Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Implementace generátoru datových pohledů pro vytváření sestav ve webové aplikaci DCIx

Plzeň 2018 Petr Dobřička

Místo této strany bude zadání práce.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 16. května 2018

Petr Dobřička

Abstract

This thesis deals with the design and implementation of a tool that allows the DCIx user to create a new data view. It is intended for users with knowledge of data in DCIx, but not knowledge of SQL. We researched six existing tools and decided to implement our own prototype. We are discussing the options and generation of the SQL SELECT query and analyze the problematics of joining entities. This is used in the prototype together with the newly defined metadata that represents the database structure of DCIx. Metadata with weight is used to help the user to join entities. We also use statistics and information to help the user. We describe what statistics and information are involved and how to obtain them. We tested the resulting prototype on beforehand prepared user scenarios.

Abstrakt

Tato práce se zabývá návrhem a implementací nástroje, který umožňuje uživateli aplikace DCIx tvorbu nového datového pohledu. Je určen pro uživatele se znalostí dat v DCIx, ale neznalostí jazyka SQL. Prozkoumali jsme šest existujících nástrojů a rozhodli se implementovat vlastní prototyp. Zabýváme se zde možnostmi a generováním dotazu SELECT jazyka SQL a rozebíráme problematiku spojování entit. To je využito v prototypu společně s nově definovanými metadaty, která přestavují databázovou strukturu aplikace DCIx. Metadata s váhami slouží k nápovědě uživateli při spojování entit. K nápovědě používáme také přidané statistiky a informace. O jaké statistiky a informace se zde jedná podrobně popisujeme a uvádíme, jak je získat. Výsledný prototyp jsme otestovali na předem připravených uživatelských scénářích.

Poděkování

Děkuji Ing. Janu Brnkovi a Ing. Martinu Zímovi, Ph.D. za cenné rady a čas, který mi věnovali.

Obsah

1 Úvod

Webová aplikace DCIx[1] je produkt, který se zaměřuje na dvě oblasti. První oblast je řízení logistiky v distribučních a logistických společnostech. Druhá je řízení logistiky a výroby ve výrobních společnostech. Tato aplikace obsahuje stovky tabulek[1] a v současné době má uživatel možnost získávat data pouze z definovaných pohledů. To ale nestačí uživateli, který má již základní představu o tom, jaká v systému používá data. Takový uživatel chce mít možnost si sám vytvořit svoji datovou sestavu. Sestavou je ve webové aplikaci DCIx myšlena obrazovka obsahující tabulku s daty, se kterou lze dále pracovat. Obvykle je založena na databázovém pohledu. Aplikace DCIx pracuje s MS SQL Server databází. Uživatel ale většinou neumí jazyk SQL[39], aby si tento pohled mohl vytvořit. Proto je potřeba najít nebo vytvořit nástroj, který mu to umožní. Abychom mohli najít nebo vytvořit nástroj, jsou zde přesně specifikované požadavky, které na něj klademe.

Na těchto požadavcích je postavena rešerše nástrojů Pentaho, Spago BI, Dremio, JasperReports, Kibana a DBxtra. Na základě rešerše jsme se rozhodli implementovat vlastní prototyp generátoru dotazů. Naším cílem bude splnit požadavky z horní poloviny jejich seznamu jiným způsobem než existující nástroje. Od toho se odvíjejí další části této práce.

Cílíme hlavně na statistiky a informace nad daty, které by mohly být přínosem pro orientaci uživatele v datech DCIx, a jejich vhodné zobrazení. Dále je potřeba splnit požadavek, který říká, že uživatel si skládá svůj vlastní pohled. Protože obvykle uživatel bude pracovat s daty z více než jedné tabulky, bude potřeba vyřešit problematiku spojování tabulek. V databázi DCIx je mezi tabulkami mnoho vazeb. Dalších vazby jsou z databáze odstraněny kvůli archivaci, ale logická vazba mezi tabulkami přetrvává. Zde vyvstává nutnost použití metadat k definici správné logické databázové struktury. Protože může být více spojení mezi dvěma tabulkami, chceme uživateli nabídnout tu nejlepší variantu z pohledu logiky DCIx. Proto se v této práci také zaměříme na tvorbu pohledů v jazyce SQL a na využití metadat. Zmíněné cíle chceme implementovat v prototypu. Jeho funkčnost nakonec ověříme několika jednoduchými scénáři.

[1]Záloha databáze, kterou jsme měli v této práci k dispozici, obsahovala 429 uživatelských tabulek.

2 Aplikace DCIx

Jak bylo zmíněno v úvodu, DCIx je produkt, který se zaměřuje na dvě oblasti. Nejprve popíšeme oblast řízení logistiky v distribučních a logistických společnostech. Systém pro řízení logistiky podporující služby s přidanou hodnotou do sebe integruje několik oblastí. Základem je řízení interních logistických procesů (WMS). To je rozšířeno o řízení kvality (QMS), propojení s dodavateli, odběrateli a klienty přes portálové rozhraní (Portal), řízení sekvenčních dodávek (JIS), plánování a řízení nakládky a vykládky (YMS) až po plánování a řízení vybraných interních procesů prostřednictvím plánovací tabule (PPS).

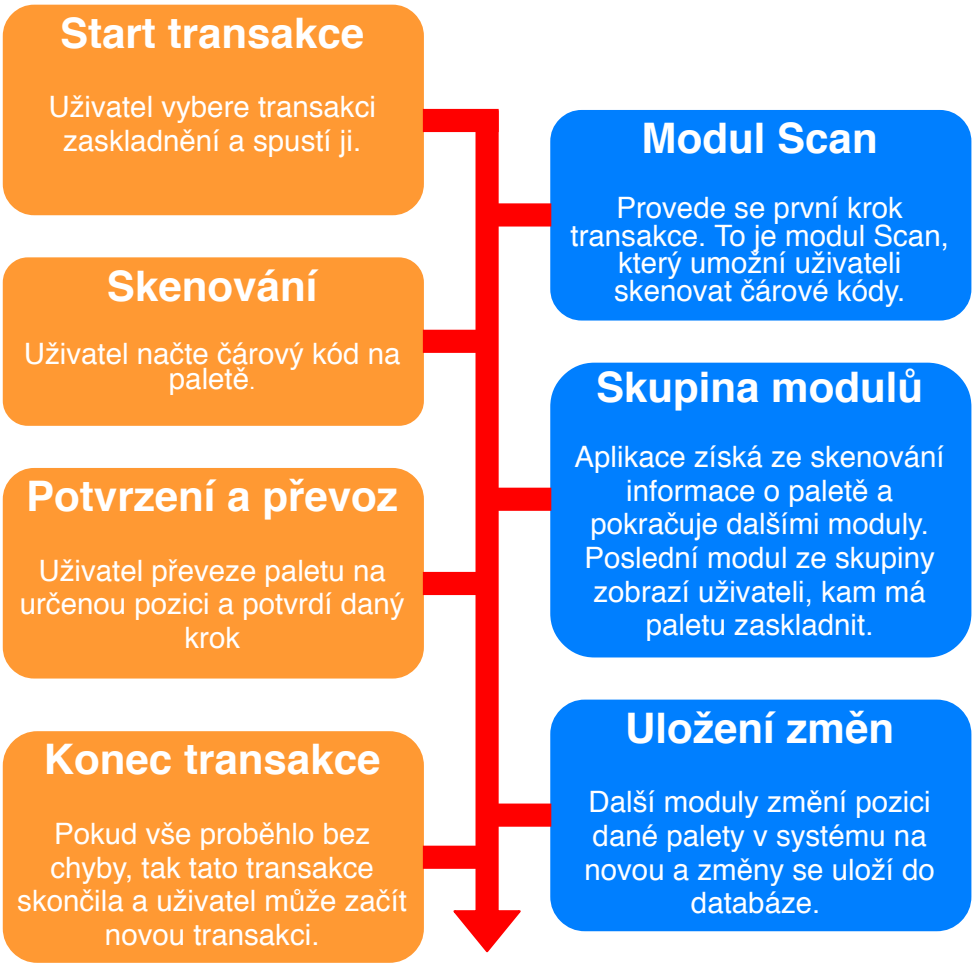
Druhou zmíněnou oblastí je řízení logistiky a výroby ve výrobních společnostech. To představuje systém z kategorie Manufacturing Operations Management (MOM), který nejužším způsobem integruje firemní procesy ve výrobě a logistice a odpovídá myšlence Industry 4.0. Je zde využito WMS, řízení a sběr dat z výroby (MES), QMS, pokročilé plánování výroby (APS) a řízení expedice (JIT/JIS).

DCIx integruje celý dodavatelsko-odběratelský řetězec, usnadňuje koordinaci a obchodní spolupráci se zákazníky, dodavateli a partnery prostřednictvím technologií EDI, WebEDI, VMI. Doplňuje zákaznické systémy o funkce, které jejich systém nepodporuje nebo umožňuje částečně či komplikovaně. Při implementaci je kladen důraz na pružnost a variabilitu systému, jež může být nasazen buď v plném rozsahu, nebo jen jako jeho dílčí části s postupným pokrýváním dalších procesů a oblastí.

V DCIx je obvykle jeden proces reprezentován transakcí. Transakce je posloupnost dílčích kroků. Tyto dílčí kroky představují jednotlivé nastavitelné moduly DCIx, kterých jsou již řádově stovky. Ty zajišťují značnou flexibilitu procesů. Pro představu na Obrázku 2.1. je podrobně zobrazeno, jak by mohl vypadat proces zaskladnění.

2.1 Procesy

V této části jsou podrobně popsány různé oblasti aplikace DCIx a u každé oblasti se nachází seznam pro ni typických procesů.



Obrázek 2.1: Proces zaskladnění

2.1.1 DCIxWMS

DCIxWMS (Warehouse Management System) [32] rozšiřuje funkcionalitu podnikových informačních systémů pro detailní řízení logistických procesů ve skladech logistických a distribučních center, nebo ve skladech a výrobě u výrobních společností. Umožňuje řízení procesů od příjmu až po expedici a přináší spolehlivý a přesný zdroj informací v reálném čase s využitím automatické identifikace manipulačních jednotek a skladových míst. To mu umožňují čárové nebo RFID kódy. Podporuje technologie Pick by Light, Pick by Voice, rozšířenou realitu a umožňuje integraci s technologiemi vážních systémů, dopravníky a poloautomatickými systémovými vozíky nebo plně automatickými regálovými zakladači. Zároveň umožňuje řízení skladových operací, efektivnější využití stávajícího skladového prostoru nebo posílení produktivity pracovníků.

Procesy typické pro tuto oblast jsou například vstupní a výstupní kontrola kvality materiálu a jeho zaskladnění. Dále jsou to příjem na sklad, přebalení, expedice nebo kontroly úplnosti dodávky vážením a expirace.Také sem patří doplňování vychystávacích míst a doplňování do výroby, nebo konsolidace zákaznických objednávek podle příkazu k vychystání. Kromě zmíněných jsou to procesy integrace přepravců práce s vratkami, evidence vratných obalů, inventury a další.

2.1.2 DCIxMES

DCIxMES (Manufacturing Execution System) [28] umožňuje kontrolu nad výrobními procesy. Zobrazuje detailní přehled o aktuálním stavu zakázek ve výrobě. Obsahuje výkazy práce o činnosti pracovníků a výrobních zařízení. Také lze kontrolovat dodržování technologických postupů. Data sbírá přímo z výrobních zařízení, montážních linek a od zaměstnanců v reálném čase. Tato data integruje do jednoho zdroje. Z něho se následně berou data pro operativní řízení výroby, plánování výroby a nákupu, zvyšování produktivity a zpřesňování norem. Umožňuje automatické odepisování vyrobených kusů a generování požadavků na objednání materiálu pro zajištění plynulého chodu výroby. Obsahuje reporty, které poskytují ucelený přehled o celkové produktivitě výroby, o prostojích, vytížení techniky nebo řádně vyrobených i vadných kusech. S využitím těchto reportů lze odhalit slabá místa v procesu výroby a následně je eliminovat.

Typické procesy pro tuto oblast jsou řízení pracoviště, komunikace se stroji, sledování výrobního zařízení nebo sledování operátora. Kromě toho sem patří i řízení procesu výroby podle technologického postupu, reportování, údržba a další.

2.1.3 DCIxQMS

DCIxQMS (Quality Management System) [31] je systém pro řízení kvality. Slouží k řízení vstupní a výstupní kontroly, monitoringu a kontroly dodržování stanovených pravidel nastavených oddělením kvality. Při vstupní kontrole je možné rozdělit dodavatele a materiálové položky do skupin. K tomu slouží pravidla, podle kterých je kontrola prováděna. Ta může být statická, nebo dynamická. Ze získaných dat lze vyhodnotit a řídit spolehlivosti dodavatelů. Evidence sledovaných hodnot je vedena v podobě dokumentů kvality. Dále slouží ke kontrole kvality rozpracované výroby a výstupní kontrole. Výsledná data jsou získávána pomocí skenerů, tabletů nebo pevných stanic.

Pro tuto oblast jsou typickým procesem nejrůznější typy definic. Mezi

ně patří definice typů a hladin kontrol, definice dynamického řízení četnosti kontroly nebo definice dokumentů kvality s konkrétními parametry kontroly. Další jsou definice kontrolního plánu pro jednotlivé díly, skupiny dílů a dodavatele nebo definice vzorkovacího plánu. Kromě definic jsou zde procesy zadržení a uvolnění kvalitou, evidence výsledku kontroly podle kontrolního plánu nebo změny hladin kontroly. Také sem patří procesy rozdělení dodavatelů do různých kategorií podle kvality dodávek, sortace a další.

2.1.4 DCIxPortal

DCIxPortal [29] je webový portál. Slouží k propojení dodavatelů, odběratelů a logistických poskytovatelů a umožňuje jim sdílet aktuální a přesné informace v reálem čase. Obsahuje ucelený a jednotný přehled o aktuálně schválených objednávkách, odvolávkách nebo kanbanech. Také obsahuje informace o stavech skladů a pohybu zásob. Dodavateli umožňuje tisknout elektronické dodací listy a unikátní etikety podle požadavků konkrétního odběratele. Tím usnadňuje identifikaci a příjem dodávek u odběratele. Odběratel pouze porovná skutečný stav pomocí skenování s předem přijatým elektronickým dodacím listem. To usnadňuje a zefektivňuje spolupráci dodavatelem a odběratelem.

Zde jsou typické procesy pracující s objednávkami a to vytvoření objednávky,odvolávky nebo kanbanu na základě plánování a upozornění dodavatele na objednávku, potvrzení objednávky dodavatelem nebo návrh dodávky ze strany dodavatele. Dále sem patří potvrzení nebo tvorba protinávrhu odběratelem, tisk zákaznických etiket a dodacího listu, přehled o připravované dodávce nebo rozdíl mezi objednaným a expedovaným zbožím. Kromě toho sem patří i automatická tvorba a zaslání elektronického dodacího listu odběrateli, příjem materiálu na základě elektronického dodacího listu, kontrola skenováním nebo hodnocení dodavatelů na základě přesnosti dodávek a další.

2.1.5 DCIxYMS

DCIxYMS (Yard Management System) [33] slouží k plánování časů nakládky a vykládky jednotlivých kamionů pro konkrétní brány skladů. Proto se jedná o grafickou plánovací tabuli přístupnou přes internet, která umožňuje dodavatelům, zákazníkům nebo dopravcům sdílet informace o vytížení bran a vytvořit rezervaci pro příjezd kamionu. Rezervace jsou automaticky validovány a zkontrolovány. Díky poskytovaným informacím je možné přesněji plánovat vlastní zdroje, například kolik ramp je potřeba mít otevřeno, kolik operátorů, nebo která manipulační technika bude v daný čas zapotřebí.

Obsahuje také informace o stavu nakládky nebo vykládky v reálném čase. Spolu s DCIxWMS optimalizuje proces příjmu materiálu a expedice.

Typické procesy pro YMS jsou práce s rezervacemi a to založení, potvrzení nebo změna. Další procesy jsou příjezd vozidla, začátek nakládky nebo vykládky a ukončení nakládky nebo vykládky. Také to jsou procesy mimořádné změny rampy, mimořádné změny pracovní doby a směru rampy, tvorba periodických rezervací, záznam historie nakládek a vykládek, reporting a další.

2.1.6 DCIxJIT/JIS

DCIxJIT/JIS [27] umožňuje automatické řízení dodávek Just-In-Time nebo Just-In-Sequence. To znamená zajistit přesnou expedici včetně označení výrobků, průvodní dokumentaci a elektronické zprávy podle požadavků zákazníka. Je integrován s EDI systémem pro výměnu elektronických zpráv. Kontrola se provádí pomocí skenerů, které snímají identifikační kódy. JIT dodávky se připravují podle plánu dodávek, který se připravuje na základě kumulativních čísel. Také je důležité zabezpečit vychystávání finálních výrobků podle FIFO. JIS dodávky je důležité správně seřadit jak na manipulační jednotce, tak i na nákladním autě.

Pro oblast JIT jsou typické procesy jako automatické zpracování odvolávek, tisk zákaznických etiket a průvodních dokumentů, podpora standardů a konceptů (RAN, Manifesty, VDA 4939 a jiné), podpora konsignačních skladů a další.

Pro oblast JIS jsou to procesy jako podpora konceptů Assembly to Sequence, Ship to Sequence, Perlenkette, tisk sekvenčních výlepů pro jednotlivé dodávané díly nebo moduly a distribuce etiket na montážní linky nebo expediční místa podle stanovených pravidel. Kromě toho sem patří tisk souhrnných etiket na speciální sekvenční kontejnery, signalizace zpoždění oproti plánovanému času vychystání, ověřování dodržení sekvence s použitím čárových kódů a další.

2.1.7 DCIxPPS

DCIxPPS (Production Planning System) [30] je grafická plánovací tabule. Zobrazuje výrobní zakázky alokované v čase na jednotlivé výrobní stroje, linky a pracoviště, stav jejich plnění a vytížení zdrojů výroby. Dále ukazuje importované výrobní zakázky vytvořené na základě plánování potřeby materiálu nebo je umožňuje zadávat ručně. Podporuje technologii Drag and Drop pro přesouvání zakázek mezi zdroji. Také umožňuje odstranění ča-

sových mezer a řetězení operací v rámci dané výrobní zakázky. Pokud je propojeno s MES, lze zobrazit aktuální plnění plánu. Z těchto dat počítá aktuální odhadovanou délku zakázky.

Nakonec procesy typické pro plánování jsou grafická reprezentace plnění plánu, přesun zakázek v čase nebo přesun zakázek mezi zdroji. Dále sem patří procesy jako změna množství, kontrola naplánovaných zakázek po termínu, zřetězení zakázek nebo odstranění mezer v plánu. Kromě těchto to jsou to i procesy definice výrobního kalendáře pro každý stroj, ruční vložení urgentní výrobní zakázky, tisk naplánovaných výrobních zakázek a další.

2.2 Datová vrstva

2.2.1 Systém transakcí

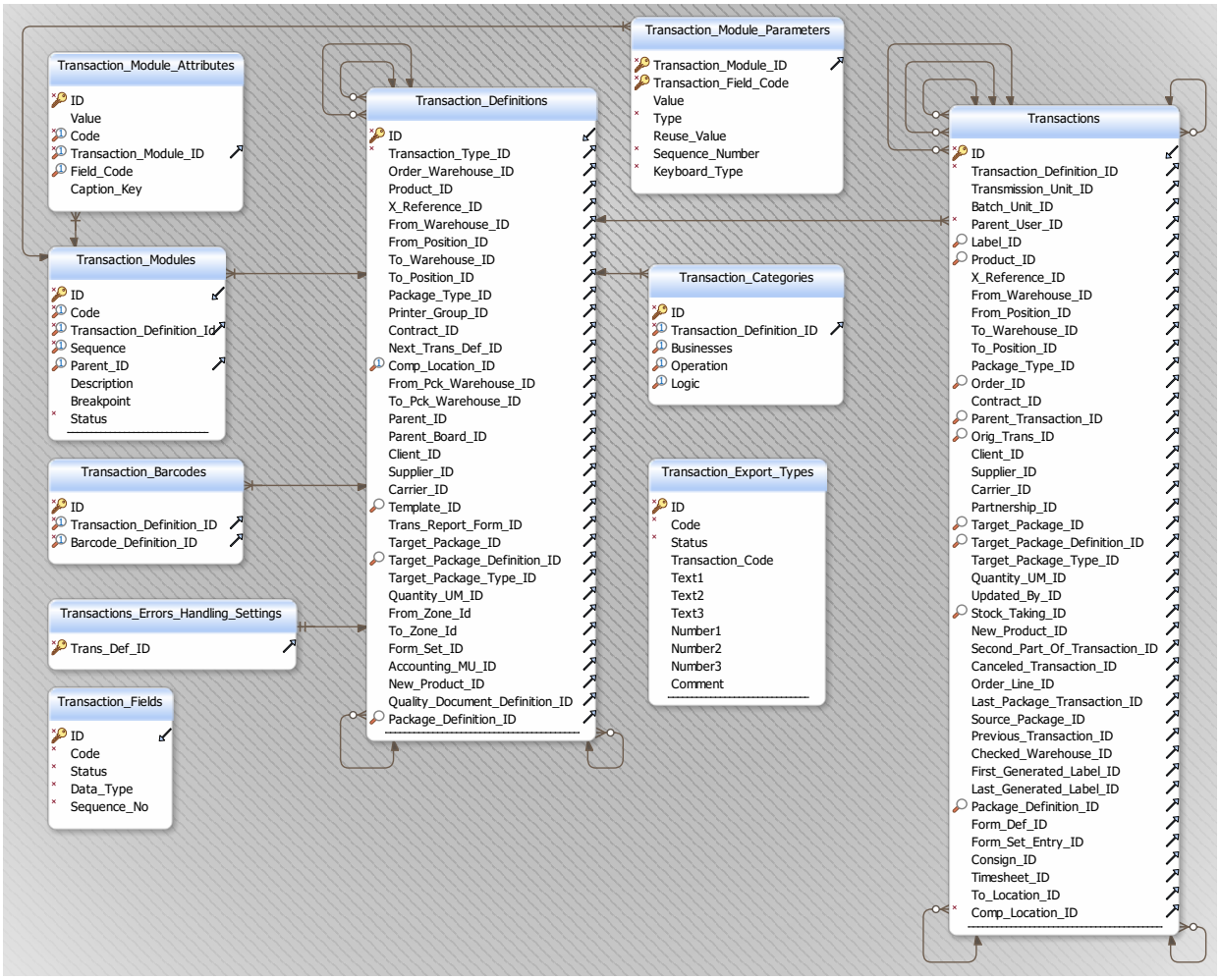
V DCIx jsou transakce nejdůležitější částí. Transakční definice představují předpis pro zákaznické procesy. Samotné transakce poté jednotlivé průběhy daného procesu. Transakční definice se skládá ze souboru modulů, které jsou na sebe navázané v daném pořadí, existují i moduly pro skoky. Každý modul má určitou specifickou funkci a svoji sadu atributů a parametrů. Atributy představují nastavení daného modulu, například na jaký stav se bude příkaz měnit. Parametry představují vstupní hodnoty z probíhající transakce, například číslo daného příkazu. Na Obrázku 2.2. je zobrazen diagram týkajících se transakcí.

2.2.2 Univerzální příkazy

Další oblastí v datech DCIx představují univerzální příkazy. Opět je zde definice příkazu, která říká, jak daný příkaz má vypadat, jaké sloupce a informace v něm uživatel chce zobrazovat. Příkaz má obvykle hlavičku a poté jednotlivé řádky. Samotné příkazy, které splňují dané definice, už mohou obsahovat kmenová data zákazníka. Například u příkazu k příjmu je dobré vědět, jaký výrobek a v jakém množství chce zákazník přijmout. Na Obrázku 2.3. je zobrazen diagram týkajících se univerzálních příkazů.

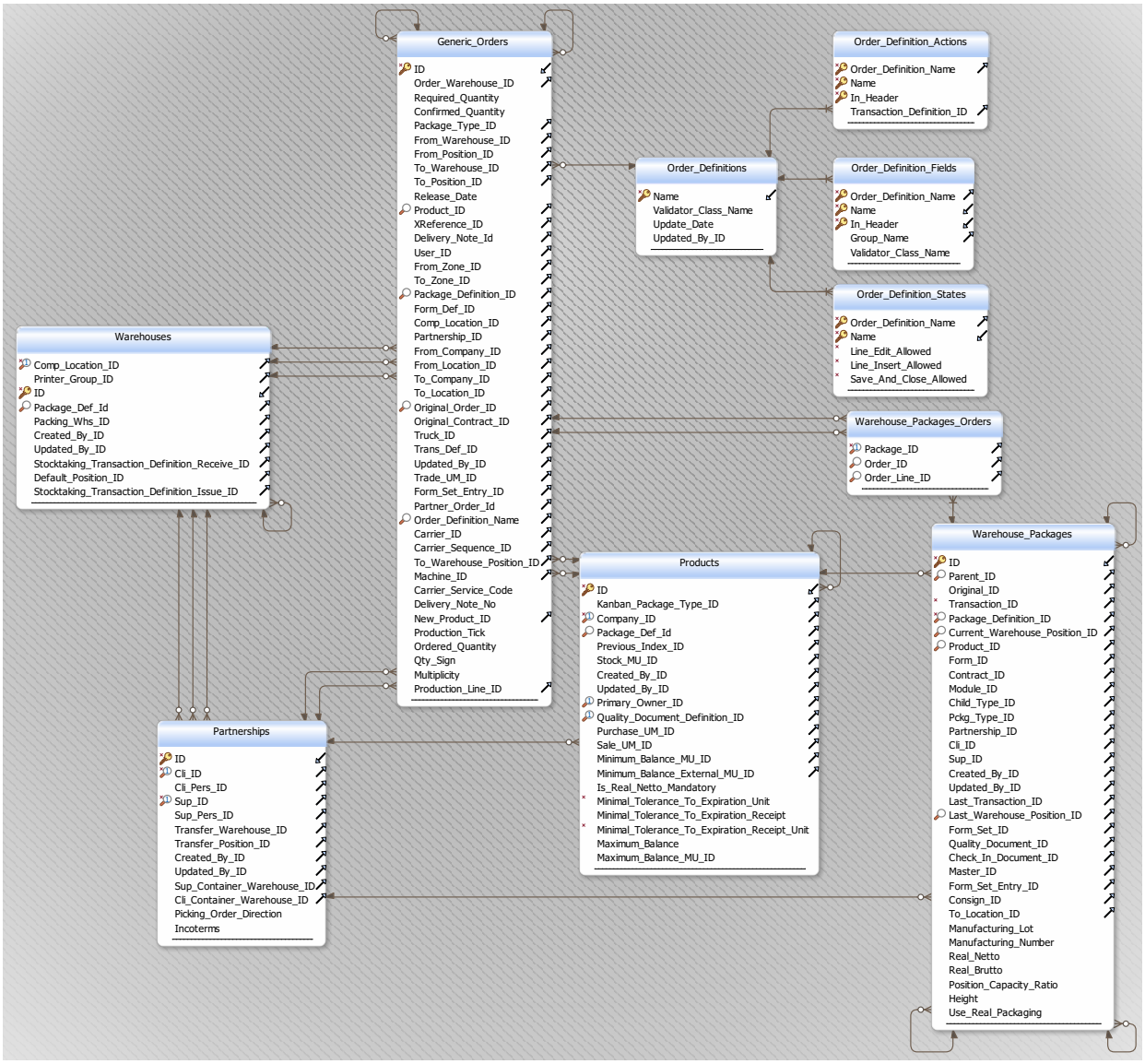
2.2.3 Kmenová data

Další důležitou částí jsou kmenová data zákazníka. To jsou data, která si definuje zákazník. Patří sem například produkty, sklady, skladové pozice, dodavatelé a zákazníci, typy a definice balení, definice čárových kódů a další.

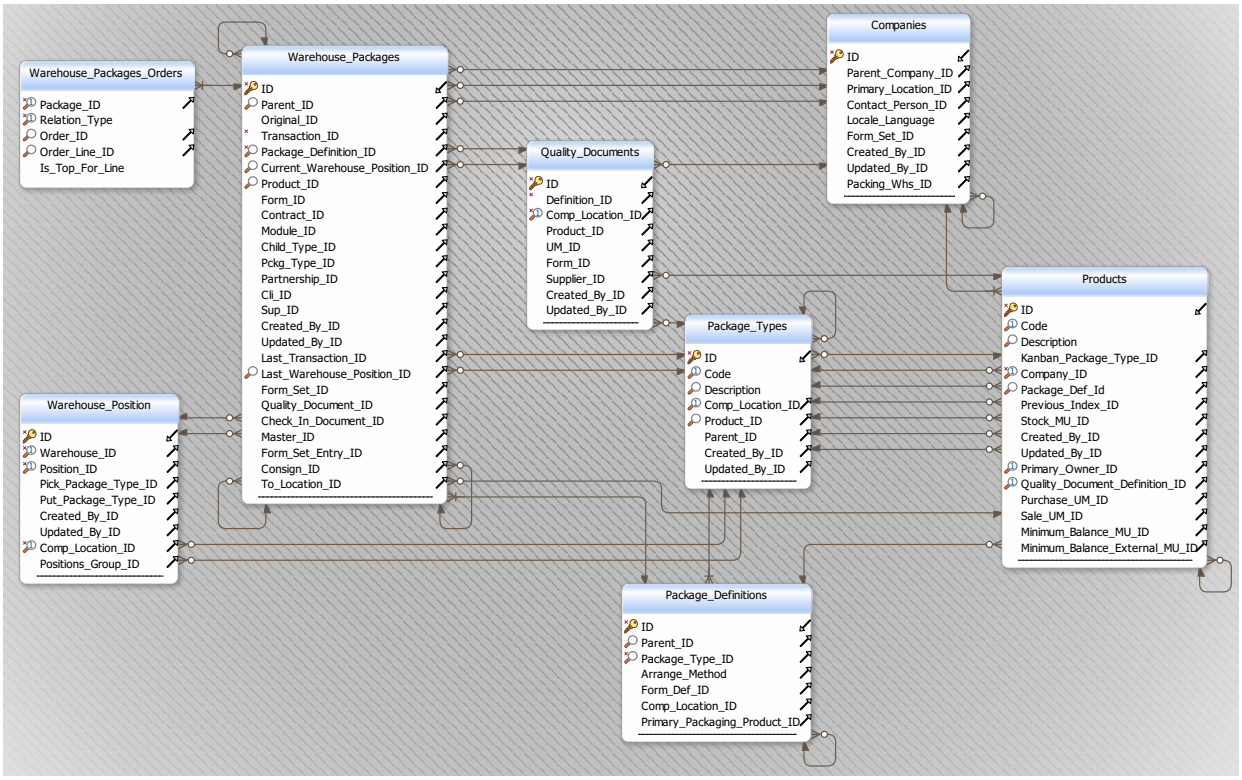


Obrázek 2.2: Transakce

Na Obrázku 2.4. je zobrazen diagram týkajících se balení ve skladu. Je v něm vidět napojení na kmenová data.



Obrázek 2.3: Univerzální příkaz



Obrázek 2.4: Balení ve skladu

3 Rešerše existujících nástrojů

Jak bylo zmíněno v úvodu, zákazníci DCIx již mají představu o datech v systému a nyní potřebují nástroj, který jim umožní se na tato data dívat z různých úhlů. Zde se zmíněný nástroj pokusíme najít. Kapitola obsahuje definici požadavků, srovnání již existujících nástrojů a jejich hodnocení. V hodnocení je i rozhodnutí, jaký ze zkoumaných nástrojů vybrat, nebo zda implementovat vlastní prototyp.

3.1 Požadavky na nástroj

V této části jsou definované požadavky na nový reportovací nástroj pro aplikaci DCIx. Z těchto deseti požadavků se vychází při zkoumání existujících nástrojů a při implementaci nástroje.

3.1.1 Požadavky na nástroj seřazeny dle priority

Požadavek 1: Statistické informace nad tabulkou a sloupci. To zahrnuje například počet záznamů v tabulce, průměr hodnot ve sloupci a podobně.

Požadavek 2: Dokumentace tabulek, sloupců a vazeb mezi daty. Určitá data v systému spolu souvisejí, ale nemají klasický vztah primární-cizí klíč (kvůli archivaci). Proto je třeba umožnit definici jejich vazeb pomocí metadat.

Požadavek 3: Full-textové vyhledávání nad názvy tabulek a sloupců.

Požadavek 4: Možnost určení podmnožiny dat. To platí pro tabulky i pro sloupce, protože není vhodné uživateli ukazovat vše v systému.

Požadavek 5: Spojování sloupců a tabulek. Uživatel bude moci libovolně spojovat sloupce, případně tabulky do svých pohledů.

Požadavek 6: Zobrazit náhled dat, pokud uživatel spojuje tabulky a sloupce. Pokud uživatel začne skládat svůj pohled, je dobré mu ukazovat několik řádek z výsledku.

Požadavek 7: Ukládání šablon pohledů a jejich možné znovu použití. Uživatel bude moci si vytvořený pohled uložit jako šablonu a používat ho opakovaně nad novými daty.

Požadavek 8: Nabízet datové rozhraní pro další části systému. Pokud by bylo potřeba daný pohled použít dále jako datový zdroj.

Požadavek 9: Zobrazení a export dat ve více formátech. Nyní je nutné pouze HTML.

Požadavek 10: Možnost spojení více zdrojů dat. Je potřeba mít možnost pracovat jak s daty DCIx, tak i s daty externích systémů.

3.2 Jednotlivé nástroje

V této sekci jsou podrobně popsány zkoumané nástroje. Jedná se o nástroje Pentaho, Spago BI, Dremio, JasperReports, Kibana a DBxtra. Zkoumání probíhalo jak čtením dokumentace a manuálů, tak i instalací a vyzkoušením volně dostupných softwarových verzí a demoverzí daného nástroje. Proto je kromě literatury, která je uvedena u každého nástroje hned v úvodu jeho popisu, u každého nástroje citována vydaná verze, která byla ozkoušena. Dále je u každého nástroje uvedeno, jak splňuje zadané požadavky.

3.2.1 Pentaho

Pentaho [13–17] má komerční i otevřenou edici. Zde se pojednává o otevřené edici, která se jmenuje Community Edition. Pentaho je složeno z několika nezávislých projektů, které z něj dělají všestranný BI nástroj. Mezi tyto projekty patří BA server, což je webová aplikace, která umožňuje uživateli přístup a práci s jeho BI obsahem. Jeho grafickým rozhraním je další projekt Pentaho User Console(PUC). Aby uživatel mohl pracovat s reporty, je nutné je vytvořit. K tomu slouží Pentaho Report Designer(PRD). K vytváření reportů jsou potřeba data. Ta jsou získávána pomocí Pentaho Data Integration(PDI). Pro tuto práci je také důležitý Pentaho Metadata Editor(PME). Ten slouží k tvorbě obchodních modelů nad databázovou vrstvou.

Kromě toho Pentaho nabízí další projekty, které již pro nás nejsou důležité. Ty se týkají data miningu a pokročilé analýzy. Jsou dva možné přístupy získání datasetu pro tvorbu reportu. První je využití přímého připojení do databáze v PRD. Data se poté připravují pomocí SQL Query Designeru (QD). Druhý způsob je vytvoření obchodního modelu v PME. V PRD se

následně použije tento model a pomocí Query Builderu (QB) se z něj vytvoří požadovaný dataset. Jakmile je dataset vytvořen v PRD se definuje přesný vzhled reportu. Ve chvíli kdy zvolím obchodní model, již není možno ovlivnit, jak jsou data pospojována. To lze pouze v PME. V QB se už nedají použít obchodní tabulky, ale pouze jen obchodní pohledy. Pokud by pohledy byly totožné s tabulkami, nelze již upravit dotaz, aby vynutil určité spojení. Řešením je vytvořit nový obchodní model přesně na míru.

Požadavek 1: Database explorer nabízí možnost zobrazit počet záznamů v tabulce.

Požadavek 2: PME umožňuje tvorbu obchodních modelů. V těchto modelech si lze nadefinovat libovolné vztahy.

Požadavek 3: Nelze prohledávat sloupce ani tabulky při tvorbě dotazu v QD.

Požadavek 4: V PME lze vybrat tabulky i sloupce při definování fyzických tabulek.

Požadavek 5: PRD obsahuje grafický QD. Funguje jako Drag&Drop nebo textový editor.

Požadavek 6: Náhled vidět není, ale lze si ho otevřít pomocí tlačítka. Předem je možné zvolit, kolik má obsahovat maximálně řádků.

Požadavek 7: V PRD lze ukládat a upravovat uložené šablony. Tyto šablony je možné plnit aktuálními daty.

Požadavek 8: V PDI lze definovat libovolnou transformaci, kterou lze poté exportovat do tabulky v databázi.

Požadavek 9: Data lze exportovat do následujících formátů: PDF, HTML, XLS, CSV, RTF a TXT.

Požadavek 10: V PDI lze spojovat přes dvacet různých druhů zdrojů dat. Mezi důležité patří MS SQL Server, CSV, JSON, XML a XLS.

3.2.2 SpagoBI

SpagoBI [23, 35] je další z řady BI nástrojů. Jedná se o webovou aplikaci, která nabízí funkčnost typickou pro tento druh nástrojů. Umožňuje vytvořit a popsat obchodní model databáze. Model se ale vytváří mimo aplikaci a jeho tvorba je složitá. Z těchto modelů si uživatel následně skládá svůj osobní dataset. Ze složeného datasetu může poté tvořit reporty. Pokud v modelu

existuje více cest mezi dvěma zvolenými entitami, je možné si ručně vybrat požadovanou trasu. Dále je zde možnost zvolit si aliasy. Nástroj podporuje také agregační funkce, filtry a možnost přidání polí s dopočítávanými hodnotami.Také si lze nechat ukázat SQL dotaz, který v nástroji tvorbou datasetu vzniká. Kromě tvorby ad-hoc reportů nabízí takzvaný *cockpit*, což je možnost poskládat si několik widgetů na jednu obrazovku. Dále podporuje tvorbu datových mash-upů.

Požadavek 1: Neobsahuje žádné statistiky týkající se procházených dat.

Požadavek 2: Jsou zde dvě skupiny metadat. Obchodní model a metadata, která fungují na principu klíč a hodnota.

Požadavek 3: Full-textové vyhledávání nad názvy tabulek a sloupců se zde vůbec nenachází. Ve chvíli, kdy se pracuje s rozsáhlým modelem, je to značný nedostatek.

Požadavek 4: Pomocí SpagoBI Qbe Enginu lze vytvořit obchodní model databáze, který lze libovolně upravit a uložit.

Požadavek 5: Je zde omezení modelem. Pokročilé nastavování je obtížné, ale je možné.

Požadavek 6: Zobrazení náhledu si lze vynutit stisknutím tlačítka při tvorbě dotazu nebo reportu.

Požadavek 7: Lze ukládat a následně načítat vytvořené datasety. Také lze ukládat a načítat reporty z těchto datasetů.

Požadavek 8: Datový zdroj nabízet neumí.

Požadavek 9: Data lze exportovat do následujících formátů: PDF, XLS a XLSX. Umí zpracovat a naplnit JasperReport, BIRT a Accessible Report.

Požadavek 10: Lze spojit více datových zdrojů pomocí Merge Queries. Množina podporovaných datových zdrojů: VoltDB, OrientDB, Ingres, MySQL, PostgresSQL, Hive, HBase, DB2, MongoDB, HSQL, Oracle, S3, SQL Server, File, Web Service, REST.

3.2.3 Dremio

Dremio [4, 5] je samoobslužná datová platforma, která uživatelům umožňuje zkoumat, čistit, zrychlit a sdílet data bez ohledu na umístění, objem nebo strukturu. Umožňuje analytikům snadno procházet složitá zdrojová

data se vnořenými strukturami nebo smíšenými typy. Toho docílí vytvořením nového virtuálního data setu. Virtuální data set může sloučit data mezi více systémy. Přináší zrychlení tím, že využívá Apache Arrow, což je sloupcová datová technologie operující v paměti, spolu s jejich vlastní inteligentní cacheovací funkcí Accelerator Dremio. Při povolené akceleraci mohou být dotazy spouštěny interaktivně a to je velká výhoda pro průzkum Big Data.

Dremio funguje jako cluster. Sestává se z jednoho nebo více koordinátorských uzlů, kde jeden slouží jako master. A z jednoho nebo více exekučních uzlů. Koordinátor plánuje dotazy, obsluhuje UI a klientská spojení. Master komunikuje s úložištěm metadat. Exekuční uzly provádí samotné dotazy. Dotazy se snaží provádět pomocí „Pushdowns“. To je nativní vykonání dotazu, proto se zvýší výkon pro filtraci, limit, řazení, agregace a projekce. V Dremiu jsou dva typy datasetů. Fyzické datasety jsou uloženy v datových zdrojích a nemohou být Dremiem změněny. Virtuální datasety se vytváří z fyzických nebo jiných virtuálních. Definují se pomocí transformací, joinů, filtrů a dalšími úpravami. Data nekopírují proto nevyužívají tolik místa. Jsou popsány SQL dotazem, lze je tedy ručně dle libosti modifikovat.

Požadavek 1: Neobsahuje žádné statistiky týkající se procházených dat.

Požadavek 2: Neobsahuje možnost si nastavit metadata ani dokumentaci nad předanou datovou strukturou.

Požadavek 3: Obsahuje case-sensitive vyhledávání nad fyzickými i virtuálními data sety. Nad sloupci pouze na obrazovce, kde se provádí spojování entit.

Požadavek 4: Každý data set lze omezit pouze na dotazování. Lze definovat skupiny s využitím LDAP správy uživatelů. Bez LDAP je správa uživatelů nepoužitelná, protože každý uživatel je administrátor. Jakákoliv jiná správa uživatelů a tudíž i práv pro zákazníka není možná.[1]

Požadavek 5: Pro Dremio nejsou rozhodující vztahy, které jsou v relační databázi. Umožňuje spojení dvou entit přes jakýkoliv atribut. K tomu stačí, aby si jeho hodnoty odpovídaly. Protože však neobsahuje metadata, které by dané atributy popsaly, musí mít uživatel představu, jaké atributy v jakých entitách si odpovídají.

[1]Bez LDAP si uživatelé libovolně mohou mazat pohledy ze sdílených složek a pokud je nový dataset odvozen ze starého pomocí funkce 'uložit jako' a zároveň je smazán původní dataset, nový nelze otevřít, protože se neuloží jako kopie staré, nýbrž jako odkaz na starý, který se dále upravuje (SELECT * FROM starydataset)

Požadavek 6: Náhled je vidět stále při editaci SQL, pouze je nutné ho aktualizovat tlačítkem. Tlačítko má na výběr ze dvou nastavení a to spustit celý dotaz (Run) nebo jen náhled (Preview).

Požadavek 7: Pohledy jsou v Dremiu virtuální datasety. Ty lze ukládat a dále s nimi pracovat. Lze je používat opakovaně, protože pracují nad živými daty.

Požadavek 8: Dremio nabízí svůj ODBC a JDBC driver. Tudíž jeho data jsou dostupná pomocí standardu SQL.

Požadavek 9: Vizualizace a export jen pomocí dalších nástrojů, kterým lze podstrčit spojení (ODBC, JDBC) do Dremia. Například MS Excel, Tableau a další.

Požadavek 10: Lze spojit různé datové zdroje do jednoho virtuálního datasetu. Množina podporovaných datových zdrojů: Elasticsearch, File Upload (XLS, JSON, Parquet, or CSV), HDFS, Hive, MapR-FS, NAS, Oracle, S3, SQL Server, DB2, Redshift, HBase, MongoDB, MySQL, PostgresSQL

3.2.4 JasperReports a JasperReports Server

JasperReports [8, 37] je knihovna pro jazyk Java, která slouží k tvorbě pixel-perfect dokumentů. Tyto dokumenty vznikají ze šablon, které se vytvářejí pomocí desktopové aplikace (Jaspersoft Studio). Uživatel si zvolí datový zdroj, ze kterého bude report plnit daty a strukturu (typický případ jsou sloupce) z něj si namapuje do proměnných. S jejich využitím si poté libovolně skládá svojí šablonu. Ta je v knihovně představena jako třída JasperDesign a lze ji uložit jako soubor typu JRXML. Lze ji vytvořit i ručně v libovolném textovém editoru, protože se pouze jedná o XML s definovaným formátem.

Šablona se následně kompiluje a tím vzniká instance třídy JasperReport. Ta lze uložit jako binární soubor. Nyní je možné tuto šablonu opakovaně plnit aktuálními daty. Data se získávají z rozhraní JRDataSource. Lze je plnit rovnou z databáze nebo si implementovat rozhraní a naplnit ho vlastními daty. Následně se již report exportuje do požadovaného formátu. Existuje i webová aplikace, která tuto knihovnu využívá, JasperServer. Ten umožňuje ad hoc tvorbu reportů a dashboardů.

Požadavek 1: Neobsahuje žádné statistiky týkající se procházených dat.

Požadavek 2: Neobsahuje možnost předdefinovat vztahy mezi daty.

Požadavek 3: Neobsahuje vyhledávání tabulek ani sloupců.

Požadavek 4: Neobsahuje možnost určit si podmnožinu dat, připojuje se přímo k databázi.

Požadavek 5: Lze tvořit libovolné spoje jako při psaní dotazu v SQL, nemusí to být pak logicky správné.

Požadavek 6: Zobrazení náhledu si lze vynutit stisknutím tlačítka při tvorbě dotazu.

Požadavek 7: Lze ukládat jak šablony, tak i samotné zkompilované reporty. Pro opětovné editování je lepší pracovat se šablonami.

Požadavek 8: Neumožňuje nabízet data dále.

Požadavek 9: Data lze exportovat do následujících formátů: HTML, PDF, RTF, ODT, ODS, TXT, PPTx, DOCx, JSON, CSV, XLS a XML.

Požadavek 10: Je možné teoreticky spojit libovolné datové zdroje implementací rozhraní JRDataSource, ale není to triviální.

3.2.5 Kibana

Kibana [9, 10] je analytická a vizualizační platforma určená pro práci s ElasticSearch. Jedná se open source nástroj, který umožňuje vyhledávat a zobrazovat data uložená v ElasticSearch indexech. Umí je vizualizovat pomocí různých map, grafů a tabulek. Pomáhá při pokročilých analýzách a analýzách velkého objemu dat. Její webové rozhraní nabízí možnost rychle vytvářet a sdílet dynamické dashboardy, které zobrazují změny v dotazech ElasticSearch v reálném čase. Aby bylo možné exportovat vizualizovaná data, je potřeba mít nainstalovaný X-Pack plugin. Ten navíc nabízí i jinou funkcionalitu. Mezi tu patří například monitoring, práce s grafy nebo strojové učení.

Požadavek 1: Obsahuje funkci Timelion, která je schopna počítat statistiky vzhledem k časové ose.

Požadavek 2: Nastavuje se mapování pro ElasticSearch.

Požadavek 3: Obsahuje vyhledávání nad schématem databáze.

Požadavek 4: Lze nastavit, jak se jednotlivé části schéma budou chovat. Například, zda se nad danou vlastností může vyhledávat, agregovat, nebo jí lze celou vyjmout a ignorovat.

Požadavek 5: Nejedná se o relační databázi. Spojování funguje jako v ElasticSearch.

Požadavek 6: Náhled je vidět stále při editaci dotazu, pouze je nutné ho aktualizovat tlačítkem.

Požadavek 7: Jsou zde tři druhy šablon a pohledů. První je dotaz nad daty, který vytvoří dataset. Druhý je z datasetu vytvořená vizualizace. Třetí je z vizualizací vytvořený dashboard. Uživatel je může ukládat, načítat i sdílet.

Požadavek 8: Neobsahuje žádné výstupní rozhraní. Je možné se připojit rovnou na ElasticSearch.

Požadavek 9: Pomocí pluginu X-Pack lze exportovat reporty pouze do PDF. Jinak je možné sdílení vložením do HTML stránky pomocí iframe, nebo případně odkazem.

Požadavek 10: Nelze, je potřeba mít vše nahrané v ElasticSearch.

3.2.6 DBxtra

DBxtra [2, 3] je placený BI reportovací nástroj. Nevyžaduje vyhrazenou IT infrastrukturu, jako jsou specializované reportovací servery. Umožňuje rychlé a jednoduché vytváření reportů a dashboardů i nezkušeným uživatelům, kteří nemají znalost SQL, programování nebo webových technologií. Skládá se z několika komponent (modulů). Report Server je hlavní modul, se kterým ostatní spolupracují. Nachází se v něm Report Repository, který obsahuje všechny definice reportů a konfigurace. Také se v něm nachází Report Web Service, která umožňuje uživatelům přístup k reportům z libovolného prohlížeče.

K tvorbě reportů se využívá Designer modul. Po připojení k datovému zdroji a vytvoření dotazu (Query), je možné tvořit různé reportovací objekty. Tyto objekty jsou datové mřížky (Data Grid), pivotovací mřížky (Pivot Grid), reporty a dashboardy. Schedule Server následně umožňuje nastavení plánování reportů, například opakované a časované generování a následné nahrání reportu na FTP server.

Požadavek 1: Neobsahuje žádné statistiky.

Požadavek 2: Vazby si nástroj bere ze struktury připojeného datového zdroje a přidání metadat pro definici struktur není možné.

Požadavek 3: Nelze prohledávat sloupce ani tabulky při tvorbě dotazu v Designeru.

Požadavek 4: Nelze nastavit podmnožinu dat. Práva se přidělují na celý datový zdroj.

Požadavek 5: V Designeru se vytvoří dotaz, ve kterém si uživatel definuje pohled nad svými daty. Následně z něj vychází report.

Požadavek 6: Náhled nahrazuje záložka s datovou mřížkou, které data zobrazí v interaktivní tabulce a přidává další funkce.

Požadavek 7: Je zde několik reportovacích objektů a každý lze ukládat a načítat. Objekty jsou zmíněné výše (Query, Data Grid, Pivot Grid, Report, Dashboard).

Požadavek 8: Lze vytvořit pouze Excel Linked Sheet a Excel Linked Pivot.

Požadavek 9: Data lze exportovat do následujících formátů: PDF, HTML, TXT, CSV, MHT, XLS, RTF, Image a XML.

Požadavek 10: Podporuje heterogenní dotazy, takže lze spojit více datových zdrojů do jednoho dotazu. Podporované datové zdroje jsou MS SQL Server, Access, Excel, Oracle DB, MySQL, IBM DB2, IBM Informix, PostgreSQL, Firebird, Text, ODBC, OLE DB.

3.3 Hodnocení a rozhodnutí

Získané poznatky jsou pro přehlednost shrnuty do Tabulky 3.1, kde jsou jednotlivé body požadavků hodnoceny plusy a mínusy. Nejlepší hodnocení jsou tři plusy a naopak nejhorší jsou tři mínusy. Nejlépe obstálo Pentaho. Po domluvě s vedoucím práce jsme se rozhodli, že implementujeme prototyp vlastního nástroje, který bude splňovat naše požadavky a umožní ověření jejich proveditelnosti. Hlavní důvod je, že ani Pentaho nesplňuje dva ze tří hlavních požadavků.

Požadavek	Pentaho	SpagoBI	Dremio	JasperReports	Kibana	DBxtra
1	- -	- - -	- - -	- - -	+ +	- - -
2	+ +	+ +	- - -	- - -	+	- - -
3	- - -	- - -	+	- - -	+ +	- - -
4	+ + +	+ +	- -	- - -	+ +	- - -
5	+ +	+ +	+	+ +	- -	+ + +
6	+ +	+	+	+	+	+ + +
7	+ + +	+ +	+	+ + +	+ + +	+ + +
8	+	- - -	+ + +	- -	- -	+
9	+ + +	+	- -	+ + +	-	+ + +
10	+ + +	+ + +	+ +	+	- - -	+ + +

Tabulka 3.1: Hodnocení jednotlivých požadavků zkoumaných nástrojů

4 Statistiky a informace nad daty

V této části jsou popsány statistiky a informace, které uvažujeme nad tabulkami a sloupci, a možnosti jejich získání. Motivací pro zobrazování statistik informací je lehčí orientace zákazníka v datech.

Protože se v DCIx pracuje s relační databází MS SQL Server, určíme si několik objektů, které budou v práci používány a co pro nás představují. Prvním objektem, který používáme, je entita (Entity). Jedná se o tabulku nebo pohled v databázi. Entita může mít vlastnosti (Properties), což jsou databázové sloupce. Také má cizí klíče (ForeignKeys), což jsou databázové cizí klíče, které určují vztahy mezi entitami.

Požadavek 1 z části 3.1 nám říká, že by nástroj měl umět zobrazovat statistické informace nad tabulkami a jejich sloupci. Jedná se o informace typu kolik je záznamů v dané tabulce, jaká je průměrná hodnota daného sloupce, jaké procento hodnot je nulových nebo jestli je daný sloupec nějak omezen. Tyto informace se mohou lišit podle zkoumaného datového typu.

Například průměrná hodnota pro sloupec obsahující různé řetězce obvykle nemá žádnou vypovídající hodnotu. Na druhou stranu, pokud sloupec s řetězci nabývá pouze malého množství různých hodnot, jejich četnosti výskytu vypovídající hodnotu mají.

V této části se pokusíme určit statistiky a informace pro různé datové typy tak, aby měly smysl a byly užitečné. Dále prozkoumáme možnosti jejich získávání z aplikace DCIx. Statistika a pravděpodobnost je obsáhlý vědní obor, a proto jsme si vybrali pouze velmi malou podmnožinu. Například míry centrální tendence, míry variability, míry korelace, centrální limitní větu, testování hypotéz a další jsou nad rámec této práce a dále je neuvažujeme.

4.1 Statistiky a informace nad tabulkami

Nad tabulkou není mnoho statistik, které by uživateli pomohly v rozhodování. Navrhujeme následující.

Počet záznamů: Jedná se o počet záznamů pro danou tabulku. Je to pravděpodobně první statistika, která uživatele napadne, pokud chce s tabulkou pracovat. Příklad: Pokud má v tabulce miliony záznamů, musí

být při připojování opatrnější. Především z důvodu zvýšení složitosti dotazu.

Vazby: Jedná se o počet odchozích a příchozích vazeb pro danou tabulku. Příklad: Uživatel lehce pozná, že se jedná o izolovanou nebo spojovací tabulku.

4.2 Statistiky a informace nad sloupci

Statistiky nad sloupci rozdělíme podle příslušného datového typu. Vycházíme z datových typů pro SQL Server 2008 a novější [12]. Pro zjednodušení jsme datové typy rozdělili do pěti skupin, kde každá skupina bude mít svojí množinu relevantních statistik a informací. Uvažujeme následující statistiky a informace.

Minimum: Jedná se o minimální hodnotu pro daný sloupec. Příklad: Pokud uživatel chce zjistit, od jakého data záznamy v tabulce pocházejí, stačí, aby se podíval na sloupec s datem poslední úpravy a jeho minimum.

Maximum: Jedná se o maximální hodnotu pro daný sloupec. Příklad: Pokud uživatel chce zjistit, kdy naposledy byla data v tabulce upravena, stačí, aby se podíval na sloupec s datem poslední úpravy a jeho maximum.

Průměr: Jedná se o průměrnou hodnotu pro daný sloupec. Příklad: Pokud uživatel chce zjistit průměrné množství, stačí, aby se podívat na sloupec s množstvím a jeho průměr.

Nulové hodnoty: Jedná se o procentuální množství nulových hodnot v daném sloupci nebo informace, že ve sloupci nemohou být nulové hodnoty. Příklad: Pokud uživatel má ve sloupci všechny hodnoty nulové, nemusí ho zohledňovat, protože se pravděpodobně nepoužívá.

Integritní omezení: Jedná se o případ, kdy daný sloupec má doménové integritní omezení na určení povolené množiny hodnot. Používá se často pro výčtové typy. Příklad: Uživatel v tabulce vidí, že všechny řádky daného sloupce mají pouze jednu stejnou hodnotu. Zajímá ho, jestli to není jen náhoda, a jaké jsou další přípustné hodnoty.

Histogram: Jedná se o histogram hodnot daného sloupce. Příklad: Uživatel chce zjistit, jak přibývají objednávky do systému v čase, stačí, aby podíval na sloupec vytvoření, a v histogramu uvidí četnosti rozdělené do časových intervalů.

Kromě těchto skoro univerzálních statistik a informací lze navrhnout libovolné jiné. Ty již budou více omezené a specifické. U určitých statistik nebo informací pro daný datový typ je v následujících sekcích uvedeno, že tato informace nebo statistika není vhodná k zobrazení. To je z důvodu, že jsme nepřišli na případ, kdy by daná statistika nebo informace měla smysl nebo by byla užitečná pro uživatele. Také v následujících sekcích již neuvádíme informaci *Nulové hodnoty*, kterou je vhodné zobrazit vždy.

4.2.1 Řetězce

Do této skupiny spadají následující datové typy:

`char`, `varchar`, `text`, `nchar`, `nvarchar`, `ntext`

Statistiky a informace pro řetězce:

Minimum: Není vhodné k zobrazení.

Maximum: Není vhodné k zobrazení.

Průměr: Není vhodný k zobrazení.

Integritní omezení: Je vhodné k zobrazení. Typické pro výčtové sloupce.

Histogram: Ohledně histogramu to je značně specifické na obsahu řetězců. V případě, kdy se ve sloupci nachází maximálně desítky různých hodnot, je vhodné ho zobrazit. V ostatních případech by byl značně nepřehledný a zobrazení nevhodné.

4.2.2 Čísla

Do této skupiny spadají následující datové typy:

`tinyint`, `smallint`, `int`, `bigint`, `decimal`, `numeric`, `smallmoney`, `money`, `float`, `real`

Statistiky a informace pro čísla:

Minimum: Je vhodné k zobrazení.

Maximum: Je vhodné k zobrazení.

Průměr: Je vhodný k zobrazení.

Integritní omezení: Je vhodné k zobrazení. Typické pro omezené stavy.

Histogram: Je vhodný k zobrazení.

4.2.3 Časové údaje

Do této skupiny spadají datové typy:

`date, smalldatetime, datetime, datetime2, datetimeoffset, time`

Statistiky a informace pro časové údaje:

Minimum: Je vhodné k zobrazení.

Maximum: Je vhodné k zobrazení.

Průměr: Není vhodný k zobrazení.

Integritní omezení: Je vhodné k zobrazení. Typické pro omezené stavy.

Histogram: Je vhodný k zobrazení.

4.2.4 Logická hodnota

Speciální skupina, které oddělila datový typ `bit` od čísel. Přidali jsme novou informaci *Pravda/Nepravda*, kterou je vhodné zobrazit pouze u této skupiny. Statistiky a informace pro logickou hodnotu:

Minimum: Není vhodné k zobrazení.

Maximum: Není vhodné k zobrazení.

Průměr: Není vhodný k zobrazení.

Integritní omezení: Není vhodné k zobrazení.

Histogram: Lze zobrazit pro vizualizaci poměru, ale není to nutné.

Pravda/Nepravda: Jedná se o procentuální poměr mezi pravdou a nepravdou pro hodnoty daného sloupce. Příklad: Uživatel hned vidí, jaké procento záznamů je neaktivních.

4.2.5 Ostatní

Zde jsme zařadili ostatní nejmenované datové typy. Také jsme sem přesunuli zbylé řetězcové typy `binary, image a varbinary`. To z důvodu, že navržené statistiky pro řetězce pro ně nemají smysl. Jak bylo zmíněno výše, pouze informaci *Nulové hodnoty* je vhodné zobrazit.

4.3 Získávání statistik

V minulé sekci jsme definovali statistiky a informace, které je vhodné zobrazovat. Nyní je potřeba je získat. Napadly nás tři možné přístupy, které lze využít. Prvním je využití jazyka SQL a připravit dotazy pro dané informace a statistiky. Druhým je využití informací, které v sobě uchovává databázový engine. A třetí přístup využívá existující nástroje.

Jsou zde tři hlavní faktory, které jsme zohlednili. Prvním je rychlost a zatížení databáze. Chce uživatel vidět aktuální informace z produkční databáze, nebo se bude dotazovat nad archivem? Druhým je přesnost. Potřebuje uživatel mít statistiky a informace přesné, nebo mu stačí přibližné? A třetím faktorem je univerzálnost. Je možné je nezávisle získat na libovolném databázovém serveru, nebo stačí MS SQL Server. Poznatky jsou shrnuty do Tabulky 4.1.

Faktor / Přístup	Rychlost a zatížení	Přesnost	Univerzálnost
SQL	−	+	+
Engine	+	−	−
Existující nástroj	×	×	+

Tabulka 4.1: Přehled možností získání statistik

Sestavili jsme upřesňující požadavky. Ty jsou, že statistiky a informace nesmí zbytečně zatěžovat databázi a pro orientaci stačí přibližné hodnoty. Poté vychází jako vítěz dotazování přímo na MS SQL Server.

4.3.1 SQL dotazy

Jedná se o sadu SQL [39] dotazů, které nám vrátí chtěné hodnoty. Tento způsob by měl být univerzální pro všechny SQL databáze, ale nachází se zde ovšem výjimky. K získání některých hodnot je potřeba využít uložených procedur a funkcí. Ty jsou již závislé na dané databázi. Například MS SQL Server má pro procedury rozšíření T-SQL a Oracle má procedurální jazyk PL/SQL. Tyto statistiky a informace jsou přesné, často i velmi přehledné a jednoduché, ale jejich získávání nemusí být rychlé a mohou zbytečně zatěžovat databázový stroj.

Počet záznamů: Existuje sloupcová funkce (column function) `COUNT`, která počítá nenulové záznamy ve sloupci nebo počet řádků ve výsledku

dotazu. Pro naše využití stačí použít tuto funkci nad primárním klíčem tabulky.

Vazby: Vazby jsou v relační databázi reprezentovány pomocí cizích klíčů. Ty lze získat ze standardního systémového katalogu (`INFORMATION_SCHEMA`). V něm se nachází pohledy na databázi z různých perspektiv. Cizí klíče lze získat s využitím pohledu `KEY_COLUMN_USAGE`. V tomto pohledu jsou důležité sloupce `REFERENCED_TABLE_NAME` a `TABLE_NAME` .

Minimum: Existuje sloupcová funkce `MIN`, která najde nejmenší hodnotu ve sloupci. Pro naše využití stačí použít tuto funkci nad daným sloupcem.

Maximum: Existuje sloupcová funkce `MAX`, která najde největší hodnotu ve sloupci. Pro naše využití stačí použít tuto funkci nad daným sloupcem.

Průměr: Existuje sloupcová funkce `AVG`, která spočítá průměrnou hodnotu ve sloupci. Pro naše využití stačí použít tuto funkci nad daným sloupcem.

Nulové hodnoty: Omezení nulových hodnot je integritní omezení. S využitím systémového katalogu pro pohled `COLUMNS` je přítomen sloupec `IS-NULLABLE`. Procentuální množství nulových hodnot lze získat SQL funkcí, která bude využívat sloupcovou funkci `COUNT` a podmínku `IS NULL`.

Integritní omezení: Zda je sloupec omezen integritním omezením typu `CHECK` lze opět zjistit ze systémového katalogu z pohledu `CHECK_CONSTRAINTS`. Jeho předpis se nachází ve sloupci `CHECK_CLAUSE`.

Histogram: Nenašli jsme rychlou standardní SQL funkci, která by vrátila histogram. Lze však napsat generickou T-SQL funkci, která umí vrátit histogram [36].

Pravda/Nepravda: Nenašli jsme rychlou standardní SQL funkci, která by vrátila poměr v procentech. Lze napsat jednoduchou T-SQL funkci s využitím `COUNT` podobně jako u nulových hodnot.

4.3.2 MS SQL Server

Zde se také jedná o sadu dotazů, ale již nevyžívají standardní tabulky a funkce. Databázový engine MS SQL serveru si v sobě udržuje statistiky [38]. Tyto statistiky využívá pro zlepšení výkonu. Jsou důležité při tvorbě plánů pro vykonávání dotazů (query execution plans). Pro optimální výkon

je vhodné je mít aktuální. Ne každý sloupec statistiky má. Pro sloupce, u kterých chceme počítat statistiky, je nutné zjistit jméno jejich odpovídající statistiky. K tomu je možné využít systémové tabulky `SYS.COLUMNS`, `SYS.STATS` a `SYS.STATS_COLUMNS`. Jakmile má sloupec přiřazenou svojí statistiku, je možné získat její data pomocí příkazu `DBCC SHOW_STATISTICS`. Je možné si zvolit, zda funkce vrátí statistickou hlavičku (`WITH STAT_HEADER`) nebo histogram (`WITH HISTOGRAM`). Statistická hlavička obsahuje následující sloupce, které jsou pro nás relevantní:

- Kdy byla statistika naposled aktualizována (`UPDATED`).
- Počet řádků kolik měla tabulka, když se počítala (`ROWS`).
- Počet řádků z kolika statistika vznikla (`ROWS SAMPLED`).
- Počet intervalů histogramu (`STEPS`)

Hodnoty v daném sloupci se automaticky rozdělují do intervalů pro histogram. Intervalů může být maximálně 200. Histogram obsahuje následující sloupce, které jsou pro nás relevantní:

- Hodnota horní hranice pro daný interval (`RANGE_HI_KEY`).
- Odhadovaný počet řádek, jejichž hodnota v daném sloupci je rovna horní hranici daného intervalu (`EQ_ROWS`).
- Odhadovaný počet řádek, jejichž hodnota v daném sloupci spadá do daného intervalu kromě horní hranice (`RANGE_ROWS`).

Díky tomu, že hodnoty jsou získávané ze systémových tabulek a nepočítají se při každém dotazu, nejsou náročné na výkon. Přepočítávání statistik ve výchozím nastavení inicializuje optimalizátor dotazů, ale lze jejich přepočítání vynutit manuálně pomocí dotazu `UPDATE STATISTICS` nebo pro všechny tabulky definované uživatelem procedurou `SP_UPDATESTATS`. Protože je většina hodnot z hlavičky a histogramu odhadovaná nebo aktuální pro daný moment počítání statistiky, tak jsou i všechny informace a statistiky z nich získané jen přibližné.

Počet záznamů: Přibližný počet záznamů lze zjistit z hlavičky `ROWS`. Také to lze zjistit s využitím tabulek `SYS.TABLES`, `SYS.INDEXES` a `SYS.PARTITIONS` a sloupce `ROWS`.

Vazby: Vazby lze získat z procedury `SP_FKEYS`. Také je lze získat s využitím systémových tabulek `SYS.FOREIGN_KEY_COLUMNS`, `SYS.TABLES` a `SYS.COLUMNS`.

Minimum: Tato hodnota lze zjistit z histogramu. Stačí najít nejmenší nenulovou dolní hranici intervalu.

Maximum: Tato hodnota lze zjistit z histogramu. Stačí najít největší nenulovou horní hranici intervalu.

Průměr: Tato hodnota lze zjistit z histogramu. Pro pro přibližnou hodnotu stačí vzít v potaz hodnoty z histogramu a dopočítat průměr.

Nulové hodnoty: Omezení nulových hodnot lze zjistit obdobně jako v 4.3.1 jen s využitím tabulky `SYS.COLUMNS`. Pro procentuální množství nulových hodnot je možné opět použít histogram. Lze využít intervalu s nulovými hodnotami.

Integritní omezení: Lze získat opět obdobně jako v 4.3.1 s využitím tabulky `SYS.CHECK_CONSTRAINTS` a sloupce `DEFINITION`.

Histogram: Lze zavolat statistiku s příznakem histogramu a intervaly a četnost záznamů určit ze sloupců `RANGE_HI_KEY, EQ_ROWS` a `RANGE_ROWS`.

Pravda/Nepravda: Obdobně jako v 4.3.1 ale s využitím hodnot z histogramu místo funkce `COUNT`.

4.3.3 Existující nástroj

Jak lze vidět v sekci 3.2, nástroje, které by uměly statistiky nejsou časté. Existují ještě nástroje, které se specializují na tento typ úloh. Patří sem nástroje typu Matlab, Excel nebo R. Pro naše potřeby integrace jsou nevhodné.

5 Generování datových pohledů

V této kapitole rozebereme možnosti získání datového pohledu pro vytvoření sestavy v aplikaci DCIx. Tabulku v sestavě uživatel může libovolně filtrovat nebo řadit. Navíc si může vytvořit a uložit vlastní pohled na tuto tabulku. V něm si může nastavit, jaké chce zobrazit sloupce a případně jejich výchozí filtr a řazení. K tomu je potřeba mít na pozadí datový pohled, ze kterého tabulka čerpá data. Datové pohledy jsou v DCIx pevně definovány, tudíž je i omezená množina tabulek, se kterými uživatel může pracovat.

Cílem je, aby si mohl uživatel datový pohled sám vytvořit a v aplikaci DCIx mu vznikla nová sestava založená na tomto pohledu. K tomu potřebuje mít expertní znalost databázové struktury DCIx a jazyka SQL, aby pohled, který si vytvoří, obsahoval smysluplná a správná data. To uživatel obvykle nemá. Proto vznikla tato práce, která by běžnému uživateli měla pomoci tento pohled vytvořit.

V první části jsou popsány obecné možnosti a struktura dotazu `SELECT` z jazyka SQL. Ten slouží k získávání dat z databáze. V druhé části specifikujeme podmnožinu těchto možností, které budou pro prototyp nástroje dostačující a budou dále implementovány. A nakonec ve třetí části popíšeme, jak přidat nový pohled a sestavu do aplikace DCIx.

5.1 Struktura a možnosti dotazu `SELECT`

Tato část se bude zabývat možnostmi a strukturou dotazu `SELECT` [39] z jazyka SQL. Rozbor je proveden pro MS SQL Server dialekt [25].Tento dotaz slouží k získávání dat a informací z databáze. Výsledek dotazu je vrácen v určitém formátu a říká se mu "result-set". Obvykle pro MS SQL Server je tímto result-setem tabulka. Ta zahrnuje jména jednotlivých sloupců, což jsou metadata. Dále obsahuje jednotlivé řádky, které jsou již vyplněny daty. Možnosti dotazu jsou spjaté s jeho strukturou, proto v následujících částech popíšeme oblasti, ze kterých se dotaz může skládat. Vycházíme ze syntaxe pro MS SQL Server, kterou jsme uvedli v syntaxi 5.1.

```
<SELECT statement> ::=
    [ WITH { [ XMLNAMESPACES ,] [ <common_table_expression> [ ,...n] ] } ]
    <query_expression>
    [ ORDER BY { order_by_expression | column_position [ ASC | DESC ] }
  [ ,...n ] ]
    [ <FOR Clause>]
    [ OPTION ( <query_hint> [ ,...n ] ) ]
<query_expression> ::=
    { <query_specification> | ( <query_expression> ) }
    [  { UNION [ ALL ] | EXCEPT | INTERSECT }
        <query_specification> | ( <query_expression> ) [...n ] ]
<query_specification> ::=
SELECT [ ALL | DISTINCT ]
    [TOP ( expression ) [PERCENT] [ WITH TIES ] ]
    < select_list >
    [ INTO new_table ]
    [ FROM { <table_source> } [ ,...n ] ]
    [ WHERE <search_condition> ]
    [ <GROUP BY> ]
    [ HAVING < search_condition > ]
```

Syntax 5.1: Dotaz SELECT pro MS SQL Server

5.1.1 Dotaz SELECT

Jedná se o část `<SELECT statement>` ze syntaxe 5.1. Skládá se z povinné části `<query_expression>`, která představuje výraz dotazu a je popsána v následující sekci 5.1.2, a několika volitelných částí, jejichž popis následuje.

První volitelnou částí je konstrukce `WITH`. Ta může obsahovat definici `XMLNAMESPACES`, což jsou jmenné prostory pro dotazy, které pracují s XML dokumenty. Nebo může obsahovat výraz `<common_table_expression>`, což je specifikace dočasného result-setu. Ty zde dále rozebírat nebudeme. V příkladu 5.1 jsme uvedli, jak může vypadat dotaz s využitím jmenných prostorů. A v příkladu 5.2 jsme uvedli, jak může vypadat použití dočasného result-setu.

```
WITH XMLNAMESPACES ('http://example.com/ns' as ns)
SELECT   ID       as 'ns:ProductID',
         Code     as 'ns:ProductCode'
FROM dciowner.Products
WHERE ProductID=666
FOR XML RAW ('ns:Product'), ELEMENTS
```

Příklad 5.1: Použití WITH XMLNAMESPACES a <FOR Clause>

Druhou volitelnou částí je konstrukce `ORDER BY`. Ta určuje, jak bude výsledný result-set seřazený. Je stejná jako ve standardu SQL. Navíc zde lze specifikovat offset, s kterým záznamy budou vráceny. To je konstrukce `<offset_fetch>` a je uvedena v syntaxi 5.2 a dále ji rozebírat nebudeme. Jak může vypadat příkaz `ORDER BY` je uvedeno na konci kapitoly v příkladu 5.4,

kde výsledek bude nejprve seřazen podle kódu transakční definice vzestupně, následně podle data vzniku sestupně a nakonec vrátí prvních sto záznamů.

```
WITH Transactions_CTE (ID, User_ID, Create_Date)
AS (
    SELECT ID, User_ID, YEAR(Create_Date) CreateYear
    FROM dciowner.Transactions
    WHERE User_ID IS NOT NULL )
SELECT User_ID, COUNT(ID) AS TotalTransactions, CreateYear
FROM Transactions_CTE
GROUP BY CreateYear, User_ID
```

Příklad 5.2: Použití WITH <common_table_expression>

```
<offset_fetch> ::=
{
    OFFSET { integer_constant | offset_row_count_expression } { ROW | ROWS }
    [
      FETCH { FIRST | NEXT } {integer_constant | fetch_row_count_expression }
        { ROW | ROWS } ONLY
    ]
}
```

Syntax 5.2: Konstrukce <offset_fetch>

Třetí volitelnou částí je konstrukce `FOR`. V MS SQL Serveru si lze zvolit ze tří možností. První možností je `FOR BROWSE`, která umožňuje změny při prohlížení výsledků v kurzoru s prohlížecím režimem. Druhou je `FOR XML`, která mění klasický formát result-setu na XML. Nakonec třetí je `FOR JSON`, která také mění formát result-setu, ale nyní na JSON. Ukázka je v příkladu 5.1.

Čtvrtou a zároveň poslední volitelnou částí je konstrukce `OPTION`. Ta slouží k nápovědám pro optimalizátor, který určuje, jak by měl engine vykonávat daný dotaz. Mezi tyto nápovědy patří například možnost vynutit určitý typ spojování (`HASH JOIN`, `LOOP JOIN`, `MERGE JOIN`), ale to je mimo téma této práce.

5.1.2 Výraz dotazu

V této části se popisuje část `<query_expression>` ze sytaxe 5.1. Obsahuje pouze jednu povinnou část. Tou je buď opět výraz dotazu v kulatých závorkách, nebo specifikace dotazu `<query_specification>`, která je popsána v následující části 5.1.3. Nad touto povinnou částí lze provést operace s dalšími výrazy dotazu nebo specifikacemi dotazu. K tomu lze využít standardní SQL operátory `UNION`, `UNION ALL`, `EXCEPT`[1] a `INTERSECT`. V příkladu 5.3 je použité sjednocení, ve výsledku budou všechny rozdílné kódy zákazníků a dodavatelů.

[1]U operátoru `EXCEPT` je potřeba si dávat pozor na pořadí operandů, protože není komutativní

```
SELECT Code
FROM dciowner.Customers
UNION
SELECT Code
FROM dciowner.Suppliers
```

Příklad 5.3: Použití <query_expression>

5.1.3 Specifikace dotazu

Jedná se o závěrečnou část `<query_specification>` ze syntaxe 5.1. Toto je jádro celého dotazu. Obsahuje také jednu povinnou část a tou je výběr, který chceme vrátit. Jedná se o část `<select_list>`, která je popsána v sekci 5.1.4. Druhá povinná část vznikne, pokud je ve výběru něco jiného než konstanty, proměnné nebo aritmetické výrazy. Tato část je konstrukce `FROM` a je popsána v části 5.1.7.

První volitelnou částí je určení, že chceme pouze unikátní řádky ve výsledku. K tomu slouží standardní `DISTINCT`. Jak jej lze použít ukazujeme v příkladu 5.4.

Druhou volitelnou částí je omezení počtu výsledných řádek. To lze zařídit pomocí klíčového slova `TOP` (ve standardu slovo `LIMIT`). Lze omezit na počet, který je zadán jako `( expression )`. Pokud chceme omezit na určité procento, použijeme klíčové slovo `PERCENT`. Pokud je použita klauzule `ORDER BY` a chceme, aby při shodnosti řádek byly vráceny všechny[2], tak použijeme klíčové slovo `WITH TIES`.

Třetí volitelnou částí je vložení do nové tabulky. To se provádí pomocí klíčového slova `INTO` a za ním se nachází jméno nové tabulky, kam budou data z result-setu dotazu vložen a struktura nové tabulky bude odpovídat metadatům.

Další volitelnou částí je konstrukce `WHERE`. Ta specifikuje vyhledávací podmínky, pro výsledný result-set. Jak se tyto podmínky specifikují je uvedeno v části 5.1.6,

Následující volitelnou částí je konstrukce `GROUP BY`. Ta slouží k rozdělení výsledného result-setu do skupin. Obvyklým účelem je použití agregací nad těmito skupinami. Tato konstrukce je popsána v syntaxi 5.3 a dále ji rozebírat nebudeme.

[2]Máme v tabulce záznamy (1), (2), (3), (3), (4) a použijeme dotaz s TOP (3) WITH TIES a ORDER BY ASC. Ve výsledném result-setu budou záznamy (1), (2), (3), (3).

```
GROUP BY {
      column-expression
    | ROLLUP ( <group_by_expression> [ ,...n ] )
    | CUBE ( <group_by_expression> [ ,...n ] )
    | GROUPING SETS ( <grouping_set> [ ,...n ]  )
    | () --calculates the grand total
} [ ,...n ]

<group_by_expression> ::=
      column-expression
    | ( column-expression [ ,...n ] )

<grouping_set> ::=
      () --calculates the grand total
    | <grouping_set_item>
    | ( <grouping_set_item> [ ,...n ] )

<grouping_set_item> ::=
      <group_by_expression>
    | ROLLUP ( <group_by_expression> [ ,...n ] )
    | CUBE ( <group_by_expression> [ ,...n ] )
```

Syntax 5.3: Konstrukce GROUP BY

5.1.4 Výběr do výsledku

Tato sekce představuje konstrukci `<select_list>` ze syntaxe 5.1, která specifikuje výběr toho, co chceme z dotazu vrátit. Podrobněji je uvedena v syntaxi 5.4. Výraz `expression` je popsán v části 5.1.5. Je zde několik dalších možností jak výběr provést.

```
<select_list> ::=
    {
      *
      | { table_name | view_name | table_alias }.*
      | {
          [ { table_name | view_name | table_alias }. ]
               { column_name | $IDENTITY | $ROWGUID }
          | udt_column_name [ { . | :: } { { property_name | field_name }
            | method_name ( argument [ ,...n] ) } ]
          | expression
          [ [ AS ] column_alias ]
        }
      | column_alias = expression
    } [ ,...n ]
```

Syntax 5.4: Konstrukce výběru sloupců <select_list>

První možností je vybrat sloupec nebo sloupce. Nejobvyklejší je zvolit objekt, který daný sloupec obsahuje a přes tečkovou notaci a název sloupce ho vybrat. Tento výběr sloupců je také v příkladu 5.4.

Poté je zde speciální znak `*`, které vybírá všechny sloupce, které do dotazu vstupují. Také lze tento znak použít i na specifickou tabulku, pohled nebo jejich alias.

Dále se zde dají specifikovat aliasy pomocí klíčového slova `AS` nebo pro verze MS SQL Server 2008 a vyšší lze využít klasické přiřazení se znakem `=`. Také lze klíčové slovo vypustit. Aliasy se hodí pokud chceme přejmenovat výsledný sloupec v result-setu.

Výrazy `udt_column_name`, `{. | ::}`, `property_name`, `field_name` a `method_name`, se vztahují k uživatelsky definovaným typům v Common Language Runtime a nejsou pro nás podstatné.

5.1.5 Výraz

Výrazem `expression` je míněná konstanta, proměnná, skalární funkce nebo libovolná kombinace sloupců, konstant, proměnných a skalárních funkcí spojených operátory. Také to může být celý poddotaz (subquery), což může být další vnořený příkaz `SELECT`, který vrací skalární výsledek. Případně lze využít i výraz s podmínkou `CASE`. Jak se dají použít lze vidět na konci celé části v příkladu 5.4.

5.1.6 Vyhledávací podmínky

V této části popíšeme vyhledávací podmínky `<search_condition>` a jejich možnosti. Jsou specifikovány v syntaxi 5.5. Vyhledávací podmínka je jeden nebo více predikátů `<predicate>` spojených logickými operátory. Obsahuje standardní logické SQL operátory `NOT`, `AND` a `OR`. Dále obsahuje standardní SQL konstrukce `IS NULL`, `BETWEEN` nebo`IN`.

```
<search_condition> ::=
    { [ NOT ] <predicate> | ( <search_condition> ) }
    [ { AND | OR } [ NOT ] { <predicate> | ( <search_condition> ) } ]
[ ,...n ]

<predicate> ::=
    { expression { = | < > | ! = | > | > = | ! > | < | < = | ! < } expression
    | string_expression [ NOT ] LIKE string_expression
  [ ESCAPE 'escape_character' ]
    | expression [ NOT ] BETWEEN expression AND expression
    | expression IS [ NOT ] NULL
    | CONTAINS
  ( { column | * } , '<contains_search_condition>' )
    | FREETEXT ( { column | * } , 'freetext_string' )
    | expression [ NOT ] IN ( subquery | expression [ ,...n ] )
    | expression { = | < > | ! = | > | > = | ! > | < | < = | ! < }
  { ALL | SOME | ANY} ( subquery )
    | EXISTS ( subquery )     }
```

Syntax 5.5: Konstrukce vyhledávacích podmínek <search_condition>

Predikát se skládá z výrazů `expression`, které jsou popsané v 5.1.5. Tyto výrazy lze spojovat pomocí operátorů. Predikát nemusí vždy vracet hodnotu pravda nebo nepravda, může být vyhodnocen i jako neznámá hodnota.

Pro práci s řetězci jsou zde další klíčová slova. Prvním je standardní SQL klíčové slovo `LIKE`. Navíc jsou zde klíčová slova `CONTAINS` a `FREETEXT`, která slouží k vyhledávání nad více sloupci.

Poddotaz je zde omezen a nesmí se použít `ORDER BY` a `INTO`, jinak platí pravidla 5.1.2. S poddotazy se pojí standardní `EXISTS`, `ALL` a synonyma `SOME` a `ANY`. Různé typy vyhledávacích podmínek jsou vidět v příkladu 5.4.

5.1.7 Konstrukce `FROM`

Kontrukce `FROM` určuje odkud se budou data vybírat. Specifikace je uvedena v syntaxi v 5.6. Také zde lze opět využít systém aliasů jako v 5.1.4. S rozdílem, že nelze využít přiřazení `=`. Tabulkový zdroj dat, lze vybrat několika způsoby.

```
[ FROM { <table_source> } [ ,...n ] ]
<table_source> ::=
{
    table_or_view_name [ [ AS ] table_alias ]
        [ <tablesample_clause> ]
        [ WITH ( < table_hint > [ [ , ]...n ] ) ]
    | rowset_function [ [ AS ] table_alias ]
        [ ( bulk_column_alias [ ,...n ] ) ]
    | user_defined_function [ [ AS ] table_alias ]
    | OPENXML <openxml_clause>
    | derived_table [ [ AS ] table_alias ] [ ( column_alias [ ,...n ] ) ]
    | <joined_table>
    | <pivoted_table>
    | <unpivoted_table>
    | @variable [ [ AS ] table_alias ]
    | @variable.function_call ( expression [ ,...n ] )
        [ [ AS ] table_alias ] [ (column_alias [ ,...n ] ) ]
    | FOR SYSTEM_TIME <system_time>
}
```

Syntax 5.6: Konstrukce FROM

První je nejjednodušší a to je uvést jméno tabulky, ze které chceme data získat. Lze k tomu využít opět nápovědu pro optimalizátor dotazů `WITH`, tentokrát pro tabulku. Také lze tabulku omezit, aby vracela jen určité množství záznamů podobně jako u `TOP`. K tomu slouží dodatek `<tablesample_clause>`.

Druhým způsobem je využití funkce, která vrací řádky. To jsou `OPENROWSET` funkce a pokud je zvolena možnost `BULK`, tak lze využít i aliasy sloupců.

Třetím je použití uživatelem definovaných funkcí. Ty musí splňovat podmínku, že vrací tabulku(table-valued function) a ne skalární hodnotu.

Dalším je `OPENXML`, který umožňuje číst data jako řádky z XML dokumentu.

Pátým způsobem je použití poddotazu, který vrací tabulku. Jedná se o část `derived_table`.

Nejdůležitějším způsobem je spojování tabulek. Tím se zabývá dodatek `<joined_table>`. Podrobně je spojování popsáno v sekci 5.1.8.

Další způsob je použití proměnných `@variable`. Lze si definovat tabulkovou proměnnou. Případně zavolat tabulkovou funkci z proměné.

Pro MS SQL Server 2016 a novější lze využít dodatek `FOR SYSTEM_TIME`, kterým lze specifikovat časové období. Poté se vyberou řádky, které byly v tomto období aktivní.

Ostatní způsoby se týkají *pivotování*. V této práci je rozebírat nebudeme.

5.1.8 Spojování tabulek

V této části je popsáno, jak lze tabulky mezi sebou propojovat. Podrobně je konstrukce `<joined_table>` popsána v syntaxi 5.7. Jedná se o důležitou část této práce, proto zde zopakujeme i základy standardu SQL. Lze mezi sebou spojovat libovolné tabulkové zdroje, které jsou již popsané v sekci 5.1.7.

```
<joined_table> ::=
{
    <table_source> <join_type> <table_source> ON <search_condition>
    | <table_source> CROSS JOIN <table_source>
    | left_table_source { CROSS | OUTER } APPLY right_table_source
    | [ ( ] <joined_table> [ ) ]
}
<join_type> ::=
    [ { INNER | { { LEFT | RIGHT | FULL } [ OUTER ] } } [ <join_hint> ] ]
    JOIN
```

Syntax 5.7: Konstrukce <joined_table>

První způsob, jak spojovat tabulkové zdroje, je vyžití `<join_type>`. V něm určujeme, jaký typ spojení chceme použít. Jsou čtyři typy `INNER`, `LEFT`, `RIGHT` a `FULL`. Dále je ke spojení potřeba specifikovat, které sloupce si musejí odpovídat. K tomu slouží klíčové slovo `ON` a podmínka je stejná jako v sekci 5.1.6. Spojení `INNER` je výchozí nastavení a nemusí se uvádět. Vrací pouze odpovídající si řádky z obou tabulkových zdrojů a ignoruje řádky, které si neodpovídají. Ostatní spojení jsou typu `OUTER` a toto klíčové slovo se nemusí uvádět. Spojení `LEFT` vrací všechny řádky z levého (prvního) tabulkového zdroje a odpovídající řádky z pravého (druhého). Podobně spojení `RIGHT` vrací odpovídající řádky z levého zdroje a všechny řádky z pravého zdroje. A nakonec spojení `FULL` vrací všechny řádky, které jsou odpovídající v levém nebo pravém zdroji. Také je možné použít jednu z nápověd pro optimalizátor dotazů. Jak lze takto spojovat je uvedeno v příkladu 5.4.

Druhým způsobem je využití `CROSS JOIN`, což je kartézský součin obou zdrojů.

Další způsobem je použití `APPLY`, který lze využít pokud pravý zdroj je tabulková funkce, která bere hodnoty z levého zdroje jako argumenty. Musí

se specifikovat, zda chceme použít `OUTER` nebo `CROSS`.

```
SELECT DISTINCT t.ID, m.*, td.Name AS DefinitionCode, t.Create_Date,
State = CASE
                WHEN t.State IN ('E', 'F')  THEN 'Failed'
                ELSE 'Succeed'
        END
FROM dciowner.Transactions t
LEFT JOIN dciowner.Transaction_Definitions td
        ON t.Transaction_Definition_ID = td.ID
LEFT JOIN (
        SELECT tm.Code AS ModuleCode, tma.Code AS AttributeCode,
                tm.Transaction_Definition_ID
        FROM dciowner.Transaction_Modules tm
        INNER JOIN dciowner.Transaction_Module_Attributes tma
                ON tm.ID = tma.Transaction_Module_ID
        WHERE tm.Code LIKE  'save%'
) as m
        ON td.ID = m.Transaction_Definition_ID
WHERE t.Create_Date > getDate() - 30
        AND ( td.Name LIKE 'TST%'
                OR td.ID IN (12, 23, 34, 45) )
        AND t.Updated_By_ID IS NOT NULL
ORDER BY DefinitionCode, t.Create_Date DESC
        OFFSET 0 ROWS
        FETCH NEXT 100 ROWS ONLY
```

Příklad 5.4: Komplexní ukázka dotazu SELECT

5.2 Výběr implementované funkčnosti

Zde zmíníme konstrukce dotazu, které prototyp sám využívá interně, nebo které je možné pomocí našeho prototypu specifikovat z uživatelského rozhraní. Dále popíšeme k čemu dané konstrukce slouží. Výsledný dotaz, což je výstup z prototypu, je validní dotaz `SELECT` a lze použít při přidávání sestavy do aplikace DCIx jak je uvedeno v sekci 5.3. Syntaxe dotazu použitá v prototypu je uvedena v syntaxi 5.8.

```
<SELECT statement> ::=
        SELECT { entity_alias.column_name  AS column_alias } [  ,...n ]
        FROM <entity_source>
<entity_source> ::= {
        entity entity_alias
        | <joined_entity>
}
<joined_entity> ::= {
        <entity_source> {INNER | LEFT OUTER} JOIN <entity_source>
                ON (<on_condition>)
}
<on_condition> ::= {
        entity_alias.column_name = entity_alias.column_name
        [ AND <on_condition> ]
}
```

Syntax 5.8: Prototyp dotaz SELECT

Nástroj pracuje s aliasy. Interně generuje automaticky aliasy pro připojené entity. Entitou může být tabulka nebo pohled jak je uvedeno v části o metadatech 6.1. Generování aliasů probíhá následovně. Bere se první volný výskyt písmen z jména entity, nebo pokud už je celé jméno entity použito přidávají se čísla na konec. Nad aliasy entit uživatel nemá v prototypu kontrolu. Proto momentálně není možné připojit jednu entitu vícekrát a vytvořit takzvaný self join. Naopak tomu je u sloupcových aliasů, které může libovolně volit a zadávat. Výchozí alias pro sloupec je složen z jména entity a jména sloupce oddělené mezerou.

Výběr do výsledku je to, co řídí prototyp. Uživatel pouze volí sloupce ze seznamu sloupců nabízených entit a prototyp na to reaguje. Podrobněji je uvedeno v uživatelské příručce v příloze B. Uživatel si pouze může volit dané sloupce a libovolně je pojmenovávat, jak bylo zmíněnu u aliasů.

Spojování je omezeno na `INNER JOIN` a `LEFT OUTER JOIN`. Typ spojení si uživatel vybírá ze select-boxu. Navíc v konstrukci `ON` mohou být pouze sloupce ze spojovaných entit vyhodnoceny přes rovnost `=` a spojeny přes spojku `AND`. Specifikují se v nastavení daného spojení pomocí check-boxů. Jaké sloupce si odpovídají a jaké entity lze spojovat je opět uvedeno v části o metadatech 6.1.

Pro zobrazení náhledu dat uživateli využíváme konstrukci `TOP`. Ta slouží k omezení počtu záznamů pro náhled na 25. Není použito pro výsledný dotaz.

Všechna ostatní syntax z dialektu MS SQL Serveru použita v prototypu není, protože by nadmíru znepřehlednila uživatelské rozhraní a pro prototyp je zanedbatelná. Její implementace je ale možná, pokud se navrhne rozumné mapování na uživatelské rozhraní a akce.

5.3 Přidání pohledu a sestavy do aplikace DCIx

Do aplikace DCIx lze přidat pohled a sestava pro čtení splněním čtyř kroků. Začneme nejprve založením nového databázového pohledu s využitím dotazu `SELECT`, který uživatel v prototypu vytvoří. Následně je potřeba najít část menu, kam budeme přidávat danou sestavu. A pomocí volání interní DCIx procedury `dciowner.addToMenu` přidat nový záznam. Poté je potřeba vytvořit výchozí pohled na danou sestavu. K tomu slouží interní procedura `dciowner.addIntoCollectionSettings`. A nakonec je nutné danou tabulku definovat v souboru `collection_collumns.xml`. Tyto kroky jsou již vztažené k samotné aplikaci DCIx, která není součástí práce, proto prototyp končí krokem, kdy uživatel vytvoří dotaz `SELECT`.

6 Spojování tabulek a metadata

V této kapitole si popíšeme, jak docílit toho, aby uživatel mohl pouze vybírat tabulky z jakých chce data čerpat a nástroj mu napovídal nejvhodnější spojení. V ideálním případě by uživatel vybíral pouze tabulky a spojování nechal na nástroji. K tomu je potřeba mít správně popsanou databázovou strukturu, nad kterou chceme operovat. Jednoduchou cestou, jak popsat databázovou strukturu jsou metadata. Proto v této kapitole uvedeme definici struktury metadat pro tento prototyp. Následně uvedeme, jak půjdou tato metadata využít pro spojování tabulek.

6.1 Struktura metadat

Pokud chceme uživateli nabídnout možnost procházení a spojování tabulek z databáze aplikace DCIx je potřeba mít její strukturu. Protože v DCIx jsou některé cizí klíče zrušené kvůli archivaci, není možné tyto informace získat rovnou z databáze. Proto bude potřeba tuto strukturu definovat mimo. Výhodou těchto metadat je, že si můžeme přesně specifikovat, jaké tabulky a pohledy budou pro daného zákazníka dostupné, jaké cizí klíče se mu budou nabízet a jaké uvidí sloupce. Tato metadata budou následně konfigurací (Config) našeho prototypu, jak je popsáno v části 7.2. Také implementací těchto metadat splňujeme požadavky 2, 4 a 5 ze sekce 3.1.

Jak je zmíněné v sekci 7.1 budeme definovat metadata jako JSON objekt. Celková konfigurace nástroje má strukturu jako v kódu 6.1. Skládá se tedy z mapy jednotlivých entit.

```
{
        "entities" : {
                "Entity1" : {...},
                "Entity2" : {...},
        ...}
}
```

Kód 6.1: Struktura metadata Config

Jednotlivé entity mají strukturu jako v kódů 6.2. Parametry `schemaUrl` a `tableUrl` představují odkazy do dokumentace DCIx, která je součástí prototypu. Parametr `referenceMap` představuje sadu odchozích vztahů a parametr `referredByMap` představuje sadu příchozích vztahů. Tyto vztahy jsou podrobněji popsány v odstavci s cizími klíči. Spojení mezi dvěma entitami

může být více a tím může být i více klíčů, proto jsme zvolili v těchto mapách pole pro jednotlivé entity.

```
{
        "name" : "Entity1",
        "schemaUrl" : "schemas/prodplan/prodplan.html",
        "tableUrl" : "tables/Entita1.html",
        "properties" : {
                "Property1": {...},
        ...},
        "referenceMap" : {
                "Entity2": [
                        {...},
                ...],
        ...},
        "referredByMap" : {
                "Entity3": [
                        {...},
                ...],
        ...}
}
```

Kód 6.2: Struktura metadata Entity

Jednotlivé vlastnosti patří pod entity. Mají strukturu popsanou v kódu 6.3. Obsahují parametr `dataType`, což je jeden z datových typů, které jsou popsány v sekci se statistikami 4.2. Dále je zde parametr `columnType`, což je datový typ přímo z databáze. Poté obsahují jméno statistiky, pokud jí daný sloupec má a značku, zda sloupec může být nulový. Nakonec parametr `enumContraints` představuje databázové omezení daného sloupce, jedná se o pole s povolenými hodnotami.

```
{
        "name" : "Property1",
        "columnType" : "nvarchar(60)",
        "dataType" : "string",
        "statisticName" : null,
        "notNull" : true,
        "enumConstraints" : [
                "Value1",
                "Value2"
        ]
}
```

Kód 6.3: Struktura metadata Property

Posledním objektem jsou cizí klíče. Patří pod danou entitu, ale jsou vztaženy i k cizí entitě a jejich struktura je popsána v kódu 6.4. Obsahují parametr lokální sloupec `localColumnName`, který specifikuje jméno sloupce dané entity. Dále obsahují parametr cizí sloupec `foreignColumnName`, který specifikuje jméno sloupce cizí entity. Definují příchozí a odchozí vztahy. Příchozí vztah je takový, kde je lokální sloupec primární klíč dané entity a cizí sloupec je sloupec z cizí entity. Odchozí stav je takový, že lokální sloupec je z dané entity a cizí sloupec je primární klíč cizí entity. Každý vztah lze

ohodnotit, jakou bude mít při výběru a spojování přednost. K tomu slouží parametr `weight`. To je z důvodu expertního spojování, které je popsáno v následující části 6.2.

```
{
        "localColumnName" : "ColumnName1",
        "foreignColumnName" : "ColumnName2",
        "weight" : 1.0
}
```

Kód 6.4: Struktura metadata ForeignKey

6.2 Expertní spojování tabulek

Tento problém lze převést na procházení grafu a hledání cesty. Entity jsou uzly a cizí klíče jsou hrany. Pomocí metadat si chceme definovat všechna možná spojení mezi entitami, ale některá spojení mohou být vhodnější než jiná. Nejlepší bude ukázat to na příkladu. Entita Warehouses je spojena s Warehouse_Position a Positions a entita Warehouse_Position je spojena s Position. Pokud by uživatel do výběru přidal vlastnost z Warehouses a následně vlastnost z Positions, naivní přístup by našel nejkratší cestu (existuje přímé spojení - výchozí pozice pro sklad) a tu vrátil uživateli. Jenže to ve většině případů nebude ta správná cesta, protože obvykle se nedotazujeme na výchozí pozici skladu. Ale je zde entita Warehouse_Position a ta je spojovací a logicky správná. Druhý přístup je najít a vrátit všechny cesty. To je s databází velikosti DCIx a množstvím vazeb a zacyklení nevhodné. Dnešní počítače jsou schopné všechny cesty po chvíli najít, ale trvá to dlouho a nechceme uživateli zobrazovat cesty o 30 entitách.

Jak z příkladu vyplývá, je potřeba vytvořit kompromis mezi nejkratší cestou a všemi cestami. Kompromis jsme zvolili jako nalezení nejkratší cesty a nalezení všech cest, které mají maximální délku menší než je délka nejkratší zvýšená o dva. Navíc jsme hledání nejkratší cesty vylepšili. Vylepšením je, že vazby mezi entitami ohodnotíme váhami. Díky tomu můžeme zvýhodňovat například vazební tabulky. Také lze zvýhodňovat některé klíče mezi dvojicí entit před ostatními z této dvojice. K nalezení nejkratší cesty používáme Dijkstrův algoritmus [34]. K nalezení ostatních cest upravené prohledávání do šířky [34] s omezenou maximální délkou.

Nyní uživateli můžeme nabídnout nejvhodnější cestu na prvním místě s nejvhodnějším cizím klíčem. Pokud by chtěl zvolit jinou cestu, má v nabídce další klíče a rozumně dlouhé cesty. Toto napovídání lze plně ovlivňovat metadaty, která obsahují vazby s váhami.

7 Výběr technologií a architektura prototypu

V této kapitole si uvedeme, jaké technologie jsme zvolili k implementaci prototypu a proč. Následovat bude návrh architektury prototypu.

7.1 Výběr technologií

Pro správu aplikace je použit nástroj Maven [11] ve verzi 3.3.9. Je to z toho důvodu, že je velmi rozšířený a umíme v něm pracovat. Alternativou mohl být nástroj Gradle.

Je zde pouze jedna technologie, která je vynucena aplikací DCIx a to je databázový server MS SQL Server 2014. Na tomto serveru se nachází databáze DCIx. Jako databázi pro ukládání pohledů je použita databáze PostgreSQL [18] ve verzi 9.6.8. Bylo potřeba vybrat jednoduchou open-source databázi, kterou jsme znali. Alternativou mohla být databáze MySQL.

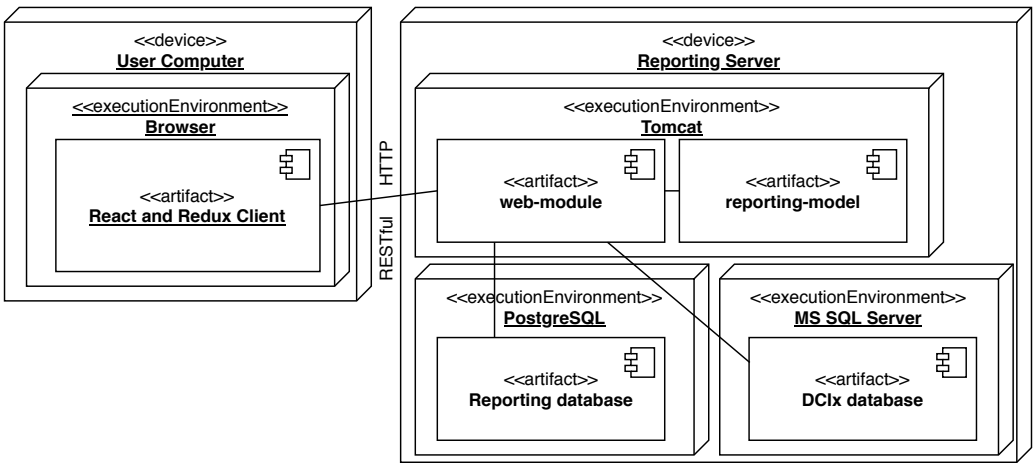
Pro serverovou část je použit jazyk Java, protože je v něm napsána aplikace DCIx a máme jeho dobrou znalost. Dále je použit framework Spring [24] a pro přístup k datům v databázích poté ORM framework Hibernate [7] verze 5.2.9. Jedná se o rozšířené nástroje, se kterými umíme pracovat a usnadňují nám práci.

Pro klientskou část je použit JavaScriptový framework React [19] a Redux [21]. React je komponentový framework pro tvorbu uživatelského rozhraní. Alternativou byl framework AngularJS, ale s tím jsme neměli takové zkušenosti, jako s Reactem a vývoj by probíhal pomalu. Redux jsme zvolili, protože slouží k udržení jednotného stavu pro celou aplikaci, což je u single-page aplikací důležité. S využitím frontend-maven-plugin [6] lze používat registr balíčků NPM a také nástroj Webpack [26] pro správu JavaScriptů. Webpack nám umožňuje kompilaci z nové verze JavaScriptu ES2015 do aktuální používané, tvorbu balíků nebo minifikaci.

Naše metadata musejí být strukturovaná, proto přicházejí v úvahu dva rozšířené formáty, v jakých je uchovávat. Prvním je XML dokument a druhým je dokument zapsaný ve formátu JSON. Pro naše použití je jejich funkčnost identická a zvolili jsme JSON z důvodu lepší čitelnosti a použití JavaScriptu na klientské části.

7.2 Architektura prototypu

S vybranými technologiemi souvisí i architektura prototypu. Celá práce se skládá ze tří Maven artefaktů. Je to z toho důvodu, že prvním artefaktem je *reporting-model* a zbylé dva artefakty ho používají. Jedná se o soubor tříd, které představují strukturu metadat. Další artefakt je *cfg-generator*, což je generátor konfigurace pro prototyp. Součástí konfigurace jsou metadata získaná z dokumentace DCIx. Posledním artefaktem je *web-module*, což je samotný prototyp a to je samostatná webová single-page aplikace. Jak jsou jednotlivé části nasazeny je zobrazeno v obrázku 7.1.



Obrázek 7.1: Diagram nasazení prototypu

7.2.1 Artefakt reporting-model

Tento artefakt představuje model metadat v této práci. Celý soubor metadat pro náš prototyp nazýváme konfigurací. Je to čtveřice tříd, která odpovídá struktuře metadat v sekci 6.1.

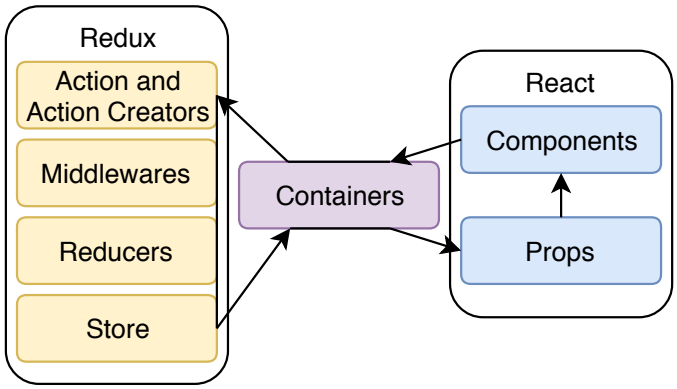
7.2.2 Artefakt cfg-generator

Tento artefakt je vedlejším produktem této práce. Bylo potřeba jednoduše získat metadata databáze DCIx, proto jsme vytvořili generátor těchto metadat z dokumentace DCIx. Obsahuje pouze jednu třídu, která vytvoří JSON soubor ze statické HTML dokumentace DCIx. V té jsou diagramy popsány i se zrušenými cizími klíči. Metadata získáváme s využitím XPath.

7.2.3 Artefakt web-module

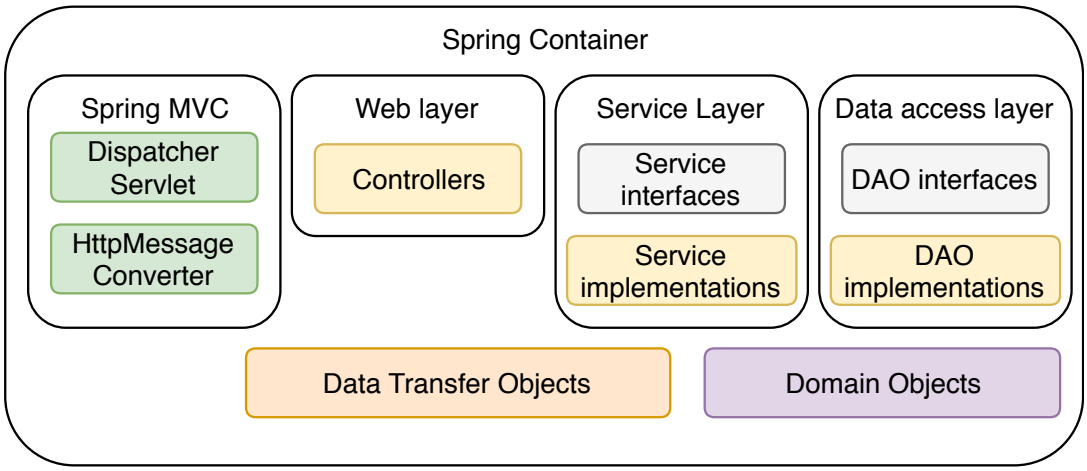
V tomto artefaktu se nachází prototyp aplikace. Nachází se zde dvě databáze. Dvě jsou z toho důvodu, že první je databáze aplikace DCIx, která je pouze pro čtení. Jeden z požadavků je, aby si uživatel mohl daný pohled uložit, proto je potřeba druhé databáze. Jediné co se v ní nachází je struktura pro ukládání uživatelských pohledů.

Dále zde máme klientskou část neboli front-end, která je postavena kolem frameworků React a Redux, které vynucují vlastní architekturu, která znázorněna na obrázku 7.2.



Obrázek 7.2: Architektura React a Redux

Nakonec zde máme serverovou část neboli back-end, která je postavena kolem Springu a Hibernate, takže také vynucuje vlastní architekturu, která je znázorněna na obrázku 7.3. Spolu s klientskou částí komunikují pomocí REST rozhraní [22].



Obrázek 7.3: Zjednodušený Spring kontejner

8 Funkcionalita prototypu

V této kapitole se podrobně podíváme na implementovanou funkcionalitu našeho prototypu. Tím je artefakt *web-module*, který rozdělíme na dvě hlavní části. První je serverová část. V té si popíšeme, její datovou a logickou vrstvu spolu s vystaveným RESTful rozhraním . Druhou je klientská část. Ta lze rozdělit, stejně jako je rozdělena obrazovka aplikace, na šest hlavních částí, kde každá má svoji specifickou funkci.

8.1 Serverová část

Při nastartování serverové části proběhne načtení konfigurace z připojeného konfiguračního souboru *reporting-config.json* do paměti. Následně je konfigurace validována vůči připojené databázi DCIx. To je z důvodu zamezení vzniku chyb, kdy jsou v konfiguraci entity, které se v dané databázi nenacházejí. Tyto entity se z konfigurace vyjmou. Dále je potřeba kontrolovat a odebírat cizí klíče. To z toho důvodu, že pokud validace odstraní entitu, ostatní na ni mohou mít vazbu a vznikla by další chyba.

Po validaci se v konfiguraci nastaví odkazy do dokumentace DCIx na aktuální správnou hodnotu. Protože se v konfiguraci nachází pouze relativní cesta, tak se přidává základ absolutní cesty, který se nastavuje v *application.properties*. Proto lze docílit, že dokumentace klidně může být na jiném serveru a odkazy budou fungovat.

Nakonec konfiguraci upraví inicializace služby statistik, která je popsána v části 8.1.2.

8.1.1 Datová vrstva

V této části popíšeme databáze a jaké se k nim pojí třídy a dotazy. Protože máme dvě databáze, zkomplikovalo se nastavení frameworku Hibernate, ale vyřešili jsme to použitím dvou transakčních managerů. První slouží k připojení do MS SQL Server databáze DCIx *dci-data* a nastavuje se v konfigurační třídě `DataDCIConfiguration`, následně stačí použít ve službách anotaci `@Transactional(value="dciTransactionManager")`. Druhý slouží k připojení do PostgreSQL databáze *reporting*, což je nová databáze pro ukládání pohledů. Nastavuje se v konfigurační třídě `DataReportingConfiguration` a anotaci má `@Transactional(value="reportingTransactionManager")`. Mapování result-setů na třídy pro native query frameworku Hibernate jsme

vytvořili v souboru *hibernate-querymappings.xml*. Native query umožňuje provádění libovolných dotazů oproti využití Java Persistence API (JPA) nebo standardnímu dotazování pomocí Hibernate Query Language (HQL). To se nám hodí, protože chceme používat speciální dotazy pro engine MS SQL Server. Nevýhodou je, že jejich použití znemožňuje přenositelnost, která je velkou výhodou frameworku Hibernate.

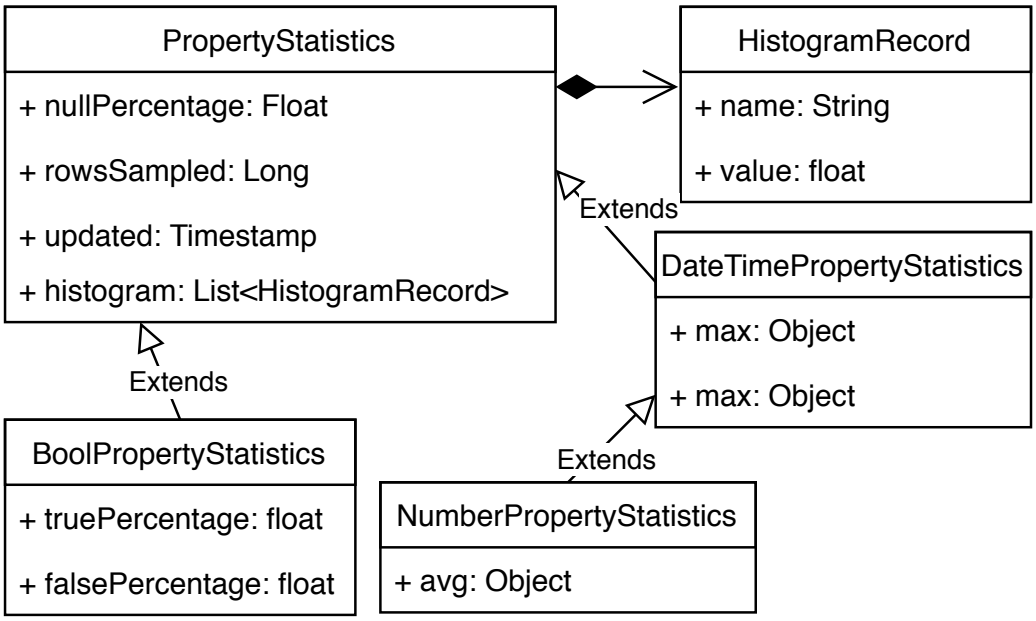
Dále v prototypu máme vrstvu pro přístup do databáze (Data Access Layer), která využívá již nakonfigurovaný framework Hibernate a obsahuje anotované `@Repository` rozhraní a třídy. V třídě `StatisticsDAOImpl` se nacházejí dotazy do enginu databáze spjaté se statistikami a jsou zmíněny níže u jejich použití. Dále je zde třída `PreviewDAOImpl`, která slouží k získání náhledu skládaného dotazu z databáze DCIx.

Nejprve popíšeme novou PostgreSQL databázi pro ukládání pohledů. Požadavek 7 na nástroj ze sekce 3.1 definuje možnost ukládat pohledy, které si uživatel vytvoří. K tomu vznikla třída a tabulka `CustomQuery`. To je jediná entita v aplikaci, pak je zde skupina mapovacích tříd pro native query dotazy a ostatní datové třídy slouží jako jednoduché Data Transfer Object (DTO) třídy. `CustomQuery` obsahuje typické informace o historii (kdo, kdy upravil nebo vytvořil záznam) a značku, zda je uložený dotaz validní. Tato značka je tam z důvodu změny konfigurace a vzniku nevalidního uloženého stavu. Také obsahuje parametry nastavení daného dotazu. Jedná se o JSON obraz důležitých částí stavu z klientské části. To umožňuje jednoduché opětovné nahrání a obnovení stavu v klientské části.

Dotaz má i svůj předpis jako třídu `QueryParameters`, která obsahuje seznam vybraných sloupců `Columns` a parametrů spojení `JoinParameters`. Třída `Columns` představuje nastavení sloupce. Obsahuje informace jako jsou jméno, alias, jméno vlastnosti, jméno entity a datový typ. Podobně třída `JoinParameters` představuje jednotlivá spojení mezi entitami. Zde jsou informace jako zvolená cesta, seznam ostatních cest, seznam cizích klíčů pro zvolenou cestu a seznam typů spojení pro zvolenou cestu.

Nyní se dostáváme k databázi aplikace DCIx. Z této databáze chceme získávat statistiky zmíněné a popsané v kapitole 4. K tomu jsme vytvořili třídy, které představují celý statistický objekt pro daný typ sloupce. Také je potřeba reprezentovat jednotlivé položky histogramu, k čemuž slouží třída `HistogramRecord`. Tyto třídy jsou popsány v diagramu tříd 8.1.

Pro získání statistických dat z databáze bylo potřeba vytvořit mapovací třídy pro framework Hibernate. Využili jsme znalostí získaných v části 4.3.2. Třída `ConstraintQueryRow` slouží k získání integritních omezení `CHECK` z databáze. Představuje objekty, jejichž seznam vrací SQL dotaz v kódu 8.5. Třída `HistogramQueryRow` slouží k získání dat pro histogram. Dotaz pro zís-



Obrázek 8.1: Diagram tříd statistik a informací

kání sloupců histogramu je v kódu 8.6. Dále třída `StatisticHeaderQueryRow` slouží k získání hlavičky histogramu. Její dotaz je uveden v kódu 8.7. Nakonec třída `StatisticsQueryRow` je využita při inicializaci statistické služby a zajišťuje mapování vlastnosti entity na její statistiku. Dotaz pro získání těchto mapování je v kódu 8.8.

```
SELECT o.name AS table_name,
       c.name AS column_name,
       cc.definition AS constraint_definition
FROM sys.check_constraints cc
       INNER JOIN sys.columns c ON cc.parent_column_id = c.column_id
              AND cc.parent_object_id = c.object_id
       INNER JOIN sys.objects o ON cc.parent_object_id = o.object_id
```

Kód 8.5: Dotaz na získání integritních omezení CHECK

```
DBCC SHOW_STATISTICS (:entitySchemedName, :statisticName ) WITH HISTOGRAM
```

Kód 8.6: Dotaz na získání řádek histogramu

```
DBCC SHOW_STATISTICS (:entitySchemedName, :statisticName ) WITH STAT_HEADER
```

Kód 8.7: Dotaz na získání hlavičky statistik

Dalším nestandardním dotazem, který používá Native Query, je odhadnutí počtu záznamů v tabulce. O odhad se jedná z důvodu výkonnostních problémů příkazu `COUNT`. Jak tento odhad získat je uvedeno v kódu 8.9. DTO pro tento počet záznamů v tabulce je třída `RowCount`.

```
SELECT c.name AS column_name, s.name AS statistic_name
FROM sys.stats AS s
INNER JOIN sys.stats_columns AS sc
        ON s.object_id = sc.object_id AND s.stats_id = sc.stats_id
INNER JOIN sys.columns AS c
        ON sc.object_id = c.object_id AND c.column_id = sc.column_id
WHERE s.object_id = OBJECT_ID(:entitySchemedName)
        AND sc.stats_column_id = 1
```

Kód 8.8: Dotaz na získání mapování vlastností na jejich statistiky

```
SELECT CAST(p.rows AS bigint) AS count
FROM sys.tables AS tbl
INNER JOIN sys.indexes AS idx
        ON idx.object_id = tbl.object_id and idx.index_id < 2
INNER JOIN sys.partitions AS p
        ON p.object_id=CAST(tbl.object_id AS int) AND p.index_id=idx.index_id
INNER JOIn sys.schemas AS s
        ON s.schema_id = tbl.schema_id
WHERE tbl.name=:entityName
        AND s.name=:schemaName
```

Kód 8.9: Dotaz na získání přibližného počtu záznamů tabulky

Nakonec je zde skupina tříd, která spadá do části vytváření dotazu a používáme ji při komunikaci s klientskou částí. Jedná se o třídy `ColumnRequest`, `ColumnResponse`, `CustomQueryRequest`, `SqlAndPreviewRequest`, `SqlAndPreviewResponse` a `ErrorResponse`. Tyto třídy obvykle obsahují již popsané základní stavební kameny `JoinParameters` a `Column` a podrobněji jsou popsány v následující části u jednotlivých koncových bodů API rozhraní.

8.1.2 Logická vrstva

V této části jsou rozebrány třídy, které jsou spojené se službami a kontrolery. Nejprve začneme kontrolery a následně si popíšeme, jaké služby využívají. Kontrolery v prototypu máme pouze dva. Prvním kontrolerem je třída `ReportingController` a slouží k obsluze dotazů, které nejsou na statické zdroje nebo na RESTové API. Je to vstupní bod do aplikace, vrací prázdnou HTML stránku s přiloženými CSS styly a JavaScript balíky. Druhým kontrolerem je `ApiController` a ten obsahuje koncové body pro RESTové API. Také obsahuje dvě obsluhy výjimek, které mohou vzniknout validací ve službách.

- *GET /entities* vrací kolekci všech entit `Collection<Entity>` z konfigurace.
- *GET /entities/{entityName}/stats/rowcount* vrací počet záznamů pro danou entitu jako objekt `RowCount`.

- *GET /entities/{entityName}/properties/{propertyName}/stats* vrací statistický objekt pro danou vlastnost a entitu `PropertyStatistics`. Pro neexistující statistiku vrací HTTP odpověď se statusem `NO_CONTENT`.

- *POST /builder/sqlandpreview* je výjimka, nevytváří perzistentní data na serveru, ale použití metody GET by bylo s ohledem na posílaná data nevhodné. Parametrem dotazu je objekt `SqlAndPreviewRequest`, který obsahuje seznam sloupců `Column` a seznam parametrů spojů `JoinParameters`. Odpovědí je objekt `SqlAndPreviewResponse`, který obsahuje vygenerovaný SQL dotaz a data náhledu.

- *POST /builder/queries* slouží k ukládání uživatelského dotazu. Parametrem je `CustomQueryRequest`, který opět obsahuje seznam sloupců `Column` a seznam parametrů spojů `JoinParameters` a jméno dotazu. Vrací URL dotazu, kde se nachází uložená entita s HTTP statusem `CREATED`.

- *GET /builder/queries* vrací pro aktuálního uživatele seznam uložených dotazů `List<CustomQuery>`.

- *PUT /builder/queries/{id}* slouží k úpravě daného uloženého dotazu. Parametrem je jako u metody POST `CustomQueryRequest`.

Nyní se podíváme na služby, které máme v prototypu tři. První službou je `StatisticsService`. Využívají ji koncové body začínající na */entities*. Naopak ona využívá `dciTransactionManager` a `StatisticsDAO`. Jak název napovídá, stará se o práci se statistikami. Obsahuje metodu `init`, která při startu provede mapování statistik na vlastnosti entit a obohatí tím konfiguraci. Dále obsahuje metodu pro získání počtu záznamů dané entity `getEntityRowCount`. Nejzajímavější je metoda `getStatistics`, která pro danou entitu a vlastnost vytvoří a naplní odpovídající objekt z hierarchie `PropertyStatistics`. Nejprve se provede validace a zjistí se typ vlastnosti. Podle toho již metoda z konfigurace a pomocí již zmíněných dotazů na statistiky získá a doplní informace do odpovídajícího statistického objektu.

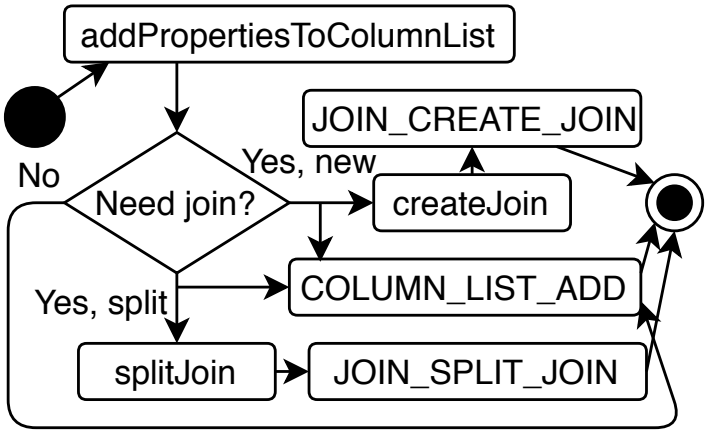
Druhou službou je `builderService`. Využívají ji koncové body začínající na */builder* a ona využívá `dciTransactionManager`, `reportingTransactionManager`, `UserService`, `PreviewDAO` a `CustomQueryDAO`. Obsahuje metodu pro získání dotazu a dat náhledu `getSqlAndPreview` a to je jádro této služby, neboť zde probíhá generování dotazu SQL z přijatých parametrů. Jak název napovídá z `JoinParamters` se generuje konstrukce `FROM` a z `Columns` konstrukce `<select-list>`. Výsledný dotaz s omezením na počet záznamů se rovnou provede pro získání náhledu. Dále jsou zde metody `saveCustomQuery`

a `getAllCustomQueriesForCurrentUser`. Ty se starají o správu vytvořených uživatelských dotazů.

Poslední službou je `UserService`. Jelikož máme pouze prototyp, chybí plnohodnotná správa uživatelů. Ta by v ideálním případě měla spolupracovat se správou uživatelů aplikace DCIx. Nyní zde metoda `getCurrentUserId` vrací konstantu 1.

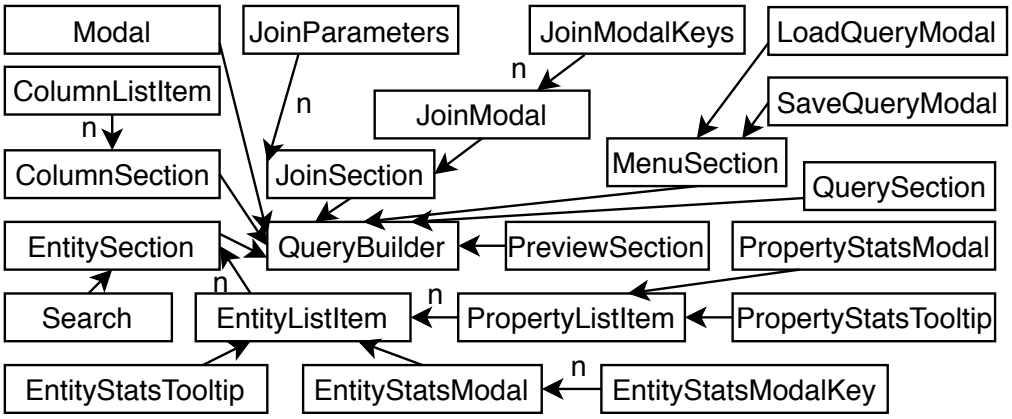
8.2 Klientská část

Zde si popíšeme jednotlivé části, na které se dělí uživatelská obrazovka. Tomu přibližně odpovídají React komponenty pojmenované jako sekce. Každá sekce má své části vynucené Reduxem. To jsou container, reducer, actions a action creators. Container zaručuje spojení mezi React komponentou a Redux store a action creators. Kde action creator slouží k vytvoření jedné nebo více akcí. Jak mohou akce a jejich creatory spolu spolupracovat je vidět na obrázku 8.2, který znázorňuje, jak voláním jednoho action creatoru *addPropertiesToColumnList* lze vytvořit více akcí. Koncem je zde myšleno zpracování akce v příslušném reduceru, protože v tom již žádné další akce vznikat nemají. Jedná se o šťastný případ, bez zobrazených reakcí na chyby. Reducer sekce zpracovává akce a upravuje stav sekce ve store. Může reagovat i na akce z jiných sekcí.



Obrázek 8.2: Příklad action a action creators

Také je zde popsána hierarchie komponent pomocí obrázku 8.3. Šipka představuje vnoření komponenty a hrot ukazuje na nadřazenou komponentu. Uvedení písmena *n* u šipky znamená, že nadřazená komponenta má v sobě kolekci podřazených.



Obrázek 8.3: Hierarchie komponent prototypu

Kromě sekcí je zde vstupní bod aplikace komponenta `App`, která nyní obsahuje pouze šablonu celého prototypu. Tou je komponenta `QueryBuilder`, která má svůj vlastní kontejner a ten se stará o správu modálních oken. Akce s modálními okny používají action creators napříč celou aplikací.

Dále je v klientské částí hlavní Redux reducer, který spojuje reducery jednotlivých komponent. Také je zde konfigurace Redux store, která uchovává stav celé aplikace.

Nakonec jsou zde pomocné části ve složce *utils*, jsou to části, které jsme použili vícekrát a nemají jasnou příslušnost. Patří sem komponenty, které řeší načítání obrazovky, notifikace nebo odchytávání kliknutí.

8.2.1 Entity

V této části jsou statistiky a konfigurace. Stručně si popíšeme komponenty, které sem patří.

`EntitySection` slouží jako kontejner. `Search` obsahuje pole pro vyhledávání nad entitami a jejich vlastnostmi a jeho změnový event volá action creator *searchEntityList*. `EntityListItem` zobrazuje jednotlivé entity. Obsahuje tlačítko pro přidání všech jejích vlastností do dotazu, k tomu využívá volání action creator *addPropertiesToColumnList* ze sloupcové sekce 8.2.2. `EntityStatsTooltip` je náhled na danou entitu, zobrazuje pouze základní informace. To jsou přibližný počet záznamů v entitě a informaci o tom, jak je entita spojena se svým okolím. Zobrazují se maximálně tři cizí entity od každého typu kvůli přehlednosti. Volá action creator *getEntityRowCount*. `EntityStatsModal` je detail statistik a informací pro danou entitu. Také obsahuje i přibližný počet záznamů a navíc jsou zde odkazy do dokumentace. Dále obsahuje seznam entit, které na danou entitu odkazují a seznam

entit, na které tato entita odkazuje. Volá action creator *getEntityRowCount*. `EntityStatsModalKey` zobrazuje cizí entitu a popisuje cizí klíče mezi cizí entitou a danou entitou.

`PropertyListItem` zobrazuje jednotlivé vlastnosti dané entity. Obsahuje tlačítko pro přidání vlastnosti do dotazu, které volá action creator *addPropertiesToColumnList*. `PropertyStatsTooltip` je náhled na danou vlastnost. Zobrazuje pouze její typ, zda má omezení na prázdnou hodnotu nebo pokud se jedná o výčtový typ i povolené hodnoty. Nakonec `PropertyStatsModal` je detail statistik a informací pro danou vlastnost. Ke všem druhům statistik a informací se chováme stejně. Pokud je ve statistickém objektu přítomna, tak je následně i zobrazena. K vykreslení histogramu používáme knihovnu Recharts [20]. Volá action creator *getEntityPropertyStats*. Pokud již není zjištěn počet ve storu, tak i *getEntityRowCount*.

Nyní stručně popíšeme akce, které jsou s touto sekcí spojené. *ENTITY_LIST_FETCH* načte konfiguraci ze serveru. *ENTITY_LIST_SEARCH* mění filtr zobrazených entit z konfigurace. *ENTITY_STATS_ROWCOUNT_FETCH* načte do konfigurace počet záznamů pro danou entitu ze serveru. *ENTITY_PROPERTY_STATS_FETCH* načte do konfigurace statistiku pro danou vlastnost ze serveru.

8.2.2 Sloupce

Tato část se týká přidaných sloupců do dotazu a obsahuje následující komponenty.

`ColumnSection` slouží jako kontejner. `ColumnListItem` zobrazuje jednotlivé přidané sloupce. Obsahuje tlačítko pro odebrání z dotazu, které volá action creator *removeColumnFromColumnList*. Také obsahuje pole pro změnu popisku, které umí volat action creator *editColumnTitle*.

Sloupcové akce jsou následující. *COLUMN_LIST_ADD* zajišťuje přidání sloupce do dotazu. Její creator rozlišuje, zda je již entita přidávané vlastnosti v dotazu a provede se jen jednoduché přidání, nebo zda bude potřeba rozpojit existující spojení pomocí action creatoru *splitJoin* ze sekce spojení 8.2.3. Případně vytvořit úplně nové pomocí action creatoru *createJoin* ze sekce spojení . *COLUMN_LIST_REMOVE* zajišťuje odebrání vlastnosti z dotazu. Také rozlišuje, zda lze vlastnost jednoduše odebrat, protože daná entita má ještě jiné vlastnosti v dotazu, nebo zda bude potřeba danou entitu odebrat a zrušit spojení, která jsou na ni navázána, pomocí action creatoru *removeEntity* ze sekce spojení.

8.2.3 Spojení

V této sekci se nachází definice spojení, které jsou v dotazu přítomna. Obsahuje následující komponenty.

`JoinSection` slouží jako kontejner. `JoinParameters` zobrazuje jednotlivá spojení, které jsou přidány v dotazu. Spojení mají formu stromu. Každé spojení obsahuje tlačítko pro jeho editaci volající action creator *openJoinEdit*. *JoinModal* zobrazuje detailní nastavení daného spojení. Slouží pro editaci nebo tvorbu nového spojení. Jsou zde select-box pole pro výběr cesty napojený na action creator *selectJoinPath*. Případně zde může být pole pro výběr místa připojení entity navázané na action creator *selectJoinStart*. Dále jsou zde pro každou část aktuálně vybrané cesty definované oblasti, které obsahují další select-box pole s výběrem typu spojení. *JoinModalKeys* jsou také součástí oblasti každé části vybrané cesty a slouží k zobrazení a vybrání cizích klíčů pro danou část cesty.

Nyní se dostáváme k dostupným akcím pro spojování. *JOIN_SPLIT_JOIN* provede rozpojení dvou nebo více existujících spojení, která uvnitř cesty obsahují přidávanou entitu. Tato akce nemění výsledný dotaz, pouze vnitřní a grafickou reprezentaci v nástroji. Z jednoho objektu `JoinParamters` vznikají dva, kde u jednoho je nová entita koncová a u druhého výchozí. *JOIN_CREATE_JOIN* je jádro aplikace. Zde vznikají nová spojení. Stačí k tomu konfigurace, výchozí a koncová entita. Implementovali jsme kroky popsané v části 6.2 pro získání nových `JoinParamters`, které jsou uživateli nabídnuty v modálním okně. *JOIN_EDIT_SAVE* slouží k uložení parametrů z modálního okna. *JOIN_EDIT_SELECT_PATH* slouží k výběru cesty ze seznamu nabízených cest. *JOIN_EDIT_JOIN_START* umožňuje nastavit entitu, ke které budeme stávající, nebo novou entitu připojovat. Ovlivňuje tím tedy nabídku nabízených cest. *JOIN_REMOVE_ENTITY* provede odebrání celé větve navázané na danou entitu.

8.2.4 Dotaz

Toto je jednoduchá sekce, které slouží k zobrazení SQL dotazu. Obsahuje pouze jednu komponentu `QuerySection`, která zobrazuje složený dotaz SQL jako text.

Také má pouze jednu akci *QUERY_UPDATE_SQL_AND_PREVIEW*, která ze serveru pošle parametry a sloupce a jako odpověď získá vygenerovaný SQL dotaz a data pro náhled.

Zajímavý je reducer, který odchytává akce ostatních sekcí, které mají vynutit aktualizaci náhledu a SQL dotazu. Jeho reakcí na tyto akce je na-

stavení vlajky *dirty* sekce, která se dostane do *props* React komponenty a z té se nakonec vynutí akce *QUERY_UPDATE_SQL_AND_PREVIEW*. Tím je dodržen jednosměrný tok dat z obrázku 7.2.

8.2.5 Náhled

Náhled je také jednoduchá sekce. Opět obsahuje pouze jednu komponentu `PreviewSection`, která z přijatých dat vytvoří tabulku s hlavičkou s jmény sloupců a řádky s daty. Nemá žádnou akci, pouze její reducer zpracovává data z akce *QUERY_UPDATE_SQL_AND_PREVIEW*.

8.2.6 Menu

Poslední sekcí je nabídka. Obsahuje tři komponenty.

`MenuSection` slouží jako nabídka s tlačítky pro ovládání aplikace a případně zobrazuje jméno dotazu. Obsahuje tlačítko pro vyčistění dotazu, které volá action creator *clear*. Dále je zde tlačítko pro vytvoření nového dotazu, které je napojené na action creator *newQuery*. Tlačítko pro načtení dotazu, je napojené na *loadAllQueries* a otevření modálního okna. Nakonec tlačítka pro uložení otevírají modální okno. `SaveQueryModal` je jednoduché modální okno pro uložení dotazu. Obsahuje pole pro zadání jména dotazu a tlačítko, které volá action creator *save*. `LoadQueryModal` je modální okno pro zobrazení všech uložených dotazů daného uživatele v tabulce, klik na řádku volá action creator *loadQuery*.

Menu akce jsou následující. MENU_NEW_QUERY vytvoří nový dotaz, její creator k tomu také použije *clear*. *MENU_CLEAR* slouží k vyčistění celého dotazu. Reagují na ní reducery ostatních sekcí a nastavují svá interní data do výchozího stavu. *MENU_LOAD_ALL_QUERIES* načítá ze serveru všechny uložené dotazy daného uživatele. *MENU_LOAD_QUERY* je pokyn pro nahrání uloženého dotazu. Nastavení ohledně dotazu je součástí akce a reagují na ni reducery ostatních sekcí. *MENU_SAVE* vezme potřebná data pro nastavení aplikace a pošle je na server. Pokud je odpověď v pořádku zavolá se ještě akce *SHOW_SAVED_OK*, která slouží k notifikaci uživatele.

9 Ověření aplikace

Ověření aplikace rozdělíme na dvě části. První část se týká zobrazovaných statistik a informací. Ukážeme, jak se aplikace zachová pro různé definované datové typy a pro entity. Druhou části je ověření tvorby dotazu. To ověříme na několika definovaných scénářích.

9.1 Ověření statistik a informací

Zde ověříme zobrazování statistik a informací v aplikaci, což je první požadavek na nástroj ze sekce 3.1. Máme dva typy zobrazení a těmi jsou náhled a detail. Jak jednotlivé typy zobrazit je popsáno v uživatelské příručce v příloze B.

9.1.1 Informace a statistiky nad entitami

Podobně jako v kapitole 4 začneme statistikami nad entitami. Náhled je zobrazen na obrázku A.2. Pro úplnou informaci použijeme detailní pohled. Ten je zobrazen na obrázku A.1. Obsah náhledu i detailu je popsán v předešlé části 8.2.1 a z obrázků je patrné, že obsahují všechny náležitosti.

9.1.2 Informace a statistiky nad vlastnostmi entit

Zde se budeme zabývat informacemi a statistikami, které jsou vztažené k vlastnostem entit. Náhled u vlastností, který je na obrázku A.3a, zobrazuje pouze její typ, zda má omezení na prázdnou hodnotu nebo pokud se jedná o výčtový typ i povolené hodnoty. Detailní pohledy obsahují informace navíc. Prvním údajem je z kolika řádků byla statistika počítána. Také je zde, kdy toto počítání naposledy proběhlo. Na obrázku A.3b jsou vidět pohledy pro logickou hodnotu, omezený řetězec a řetězec, který má tolik rozdílných hodnot, že histogram nemá smysl. Následuje obrázek A.6, ve kterém je zobrazen sloupec typu časového údaje. Nakonec je zde obrázek A.7, který má číselný typ. Jak je z obrázků vidět, informace definované v části 4.2 pro jednotlivé sloupce zobrazujeme a rozlišujeme.

9.2 Ověření tvorby dotazu

V této části na třech scénářích ověříme naše expertní spojování entit a tvorbu dotazu. Scénáře začínají vždy ve výchozím stavu aplikace, to znamená, že nejsou vybrány žádné vlastnosti ani entity a dotaz je prázdný.

9.2.1 Scénář netriviálního spojení

Prvním scénářem bude spojení dvou entit, které mají mezi sebou mnoho možných spojení, ale preferované spojení není to nejjednodušší. Jedná se o entity skladu a položky. Mějme následující kroky a očekávané chování aplikace.

1. Klikneme na přidání vlastnosti `Code` z entity `Warehouses`.

   V sekci s vybranými vlastnostmi se objeví nová vlastnost *Warehouse Code*, v sekci s dotazem je zobrazen jednoduchý dotaz pro výběr dat. Dále je zobrazen náhled, který obsahuje jeden sloupec s kódy skladů.

2. Klikneme na přidání vlastnosti `Code` z entity `Products`.

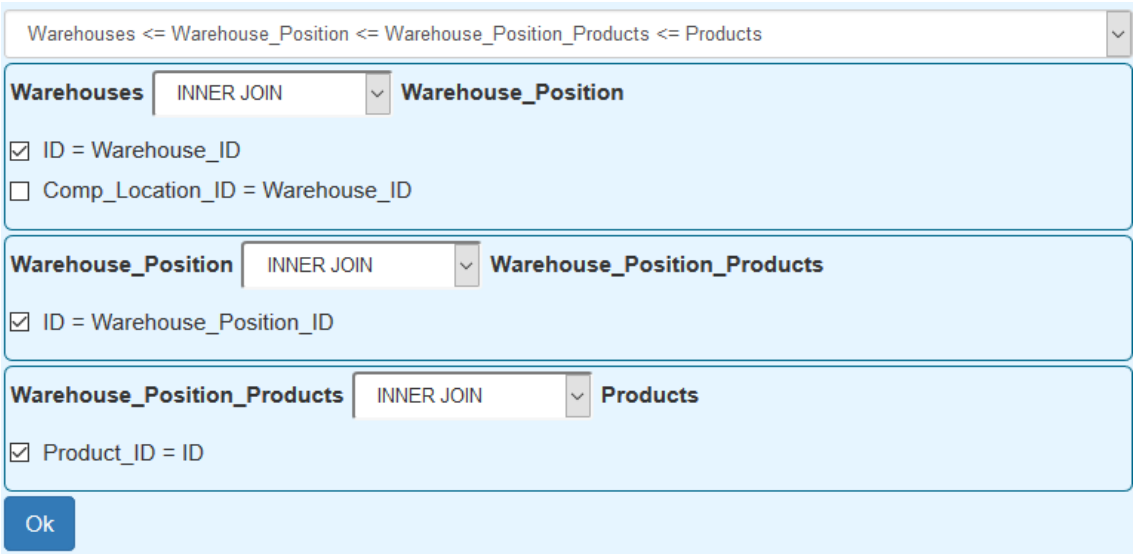
   Aplikace nám nabídne její návrh, jak je vhodné dané entity spojit. Máme možnost si zvolit i jinou cestu spojení, zvolit typ spojení a případně vybrat klíče v nabízeném spojení. Toho nyní nevyužijeme.

3. Potvrdíme nabízené spojení.

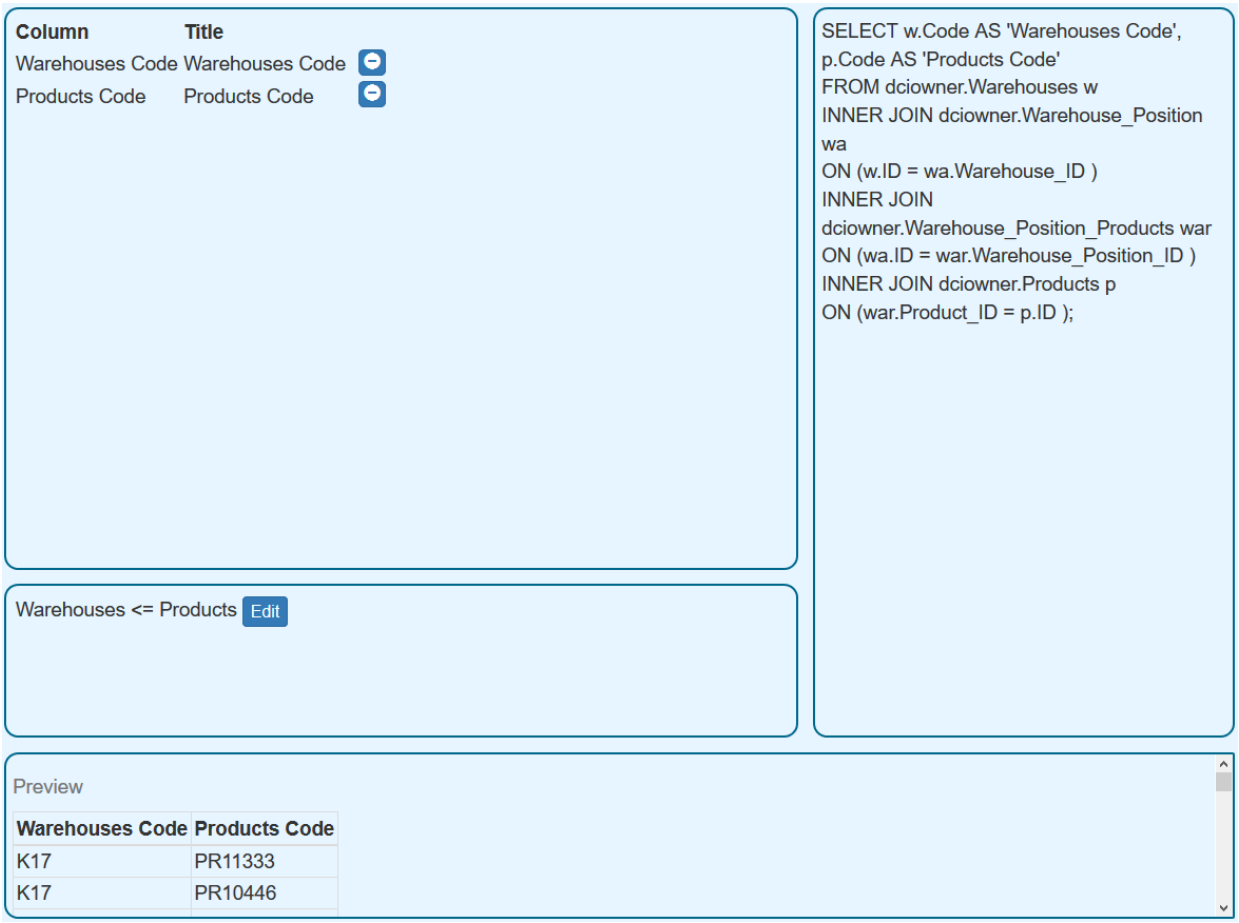
   V sekci s vybranými vlastnostmi přibyla nová vlastnost *Products Code*. V sekci s dotazem je zobrazen dotaz pro výběr dat, který odpovídá dotazu z kódu 9.10 a po provedení vrací stejná data. Dále je zobrazen náhled, který obsahuje dva sloupce, první s kódy skladů a druhý s kódy položek. V sekci se spojeními se zobrazilo nové spojení mezi `Warehouses` a `Products`.

```
SELECT w.Code AS Warehouse_Code, p.Code AS Product_Code
FROM dciowner.Warehouses w
        INNER JOIN dciowner.Warehouse_Position wp
                ON w.ID = wp.Warehouse_ID
        INNER JOIN dciowner.Warehouse_Position_Products wpp
                ON wp.ID = wpp.Warehouse_Position_ID
        INNER JOIN dciowner.Products p
                ON wpp.Product_ID = p.ID
```

Kód 9.10: Dotaz spojení mezi Warehouses a Products



Obrázek 9.1: Nabídka spojení mezi `Warehouses` a `Products`



Obrázek 9.2: Spojení mezi `Warehouses` a `Products`

Cílem scénáře bylo ověřit možnosti nabízeného prioritního spojení. Jak jsme již zmínili, spojeních mezi skladem a položkou je mnoho. Nejjednodušším je spojení přes entitu `Warehouse_Products`. Ale spojením, které je aplikaci DCIx nejvíce používáno a my ho považujeme za prioritní, je spojení z kódu 9.10 přes entity `Warehouse_Position` a `Warehouse_Position_Products` a uvedené klíče.

Stav po druhém kroku je na obrázku 9.1 a výsledné spojení po třetím kroku je na obrázku 9.2. Jak je vidět, aplikace nabízí správné spojení jako první a výsledné dotazy si odpovídají, proto daný scénář aplikace splňuje dobře.

9.2.2 Scénář výběru zdroje spojení

Druhým scénářem bude spojení více entit k jedné výchozí. To znamená, že máme entitu transakce a k ní budeme připojovat entity transakčních definic, položek, pozic a balení. Postupovat budeme podle následujících kroků a očekáváme zmíněné chování.

1. Klikneme na přidání vlastnosti `Create_Date` z entity `Transactions`.

   V sekci s vybranými vlastnostmi se objeví nová vlastnost *Transaction Create_Date*, v sekci s dotazem je zobrazen jednoduchý dotaz pro výběr dat. A je zobrazen náhled, který obsahuje jeden sloupec s daty vytvoření transakcí.

2. Klikneme na přidání vlastnosti `Name` z entity `Transaction_Definitions`.

   Aplikace nám nabídne její návrh, jak je vhodné dané entity spojit. Jako v předchozím scénáři máme možnost volit parametry spojení, toho nyní také nevyužijeme.

3. Potvrdíme nabízené spojení.

   V sekci s vybranými vlastnostmi se objeví nová vlastnost *Transaction_Definitions Name*. V sekci s dotazem je zobrazen dotaz spojení. Dále je zobrazen náhled, který obsahuje dva sloupce, první s daty vytvoření transakcí a druhý s jmény jejich definice. V sekci se spojeními se zobrazilo nové spojení mezi `Transactions` a `Transaction_Definitions`.

4. Klikneme na přidání vlastnosti `Code` z entity `Products`.

   Aplikace nám nabídne její návrh, jak je vhodné dané entity spojit. Také jsou zde modifikovatelné parametry spojení. Navíc je možné vybrat výchozí entitu, kam chceme novou entitu připojit, která bude nastavena na poslední připojenou entitu a tou byla transakční definice.

5. Vybereme jako výchozí entitu `Transactions`.

   Aplikace nám nabídne návrh s cestou mezi transakcemi a položkami. Opět máme možnost zvolit si parametry spojení.

6. Změníme typ spojení na `LEFT OUTER JOIN`. A potvrdíme spojení.

   V sekci s vybranými vlastnostmi přibyla nová vlastnost *Products Code*, v sekci s dotazem je zobrazen aktuální dotaz spojení. Dále je zobrazen náhled, ve kterém přibyl sloupec s kódy položek. V sekci se spojeními se zobrazilo nové spojení mezi `Transactions` a `Products`.

7. Klikneme na přidání vlastnosti `Code` z entity `Positions`. Aplikace nám nabídne její návrh, jak je vhodné dané entity spojit. Opět máme možnost zvolit si parametry spojení.

8. Vybereme jako výchozí entitu `Transactions`.

   To nabídne nové spojení jako v předchozích krocích s možností nastavení. Nabízené spojení by mělo být přímé.

9. Změníme typ spojení na `LEFT OUTER JOIN`. Nastavení potvrdíme.

   V sekci s vybranými vlastnostmi přibyla nová vlastnost *Positions Code*, v sekci s dotazem je zobrazen aktuální dotaz spojení. Dále je zobrazen náhled, ve kterém přibyl sloupec s kódy pozice. V sekci se spojeními se zobrazilo nové spojení mezi `Transactions` a `Positions`.

10. Klikneme na přidání vlastnosti `Label_Number` z entity `Warehouse_-Packages`.

    Aplikace nám nabídne návrh, jak je vhodné dané entity spojit. Opět máme možnost zvolit si parametry spojení.

11. Vybereme jako výchozí entitu `Transactions`.

    To nabídne nové spojení jako v předchozích krocích s možností nastavení. Nabízení spojení by mělo být přímé.

12. Změníme typ spojení na `LEFT OUTER JOIN` a cizí klíč na `Label_ID`. Nastavení potvrdíme.

    V sekci s vybranými vlastnostmi přibyla nová vlastnost *Warehouse_-Packages Label_Number* a celkem je jich pět. V sekci s dotazem je zobrazen dotaz spojení, který odpovídá dotazu z kódu 9.11 a po provedení vrací stejná data. Dále je zobrazen náhled, který zobrazuje tabulku s pěti sloupci naplněnou daty. V sekci se spojeními se zobrazilo

nové spojení mezi `Transactions` a `Positions` a nyní všechna čtyři spojení vedou na entitu transakce.

```
SELECT t.Create_Date AS Create_Date, td.Name AS Definition_Name,
       p.Code AS Product_Code, po.Code AS Position_Code,
       wp.Label_Number AS Label_Number
FROM dciowner.Transactions t
       INNER JOIN dciowner.Transaction_Definitions td
              ON t.Transaction_Definition_ID = td.ID
       LEFT OUTER JOIN dciowner.Products p
              ON t.Product_ID = p.ID
       LEFT OUTER JOIN dciowner.Positions po
              ON t.From_Position_ID = po.ID
       LEFT OUTER JOIN dciowner.Warehouse_Packages wp
              ON t.Label_ID = wp.ID
```

Kód 9.11: Dotaz spojení Transakce - Trn. definice - Položka - Pozice - Balení

Cílem tohoto scénáře bylo ověřit možnosti nastavení výchozích entit spojení, typů spojení a cizích klíčů spojení. Tento scénář byl již delší, proto zde nebudeme ukazovat obrázky všech kroků. Obrázek v příloze A.8 zobrazuje nabídku spojení těsně před potvrzením v kroku 12. Obrázek v příloze A.10 ukazuje ten samý krok po dokončení, to je finální stav scénáře. Jak lze na obrázcích vidět, aplikace nabízí možnost zvolit si zdroj spojení a výsledné dotazy si také odpovídají, proto i tento scénář aplikace splňuje dobře.

9.2.3 Scénář zřetězení a rozštěpení spojení

Posledním scénářem je zřetězení spojení a následné jeho rozštěpení na více spojení. To znamená vytvoření netriviálního spojení mezi sklady a pozicemi. Následné netriviální spojení mezi pozicemi a zónami a nakonec rozštěpení spojení přes skladové pozice, které obě tato spojení používala.

1. Klikneme na přidání vlastnosti `Code` z entity `Warehouses`.

2. Klikneme na přidání vlastnosti `Code` z entity `Positions`.

   Očekáváme nabízené spojení přes entitu `Warehouse_Position`.

3. Změníme typ spojení mezi skladovou pozicí a pozicí na `LEFT OUTER JOIN`. Potvrdíme nabízené spojení.

   Vznikne spojení mezi sklady a pozicemi.

4. Klikneme na přidání vlastnosti `Code` z entity `Zones`.

   Očekáváme nabízené spojení přes entity `Warehouse_Position` a `Warehouse_Position_In_Zones`.

5. Změníme typy spojení na `LEFT OUTER JOIN`. Potvrdíme nabízené spojení.

   Vznikne spojení mezi pozicemi a zónami. V sekci s dotazem je zobrazen dotaz spojení, který odpovídá dotazu z kódu 9.12 a po provedení vrací stejná data.

6. Klikneme na přidání vlastnosti `Type` z entity `Warehouse_Position`.

   Aplikace by měla poznat, že entita skladové pozice je již uvnitř spojení, které jsme definovali v předchozích krocích. Proto chceme, aby vznikla nová spojení, která jsou následující: První spojení je mezi skladem a skladovou pozicí. Druhé spojení je mezi skladovou pozicí a pozicí. Nakonec třetí spojení je mezi skladovou pozicí a zónou. Ostatní nastavení spojeních jsou stejné jako v minulém kroku. V sekci s dotazem je zobrazen dotaz spojení, který odpovídá dotazu z kódu 9.12, pouze je zde navíc přidaný sloupec skladových pozic Type ve výběru.

```
SELECT w.Code AS Warehouse_Code, p.Code AS Position_Code, z.Code AS Zone_Code
FROM dciowner.Warehouses w
INNER JOIN dciowner.Warehouse_Position wp
        ON w.ID = wp.Warehouse_ID
LEFT OUTER JOIN dciowner.Positions p
        ON wp.Position_ID = p.ID
LEFT OUTER JOIN dciowner.Warehouse_Positions_In_Zones wpiz
        ON wp.ID = wpiz.Warehouse_Position_ID
LEFT OUTER JOIN dciowner.Zones z
        ON wpiz.Zone_ID = z.ID
```

Kód 9.12: Dotaz spojení Sklad - Skladová pozice - Pozice - Zóna

Cílem tohoto scénáře bylo ověřit možnost zřetězit spojení, což testují všechny kroky kromě posledního. Poslední krok testuje možnost přidat vlastnost entity, která je již obsažena uvnitř některého spojení. Jak je vidět na obrázku v příloze A.11, který zachycuje stav po přidání kódu zóny po kroku pět, spojení lze řetězit a zobrazený dotaz odpovídá dotazu z kódu 9.12. Nakonec na obrázku v příloze A.12 lze vidět, jak se aplikace zachovala po přidání sloupce Type a rozštěpení spojení. Dotaz zůstává až na přidaný sloupec ve výběru stejný, pouze část se spojeními je přehlednější. Nakonec i tento scénář aplikace splňuje dobře.

10 Závěr

Hlavním cílem práce bylo najít nebo vytvořit nástroj, který by sloužil jako generátor datových pohledů pro sestavy aplikace DCIx. Provedli jsme několik postupných kroků, které vedly k funkčnímu prototypu generátoru dotazu `SELECT` jazyka SQL. Nejprve jsme se seznámili s procesy a daty v aplikaci DCIx a to nám dalo potřebný přehled pro tuto práci. Poté jsme stanovili požadavky v části na hledaný nástroj a podle nich jsme provedli rešerši existujících nástrojů Pentaho, Spago BI, Dremio, JasperReports, Kibana a DBxtra. Jejich hodnocení je uvedeno v části 3.3 a bylo důvodem k tvorbě vlastního prototypu. Ten jsem se rozhodli směrovat tak, aby splňoval nejvíce prioritní požadavky.

Proto se v práci nachází podrobný rozbor statistik a informací, které chceme zobrazovat a které jsme implementovali do prototypu. Jsou rozděleny podle toho, zda se vztahují k entitě nebo k vlastnosti. Pro vlastnosti se dělí dále podle jejich datového typu. To splňuje Požadavek 1.

Dále jsme provedli rozbor dotazu `SELECT` jazyka SQL a vybrali funkčnost, která je součástí prototypu. Poté jsme navrhli a vygenerovali metadata, která využíváme pro definování spojení mezi tabulkami a další užitečné informace. Hledání spojení mezi tabulkami jsme převedli na problém hledání cesty v ohodnoceném grafu. Tím jsme splnili požadavky 2, 4 a 5. Navíc se nám z prototypu stal jednoduchý expertní systém, protože umí uživateli napovídat nejvhodnější spojení.

Prototyp nabízí uživateli náhled dat pro skládaný dotaz, vyhledávání v sekci s nabízenými entitami a vlastnostmi a také ukládání a načítání tvořeného dotazu.To splňuje požadavky 3, 6 a 7.

Protože jsme plánovali splnit horní polovinu požadavků na nástroj a nakonec se povedlo splnit 7 z 10 požadavků a prototyp prošel zadanými scénáři bez problémů. Z těchto důvodů považujeme tento prototyp i celou práci za úspěšné.

Jelikož se jedná o prototyp, je zde mnoho dalšího prostoru pro vylepšení. Jako první je samotné napojení na aplikaci DCIx, které v práci nebylo možné. Druhým vylepšením by mohlo být přidání uživatelského rozhraní pro správu aliasů entit a tím rozšíření možností sestavovaného dotazu. Dále by jednoduše šlo přidat uživatelské rozhraní a podpora generátoru pro definování agregačních funkcí nad vlastnostmi. Kromě toho můžeme splnit také zbylé požadavky 8, 9 a 10, ale jejich implementace by již nebyla triviální. A mnoho dalších.

Přehled zkratek

API	Application Programming Interface
APS	Advanced Planning and Scheduling
BA	Business Analytics
BI	Business Intelligence
DCI	Delivery Chain Integrator
DTO	Data Transfer Object
EDI	Electronic Data Interchange
FIFO	First In First Out
HQL	Hibernate Query Language
HTML	HyperText Markup Language
JDBC	Java Database Connectivity
JIS	Just In Sequence
JIT	Just In Time
JPA	Java Persistence API
JRXML	JasperReports XML
JSON	JavaScript Object Notation
LDAP	Lightweight Directory Access Protocol
MES	Manufacturing Execution System
MOM	Manufacturing Operations Management
ODBC	Open Database Connectivity
ORM	Object-Relational Mapping
QB	Query Builder
QD	Query Designer
QMS	Quality Management System
RFID	Radio Frequency Identification
PDI	Pentaho Data Integration
PME	Pentaho Metadata Editor
PPS	Production Planning System
PRD	Pentaho Report Designer
PUC	Pentaho User Console
REST	Representational State Transfer
SQL	Structured Query Language
UI	User Interface
VMI	Vendor Managed Inventory
WMS	Warehouse Management System
XML	Extensible Markup Language
YMS	Yard Management System

Literatura

[1] *DCIx* [online]. Aimtec a.s., 2018. [cit. 13.01.2018]. Dostupné z: https://www.aimtecglobal.com/dcix/.

[2] *DBxtra Online Documentation* [online]. DBxtra, 2017. [cit. 2017/12/11]. Dostupné z: http://www.dbxtra.com/documentation/.

[3] DBxtr trial 10.0.1, 2018. Dostupné z: http://www.dbxtra.com/download.htm.

[4] *Dremio Documentation* [online]. Dremio Community, 2017. [cit. 2017/12/11]. Dostupné z: https://docs.dremio.com/.

[5] Dremio Community Edition 1.3.1, 2017. Dostupné z: https://www.dremio.com/download/.

[6] *frontend-maven-plugin* [online]. Eirik Sletteberg, 2018. [cit. 2018/04/23]. Dostupné z: https://github.com/eirslett/frontend-maven-plugin.

[7] *Hibernate ORM Documentation - 5.2* [online]. Red Hat, 2018. [cit. 2018/04/23]. Dostupné z: http://hibernate.org/orm/documentation/5.2/.

[8] Jaspersoft Studio 6.4.3, 2017. Dostupné z: https://community.jaspersoft.com/project/jaspersoft-studio/releases.

[9] *Kibana User Guide* [online]. Elastic, 2017. [cit. 2017/12/21]. Dostupné z: https://www.elastic.co/guide/en/kibana/6.1/index.html.

[10] Kibana 6.1.1, 2017. Dostupné z: https://artifacts.elastic.co/downloads/kibana/kibana-6.1.1-windows-x86_64.zip.

[11] *Release Notes – Maven 3.3.9* [online]. The Apache Software Foundation, 2018. [cit. 2018/04/24]. Dostupné z: https://maven.apache.org/docs/3.3.9/release-notes.html.

[12] *Data types (Transact-SQL)* [online]. Microsoft, 2017. [cit. 2017/12/29]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql.

[13] Pentaho Data Integration CE 8.0, 2017. Dostupné z: https://sourceforge.net/projects/pentaho/files/Pentaho%208.0/client-tools/.

[14] Pentaho Metadata Editor CE 8.0, 2017. Dostupné z: https://sourceforge.net/projects/pentaho/files/Pentaho%208.0/client-tools/.

[15] Pentaho Report Designer CE 8.0, 2017. Dostupné z: https://sourceforge.net/projects/pentaho/files/Pentaho%208.0/client-tools/.

[16] Pentaho Server CE 8.0, 2017. Dostupné z: https://sourceforge.net/projects/pentaho/files/Pentaho%208.0/server/.

[17] *Pentaho Community Wiki* [online]. Pentaho Community, 2017. [cit. 2017/12/11]. Dostupné z: https://wiki.pentaho.com/.

[18] *PostgreSQL 9.6.8 Documentation* [online]. The PostgreSQL Global Development Group, 2018. [cit. 2018/04/24]. Dostupné z: https://www.postgresql.org/docs/9.6/static/index.html.

[19] *Thinking in React* [online]. Facebook, 2018. [cit. 2018/04/23]. Dostupné z: https://reactjs.org/docs/thinking-in-react.html.

[20] *Recharts Getting Started* [online]. Recharts Group, 2018. [cit. 2018/04/23]. Dostupné z: http://recharts.org/en-US/guide/getting-started.

[21] *Redux usage with React* [online]. Redux community, 2018. [cit. 2018/04/23]. Dostupné z: https://redux.js.org/basics/usage-with-react.

[22] *What Is REST?* [online]. RestApiTutorial.com, 2018. [cit. 2018/04/23]. Dostupné z: http://www.restapitutorial.com/lessons/whatisrest.html.

[23] All-In-One-SpagoBI-5.2.0, 2016. Dostupné z: http://forge.ow2.org/project/showfiles.php?group_id=204.

[24] *Spring Guides* [online]. Pivotal, 2018. [cit. 2018/04/23]. Dostupné z: https://spring.io/guides.

[25] *SQL Server Documentation* [online]. Microsoft, 2017. [cit. 2018/01/10]. Dostupné z: https://docs.microsoft.com/en-us/sql/t-sql/language-reference.

[26] *Webpack Concepts* [online]. Webpack Community, 2018. [cit. 2018/04/23]. Dostupné z: https://webpack.js.org/concepts/.

[27] Aimtec. *Řízení denních odvolávek a sekvenčních dodávek*, 2017.

[28] Aimtec. *Detailní sledování a evidence rozpracované výroby*, 2017.

[29] Aimtec. *Portálová kooperace odběratelů s dodavateli*, 2017.

[30] Aimtec. *Systém pro podporu operativního plánování a organizování výroby*, 2017.

[31] Aimtec. *Systém pro řízení kvality procesů*, 2017.

[32] Aimtec. *Řízení skladů a materiálové logistiky*, 2017.

[33] Aimtec. *Systém pro řízení nakládky a vykládky*, 2017.

[34] Cormen, T. H. et al. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

[35] Engineering Group. *SpagoBI User Manual*. SpagoBI Labs, release 5.x edition, 2014.

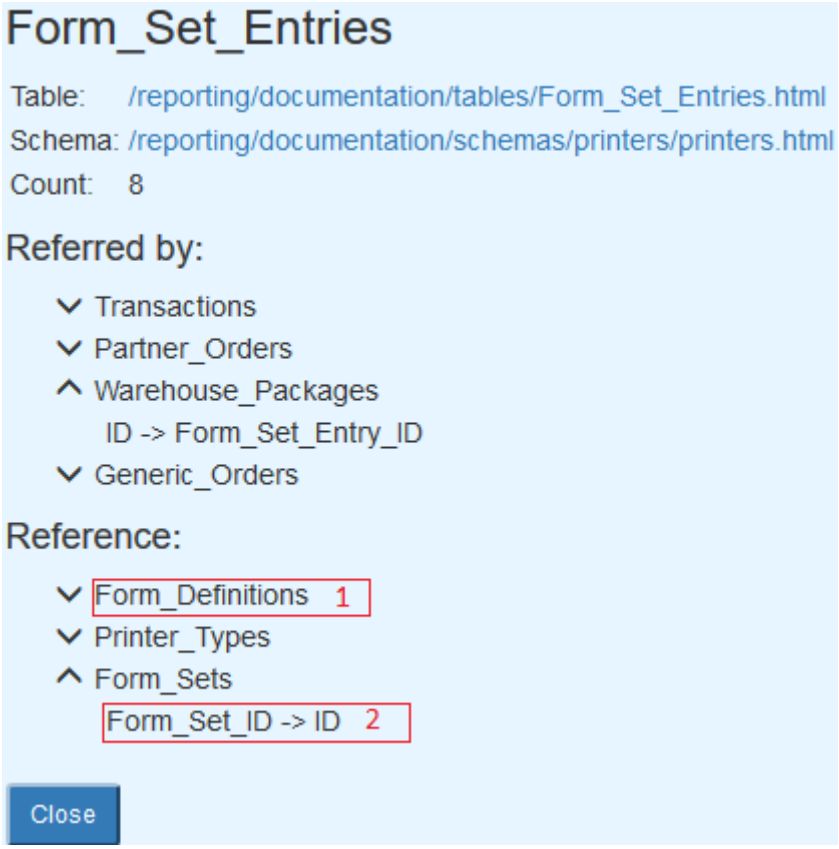
[36] Hutmacher, D. *How to build a histogram in T-SQL* [online]. 2014. [cit. 2018/01/13]. Dostupné z: `https://sqlsunday.com/2014/09/14/histogram-in-t-sql/`.

[37] Jaspersoft Corporation. *The JasperReports Ultimate Guide*. Jaspersoft Corporation, 3rd edition, 2011.

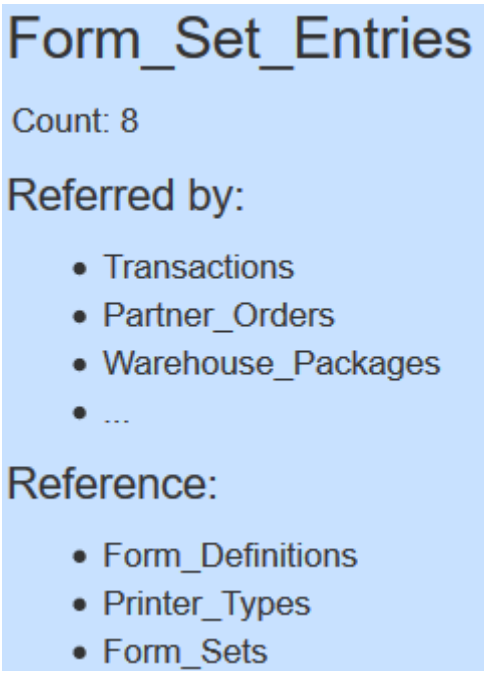
[38] Schmeling, H. *SQL Server Statistics*. Simple-Talk Publishing, 2010.

[39] Weinberg, P. – Groff, J. – Oppel, A. *SQL The Complete Reference,Third Edition*. McGraw-Hill Education, 3rd edition, 2010.

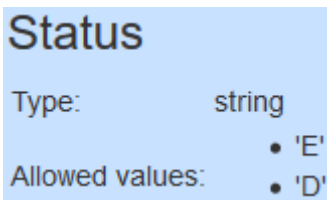
Příloha A Obrázky



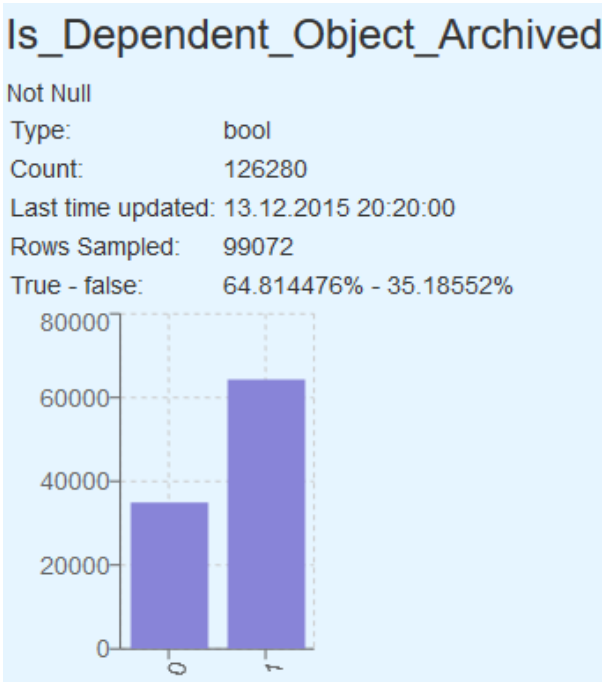
Obrázek A.1: Detail statistiky a informace entity



Obrázek A.2: Náhled statistiky a informace entity

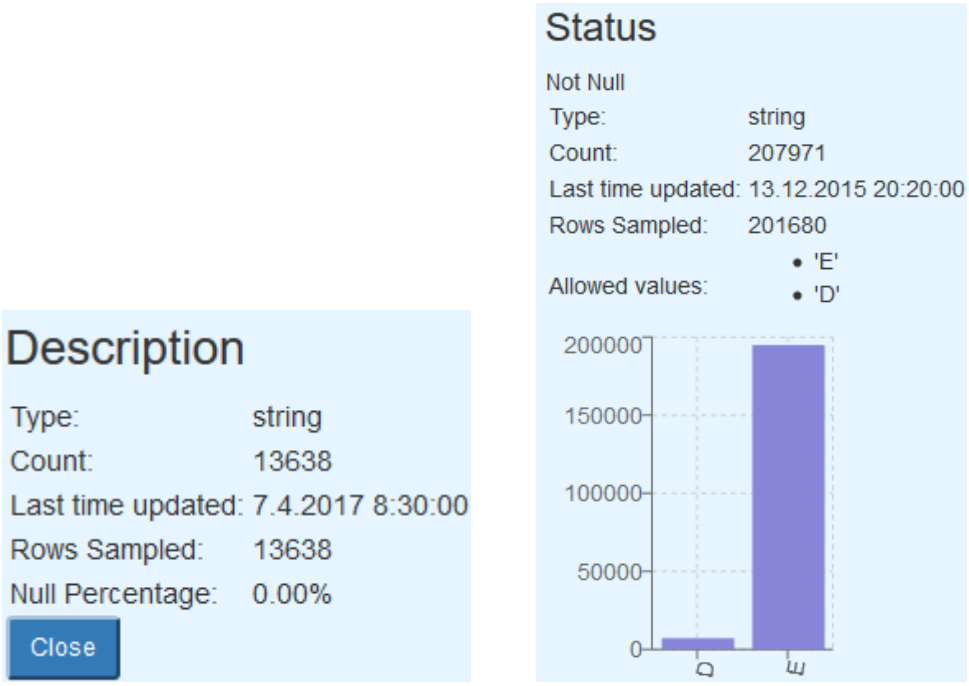


(a) Náhled vlastnosti



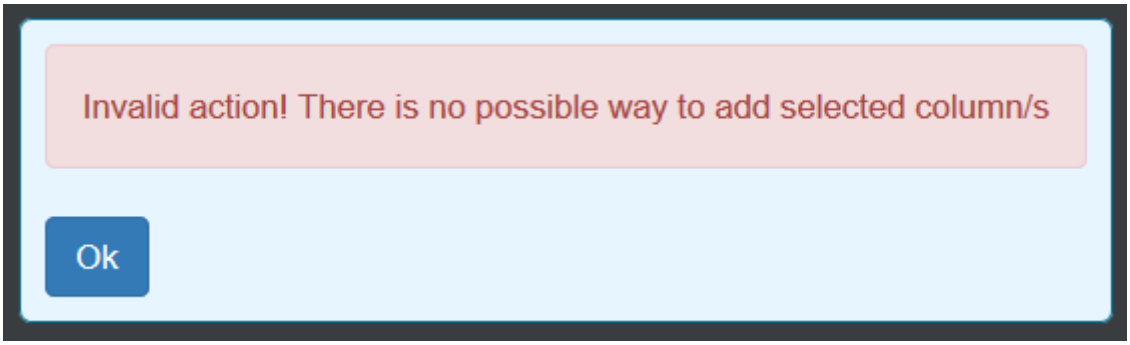
(b) Detail logické hodnoty

Obrázek A.3: Statistiky a informace vlastností

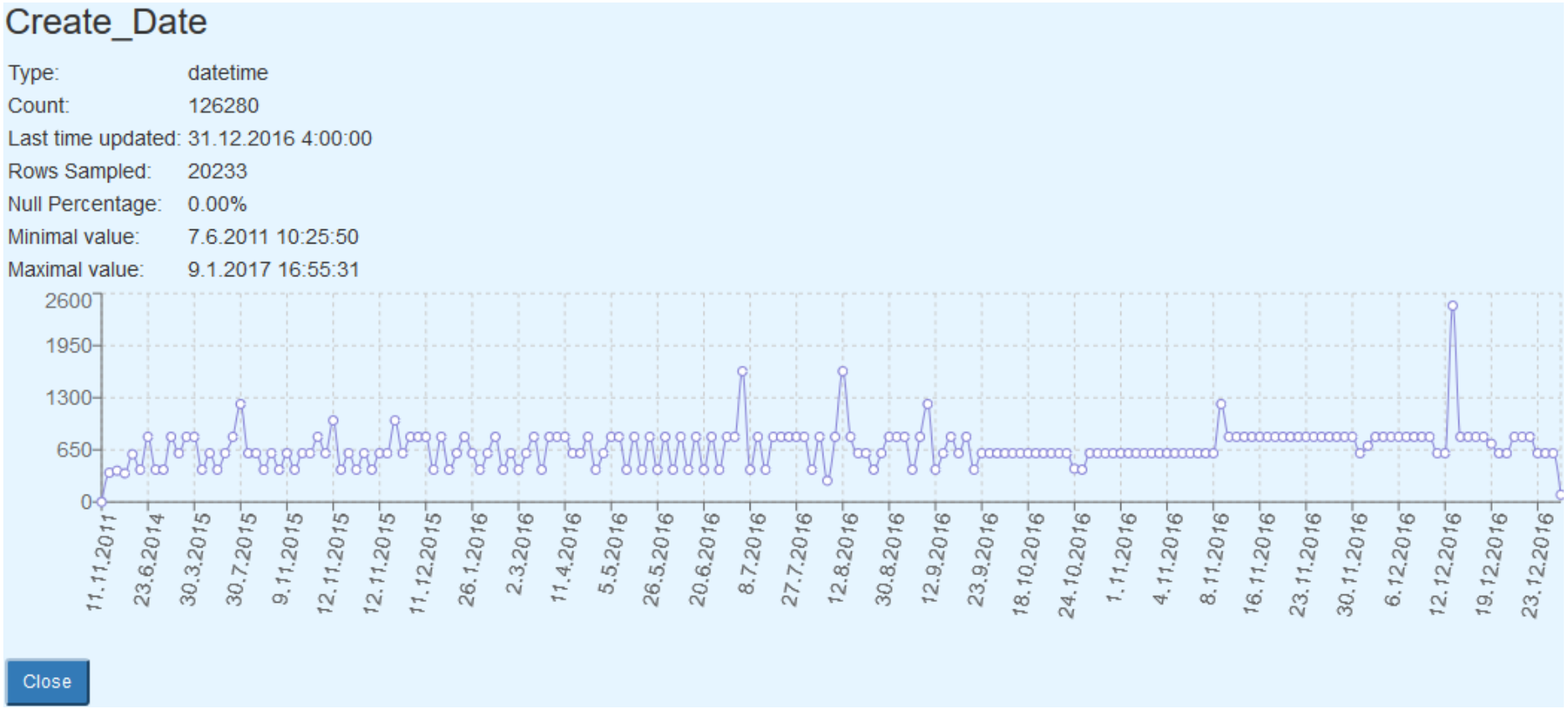


(a) Detail řetezce bez histogramu (b) Detail řetězce s histogramem

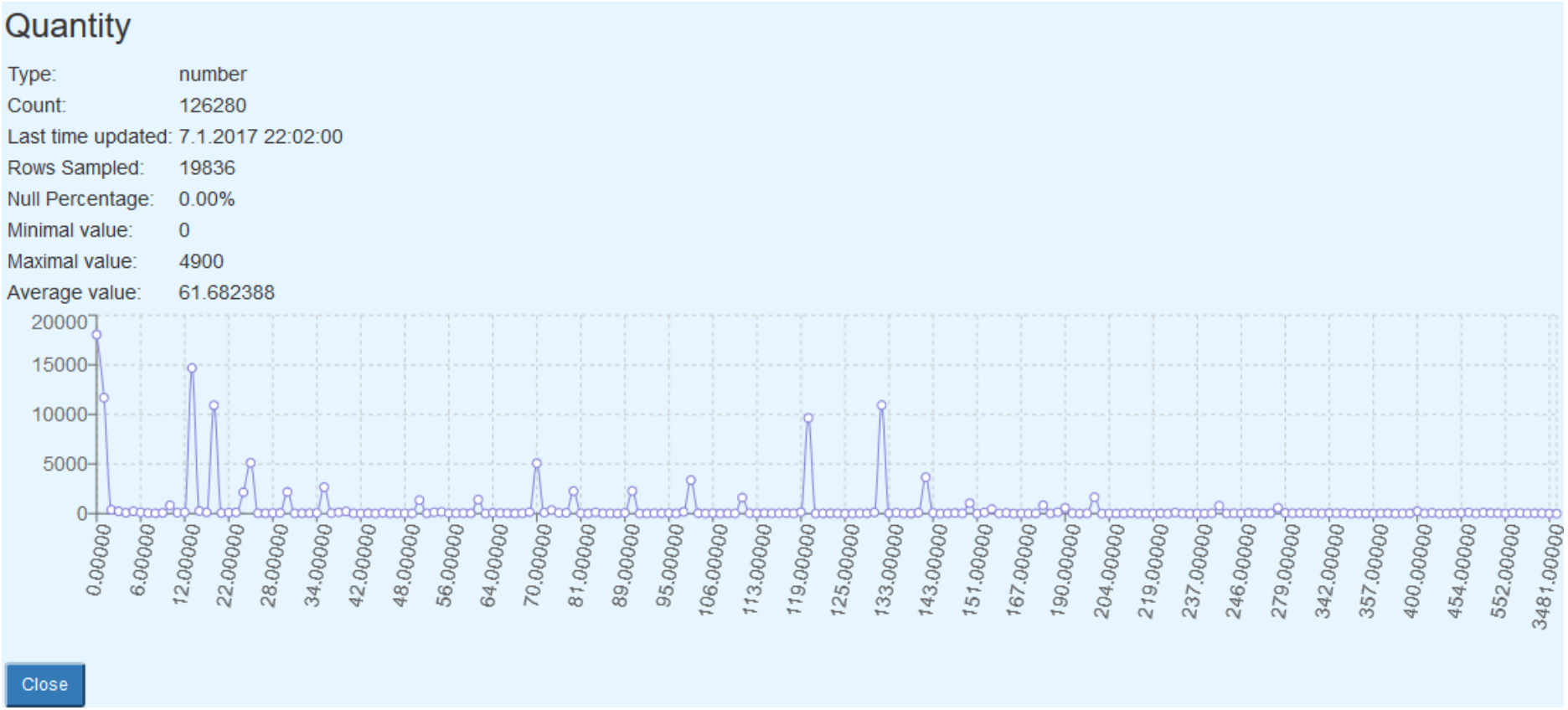
Obrázek A.4: Statistiky a informace řetězců



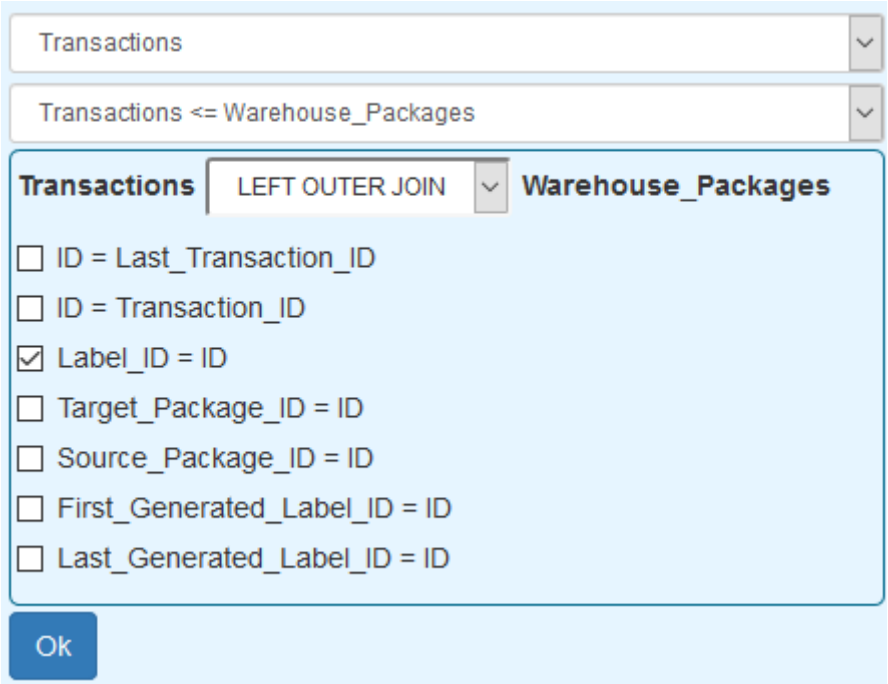
Obrázek A.5: Chyba, která vznikne pokud neexistuje cesta, jak přidat entitu



Obrázek A.6: Detail časové vlastnosti



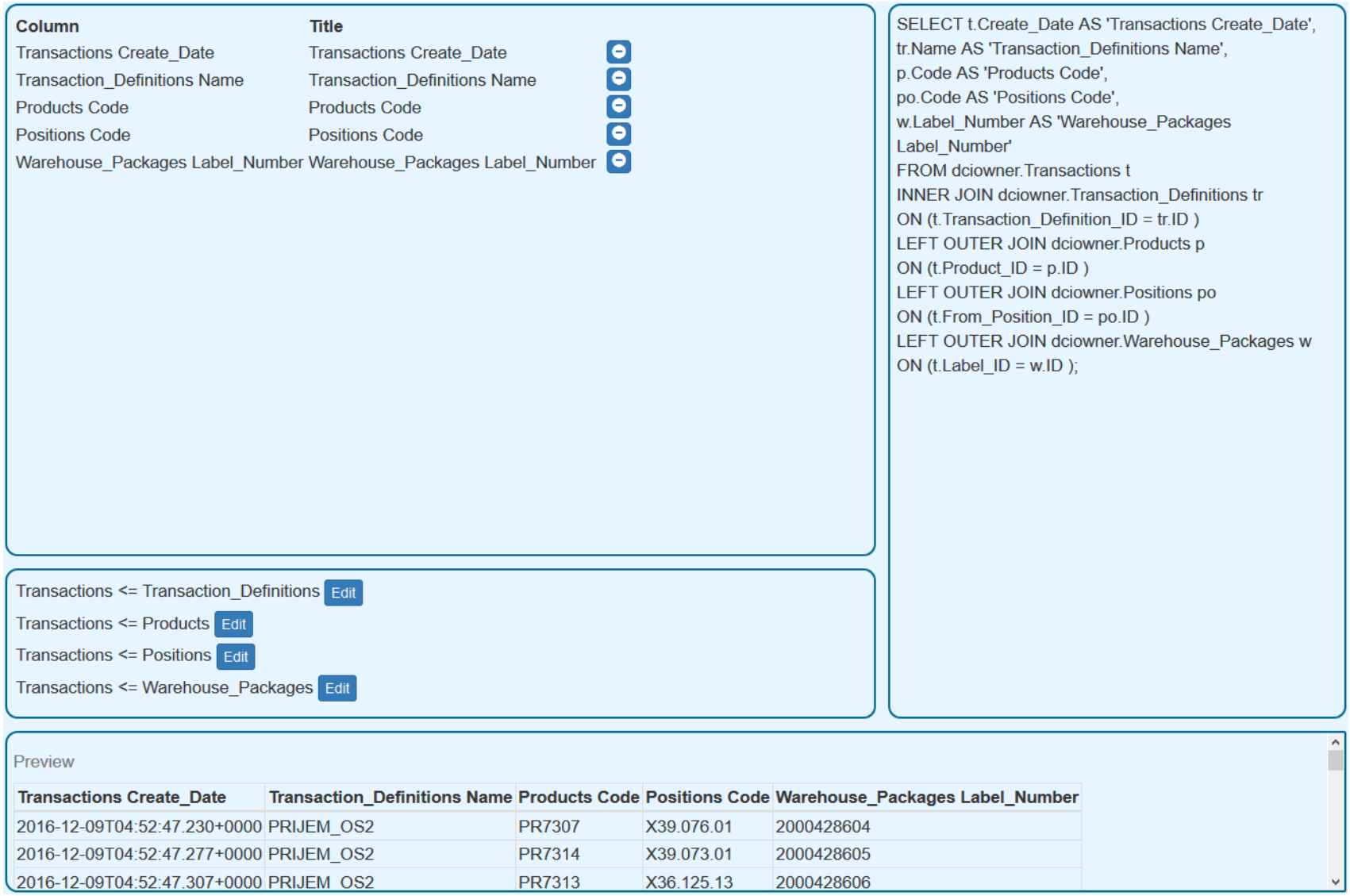
Obrázek A.7: Detail číselné vlastnosti



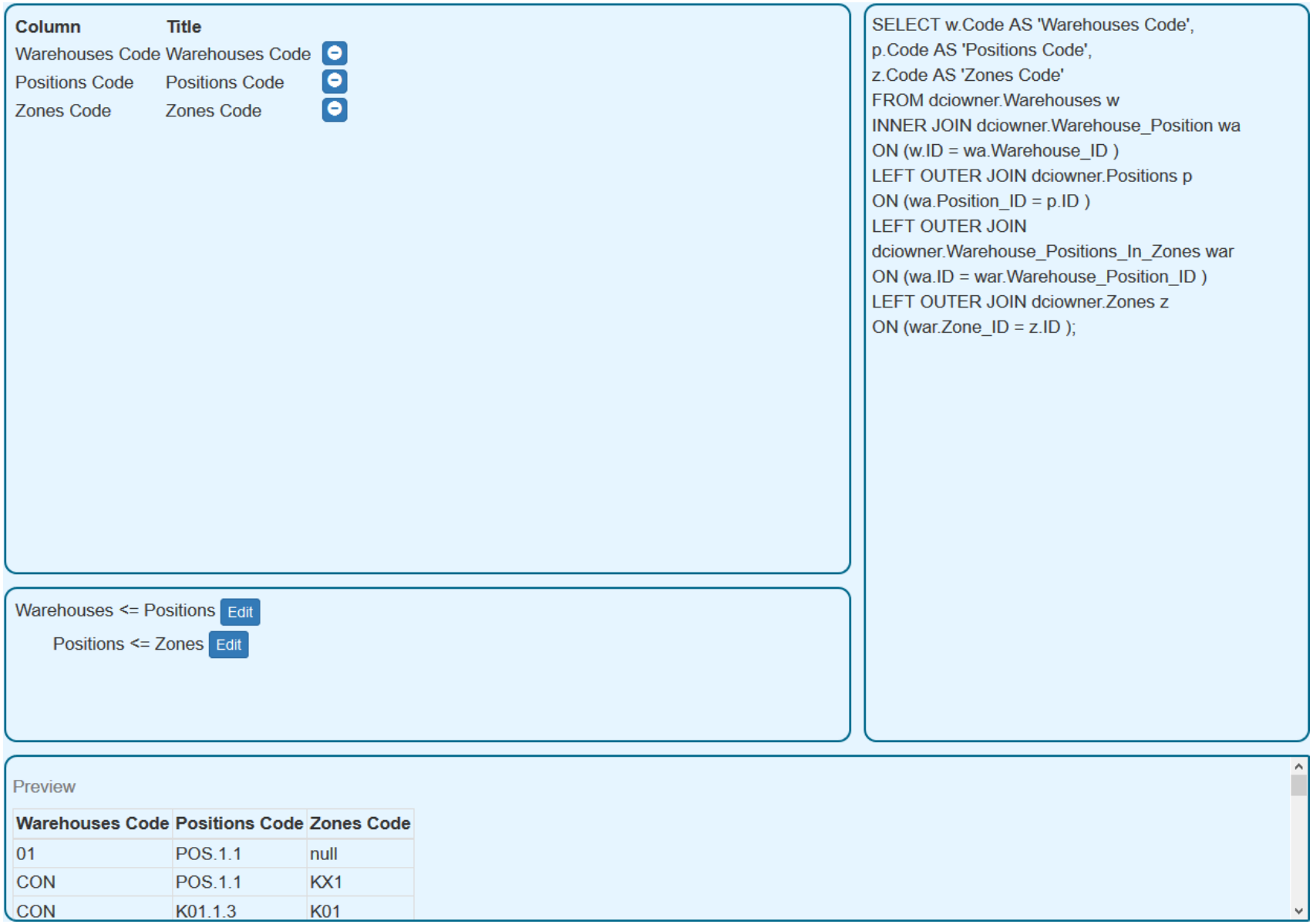
Obrázek A.8: Nabídka spojení mezi `Transactions` a `Warehouse_Packages`

(a) Uložení dotazu (b) Načtení dotazu

Obrázek A.9: Modální okna pro uložení a načtení dotazu

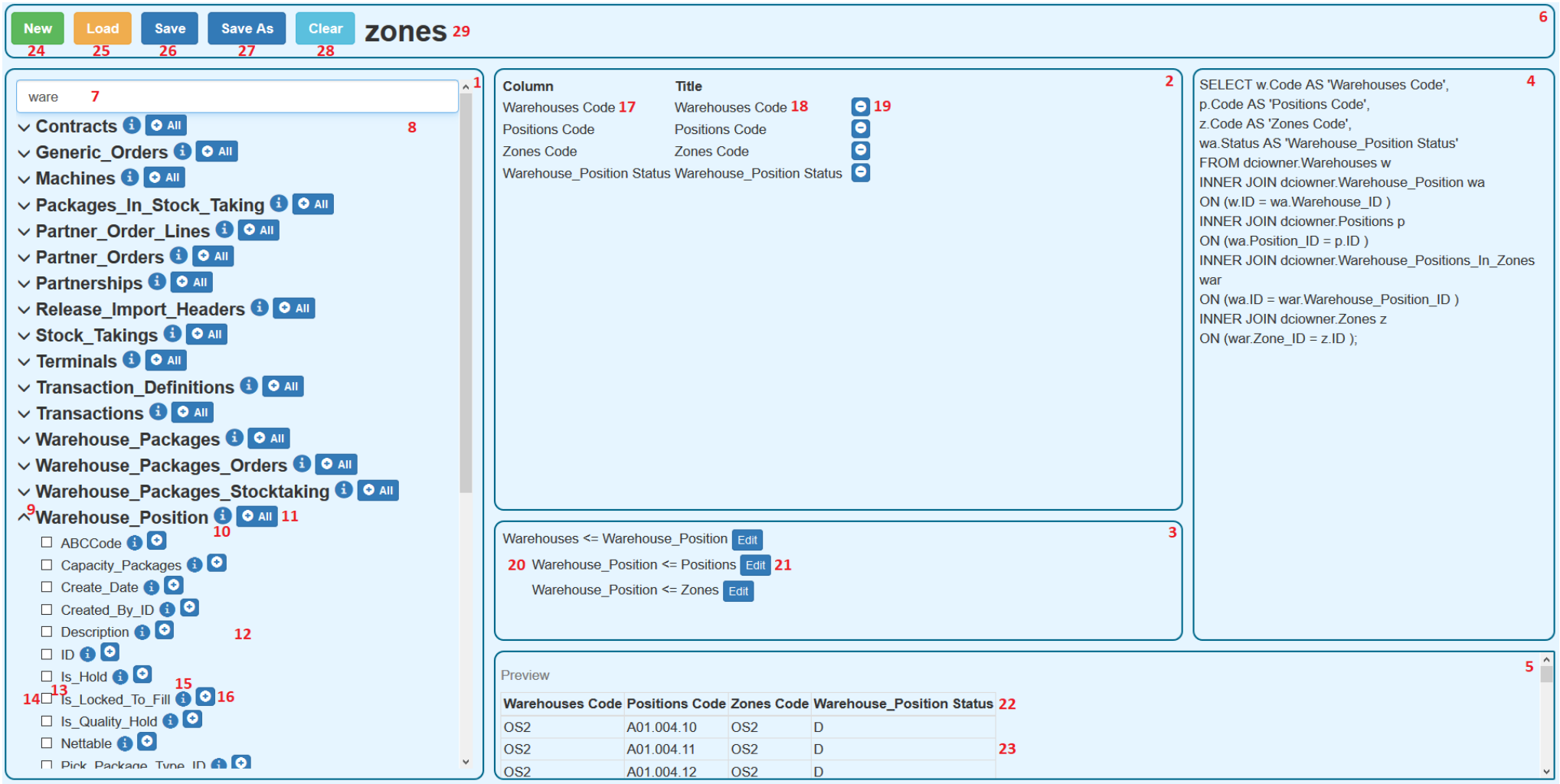


Obrázek A.10: Spojení mezi `Warehouses` a `Products`



Obrázek A.11: Nabídka spojení mezi `Transactions` a `Warehouse_Packages`

Obrázek A.12: Spojení mezi `Warehouses` a `Products`



Obrázek A.13: Obrazovka aplikace

Příloha B Uživatelská příručka

Aplikace má šest oddělených částí. Tyto části se zde popíšeme na grafickém rozhraní ukážeme.

B.1 Entity a vlastnosti

Jedná se o část (1) z obrázku A.13.

(7) Pole slouží k vyhledávání entit a vlastností. Zobrazí v seznamu entit (8) všechny entity, které ve svém názvu obsahují vyhledávaný řetězec a také všechny entity, jejichž vlastnost ho obsahuje.

(8) Seznam filtrovaných entit.

(9) Entita, kliknutím lze rozbalit a zobrazit její vlastnosti (12), opětovným kliknutím zase zabalit a schovat. Umožňuje náhled popsáný v části B.1.1.

(10) Tlačítko *i*, které slouží k zobrazení detailního náhledu pro entitu v modálním okně, který je popsán v části B.1.2

(11) Tlačítko *+ All*, které slouží k přidání vybraných vlastností do dotazu. K vybrání vlastností slouží jejich checkbox (14). Může vyvolat chybu jako na obrázku A.5.

(12) Seznam vlastností pro danou entitu.

(13) Vlastnost entity, umožňuje náhled popsáný v části B.1.3.

(14) Checkbox pro vybrání vlastnosti entity.

(15) Tlačítko *i*, které zobrazuje detailní pohled na vlastnost v modálním okně, který je popsán v částí B.1.4.

(16) Tlačítko *+*, které slouží k přidání dané vlastností do dotazu. Může vyvolat chybu jako na obrázku A.5.

B.1.1 Náhled entity

Pokud na entitu najedeme kurzorem, po vteřině se zobrazí náhled statistik a informací, který vypadá jako na obrázku A.2. Ten obsahuje jméno entity, předpokládaný počet záznamů *Count*, seznam entit, na které tato entita odkazuje *Reference*, a seznam entit, které se odkazují na ni *Referred by*.

B.1.2 Detail entity

Detail entity je zachycen na obrázku A.1. Obsahuje jméno entity, předpokládaný počet záznamů *Count*, odkaz na dokumentaci entity *Table* a odkaz

na schéma v dokumentaci *Schema*, ve kterém se entita nachází. Dále se zde jako v náhledu B.1.1 nachází seznam odkazovaných a odkazujících entit. Na tyto entity (A.1.1) lze kliknout a provede se zobrazení jednotlivých cizích klíčů (A.1.2) pro konkrétní spojení, dalším kliknutím na entitu cizí klíče opět zmizí.

B.1.3 Náhled vlastnosti

Po najetí kurzorem vlastnost se se po vteřině zobrazí náhled statistik a informací, který je zachycen na obrázku A.3a. Obsahuje jméno vlastnosti, její základní typ, což je jedna z části 4.2. Dále je zde informace, zda nesmí vlastnost obsahovat nulové hodnoty *NotNull* a pokud je omezena, tak obsahuje výčet povolených hodnot *Allowed values*.

B.1.4 Detail vlastnosti

Detail statistik a informací pro danou vlastnost. Tento detail obsahuje jméno vlastnosti, její základní typ, což je jedna vlastnost z části 4.2. Dále obsahuje předpokládaný počet záznamů *Count*, datum posledního počítání statistiky *Last time updated* a z kolika vzorků byla statistika počítána *Rows Sampled*. Poté je zde informace, zda může obsahovat nulové hodnoty *NotNull* nebo procento nulových hodnot *Null percentage*. Nakonec, pokud je to možné, je zobrazen histogram. Také je zde zobrazeno, pokud je daná vlastnost omezena na určité povolené hodnoty *Allowed values* a tyto hodnoty.

Další vlastnosti závisejí na datovém typu. Pro logické hodnoty odpovídá obrázek A.3b a obsahuje poměr hodnot pravdy a nepravdy *True-False*. Pro řetězce jsou to obrázky A.4a a A.4b. Časovým údajům odpovídá obrázek A.6, který navíc obsahuje ještě minimální hodnotu vlastnosti *Minimal value* a maximální hodnotu vlastnosti *Maximal value*. Nakonec pro číselné typy to je obrázek A.7, který obsahuje kromě minimální a maximální hodnoty ještě průměrnou hodnotu vlastnosti *Average value*.

B.2 Výběr do dotazu

Jedná se o část (2) z obrázku A.13. Obsahuje seznam vybraných vlastností do výsledného dotazu.

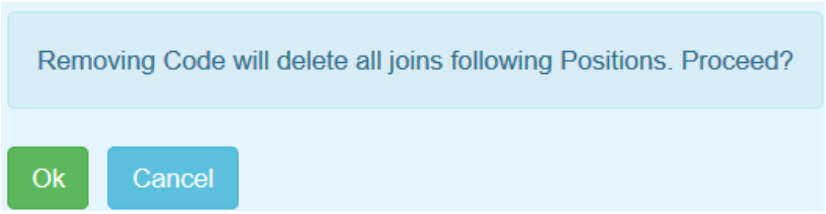
(18) Orientační jméno vybrané vlastnosti.

(19) Jméno vlastnosti v dotazu. Lze upravit, nejprve je potřeba kliknout na současné jméno, poté se objeví pole, ve kterém lze naspat jméno nové

a následně kliknutím mimo pole se provede změna jména. Pole v editačním módu je na obrázku B.1.

(18) Tlačítko -, které slouží k odebrání vlastnosti. Pokud je odebíraná vlastnost poslední z dané entity, je zobrazeno varování, které provede odpojení celé větve dané entity jako na obrázku B.2.

Obrázek B.1: Úprava jména vlastnosti



Obrázek B.2: Varování před odpojením entity

B.3 Seznam spojení

Jedná se o část (3) z obrázku A.13. Obsahuje seznam zvolených spojení do výsledného dotazu.

(20) Je definované spojení, jak bylo vybráno, obsahuje pouze počáteční a koncovou entitu. Celá cesta není zobrazena.

(21) Tlačítko *Edit*, které slouží k otevření modálního okna spojení, z následující části B.3.1, pro úpravu daného spojení.

B.3.1 Úprava nebo přidání spojení

Jedná se o modální okno, které se zobrazí, pokud do dotazu přidáváme vlastnost, jejíž entita tam ještě nebyla nebo pokud chceme upravit stávající spojení. Je zobrazeno na obrázku B.3.

(B.3.1) Selectbox, který slouží k výběru entity, kam se připojit. Pokud je v dotazu pouze jedna entita, selectbox se nezobrazuje, protože je automaticky dána.

(B.3.2) Selectbox, který slouží k výběru cesty mezi přidávanou (upravovanou) entitou a tou, která je vybrána jako přípojná.

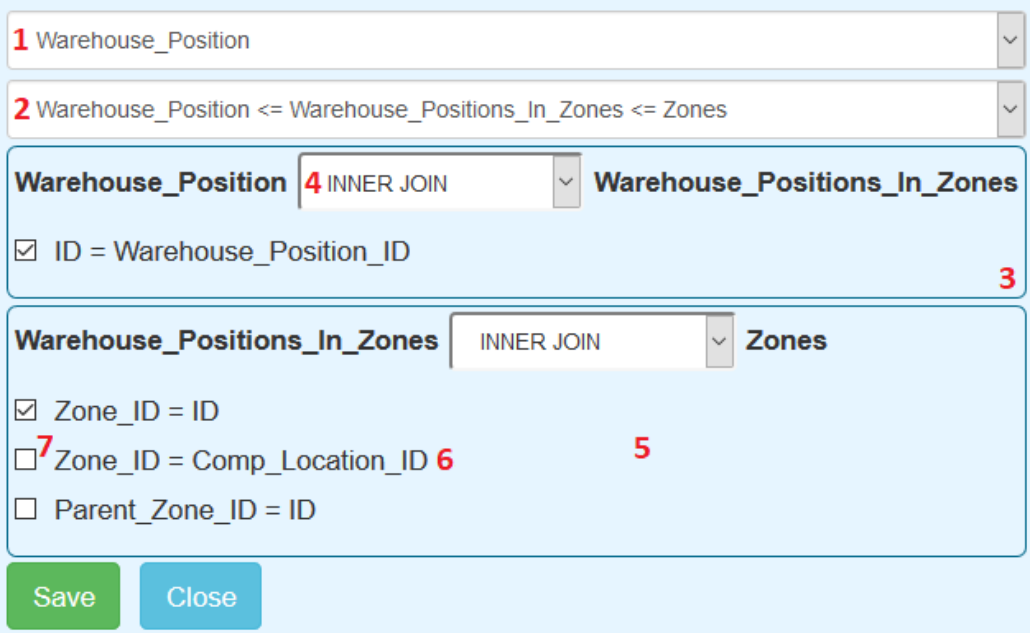
(B.3.3) Oblast spojení. Představuje část vybrané cesty mezi dvěma entitami, které v cestě sousedí.

(B.3.4) Selectbox, který slouží k výběru typu spojení pro danou část cesty. Část je specifikována názvy entit, které jsou uvedené kolem tohoto výběru.

(B.3.5) Oblast cizích klíčů. Lze vybrat i více klíčů zároveň, ale vždy musí být vybrán alespoň jeden klíč pro danou část spojení.

(B.3.6) Cizí klíč dané části spojení.

(B.3.7) Checkbox pro výběr cizího klíče.



Obrázek B.3: Úprava stávajícího spojení

B.4 SQL dotaz

Jedná se o část (4) z obrázku A.13, která obsahuje pouze vygenerovaný SQL dotaz pro daný výběr a spojení.

B.5 Náhled dotazu

Jedná se o část (5) z obrázku A.13, která obsahuje pouze náhled daného dotazu jako tabulku.

(22) Hlavička tabulky odpovídá nastavení ve výběru vlastností.

(23) Řádky tabulky obsahují již reálná data. Náhled je omezen na 25 řádek.

B.6 Nabídka aplikace

Jedná se o část (6) z obrázku A.13, která obsahuje ovládání aplikace.

(24) Tlačítko *New*, které slouží k vytvoření nového pohledu.

(25) Tlačítko *Load*, které otevře výběr uložených dotazů ze sekce B.6.1.

(26) Tlačítko *Save*, které uloží aktuální dotaz, pokud již byl uložen, nebo otevře uložení dotazu ze sekce B.6.1.

(27) Tlačítko *Save As*, které otevře uložení dotazu ze sekce B.6.1.

(28) Tlačítko *Clear*, které odstraní vše z výběru do dotazu a tím se vyčistí celý dotaz.

(29) Jméno aktuálního uloženého. Pokud zde jméno není jedná se o nový, ještě neuložený dotaz.

B.6.1 Uložení a načtení dotazu

Pro uložení dotazu je zde modální okno na obrázku A.9a. To obsahuje pouze pole pro vložení jména. Jméno nesmí být prázdné.

Pro načtení dotazu je zde modální okno, které je na obrázku A.9b. Obsahuje tabulku, která obsahuje jednotlivé uložené dotazy. K nahraní dotazu je potřeba kliknout na danou řádku tabulky.

(A.9b.1) Jméno uloženého dotazu.

(A.9b.2) Datum poslední úpravy dotazu.

Příloha C Instalační příručka

Soubory potřebné pro instalaci se nachází na přiloženém DVD ve složce *Install*.

1. MS SQL Sever 2014 na DVD není, protože se nejedná o volně dostupný software. Složka obsahuje pouze zálohu databáze, nad kterou jsme tuto práci zkoušeli. Je potřeba tuto zálohu obnovit se jménem databáze *dci-data*. Pro vytvoření uživatele slouží skript *setup-dci-user.sql* ze složky *Install*.

2. Dále je potřeba mít PostgreSQL databázi. Instalátor je přiložen. A spustit příkaz *psql -U postgres -a -f setup-postgres-reporting.sql* ve složce *Install*. Nebo upravit cestu k *setup-postgres-reporting.sql* skriptu.

3. Posledním softwarem, který je potřeba mít nainstalovaný je Apache Tomcat 7.0 webový server. Instalátor je přiložen.

4. Nyní stačí zkopírovat soubor s aplikací *reporting.war* do webového serveru, obvykle *C:\Program Files\Apache Software Foundation\Tomcat 7.0\webapps*.

5. Pokud jsme někde nepoužili výchozí porty nebo uvedená jména, je potřeba upravit konfiguraci aplikace *application.properties*.

6. Nakonec je potřeba spustit službu Tomcat a aplikace by měla být dostupná. Výchozí adresa je obvykle *http://localhost:8080/reporting*.