Platforma průmyslové spolupráce

CZ.1.07/2.4.00/17.0041

Název

JBoss Teiid konektor pro NoSQL databázi Apache Cassandra

Popis a využití

- demonstrace převodu z NoSQL do SQL světa
- výuka: pokročilá Java

Jazyk textu

- český

Autor (autoři)

- Radek Koubský

Oficiální stránka projektu:

- http://lasaris.fi.muni.cz/pps

Dostupnost výukových materiálů a nástrojů online:

- http://lasaris.fi.muni.cz/pps/study-materials-and-tools

Obsah

Kapitola 1

Úvod

S vývojem počítačových systémů (zejména informačních systémů) se současně vyvíjel i způsob ukládání dat, které tyto systémy spravují. Důsledkem takového vývoje začalo vznikat velké množství odlišných datových zdrojů (CSV soubory, relační databáze, distribuované úložné systémy apod.) v závislosti na požadavcích jednotlivých systémů. Existence různorodých datových zdrojů přináší komplikaci v případě, že je potřeba spojit více informačních systémů a sjednotit jejich data. Tento požadavek typicky vzniká při seskupování různých orgranizací, společností a jiných skupin do jednoho celku.

Řešením problematiky sjednocení heterogenních datových zdrojů je federovaný databázový systém neboli datová federace (jedna definice federovaného datového systému je uvedena v [14], viz. 2.5.1) spolu s datovou integrací (*Enterprise Information Integration* [8]). Datová integrace je v současné době efektivní způsob pro získávání, vyhledávání a dolování dat v rozsáhlém měřítku, poskytuje schopnost hlubší analýzy dat. K rozvoji datové integrace přispěl přechod vývoje podnikového softwaru na principy SOA (*Service Oriented Architecture*). SOA svým komponentovým přístupem přirozeně vytváří požadavky na integraci dat mezi jednotlivými komponentami systému.

Během vývoje datové federace a datové integrace vznikají technologická řešení pro programovací platformy (Java EE, .NET) od různých vydavatelů softwaru (IBM, Microsoft, Red Hat). Tato práce se zaměřuje na produkt JBoss Teiid, který je součástí skupiny projektů *Red Hat JBoss Data Virtualization*.

JBoss Teiid je původní projekt společnosti MetaMatrix, která jako první přišla s produktem poskytující integraci dat. V roce 2007 koupila MetaMatrix společnost Red Hat, která pokračuje ve vývoji systému pro integraci dat.

1.1 Cíle práce

Cílem této práce je rozšíření nástroje JBoss Teiid o další datový zdroj, na kterém mohou být spouštěny čtecí dotazy přes relační abstrakci v rámci datové federace. Datový zdroj představuje NoSQL databázi Apache Cassandra. Poskytnutí přístupu právě k databázi Cassandra skrze Teiid vychází z požadavku *TEIID-1311 Consider developing a connector/translator to Cassandra/Thrift*, který lze najít na oficiálním webu JBoss.

Hlavní cíl práce lze rozdělit na tři části. První část je nastudování architektury a zdrojového kódu nástroje Teiid, aby bylo možné navrhnout a implementovat rozšíření pro databázi Cassandra podle standardního modelu v Teiid. Teiid pro zařazení nového datového zdroje definuje své nejlepší praktiky, které poskytují znovupoužitelnost, přehlednost a dobrou čitelnost zdrojového kódu. Na tento fakt byl kladen důraz během celého návrhu a implementace konektoru.

Druhá část cíle obsahuje návrh a implementaci konektoru pro databázi Cassandra. Konektor v nástroji Teiid se dělí na dvě části, *Connector* a *Translator*. Architektura konektoru by měla být navržena tak, aby *Connector* a *Translator* tvořily dva samostatné moduly, které mezi sebou komunikují a využívají navzájem své služby. *Connector* by měl poskytovat vytvoření připojení k databázi Cassandra a získání metadat. Role modulu *Translator* je umožnit přechod mezi relačním modelem v Teiid a NoSQL logikou databáze Cassandra.

Poslední částí cíle této práce je testování funkčnosti a použitelnosti konektoru a zařazení konektoru do Open Source projektu JBoss Teiid. Splnění poslední části cíle vyplývá ze splnění předcházejících dvou částí.

1.2 Struktura práce

Tato sekce stručně popisuje kapitoly, které se zabývají jednotlivými cíly této práce.

Kapitola *JBoss Teiid* se zaměřuje na popis nástroje Teiid a jeho komponent v rámci datové federace, která bude předvedena na ukázkovém příkladu na konci kapitoly. Informace v této kapitole slouží zejména pro pochopení nástroje Teiid a datové federace, aby bylo možné objektivně navrhnout konektor pro databázi Cassandra.

Kapitola *Databáze Apache Cassandra* pojednává o databázi Cassandra, která představuje nový externí zdroj pro datovou federaci v Teiid. V kapitole je popsán datový model databáze, který je odlišný od tradičního schématu relační databáze. Tento problém částečně řeší výběr klienta pro přístup k databázi. Důležitá je volba správného klienta, aby nedocházelo ke zbytečným omezením, kterým se lze vyhnout. Analýza jednotlivých způsobů přístupu je popsána v této kapitole.

Po získání dostatečných znalostí o nástroji Teiid a databázi Cassandra může začít samotný vývoj konektoru. Celý průběh vývoje popisuje kapitola *Analýza, návrh a implementace*. Obsah kapitoly kopíruje standardní postup při vývoji aplikace. První část se zaměřuje na specifikaci požadavků pro konektor, kterou následuje návrh architektury a nakonec je rozebrána implementaci s použitými technologiemi.

Nezbytnou součástí vývoje aplikace je provedení testů, kterými se zabývá kapitola *Testování konektoru*. Testy jsou provedeny dva, první test ověřuje správnou funkcionalitu konektoru, druhý test testuje konektor pod zátěží.

Kapitola *Závěr* shrnuje výsledný produkt této práce, jeho dopad na komunitu JBoss a další plány pro rozšíření konektoru.

Kapitola 2

JBoss Teiid

Tato kapitola slouží k seznámení s *middleware* produktem JBoss Teiid (dále jen Teiid) a jeho použitím. Budou zde popsány základní principy, funkce a struktura nástroje Teiid. Na konci kapitoly bude vysvětlena datová federace s ukázkou jednoduchého případu užití. Podrobnou dokumentaci, návody a průvodce pokrývající téměř každý aspekt tohoto rozsáhlého projektu naleznete v různých verzích na oficiálním webu `<http://www.jboss.org/teiid/docs>`. Tato práce čerpá především z dokumentace pro verzi 8.5 [10].

2.1 Úvod do Teiid

Teiid je nástroj pro virtualizaci dat, který umožňuje aplikacím používat data z různých, heterogenních datových úložišť [11]. Teiid můžeme vidět jako sadu nástrojů, komponent, služeb a jejich použití za účelem vytváření a spouštění datových služeb typu *bi-directional*[1].

Teiid představuje komplexní pohled na integraci dat (trojice písmen eii ve slově T**eii**d symbolizují zkratku EII[2]), který je v době mnoha různorodých technologií a způsobů ukládání dat velmi žádaný. Poskytuje vysoce škálovatelné a výkonné řešení v oblasti integrace dat, zjednodušuje přístup k datům pro koncové uživatele.

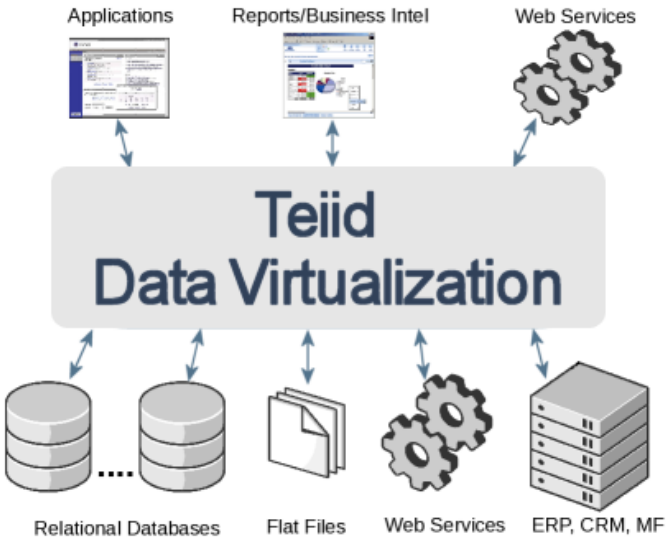
Přístup k datům na abstraktní úrovni je realizován pomocí datové federace 2.5.1, data jsou následně integrována v reálném čase napříč distribuovanými zdroji bez jakékoli replikace nebo přesouvání dat z původního zdroje.

---

1. *Bi-directional* je obousměrný přístup k datům, umožňuje čtení i zápis dat
2. EII (*Enterprise Information Integration*) je typ *middleware* softwaru, který umožňuje společnostem spojovat data z rozličných zdrojů do jedné aplikace[8, str. 1]

Teiid nabízí širokou škálu možností pro dotazování skrze relační abstrakci. Pro dotazování se používá standardizovaný SQL jazyk. Pomocí tohoto jazyka může uživatel klást dotazy na relační databáze, XML dokumenty, textové soubory atd. Například dotazování XML dokumentů je stejné jako dotazování relačních databází (speciální skalární funkce v SQL umožňují vrátit požadovanou část XML dokumentu). XML dokumenty jsou dynamicky konstruovány pomocí *Document Object Model* nebo použitím jazyka Xquery a jeho XMLQuery funkcí.

Následující obrázek ilustruje roli nástroje Teiid v řešení virtualizace dat [11]:



Obrázek 2.1: Virtualizace dat nástrojem Teiid

Teiid je open source a je důležitou součástí skupiny projektů *JBoss Community*[3], jak je uvedeno v [10, Reference guide]:

„Teiid is a Professional Open Source project and a critical component of the JBoss Enterprise Data Services Platform.“

3. *JBoss Community* - skupina vývojářů specializující se na vývoj open source *middleware* softwaru ve společnosti Red Hat

2.2 Struktura Teiid

V této podkapitole budou popsány všechny části nástroje Teiid, které jako celek poskytují efektivní způsob integrace dat zmíněný v předcházejícím textu. Podrobnější popis bude věnován pouze těm částem, které jsou potřebné pro implementaci konektoru pro databázi Apache Cassandra.

Teiid obsahuje sadu komponent a pomocných nástrojů. Každá komponenta je navržena pro specifický účel, pomocné nástroje slouží k monitorování, údržbě a zjednodušení práce v nástroji Teiid. Zde je seznam částí, z kterých se Teiid skládá [11]:

- ***Query engine***: *engine* pro zpracování různých typů dotazů
- ***Embedded***: odlehčená verze Teiid, která může být použita v libovolné java aplikaci bez potřeby aplikačního serveru
- **Server**: škálovatelný *enterprise* server běžící uvnitř JBoss AS[4], který poskytuje dodatečné bezpečnostní vlastnosti a nástroje pro správu
- **Konektory**: konektory poskytující připojení k různým datovým zdrojům
- **Ostatní nástroje**: Teiid Designer, webová konsole a další

2.3 Query Engine

Srdce nástroje Teiid je vysoce výkonný dotazovací *engine*, který zpracovává relační, XML, XQuery a procedurální dotazy z federovaných datových zdrojů [11]. Následující text popisuje pouze zpracování relačních dotazů, které se využívají v implementaci konektoru pro databázi Cassandra.

Dotazovací *engine* využívá jazyka SQL, konkrétně specifikace SQL-92 DML[5]. Vlastnosti z novějších verzí jsou postupně přidávány podle potřeby. Opět zde budou popsány pouze nejvýznamnější aspekty jazyka SQL, které jsou využity pro dotazování v nástroji Teiid.

---

4. JBoss AS - aplikační server na platformě Java EE [9]
5. `<http://en.wikipedia.org/wiki/SQL-92>`

SQL dotazy obsahují odkazy na tabulky a sloupce. Tyto odkazy jsou reprezentovány formou identifikátorů, které jednoznačně určují tabulky a sloupce v konkrétním SQL příkazu. Zpracování všech dotazů probíhá v kontextu virtuální databáze.

Dotazování tabulek a sloupců musí být prováděno podle určitých pravidel, aby nedocházelo ke konfliktům z důvodu federace více datových zdrojů. Toto omezení je zaručeno pomocí schémat, které obsahují informace pro každý datový zdroj.

Jména tabulek a sloupců jsou v následujícím tvaru [10, Identifiers]:

- Tabulka: `<schéma>.<tabulka>`
- Sloupec: `<schéma>.<tabulka>.<sloupec>`

Správné použití identifikátorů pro tabulku a sloupec řešící konflikt v důsledku více datových zdrojů je uveden v ukázce 2.1:

```
SELECT MujZdroj1.osoba.jmeno, MujZdroj2.osoba.jmeno
  FROM MujZdroj1.osoba, MujZdroj2.osoba
```

Ukázka 2.1: Dotaz na dvě stejně pojmenované tabulky v různých zdrojích

Identifikátory, literály a funkce spolu v kombinaci tvoří výraz, který může být použit kdekoliv v dotazu - SELECT, FROM, WHERE, GROUP BY, HAVING, nebo ORDER BY.

Teiid podporuje následující typy výrazů, podrobný popis viz. dokumentace [10, Expressions]:

- Identifikátory sloupců
- Literály
- Skalární funkce
- Agregační funkce
- *Window* funkce
- *CASE* a *Searched Case* výrazy
- Skalární subdotazy
- Odkazy na parametry
- Kritéria
- Pole

2.4 Konektory

Teiid disponuje řadou konektorů, které umožňují přístup k mnoha různým datovým zdrojům. Architektura konektorů TCA (*Teiid Connector Architecture*) poskytuje robustní mechanismus pro integraci externích systémů. TCA definuje uživatelské rozhraní mezi nástrojem Teiid a externím systémem. Toto rozhraní zahrnuje metadata, která se využívají při zpracování SQL dotazu v *Query Engine*, a schopnost importu metadat z externího systému. Konektor se typicky skládá ze dvou částí - *Resource adapter* a *Translator*. V některých případech konektor obsahuje dodatečné API nutné pro přístup k externímu systému.

2.4.1 *Resource Adapter*

V TCA je označován často jako *Connector*. Konektor slouží pro přístup k externímu zdroji. Poskytuje připojení, získání metadat, autentizaci a další potřebné služby pro konkrétní datový zdroj.

Teiid definuje konektor podle specifikace JCA[6]. Konektory v nástroji Teiid jsou vlastně konektory JCA, jejichž implementace může být upravena pro Teiid. Implementace konektoru může být provedena dvěma způsoby v závislosti na požadavcích.

V případě, že konektor vyžaduje pokročilou konfiguraci, správu bezpečnosti, pooling apod., je doporučeno využít specifikace JCA[13], která poskytuje dostatečnou podporu pro takovou implementaci. Pokud již existuje konektor JCA pro cílový EIS, může být použit jako plnohodnotný konektor v nástroji Teiid.

Další možnost, jak implementovat konektor v nástroji Teiid, je využití služeb *Teiid API*. Součástí *Teiid API* je framework, který pomáhá vývojářům při návrhu architektury a následné implementace vlastního konektoru. Teiid poskytuje několik tříd pro model konektoru umístěné v balíku *org.teiid.resource.api* [10, Using the Teiid Framework].

Třída *MyManagedConnectionFactory* slouží pro manipulaci s připojením k EIS, rozšiřuje třídu *BasicManagedConnectionFactory*. Obsahuje atributy jako adresa, uživatel, heslo atd. potřebné pro vytvoření připojení. Důležité je překrytí metody *createConnectionFactory()*, která poskytuje tovární metodu pro získání připojení. Příklad třídy je

6. JCA (*Java EE Connector Architecture*) - specifikace pro budování konektorů JCA[13]

uveden v ukázce 2.2:

```
public class MyManagedConnectionFactory extends
    BasicManagedConnectionFactory
{
    @Override
    public Object createConnectionFactory() throws
        ResourceException {
        return new MyConnectionFactory();
    }
    /* konfigurační atributy (metada pro tyto
        atributy jsou definována v ra.xml, viz níže)
        */
    String jmenoUzivatele;
    Integer pocet;
    //gettry, settry...
}
```

Ukázka 2.2: Třída pro práci s připojením, převzato (a částečně upraveno) z [10, Using the Teiid Framework]

Třída *MyConnectionFactory* překrývá metodu *getConnection()* zděděnou z třídy *BasicConnectionFactory*. Implementace metody *getConnection()* poskytuje připojení k externímu zdroji. Protože třída *MyManagedConnectionFactory* vytváří instanci *MyConnectionFactory*, může metoda *getConnection()* používat atributy z *Managed* třídy v ukázce 2.2. Příklad třídy je uveden v ukázce 2.3:

```
public class MyConnectionFactory extends
    BasicConnectionFactory
{
    @Override
    public MyConnection getConnection() throws
        ResourceException {
        return new MyConnection();
    }
}
```

Ukázka 2.3: Třída s metodou pro vytvoření připojení, převzato z [10, Using the Teiid Framework]

Třída *MyConnection* reprezentuje jednotlivé instance připojení. Objekt této třídy se objevuje i ve třídě *ExecutionFactory* (ukázka 2.6), kterou využívá *Translator*. Implementace třídy *MyConnection* obsahuje atributy a metody potřebné pro spojení s externím zdrojem. Pro manipulaci s připojením ve frontě je možno překrýt metody *isAlive()* a *cleanup()*. Jednoduchý příklad třídy je uveden v ukázce 2.4:

```
public class MyConnection extends BasicConnection
{
    public void provedNejakouOperaci(prikaz){
    // proved operaci na EIS, tuto metodu poziva
        Translator
    }
    @Override
    public boolean isAlive() {return true;}
    @Override
    public void cleanUp() {        }
}
```

Ukázka 2.4: Třída reprezentující připojení, převzato z [10, Using the Teiid Framework]

Soubor *ra.xml* uvnitř konektoru definuje konfiguraci pro každý atribut třídy *MyManagedConnectionFactory*. Reálné hodnoty atributů jsou předávány kontejnerem aplikačního serveru ze souboru *standalone-teiid-.xml*[7]. Příklad definice konfiguračního atributu je uveden v ukázce 2.5:

```
<config-property>
    <description>
        {$display:"${display-name}",$description:"${
            description}"}
    </description>
    <config-property-name>${property-name}</config-
        property-name>
</config-property>
```

Ukázka 2.5: Definice konfiguračního atributu převzato (a částečně upraveno) z [10, Using the Teiid Framework]

7. standalone-teiid.xml - konfigurační soubor pro server, obsahuje informace o datových zdrojích, jednotlivých instancích konektorů atd.

Jakmile jsou definovány všechny třídy a soubor *ra.xml*, může být konektor sestaven. Sestavení konektoru se provádí pomocí nástroje *Maven*, který se využívá pro sestavení celého projektu Teiid a jeho modulů. Výstupem sestavení konektoru je soubor RAR (*Resource Adapter Archive*), který obaluje třídy, soubory a jiné potřebné informace do jednoho archivu.

Nasazení konektoru na aplikační server se provádí pouze jednoduchým překopírováním RAR souboru.

2.4.2 *Translator*

Translátor je srdce TCA a chová se jako logický můstek mezi nástrojem Teiid a externím systémem [10, Translators]. Cílem translátoru je poskytnout převod z relačního modelu nástroje Teiid do specifické logiky externího zdroje, aby bylo možné pokládat dotazy na všechny zdroje v unifikovaném dotazovacím jazyku. Tento proces vyžaduje zpracování metadat, překlad dotazu, stanovení omezujících podmínek a jiné doplňující konfigurace nutné pro korektní provedení převodu.

Implementace translátoru v TCA vyžívá Teiid API stejně jako *Resource Adapter* 10. Základní třídou translátoru je *org.teiid.translator.ExecutionFactory*, která poskytuje připojení, získání metadat a volání dotazů. Proto vlastní translátor musí rozšiřovat tuto třídu a přepsat nezbytné metody pro účely zmíněné v předchozím odstavci. Příklad rozšiřující třídy *CustomExecutionFactory* s bezparametrickým konstruktorem je uveden v ukázce 2.6:

```
package org.teiid.translator.custom;

@Translator(name="custom", description="Connect to
    My EIS")
public class CustomExecutionFactory extends
    ExecutionFactory<MyConnectionFactory,
    MyConnection> {
     public CustomExecutionFactory() {
     }
}
```

Ukázka 2.6: Základní třída translátoru, převzato z [10, Extending the ExecutionFactory Class]

Jako první krok je potřeba definovat třídy *MyConnectionFactory* (ukázka 2.3) a *MyConnetion* (ukázka 2.4) pro rozšíření třídy *ExecutionFactory*. Tyto třídy často implementují rozhraní, které je použito v translátoru místo samotné implementace. Implementace obou tříd jsou pak definovány v *Resource Adapter* 10.

Třída *CustomExecutionFactory* může mít vlastní konfigurační atributy, pokud translátor pořebuje externí konfiguraci při sestavení virtuální databáze (ukázka 2.8). Každý konfigurační atribut je vybaven „gettry“ a „settry“ a musí mít anotaci *@TranslatorProperty*, která definuje metadata pro tento atribut.

Další konfigurační mechanismus translátoru jsou *Translator Capabilities*. Třída *ExecutionFactory* obsahuje sadu metod, které definují možnosti translátoru při zpracování dotazu. Metody poskytujicí možnosti podporované ve vlastním translátoru musí být přepsány v rozšiřující třídě *CustomExecutionFactory*. Seznam všech možností je uveden v dokumentaci [10, Translator Capabilities]. Příklad metody pro vnitřní spojení je uveden v ukázce 2.7:

```
@Override
        public boolean supportsInnerJoins() {
                return true;
        }
```

Ukázka 2.7: Definice metody, která povoluje možnost vnitřního spojení.

Získání metadat pro translátor poskytuje metoda *getMetadata()* ve třídě *ExecutionFactory*. Překrytí metody v rozšiřující třídě by mělo zajistit převod datového modelu externího zdroje na relační model v nástroji Teiid, aby bylo možné provést přeložení dotazu.

Spuštění dotazu je provedeno pomocí několika metod v závislosti na typu dotazu. Pro čtení dat z externího zdroje slouží metoda *createResultSetExecution(parametry)*. Opět je potřeba tuto metodu překrýt v rozšiřující třídě. Standardní implementace této metody poskytuje přeložení dotazu v jazyce SQL a následné provedení vlastní operace definované ve třídě *MyConnection* (ukázka 2.4). Pro vložení nebo aktualizaci dat slouží metoda *createUpdateExecution* s analogickým postupem v předchozí větě.

Aby bylo možné reagovat na vzniklé chyby během spouštění translátoru, Teiid poskytuje vlastní třídu výjimek *org.teiid.translator.Translat-*

orException a systém logování *org.teiid.logging.LogManager*.

2.5 Datová federace

V předchozích částech této kapitoly byla vysvětlena architektura a funkcionalita jednotlivých komponent nástroje Teiid. Znalost informací z předcházejícího textu je důležitý předpoklad pro lepší pochopení datové federace v nástroji Teiid, která bude ukázána na jednoduchém příkladu v této sekci. Nejdříve bude představena datová federace jako obecný princip.

2.5.1 Úvod do datové federace

Datová federace neboli virtualizace dat je široký pojem, který lze vyjádřit několika způsoby. Datovou federaci můžeme chápat jako produkt, který vznikne z federace datových zdrojů.

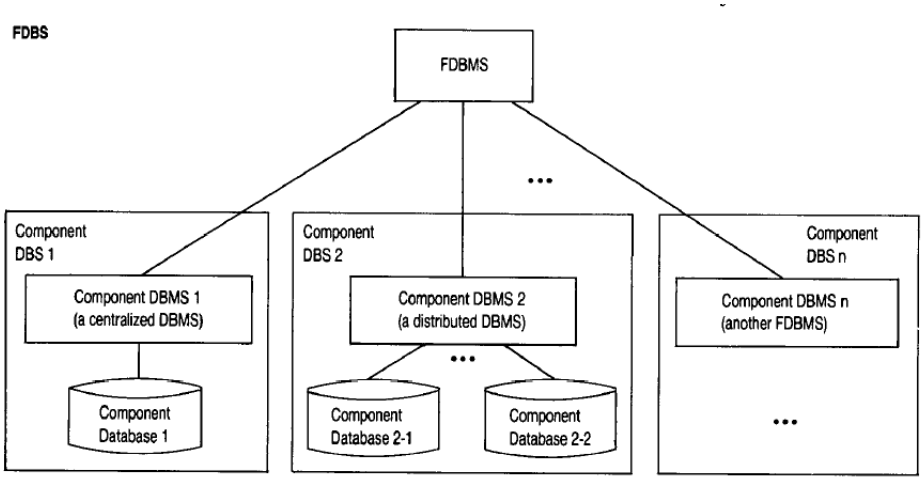
Federace datových zdrojů je reprezentována federovaným databázovým systémem. Pojem *fedarated database system* byl vytvořen Hammerem a McLeodem v roce 1979 a Heimbignerem a McLeodem vroce 1985 [14, s. 2]. Federovaný databázový systém může být definován následujícím způsobem.

Databázový systém (*DBS - database system*) se skládá ze softwaru, který se nazývá *database management system* (*DBMS*), a z jedné nebo více databází, které tento software spravuje. Federovaný databázový systém (*FDBS - federated database system*) je kolekce jednotlivých spolupracujících databázových systémů, které se chovají autonomně. Software, který poskytuje řízenou a koordinovanou obsluhu jednotlivých databázových systémů, se nazývá *federated database management system (FDBMS)* (obrázek 2.2) [14, s. 1-2].

Jednotlivé databázové systémy (*component DBSs*) se skládají z *component database* a *component DBMS*. *Component database* odkazuje na databázi konkrétního *component DBS*. *Component DBS* se může účastnit více datových federací. *DBMS* uvnitř *component DBS* nebo *component DBMS* může být centralizovaný *DBMS*, distribuovaný *DBMS* nebo další *FDBMS* [14, s. 2].

Component DBMSs se mohou lišit v několika aspektech jako jsou datové modely, dotazovací jazyky nebo různé možnosti managementu transakcí [14, s. 2]. Následující obrázek ilustruje model federovaného

databázového systému popsaný výše.



Obrázek 2.2: Model federovaného databázového systému, převzato z [14, s. 3]

2.5.2 Příklad datové federace v nástroji Teiid

Vytvoření datové federace v nástroji Teiid znamená agregaci datových zdrojů do jedné virtuální databáze. V této části bude předveden konkrétní scénář datové federace na jednoduchém ukázkovém příkladu. Ukázkový příklad představuje dva nezávislé systémy. První systém eviduje docházku v nějaké společnosti, druhý systém slouží pro plánování a efektivitu práce ve stejné společnosti. Oba systémy používají jako zdroj databázi PostgreSQL.

První zdroj s názvem *dochazkovy_system* obsahuje dvě tabulky, *zamestnanci* a *dochazka*. Každý záznam v tabulce *dochazka* je vždy spojen cizím klíčem *id_zam* s konkrétním zaměstnancem v tabulce *zamestnanci*.

dochazkovy_system_zamestnanci				
id	jmeno	prijmeni	adresa	PSC
58	Radek	Koubský	Brno 512	625 00
31	Jan	Novák	Brno 128	625 00

Tabulka 2.1: Tabulka zaměstnanci s ukázkovými daty

dochazkovy_system_dochazka					
id	id_zam	datum	prichod	odchod	pocet_hodin
1	58	2013-09-25	9:00	17:00	8
2	58	2013-09-26	9:00	17:00	8
1	58	2013-09-27	9:00	17:00	8
1	31	2013-09-25	9:00	17:00	8

Tabulka 2.2: Tabulka docházka s ukázkovými daty

Druhý zdroj *planovaci_system* a obsahuje opět dvě tabulky, *zamestnanci* a *pracovni_vykaz*. Vztah mezi tabulkami je podobný jako v předchozím případě, každý záznam v tabulce *pracovni_vykaz* se odkazuje pomocí cizího klíče *id_zam* do tabulky *zamestnanci*.

planovaci_system_zamestnanci				
id	jmeno	prijmeni	utilizace	hod_mzda
58	Radek	Koubský	100%	150
31	Jan	Novák	100%	200

Tabulka 2.3: Tabulka zaměstnanci s ukázkovými daty

planovaci_system_pracovni_vzkaz			
id	id_zam	datum	pocet_hodin
1	58	2013-09-25	8
2	58	2013-09-26	7
1	58	2013-09-27	7.5
1	31	2013-09-25	8

Tabulka 2.4: Tabulka pracovni vykaz s ukázkovými daty

Následující krok zahrnuje vytvoření virtuální databáze, která bude obsahovat dva modely, každý s jedním zdrojem. V tomto případě by bylo možné vytvořit jeden vícezdrojový model, protože metadata zdrojů jsou stejného typu. Rozdělení do dvou modelů je použito pouze pro ukázkový účel.

Virtuální databáze v nástroji Teiid neboli VDB je definována pomocí XML souboru. Vytvoření virtuální databáze může být provedeno dvěma způsoby.

První způsob je využití nástroje *Teiid Designer*[8], který umožňuje vytváření virtuální databáze, přidávání zdrojů, vytváření dotazů, testování datových služeb a mnoho dalších funkcí. Všechny operace se volají skrze uživatelské rozhraní, není potřeba jakéhokoli programátorského zásahu.

Přímočařejší postup je definice dynamické virtuální databáze *Dynamic VDB* bez využití nástroje *Teiid Designer*. Definice *Dynamic VDB* je uložena v souboru *nazev-vdb.xml*. Tento soubor obsahuje informace o VDB, importovaných zdrojích, typu translátoru a požadavcích pro import metadat. Způsob vytvoření virtuální databáze pomocí *Dynamic VDB* bude použit v ukázkovém příkladu.

Dynamic VDB mojeVDB-vdb.xml je uvedena v ukázce 2.8:

```
<?xml version="1.0"  standalone="yes"?>
<vdb name="mojeVDB" version="1">
<model name="dochazka">
    <property name="importer.useFullSchemaName"
        value="true"/>
    <source name="dochazka" translator-name="
        postgresql" connection-jndi-name="java:/
        dochazka"/>
</model>

<model name="plansys">
    <property name="importer.useFullSchemaName"
        value="true"/>
    <source name="plansys" translator-name="
        postgresql" connection-jndi-name="java:/
        plansys"/>
</model>
</vdb>
```

Ukázka 2.8: Definice virtuální databáze pro ukázkový příklad

Element *<vdb>* obsahuje atribut *name* definující jméno VDB při spuštění. Uvnitř elementu *<vdb>* se nachází jednotlivé modely.

Každý model ve virtuální databázi je definován v elementu *<model>*. Model se odkazuje na jednotlivé zdroje pomocí elementu *<source>*.

8. `<http://www.jboss.org/teiiddesigner>`

Element *<source>* obsahuje atributy pro jméno zdroje v tomto modelu, jméno translátoru a *connection-jndi-name* pro identifikaci zdroje v aplikačním serveru JBoss. Model může obsahovat elementy *<property>* pro specifikaci požadavků na zpracování metadat v translátoru. V ukázkovém příkladu je nastaveno *importer.useFullSchemaName* na hodnotu *true*, tento požadavek způsobuje, že odkaz na tabulky při sestavení dotazu bude ve tvaru *<model name>.<table name>*, který je uveden v ukázce 2.1.

Definice virtuální databáze využívá širokou škálu nastavení, podrobný popis je uveden v dokumentaci [10, VDBs]

Po definici VDB je potřeba vytvořit instanci třídy *ConnectionFactory*, která vytvoří spojení s datovým zdrojem. Příklad vytvoření instance *ConnectionFactory* je uveden v [10, Deploying the Adapter].

Jakmile je spojení navázáno, uživatel může sestavovat dotazy, které využívají data z obou zdrojů ve virtuální databázi. Příklad dotazu SQL, který vrátí počet hodin pobytu v práci a počet skutečně odpracovaných hodin pro konkrétního uživatele, je uveden v ukázce 2.1:

```
SELECT dsz.jmeno, dsz.prijmeni, dsd.datum, dsd.
    pocet_hodin as dochazka, pspv.pocet_hodin as
    vykazano FROM dochazkovy_system.zamestnanci as
    dsz, dochazkovy_system.dochazka as dsd,
    planovaci_system.pracovni_vykaz as pspv,
    planovaci_system.zamestnanci as psz WHERE dsz.id
     = psz.id AND dsd.datum = psvp.datum AND dsz.id
    = 58
```

Ukázka 2.9: Dotaz na virtuální databázi mojeVDB-vdb.xml

Použití plně kvalifikovaných jmen při identifikaci tabulek (ukazka 2.1) je nutný, protože zdroje ve virtuální databázi obsahují tabulky se stejnými jmény.

Cílem ukázky datové federace bylo poskytnutí jednoduchého, ale výstižného náhledu do logiky nástroje Teiid. Změny datových zdrojů, použití jiných translátorů a ostatních nastavení se provádějí analogickým postupem, který je ukázán v tomto demonstrativním příkladu.

Kapitola 3

Databáze Apache Cassandra

V této kapitole bude popsána databáze Cassandra, která představuje externí zdroj pro Teiid 2. Databáze Cassandra patří do skupiny NoSQL databází, které mají odlišný model od relačních databází. Nicméně srovnání NoSQL a relačních databází je přirozený proces, proto bude i v této kapitole provedeno srovnání v některých aspektech.

Cassandra je distribuovaný úložný systém pro řízení velkého množství strukturovaných dat, která jsou rozprostřena napříč mnoha servery. Poskytuje vysoce dostupnou a spolehlivou službu [12]. Takový úložný systém umožňuje vysokou škálovatelnost a konzistenci dat. Systém Cassandra je vyvíjen společností Apache [1] jako open-source projekt od roku 2008.

3.1 Datový model

Datový model databáze Cassandra je sloupcově-orientovaný model s dynamickým schématem. To znamená, že uživatel, na rozdíl od relačních databází, nepotřebuje předem definovat všechny sloupce pro uživatelskou aplikaci, protože každý řádek nemusí obsahovat stejný počet sloupců. Sloupce s jejich metadaty mohou být libovolně přidávány uživatelskou aplikací podle potřeby [5, s. 52].

Datový model je navržen pro data distribuovaná v obrovském rozsahu. Na rozdíl od relačních databází, kde se modelují tabulky a vztahy mezi nimy, návrh datového modelu pro databázi Cassandra zavisí na dotazech, které budou na tomto modelu spouštěny [5, s. 52].

Datový model databáze Cassandra je založen na principu klíče a hodnoty. Ke každému klíči je přiřazena hodnota, kterou reprezentuje pole bytů. Databázi Cassandra si lze představit jako obrovskou mapu.

Základní koncept databáze Cassandra se skládá z následujících bodů

[5, s. 52]:

- ***Cluster***: představuje stroje (uzly) v logické instanci Cassandra, mohou obsahovat více prostorů s klíči
- ***Keyspace***: kontejner pro jednotlivé *ColumnFamilies*, typicky jeden pro aplikaci
- ***ColumnFamilies***: obsahují sloupce, každý sloupec má jméno, hodnotu, časovou značku a odkaz na *row key*
- ***SuperColumns***: sloupce, ve kterých jsou vnořeny další podsloupce
- ***Row key***: odkaz na skupinu spolu souvisejících sloupců v *ColumnFamily*

3.1.1 Cluster

Cluster představuje množinu uzlů v instanci Cassandra (instance představuje konkrétní případ databáze Cassandra). Uzly mezi sebou komunikují prostřednictvím protokolu *Gossip*[1]. Pomocí protokolu *Gossip* může uzel zjistit umístění a informace o stavu ostatních uzlů, které jsou součástí stejného clusteru. Výměna stavových informací probíhá každou sekundu až na třech uzlech. Využitím komunikace peer-to-peer se informace o uzlech v clusteru šíří velmi rychle.

3.1.2 Keyspace

Keyspace obsahuje všechna data aplikace, která jsou uložena v *ColumnFamilies*. Podobá se schématu relační databáze. Keyspace slouží pro seskupení *ColumnFamilies* do jednoho celku.

3.1.3 ColumnFamily

ColumnFamily je tabulka v databázi Cassandra, obsahuje sloupce a řádky. Každý řádek *ColumnFamily* musí být odkazován svým klíčem (*row key* 3.1.6). Jednotlivé řádky nemusí obsahovat stejné sloupce.

1. Protokol Gossip - speciální protokol pro efektivní komunikaci mezi uzly v distribuovaném systému [5, s. 2-3]

Každá *ColumnFamily* si definuje svá metadata ve sloupcích, ale aktuální počet sloupců tvořících jeden konkrétní řádek je dán požadavky uživatelské aplikace.

Existují dva typy *ColumnFamilies* [5, s. 55]:

- *Static ColumnFamily*: standardní tabulka Cassandra databáze
- *Dynamic ColumnFamily*: definice vlastního datového typu

3.1.4 Column

Sloupec je nejmenší údaj v databázi Cassandra, je to n-tice obsahující jméno, hodnotu a časovou značku [5, s. 6]. Jméno je povinný atribut, může být statický (např. „jméno“ nebo „příjmení“) nebo se mění dynamicky při vytváření sloupce uživatelskou aplikací. Hodnota sloupce je nepovinný atribut, v některých případech může být prázdná.

Časová značka udává čas poslední změny sloupce a je určena uživatelskou aplikací. Pokud chce několik uživatelů změnit stejný sloupec současně, uloží se změna s poslední časovou značkou.

3.1.5 SuperColumn

Více sloupců může být vloženo do jednoho *SuperColumn*, který obsahuje jméno sloupce a seznam vnořených sloupců (podsloupců). Využívá se při seskupování sloupců se stejnou vyhledávanou hodnotou. Všechny sloupce v *SuperColumn* musí být zrekonstruovány tak, aby bylo možné přečíst jejich hodnotu. Použití *SuperColumn* je vhodné v případě, že počet sloupců je relativně malé číslo [5, s. 59].

3.1.6 Row

Řádek v tabulce *ColumnFamily*. Každý řádek *ColumnFamily* je jednoznačně identifikovatelný svým klíčem (*row key*), který má stejnou funkci jako primární klíč v relační databázi. ColumnFamily je rozdělena pomocí klíče, který je automaticky indexován.

3.1.7 Datové typy

Databáze Cassandra definuje datový typ pro řádek (nebo hodnotu klíče) a sloupec. Datový typ pro řádek resp. hodnotu v daném sloupci se na-

zývá *validator*. Datový typ pro jméno sloupce se nazývá *comparator*. Datový typ může být definován při vytváření schématu *ColumnFamily*. Definice datového typu není povinná. Cassandra ukládá jméno sloupce a hodnotu jako pole bytů, tato konfigurace je nastavena defaultně. Cassandra poskytuje soubor datových typů [5, s. 60], které mohou být použity pro *validator* i *comparator*. Výjimku tvoří typ *CounterColumnType*, který je povolen pouze pro hodnotu ve sloupci.

3.1.8 Indexace

Index je datová struktura, která umožňuje rychlé, efektivní vyhledávání dat odpovídajících dané podmínce [5, s. 61]. Cassandra používá index klíče (*row key*) u *ColumnFamily* jako primární index. Tento index si uchovává každý uzel, který zpracovává data s tímto indexem spojená. Primární index v databázi Cassandra umožňuje vyhledávání řádků tabulky podle jejich klíče. Každý uzel si pamatuje klíče, které spravují ostatní uzly. Vyhledávání požadovaných řádků je efektivnější, protože se prohledává pouze na určitých uzlech.

3.2 Přístup k databázi Cassandra

Pro přístup k databázi Cassandra existuje několik možností, které se vyvíjely společně s vývojem databáze. K databázi lze přistupovat prostřednictvím různých klientů využívajících své vlastní API nebo pomocí jazyka CQL.

3.2.1 Thrift API

Thrift API bylo představeno v první verzi databáze Cassandra. Jazykově nezávislé API založeno na technologii RPC[2] sloužilo jako spodní vrstva pro vývojáře klientů v různých programovacích jazycích.

Ukázalo se, že tento přístup není optimální. Použití Thrift API bez zastřešujícího klienta je nevhodné, protože přístup se děje na nízké úrovni, a podpora nových vlastností (například sekundární indexy ve verzi 0.7) se stala těžce udržitelná pro klienty v různých programovacích jazycích [5, s. 70].

2. `<http://en.wikipedia.org/wiki/Remote_procedure_call>`

Seznam klientů používajících Thrift API je uveden v [1, ClientOptionsThrift]

3.2.2 Cassandra CLI

Rozhraní pro příkazový řádek se objevilo ve verzi 0.7, může být použito pro manipulaci s daty pomocí jazyka DDL a DML. Samotné rozhraní není určeno pro vývoj aplikací, slouží jako prostředek pro datové modelování a pro seznámení se s databází Cassandra [5, s. 71].

3.2.3 CQL

Po zkušenostech s Thrift API a Cassandra CLI příchází Cassandra 0.8 s novým rozhraním CQL 1.0, které poskytuje standardní přístup k databázi Cassandra.

Cassandra Query Language (CQL) je dotazovácí jazyk pro databázi Cassandra vycházející z jazyka SQL (*Structured Query Language*). Ačkoli má CQL mnoho podobných vlastností s jazykem SQL, existují mezi nimi základní rozdíly. Například adaptace CQL na datový model a architekturu databáze Cassandra nepodporuje operace spojení tabulek, jejichž použití nemá význam v rámci NoSQL databáze [5, s. 117].

CQL přidává abstraktní vrstvu, která skrývá implementační detaily a poskytuje nativní syntaxi pro sestavení dotazů. Použití CQL přináší pro nové aplikace jednodušší přístup než Thrift API 3.2.1.

CQL se vyvíjelo společně s databází Cassandra. CQL 2.0 bylo vydáno společně s Cassandra 1.0, ale je nekompatibilní s verzí Cassandra 0.8.

Ve verzi Cassandra 1.1 se CQL stalo primárním rozhraním pro DBMS databáze Cassandra. CQL specifikace byla označena CQL 3. Největší přínos v CQL 3 je podpora pro složené primární klíče a umožnění velmi širokých řádků [5, s. 71].

CQL je budoucnost pro vývoj klientského API databáze Cassandra. Klienti používajících CQL jsou součástí projektu Apache Cassandra.

Seznam klientů je uveden v [1, ClientOptions]

Kapitola 4

Analýza, návrh a implementace

Konektor pro databázi Cassandra je vyvíjen jako samostatný modul v nástroji Teiid. Teiid definuje vlastní prostředky pro vývoj konektorů, které jsou součástí *Teiid Connector Architecture*. Konektor pro databázi Cassandra vychází z TCA při návrhu své vlastní architektury. Cílem tohoto přístupu bylo navrhnout konektor tak, aby byly splněny požadavky pro korektnost konektoru a bylo možné zařadit konektor oficiálně do projektu.

Teiid je vyvíjen na platformě Java, proto použité technologie při implementaci konektoru pracují na stejné platformě.

V následujícím textu bude popsán postup při vývoji konektoru, který zahrnuje specifikaci požadavků, návrh, použité technologie a implementaci.

4.1 Specifikace požadavků

Hlavním požadavkem na tuto práci bylo vytvoření JBoss Teiid konektoru pro databázi Apache Cassandra, aby bylo možné provádět čtecí dotazy přes relační abstrakci. Tento požadavek lze rozdělit do několika funkčních požadavků popsaných v následujících bodech:

- **Vytvoření virtuální databáze:**

  Aby mohl uživatel sestavovat dotazy, musí nejdříve vytvořit virtuální databázi (viz ukázka 2.8), na které budou dotazy spouštěny.

- **Připojení k externímu zdroji:**

  Předpoklad pro připojení k externímu zdroji je zařazení zdroje do některého modelu ve virtuální databázi a vytvoření *connection factory* s požadovanou konfigurací v aplikačním serveru.

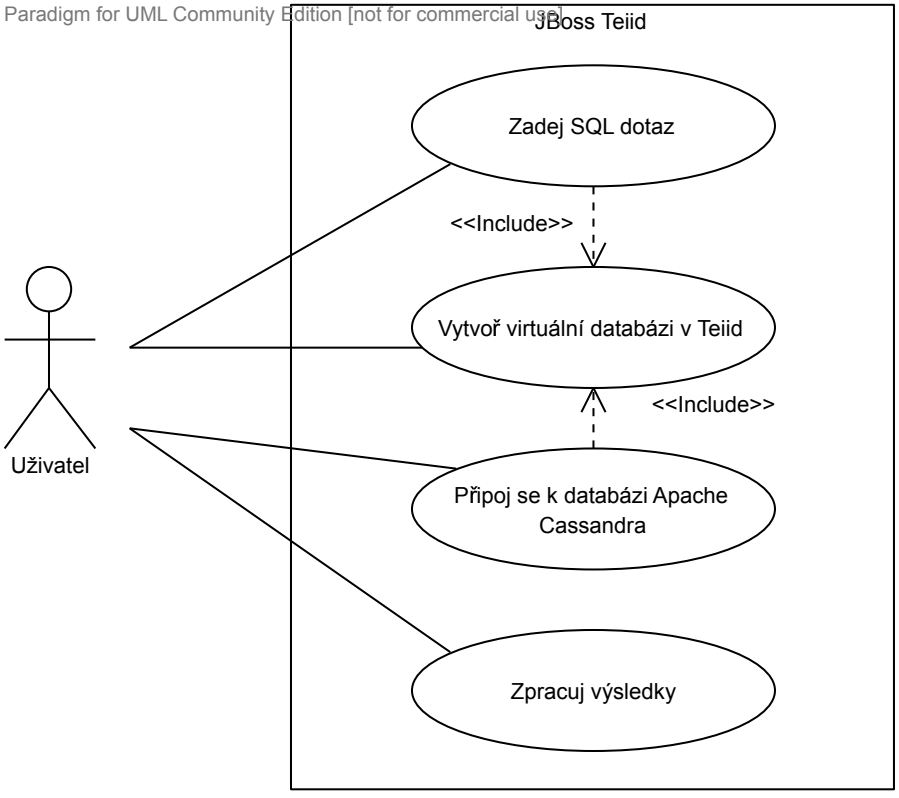
- **Zadání dotazu:**

  Jakmile je zadán SQL dotaz, translátor 2.4.2 provede přeložení dotazu do podoby, kterou je externí zdroj schopen zpracovat.

- **Zpracování dat:**

  Za předpokladu, že nenastala chyba během připojení ke zdroji a překladu dotazu, translátor provede zpracování výsledků z externího zdroje do čitelné podoby.

Specifikace funkční požadavků pomocí jazyka UML je uvedena v diagramu případů užití:



Obrázek 4.1: Diagram případů užití.

4.2 Návrh

Jak již bylo zmíněno na začátku kapitoly, při návrhu konektoru bylo využito specifikace *Teiid Connector Architecture* (viz 2.4). Cílem návrhu bylo vytvořit konektor tak, aby byly splněny požadavky vycházející z TCA a současně bylo poskytnuto rozhraní, které umožňuje připojení k externímu zdroji, zpracování metadat, překlad dotazu a následné vrácení dat uživateli.

Důležitým krokem při návrhu konektoru byla volba klienta pro přístup k databázi Cassandra. Nejprve byla provedena analýza klientů s různým přístupem k databázi (viz 3.2). Po analýze následovalo testování klientů na jednoduchých CRUD operacích pomocí nástroje *JUnit*[1]. Pro testování byly vybráni 3 klienti. *Firebrand*[2] a *Playorm*[3] ze skupiny klientů používajících Thrift API a klient *DataStax Java driver*[4] ze skupiny klientů využívajících jazyka CQL.

Na základě dokumentace a výsledků z testování se autor rozhodl použít klienta *DataStax Java driver* z následujících důvodů:

- Oba klienti na bázi Thrift API používají ORM[5]. Použití ORM v TCA by vyžadovalo dodatečné netriviální řešení pro import metadat a překlad dotazu v translátoru. *DataStax Java driver* poskytuje jednoduché rozhraní, které umožňuje spouštět dotazy v nativním formátu CQL a poskytuje funkce pro import metadat.

- Podpora klientů s Thrift API se pomalu snižuje, naopak klienti na bázi CQL procházejí neustálým vývojem, který reaguje na nové verze databáze.

Konektor v TCA se typicky dělí na dvě komponenty, *Resource Adapter*(někdy taky nazývaný konektor) a *Translator*.

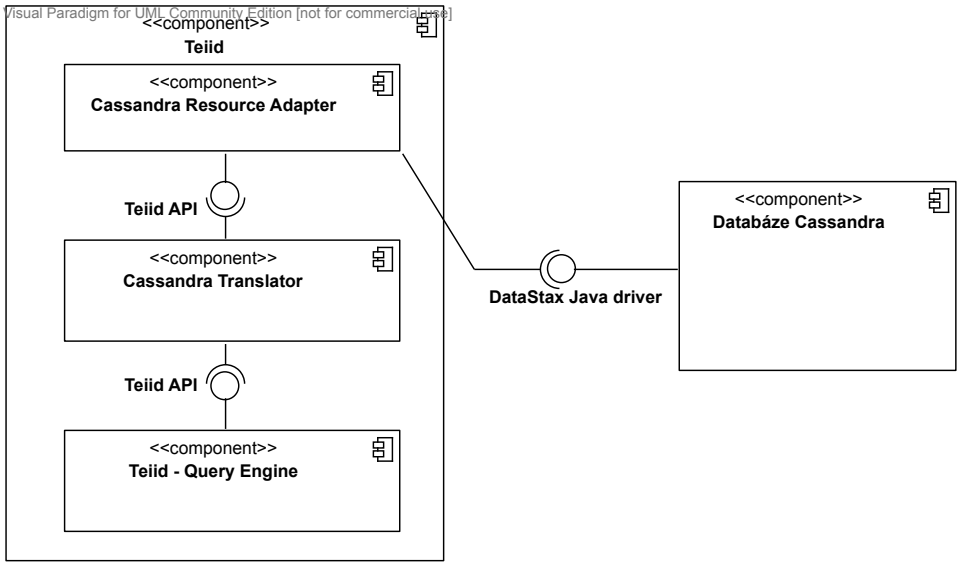
Resource Adapter poskytuje připojení k databázi Cassandra prostřednictvím klienta a definuje potřebné funkce, které jsou volány v translátoru. Funkce slouží zejména pro konfiguraci připojení a pro přístup k datům v databázi.

---

1. <http://junit.org/>
2. <http://firebrandocm.org/>
3. <http://buffalosw.com/wiki/playorm-documentation/>
4. <https://github.com/datastax/java-driver>
5. <http://en.wikipedia.org/wiki/Object-relational_mapping>

Translátor poskytuje 3 zásadní funkce vytvářející logickou vrstvu mezi nástrojem Teiid a databází Cassandra. Jako první se provede transformace metadat do relačního modelu. Jakmile jsou metadata v Teiid vytvořena, translátor provede překlad dotazu sestaveného v jazyce SQL do jazyka CQL. Korektně přeložený dotaz může být následně použit v konektoru, který definuje funkci pro dotazování v jazyce CQL. Poslední krok je transformace výsledných dat. Data jsou v databázi Cassandra uložena jako pole bytů, proto je nutný převod dat do čitelné podoby.

Architektura konektoru a jeho spolupráce s ostatními komponentami v nástroji Teiid je uvedena v následujícím diagramu komponent:



Obrázek 4.2: Diagram případů užití.

4.3 Použité technologie

V této sekci budou popsány technologie a nástroje, které byly použity při implementaci konektoru. Na použité technologie v nástroji Teiid se vztahují následující požadavky:

- Teiid je *Open Source* a všechny technologie, které byly použity při vývoji, musí být vedeny jako *Open Source*.

- Teiid je naprogramován v jazyce Java, na tento fakt je brán ohled při výběru technologií použitých v Teiid

4.3.1 Eclipse

Pro vývoj robustního softwaru jako je Teiid je nezbytné využití výhod, které poskytuje vývojového prostředí. Vývojové prostředí obsahuje nástroje pro zjednodušení práce programátora a urychlení samotného vývoje. Mezi hlavní nástroje patří editor zdrojového kódu, nástroj pro hledání chyb a podpůrné nástroje pro sestavení a překlad kódu. Teiid se standardně vyvíjí v prostředí Eclipse, které patří do *Open Source* a slouží zejména pro vývoj aplikací na platformě Java. Eclipse obsahuje mnoho zásuvných modulů pro Teiid, např. modul pro Teiid Designer[6], modul pro aplikační server JBoss nebo moduly pro Maven[7] a Git[8].

4.3.2 Maven

Standardní postup při vývoji aplikací zahrnuje výběr nástroje pro automatizaci sestavení a překladu zdrojového kódu. Teiid používá nástroj Apache Maven pro správu celého projektu a sestavení zdrojového kódu. Maven je jeden z předních nástrojů pro správu projektů na platformě Java.

Maven sestavuje projekt na základě svého POM (*Project Object Model*), který je definován v souboru *pom.xml*, a sadou pluginů, které jsou sdíleny ve všech projektech sestavených v Maven [3]. Poskytuje řadu nastavení a funkcí, které může vývojář použít pro sestavení Java aplikace. Několik nejdůležitějších funkcí, které Maven poskytuje Teiid, jsou popsány v následujícím textu.

Konfigurace a popis projektu je první krok při vytváření projektu v Maven. Každý projekt v Maven umožňuje definovat množinu vlastností a informací o projektu v souboru *pom.xml*. V Teiid je zavedena standardní definice pro každý projekt/modul: Jméno, verze, popis, identifikační číslo, identifikační číslo skupiny projektů, informace o rodičovském projektu, definice výstupního souboru po sestavení projektu (např. RAR soubor pro *Resource Adapter*).

---

6. `<http://www.jboss.org/teiiddesigner>`
7. `<http://www.eclipse.org/m2e/>`
8. `<http://www.eclipse.org/egit/>`

Závislosti patří mezi nejvýznamnější prvek v nástroji Maven. Tento mechanismus umožňuje dynamicky spravovat knihovny, které jsou použity v projektu. Knihovny jsou projekty Maven umístěny v nějakém repozitáři. Definice závislosti na nějaké knihovně obsahuje: identifikační číslo skupiny projektů, identifikační číslo projektu a rozsah (rozsah definuje omezení v kontextu zpracování závislosti). Během sestavení projektu se knihovny stahují ze vzdáleného repozitáře, kde jsou uloženy zdrojové kódy jednotlivých knihoven. Teiid a jeho moduly obsahují mnoho závislostí na různých knihovnách a jednotlivé moduly jsou závislé i na jiných modulech v Teiid.

Jak již bylo řečeno v předchotím odstavci, knihovny jsou importovány z určitého repozitáře pomocí závislostí. Maven Central Repository je centrální veřejný repozitář obsahující velké množství knihoven, které jsou volně dostupné. Maven umožňuje uživateli vytvořit vlastní repozitář s knihovnami, pokud je to vyžadováno. Přístup ke všem repozitářům je definován v souboru *settings.xml*, který je vytvořen při instalaci Maven na uživatelské stanici a obsahuje pouze přístup k centrálnímu repozitáři. Teiid vyžaduje přidání dalšího repozitáře ze skupiny repozitářů JBoss[9].

4.3.3 Git

Teiid používá verzovací systém Git pro správu zdrojového kódu a ostatního obsahu. Hlavní výhodou Git je jeho distribuovanost, která umožňuje vlastní lokální repozitář pro každého vývojáře. Sdílení otevřeného kódu s vnějším světem je zprostředkováno webovou službou GitHub[10].

4.3.4 DataStax Java driver

Jak již bylo zmíněno v části o návrhu 4.2, pro přístup k databázi Cassandra byl vybrán klient DataStax Java driver. Klient pracuje s CQL 3 a binárním protokolem databáze, který byl představen ve verzi Cassandra 1.2 [6].

Architektura klienta je rozdělena do vrstev. Spodní vrstvu tvoří jádro. Jádro obsluhuje všechny operace související s připojením ke clusteru databáze Cassandra (např. správa připojení, nalezení nových uzlů

9. <https://community.jboss.org/wiki/MavenRepository>
10. <http://en.wikipedia.org/wiki/GitHub>

atd.) a poskytuje relativně nízkoúrovňové API, na kterém může být budována další vrstva pro vyšší úrovně aplikace. Dotazy mohou být spouštěny synchronně i asynchronně, klient poskytuje podporu připravených příkazů a obsahuje podpůrnou třídu, která může být použita pro dynamické sestavení dotazů [6].

4.3.5 JBoss AS

JBoss AS je aplikační server na platformě Java EE vyvíjený pod záštitou komunity JBoss ve společnosti Red Hat. Server prošel mnohaletým vývojem, ve kterém se několikrát měnilo jeho pojmenování. Poslední verze JBoss AS byla přejmenována na WildFly [11].

Teiid má stejnou strukturu jako aplikační server JBoss. Nasazení Teiid na JBoss AS pouze přidá artefakty a konfigurační soubory do struktury aplikačního serveru. Při spuštění JBoss AS se spustí i Teiid a jeho komponenty.

4.4 Implementace

V této sekci bude popsána implementace konektoru pro databázi Cassandra s využitím technologií popsaných v předcházející sekci. Konektor je rozdělen na dva moduly Maven:

- *connector-cassandra*: modul představující *Resource Adapter* 2.4.1
- *translator-cassandra*: modul představující *Translator* 2.4.2

Moduly se nacházející uvnitř rodičovského projektu *connectors*, který obsahuje všechny konektory definované v Teiid.

4.4.1 Cassandra Resource Adapter

Hlavní funkce Cassandra Resource Adapter je přímá komunikace s databází Cassandra. Tento proces zahrnuje připojení k databázi, získání metadat a spuštění CQL dotazu. Třídy poskytující funkcionalitu jsou umístěny v balíku *org.teiid.resource.adapter.cassandra*. Obecné třídy v Teiid obsluhující připojení implementují standardní rozhraní z balíků

11. `<http://wildfly.org/>`

javax.resource.cci a *javax.resource.spi*. Násjedující popis implementace vychází s několika modifikacemi z 2.4.1.

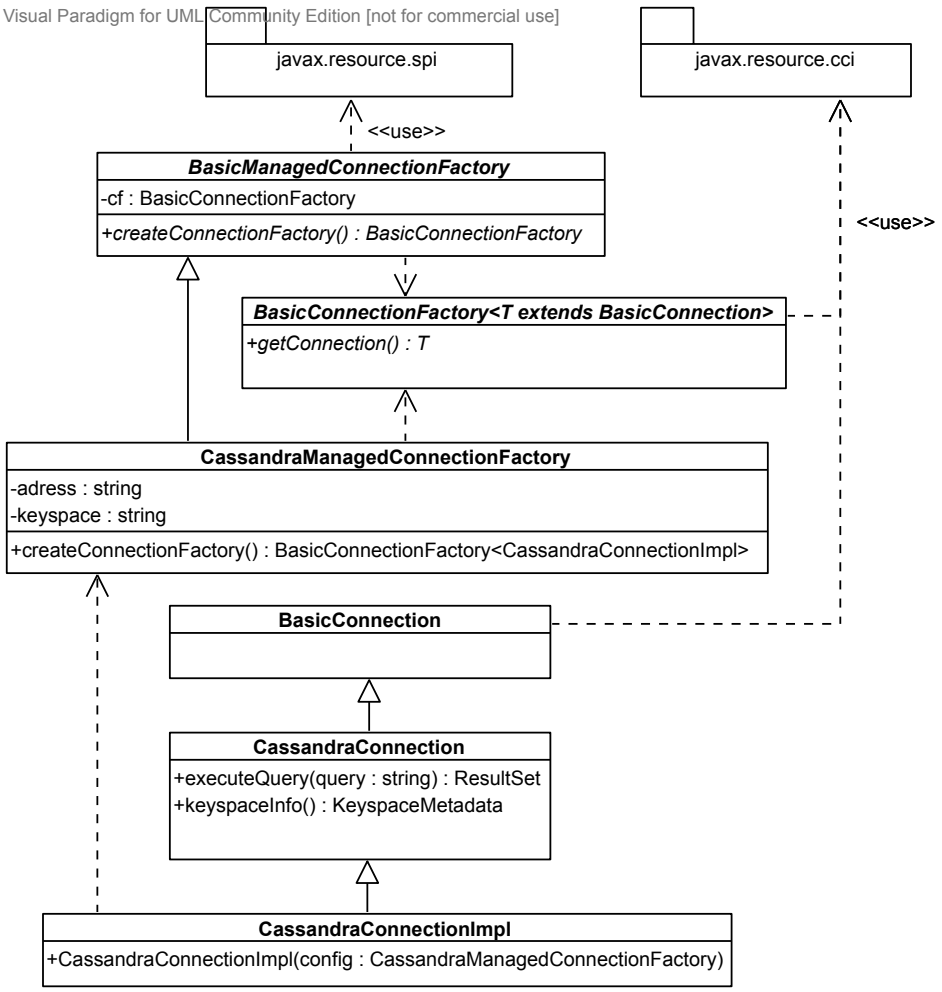
Primární třída *CassandraManagedConnectionFactory* rozšiřuje abstraktní třídu *ManagedConnectionFactory* a přepisuje její metodu *createConnectionFactory()*. Obsahuje atributy *address* a *keyspace*, které definují adresu uzlu v clusteru a specifické jméno pro *keyspace*.

Metoda *createConnectionFactory()* vrací novou instanci *BasicConnectionFactory*. Ve vytvořené instanci *BasicConnectionFactory* je typový parametr *T* nahrazen třídou *CassandraConnectionImpl* a současně je přepsána abstraktní metoda *T getConnection()*. Metoda *T getConnection()* vrací instanci *CassandraConnectionImpl*, která je vytvořena pomocí konstruktoru s jedním parametrem, který je nahrazen počáteční třídou *CassandraManagedConnectionFactory*. Tímto je zajištěno, že instance *CassandraConnectionImpl* má přístup k datům *Managed* třídy a může se připojit k databázi.

Po vykonání předešlých kroků je instance *BasicConnectionFactory* vrácena metodou *createConnectionFactory()*, která je volána v instanci *CassandraManagedConnectionFactory*.

Třída *CassandraConnectionImpl* implementuje rozhraní *CassandraConnection*, které je umístěno v translátoru a poskytuje 2 metody. Metoda *executeQuery(String query)* spustí CQL dotaz, který je předán jako parametr, a vrací data v binární podobě. Metoda *keyspaceInfo()* vrací metada popisující *keyspace* (schéma) databáze.

Na následující straně je uveden diagram tříd, který obsahuje třídy využité v implementaci Cassandra Resource Adapter:



Obrázek 4.3: Diagram tříd pro Cassandra Resource Adapter.

4.4.2 Cassandra Translator

Translátor zastupuje nejvýznamnější prvek v konektoru. Poskytuje přechod mezi relačním modelem v Teiid a NoSQL logikou databáze Cassandra. Translator je rozdělen do tří balíku obsahujících jednotlivé třídy: *org.teiid.translator.cassandra*, *org.teiid.translator.cassandra.execution* a *org.teiid.translator.metadata*. Hlavní třída představující přístupový bod translátoru je *CassandraExecutionFactory*, která rozšiřuje třídu *Execu-*

tionFactory<F, C>. Důležitým krokem je nahrazení typového parametru *C* rozhraním *CassandraConnection*, které je využíváno v metodách přístupové třídy. Celý proces odehrávající se v translátoru je rozdělen do několika kroků.

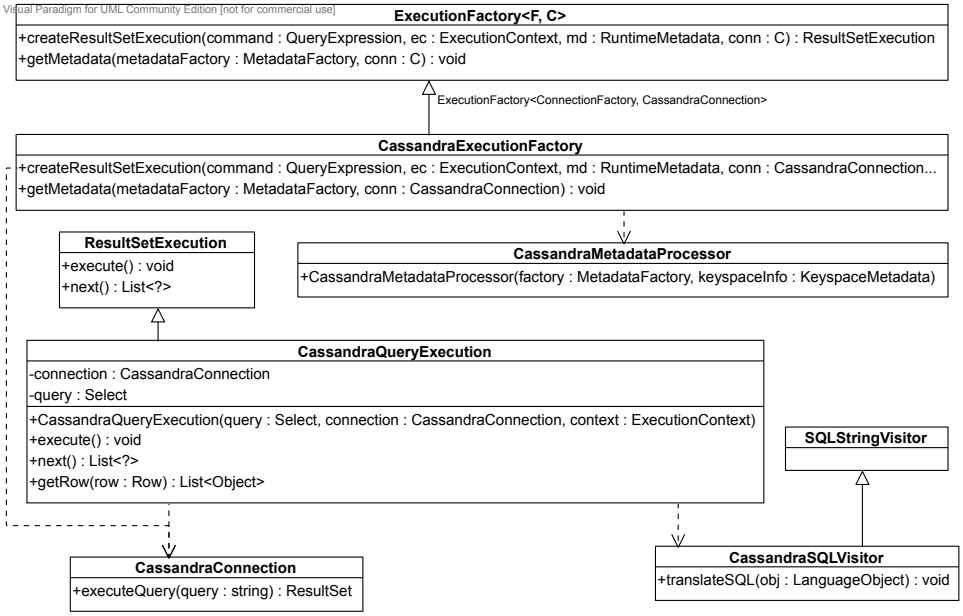
Nejdříve je potřeba získat informace o metadatech externího zdroje, aby mohl být proveden překlad dotazu. Získání metadat poskytuje implementace metody *getMetadaata(MetadataFactory metadataFactory, CassandraConnection conn)*, která je zděděna z třídy *ExecutionFactory<F, C>*. V těle metody je vytvořena instance třídy *CassandraMetadataProcessor*, na které je zavolána metoda *processMetadata()*, která převede schéma databáze Cassandra do relačního modelu v Teiid.

Jakmile jsou metadata získána, translátor může zahajít překlad dotazu a následně čtení dat. Operace pro překlad dotazu a čtení dat je inicializována metodou *CassandraQueryExecution createResultSetExecution(QueryExpression command, ExecutionContext executionContext, RuntimeMetadata metadata, CassandraConnection connection)* uvnitř třídy *CassandraExecutionFactory*. Metoda vrací novou instanci třídy *CassandraQueryExecution*, které předává parametry pro další zpracování. Třída *CassandraQueryExecution* obsahuje metodu *execute()*. Uvnitř metody *execute()* je volána metoda *ResultSet executeQuery(String query)*, která vloží data do objektu *ResultSet* . Před nahrazením parametru reálným dotazem musí být proveden překlad. Překlad poskytuje metoda *translateSQL(LanguageObject obj)* v třídě *CassandraSQLVisitor*.

Poslední krok je zpracování vrácených dat. Protože metoda *ResultSet executeQuery(String query)* vrací data z databáze v binární podobě, je nutné provést transformaci dat do čitelné podoby. K tomuto účelu slouží metoda *List<Object> getRow(Row row))* ve třídě *CassandraQueryExecution*. Metoda *List<Object> getRow(Row row))* využívá pomocné metody z klienta *DataStax Java Driver*. Nejdříve je zjištěn datový typ sloupce v jazyce CQL[12], aby mohla být zavolána metoda pro převedení binární hodnoty sloupce na hodnotu příslušného datového typu v jazyce Java. Tento proces se opakuje pro každý sloupec v právě zpracovávaném řádku.

12. Seznam všech datových typů v jazyce CQL: `<http://www.datastax.com/documentation/cql/3.0/webhelp/index.html#cql/cql_reference/cql_data_types_c.html#concept_ds_wbk_zdt_xj>`

Následující diagram tříd obsahuje třídy využité v implementaci *Cassandra Translator*:



Obrázek 4.4: Diagram tříd pro Cassandra Translator.

Kapitola 5

Testování konektoru

Testování softwaru je nepostradatelnou součástí každého vývoje, konektor pro databázi Cassandra není žádnou výjimkou. Tato kapitola je rozdělena do dvou částí. V první části bude testována funkcionalita konektoru (připojení k databázi, získání metadat, překlad dotazu a převod vrácených dat z binárního tvaru). Druhá část je věnována jednoduchým testům výkonu, které ověřují konektor pod zátěží. Testuje se pouze čtení dat z databáze.

Při testování byl konektor nasazen na aplikační server JBoss AS 7.1.1 jako součást celého projektu Teiid 8.3.0. Databáze s testovacími daty byla použita ve verzi Cassandra 1.2.5. Pro testování byla vytvořena virtuální databáze *cassandra-vdb.xml* s jedním modelem *Cassandra-NoSql* (viz 2.5.2).

5.1 Test č. 1

První test ověřuje funkčnost konektoru. Pro tento test byly vytvořeny dvě *ColumnFamily*, *testtypes1* a *testtypes2*. Tabulky obsahují sloupce, které zastupují většinu datových typů v jazyce CQL. Smysl testu je ověřit, že převod všech sloupců a jejich binárních hodnot proběhne správně a uživatel uvidí vrácená data v čitelné podobě. Následující tabulky ukazují výsledky vrácených hodnot (řádky jsou zalomeny kvůli velké šířce) po zavolání dotazu *SELECT * FROM CassandraNoSql.testtypes1* a *SELECT * FROM CassandraNoSql.testtypes2*.

Výsledky ukazují, že transformace binárních dat proběhla v pořádku, data jsou v čitelné podobě. Převod datového typu BLOB[1] nemá žádný význam, neboť jeho přirozená podoba je v binárním tvaru. Ko-

1. <http://en.wikipedia.org/wiki/Binary_large_object>

testuuid	testbigint	testboolean	testdecimal
756716f7-2e54-4715-9f00-91dcbea6cf50	100	true	15.5
testdouble	testfloat	testint	testtext
20.45	20.05	3	testtext
testvarint	testtimestamp	testvarchar	
12	2013-05-13	testvarchar	

Tabulka 5.1: Tabulka testtypes1 s testovanými daty

testascii	testblob	testinet
104101108108111	NULL	/127.0.0.1
testlist	testmap	testset
[17, 4, 2]	{band=Beatles, fruit=apple}	[cat, kitten, pet]

Tabulka 5.2: Tabulka testtypes2 s testovanými daty

rektní připojení a překlad dotazu je zřejmý, protože byla vrácena data v důsledku zavolání dotazu.

5.2 Test č. 2

V druhém testu je konektor vystaven různým zátěžovým podmínkám, které testují výkon konektoru. Test by proveden na procesoru Intel(R) Core(TM) i3 2.27 GHz a paměti o kapacitě 3 GB. Pro testování byl použit nástroj Apache JMeter[2], který umožňuje simulovat různé podmínky.

Pro tento test byly vytvořeny tři *ColumnFamily*, *small_family* s obsahem 100 řádků, *mediuml_family* s obsahem 1000 řádků a *large_family* s obsahem 10 000 řádků. Každá tabulka obsahuje sloupec „id“ (primární klíč) typu *timeuuid* a trojici sloupců typu *text*, které obsahují retězec reprezentující náhodně vygenerovanou hodnotu typu UUID[3].

2. <http://jmeter.apache.org/>
3. <http://en.wikipedia.org/wiki/Universally_unique_identifier>

Test je rozdělen do dvou scénářů v závislosti na počtu klientů a počtu opakovaní jednotlivých dotazů. Dotazy jsou 3, každý čte vždy všechna data z příslušné *ColumnFamily*. Pro každý scénář jsou provedeny tři měření. Výsledky z jednotlivých testů v nástroji Apache JMeter jsou uvedeny v tabulkách, které obsahují následující sloupce [2, Summary Report]:

- Štítek - označení pro vzorek. Vzorek reprezentuje požadavek např. na vzdálený server apod.
- Vzorky - počet vzorků se stejným štítkem
- Průměr - průměrný čas pro celou skupinu vzorků
- Min - nejnižší uplynulý čas vzorků se stejným štítkem
- Max - nejvyšší uplynulý čas vzorků se stejným štítkem
- Chyby - chybné požadavky v procentech

Měřený čas je uveden v milisekundách.

První případ: 10 uživatelů 10 opakování požadavků, tři spuštění JMeter jsou uvedeny v následujících tabulkách:

Štítek	Vzorky	Průměr	Min	Max	Chyby
small_family	100	339	39	2478	0.0 %
medium_family	100	415	58	2252	0.0 %
large_family	100	1849	416	4923	0.0 %
TOTAL	300	868	39	4923	0.0 %

Štítek	Vzorky	Průměr	Min	Max	Chyby
small_family	100	273	47	2311	0.0 %
medium_family	100	412	105	1839	0.0 %
large_family	100	1805	822	5293	0.0 %
TOTAL	300	830	47	5293	0.0 %

Štítek	Vzorky	Průměr	Min	Max	Chyby
small_family	100	405	41	3795	0.0 %
medium_family	100	425	122	2265	0.0 %
large_family	100	1781	593	4484	0.0 %
TOTAL	300	870	41	4484	0.0 %

Z výsledků vyplývá, že rozdíl hodnot průměrného času vzorků mezi *small_family* a *medium_family* je zanedbatelný. U tabulky *large_family*, která obsahuje mnohem více dat, je vidět 4 násobný nárůst v průměrném čase vzorků.

Druhý případ: 100 uživatelů, provedení požadavku právě jednou, tři spuštění JMeter jsou uvedeny v následujících tabulkách:

Štítek	Vzorky	Průměr	Min	Max	Chyby
small_family	100	5152	1369	12169	4.0 %
medium_family	100	4535	164	10061	6.0 %
large_family	100	7620	1387	11797	6.0 %
TOTAL	300	5769	164	12169	5.0 %

Štítek	Vzorky	Průměr	Min	Max	Chyby
small_family	100	5709	1551	12046	3.0 %
medium_family	100	4647	149	10347	8.0 %
large_family	100	6751	1412	10016	5.0 %
TOTAL	300	5702	149	12046	5.3 %

Štítek	Vzorky	Průměr	Min	Max	Chyby
small_family	100	4355	1413	11658	3.0 %
medium_family	100	5192	117	10440	9.0 %
large_family	100	7390	1315	10675	2.0 %
TOTAL	300	5646	117	11658	4.6 %

V druhém scénaři je vidět velký nárůst hodnot průměrného času vzorků, který způsobuje desetinásobný počet klientů. S větším množstvím klientů se zvýšila i chybovost v celkovém množství požadavků vyjádřených v procentech.

Kapitola 6

Závěr

Cílem této práce bylo navrhnout JBoss Teiid konektor pro databázi Apache Cassandra. Implementace konektoru splňuje požadavky architektury konektorů nástroje Teiid a zdrojový kód splňuje standardní formát specifický pro Teiid. Pro komunikaci s databází Cassandra byl vybrán klient *DataStax Java Driver*, který k databázi přistupuje skrze rozhraní CQL. Celý vývoj probíhal v prostředí Eclipse.

Funkcionalitu konektoru ověřuje jednoduchý test, který spouští čtecí dotazy na konektor při reálném nasazení na aplikačním serveru a srovnává výstup konektoru. Použitelnost konektoru pod zátěží testuje další test pomocí nástroje JMeter, který ukládá výstupní data z testu do tabulek.

Dalším cílem práce bylo přijetí konektoru komunitou JBoss a zařazení konektoru do *Open Source* projektu JBoss Teiid, aby mohl být libovolně používán a upravován ostatními uživateli. Po komunikaci s hlavními vývojáři projektu byl autor schválen pomocí CLA[1] a zdrojový kód byl označen licencí GPL2[2]. Konektor byl po přijetí komunitou vystaven různým testům. Byla přidána implementace pro operace *INSERT* a *UPDATE*, původní zadání konektoru bylo specifikováno pouze pro čtecí dotazy.

Možnost rozšíření konektoru souvisí zejména s datovým modelem databáze Cassandra. V práci je několikrát rozebírána odlišnost databáze od relačních databází. Tento fakt způsobuje omezení pro některé relační operace, které nelze provést v kontextu NoSQL databází. Jedno z možných rozšíření konektoru je ošetření sekundárních indexů[3] v databázi Cassandra.

---

1. `<http://en.wikipedia.org/wiki/Contributor_License_Agreement>`
2. `<http://en.wikipedia.org/wiki/GNU_General_Public_License>`
3. `<http://wiki.apache.org/cassandra/SecondaryIndexes>`

Literatura

[1] Apache: Cassandra. Dostupné na: <http://wiki.apache.org/cassandra/DataModel>, [Online], 2009 [cit. 2012-11-11].

[2] Apache: JMeeter. Dostupné na: <http://jmeter.apache.org/>, [Online], 2009 [cit. 2013-12-15].

[3] Apache: Maven. Dostupné na: <http://maven.apache.org/what-is-maven.html>, [Online], 2013 [cit. 2013-12-13].

[4] CAPRIOLO, E.: *Cassandra high performance Cookbook*. Olton, Birmingham, U.K.: Packt Pub, 2011, ISBN 9781849515139.

[5] Datastax: Apache Cassandra 1.1 Documentation. Dostupné na: <http://www.datastax.com/doc-source/pdf/cassandra11.pdf>, [Online], 2012 [cit. 2012-11-11].

[6] Datastax: Datastax Java driver. Dostupné na: <http://www.datastax.com/documentation/developer/java-driver/1.0/webhelp/index.html>, [Online], 2013 [cit. 2013-12-14].

[7] HEROUT, P.: *Ucebnice jazyka Java*. Ceské Budejovice: Kopp nakladatelství, 2011, ISBN ISBN 9788072323982.

[8] Ipedo: Guide to enterprise information integration (EII). Dostupné na: <http://www.ipedo.com/downloads/Ipedo_EII_Guide.pdf>, [Online], 2013 [cit. 2013-10-29].

[9] JBoss: JBoss AS Documentation. Dostupné na: <https://docs.jboss.org/author/display/JBASDOC/Home>, [Online], 2013 [cit. 2013-10-25].

[10] JBoss: JBoss Teiid Documentation. Dostupné na: <https://docs.jboss.org/author/display/teiid85final/Home>, [Online], 2013 [cit. 2013-10-25].

[11] JBoss: JBoss Teiid Main Page. Dostupné na: <http://www.jboss.org/teiid/>, [Online], 2013 [cit. 2013-10-25].

[12] LAKSHMAN, A.; MALIK, P.: Cassandra – A decentralized structured storage system. Dostupné na: <http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf>, [Online], 2009 [cit. 2012-11-11].

[13] Oracle: JCA Documentation. Dostupné na: <http://www.oracle.com/technetwork/java/javaee/index-138715.html>, [Online], 2013 [cit. 2013-10-25].

[14] SHETH, A. P.; LARSON, J. A.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. Dostupné na: <http://csis.pace.edu/~marchese/CS865/Papers/p183-sheth.pdf>, [Online], 2013 [cit. 2013-11-22].

[15] Srirangan: *Apache Maven 3 Cookbook*. Birmingham: Packt Open Source, 2011, ISBN 9781849512459.

Příloha A

Obsah přiloženého CD

Obsah přiloženého CD:

- Adresář *text_prace*, který obsahuje:
  - zdrojový kód této práce
  - obrázky použité v této práci
  - UML diagramy
  - tabulky z testů exportované do souborů *.csv*
  - text práce ve formátu PDF
- Adresář *zdrojove_kody*
  - zdrojový kód Teiid 8.3.0 rozšířený o zdrojový kód konektoru pro databázi Apache Cassandra