněními, které umožňují běh replikačních procesů. I/O vlákna replik učiní TCP/IP připojení k masteru, což znamená, že na obou masterech je zapotřebí vytvořit nové replikační účty a přidělit jim patřičná oprávnění. Tyto oprávnění musí být vytvořena, aby se I/O vlákna mohla připojit a číst binární logy mastera. Uživatelské účty jsou nazvány *replication* a vytvoří se následovně:

Server 1:

```
mysql > GRANT REPLICATION SLAVE, REPLICATION CLIENT ON timplant.* TO
     -> replication@'192.168.123.27' IDENTIFIED BY '<heslo>';
```

Server 2:

```
mysql > GRANT REPLICATION SLAVE, REPLICATION CLIENT ON timplant.* TO
     -> replication@'192.168.123.28' IDENTIFIED BY '<heslo>';
```

Oprávnění *replication slave* a *replication klient* umožňuje, aby se server choval jako master a zároveň jako replika a tedy mohl přijímat data od druhého serveru a naopak. Dále je zapotřebí nakonfigurovat oba servery.

6.3.2 Konfigurace serverů

Dalším krokem je zapnutí několika nutných nastavení na obou serverech. Nejdříve je třeba zapnout zaznamenávání do binárního logu a specifikovat ID serveru. Nastavení se provádí v konfiguračním souboru *my.ini* MySQL serveru a přidává se do sekce `[mysqld]`. Pod Windows lze nalézt ve složce, kde je server nainstalován (např. C:\Program Files\MySQL\MySQL Server 5.0\).

Server 1 a Server 2:

```
log_bin=mysql-bin # záznam do binárního logu
log_bin_index = mysql-bin.index # název indexového souboru binárního
                                logu
```

Server 1:

```
server_id=28 # ID serveru
```

Server 2:

```
server_id =27 # ID serveru
```

Nastavení jedinečného ID pro každý server je nezbytné pro správný chod replikace, většinou se používá poslední oktet IP adresy. Pokud ID serveru není specifikováno, tak server automaticky vybere výchozí hodnotu jako 1. Hodnota 1 tak může snadno zmást a může být v konfliktu s jinými servery, které nemají explicitní ID. Další nastavení, které je nutné zapnout, je specifikace relay logu. Je také dobré nastavit nepovinné konfigurační parametry pro indexový soubor relay logu.

Server 1 a Server 2:

```
relay_log = mysql-relay-bin # specifikuje umístění a název relay logu
relay_log_index = mysql-relay-bin.index # specifikuje název
                                        indexového souboru relay logu
```

6.3.3 Nastartování replik

Posledním krokem je sdělit replikám, jak se mají připojit k masteru a že mají začít s přehráváním jeho binárních logů. Toto nastavení lze také explicitně zapsat do konfiguračního souboru *my.ini*, avšak dle doporučení[11] je lepší použít příkaz `CHANGE MASTER TO`, který kompletně nahrazuje odpovídající nastavení v *my.ini*. Podoba tohoto příkazu, kterým se na replikách nastartuje replikace, vypadá takto:

---

[11] SCHWARTZ, Baron, et al. *MySQL profesionálně – optimalizace pro vysoký výkon*. Brno : Zoner Press, 2009. Replikace, s. 375. ISBN 978-80-7413-035-9.

Server 1:

```
mysql > CHANGE MASTER TO MASTER_HOST=‘192.168.123.27‘,
    -> MASTER_USER=‘replication‘,
    -> MASTER_PASSWORD=‘heslo‘,
    -> MASTER_LOG_FILE=‘mysql-bin.000001‘,
    -> MASTER_LOG_POS=0
    -> MASTER_PORT=3306;
```

Server 2:

```
mysql > CHANGE MASTER TO MASTER_HOST=‘192.168.123.28‘,
    -> MASTER_USER=‘replication‘,
    -> MASTER_PASSWORD=‘heslo‘,
    -> MASTER_LOG_FILE=‘mysql-bin.000001‘,
    -> MASTER_LOG_POS=0
    -> MASTER_PORT=3306;
```

Parametr `MASTER_LOG_FILE` nemusí vždy odpovídat dané hodnotě. Pokud už někdy v minulosti bylo povoleno zaznamenávání do binárního logu, tak tato hodnota bude zajisté jiná. Pro zjištění správné hodnoty lze v MySQL využít příkaz `SHOW MASTER STATUS`, který zobrazí aktuální název souboru pro replikaci. Pokud byla replikace nyní nastavena, tak je potřeba před tímto příkazem restartovat server a tím spustit zaznamenávání do binárního logu. V kapitole 7.1 bude nastavení serverů, jejich spuštění a vyladění probráno do větších podrobností.

Parametr `MASTER_LOG_POS` je nastaven na 0, protože se jedná o začátek binárního logu. Jestliže se příkaz vykoná, lze pomocí příkazu `SHOW SLAVE STATUS` zkontrolovat, zdali jsou nastavení replik správná. Po zadání příkazu jsou nejzajímavější sloupce `Slave_IO_State`, `Slave_IO_Running` a `Slave_SQL_Running`. Tyto hodnoty by měly ukazovat, že procesy repliky neběží a proto replikaci na obou serverech je třeba nastartovat pomocí příkazu:

Server 1 a Server 2:

```
mysql > START SLAVE;
```

Tento příkaz by neměl vyprodukovat žádné chyby, ani žádný výstup. Pomocí příkazu `SHOW SLAVE STATUS` lze ověřit, že replikace běží na obou serverech. Sloupec `Slave_IO_State` by měl zobrazovat `Waiting for master to send event`, sloupce `Slave_IO_Running` a `Slave_SQL_Running` by měly vykazovat hodnotu `Yes` (zapnuto). Další sloupce jsou zajisté taky důležité pro správný běh synchronizace a proto je zapotřebí tyto hodnoty překontrolovat.

Pokud se příkaz `START SLAVE` úspěšně provedl, tak je synchronizace master-master připravena k použití. Synchronizaci lze ověřit změnou dat v jedné databázi a pozorování změn v databázi druhé. Pokud je vše nastaveno správně, tak by se data v jedné databázi měla automaticky replikovat na databázi druhou. Výhoda této techniky je zejména v tom, že synchronizace je prováděna vždy po každé změně v databázi. Na obou serverech jsou tedy vždy aktuální data. V případě výpadku spojení je synchronizace automaticky prováděna, až je spojení se servery obnoveno. Pokud je spojení přerušeno a mezitím jsou na obou databázích prováděny změny, může po opětovném spojení dojít ke

konfliktům dat a synchronizace může být pozastavena. Problémy, které mohou při použití replikace nastat budou probrány v kapitole 7.1 a 7.2.

6.4 Porovnání nástrojů vzhledem k realizaci

V předchozích kapitolách byly vyzkoušeny tři různé techniky pro synchronizaci databází MySQL. Každý z těchto nástrojů je svým způsobem specifický a nelze najít mezi těmito nástroji nějakou spojitost, avšak dosahují stejných výsledků. Každý nástroj je postaven na jiném základu a nasazení tohoto systému vyžaduje specifické podmínky. Při porovnání těchto technik se bude brát ohled zejména na snadnost nasazení, spouštění, změny nutné v databázi a funkčnost.

První z vyzkoušených nástrojů byl SQLyog. Tento nástroj má pěkné uživatelské prostředí a práce s ním je velmi intuitivní a příjemná. První nevýhoda je v tom, že tento program není zdarma a pro delší využití musí být zakoupen za 139$, což je poměrně vysoká částka na jeden rok. Jinak program nabízí nemalou řadu funkcí i mimo synchronizaci a proto by mohl najít i uplatnění v jiných partiích. Z hlediska změn, které musí být provedeny v databázi, je pouze přidání primárního klíče do jedné tabulky, takže nic drastického. Nastavení synchronizačního procesu není nijak náročné a lze ho pomocí několika kroků snadno provést. Toto nastavení lze také uložit a pomocí Windows plánovače úloh lze spouštění této synchronizace nastavit na konkrétní čas. Nastavení plánovače je určitě dobrá věc, ale není postačující. Data změněna v jedné databázi by měla být okamžitě distribuována na druhou databázi, což v tomto případě není možné, protože plánovač úloh neví, kdy se změny provedly. Synchronizace by musela být spouštěna v předem daný čas, což je značně neefektivní. Druhou možností spuštění by bylo přímo z TiSoftu, kde při každé změně v databázi by se spustila tato synchronizační rutina, což je to, co zrovna potřebujeme. Po funkční stránce je SQLyog nejslabší ze všech uvedených nástrojů. Po několika různých nastavení a testování se mi nepodařilo dosáhnout dobrých výsledků a změny dat v databázích nebyly vždy prováděny. Příkazy INSERT proběhly pokaždé v pořádku, avšak v případě příkazů UPDATE a DELETE data byla distribuována jen částečně nebo vůbec. Funkčnost synchronizace je základní požadavek pro implementaci v TiSoftu. Jelikož SQLyog vykazuje neuspokojivé výsledky, tak nemůže být pro realizaci využit.

Druhou vybranou technikou byl Microsoft Sync Framework. Tento nástroj uchovává změny v databázích v podobě metadat. Předem nastavené triggery zajišťují úpravu metadat v rámci databáze. Pro použití frameworku je zapotřebí změnit strukturu databáze. Tyto změny vyžadují přidání tří tabulek pro smazané položky a pro každou původní tabulku přidat dva sloupce pro metadata o vložených řádcích a dva sloupce o upravených řádcích. Změna v celé struktuře databáze je docela zásadní, avšak pro použití frameworku nutná. Implementace synchronizace je poměrně složitá záležitost a vyžadovalo by to napsání šesti triggerů, dvaceti-sedmi procedur pro tabulky a implementaci složitých spouštěčů pro synchronizaci. Použití Sync frameworku pro MySQL databáze je velice složité implementovat a nutné změny ve struktuře databáze ještě více přispívají k tomu, že tento nástroj nebude pro realizaci vhodný. Celý synchronizační proces by se prováděl spuštěním rutiny, která by byla napsána v jazyce C#. Jako v případě SQLyogu by bylo také možné spouštět tuto rutinu z TiSoftu po každé změně v databázi. Použití Sync Frameworku ovšem přináší více režie a v případě častějších změn by to znamenalo větší výpočetní nároky. Při testování byly změny v obou databázích prováděny korektně a data byla vždy správně distribuována. Jelikož je implementace

velice náročná a změny v databázi jsou poněkud rozsáhlé, tak pro realizaci není tento nástroj také vhodný, avšak nabízí schopné řešení.

Poslední implementovanou technikou je MySQL replikace. Na rozdíl od ostatních nástrojů je tato technika zabudována přímo v MySQL serveru, což je obrovskou výhodou. Oproti Sync frameworku ovšem neumožňuje synchronizaci databází v rámci serveru, což v rámci realizace není zapotřebí. Replikace nabízí mnoho nastavení a funkcí pro konkrétní realizaci a synchronizaci lze tedy implementovat do větších detailů. Jako jediný nástroj nevyžaduje žádný zásah do struktury databáze a lze tedy replikaci nasadit na jakoukoli strukturu. Základní realizace replikace se skládá z několika nutných nastavení serveru a databáze. Po specifikaci nastavení je replikace nastavena a lze ji tedy „navždy“ spustit jednoduchým příkazem. Synchronizace dat se okamžitě provádí při každé změně dat v databázi, což je velkou výhodou a tak na obou serverech jsou vždy nejnovější data. Synchronizace se tedy narozdíl od předchozích nástrojů nemusí spouštět ručně. K zastavení synchronizačního procesu může dojít pouze v případě zastavení jednoho ze serverů nebo konfliktu dat. V případě zastavení jednoho ze serverů je replikace znovu automaticky spuštěna po zjištění dostupnosti toho serveru. Další výhodou replikace je také snadné rozšíření na více serverů. Replikace z hlediska funkčnosti vykazovala velmi dobré výsledky a při testování se nevyskytly žádné chyby v distribuovaných datech. Rychlost replikace byla také velmi dobrá a šíření dat probíhalo se zanedbatelným zpožděním.

Z výsledného porovnání lze jasně vidět, že replikace od MySQL je nejlépe vyhovující synchronizační nástroj pro realizaci synchronizace na databázovém systému implantabilních zdravotnických prostředků. Replikace, oproti ostatním nástrojům, nabízí velice schopné řešení, které dosahuje velmi dobrých výsledků a všechny požadavky vypsané v kapitole 5.2 lze touto technologií splnit. Pro vlastní řešení synchronizace dat v TiSoftu bude tedy použita replikace.

7 Vlastní řešení synchronizace

Pro realizaci v TiSoftu byla vybrána technika replikace od MySQL. Základní koncept pro implementaci byl probrán v kapitole 6.3. Tento nástin synchronizace však bude vyžadovat několik dalších nastavení a úprav pro nasazení v praxi. Topologie celého procesu distribuce dat je *master-master*, ve které je zapotřebí se vyvarovat několika problémů, které mohou při používání nastat.

Databáze pro TiSoft obsahuje 5 tabulek (ZAKAZNIK, ZBOZI, KATALOG, POZNAMKA, UZIVATEL) z nichž tabulky POZNAMKA a UZIVATEL nebudou pro replikaci nastaveny. Tabulka POZNAMKA obsahuje poznámky o zákaznících, ve kterých je uloženo datum pro připomenutí. Po dohodě s firmou Timplant s.r.o. nebudou tyto tabulky synchronizovány, protože slouží pouze pro přihlášeného uživatele TiSoftu a synchronizace těchto dat by neměla význam. Tabulka UZIVATEL obsahuje pouze data pro přihlášení uživatelů do systému. Tabulka je vytvořena a naplněna při instalaci systému a data v ní nejsou modifikována. Zbývající tři tabulky budou plně podléhat replikaci.

Vzhledem k tomu, že oba servery nemusí mít vždy možnost mezi sebou komunikovat, tak může nastat různé rozložení, kde je jeden či druhý vypnutý a to musí být bráno v úvahu. Servery se pro přehled jmenují *Server 1* (hlavní počítač) a *Server 2* (vzdálený počítač). Jedná se o tyto tři situace:

1. Oba servery jsou zapnuté (mohou mezi sebou komunikovat)
2. Zapnutý je pouze *Server 1* (nemůže komunikovat se *Serverem 2*)
3. Zapnutý je pouze *Server 2* (nemůže komunikovat se *Serverem 1*)

Proces replikace byl implementován tak, že pokud jsou zapnuté oba servery a mohou mezi sebou komunikovat, tak *Server 1* se stane „hlavním“ serverem a program TiSoft se ze vzdáleného počítače automaticky připojí k tomuto serveru. Neboli TiSoft na hlavním počítači se připojuje na svou lokální databázi (*Server 1*) a TiSoft na vzdáleném počítači se také připojuje na *Server 1* a tedy oba se připojují na stejnou databázi. Replikace v tomto případě pořád běží a na obou serverech jsou aktuální data. V případě vypnutí *Serveru 1* se vzdálený počítač automaticky přepojí na svou lokální databázi (*Server 2*). Implementace tohoto postupu sebou přináší značné množství výhod a to zejména jako prevence proti případným konfliktům v databázi. Navíc se předpokládá, že při používání TiSoftu budou většinou oba servery zapnuty.

Pokud je spuštěný pouze *Server 1*, tak se situace pro hlavní počítač nijak nemění, TiSoft se připojuje k „hlavní“ databázi, úpravy dat jsou zaznamenávány do binárního logu, ale replikace je prováděna až v případě obnovení spojení se *Serverem 2*. Po obnovení spojení však může nastat konflikt dat, což je probráno v kapitole 7.2.

V případě spuštění pouze *Serveru 2* je TiSoft na vzdáleném počítači připojen ke své lokální databázi a můžou se provádět změny, které se také ukládají do binárního logu. TiSoft zkoumá každou minutu, zda je stále *Server 1* vypnutý. Pokud zjistí, že se *Server 1* spustil a mohou mezi sebou komunikovat, tak se automaticky přepojí na hlavní databázi, čímž předchází možným konfliktům dat. Po obnovení spojení je také automaticky provedena synchronizace dat. Přepojení TiSoftu na vzdálený počítač nemá na proces synchronizace žádný vliv, pouze předchází možným konfliktům.

Jedním z požadavků firmy bylo zabezpečení dat a to z hlediska možnosti útoku na synchronizační proces a získání firemních dat. Tento požadavek je sice v jisté míře samozřejmostí, ale byl zdůrazněn z toho důvodu, že prozrazení firemních dat by znamenalo velké problémy pro firmu. Replikace se připojuje na vzdálený server přes TCP/IP připojení. Přístup k danému serveru má pouze uživatel, který má toto oprávnění na serveru nastaveno. Pro replikaci se musel vytvořit replikační účet na každém serveru, který umožňuje běh replikačních procesů. Tento účet umožňuje pouze zjistit, kde je master nebo slave server a umožňuje replikaci číst binární logy z mastera, žádné jiné oprávnění tento účet neumožňuje. Příkazy jako SELECT, DROP, UPDATE atd. nejsou v tomto účtu povoleny. Tento účet na serveru umožňuje připojení pouze uživateli z dané IP adresy a připojení je také zabezpečeno heslem.

Další účet musel být vytvořen na *Serveru 1* a to pro připojení Tisoftu ze vzdálené místnosti. Tento účet umožňuje čtení, zápis, úpravu dat a provádění změn v tabulkách. Při vytváření tohoto účtu byla zadána pouze IP adresa, uživatelské jméno a heslo vzdáleného počítače, a proto přístup na daný server z jiné IP adresy není možný a server dané připojení odmítne.

Jinou situací může být odposlech na síti. Útočník by mohl odposlouchávat data, které přes TCP/IP spojení procházejí, ovšem data jsou při procházení sítí šifrována a tedy z ukradených dat by útočník nic neměl. Synchronizační události nejsou posílány v textové (otevřené) podobě, ale v binární a i po rozšifrování by útočník data nerozpoznal.

Z těchto důvodu by nemělo k porušení bezpečnosti dat dojít a proces replikace se zdá být bezpečný.

7.1 Úprava a spuštění replikace

Základní nastavení topologie master-master bylo popsáno v kapitole 6.3. Jednalo se o vytvoření replikačních účtů, konfiguraci serverů a nastartování replik. Pro plynulý chod replikace je zapotřebí nastavení ještě několika parametrů v konfiguračním souboru na prvním i druhém serveru.

Server 1 a Server 2:

```
sync_binlog=1
```

Nastavení tohoto parametru způsobí, že MySQL bude synchronizovat obsah binárního logu na disk pokaždé, když potvrdí příkaz, takže v případě havárie se nepřijde o události logu. Bude-li tato volba vypnutá, tak server bude mít méně práce, nicméně pokud zhavaruje, položky binárního logu mohou být poškozené nebo mohou chybět.

Další nastavení, které zdokonaluje replikační proces, je mazání starých souborů s binárními logy. Mazání starých logů se používá zejména proto, aby se harddisk nezaplnil těmito soubory, které už jsou zbytečné. Toto nastavení způsobí to, že se mažou záznamy, které jsou starší než daný počet dnů. Musí se také brát ohled na to, že replikační proces nemusel být nějakou dobu spuštěný a druhý server nemusí mít tyto data načtené, a proto je nutné tuto hodnotu předem pořádně zvážit. Nastavení vypadá následovně:

Server 1 a Server 2:

```
expire_logs_days=100
```

Pomocí příkazu `SHOW MASTER STATUS` lze zjistit aktuální pozici, konfiguraci a seznam všech binárních logů, které jsou umístěny na disku. Tato informace je užitečná, je-li zapotřebí určit parametry do příkazu `PURGE MASTER LOGS`. Tento příkaz umožňuje odstranit všechny protokoly, před zadaným binárním protokolem. V některých případech je nutné tyto binární logy odstranit ručně, aby zůstaly jen ty potřebné a uvolnilo se tak místo na disku.

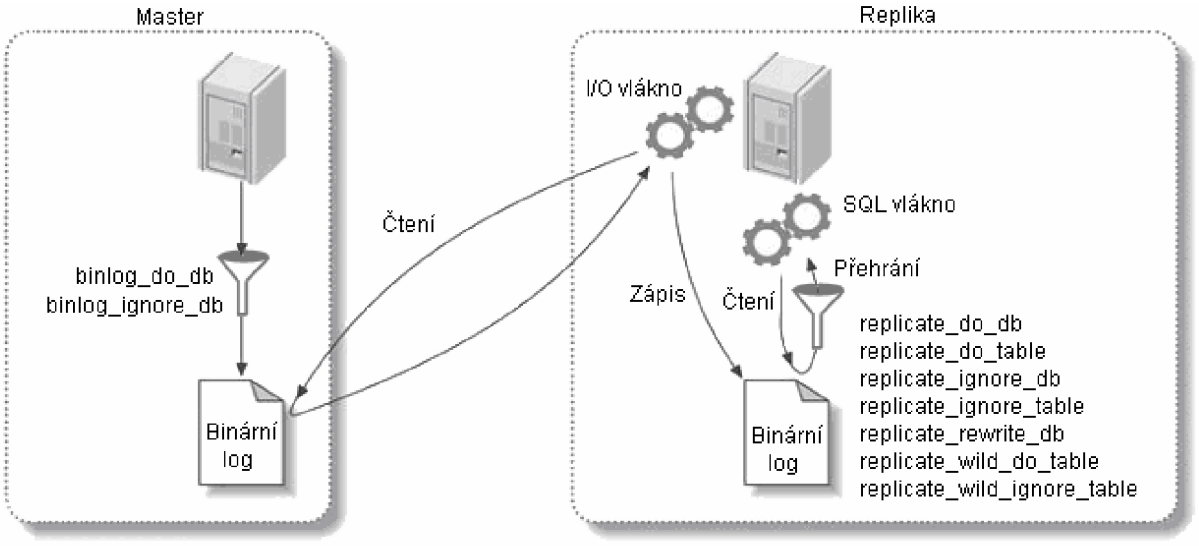
Další nastavení, které je dobré provést, je příkaz *log_slave_updates*, který umožňuje použít danou repliku jako mastera pro jiné repliky. Instruuje MySQL, aby zapsal události, které vykonává SQL vlákno, do svého vlastního binárního logu, které pak mohou získat a vykonat jeho vlastní repliky. Tato konfigurace znamená, že změny na jednom serveru se dají propagovat i na jiné servery, které nejsou k němu přímo propojeny. Toto nastavení se provede pouze pro *Server 1* a umožňuje přidání dalšího počítače (*Server 3*) do synchronizační topologie. Nový *Server 3* bude nakonfigurován podobně jako *Server 2* a bude spojen se *Serverem 1* jako master-master a I/O vlákno bude číst příkazy z binárního logu *Serveru 1*. Při reálné situaci nám toto nastavení umožňuje, aby byla data ze *Serveru 2* přenesena na *Server 1* a z toho pak na *Server 3*.

Server 1:
```
log_slave_updates=1
```

Jak již bylo řečeno, tak v případě, že jsou oba servery spuštěné a mohou mezi sebou komunikovat, tak se TiSoft automaticky připojuje na „hlavní“ server, tedy *Server 1*. Pro připojení je nutné na *Serveru 1* nastavit příslušné práva, aby se program mohl připojit a měnit data v databázi. Tyto práva se nastaví následujícím příkazem:

```
mysql > GRANT CREATE, DELETE, DROP, FILE, SELECT, SHOW DATABASES,
        UPDATE, INSERT ON timplant.* TO root@'192.168.123.27'
        IDENTIFIED BY 'heslo';
```

Další částí nastavení replikace je použití takzvaných filtrů. Filtrovací volby replikace umožňují replikovat pouze část dat serveru. Existují dva druhy filtrů. První filtruje události z binárního logu na masteru a druhý filtruje události přicházející z relay logu na replice. Oba tyto typy jsou ilustračně znázorněny na následujícím obrázku 7.1.



Obr. 7.1 Filtrovací volby replikace

Replikační filtry budou nastaveny tak, aby se na replikách zabránilo replikaci příkazů GRANT a REVOKE, které slouží pro přidělení práv. Tyto příkazy jsou důležité zejména do budoucna pro přidání nových uživatelů či omezení příslušných práv. Explicitně je také dobré specifikovat databázi, která se má replikovat, a databázi, jejichž příkazy se budou zaznamenávat do binárního logu. V případě topologie master-master je nutné nastavit všechny příkazy do konfiguračního souboru, protože každý se serverů se chová jako replika a zároveň jako master.

Server 1 a Server 2:

```
replicate_ignore_table=mysql.columns_priv
replicate_ignore_table=mysql.db
replicate_ignore_table=mysql.host
replicate_ignore_table=mysql.procs_prov
replicate_ignore_table=mysql.tables_priv
replicate_ignore_table=mysql.user
replicate_ignore_table=timplant.uzivatel
replicate_ignore_table=timplant.poznamka
replicate_do_db=timplant #databáze pro replikaci
binlog_do_db=timplant #databáze pro záznam do binárního logu
```

Dodatečným nastavením v předchozích odstavcích se dosáhlo lepších synchronizačních vlastností celé replikace. Některé nastavení nebyly nutné, avšak zabraňují problémům, které by mohly v budoucnosti nastat. Replikace umožňuje všelijaká nastavení a vybrání těch správných pro konkrétní řešení je v některých případech docela obtížné.

Po nastavení všech potřebných parametrů je zapotřebí restartovat oba MySQL servery. Spuštění celého synchronizačního procesu vyžaduje několik důležitých kroků, kde je nutné se přesvědčit, že všechna nastavení jsou v pořádku. Prvním krokem by se mělo ověřit, zda je na obou masterech vytvořen soubor binárního logu a zkontrolovat, zdali se dosáhne správného výstupu, který

je podobný následujícímu (MySQL přidává na konec názvu souboru specifické číslice, takže soubor se nemusí přesně shodovat a taktéž hodnota `Position` může být jiná):

```
mysql > SHOW MASTER STATUS;
+------------------+----------+--------------+------------------+
| File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
+------------------+----------+--------------+------------------+
| mysql-bin.000001 |       98 | timplant     |                  |
+------------------+----------+--------------+------------------+
```

Pokud již nebyla synchronizace spuštěna, tak pomocí příkazu `START SLAVE` musí být na obou serverech replikační proces spuštěn. Tento příkaz by neměl vyprodukovat žádné varování ani chyby. Pokud příkaz proběhl úspěšně na obou serverech, tak je zapotřebí zkontrolovat, zdali jsou nastavení repliky správná. Výpis příkazu pro *Server 1* by měl být podobný následujícímu:

```
mysql > SHOW SLAVE STATUS\G
*************************** 1. row ***************************
             Slave_IO_State: Waiting for master to send event
                Master_Host: 192.168.123.27
                Master_User: replication
                Master_Port: 3306
              Connect_Retry: 60
            Master_Log_File: mysql-bin.000001
        Read_Master_Log_Pos: 183
             Relay_Log_File: mysql-relay-bin.000001
              Relay_Log_Pos: 183
      Relay_Master_Log_File: mysql-bin.000001
           Slave_IO_Running: Yes
          Slave_SQL_Running: Yes
            Replicate_Do_DB: timplant
        Replicate_Ignore_DB:
         Replicate_Do_Table:
     Replicate_Ignore_Table: mysql.user,mysql.columns_priv,mysql.host,
                             mysql.tables_priv,mysql.db,mysql.procs_prov,
                             timplant.poznamka, timplant.uzivatel
    Replicate_Wild_Do_Table:
Replicate_Wild_Ignore_Table:
                Last_Errno: 0
                Last_Error:
                       ...
```

Tento výpis zobrazuje důležité údaje o replikačním procesu. Jedná se o přehrávání změn v datech, které se provedly na *Serveru 2* a přenáší se na *Server 1*. Nejdůležitější hodnotou je `Slave_IO_Running` a `Slave_SQL_Running`. Lze vidět, že I/O vlákno repliky běží a tedy události jsou zaznamenávány do relay logu. SQL vlákno repliky je také aktivní, což znamená, že je spuštěno přehrávání událostí z relay logu. Pokud je vše dobře nastaveno, tak obě vlákna musí být spuštěná.

I/O vlákno čeká na nějakou událost od mastera (*Serveru 2*), což znamená, že načetlo všechny *binární logy* mastera. Pokud se na *Serveru 2* provedou nějaké změny, tak by mělo jít vidět, jak se na replice inkrementují různá souborová a poziční nastavení a samozřejmě změny v databázi.

Důležitou hodnotou je také `Connect_Retry`, která je automaticky nastavena na 60. Pokud se nějakým způsobem nelze připojit na vzdálený server (*Server 2*), tak se každých 60 sekund pokouší server znovu připojit na vzdálený server a obnovit proces synchronizace. Tuto hodnotu lze změnit pomocí příkazu `CHANGE MASTER TO MASTER_CONNECT_RETRY=<hodnota>`, avšak 60 sekund je pro nastavení optimální. Nastavení této hodnoty je velice důležité, protože servery nejsou zapínány souběžně a proto se musí ve vhodném intervalu kontrolovat dostupnost vzdáleného serveru pro obnovení synchronizace.

Pokud jsou obě vlákna spuštěná, tak na obou serverech musí být tyto vlákna v seznamu procesů. Seznam procesů musí zahrnovat připojení I/O vlákna repliky, I/O vlákno a SQL vlákno (seznam procesů obsahuje i další procesy, které ovšem s replikací nesouvisí):

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 26
   User: replication
   Host: dluhos-ondrej:1614
     db: NULL
Command: Binlog Dump
   Time: 2279
  State: Has sent all binlog to slave; waiting for binlog to be
         updated
   Info: NULL
*************************** 2. row ***************************
     Id: 27
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 2092
  State: Waiting for master to send event
   Info: NULL
*************************** 3. row ***************************
     Id: 29
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 30
  State: Has sent all relay log; waiting for the slave I/O thread to
         update it
   Info: NULL
   ...
```

Výstup pochází ze serverů, které už nějakou dobu běží, a proto má sloupec `Time` I/O vlákna (*2. row*) velkou hodnotu. SQL vlákno bylo na replice 30 sekund nečinné, což znamená, že po dobu 30 sekund se nepřehrály žádné události. Tyto procesy vždy běží pod uživatelským účtem *system user*, ale hodnoty v ostatních sloupcích mohou být jiné. Pokud například SQL vlákno přehrává na replice nějakou událost, bude ve sloupci `Info` dotaz, který vykonává.

Vykonání všech dotazů, které byly vypsány v předchozích odstavcích, je velmi důležité pro zjištění chodu celého synchronizačního procesu. Těmito příkazy lze ověřit správné nastavení a také pochopit, jak celý synchronizační proces probíhá. Pokud je vše dobře nastavené a spuštěné, tak se můžou provádět změny v obou databázích a data budou obousměrně synchronizována.

7.2 Konflikty dat

Celý synchronizační proces byl navrhnut tak, aby se co nejvíce zabránilo možným konfliktům dat. Obecně lze říci, že ke konfliktu dat dojde v případě, že je nad databází proveden dotaz, který vrací chybu a nemůže být proveden. V případě této realizace jsou konflikty většinou způsobeny v přidávání nových záznamů do tabulky, kde je primární klíč (tabulky ZAKAZNIK a KATALOG). Konflikt může nastat tehdy, pokud je přerušeno spojení mezi servery a v obou databázích jsou přidávány nové položky. Nové záznamy pro tabulku ZAKAZNIK mají primární klíč, který je v TiSoftu nastaven jako nejvyšší hodnota ID v tabulce zvýšená o jedna (nepoužívá se funkce MySQL `auto_increment`). V obou databázích je tedy přidán řádek, který má primární klíč (např. 12). Po obnovení spojení obou serverů je synchronizační proces znovu spuštěn a nastane konflikt dat, protože v rámci výměny dat se musí přidat do každé databáze řádek z druhé databáze s daným primárním klíčem (12), avšak primární klíč (12) už v tabulce existuje. Vznikne tedy konflikt dat a replikace obou databází je pozastavena, protože chyba obvykle znamená nekonzistentní data a jsou zapotřebí ruční úpravy. V případě tabulky KATALOG je situace stejná, akorát primární klíč není typu *integer*, ale *varchar(4)*.

MySQL v případě nějakého konfliktu dat automaticky pozastavuje synchronizační proces a tedy SQL vlákno (Slave_SQL_running) je zastaveno do doby, než je tato chyba odstraněna. Chyba, která způsobila zastavení replikace je uložena do souboru chyb, který se nachází ve složce *data*, která je umístěna v instalační složce MySQL serveru. Soubor je většinou pojmenován jako název počítače s příponou *err*, např. *dluhos_pocitac_doma.err*. Tento soubor obsahuje různé poznámky, varování a chyby, které nastaly během používání replikace. Všechny údaje jsou chronologicky seřazeny a nejnovější záznamy lze nalézt na konci souboru. Chyba, která nastala v případě předchozího odstavce, může vypadat následovně:

```
100515 11:14:28 [ERROR] Slave: Error 'Duplicate entry '506' for key 1'
on query. Default database: 'timplant'. Query: 'INSERT INTO ZAKAZNIK
VALUES('1',506,'Ing.','','Novák','Aleš','','1','Sjízdná 23','Ostrava',
'72525','CZ','','','','','','','','','','','CZ','','','',null,false
,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE)', Error_code: 1062
```

Z tohoto chybového hlášení lze jasně poznat příčinu chyby. Jedná se o duplikaci primárního klíče, který musí být jedinečný. Další možností, jak určit, zda nastal nějaký konflikt při replikaci je zobrazení informací o synchronizačním procesu pomocí příkazu `SHOW SLAVE STATUS`. Pro detekci chyby jsou důležité řádky `Slave_SQL_running`, `Last_Errno` a `Last_Error`. Výpis těchto řádku může vypadat následovně:

```
mysql > SHOW SLAVE STATUS\G
******************************** 1. row ***************************
        ...
        Slave_SQL_running: No
        Last_Errno: 1062
        Last_Error: Error 'Duplicate entry '506' for key 1' on query
                    Default database: 'timplant'. Query: 'INSERT INTO
                    ZAKAZNIK VALUES('1',506,'Ing.','','Novák','Aleš',
                    '','1','Sjízdná 23','Ostrava','72525','CZ','','',
                    '','','','','','','','','CZ','','','',null,
                    false, FALSE,FALSE,FALSE,FALSE,FALSE,FALSE)'
        ...
```

Popis chyby je stejný jako v případě zobrazení z chybového souboru a po odstranění chyby by mělo SQL vlákno znovu běžet a nemělo by se zobrazit žádné chybové hlášení.

Odstranění chyby a znovuspuštění replikace lze dosáhnout tím, že se přeskočí následující příkaz v binárním protokolu serveru a bude pokračovat z nového místa. K tomu se používá příkaz `SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1`. Tento příkaz lze použít pouze, pokud je replikování zastaveno. Jakmile se vykoná, lze spustit server, který bude normálně pokračovat. Postup může vypadat následovně:

```
mysql > SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1;
Query OK, 0 rows affected (0.00 sec)
mysql > START SLAVE;
Query OK, 0 rows affected (0.00 sec)
mysql > SHOW SLAVE STATUS\G
...
```

Po vykonání těchto příkazů by se měla replikace znovu nastartovat a pokračovat v provádění dalších příkazů z relay logu. Posledním příkazem lze zkontrolovat, že je replikace pořádku. V případě, že po vykonání dalšího příkazu vznikne chyba, tak se musí celý proces odstranění chyby a spuštění opakovat.

MySQL umožňuje nastavit konfigurační soubor pomocí příkazu `slave-skip-errors=[chyb_kod1, chyb_kod2, ... | all]` tak, aby v případě nastavené chyby tuto chybu automaticky přeskočil a pokračoval v provádění dalších příkazů. Tuto volbu je ovšem nebezpečné používat, protože její nesprávné použití může vést ke ztrátě synchronizace s daty ostatních serverů. Toto nastavení se při realizaci nepoužívá a chyby jsou odstraňovány pomocí předchozího příkazu, kde jeho zavedení do automatické podoby bude probráno v následující kapitole.

Problém s daty může také nastat v případě, kdy jsou simultánně prováděny dva UPDATE příkazy, kde je upravován stejný řádek, avšak s jinou hodnotou (příklad uveden v kapitole 5.3.1). Synchronizace způsobí to, že příkaz provedený na prvním serveru se projeví na druhém a naopak, protože synchronizace přepíše data původní. Příkaz přepíše data na svém serveru, ale v zápětí jsou přepsána daty z repliky. Oba servery budou mít jiné data, ovšem se neobjeví žádné synchronizační chyby.

Tato situace je ošetřena tak, že v případě spuštění obou serverů a možnosti komunikace mezi sebou je TiSoft automaticky přepojen na „hlavní“ databázi a žádný konflikt dat dále nemůže být způsoben. Synchronizační proces v tomto případě dále nezávisle běží a po vypnutí „hlavního“ serveru je TiSoft automaticky přepojen na „lokální“ databázi, kde může provádět libovolné změny. Připojení na hlavní server s sebou přináší velké výhody a zabraňuje se tak některým možným konfliktům dat, které mohou nastat.

Konfliktům dat v databázi lze určitým způsobem předcházet, ale nelze je úplně vyloučit. Problém nastává v tom, že oba servery mají právo zapisovat a spojení mezi nimi může být přerušeno. Tuto situaci ovšem nelze jiným způsobem řešit. Existuje bezpečnější topologie *master-master* v režimu *aktivní-pasivní*, kde je jeden server vždy nastaven jen pro čtení. Tato topologie ovšem nesplňuje požadavky na synchronizační proces, protože jeden server je vždy nastaven pro zápis a druhý jenom pro čtení. Tyto role lze mezi sebou přehazovat, avšak přehození rolí není jednoduché a master-master v režimu *aktivní-pasivní* se většinou používá v případech, kdy je zapotřebí nahradit hlavní server. V případě přerušení spojení mezi servery by jeden server měl povoleno pouze čtení, což by nepřineslo žádné zlepšení.

Vzhledem k tomu že se databáze využívají především pro čtení, a zápis do obou databází, kde jsou primární klíče, se provádí oproti čtení velmi málo, tak je zápis oběma serverům povolen. Při vkládání nových záznamů se zejména jedná o tabulku ZBOZI, kde jsou vkládány nově expedované výrobky. Vzhledem k tomu, že tato tabulka neobsahuje primární klíč, tak v případě výpadku spojení obou serverů a přidávání nových řádků by neměly konflikty dat v databázích nastat. V TiSoftu je programově nastavena automatická detekce případných chyb a jejich odstranění.

7.3 Monitorování replikace

Proces replikace nepochybně zvyšuje složitost monitorování MySQL serveru. Replikace ve skutečnosti probíhá jak na masteru, tak i na replice, avšak většina práce se dělá na replice. Zde se také mohou vyskytnout nejběžnější problémy, zda replikují obě repliky, zda nedošlo na jedné z replik k nějaké chybě, atd. V následujících podkapitolách budou vypsány důležité informace a úkony, které jsou nutné pro monitorování celého synchronizačního procesu.

7.3.1 Rychlost

MySQL replikace je obvykle velmi rychlá a běží tak rychle, jak rychle dokáže MySQL kopírovat události z mastera a přehrávat je. Zpoždění lze zjistit ve sloupci `Seconds_behind_master` z výstupu příkazu `SHOW SLAVE STATUS`. Replika spočítá tuto hodnotu podle [4] tak, že porovná aktuální časovou známku serveru s časovou známkou zaznamenanou v události binárního logu, takže

replika může oznámit své zpoždění, pouze pokud zpracovává nějaký dotaz. Pro úplné změření rychlosti replikace by to vyžadovalo změřit každý krok celého postupu a zjistit, které kroky zabírají v aplikaci většinu času. Jelikož je mezi servery docela rychlé síťové spojení, tak je obvykle jen velmi malé zpoždění mezi zaznamenáváním událostí do binárního logu a jejich zkopírováním do relay logu. V celé synchronizační topologii se navíc předpokládá převaha čtecích dotazů, a tedy zátěž celého procesu by měla být minimální.

7.3.2 Konzistence dat

Dalším důležitým krokem je zjištění, zda jsou repliky konzistentní s masterem neboli zda oba servery mají stejná data. Při replikaci mohou nastat chyby, které způsobí, že data repliky se prostě ztratí, takže už nebudou synchronizována s masterem. Problémy mohou také nastat tehdy, když evidentně k žádným chybám nedochází, ale i přesto se mohou repliky dostat do nesynchronizovaného stavu, například kvůli některým funkcionalitám MySQL, které nelze replikovat, kvůli poruchám v síti, kvůli chybám v MySQL, všelijakým haváriím, násilným ukončením atd. Čas od času je tedy zapotřebí zkontrolovat, zda uchovávají oba servery stejná data v databázi. MySQL nemá žádnou zabudovanou funkci, kterou by se dalo určit, zdali jeden server má stejná data jako server druhý. Poskytuje pouze jisté stavební bloky pro kontrolní součty tabulek a dat, jako je `CHECKSUM TABLE`. Tento příkaz umožňuje zobrazit kontrolní součet tabulky. `CHECKSUM` závisí na datech v tabulce, pokud se změní, tak se také změní `CHECKSUM`. Použití příkazu pro *Server 1* vypadá následovně:

```
mysql > CHECKSUM TABLE ZAKAZNIK, ZBOZI, KATALOG;
+----------------------+---------------+
| Table                | Checksum      |
+----------------------+---------------+
| timplant.zakaznik    |   909208172   |
| timplant.zbozi       |  3302340078   |
| timplant.katalog     |  1322256458   |
+----------------------+---------------+
```

Pro každou tabulku je vygenerována hodnota, kde její velikost není podstatná, ale použije se pro kontrolu se *Serverem 2*. V případě, že oba servery mají v tabulkách ZAKAZNIK, ZBOZI a KATALOG stejné data, hodnoty `CHECKSUM` by měly být stejné. Pro ověření není nic jednoduššího, než na druhém serveru provést ten samý příkaz a hodnoty porovnat. Jedná se o velice jednoduchou metodu na ověření, zdali jsou obě databáze konzistentní.

Toto ověření bylo také implementováno do TiSoftu, protože je nezbytné, aby obě databáze obsahovaly stejná data.

7.3.3 Kontrola synchronizačního procesu a detekce chyb

Nedílnou součástí celého procesu je monitorování běhu synchronizace, případná detekce chyb a jejich odstranění. Po spuštění TiSoftu je provedena kontrola, zda oba servery spolu komunikují. Program se pokusí o navázání spojení na server, jehož IP adresa je zadána v konfiguračním souboru programu pod názvem `ip_main_database`. V případě, že servery spolu nekomunikují, tak se vypíše

informační zpráva, že synchronizace není spuštěná a bude spuštěna, až se mezi oběma servery naváže spojení. Do té doby se mohou data v databázi normálně upravovat. Tato situace nastane při vypnutí jednoho z počítačů, výpadku serveru, výpadku sítě atd. a je zapotřebí tuto situaci uživateli oznámit.

Pokud servery mezi sebou komunikují, tak se provede kontrola celého synchronizačního procesu. Připomenu, že kontrola synchronizace je zcela v režii TiSoftu. Nad aktuální databází je proveden příkaz `SHOW SLAVE STAUS`, který vrací informace o aktuálním běhu celé synchronizace. Důležité jsou zejména údaje o vláknech `Slave_IO_Running` a `Slave_SQL_Running`. Pokud jsou obě vlákna spuštěná, tak je synchronizace správně nastavená a všechno běží v naprostém pořádku.

Nejdříve je kontrolováno I/O vlákno replikace (`Slave_IO_Running`). Pokud je toto vlákno pozastaveno (hodnota je No), tak je replikace z nějakého důvodu vypnuta. Pozastavení tohoto vlákna způsobuje, že nejsou přenášeny události z jednoho serveru na druhý. Většinou se jedná o konflikt dat, špatné nastavení, chyby způsobené výpadkem serveru, chyby způsobené při havárii serveru a jeho opětovné spuštění apod. Nejdříve je zapotřebí zjistit, zda nejde synchronizaci spustit a obnovit tak spuštění tohoto vlákna. Program automaticky provede MySQL příkaz `START SLAVE`, který spouští replikaci. Provede se znova kontrola, zda toto vlákno běží či nikoliv. Pokud se vlákno spustilo (hodnota je Yes), tak je vypsána informační zpráva o tom, že vlákno nebylo spuštěné, ale podařilo se spustit a synchronizační proces je obnoven. V opačném případě je vypsána varovná správa, že je replikace nějakým způsobem poškozena, nepodařilo se chybu odstranit a příslušná chyba byla zaslána administrátorovi jako mail. Uživatelé TiSoftu nejsou žádní programátoři a proto je zapotřebí tuto chybu odchytit a poslat mi (autorovi této práce) na mail, jakožto administrátorovi. Jelikož jsem tento systém sám programoval, tak odstranění chyby je zcela na mě. Pokud je replikace správně nastavená, tak by chyby při replikaci neměly vznikat, ale nastat můžou a proto je třeba nad tímto synchronizačním procesem mít patřičný dohled.

Jako další je kontrolováno SQL vlákno (`Slave_SQL_Running`), které přehrává všechny události vyskytující se v relay logu. Činnost toho vlákna je také velmi důležitá a musí být spuštěno. V programu je automaticky kontrolován běh tohoto vlákna a v případě zastavení jsou prováděny příslušné kroky, vysvětlené dále. Pokud servery mezi sebou komunikují, tak nečinnost tohoto vlákna je většinou způsobena z důvodu chyby při přehrávání událostí. Pravděpodobně došlo ke konfliktu dat a toto vlákno bylo pozastaveno do doby, než se daná chyba odstraní. V programu je zkoumáno, zda došlo ke konfliktu a v případě detekce konfliktu se program automaticky pokusí o odstranění chyby. Odstranění této chyby se provádí přeskočením události z relay logu, která toto pozastavení způsobila. V prvním kroku se zastaví replikační proces příkazem `SLOP SLAVE`, poté se přeskočí jedna událost pomocí `SET GLOBAL SQL_SLAVE_SKIP_COUNTER=1` a spustí se znovu replikační proces příkazem `START SLAVE`. Následně se zkoumá, zda se SQL vlákno spustilo a v případě, že následuje další chyba, tak jsou chyby v konečné smyčce přeskočeny. Každá ze zaznamenaných chyb je automaticky poslána administrátorovi pro bližší prozkoumání. Pokud se podaří chyby odstranit, tak je vypsána informační zpráva o tom, že nastaly chyby, ale byly odstraněny. Nicméně příkazy, které měly být provedeny, tak byly zahozeny a je třeba zásahu administrátora a chyby napravit.

7.3.4 Zásahy administrátora

Návrh a implementace celého synchronizačního procesu probíhala tak, aby zásah administrátora byl jen minimální a to většinou v případech, kdy došlo k zastavení synchronizace a chyby se nepodařilo

automaticky odstranit. Detekce chyb probíhá automaticky, při každé zaznamenané chybě se nejdříve software snaží tyto chyby vyřešit, ale v případě neúspěchu je potřeba zásahu administrátora. V dnešní době je nutné, aby každý běžící server měl svého administrátora, a ten mohl monitorovat chování příslušných procesů. U implementovaného synchronizačního procesu tomu není jinak, a jako administrátor jsem já (autor této práce, dále jen administrátor).

Jak již bylo zmíněno, tak v případě, že se detekuje chyba a nelze automaticky odstranit, tak se odešle mail s popisem chyby administrátorovi. Data v databázi mohou být do odstranění chyby upravována, avšak nebudou distribuována na další server. Provedené změny se totiž automaticky zaznamenávají do binárního logu a zapnutí, vypnutí, pozastavení či spadnutí celého synchronizačního procesu nemá na záznam událostí vliv. Všechny změny jsou ukládány a v případě obnovení jsou tyto změny přeneseny na druhý server.

Postup administrátora by znamenal prozkoumání chyby, která byla zaslána mailem a následné zprovoznění synchronizace. Zprovoznění by znamenalo ruční kontrolu procesu např. pomocí příkazů `SHOW SLAVE STATUS`, `SHOW PROCESSLIST`, `SHOW MASTER STATUS` nebo nahlédnutí do souboru, ve kterém se zaznamenávají některé poznámky, varování a chyby MySQL serveru. Soubor se obvykle nachází ve složce `data`, tam kde je nainstalovaný MySQL a obvykle má název počítače s příponou *err*, např. *muj_pocitac.err*. Tento soubor zaznamenává datum a danou informaci o nějaké události. Z daného výpisu lze většinou přesně specifikovat chybu a následně ji odstranit.

V kapitole 7.2 bylo zmíněno, že můžou nastat konflikty dat. Jedním z případů bylo přidání do obou databází nového zákazníka, když je přerušeno spojení mezi servery. Po obnovení spojení nastane konflikt dat, protože se přehrávají příkazy s daným primárním klíčem, ovšem tento primární klíč už v databázi existuje. Synchronizační proces se pozastaví a po detekci je zaslána chyba administrátorovi. Odstranění chyby v tomto případě znamená smazání přidaného záznamu z jedné či druhé databáze, čímž se uvolní potřebný primární klíč a záznam o novém zákazníkovi musí být vložen ručně. Záznam, který byl smazán, tak musí být také vložen ručně na obě databáze. Tímto způsobem se odstraní daný konflikt dat a synchronizace může být spuštěna na obou serverech příkazem `START SLAVE`.

7.4 Úprava TiSoftu

Použití replikace mezi oběma servery sebou přináší některé nutné změny, které se musely v TiSoftu provést. Tento databázový software byl vyvinut v roce 2008 pouze pro práci s databází MySQL na localhostu a žádné přípravy či návrhy pro síťovou verzi nebyly implementovány.

Jak již bylo řečeno, tak v případě, že mohou oba servery mezi sebou komunikovat, tak je vždy jako „hlavní server“ brán v úvahu *Server 1* a TiSoft se z vedlejší místnosti připojuje na tento server přes TCP/IP spojení. Tato možnost sebou přináší značné výhody a zabraňuje se tak možným konfliktům dat, které by mohly nastat.

Uživatelský software byl upraven tak, aby zobrazoval s jakou databází uživatel aktuálně pracuje. V případě „vedlejšího“ počítače je pod hlavním menu zobrazen název *lokální* či *hlavní* databáze. Vždy je tedy jasně vidět se kterou databází uživatel pracuje. TiSoft v kanceláři majitele firmy se připojuje pouze na svůj server a název databáze je tedy vždy *hlavní*.

Jak již bylo zmíněno v kapitole 7.3.3, tak do TiSoftu byla zabudována kontrola celého synchronizačního procesu, detekce chyb a případné odstranění chyby nebo zaslání mailu administrátorovi. Kontrola celého procesu se provádí automaticky po spuštění tohoto software a o případných chybách je uživatel informován.

Malou úpravou také prošlo zálohování dat databáze. Zálohování se provádí pomocí `SELECT INTO OUTFILE`, což je trochu zastaralý způsob zálohování, ale má své opodstatnění. Vybraná data se exportují do textových souborů, které jsou poté šifrovány. Soubory lze uložit pouze na databázový server a tudíž zálohu tímto způsobem nelze uložit na nějakém klientském, vzdáleném počítači. Tady vzniká malý problém, protože uživatel, který se připojuje na vzdálený server, nemá oprávnění pro přístup na daný počítač a tedy zálohu uložit. Po domluvě s firmou Timplant s.r.o. byla „vzdálenému" uživateli záloha zcela znemožněna a zálohu může tedy provádět pouze uživatel, který se připojuje k serveru jako localhost, což je vlastně majitel firmy, kde jeho počítač představuje „hlavní" server.

V předchozí podkapitole bylo zmíněno monitorování replikace. Při synchronizaci dat je zapotřebí zkoumat, zda jsou repliky konzistentní s masterem neboli zda oba servery mají stejná data. V TiSoftu byla přidána kontrola konzistence obou databází pomocí MySQL funkce CHECKSUM. Celý proces funguje tak, že funkce je nejdříve aplikována na tabulky lokální databáze, poté na tabulky vzdálené databáze a výsledné hodnoty jsou mezi sebou porovnány. Pokud mají tabulky stejná data, tak funkce CHECKSUM musí vracet stejné výsledky. Zobrazení výsledku porovnání je provedeno přes dialogové okno. Tuto kontrolu lze spustit v hlavním menu pomocí tlačítka *Služby* a následně v panelu volbou *Kontrola synchronizace*.

8 Závěr

Náplní této diplomové práce bylo seznámit se s různými prostředky pro synchronizaci databází MySQL, zvolit vhodné nástroje pro nasazení v praxi, tyto nástroje implementovat a zhodnotit jejich funkčnost, flexibilitu, snadnost nasazení, výhody a nevýhody. Na základě těchto poznatků implementovat nejlépe vyhovující nástroj na distribuovaném databázovém systému evidence implantabilních zdravotnických prostředků, který byl vytvořen a nasazen ve firmě Timplant s.r.o. v rámci bakalářské práce.

Zadání této práce bylo vytvořeno z mě iniciativy, protože mě problematika synchronizací databází velmi zajímala a chtěl jsem získat jak teoretické, tak i praktické zkušenosti při rozboru požadavků, návrhu, realizaci a testování synchronizace dat mezi databázemi. V této souvislosti se mi nabídla možnost systém nasadit v praxi.

Po nastudování problematiky synchronizace byly vybrány nástroje SQLyog, Microsoft Sync Framework a MySQL replikace, které jsou vhodné pro implementaci na databázovém systému. Každý z těchto nástrojů je svým způsobem specifický, synchronizaci dat provádí na jiném principu, ale dosahuje stejných výsledků jako ostatní. Všechny tyto techniky byly důkladně nastudovány a vyzkoušeny a byla vybrána MySQL replikace, která nejlépe vyhovuje požadavkům synchronizace na databázovém systému evidence implantabilních zdravotnických prostředků. Jako synchronizační topologie byla vybrána master-master v režimu aktivní-aktivní, která s sebou přináší nemalé komplikace při realizaci, avšak nejlépe vystihuje požadavky firmy. Při implementaci se kladl velký důraz na distribuci dat a zachování konzistence obou databází. Výsledek realizace splnil ve všech krocích požadavky firmy a zamezilo se některým možným konfliktům dat, které mohou při používání nastat. Nutné byly také změny uživatelského software a to zejména pro kontrolu správného chodu celého synchronizačního procesu. Přidána byla také kontrola, zda mají oba mastery stejné data v tabulkách databáze.

Celý synchronizační proces byl implementován a otestován ve firmě. Majitel firmy a zaměstnanci jsou s realizací plně spokojeni, protože jim zcela odpadá problém vlastní distribuce dat na ostatní počítače. Vše se vykonává zcela automaticky. Před uvedením systému do řádného provozu byla diskutována možnost poškození nebo ztráty dat, která by měla pro činnost firmy fatální důsledek Přesto byl systém plně nasazen v polovině března 2010 a je dosud intenzivně sledován. Prozatím nebyly zjištěny žádné nesrovnalosti či problémy s distribucí dat.

Hlavním přínosem pro mě byla teoretická a praktická stránka celé práce. Seznámil jsem se s různými prostředky pro synchronizaci a to nejen pro databáze. Některé techniky a postupy lze využít obecně pro synchronizaci libovolných dat v rámci počítačů v síti, jednoho počítače například pro zálohu nebo počítače a jiného zařízení.

Celkově byla tvorba této diplomové práce velice zajímavým seznámením s moderními nástroji pro synchronizaci databází, které se v dnešní době velice používají i ve velkých firmách a společnostech. Dalším přínosem pro mě také bylo reálné nasazení výsledku této práce do praxe a jeho využívání ve firmě, která ocenila implementaci tohoto systému a přišla s dalším požadavkem na řešení síťové problematiky v souvislosti s rozšířením a zdokonalováním jejich managementu jakosti. Zajímavé bude zajisté sledovat, jak bude nasazený systém fungovat v delším časovém horizontu.

Literatura

[1] Hauzar, D.: *Tvorba databází v MySQL - I* [online]. 20. 3. 2003 [cit. 2010-01-05]. Dostupný z WWW: <http://www.abclinuxu.cz/clanky/navody/tvorba-databazi-v-mysql-i>.

[2] Zajíc, P.: MySQL (1) - pestrý svět databází [online]. 2005, 1.3.2005 15:00 [cit. 2009-12-15]. Dostupný z WWW: <http://www.linuxsoft.cz/article.php?id_article=731>.

[3] Kofler, M.: *Mistrovství v MySQL 5 : Kompletní průvodce webového vývojáře*. [s.l.] : [s.n.], 2007. 805 s. ISBN 978-80-251-1502-2.

[4] Schwartz, B.; Zaitsev, P.; Tkachenko, V.; aj.: *MySQL profesionálně: Optimalizace pro vysoký výkon*. [s.l.] : [s.n.], 2009. 712 s. ISBN 978-80-7413-035-9.

[5] Welling, L.; Thomson, L.: *MySQL : Průvodce základy databázového systému*. Brno: CP Books, 2005. 255 s. ISBN 80-251-0671-3.

[6] Shneider, R. D.: *MySQL : Oficiální průvodce tvorbou, správou a laděním databází*. [s.l.] : [s.n.], 2006. 371 s. ISBN 80-247-1516-3.

[7] *Microsoft Sync Framework* [online]. c2010 [cit. 2010-01-05]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/sync/bb887625.aspx>.

[8] Činčura, J.: *Microsoft Sync Framework : Synchronizace databází (1)* [online]. 2007, 21. prosince 2007 [cit. 2009-12-22]. Dostupný z WWW: <http://www.vyvojar.cz/Articles/573-microsoft-sync-framework-synchronizace-databazi-1.aspx>.

[9] *Selecting* the Appropriate Sync Framework Components [online]. c2010 [cit. 2010-01-05]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/library/dd937565(SQL.105).aspx>.

[10] *Introduction to Microsoft Sync Framework* [online]. 2009 [cit. 2009-12-22]. Dostupný z WWW: <http://msdn.microsoft.com/en-us/sync/bb821992.aspx>.

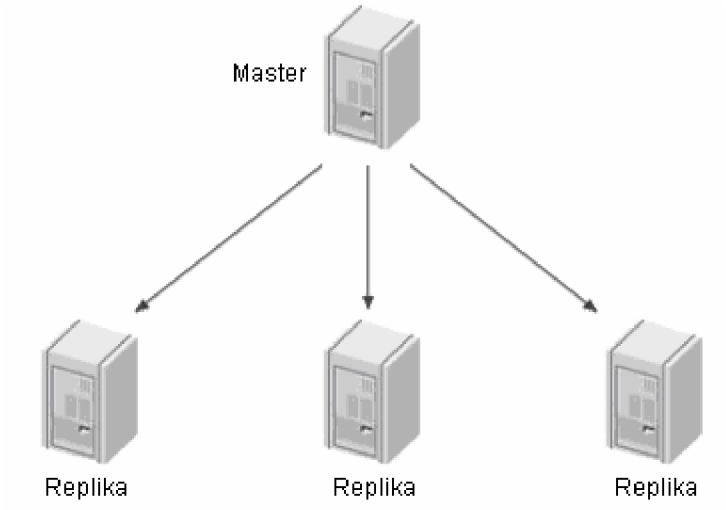
[11] *Microsoft developer network* [online]. c2010 [cit. 2010-05-11]. Architecture and Classes for Peer-to-Peer Synchronization. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/bb902829.aspx>.

Seznam příloh

Dodatek A

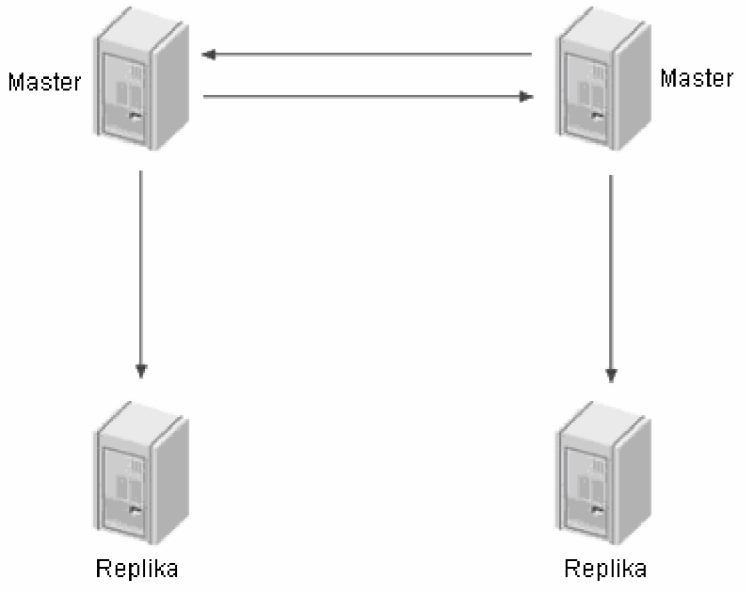
MySQL replikační topologie

V této příloze jsou ukázány různé replikační topologie, které replikace od MySQL podporuje a na základě těchto topologií mohou být sestaveny složitější konfigurace pro několik serverů. Využití těchto topologií je podrobně probráno v literatuře [3], [4] a [6] odkud jsem také čerpal.

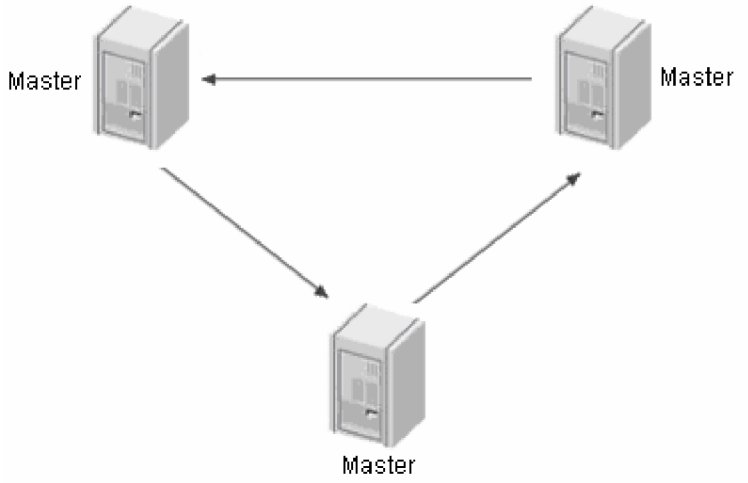


Obr. A.1 Master s několika replikami

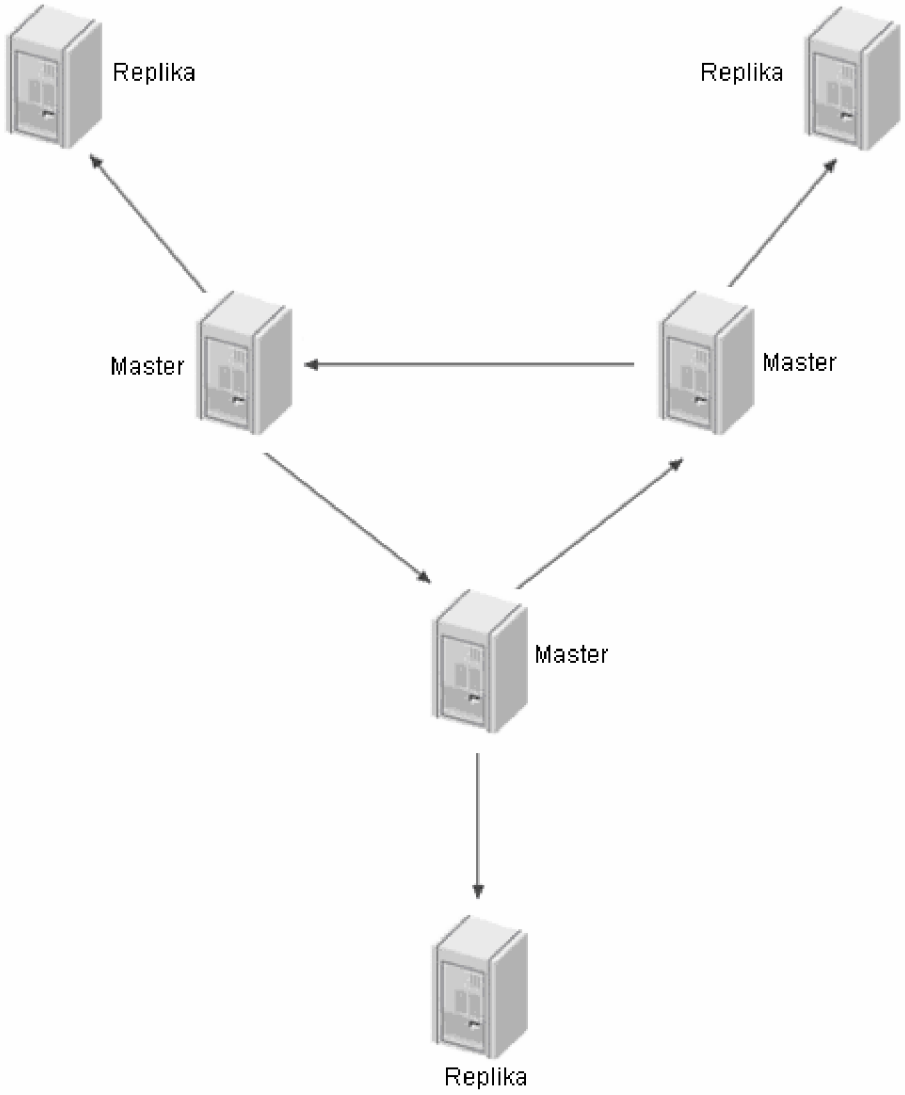
Obr. A.2 Replikace master-master v režimu aktivní-pasivní



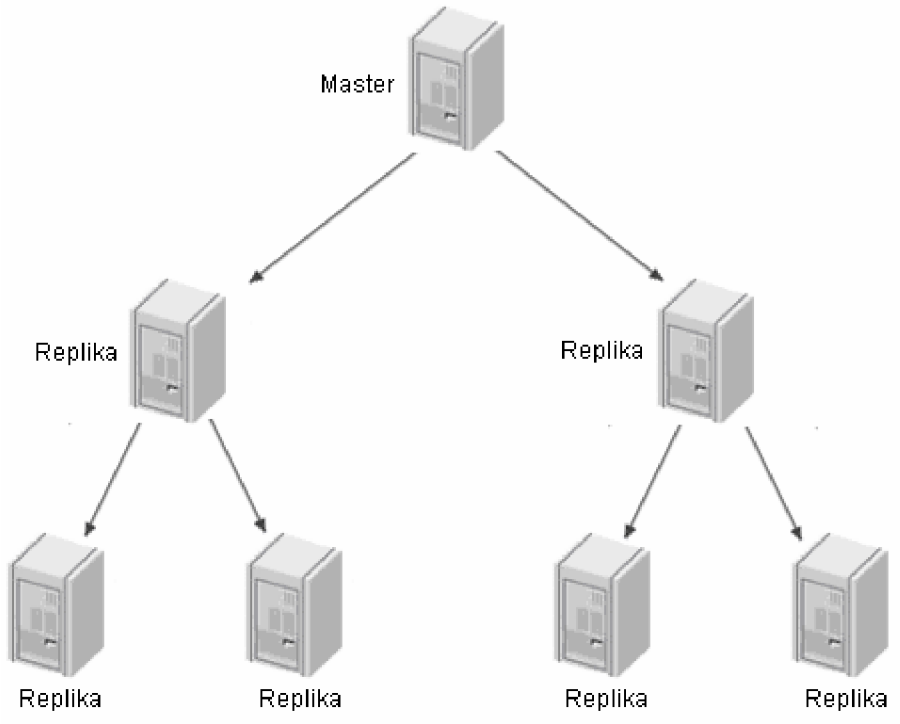
Obr. A.3 Replikace master-master s replikami



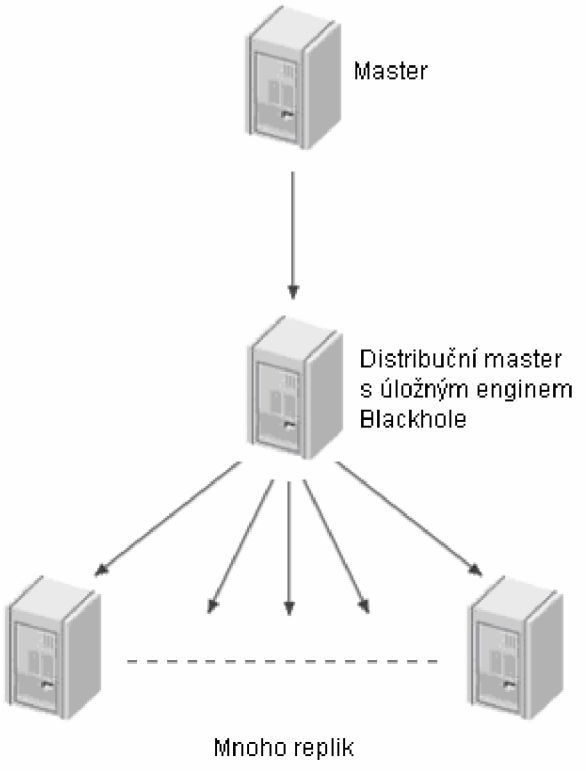
Obr. A.4 Prstencová replikační topologie



Obr. A.5 Prstencová replikační topologie s replikami ve všech uzlech



Obr. A.6 Pyramidová topologie replikace



Obr. A.7 Master, distribuční master a mnoho replik

Dodatek B

Obsah přiloženého CD

- adresář `bin` - spustitelný soubor software *TiSoft*
- adresář `docs` – návod na zprovoznění serverů, replikace a spuštění TiSoftu
- adresář `install` – instalační soubor MySQL serveru a SQL skript pro vytvoření tabulek
- adresář `src` – zdrojové kódy
- adresář `thesis` – text diplomové práce ve formátu PDF