FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Překladač jazyka kontrol do SQL
Student:	Otto Šleger
Vedoucí:	Ing. Štěpán Plachý
Studijní program:	Informatika
Obor / specializace:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Jazyk kontrol je validační jazyk definovaný Českou národní bankou v rámci systému SDAT. Jeho syntaxe je podobná excelovým výrazům. Česká národní banka pomocí tohoto jazyka definuje vlastnosti, které musí dodržet data v odevzdávaných výkazech v rámci reportingu. Validace se dělí na jednovýkazové, mezivýkazové a formátové. Požadovaná struktura reportu společně s validačními skripty je zapsaná ve formátu XML.

1. Pro jednovýkazové validace vytvořte překladač, který vytvoří validační SQL skript, který nad databází s daty pro report vrátí seznam buněk, které neodpovídají požadavkům validace.

2. Vaše řešení otestujte.

Dokumentace jazyka kontrol je dostupná zde:
https://www.cnb.cz/cs/statistika/sdat/dokumentace-technicka-specifikace/

Elektronicky schválil/a doc. Ing. Jan Janoušek, Ph.D. dne 1. března 2023 v Praze.

Bakalářská práce

PŘEKLADAČ JAZYKA KONTROL DO SQL

Otto Šleger

Fakulta informačních technologií
Katedra teoretické informatiky
Vedoucí: Ing. Štěpán Plachý
11. ledna 2024

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek ix

Úvod 1

1 Jazyk kontrol a systém SDAT 3
- 1.1 Česká národní banka a systém SDAT . . . 3
- 1.2 Metodika v systému SDAT . . . 3
  - 1.2.1 Metodická knihovna . . . 4
  - 1.2.2 Výkazy a jejich datové oblasti . . . 5
  - 1.2.3 Kontroly . . . 5
- 1.3 Jazyk kontrol . . . 6

2 Metoda analýzy kódu pomocí LL(1) překladačů 7
- 2.1 Překladač jazyků . . . 7
  - 2.1.1 Překladač programovacích jazyků . . . 7
- 2.2 LL(1) analýza kódu . . . 8
  - 2.2.1 Lexikální analyzátor . . . 8
  - 2.2.2 Syntaktický analyzátor . . . 9
- 2.3 Finální převod do výsledného kódu . . . 9

3 Databáze a jazyk SQL 11
- 3.1 Databáze . . . 11
- 3.2 Relační databáze . . . 11
- 3.3 Jazyk SQL . . . 12

4 Analýza jazyka kontrol 13
- 4.1 Syntaxe a gramatika jazyka kontrol . . . 13
  - 4.1.1 Úprava gramatiky z uživatelského na sémantický tvar . . . 13
  - 4.1.2 Oddělení tokenů od gramatických pravidel . . . 13
  - 4.1.3 Úprava gramatiky na LL(1) podobu . . . 14
- 4.2 Problém překladu z jazyku kontrol do SQL dotazu . . . 14
- 4.3 Analýza základních prvků jazyka kontrol . . . 14
  - 4.3.1 Aritmetické operátory . . . 14
  - 4.3.2 Porovnávací operátory . . . 14
  - 4.3.3 Množina a výběr číselníku . . . 15
  - 4.3.4 Výraz FILTER_CHECK . . . 15
  - 4.3.5 Statický výraz FOR . . . 15
  - 4.3.6 Dynamický výraz FOR . . . 15

4.3.7 Výraz WITH . . . 16
4.3.8 Proměnné . . . 16
4.3.9 Ukazatel . . . 16
4.3.10 Dynamický parametr . . . 16
4.4 Analýza funkcí jazyka kontrol . . . 17
4.4.1 Funkce if . . . 17
4.4.2 Funkce filter . . . 17
4.4.3 Funkce filter_apply . . . 17
4.4.4 Funkce dostupné v jazyce SQL . . . 17
4.4.5 Do definovatelné funkce v jazyce SQL . . . 17
4.4.6 Agregační funkce . . . 18
4.4.7 Booleaonvské operátory . . . 18
4.4.8 Funkce spojující tabulky . . . 18
4.4.9 Funkce doplňující proměnné hodnoty do validace . . . 18
4.4.10 Funkce vynechané z této práce . . . 18
4.5 Nutné prvky v SQL AST . . . 18

5 Implementace překladače jazyka kontrol 21
5.1 Struktura překladače a vlastnosti implementace . . . 21
5.1.1 Překladač z jazyka kontrol do SQL AST . . . 21
5.1.2 Generátor finálního SQL . . . 22
5.1.3 Výhody zvolené struktury překladače . . . 22
5.1.4 Shrnutí . . . 22
5.2 Vstupní proudy . . . 22
5.2.1 Souborový vstupní proud . . . 22
5.2.2 Řetězcový vstupní proud . . . 24
5.3 Lexikální analyzátor pro jazyk kontrol . . . 24
5.4 Syntaktický analyzátor pro jazyk kontrol . . . 24
5.5 Transformace AST jazyka kontrol na SQL AST . . . 24
5.6 Detekce chyb . . . 25
5.7 Ovladač pro transformaci SQL AST do BASE64 řetězce . . . 25
5.8 Načtení z BASE64 řetězce do ovladače příslušné databáze . . . 25
5.9 Konečná trnsformace z AST ovladače na výsledný SQL dotaz . . . 25
5.10 Použité prostředky pro implementaci . . . 25

6 Testování vzniklých SQL dotazů 27
6.1 Úvod . . . 27
6.2 Společnost evosoft a její software použitelný pro testování . . . 27
6.3 Metodika testování . . . 28
6.4 Výsledky fáze 1 . . . 28
6.5 Výsledky fáze 2 . . . 28

7 Závěr 31

A Tokeny pro lexikální analýzu sémantického tvaru jazyka kontrol 33

B LL(1) gramatika sémantického tvaru jazyka kontrol 35

C Vzhled vzorové aplikace dodané ke kompilátoru 39

Seznam obrázků

5.1 Diagram celkové struktury modulů v překladači jazyka kontrol 23

C.1 Vzhled vzorové aplikace dodané ke kompilátoru 40

Seznam tabulek

6.1 Seznam metodik použitých v první části testování a jejich výsledky 28
6.2 Seznam datových zdrojů použitý pro testování ve fázi 2 a jejich výsledky 29

Velké díky patří společnosti evosoft s.r.o, která mi dodala nápad na tuto práci a prostředky pro její otestování. Dále bych rád poděkoval Ing. Štěpánu Plachému, který mi dodal odvahu k vypracování této práce a podporu k jejímu dokončení. Chtěl bych také v neposlední řadě poděkovat Lence za její podporu, bez které by tato práce nikdy nevznikla.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. ledna 2024 .....

Abstrakt

Tato bakalářská práce se zabývá analýzou, návrhem a implementací překladače jazyka kontrol navrženého Českou národní bankou. Tento překladač umožňuje překlad jazyka kontrol do jazyka SQL, který je následně možné spustit přímo nad databázovým serverem, kde se nachází data, která výraz v jazyce kontrol validuje. V rámci práce je provedena analýza jednotlivých částí tohoto jazyka a jejich možnosti pro překlad pro vícero druhů relačních databází. Je vypracován konkrétní návrh struktury tohoto překladače a jeho jednotlivých částí, na základě kterého je překladač implementován pomocí programovacího jazyka PHP.

Klíčová slova překladač, Česká Národní Banka, SDAT, jazyk kontrol, validace, výkaznictví, PHP

Abstract

This bachelor thesis deals with the analysis, design and implementation of compiler for control language designed by the Czech National Bank. This compiler allows the translation of the control language into the SQL language, which can then be run directly over the database server where the data that the control language validates is located. Within the scope of this work, an analysis of the individual parts of this language and their possibilities for compilation for several types of relational databases is performed. A concrete proposal of the structure of the compiler and its individual parts is developed, based on which the compiler is implemented using PHP programing language.

Keywords compiler, Czech National Bank, SDAT, control language, validation, reporting, PHP

Seznam zkratek

SQL	Structured Query Language
ČNB	Česká národní banka
JK	Jazyk kontrol
JVK	Jednovýkazová validace
MVK	Mezivýkazová validace
AST	Abstraktní syntaktický strom

Úvod

Česká národní banka (dále jen ČNB) v rámci sytému SDAT definuje výkazy. Výkaz je formulář, který na základě vykazovacích povinností (taktéž definovaných ČNB) jsou fyzické a právní osoby povinné odevzdávat České národní bance ke stanovenému dni. Tyto formuláře obsahují pole pro zadání hodnot v různých formátech a tvarech (dále popisuji jako buňky), a tyto buňky jsou svázány jistými pravidly.

Tato pravidla se nazývají validace a dělí se na formátové, jednovýkazové a mezivýkazové. Zatímco formátové validace jsou definovány pomocí vlastností buňky (např. délka či datový typ), validace jednovýkazové a mezivýkazové určují vztahy mezi buňkami. Tyto vztahy jsou definovány pomocí jazyka kontrol.

Jazyk kontrol je validační jazyk definovaný Českou národní bankou v rámci systému SDAT, do kterého se výkazy odevzdávají. Jeho syntaxe je podobná excelovým výrazům.

Cílem práce je vytvoření překladače pro jednovýkazové kontroly z jazyka kontrol do SQL dotazů. Překladačem se zde rozumí program, který převede textový výraz z jedné formy do jiné formy tak, aby jeho význam zůstal ekvivalentní. SQL dotaz po doplnění vhodných parametrů týkajících se vybraného výkazu a spuštění nad databází vrátí seznam buněk, které neodpovídají vybranému výrazu validace.

Dalším cílem práce je tento překladač důsledně otestovat, a to nejen vůči virtuálním (vymyšleným) údajům, ale zkusit ho i vůči reálným, a optimálně je porovnat vůči výstupům samotného systému SDAT.

Systém SDAT vznikl kolem roku 2018 a v rámci něj byl poprvé zveřejněn i jazyk kontrol. Validace v něm napsané je možné použít k automatizaci testování výkazu před jejich odesláním, pro což neeviduji jiné existující řešení. Výsledek této práce může usnadnit tvorbu těchto testů, a to kompletně automatizovaně. Osobně jsem se s touto problematikou setkal v pracovním prostředí, kde tyto testy jsou vytvářeny zdlouhavě a ručně.

Tato práce seznámí s detaily o systému SDAT, způsoby překladu jazyka kontrol a následně jeho analýzou. Poté provede samotnou implementací a testováním této implementace vůči samotnému systému SDAT.

Kapitola 1

Jazyk kontrol a systém SDAT

Tato kapitola rozebírá, co je systém SDAT, jak vypadají metodiky v něm definované, a konečně co je to jazyk kontrol a jeho využití v systému SDAT.

1.1 Česká národní banka a systém SDAT

Česká národní banka je státní orgán, který dohlíží na celý finanční trh v České republice. V rámci této povinnosti však taktéž má na starost finančně regulované subjekty, nad kterýma vykonává dohled. Proto je potřeba, aby tyto subjekty ČNB pravidelně vykazovaly potřebné dokumenty a hlásily změny, které je nutné evidovat.[1]

ČNB si tyto povinnosti a jejich četnost vůči jednotlivým subjektům určuje sama a tyto povinnosti se většinou týkají bankovních společností fungujících na území České republiky a obchodníků s cennými papíry. Za nedodržení těchto povinností nebo nesprávnosti či nepřesnosti v datech hrozí subjektům nemalé pokuty, na které má ČNB z hlediska zákona nárok. Pro tyto účely si ČNB zadala zakázku pro vyvinutí systému, který od doby jeho existence je využíván ke sběru a automatickému zpracování požadovaných výkazů od jednotlivých subjektů. Vedlejší funkcí systému je zveřejňování vykazovacích povinností pro jednotlivé subjekty a tvar požadovaných výkazů.[1]

Systém SDAT začal vznikat kolem roku 2018. Jeho veřejná prezentace je umístěna na stránce https://sdat.cnb.cz/. Jeho hlavní výhodou oproti původnímu systému pro výkaznictví je možnost automatizovaného zpracování výkazů a publikované API služby. Ty nově dávají k dispozici veškeré vykazovací povinnosti pro jednotlivé subjekty, digitalizovaný tvar výkazů ve formátu XML a validace, které jsou nově psány v jazyce kontrol, který je na rozdíl od původních slovních popisům strojově čitelný.[1]

ČNB rozděluje nařízení vykazovacích povinností do tzv. metodik. Ty obsahují veškeré informace o tom, jak má výkaz vypadat, jaká pravidla má splňovat a kdo je povinen tyto výkazy nahlašovat. Každá metodika se zaměřuje na jiné aspekty finančního trhu, a tudíž povinnost nahlašovat reporty dané metodiky je vázaná na oblasti, ve kterých se vykazovací subjekt pohybuje, ne vždy však platí, že subjekt je povinen zasílat všechny reporty z dané metodiky.[1]

1.2 Metodika v systému SDAT

Metodika je největší jednotkou systému SDAT. Zahrnuje veškeré informace, které je nutné znát pro výkaznictví v dané oblasti. Metodika se skládá z knihoven, datových výkazů a kontrol.[1]

1.2.1 Metodická knihovna

Knihovna je médium pro sdílení prostředků napříč metodikami. Jedna metodika může používat vícero knihoven a knihovna může být použita vícero metodikami. Knihovna se skládá z následujících komponent.[1]

1.2.1.1 Číselníky

Jedná se o seznamy hodnot používané v rámci výkazů. Tyto číselníky jsou jednoznačně identifikovány pomocí textového identifikátoru a jejich obsahem jsou vždy textové hodnoty, ke kterým ve většině případů existuje i popis, co daná hodnota znamená. Tyto seznamy jsou použity zejména k validaci vykazovaných dat, a to na všech třech úrovních kontrol. Příkladem takového číselníku může být třeba seznam všech zemí světa.[1]

1.2.1.2 Domény

Jedná se vždy o podmnožinu hodnot z vybraného číselníku. Doména je jednoznačně identifikována svým názvem. Obsahem domény je název číselníku, ze kterého jsou hodnoty vybrány, a zároveň seznam vybraných hodnot. Účel domény je stejný jako číselníku samotného, ovšem tento princip zamezuje zbytečné duplikaci hodnot v systému a umožňuje jistou úroveň zanoření. Příkladem domény je seznam zemí Evropské unie, který je vybrán z číselníku všech zemí světa.[1]

1.2.1.3 Datové typy

Datové typy slouží k definici formátu vykazovaných dat. Tyto vlastnosti jsou testovány v rámci formátových validací a jejich nedodržení automaticky zastavuje proces přijetí výkazu Českou národní bankou. Datový typ je opět jednoznačně identifikován svým názvem a možnosti jeho vlastnosti se řídí primárně podle hodnoty v atributu „Základní datový typ“. Ty mohou být:

- Text
- Číslo
- Datum
- Binární

Podle této volby se posléze doplňují další vlastnosti (například pro datum je to „Maska“, „Horní mez“ a „Dolní mez“). Příkladem datového typu může být „všechna celá čísla větší než 10“.[1]

1.2.1.4 Parametry

Každý má název, který je unikátní napříč celou knihovnou a v rámci výkazu může být použit jako dynamický, nebo statický.

Pokud se jedná o statický parametr, jeho hlavní úlohou je rozlišení množiny vykazovaných hodnot od sebe za předpokladu, že mají totožný název. To se může stát například z důvodu, že určitou stejnou hodnotu je nutné vykázat ve dvou různých měnách. Tyto buňky (Ukazatelé) mají jinak stejné vlastnosti, rozdílné jsou pouze v tomto detailu, který SDAT rozdělí pomocí statického parametru.

Pokud se jedná o dynamický parametr, znamená to, že výkaz obsahuje dynamický rozměr. To může například znamenat, že jeden řádek výkazu se zasílá vícekrát pro různé společnosti. Tyto řádky je však nutné od sebe odlišit a zároveň určit, že hodnoty na řádku jsou seskupeny. Dynamický parametr se používá právě pro tento účel.

Výkazy mohou být dynamické po vícero osách a pro identifikaci jednoho dynamického rozměru nemusí stačit pouze jeden dynamický parametr (například když musí být vyplněn jeden řádek pro vícero společností za každý rok jejich existence). Pro tyto případy se využívá kombinace vícero dynamických parametrů. Je zaručeno, že kombinace dynamických parametrů je vždy unikátní v rámci jedné datové oblasti.[1]

1.2.1.5 Ukazatele

Jedná se o buňky určené pro vykazované metody. Jejich identifikace není jednoznačná podle jména, ale teprve kombinací jména a seznamu parametrů, u kterých je zvoleno, zda se jedná o statické, nebo dynamické. U statických parametrů ukazatel už má hodnoty předvyplněny, zatímco u dynamických nikoliv. Ukazatel dále nese informaci, jakého datového typu je.[1]

1.2.2 Výkazy a jejich datové oblasti

Výkaz je nejmenší jednotkou odesílatelnou do ČNB, která v rámci vykazovacích povinností definuje, které subjekty jsou povinné vykazovat které výkazy. Výkaz se skládá z datových oblastí. Datová oblast je vždy ve výkazu alespoň jedna. Datová oblast samotná již obsahuje vyplnitelné ukazatele, které jsou pro zachování struktury rozděleny do os. SDAT umí pracovat pouze se třemi rozměry datové oblasti, a to jsou X, Y a Z. Každá z os je umístěná do jednoho rozměru a je jí určen identifikátor a pořadí, ve kterém se na zvoleném rozměru zobrazí.

Osa zároveň eviduje, zda je statická, nebo dynamická. Pokud je osa statická, tak se na zvoleném rozměru vyskytuje pouze jednou. Její dynamická varianta však znamená, že do zvoleného řádku/sloupce je možné zadat vícero záznamů, což v principu znamená, že osa může být virtuálně duplikována. Dynamická osa ovšem nemusí v odeslaných datech existovat vůbec za předpokladu, že v ní nejsou vyplněná žádná data. Pro rozlišení jednotlivých instancí v dynamické ose je nutné do ní přidat dynamické parametry, které pak slouží jako jednoznačný identifikátor zvoleného řádku/sloupce.

Příkladem dynamické osy může být řádek obsahující seznam provedených obchodů, kdy osa má v sobě dynamický parametr s číslem obchodu a ukazatele, s kým byl obchod proveden a kolik peněz v obchodě bylo. Pokud subjekt za vykazované období měl obchodů 10, odešle tuto osu 10x, pokud však neměl obchod žádný, osa nebude poslána vůbec.[1]

1.2.3 Kontroly

Těmato kontrolami jsou myšleny jednovýkazové a mezivýkazové kontroly (dále jen jako JVK a MVK). Zatímco formátové kontroly jsou určeny parametry datového typu, JVK a MVK určují již vztahy mezi jednotlivými ukazateli v rámci výkazu (v případě JVK) nebo celé metodiky (v případě MVK). Tyto kontroly jsou zapsány v tzv. jazyce kontrol, který systém SDAT definuje. Zatímco MVK jsou umístěny volně v metodice, JVK se nacházejí uvnitř výkazů.

V situaci odeslání výkazů do ČNB se výkaz validuje. Validace se provádějí v pořadí:

- Formátové validace
- JVK
- MVK

Selhání validace znamená, že výkaz nemá data v pořádku a je nutné ho opravit. Pokud kontrola selže ve formátových validacích nebo JVK, výkaz je automaticky odmítnut a následující úrovně se nespouští (například po selhání jedné formátové validace se už nespustí žádné JVK a je potřebné poslat opravu výkazu, aby validace byly opět spuštěny). V případě MVK je situace složitější, protože MVK může čekat na odeslání zbývajících výkazů, které ještě ČNB neobdržela.[1, 2]

1.3 Jazyk kontrol

Jazyk kontrol je výrazový jazyk definující vztahy mezi jednotlivými parametry a ukazateli ve výkazech (přesněji v datových oblastech). V systému SDAT slouží primárně jako unifikovaný způsob vyjádření prováděných validací. Předchůdce systému SDAT měl validace definovány pouze slovním popisem, který se, jak známo, při špatné komunikaci dá vyložit mnohými způsoby. Dokumentace jazyka kontrol je dostupná na webu cnb.cz.[2]

Jazyk v mnoha ohledech připomíná excelové výrazy, kdy validace jako celek tvoří jeden výraz, jehož návratovou hodnotou je úspěch (pokud validace proběhla úspěšně), nebo neúspěch (pokud validace objevila chyby). Jazyk obsahuje sám o sobě dvě iterační funkce sloužící k iteraci nad dynamickými nebo statickými parametry, obsahuje aritmetické operace a má možnost definovat proměnné (jak jsou nazvány v dokumentaci), které reálně ale v jazyce fungují jako konstanty pro zabránění opakovanému vyhodnocení výrazu.

Zbytek jazyka je definován pomocí velkého množství funkcí. Dá se říct, že jádro jazyka bylo navrženo tak, aby splnilo veškeré potřeby pro validace, však veškeré detailnější operace jsou definovány pomocí funkcí, které je možné snadno dodefinovat v rámci implementace jazyka. Jazyk samotný neumožňuje uživatelsky definované funkce.

Jazyk kontrol má dohromady čtyři varianty, ovšem dají se vyjádřit jako kombinace dvou binárních faktorů. Prvním je, zda se jedná o JVK, nebo MVK, druhým, zda se jedná o sémantický, nebo uživatelský tvar.

Hlavním rozdílem mezi JVK a MVK výrazy v jazyce kontrol je, že JVK výrazy neobsahují pro identifikaci ukazatele/parametru informaci o reportu, ze kterého je validovaná buňka chtěná. Tyto validace musí již v rámci spuštění přijmout informaci, nad kterým reportem se spouští. V této práci se primárně rozebírá tato varianta.

MVK validace oproti tomu nejsou vázány k žádnému výkazu. Při každé identifikaci buňky u ní musí být uvedeno, ve kterém konkrétním reportu je buňka validována. Ovšem toto označení reálně znamená jen relativní přechod, protože reálně v systému SDAT je od jednoho vykazovacího subjektu mnoho instancí výkazu stejného typu. V tu chvíli se validace spouští nad tzv. skupinou MVK, která sama o sobě seskupuje vícero validací MVK a vícero instancí výkazů. Tyto výkazy mají (ve většině případech) společné období, za které byly odesílány, a je díky tomu možné je snadno seskupit.

Uživatelský tvar se od sémantického tvaru liší taktéž ve způsobu identifikace buněk. Uživatelský tvar je určen spíš pro čtení lidmi, přičemž obsahuje vždy název datové oblasti, ze které je buňka brána, číslo řádku, na kterém se nachází a číslo sloupce na kterém se nachází. Je však mnoho situací, kdy tento systém identifikace nestačí, díky čemu se v některých validacích napsaných v uživatelském tvaru vyskytuje použití identifikace pomocí způsobu v sémantickém tvaru. Implementace kompilátoru pro uživatelský tvar by proto musela obsahovat i implementaci pro sémantický tvar jazyka kontrol. Uživatelský tvar svým formátem nerozlišuje parametry od ukazatelů.

Sémantický tvar místo toho používá k identifikaci názvy. Zatímco parametr má unikátní identifikátor, ukazatelé bohužel nemají. Název parametru je při identifikaci umístěn do dvojitých hranatých závorek, zatímco název ukazatele do jednoduchých. Při identifikaci ukazatele za názvem ukazatele následuje seznam parametrů a jejich hodnoty. Tento seznam však nemusí jednoznačně identifikovat ukazatel, může označit jejich množinu, nad kterou je posléze možné provádět agregační funkce.[2]

Kapitola 2

Metoda analýzy kódu pomocí LL(1) překladačů

Tato kapitola seznamuje s principy překladačů a konkrétně rozebírá LL(1) metodu analýzy počítačových jazyků, její komponenty a celkovou strukturu.

2.1 Překladač jazyků

Překladač jazyků je softwarová aplikace nebo online služba, která umožňuje uživatelům překládat text z jednoho jazyka do druhého. Tyto nástroje jsou určeny k rychlému a snadnému překladu psaného textu a lze je použít k různým účelům, například ke komunikaci s lidmi, kteří hovoří různými jazyky, k překladu dokumentů nebo webových stránek nebo k jednoduchému učení se nového jazyka. Ovšem překladač může být software, který překládá z jednoho programovacího jazyka do jiného, nebo dokonce do strojového kódu, kterému rozumí přímo počítač.

Překladače jazyků používají k překladu textu různé techniky, včetně systémů založených na pravidlech, statistických modelech a neuronových sítích. Některé nástroje poskytují přesnější překlady než jiné a kvalita překladu může záviset na řadě faktorů, jako je složitost textu/kódu, překládaný jazykový pár a kontext textu.[3]

2.1.1 Překladač programovacích jazyků

Překladač počítačových jazyků je softwarový program, který převádí zdrojový kód napsaný ve vysokoúrovňovém programovacím jazyce do strojového kódu, který může být přímo spuštěn počítačem, nebo do jiného programovacího jazyka, který je už možné zpracovat jiným kompilátorem nebo interpreterem. Proces převodu vysokoúrovňového zdrojového kódu na strojový kód se nazývá kompilace a nástroj, který tento úkol provádí, se nazývá kompilátor.

Existují různé typy překladačů počítačových jazyků, včetně kompilátorů, interpreterů a assemblerů. Kompilátory překládají zdrojový kód do strojového kódu najednou, zatímco interpretery provádějí zdrojový kód řádek po řádku. Assemblery překládají kód assembleru do strojového kódu.

Účelem překladače počítačových jazyků je umožnit vývojářům softwaru napsat kód ve vysokoúrovňovém programovacím jazyce, který je snáze čitelný a srozumitelný než strojový kód, a nechat tento kód přeložit do strojového kódu, který může být přímo spuštěn počítačem. Vývojáři tak mohou snadněji a efektivněji vytvářet složité softwarové aplikace.[3]

Psaní validací do systému SDAT by bylo velice náročné vzhledem k velkým rozdílům mezi jednotlivými databázovými softwary a jejich podporovaným funkcím. Proto v případě systému

SDAT byla zvolena varianta vytvoření si vlastního vysokoúrovňového jazyka, který bude snadné napsat, jednoduché číst a možné překládat pomocí překladačů do různých variant.

2.2 LL(1) analýza kódu

Analýza LL(1) je typ syntaktické analýzy nebo techniky parsování používané v informatice k určení, zda lze daný řetězec tokenů odvodit z dané bezkontextové gramatiky. LL(1) je zkratka pro „Left-to-right, Leftmost derivation, 1 lookahead symbol“.

Při analýze LL(1) čte parser vstupní řetězec zleva doprava a k sestavení stromu parsování používá nejlevější derivaci gramatiky. Parser používá look-ahead jednoho symbolu, aby se rozhodl, které produkční pravidlo použije v každém kroku.

Analýza LL(1) vyžaduje, aby parsovaná gramatika byla sama o sobě LL(1), což znamená, že je jednoznačná a lze ji parsovat pomocí jediného tokenu při pohledu dopředu. Toho se dosahuje mimo jiné odstraněním levé rekurze a faktorizací gramatiky.

Parsery LL(1) se běžně používají při implementaci programovacích jazyků, protože jsou relativně jednoduché na implementaci a dokážou zpracovat velkou třídu bezkontextových gramatik. Mají však některá omezení, jako například nemožnost zpracovat levorekurzivní nebo nejednoznačné gramatiky. Oproti tomu samotná analýza textu je lineární, a proto vcelku rychlá.

Parser LL(1), nazývaný také analyzátor LL(1), se obvykle skládá z několika modulů nebo komponent, které společně provádějí proces parsování. Mezi tyto moduly patří:

- Lexer neboli lexikální analyzátor: Tento modul je zodpovědný za rozdělení vstupního zdrojového kódu na posloupnost tokenů, z nichž každý představuje určitou lexikální jednotku, například klíčové slovo, identifikátor nebo operátor.
- Parser neboli syntaktický analyzátor: Tento modul používá algoritmus parsování LL(1) k sestavení stromu parsování z posloupnosti tokenů vygenerovaných lexerem. Parser používá sadu parsovacích tabulek, které definují produkční pravidla a lookahead symboly pro každý neterminální symbol v gramatice.
- Obsluha chyb: Tento modul je zodpovědný za detekci a hlášení syntaktických chyb ve vstupním zdrojovém kódu. Při zjištění chyby obsluha chyb vygeneruje příslušnou chybovou zprávu a provede odpovídající akci, například ukončí proces parsování.

Celkově je analyzátor LL(1) složitý systém, který zahrnuje více komponent spolupracujících při provádění procesu parsování. Díky rozdělení procesu parsování na jednotlivé moduly, z nichž každý je zodpovědný za konkrétní úkol, je analyzátor LL(1) schopen efektivně a přesně analyzovat zdrojový kód napsaný v daném programovacím jazyce.[3, 4]

2.2.1 Lexikální analyzátor

V analýze LL(1) je lexer, známý také jako skener nebo tokenizér, prvním modulem v procesu analýzy. Jeho hlavní funkcí je přečíst vstupní zdrojový kód a rozdělit jej na posloupnost tokenů, které jsou poté předány parseru k dalšímu zpracování.

Token je posloupnost znaků, která představuje jednu lexikální jednotku v analyzovaném programovacím jazyce, například klíčové slovo, identifikátor, operátor nebo interpunkční znaménko. Lexer rozpozná každý token tak, že analyzuje vstupní zdrojový kód po jednotlivých znacích a pomocí sady pravidel nebo regulárních výrazů přiřadí každé sekvenci znaků odpovídající typ tokenu.

Lexer pak pro každý rozpoznaný token vytvoří objekt tokenu, který obsahuje informace o typu tokenu, hodnotě a pozici ve zdrojovém kódu. Objekty tokenů jsou obvykle uloženy v proudu tokenů nebo ve vyrovnávací paměti, k nimž má parser přístup a může z nich sestavit strom rozboru.

Lexer je důležitou součástí procesu analýzy LL(1), protože zjednodušuje úlohu parsování tím, že vstupní zdrojový kód rozkládá na menší, lépe zvládnutelné jednotky. To parseru usnadňuje rozpoznání struktury kódu a konstrukci stromu parsování, který reprezentuje syntaxi programu.[3, 4]

2.2.2 Syntaktický analyzátor

V analýze LL(1) je parser nebo syntaktický analyzátor modul, který vezme posloupnost tokenů vytvořenou lexerem a zkonstruuje parsovací strom, který reprezentuje syntaktickou strukturu analyzovaného programu.

Parser používá k analýze posloupnosti tokenů a konstrukci stromu parsování sadu pravidel neboli produkčních pravidel, která definují gramatiku analyzovaného programovacího jazyka. Tato pravidla určují, jak mohou být neterminální symboly v gramatice nahrazeny jinými symboly nebo posloupnostmi symbolů, což nakonec vede ke konstrukci stromu parsování.

Parser LL(1) používá k implementaci algoritmu parsování přístup založený na tabulkách. Parsovací tabulka je generována z gramatiky a obsahuje informace o produkčních pravidlech a lookahead symbolech pro každý neterminální symbol v gramatice. Parser používá tuto tabulku k určení vhodného produkčního pravidla, které se má použít na základě aktuálního neterminálního symbolu a následujícího lookahead symbolu.

Pokud je vstupní posloupnost tokenů syntakticky správná, parser zkonstruuje parsovací strom, který reprezentuje syntaktickou strukturu programu. V opačném případě, pokud vstupní posloupnost obsahuje syntaktické chyby, parser tyto chyby odhalí a oznámí je uživateli.

Parser je kritickou částí procesu analýzy LL(1), protože převádí posloupnost tokenů na strukturovanou reprezentaci, kterou lze použít pro další analýzu nebo překlad.[3, 4]

2.2.2.1 Abstraktní syntaktický strom

Abstraktní syntaktický strom (neboli AST) je datová struktura používaná v informatice k reprezentaci syntaktické struktury programu. Jedná se o stromovou strukturu, která zachycuje základní prvky syntaxe programu, přičemž se odprošťuje od detailů, jako jsou bílé znaky, komentáře a formátování.

AST vytváří parser nebo syntaktický analyzátor jako součást procesu kompilace po tokenizaci vstupního zdrojového kódu. AST představuje strukturu programu ve formě, která je lépe přístupná analýze a optimalizaci.

Každý uzel v AST představuje syntaktický prvek programu, například výraz, příkaz nebo deklaraci. Uzly jsou spojeny hranami, které představují vztahy mezi nimi, například vztahy rodič-dítě a sourozenecké vztahy.

AST se používá v mnoha různých kontextech, včetně překladačů, interpreterů a nástrojů statické analýzy. Poskytuje pohodlný způsob manipulace se syntaxí programu a lze jej použít k provádění široké škály úloh, jako jsou generování kódu, optimalizace a kontrola chyb.[3, 4]

2.3 Finální převod do výsledného kódu

Překladač vygeneruje konečný kód z AST. To zahrnuje mapování kódu v AST na specifické instrukce, které lze spustit na cílové platformě, například x86 nebo ARM (nebo v případě této práce na databázi).[3, 4]

Kapitola 3

Databáze a jazyk SQL

Tato kapitola rozebírá, co jsou to databáze a co je jazyk SQL, do kterého kompilátor tvořený v této práci bude výrazy kompilovat.

3.1 Databáze

Databáze je soubor souvisejících dat, která jsou uspořádána strukturovaným způsobem tak, aby je bylo možné snadno ukládat, přistupovat k nim, spravovat je a aktualizovat. Databáze se používají k ukládání a správě velkého množství dat a jsou klíčovou součástí mnoha softwarových aplikací a systémů.

V databázi jsou data obvykle uspořádána do tabulek, které se skládají z řádků a sloupců. Každý řádek v tabulce představuje jeden záznam, zatímco každý sloupec představuje určitý údaj, například jméno, datum nebo číslo. Sloupce v tabulce jsou definovány schématem tabulky, které určuje datový typ, velikost a omezení pro každý sloupec.

Databáze lze obecně rozdělit na dva typy: relační a nerelační. Relační databáze používají ke správě dat a manipulaci s nimi jazyk SQL (Structured Query Language), zatímco nerelační databáze používají jiné datové modely, například model klíč-hodnota, dokument nebo graf.

Databáze se používají v široké škále aplikací, od jednoduchých systémů pro správu kontaktů až po rozsáhlé systémy na podnikové úrovni, které používají banky, letecké společnosti a další organizace. Mezi běžné příklady databází patří MySQL, Oracle, MongoDB a PostgreSQL.[5, 6, 7]

3.2 Relační databáze

Relační databáze je typ databáze, která ukládá a organizuje data strukturovaným způsobem na základě relačního modelu dat. V relační databázi jsou data uložena v tabulkách, které spolu souvisejí na základě společných datových atributů nebo klíčů.

Relační databáze používají sadu pravidel nazývanou normalizace, která zajišťuje efektivní a konzistentní ukládání dat. Normalizace zahrnuje rozdělení velkých tabulek na menší, lépe spravovatelné tabulky a vytvoření vztahů mezi nimi, aby bylo možné data rychle a přesně vyhledávat a aktualizovat.

Relační databáze používají ke správě a manipulaci s daty jazyk SQL.[7]

3.3 Jazyk SQL

SQL (Structured Query Language) je programovací jazyk používaný pro správu a manipulaci s daty v relačních databázích. SQL se používá k vytváření, úpravám a dotazování databází a používají ho vývojáři, správci databází a analytici pro práci s daty uloženými v relačních databázích.

Jazyk SQL je deklarativní jazyk, což znamená, že se používá spíše k popisu toho, jaká data mají být vyhledávána nebo jak s nimi má být manipulováno, než jak to má být provedeno. Díky tomu je jazyk SQL snadno použitelný a srozumitelný i pro ty, kteří nemají zkušenosti s programováním.

Jazyk SQL je v průmyslu široce používán a je podporován velkým počtem relačních systémů pro správu databází, včetně Oracle, MySQL, SQL Server, PostgreSQL a SQLite. Je to výkonný a všestranný jazyk, který umožňuje uživatelům pracovat s daty různými způsoby a je základním nástrojem pro každého, kdo pracuje s relačními databázemi.[7, 6]

Kapitola 4
Analýza jazyka kontrol

Tato kapitola probírá samotnou strukturu jazyka kontrol a možné zpracování jeho jednotlivých komponent.

4.1 Syntaxe a gramatika jazyka kontrol

Základní popis syntaxe sémantického tvaru jazyka kontrol je popsán v kapitole 4 jeho dokumentace, kde je popis základních konstrukcí, způsobů identifikace jednotlivých údajů a parametrů a samotná práce s daty. Co zde bohužel není, je zápis gramatiky. Ta je dostupná pouze pro uživatelský tvar, který se bohužel v rámci této práce nezpracovává. Tato gramatika ovšem dává dobrý náhled na to, co je od jazyka očekáváno, a dá se od ní odrazit. Gramatika uživatelského tvaru je k dispozici v dokumentaci jazyka kontrol v sekci 5.2.[2]

Gramatika dodaná dokumentací bohužel potřebuje několik úprav z důvodů zmíněných výše. První z nich je úprava gramatiky na sémantický tvar, druhá je rozdělení tokenů od gramatických pravidel a třetí je úprava gramatiky na LL(1) podobu.

4.1.1 Úprava gramatiky z uživatelského na sémantický tvar

Úprava na sémantickou podobu spočívá v nahrazení výběrových výrazů ukazatelů a parametrů za jejich sémantickou podobu a úprava výrazu FOR, aby byl schopný pracovat se statickými parametry, se kterými uživatelský tvar nepočítá. Výraz FILTER_CHECK může zůstat beze změny, protože jeho syntaxe je používaná stejně i v uživatelském tvaru.

4.1.2 Oddělení tokenů od gramatických pravidel

Oddělení tokenů od gramatických pravidel je nutné provést pro všechny prvky v sekci „Pravidla základních elementů“ v dodané gramatice.[2] U některých těchto elementů dochází k jejich sloučení do jednotlivých tokenů (např. v případě gramatiky pro <DIGIT>). Některé výrazy jsou brány automaticky jako klíčová slova jazyka (např. <NULL>). Po tomto kroku je nutné vyjmout z gramatiky veškeré volně použité symboly a nahradit je tokeny (toto platí pro všechny operátory).

Naopak co nelze vyjmout jako tokeny, jsou ukazatele a parametry, které nejsou ještě jednoznačně rozhodnutelné na úrovni lexikální analýzy, proto jsou skládány z několika tokenů typu závorek, identifikátorů a operátorů.

Je nutné zmínit, že ukazatel samotný může obsahovat výrazy jazyka kontrol uvnitř sebe a je na základě nich vybírána jen určitá množina výsledných ukazatelů.

Z těchto úprav už máme k dispozici kompletní seznam tokenů, který je k dispozici v příloze A.

4.1.3 Úprava gramatiky na LL(1) podobu

Posledním krokem je úprava na LL(1) gramatiku. Ačkoliv to z původní gramatiky ani dokumentace není na první pohled zřejmé, operátory v jazyce kontrol mají obdobné přednosti, jako v jiných programovacích jazycích. Přednosti jsou v tomto případě vždy levé a operátory v původní gramatice mají vyšší přednost, pokud jsou v gramatice umístěny níže než jejich konkurent.

Co zde vzniká za zajímavost je, že dodaná gramatika vůbec nepočítá se zřetězením operátorů sčítání, násobení, dělení a násobení. Ovšem z příkladů v dokumentaci můžeme jasně identifikovat, že jazyk s tím počítá. Proto je nutné gramatiku opět adekvátně upravit, aby tato vlastnost byla umožněna. Ve finální podobě gramatiky proto musí být počítáno nejen s předností operátorů, ale i jejich zřetězením.

O co se konkrétně není nutné příliš zajímat, jsou pragma výrazy, které v jazyce kontrol fungují jako příkazy pro preprocesor. Ty je sice nutné rozebrat, ovšem ve výsledku jsou pouze dodány v textové podobě v páru název a obsah.

Úpravy na LL(1) jsou prováděny podle standartních pravidel pro převod bezkontextové gramatiky na LL(1) gramatiku, a to pomocí pravidel odstranění levé rekurze, levé faktorizace, rohové substituce a pohlcení terminálního symbolu. Výsledná gramatika je k vidění v příloze B.

4.2 Problém překladu z jazyku kontrol do SQL dotazu

Jazyk SQL je velmi odlišný od jazyka kontrol a jejich jednotlivé prvky syntaktických stromů rozhodně nejsou ekvivalentní. Ovšem jsou způsoby, jak je na sebe napárovat. Z toho důvodu je vhodné si z AST jazyka kontrol nejdříve vytvořit AST SQL a ten až následně převádět na finální SQL dotazy. To má i tu výhodu, že bude finální výsledek možné snadno interpretovat do více databázových standardů, a ne jen do jednoho.

V následujících sekcích je pojednáváno o jednotlivých prvcích jazyka kontrol, a jak je tyto prvky možné transformovat do SQL AST.

4.3 Analýza základních prvků jazyka kontrol

V jazyce kontrol je definováno v rámci základní syntaxe hned několik výrazů. V této sekci je analyzováno, jak fungují a jakým způsobem bude probíhat jejich překlad do jazyka SQL.

4.3.1 Aritmetické operátory

Jedná se o operátory sčítání, odčítání, násobení a dělení. Díky tomu, že jejich přednosti jsou řešeny na úrovni parseru, zde tento problém už nemusíme řešit. V případě těchto operátorů existuje jejich ekvivalent v jazyce SQL, proto překlad je přímý. Pro jejich použití si v AST SQL definujeme aritmetický operátor.

4.3.2 Porovnávací operátory

S porovnávacími operátory je to stejné jako s operátory aritmetickými, a to v tom, že v jazyce SQL existuje jejich ekvivalent. Bohužel tento ekvivalent není možné vždy použít kvůli klíčovému slovu MARGIN.

Pokud se klíčové slovo MARGIN nachází těsně za operátorem porovnání, znamená to, že porovnání je validní i v případě, že absolutní rozdíl mezi hodnotami je menší než číslo uvedené za klíčovým slovem MARGIN.

Pokud je tedy klíčové slovo přítomno, je nutné výraz aritmeticky upravit tak, aby podmínka byla splněna.

Samotné operátory po aritmetické úpravě již fungují stejně jako v jazyce SQL. V SQL AST pro ně zadefinujeme jejich vlastní prvek.

4.3.3 Množina a výběr číselníku

Množina a výběr číselníku jsou prvky jazyka kontrol, které vrací seznam hodnot. Jejich překlad proto bude probíhat do jednosloupcového selectu.

Pro oba tyto prvky však bude zapotřebí zadefinovat dva různé prvky. Zatímco množina může pracovat s libovolným výčtem hodnot, včetně hodnot z číselníku, číselník vyžaduje načtení seznamu z externího zdroje.

4.3.4 Výraz FILTER_CHECK

FILTER_CHECK je speciální výraz aplikující iteraci nad jednotlivými záznamy v dynamické části výkazu.[2] Princip je, že nad každým řádkem zvlášť spustí podmínku, a pokud jeden ze řádků podmínku nesplní, je celý výraz vyhodnocen jako chybný. FILTER_CHECK má možnost i vybrat pouze výčet konkrétních řádků, nad kterými se validace spustí.

V případě této práce je však význam FILTER_CHECK rozšířen o schopnost sdělit, které konkrétní řádky validací neprošly. Jeho samotná implementace vypadá jako SELECT z konkrétního datového zdroje, kde jako podmínka WHERE je kombinace podmínky pro spuštění validace nad konkrétním řádkem a negace podmínky pro každý řádek.

4.3.5 Statický výraz FOR

Výraz FOR situaci dost kompiluje kvůli jeho hned dvěma různým variantám. Varianty jsou statický for a dynamický for.[2]

Statický FOR dostává seznam statických parametrů a rozsahy hodnot, kterých mají tyto parametry nabývat. Pokud je posléze zmíněn ukazatel X ve validaci, tak ukazatel X je zvalidován a tak jsou validovány ve výrazu všechny ukazatele, které mají jako kód vyplněný X a mají jako statické parametry libovolnou kombinaci statických parametrů uvedených ve FOR výrazu. V tomto případě se všechna X nemusí nacházet ve stejné části výkazu.[2]

Tato varianta neexistuje v uživatelském tvaru jazyka kontrol, ovšem dá se z ní vzít příklad pro implementaci. V uživatelském tvaru není možné iterovat nad statickým seznamem ukazatelů, proto jsou v něm vždy vypisovány jednotlivé kombinace validací a všechny jsou spojeny pomocí AND operátoru.

Tento postup znamená, že překlad porovnávacího výrazu je aplikován na každou kombinaci statických parametrů zvlášť, ovšem finální výsledek je možné interpretovat jako select nad jedním řádkem naplněným statickými hodnotami a nad tímto řádkem spustit jednu kombinovanou podmínku sestrojenou překladem pro každou kombinaci.

4.3.6 Dynamický výraz FOR

Dynamický FOR se od statického liší tím, že přijímá seznam dynamických parametrů místo statických. Dopadem této vlastnosti je, že dynamický FOR se chová jako FILTERCHECK, ovšem stejně jako ve statickém FOR není zaručeno, že všechny ukazatele se budou nacházet ve stejné části výkazu. V tomto případě je nutné spojit jednotlivé části dohromady tak, aby dynamické řádky měly vždy shodu v dynamických parametrech, které byly ve for výrazu vyplněny.[2]

Překlad může proběhnout stejně jako ve FILTER_CHECK, ovšem musíme umožnit v rámci SELECT možnost výběru hned z několika dynamických zdrojů, které je pak vhodnou podmínkou nutné spojit.

4.3.7 Výraz WITH

WITH je jeden z jednodušších výrazů. Obsahuje pouze seznam statických parametrů s hodnotami. Tyto hodnoty jsou následně aplikovány na všechny ukazatele nacházející se ve validaci.[2]

Jeho jediný smysl je ušetření si práce s výpisem statických parametrů za předpokladu, že mnoho ukazatelů je má totožné mezi sebou.

Vzhledem k tomu, že je používán pouze k identifikaci ukazatelů, není nutné ho do překladu přidávat

4.3.8 Proměnné

Proměnné jsou reálně výrazy, které jsou definovány mimo dosah výrazů upravujících způsoby hledání ukazatelů (WITH, FOR, FILTER_CHECK).[2] Nejen, že se pomocí nich dá validační kód zkrátit, pokud se v něm výraz opakuje, ale mohou se pomocí nich vybírat ukazatele, které by jinak byly nedostupné právě kvůli přítomnosti modifikátorů parametrů.

Jejich reálný překlad je možné provést jako unární operátor závorky, kdy při každém výskytu proměnné v kódu je výraz překládán znova. V rámci překladu vnitřku proměnné je nutné ignorovat veškeré modifikátory parametrů.

4.3.9 Ukazatel

Překlad ukazatele je nutné rozdělit hned na tři části. První je formát. Je počítáno s tím, že veškeré datové zdroje budou vstupovat jako textové hodnoty (neboli formát, ve kterém je očekává systém SDAT), proto je potřeba provést převod hodnoty ukazatele na správný datový typ pomocí operátoru přetypování. V případě dat je potřeba získat formát data a provést konverzi pomocí něj. Dále v případě celočíselného nebo desetinného datového typu je potřeba hodnotu přenásobit hodnotou jednotek, protože hodnoty mohou být udávány například v tisících.

Druhou částí je registrace potřeby datového zdroje, ve kterém se ukazatel nachází. Pokud datový zdroj byl již zaregistrován, je stejně nutné do něj doplnit požadavek na konkrétní ukazatel.

Třetí částí je volba názvu. Ukazatele nemají nekombinovaný jednoznačný identifikátor, ale jejich identifikace vzniká kombinací kódu ukazatele a jeho statických parametrů. Pro tento princip je tedy nutné, aby si datový zdroj zvolil jednoznačný textový identifikátor ukazatele, který je unikátní, pokud možno napříč všemi metodikami.

Kvůli těmto všem komplikacím je vhodné pro ukazatel vytvořit samostatný prvek v SQL AST.

4.3.10 Dynamický parametr

Dynamický parametr trpí stejnými problémy jako ukazatel. Z toho důvodu je jeho překlad a zpracování vhodné udělat stejně, jako tomu je u ukazatelů.

Jediným rozdílem je systém vyhledávání, ovšem to je problematika určená spíše pro definici výkazu.

Unikátnost zvoleného jména by měla být zachována i v rámci parametrů. Neměl by existovat parametr se stejným názvem jako ukazatel. Stejný parametr v různých částech výkazu musí mít různé pojmenování (jedno pro jednu část, jedno pro druhou, jedná se reálně o různé validované buňky a je potřeba je odlišit).

4.4 Analýza funkcí jazyka kontrol

Jazyk kontrol sice nemá mnoho zadefinovaných výrazů, ovšem hlavním důvodem tohoto jevu je, že mnoho výrazů bylo přesunuto do funkcí (např. AND, OR, IF). Téměř všechny tyto funkce je vhodné překládat do AST SQL jako volání funkce. Je očekáváno, že AST SQL bude pro různé ovladače provádět různé překlady jednotlivých funkcí.

Tato sekce pojednává o možných zpracování těchto funkcí, při již finálním překladu z SQL AST do finálního SQL.

4.4.1 Funkce if

Funkce if se v základu jeví jako jednoduchá pro překlad, ovšem v jazyce SQL má dva možné překlady.

První je pomocí syntaxe CASE WHEN. Tato syntaxe se jeví jako vhodná, protože dělá přesně to, co dokumentace požaduje.

Druhý zápis je pomocí kombinace binárních operátorů AND a OR a unární negace. Tato varianta se jeví na první pohled nepoužitelně, protože není schopná zvládnout případy, kdy chceme na základě podmínky vrátit jednu z hodnot.

Do toho bohužel ale zasahuje Oracle, který nemá datový typ boolean.[5] Místo něj se používá číslo o hodnotě 0 nebo 1. V případě použíti první varianty se dostáváme do situací, kdy potřebujeme provést 1 AND (bool), což Oracle nezvládne. Druhá varianta zase ale nezvládá výše zmíněný výběr mezi dvěma hodnotami.

Řešením je kombinace obojího, a to na základě typu hodnoty, který by if měl vrátit. Pokud hodnota je boolean, je vhodné použít překlad druhého typu. Pokud cokoliv jiného, je vhodné použít variantu první.

4.4.2 Funkce filter

Tato funkce se dá přeložit jako jednoduchý select, do kterého je vložen datový zdroj, a podle podmínky jsou hodnoty vyfitrovány.

4.4.3 Funkce filter_apply

Tato funkce se podle dokumentace jeví jako hodně komplikovaná, realita je ovšem jinde. Cílem funkce je, že z tabulky vybere pouze 1 konkrétní sloupec a ten vrátí jako výpis hodnot po řádcích. Implementace je možná pomocí SELECT výrazu v jazyce SQL.

4.4.4 Funkce dostupné v jazyce SQL

Mnoho funkcí v jazyce kontrol mají svůj ekvivalent v jazyce SQL. Některé je sice nutné přejmenovat, ale to již není tak závažný problém. Mezi tito funkce patří například funkce LEFT.

4.4.5 Do definovatelné funkce v jazyce SQL

Některé funkce sice nemají svůj ekvivalent v jazyce SQL, ale dají se vydefinovat procedury na databázové úrovni, které tuto funkci budou plnit. Mezi takové funkce patří například RC_CHECK, která ověřuje, zda je rodné číslo platné.[2]

Tato sada funkcí má proto vlastní prvek v AST SQL, který je určený pro spojování tabulek a sám o sobě slouží jako zdroj tabulky.

4.4.6 Agregační funkce

Agregační funkce se vyskytují i v jazyce SQL, je ovšem nutné pro ně udělat jisté úpravy, aby odpovídaly definičně přesně tak, jak je požadováno dle dokumentace.[2] Například funkce COUNT a COUNTA rozlišuje fakt, zda počítají prázdné buňky do počtu.

Druhý problém agregačních funkcí je ten, že občas nemusí být použity jako agregační, ale jako funkce, co vykoná její činnost s ukazateli, které jsou v ní vyjmenovány jako parametry. V této situaci je vhodné funkci přeložit jako příslušný operátor, nebo její jinak pojmenovaný protipříklad v databázi. Takovým příkladem může být pro funkci MAX funkce GREATEST.

4.4.7 Booleaonvské operátory

Jazyk kontrol implementuje operátory AND a OR jako funkce.[2] V jazyce SQL jsou vedeny jako operátory, proto překlad bude proveden do nich.

4.4.8 Funkce spojující tabulky

Jedná se přesně o funkcionality, jak je známe z JOIN výrazů v jazyce SQL[2, 7], jejich překlad je proto možné provést přímo do nich.

Jediný faktor vyžadující pozornost je rozdíl mezi funkcemi *_CHECK as *_APPLY. Zatímco APPLY varianty jsou jen čisté spojování datových tabulek, CHECK varianty obsahují navíc podmínku, jejíž negativní vyhodnocení na řádku vede k selhání celé validace.

4.4.9 Funkce doplňující proměnné hodnoty do validace

V jazyce kontrol je několik funkcí, které mají být nahrazeny konstantou doplněnou za vykazující subject.[2] Příkladem takové funkce je REF_OSOBA_LEI. V případě těchto funkcí je na jejich místo možné doplnit zatupný řetězec s dvojtečkou na začátku. Pří spuštění vygenerovaného SQL se tento zástupný text bude jevit jako proměnná potřebná doplnění, jako je tomu při běžné přípravě SQL dotazu a následném spuštění nad SQL serverem.

4.4.10 Funkce vynechané z této práce

Je několik funkcí, které jsou z práce vynechány, a to z různých důvodů. První z těchto funkcí je LINREG, která má počítat, zda hodnota je v souladu lineární regrese. Popis této funkce autorovi práce moc neřekl, proto byl odeslán e-mail na ČNB s dotazem, jak má funkce fungovat. Odpověď byla dost neurčitá, ovšem po přijetí odpovědi byly v systému SDAT vypnuty všechny validace tuto funkci používající.

Další sadou neimplementovaných funkcí je sada IAF_*. Tyto funkce mají provádět podobné chování, jako klíčové slovo MARGIN, ovšem odchylka je místo v rámci kódu jazyka kontrol uvedena v jednotlivých ukazatelích. Autor této práce si není jist, jak tyto odchylky správně kombinovat, a odeslání dalšího e-mailu se obával.

Vynechány jsou taktéž funkce pracující s externími číselníky, protože se autorovi práce nepovedlo dohledat jejich relevantní legální cenově dostupný zdroj.

Poslední sadou vynechaných funkcí jsou funkce určené pro práci s MVK validacemi. Těmito se tato práce zatím nezaobírá.

4.5 Nutné prvky v SQL AST

Z výše zmíněné analýzy vznikl seznam prvků nutných pro úspěšnou tvorbu SQL AST. V této sekci je tento seznam podrobněji rozebrán.

- Kořen validace
  - Jedná se o kořenový prvek starající se o transformaci výsledků do seznamu datových zdrojů a jejich řádků, ve kterých nastala chyba.
- Aritmetický operátor
  - Prvek překládající se na jeden ze čtyř aritmetických operátorů v jazyce SQL.
- Transformační operátor
  - Prvek zaobalující výraz, který po vyhodnocení převádí na jiný datový typ.
- Porovnávací operátor
  - Jedná se o prvek schopný generování všech šesti porovnávacích operátorů. MARGIN již tento prvek neřeší, ten byl přeložen již jinak.
- Unární operátor
  - Jedná se o prvek obsahující výraz, který je po překladu do finálního SQL zaobalen jedním ze tří unárních operátorů.
- Konstanta
  - Jedná se o prvek nesoucí konstantu. Zaštiťuje tři různé datové typy konstant.
- Výběr datového zdroje
  - Výběr datového zdroje neboli prostý SELECT. Objevuje jeden, čí více datových zdrojů jako seznam, nad kterýma může být aplikovaná podmínka jakožto výraz. Při finálním překladu se doptává na SQL pro datové zdroje, které požaduje.
- Množina
  - Obsahuje výčet výrazů, které vrací jednu hodnotu, nebo dokonce seznam hodnot. Tento prvek vytvoří jednosloupcový SQL SELECT výraz.
- Doména číselníku
  - Obsahuje jméno domény a jméno číselníku. Při finálním překladu se doptává na zdrojové SQL tohoto číselníku.
- Spojení tabulek
  - Obsahuje dvě tabulky a podmínku, na základě které se mají spojit. Umožňuje použití všech druhů JOIN výrazů, které Oracle a PostgreSQL podporují.
- Výběr sloupce
  - Obsahuje tabulku a název sloupce, který má z ní být vybrán. Výsledné SQL obsahuje SELECT nad touto tabulkou, ovšem vybrán je pouze jeden tvořený sloupec z ní.
- Volání funkce
  - Umožňuje provolání funkcí přímo nad databází. Obsahuje seznam výrazů jako argumenty a jméno provolané funkce. Každý ovladač má možnost funkce ještě transformovat pro potřeby konkrétní databáze.

Kapitola 5

Implementace překladače jazyka kontrol

Tato kapitola pojednává samotnou strukturu finálního překladače, jeho jednotlivé části a samotnou komunikaci částí mezi sebou.

5.1 Struktura překladače a vlastnosti implementace

Jak bylo pospáno v kapitolách zabývajících se analýzou jazyka, překladač musí být rozdělen do několika částí. Vstupem překladače je samozřejmě textová podoba jazyka kontrol, ta ovšem samotná jako vstup nestačí.

Je nutné ještě překladači dodat informace o výkazu, který výrazem jazyka kontrol bude kontrolován, protože bez něj překladači chybí nutné informace, jako například datové typy jednotlivých validovaných buněk.

Překladač samotný je možné rozdělit do dvou částí, které je možné vykonávat samostatně. Jedná se překladač z jazyka kontrol do SQL AST a generátor finálního SQL.

5.1.1 Překladač z jazyka kontrol do SQL AST

Textový vstup je podporován jak ze souboru, tak jako vložený datový typ string. Obě varianty je nutné číst znak po znaku, proto je prvním modulem, do kterého validace vstoupí vstupní proud. Ten má ve výsledné aplikaci dvě různé implementace. Jednu pro textový vstup a druhou pro čtení ze souboru. Aplikace se nemusí omezovat ale jen na tyto vstupní proudy a je snadné doimplementovat později další.

Následujícím modulem je lexer, který provede převod řetězce rozděleného po znacích na tokeny, které obohacuje i o jejich potřebná data. Jeho další funkcí je udržování aktuálního tokenu v pamětí, schopnost na požádání načíst další, popřípadě sdělit, zda už tokeny nedošly, a tudíž nejsme na konci souboru nebo vstupního řetězce.

Parser je modul, který následuje. Jedná se o modul, který z toku tokenů dává dohromady syntaktický strom jazyka kontrol. Tento strom již svojí hierarchií určuje přednost jednotlivých operátorů. Parser je schopný díky přesně zadefinovaným gramatikám odhalit i chyby, které se v kódu nacházejí.

Nyní přichází krok samotného překladu, ale ne rovnou do jazyka SQL. Ten by byl možný, kdyby jazyky měly podobnější syntaxi, bohužel v tomto případě tomu tak není. Dalším důvodem nepřímého překladu je možnost podpory pro vícero databázových serverů, které mají syntaxi dost podobnou, ovšem liší se v detailech. Překlad z tohoto důvodu probíhá do SQL abstraktního

syntaktického stromu. V tomto kroku již dochází na transformaci jednotlivých prvků jazyka kontrol na prvky jazyka SQL. Taktéž zde již dochází k doplnění chybějících informací v jazyce z dodané definice výkazu. Kromě AST SQL je ještě výstupem seznam požadovaných datových zdrojů a seznam požadovaných číselníků, které byly v rámci validace použity.

Z důvodů, že předchozí kroky jsou výkonnostně celkem náročné a předchozí kroky nejsou závislé na samotných datech v reportu, je možné v tomto kroku vytvořit uložitelný formát. K tomu zde přichází na řadu ovladač, který je schopný AST SQL uložit do binární podoby, a z důvodů možného uložení do databáze a vyhnutí se chybám textového kódování je binární podoba posléze převedena na formát BASE64. Z tohoto formátu je možné AST SQL opět získat. V tomto kroku jsou navíc používány binární streamy, které zajišťují zápis po jednotlivých bitech.

5.1.2 Generátor finálního SQL

Z formátu BASE64 je možné získat i AST jednotlivých ovladačů pro jednotlivé databázové servery. Ovladače jsou v aplikaci implementovány dva, a to pro PostgreSQL databázi a pro Oracle databázi. Další ovladače je možné snadno dopsat, stačí udržet jen strukturu, která je v ovladačích navržená.

Cílem ovladačů je transformace na finální SQL dotaz dle požadovaného databázového formátu. V tomto kroku je už nutné překladači dodat datové zdroje a číselníky, o které si požádal v kroku překladu. Z ovladače je následně výstupem textový řetězec, který je sám o sobě spustitelným SQL dotazem.

5.1.3 Výhody zvolené struktury překladače

Tato celková struktura přináší mnoho výhod. Tou nejdůležitější je z pohledu autora práce možnost exportovat SQL pro vícero druhů databázových serverů. Dalšími podstatnými vlastnostmi je možnost uložit AST SQL do řetězce pro možné opakované spuštění bez kompletního překladu od začátku a možnost použití vícero druhů vstupních proudů.

5.1.4 Shrnutí

Celková struktura se skládá dohromady ze šesti úrovní. Ty jsou zakresleny v diagramu 5.1.

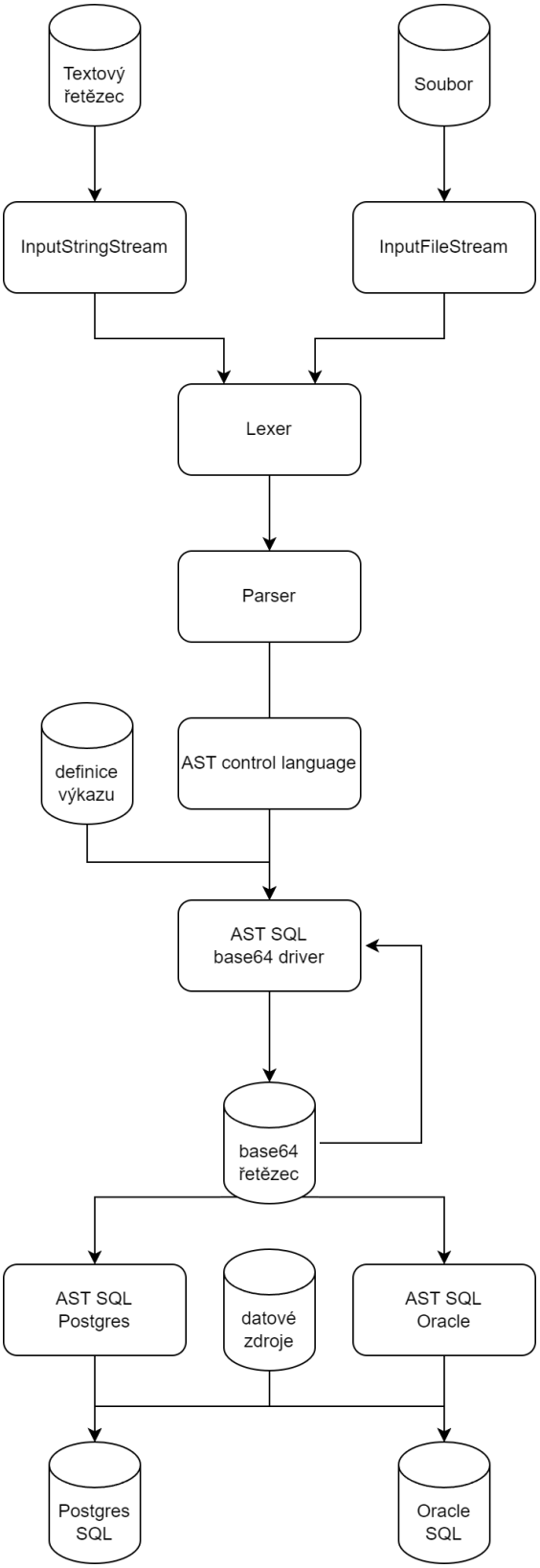
V následujících sekcích jsou tyto moduly detailně popsány včetně použité implementace.

5.2 Vstupní proudy

Implementace vstupních proudů je celkem přímá. Proudy obsahují metody pro získání jednoho znaku (buďto aktuálního, nebo následujícího). Každá z implementací se na základě provolání těchto metod rozhoduje, jak znaky načítat. Proudy zároveň evidují, na kterém řádku a pozici se v souboru/řetězce nachází. Tato vlastnost je důležitá pro ladění, protože lexer/parser jsou na základě této informace schopný vrátit přesnější chybu. Vstupní proudy po nárazu na konec vrací NULL.

5.2.1 Souborový vstupní proud

Tento proud dostává jako stavící parametr cestu k souboru, ze kterého se validace bude načítat. Proud provede otevření souboru a posléze na základě provolání metod soubor čte. Konec souboru je evidován pomocí detekce EOF, na základě které je poslán znak NULL. Proud má na starost i bezpečné uzavření souboru.



Obrázek 5.1 Diagram celkové struktury modulů v překladači jazyka kontrol

5.2.2 Řetězcový vstupní proud

Tento proud dostává jako stavící parametr řetězec, ve kterém je načtená celá validace. Tento řetězec si ukládá index, ze kterého byl načten poslední znak. Na metody říkající o následující znak posunuje index o 1 dopředu. Konec souboru je registrován, pokud index přesáhne délku řetězce.

5.3 Lexikální analyzátor pro jazyk kontrol

Pro lexikální analyzátor je vhodné použít stavový automat, který na základě pohledu a jednoho znaku dopředu bude postupovat až do koncových uzlů, které budou navracet jednotlivé tokeny. V případě, že automat končí v nekoncovém uzlu, bude vyhozena výjimka, chyby čtení tokenu.

Lexikální analyzátor bude obsahovat metodu pro vrácení následujícího tokenu. Vždy následně pomocí automatu provede detekci tokenu ze vstupního řetězce, získaný token ze vstupního řetězce odebere a vrátí nový token v rámci návratové hodnoty. Lexer bude taktéž obsahovat několik pomocných metod, které budou schopny pracovat s posledním uloženým tokenem, token přeskočit, nebo pokud parser ví, že musí následovat zvolený token, umožnit tento token odebrat ze vstupního řetězce.

Vhodnou implementací je v tomto případě mít každý node implementovaný labelem a v něm mít vždy SWITCH, který na základě dalšího znaku provede GOTO nebo již vrátí vyhodnocený token.

Některé tokeny mají několik možných zápisů, ty však je možné spojit v rámci automatu pro jejich čtení.

5.4 Syntaktický analyzátor pro jazyk kontrol

Syntaktický analyzátor má k dispozici lexer, ze kterého postupně dle potřeby získává tokeny. Parser je vhodné implementovat jako zásobníkový automat, který vždy na základě jednoho tokenu vyhodnotí, jakým směrem se vydat. V případě LL(1) analýzy je to možné, je však nutné upravit gramatiku jazyka kontrol tak, aby vždy na základě jednoho tokenu byl schopný identifikovat cestu dál. V dokumentaci jazyka kontrol sice je gramatika uvedena, není ovšem jednoznačná ani vhodná pro LL(1). Upravená gramatika je k dispozici v příloze B.

Výstupem parseru by měl být již AST. Vhodnou implementací zásobníkového automatu je přístup, ve kterém funkce/metoda reprezentuje jedno gramatické pravidlo. V metodě samotné je pak switch, který na základě tokenu vyhodnotí, která varianta pravidla se použije, v těle casu následuje už samotný výčet tokenů/volání dalších pravidel, které dané gramatické pravidlo používá. Výstupem každého pravidla by měl být nějaký z prvků AST. Kořenová metoda syntaktického analyzátoru by měla vrátit již výsledný AST se všemi jeho patřičnostmi.

5.5 Transformace AST jazyka kontrol na SQL AST

Každý z prvků z AST jazyka kontrol obsahuje metodu, která z prvku samotného vytvoří jeho ekvivalent nebo kombinaci ekvivalentů z AST SQL.

Když už existuje AST jazyku kontrol, tak dalším krokem je spuštění zahořovací funkce nad tímto stromem. Kořenový prvek, nad kterým metoda byla provolána, pak následně vrátí v případě úspěchu kořenový prvek SQL AST.

Při tomto kroku dochází i k narovnávání datových typů či dohledávání ukazatelů a parametru v definici reportu. V tomto kroku jsou použity veškeré analýzy provedené v sekcích 4.3 a 4.4.

5.6 Detekce chyb

Parser je primární zdroj detekce chyb v překládaném kódu. Díky znalosti gramatik je schopný identifikovat, že z lexeru přišel token, se kterým nepočítal. V tomto případě vyhazuje vhodnou výjimku a přerušuje překlad.

Transformace AST jazyka kontrol na SQL AST je poslední krok, který je schopen detekce chyb. Chyby detekované v tomto kroku jsou primárně typu, že je použit prvek, který je kompilátoru neznámý. Příkladem může být neznámé jméno funkce. Taktéž tato úroveň detekuje chyby druhu nesprávného datového typu použitého v operaci (např. když do operátoru IN na pravé straně přijde textový řetězec).

V případě nalezení jedné z těchto chyb je vyhozena patřičná výjimka a překlad je ukončen.

5.7 Ovladač pro transformaci SQL AST do BASE64 řetězce

Vytváření SQL AST v přechozím kroku rovnou vytváří instance tříd ovladače pro transformaci SQL AST do base64 řetězce. Nad kořenem stromu tohoto ovladače je stejným způsobem jako v předchozím kroku provolána metoda pro vygenerování finálního řetězce, jen s rozdílem, že parametrem této metody je výstupní bitový proud.

Každý z prvků stromu tomuto prvku sdělí postupně sekvenci bitů, které mají být zapsány do finální binární podoby. Po zápasu je sekvence bitů zarovnána nulami tak, aby počet bytů v sekvenci by dělitelný osmi a celá sekvence je posléze převedena na řetězec do formátu BASE64.

5.8 Načtení z BASE64 řetězce do ovladače příslušné databáze

Jedná se o opačný krok, než je ten předchozí, jen s rozdílem, že je možné si vybrat, do jakého ovladačového stromu bude řetězec převeden. Díky tomu, že sekvence bitů vždy obsahuje na svém začátku typ prvku, který je zapsán na konkrétním místě, může být nad zvoleným typem prvku provolána metoda pro jeho načtení ze sekvence. Metoda již konkrétního prvku ví, co obsahuje za prvky, a je si je schopná načíst. Pokud prvek obsahuje jiný prvek, opět bude na jeho začátku zadán binárně jeho datový typ a celý proces se opakuje.

5.9 Konečná trnsformace z AST ovladače na výsledný SQL dotaz

Stejným způsobem, jako dochází pomocí ovladače SQL AST překladu stromu do řetězce BASE64, tak stejným způsobem probíhá sestrojení finálního SQL. V tomto případě již není používaný bitový proud, ale jednotlivé části jsou posílány návratovou hodnotou. Argumentem do kořenové hodnoty vstupuje pomocná třída, která obsahuje všechny potřebné datové zdroje validace ve formátu SQL a stejně tak i číselníky.

5.10 Použité prostředky pro implementaci

Překladač pro jazyk kontrol byl implementován za pomocí programovacího jazyka PHP verze 8.2. Díky tomu tento překladač může běžet v rámci webové aplikace bez nutnosti dalšího rozvoje.

Obliba jazyka PHP zajišťuje rozsáhlou skupinu vývojářů, kteří tento jazyk znají, což bude prospěšné pro možnost budoucího rozvoje. Syntaxe jazyka PHP je navíc jednoduchá a čistá, což usnadňuje vývoj robustního a efektivního kompilátoru.

Další významnou výhodou použití jazyka PHP pro překladač jazyka kontrol je jeho kompatibilita napříč platformami. Aplikace PHP mohou bezproblémově běžet na různých operačních systémech a dokonce i běžných, cenově dostupných webových hostech, čímž odpadají obavy související s problémy specifickými pro danou platformu. Tato multiplatformní schopnost zvyšuje dostupnost a použitelnost překladače a umožňuje jeho nasazení v různých prostředích bez snížení výkonu.

Otevřený kód jazyka PHP přispívá k transparentnosti a podporuje spolupráci v rámci vývojářské komunity. Díky běhu nad čistým PHP a tudíž nezávyslosti na žádné externí knihovně nehrozí odstavení překladače do nefunkčního stavu z důvodů zastaralých závislostí.

Součástí implementace je přiložená i demo aplikace pro ukázku možného použití. Informace o ní jsou dostupné v jejím vlasním readme souboru.

Kapitola 6

Testování vzniklých SQL dotazů

Tato kapitola pojednává o možném způsobu testování, vzniku prostředí pro jeho umožnění a samotných výsledcích testů.

6.1 Úvod

Pro testování a zároveň odladění překladače je možné použít hned několik metod, ovšem jako nejúčinnější se jeví test nad reálnými daty vůči systému SDAT.

Systém SDAT má dvě prostředí, a to testovací a produkční. Zatímco produkční prostředí slouží pro závazné posílání výkazu přímo ČNB, na testovací prostředí je možné nezávazně odeslat data a získat report o chybách v těchto datech.

Co je v tomto testu překážkou je, že pro přístup k testovacímu prostředí, samotným definicím výkazů a jednotlivým validacím je potřeba mít přístupové údaje. Pro jejich zisk je nutné být registrován u ČNB jako subjekt povinný vykazovat nebo jako subjekt vykazující za někoho jiného. Autor této práce není ani jedním, proto se spojil se společností evosoft s. r. o..

6.2 Společnost evosoft a její software použitelný pro testování

Společnost evosoft je česká firma založená v roce 2015. Mimo tvorby softwarů na míru je jejím primárním zaměřením právě software pro výkaznictví do ČNB jménem galaxii. Díky tomu společnost evosoft má k dispozici nejen přístupy k datům ze systému SDAT, ale taktéž i testovacímu prostředí tohoto systému.[8]

Pro testování této aplikace je taktéž vhodné použít systém universii od společnosti evosoft, který automaticky stahuje a zpracovává definice výkazů ze systému SDAT a vytváří datové balíčky pro systém galaxii. Galaxii pomocí těchto balíčku ví, jakým způsobem mají být výkazy odeslané do ČNB strukturovány.

Obě tyto aplikace jsou používány na produkční úrovni, a jsou i proto vhodné jako zdroje dat pro tuto práci. Ze softwaru universii budou získány formáty výkazů pro validace, ze softwaru galaxii samotné datové zdroje pro validace.

Za propůjčení těchto systémů společnosti evosoft děkuji.

6.3 Metodika testování

První fáze je pokus o překlad všech JVK umístěných ve vybraných metodikách definovaných systémem SDAT. Seznam těchto validací a struktura výkazů bude poskytnuta systémem universii, který má tato data předzpracována.

V této fázi se bude hodnotit úspěšnost vygenerovaní validací. Optimální scénář je, že všechny validace jsou přeloženy.

Druhá fáze je vytvoření samotných SQL nad konkrétními daty pro vybrané metodiky. K zisku datových zdrojů bude použit systém galaxii, který je data schopný zformulovat do požadované struktury. Systém galaxii ovšem ve výchozím nastavení neobsahuje žádná data a klientská data není možné použít. Naštěstí si společnost evosoft vyvinula soubor testovacích dat, nad kterými jsou softwary universii a galaxii testovány právě vůči systému SDAT. Za propůjčení těchto dat taktéž děkuji společnosti evosoft.

Testovací data budou nejprve odeslána do systému SDAT, a tím bude ověřeno, jak hodně jsou v pořádku. Následně budou data nahrána do překladače jako datové zdroje a validace budou spuštěny na lokálních SQL serverech (PostgreSQL a Oracle). Poté bude možné porovnat množství neshod mezi výstupy. Nad výkazem, který ČNB označila jako naprosto v pořádku, je očekáváno, že lokální spuštění nenajde žádné chyby a naopak.

6.4 Výsledky fáze 1

Metodika	Počet validací	Úspěch	Deaktivováno	Selhání
MKT20240101	705	501	31	172 + 1
FOFI20230101	603	563	40	0
KT20231201	395	372	6	17
PLT20230101	147	133	0	14

Tabulka 6.1 Seznam metodik použitých v první části testování a jejich výsledky

Tabulka 6.1 reprezentuje seznam testovaných metodik a statistiku úspěšnosti. Sloupec počet validací udává celkový počet validací, které jsou k dispozici ve vybrané metodice. Následuje sloupec úspěch, který indikuje počet validací, které překladač úspěšně rozpoznal, a vytvořil výsledný strom. Předposlední sloupec deaktivováno sděluje, kolik validací je podle systému SDAT nefunkční, proto se neprováděl ani pokus o jejich překlad. V posledním sloupci selhání je uveden počet validací, které se nepodařilo z nějakého důvodu přeložit.

Z tabulky je patrné, že selhalo 204 validací. Z procházení výstupů překladače vyplývá, že u 202 validací se nepovedl překlad z důvodů nepodporované funkce. Všechny tyto byli z této práce ovšem vyjaty, jak je pospáno v sekci 4.4.10.

Poslední zbývající aplikace vyhodila chybu na úrovni sémantického tvaru. Chyba spočívala v použití dynamického parametru uvnitř výrazu WITH, který tento zápis podle dokumentace nepodporuje. Po výměně e-mailů byla validace upravena na straně systému SDAT, bohužel však k opětovnému testu již nedošlo.

6.5 Výsledky fáze 2

Tabulka 6.2 reprezentuje seznam datových množin, které byly testovány vůči systému SDAT a zároveň vůči SQL dotazům vygenerovaných překladačem. Sloupec počet validací označuje celkový počet validací, které se nad reportem měly spustit. Sloupce SDAT, Oracle a PostgreSQL určují, kolik z těchto validací skončilo chybou dat. Závěrem sloupec selhání říká, kolik validací se nepodařilo spustit vůbec nad některým z databázových serverů.

Report	Dataset	Počet validací	SDAT	Oracle	Postgres	selhání
KT - JISIFE51	OK	32	0	0	0	0
KT - JISIFE51	FAIL	32	32	31	31	0
KT - JISIFE51	SEMI	32	14	14	14	0
KT - JISIFE52	OK	16	0	0	0	0
KT - JISIFE52	FAIL	16	16	16	16	0
KT - JISIFE52	SEMI	16	4	4	4	0
MKT - TRAFIM10	OK	96	0	0	0	0
MKT - TRAFIM10	FAIL	96	96	88	88	0
MKT - TRAFIM10	SEMI	96	52	51	51	0
MKT - TRAFIM11	OK	8	0	0	0	0
MKT - TRAFIM11	FAIL	8	8	6	6	0
MKT - TRAFIM11	SEMI	8	5	5	5	0

Tabulka 6.2 Seznam datových zdrojů použitý pro testování ve fázi 2 a jejich výsledky

Ani jedna z validací neselhala při spuštění, a o co lépe, v případech validních reportů nedošlo k chybnému vyhodnocení, že data jsou špatně. V případě nevalidních reportů dochází u jistých dat k drobným odchylkám. Po analýze rozdílu je však zřejmé, že rozdíly vznikají kvůli validacím, které se ani nepodařilo přeložit. Ty bohužel na straně systému SDAT vypnout nelze.

Kapitola 7
Závěr

Cílem této bakalářské práce bylo vytvořit překladač jednovýkazových validací z jazyka kontrol vyvinutého Českou Národní Bankou do jazyka SQL určeného pro databáze.

Pro tento úkol byla provedena analýza struktury jazyka kontrol. Na jejím základě vznikl seznam tokenů potřebný pro dekompozici a LL(1) gramatika určená pro čtení toku tokenů. Po vytvoření gramatiky byla provedena analýza jednotlivých prvků syntaxe jazyka kontrol a jejich možnosti překladu do jazyka SQL.

Po této analýze byla navržená hrubá struktura modulů pro implementaci. Tyto moduly byly posléze samostaně analyzovány a byly navrženy jejich implementace.

Na základě tohoto návrhu byl překladač implementován v jazyce PHP bez použití externích knihoven nebo rozšíření. Aplikace je psána objektově orientovaně a modulovatelně, s možnostmi rozšíření o nové ovladače nebo nahrazení.

Finální podoba aplikace byla vyzkoušená nad smyšlenými daty od společnosti evosoft vůči samotnému systému SDAT od ČNB, který validace provádí. Výsledky těchto testů byly lepší než očekávání autora této práce.

Hlavním nedostatkem práce je vynechání několika funkcí, které jazyk kontrol podporuje. Primárním důvodem je nutnost upřesnění jejich specifikace, která je z pohledu autora v dokumentaci nepřesná. Již probíhá e-mailová komunikace s Českou národní bankou, díky které tyto funkce bude možné v budoucnosti aplikace doimplementovat. Velké možné rozšíření je možnost překladu i mezivýkazových validací, nebo možnosti provedení i formátových validací. Zajímavá by mohla být i možnost získávání validací či samotných výkazů rovnou z SDAT API.

Cíle práce byly splněny v plném rozsahu.

Příloha A

Tokeny pro lexikální analýzu sémantického tvaru jazyka kontrol

- Základní tokeny
  - PLUS – ’+’
  - MINUS – ’-’
  - MULTIPLE – ’*’
  - DIVIDE – ’/’
  - LESS THAN – ’<’
  - LESS EQUAL – ’<=’
  - EQUAL – ’=’
  - GREATER THAN – ’>’
  - GREATER EQUAL – ’>=’
  - NOT EQUAL – ’<>’, ’!=’
  - B OPEN – ’(’
  - B CLOSE – ’)’
  - C OPEN – ’{’
  - C CLOSE – ’}’
  - NEGATION – ’∧’
  - ASSIGN – ’:=’
  - Q OPEN – ’[’
  - Q CLOSE – ’]’
  - QQ OPEN – ’[[’
  - QQ CLOSE – ’]]’
  - COMMA – ’,’
  - DD – ’:’
  - ARROW – ’->’
  - PERCENT – ’%’
  - SEMI – ’;’

- Klíčová slova
  - MARGIN
  - IN
  - FOR
  - NULL
  - TRUE
  - FALSE
  - WITH
  - FILTER CHECK
- Textové tokeny
  - STRING
  - VARIABLE
  - IDENTIFIER
  - CODE
  - DOMAIN
  - PRAGMA
- Numerické tokeny
  - INT
  - FLOAT
- Speciální
  - EOF

Příloha B

LL(1) gramatika sémantického tvaru jazyka kontrol

Dále je vypsána gramatika sémantického tvaru jazyka kontrol. V kulatých závorkách jsou terminální tokeny, v hranatých neterminální.

$[START] -> [PB][VAR][MAIN]$

$[PB] -> (PRAGMA)[PB']$
$[PB] -> \mathcal{E}$

$[PB'] -> (COMMA)(PRAGMA)[PB']$
$[PB'] -> \mathcal{E}$

$[VAR] -> (VARIABLE)[VAR']$
$[VAR] -> \mathcal{E}$

$[VAR'] -> (ASSIGN)[E][VAR'']$

$[VAR''] -> [VAR]$
$[VAR''] -> (SEMI)[VAR]$

$[MAIN] -> (WITH\_KW)(B\_OPEN)[PARLIST](B\_CLOSE)[ROW]$
$[MAIN] -> [ROW]$

$[PARLIST] -> (IDENTIFIER)[PAR][PARVAL][PARLIST']$
$[PARLIST] -> \mathcal{E}$

$[PARLIST'] -> (COMMA)(IDENTIFIER)[PAR][PARVAL][PARLIST']$
$[PARLIST'] -> \mathcal{E}$

$[PAR] -> (DD)$
$[PAR] -> (EQUAL)$

$[PARVAL] -> (IDENTIFIER)$
$[PARVAL] -> (CODE)$
$[PARVAL] -> (INTEGER)$
$[PARVAL] -> (FLOAT)$
$[PARVAL] -> (STRING)$
$[PARVAL] -> (DOMAIN)$
$[PARVAL] -> (MULTIPLE)$
$[PARVAL] -> (NULL\_KW)$
$[PARVAL] -> [SET]$

$[ROW] -> (FOR\_KW)(B\_OPEN)[PARLIST](B\_CLOSE)[E][ROW']$
$[ROW] -> [E][ROW']$
$[ROW] -> (FILTER\_CHECK\_KW)$
$(B\_OPEN)(IDENTIFIER)[FILTER](B\_CLOSE)[E][ROW']$

$[ROW'] -> (SEMI)$
$[ROW'] -> \mathcal{E}$

$[FILTER] -> (COMMA)[E][FILTER]$
$[FILTER] -> \mathcal{E}$

$[E] -> [E1][E']$

$[E'] -> (IN\_KW)[E1][E']$
$[E'] -> \mathcal{E}$

$[E1] -> [E2][E1']$

$[E1'] -> (LESS\_THAN)[E2][MA][E1']$
$[E1'] -> (LESS\_EQUAL)[E2][MA][E1']$
$[E1'] -> (GREATER\_THAN)[E2][MA][E1']$
$[E1'] -> (GREATER\_EQUAL)[E2][MA][E1']$
$[E1'] -> (EQUAL)[E2][MA][E1']$
$[E1'] -> (NOT\_EQUAL)[E2][MA][E1']$
$[E1'] -> \mathcal{E}$

$[E2] -> [E3][E2']$

$[E2'] -> (PLUS)[E3][E2']$
$[E2'] -> (MINUS)[E3][E2']$
$[E2'] -> \mathcal{E}$

$[E3] -> [E4][E3']$

$[E3'] -> (MULTIPLE)[E4][E3']$
$[E3'] -> (DIVIDE)[E4][E3']$
$[E3'] -> \mathcal{E}$

$[E4] -> (MINUS)[E5]$
$[E4] -> [E5]$

$[E5] -> (B\_OPEN)[E](B\_CLOSE)$
$[E5] -> (IDENTIFIER)[KEY]$
$[E5] -> (VARIABLE)$
$[E5] -> (STRING)$
$[E5] -> (DOMAIN)$
$[E5] -> (NULL\_KW)$
$[E5] -> (TRUE\_KW)$
$[E5] -> (FALSE\_KW)$
$[E5] -> (FLOAT)$
$[E5] -> (INTEGER)$
$[E5] -> [SET]$
$[E5] -> (Q\_OPEN)(IDENTIFIER)[PARLIST'](Q\_CLOSE)$
$[E5] -> (QQ\_OPEN)(IDENTIFIER)(QQ\_CLOSE)$

$[MA] -> (MARGIN\_KW)[MA']$
$[MA] -> \mathcal{E}$

$[MA'] -> (INTEGER)$
$[MA'] -> (FLOAT)$

$[KEY] -> (B\_OPEN)[FARG](B\_CLOSE)$
$[KEY] -> (Q\_OPEN)(IDENTIFIER)[PARLIST'](Q\_CLOSE)$
$[KEY] -> (QQ\_OPEN)(IDENTIFIER)(QQ\_CLOSE)$
$[KEY] -> (ARROW)[ARROW]$
$[KEY] -> (DD)[PARVAL]$
$[KEY] -> (EQUAL)[PARVAL]$
$[KEY] -> (PERCENT)[KEY']$
$[KEY] -> \mathcal{E}$

$[KEY'] -> (MINUS)(INTEGER)[KEY'']$
$[KEY'] -> (PLUS)(INTEGER)[KEY'']$
$[KEY'] -> (INTEGER)[KEY'']$

$[KEY''] -> (Q\_OPEN)(IDENTIFIER)[PARLIST'](Q\_CLOSE)$

$[FARG] -> [E][FARG']$
$[FARG] -> \mathcal{E}$

$[FARG'] -> (COMMA)[E][FARG']$
$[FARG'] -> \mathcal{E}$

$[ARROW] -> (IDENTIFIER)$
$[ARROW] -> (CODE)$
$[ARROW] -> (INTEGER)$

$[SET] -> (C\_OPEN)[SET'']$

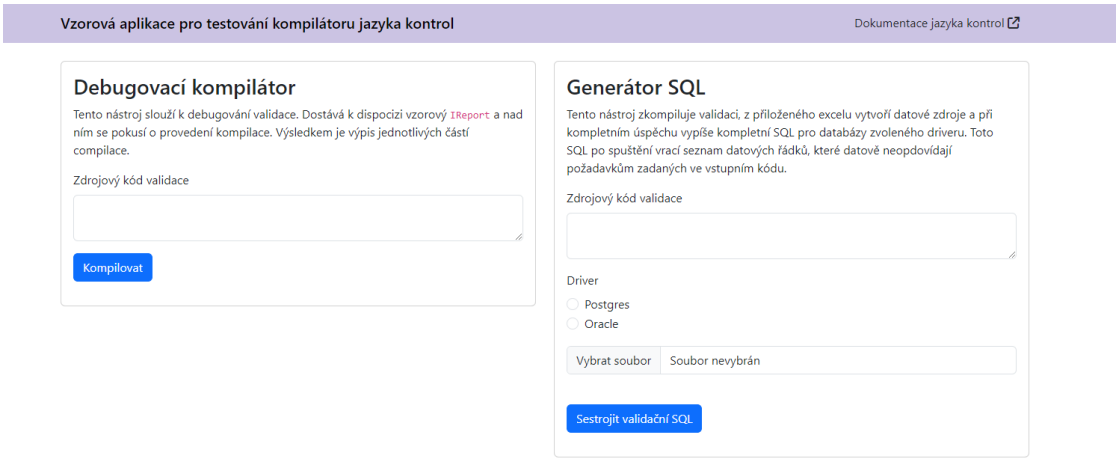
$[SET''] -> [NEGE][SET']$
$[SET''] -> (C\_CLOSE)$

$[SET'] -> (COMMA)[NEGE][SET']$
$[SET'] -> (C\_CLOSE)$

$[NEGE] -> (NEGATION)[E]$
$[NEGE] -> [E]$

Příloha C

Vzhled vzorové aplikace dodané ke kompilátoru



Obrázek C.1 Vzhled vzorové aplikace dodané ke kompilátoru

Bibliografie

1. ČESKÁ NÁRODNÍ BANKA. *Technická specifikace SDAT - Popis systému*. 2021. Č. 1. Dostupné také z: `https://www.cnb.cz/export/sites/cnb/cs/statistika/.galleries/sdat/SDAT_TS_1_PopisSystemu.docx`. Informace o struktuře reporů a výkaznictví samotném.
2. ČESKÁ NÁRODNÍ BANKA. *Technická specifikace SDAT - Popis jazyka kontrol*. 2021. Č. 8. Dostupné také z: `https://www.cnb.cz/export/sites/cnb/cs/statistika/.galleries/sdat/SDAT_TS_8_PopisKontrol.docx`.
3. ČEŠKA, Milan; MELICHAR, Jan; JEŽEK, Karel; RICHTA, Karel. *Konstrukce překladačů*. Praha: Vydavatelství ČVUT, 2006. ISBN 80-01-02028-2.
4. AHO; LAM; SETHI; ULLMAN. *Compiler: Principles, Techniques and Tools*. 2nd. Addison-Wesley, 2010. ISBN 978-0321486813.
5. ORACLE CORPORATION. *Oracle Database Documentation* [online]. [cit. 2023-05-15]. Dostupné z: `https://docs.oracle.com/en/database`.
6. ORACLE CORPORATION. *Oracle SQL Language Reference* [online]. [cit. 2023-05-15]. Dostupné z: `https://docs.oracle.com/en/database/oracle/oracle-database/23/sqlrf/index.html#Oracle%C2%AE-Database`.
7. POSTGRESQL GLOBAL DEVELOPMENT GROUP. *PostgreSQL Documentation* [online]. [cit. 2023-05-18]. Dostupné z: `https://www.postgresql.org/docs/`.
8. EVOSOFT S. R. O. *Webová prezentace společnosti evosoft* [online]. [cit. 2024-01-05]. Dostupné z: `https://evosoft.cz/`.