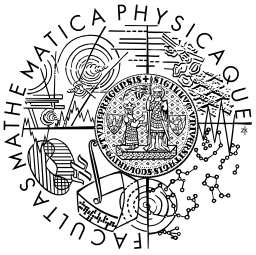


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Michal Bartoš

Prezentace transformací atributů v SQL pro potřeby data governance

Katedra softwarového inženýrství

Vedoucí diplomové práce:	doc. Ing. Jan Janoušek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Softwarové systémy

Praha 2018

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů (dále jen „autorský zákon“), především § 35 a § 60 autorského zákona upravující školní dílo. V případě počítačových programů, jež jsou součástí mojí práce či její přílohou, a veškeré související dokumentace k počítačovým programům (dále jen „software“), uděluji tzv. MIT Licenci.

Obsahem MIT Licence je oprávnění bezúplatně užít software. Toto oprávnění uděluji každé osobě, která má o užití software zájem.

Každá osoba je oprávněna pořídit si kopii software (včetně související dokumentace) bez jakéhokoli omezení a dále je oprávněna bez jakéhokoli omezení software zejména užívat, kopírovat, upravovat, sloučit, publikovat, distribuovat, poskytovat podlicence a / nebo prodávat kopie software a umožnit výkon těchto práv osobám, kterým bude dále software poskytnut. Způsoby užití software ani rozsahu tohoto užití nejsou jakkoli omezeny.

Osoba, která má o užití software zájem je povinna připojit text licenčních podmínek následujícího znění:

Copyright (c) 2018 Michal Bartoš
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:
The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.
THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

V ..... dne ..... Podpis autora

Tímto bych rád poděkoval následujícím lidem, kteří mi pomáhali s realizací této práce ať už konzultací, odbornou radou nebo trpělivostí a podporou. Především jim děkuji za umožnění tvorby prototypové implementace této práce do nástroje Manta a odbornou pomoc se zorientováním se v jeho struktuře.

Děkuji tímto následujícím lidem:

- doc. Ing. Janu Janouškovi Ph.D., vedoucímu diplomové práce
- Mgr. Jiřímu Touškovi, za technickou pomoc s Mantou
- RNDr. Lukáši Hermannovi, za technickou pomoc s Mantou
- ostatním mým blízkým za trpělivost a podporu

Název práce: Prezentace transformací atributů v SQL pro potřeby data governance

Autor: Michal Bartoš

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: doc. Ing. Jan Janoušek, Ph.D., Katedra softwarového inženýrství

Abstrakt: Práce se zabývá extrakcí popisů transformací jednotlivých atributů z Oracle SQL skriptů a způsobem jejich prezentace uživatelům. Porovnává vhodnost několika variant výstupního formátu a pro vybraný formát následně rozebírá detaily jeho tvorby a rozhodnutí, která k němu vedla. Jedním z významných rozhodnutí byla volba vstupních datových struktur: abstraktního syntaktického stromu a grafu datových toků. Tato volba poskytuje iniciální analýzu vstupních SQL skriptů, ale má také výrazný vliv na další průběh zpracování. Výstupy této práce byly ověřeny pomocí prototypové implementace v nástroji Manta, na které je prezentována vhodnost zvoleného řešení pro přehlednou prezentaci uživatelům.

Klíčová slova: SQL Oracle analýza toku dat

Title: The Presentation of Transforming Attributes in SQL for Data Governance Support

Author: Michal Bartoš

Department: Department of Software Engineering

Supervisor: doc. Ing. Jan Janoušek, Ph.D., Department of Software Engineering

Abstract: This thesis examines extraction of attributes' transformation descriptions from Oracle SQL scripts and possibilities of their presentation to users. This thesis compares several variants of output format. Then it describes details of construction for the chosen format and choices that led to it. One of the main decisions was the choice of input data structures: abstract syntax tree and dataflow graph. Those data structures provide initial analysis of input SQL scripts but they also strongly influence the rest of processing. Results of this thesis were verified by prototype implementation in software Manta. The prototype confirmed suitability of the chosen approach for convenient presentation to users.

Keywords: SQL Oracle data flow analysis

Obsah

1. Úvod

Cílem této práce bylo vytvořit modul do software Manta Flow (dále již pouze Manta), který by jej rozšířil o možnost vyhledávat a zjednodušeně zobrazit transformace konkrétních atributů v SQL/PLSQL skriptech pro Oracle dialekt (dále pouze SQL). Dále v textu této diplomové práce bude tento modul, který vznikl v rámci implementační (prototypové) fáze nazýván „Expressions modul".

Výsledek tohoto modulu je integrován do stávajících výstupů Manty a umožní analytikům podrobnější pohled na vznik jednotlivých atributů ukládaných a tekoucích v Oracle databázi. Slouží jako alternativa k zobrazení celého SQL skriptu, který je při větších transformacích již příliš komplikovaný a nepřehledný. Takto je možno si na jednotlivých datových transformacích zobrazit operace ovlivňující pouze zkoumaný atribut.

Hlavním přínosem by mělo být umožnit snazší analyzování, jak vznikají některé atributy i pro neprogramátory nebo lidi nezběhlé v SQL. Pro složité transformace není možné se od zdrojového SQL plně oprostit a zůstává nutnost konzultace s osobou znalou SQL. Ale pro snazší transformace by mělo být možné i pro tyto lidi zjistit, jak příslušné atributy vznikají. Toto se týká nezanedbatelné části dat obsažených v databázi, které nevznikají složitými transformacemi, ale pouze neintuitivními nebo místy méně přehlednými způsoby.

Výstup však může být přínosný i pro programátory a lidi zběhlé v SQL, protože jim poskytne přehlednější pohled na vznik konkrétního atributu očištěný o ostatní zdrojový kód SQL skriptu. Toto se hodí především u rozsáhlých SQL skriptů, kde dochází ke snížení čitelnosti v důsledku rozsáhlosti. Dalším přínosem je, že transformace jsou prezentovány v modulem definovaného formátu, což může být přínosné především při přebírání cizího zdrojového kódu s nevhodným a špatně čitelným formátováním.

1.1 Struktura práce

Práce je členěna do čtyř logických celků rozložených do devíti kapitol.

V kapitole číslo 2 je obecně představen software Manta, pro poskytnutí kontextu, ve kterém byla tato práce řešena. Další rozšíření přináší kapitola číslo 3, ve které jsou již podrobněji rozebírány jednotlivé funkčnosti, které poskytuje prostředí Manta pro potřeby této práce.

Kapitoly číslo 4 a 5 se zabývají formátem generovaného výstupu. V kapitole číslo 4 jsou rozebírány podmínky a několik zvažovaných formátů spolu s důvodem, proč nebyly výsledně použity. Zatím co analýza zvoleného formátu spolu s informacemi o něm byly, z důvodu důležitosti pro tuto práci, odděleny do samostatné kapitoly číslo 5.

V kapitole číslo 6 je popsán základní princip fungování a informace o implementaci Expressions modulu. Tím představuje úvod pro následující dvě kapitoly číslo 7 a 8, které podrobněji rozebírají obě fáze zpracování. První fáze je popsána v kapitole číslo 7 a zabývá se především předzpracováním dat z jednotlivých konstruktů Oracle dialektu SQL. Druhá fáze je popsána v kapitole číslo 8 a řeší především tvorbu výsledného popisu transformace pro jednotlivé atributy definovaného v kapitole číslo 5.

Kapitola číslo 9 již popisuje jak byla prototypová implementace testována a využití jednotlivých typů testovacích scénářů.

Práce je zakončena závěrem, který shrnuje celou práci a nabízí možnosti jejího dalšího rozšíření.

2. Manta Flow

2.1 K čemu slouží

Manta (viz Literatura [3]) je software sloužící ke zpracování zdrojového kódu SQL procedur a Java kódu analyzovaného systému, ze kterých generuje globální popis datových transformací a toků v něm obsažených, nebo-li Data Lineage (viz Literatura [2]).

Výhodou takto generovaného popisu oproti ručně psané dokumentaci je, že je možné jej opakovaně vygenerovat v relativně krátké době (v závislosti na složitosti systému až do několika hodin). Tím je zaručena jeho aktuálnost, zatímco údržba ručně psané dokumentace je nákladnou součástí vývoje. Tohoto se v praxi využívá pro generování dokumentace vyžadované po některých systémech pro potřeby regulačních úřadů.

Tím, že tento generovaný popis nabízí globální pohled napříč celým systémem, umožňuje provádět dopadové analýzy při jednotlivých dílčích změnách a optimalizaci datových skladů.

Výsledný popis je možné použít samostatně prostřednictvím grafického rozhraní nebo jako vstup pro další zpracování jinými nástroji typu Metadata management / Data Governance. Příkladem těchto navazujících nástrojů mohou být: Informatica Metadata Management, IBM InfoSphere Information Governance Catalog nebo Collibra Data Governance Center.

Hlavní využití nachází především v datových skladech a pro použití v RISK odděleních bank.

2.2 Historie

Manta původně vznikla v roce 2008 jako interní nástroj konzultační společnosti Profinit, s.r.o. za účelem řešení konkrétního problému. Díky univerzálnímu způsobu napsání bylo nožné jej využívat i k usnadnění řešení dalších obdobných problémů a docházelo k rozvoji a rozšiřování funkcionalit. V roce 2013 se vývoj Manty oddělil od společnosti Profinit, s.r.o. jako samostatná společnost Manta Tools s.r.o., známá jako MANTA.

Do zahraničí se Manta dostala především díky vítězství v soutěži „Czech ICT Incubator @ Silicon Valley“, pořádanou Czech ICT Alliance v roce 2014, které dopomohlo k založení první americké pobočky v San Francisku. Aktuálně má Manta, prostřednictvím vlastních poboček a sítě regionálních partnerů zákazníky z celého světa. Mezi její zákazníky patří například společnosti Paypal, OBI, Vodafone nebo Comcast.

2.3 Způsob použití

Manta je využívána jako samostatně běžící aplikace, která dostane přístup ke zdrojovému kódu analyzovaného systému nebo se pomocí databázových konektorů připojí k jeho běžící databázi, ze které načítá potřebné informace (jednotlivá databázová schémata a skripty).

Pro každý systém je připravena specifická konfigurace podle jeho konkrétních požadavků, zahrnující nejen jednotlivé parametry, ale i seznam modulů, které mají být využity. Každý modul Manty je zodpovědný za určitou část zpracování a přidává další funkčnosti.

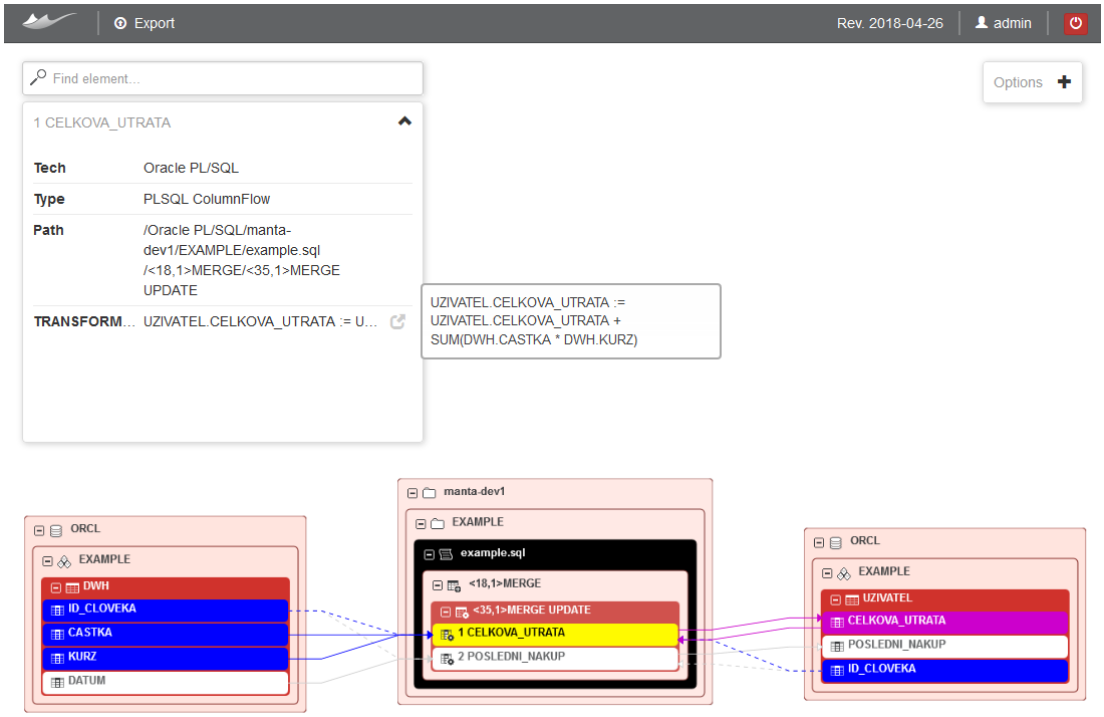
Všechny vstupy jsou zpracovány a je z nich vygenerován globální graf datových toků, popisující celou zkoumanou databázi. Tento graf je z důvodu opakovaného použití výsledku zpracování a náročnosti jeho přípravy uložen do grafové databáze. Odkud je dále zkoumán ve vizualizačním nástroji Manty nebo slouží jako vstupní data pro další navazující nástroje.

2.4 Výstup

Pro ukázku jak vypadá zobrazení výsledku analýzy Mantou, byl připraven menší SQL skript, na kterém je předvedena jeho vizualizace. Stejným způsobem jsou vizualizovány i podstatně rozsáhlejší vstupy, pro které je Manta určena. Pro její představení a udržení zde prezentovatelného rozsahu, byl ovšem příklad omezen pouze na jeden SQL příkaz.

Vstupní Oracle SQL skript zpracovaný Mantou pro přípravu následujícího příkladu grafické vizualizace:

```
MERGE INTO uzivatel u
USING (
  SELECT
    id_cloveka,
    SUM(castka_czk) AS utrata,
    MAX(datum) AS posledni_datum
  FROM (
    SELECT
      id_cloveka,
      id_nakupu,
      castka * kurz AS castka_czk,
      datum
    FROM dwh
  )
  GROUP BY id_cloveka
) p
ON (u.id_cloveka = p.id_cloveka)
WHEN MATCHED THEN UPDATE SET
  u.celkova_utrata = u.celkova_utrata + p.utrata,
  u.posledni_nakup = posledni_datum
;
```



Obrázek 2.1: Příklad zobrazení výstupů Manty v jejím vizualizačním nástroji.

V dolní části vizualizace je vidět vstupní tabulka (`DWH`) s jejími sloupci, ze které jsou šipkami znázorněny datové toky pro jednotlivé atributy do transformace a následně do výsledné tabulky (`UZIVATEL`). V levé horní části je zobrazen detail vybraného atributu `CELKOVA_CASTKA` z transformace. Je zde vidět i nový atribut `TRANSFORMATION_DESC`, který reprezentuje popis transformace tohoto atributu, který vznikl jako výstup prototypové implementace této práce.

3. Funkčnosti poskytované Mantou

Expressions modul byl implementován jako jeden z modulů v řetězci zpracování vstupů v rámci Manty. Z toho vyplývá pro modul široká řada požadavků a omezení, ale také poskytovaných funkčností.

Mezi hlavní funkčnosti, které poskytuje prostředí Manta, patří:

- syntaktická analýza SQL skriptů v Oracle dialektu do abstraktního syntaktického stromu
- sestavení cílových prezentovaných grafů datových toků
- datové struktury obecně reprezentující jednotlivé logické prvky z SQL
- verzovací systém pro vývoj (v tomto případě SVN)
- vývojové prostředí, zahrnující automatické sestavení aplikace a spouštění testů
- způsob načtení jednotlivých SQL skriptů ze zkoumaného systému
- způsob uložení výsledku běhu aplikace pro další zobrazení nebo zpracování

3.1 Abstraktní syntaktický strom

Tento strom je hlavním zdrojem dat pro tvořený Expressions modul. Struktura stromu vychází z definice AST (Abstract syntax tree, viz Literatura [3]), ale je upraven pro potřeby Manty. Dále v textu bude tento strom označován jako AST.

AST tvořený pro potřeby Manty pro Oracle dialekt SQL se ukázal jako vhodným zdrojem dat i pro Expressions modul. Obsahuje totiž veškerá data potřebná pro extrakci transformací z SQL skriptů. Informace jsou již uloženy v AST v Java objektech reprezentující jednotlivé typy operací z SQL, spolu s dalšími doplňujícími meta daty.

I pro jednoduchý SQL skript jako je například:

```
SELECT a * b AS c
FROM tbl
;
```

je generovaný AST příliš rozsáhlý pro jeho kompletní výpis v rámci této práce. Z tohoto důvodu je vždy prezentována pouze pro nás podstatná část a zbytek je nahrazen „. . . “, jako v tomto případě:

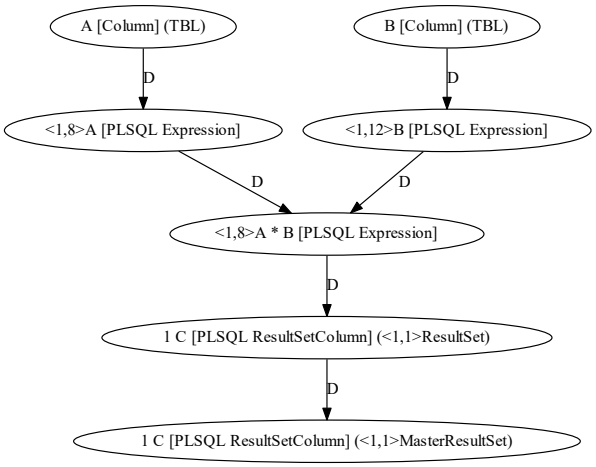
```
...
<1,8> AST_RESULTSET_COL <AstResultSetColumn> - ...
  <1,8> AST_EXPR <OracleAstNode>
    <1,8> AST_OPERATION <AstOperation>
      <1,8> AST_AGG_REF <AstAggRef> - ...
```

```
        <1,8> AST_NAME_SEGMENT <AstNameSegment> - ...
          <1,8>  <OracleAstNode>
          <1,8> a <OracleAstNode>
      <1,10> AST_OPERATOR <OracleAstNode>
        <1,10> * <OracleAstNode>
      <1,12> AST_AGG_REF <AstAggRef> - ...
        <1,12> AST_NAME_SEGMENT <AstNameSegment> - ...
          <1,12>  <OracleAstNode>
          <1,12> b <OracleAstNode>
  <1,14> AS <OracleAstNode>
  <1,17> AST_ALIAS <AstAlias>
    <1,17> c <OracleAstNode>
...
```

3.2 Výsledný graf datových toků

Jedná se o graf reprezentující datové toky ve zkoumaném systému a jedná se o výstup z Manty. Výsledný graf vzniká primárně z AST a následně je modifikován (obohacován informacemi) při průchodu jednotlivými moduly. Expressions modul je jedním z těchto modulů a přispívá tak k doplnění dalších informací do generovaného výsledku. Dalším důležitým modulem pro Expressions modul je takzvaný „Filter Task“ (viz kapitola číslo 3.3).

V rámci vývoje Expressions modulu bylo nutno řešit několik specifik při generování grafu datových toků z AST, které bylo nutno upravit pro jeho potřeby. S těmito změnami souvisela i příslušná úprava existujících automatických testů. Jednalo se ale pouze o přidání vrcholů grafu, které nebyly pro zbytek Manty využívány a z tohoto důvodu byly z generování vynechány. Díky v praxi běžně používanému modulu závěrečné filtrace (viz kapitola číslo 3.3), nedošlo ke změně finálního výstupu.



Obrázek 3.1: Příklad grafu datových toků.

Obrázek číslo 3.1 je příkladem grafu datových toků generovaným Mantou a zobrazeným pomocí knihovny `dot` (viz kapitola číslo 3.4) pro předchozí SQL

skript z kapitoly číslo 3.1. Grafy datových toků obsahují další technické vrcholy, které nejsou pro popisovaný případ podstatné a jsou proto z důvodu přehlednosti v rámci této práce vynechány z přikládaných obrázků.

3.3 Filter Task

Jedná se o modul Manty běžně používaný pro tvorbu finálního grafu. Jeho úkolem je z grafu odfiltrovat většinu technických vrcholů nepřinášejících uživateli důležité informace pro Data Lineage.

Při tomto procesu dochází k odstraňování jednotlivých vrcholů grafu, kdy je ovšem nutné v něm zachovat všechny existující cesty s výjimkou těch začínajících nebo končících v odstraňovaném vrcholu. Jinými slovy, pokud mezi dvěma vrcholy existovala cesta před spuštěním Filter Tasku a ani jeden z vrcholů nebyl odstraněn, pak mezi nimi existuje cesta i po jeho dokončení.

Tento modul byl pro vývoj Expressions modulu podstatný ze dvou důvodů. Za prvé umožnil rozšíření generovaného grafu z AST, bez dopadu na výsledek prezentovaný uživateli. Za druhé výrazněji ovlivnil nutnost určení finálního vrcholu grafu, do kterého se ukládá výsledná poskládaná transformace pro jednotlivé atributy (viz kapitola číslo 8.3).

3.4 Knihovna dot z balíku Graphviz

Graphviz je open source software pro generování grafické reprezentace grafu (viz Literatura [4]). Pro potřeby Manty se jedná o balík `dot`, který je využit pro generování png obrázků grafů s orientovanými hranami z jejich textové reprezentace.

Knihovna není využita pro finální fungování Manty, ale k usnadnění vývoje. Poskytuje totiž možnost rychlého vygenerování grafické reprezentace grafu vzniklého ze vstupního SQL skriptu. Díky tomu je možné při vývoji rychleji zjistit jak bude vypadat výsledný graf a zorientovat se v něm. Toto bylo důležité především pro druhou fázi fungování Expressions modulu (viz kapitola číslo 8), kde je jako zdroj dat využit právě tento graf.

Při práci se ukázalo že takto generované obrázky grafů jsou přínosné pouze pro samostatné SQL příkazy nebo velmi malé skripty. U větších skriptů dochází ke vzniku příliš velikého počtu vrcholů a obrázek přestává být přehledný a tedy užitečný. Dalším omezením je složitost tohoto procesu, kdy pro větší grafy je generování obrázku pomalé a velmi náročné na paměť počítače.

4. Omezení kladená na výstupní formát

Jednou z hlavních součástí této práce byl návrh formátu prezentace jednotlivých transformací klientovi. Bylo ovšem nutno dodržet sadu následujících podmínek, aby bylo možné výsledek prezentovat v Mantě a zároveň jej bylo možno použít i jako vstupní data pro zpracování dalšími analytickými nástroji, které jsou již nyní navázány na výstup z Manty.

4.1 Podmínky na formát výstupu

1. Výstup musí být textový. Tato podmínka je dána tím, že bylo požadováno zahrnutí výsledku do stávajících standardních výstupů Manty jako jeden z atributů v rámci vrcholů výsledného grafu datových toků. Tím je také umožněno automatické uložení spolu s výsledky z Manty do grafové databáze.

2. Výstup musí být jeden ucelený blok textu pro každou transformaci. Toto je doplňková podmínka k podmínce číslo 1, z důvodu použití v Mantě.

3. Výstup musí být čitelný samostatně bez kontextu SQL skriptu, ve kterém byl definován. Tato podmínka indikuje, že je nutné v rámci výstupu nepoužívat lokální jména jednotlivých databázových atributů a proměnných, ale jednoznačná kvalifikovaná jména.

4. Výstup musí být prostý text. Z důvodu zachování možnosti dalšího zpracování výstupu Expressions modulu nelze použít žádný způsob úpravy stylu jednotlivých částí textového výstupu (například velikost fontu nebo tučné písmo).

5. Není potřeba lokalizovat. Tato podmínka nebyla povinná, ale vzhledem ke globálnímu využití Manty vhodnější. Není poté nutné chystat lokalizaci výstupu pro jednotlivá jazyková prostředí, ve kterých je nasazena.

4.2 Zvažované nepoužité výstupní formáty

Jedním z prvních rozhodnutí, které bylo nutno na počátku této práce učinit, byla volba typu formátu výsledného popisů transformací jednotlivých atributů.

Tato kapitola popisuje zvažované formáty a důvody, kvůli kterým byly zavrhnuty pro další podrobnější analyzování.

4.2.1 Grafická reprezentace

Tento formát měl reprezentovat výstup pomocí vývojového grafu. Jednotlivé vstupy transformace by byly reprezentovány vrcholy pouze s výstupními hranami a vrchol pouze se vstupními hranami by reprezentoval cíl transformace. Ostatní vrcholy grafu by reprezentovaly jednotlivé transformace.

Výhodou takovéhoto popisu je čitelnost i pro uživatele bez technických znalostí a znalosti SQL. Hodí se také jako podkladový materiál do prezentací o zkoumaném systému, jako grafický popis transformace o kterém je možno dále hovořit.

Její nevýhodou je oproti tomu rozsáhlost při čtení (pro složité transformace může graf obsahoval veliké množství vrcholů), horší použitelnost při vyhledávání ve výsledcích a především obtížná použitelnost pro další zpracování.

Tato reprezentace byla zamítnuta z důvodů:

- Jedou ze dvou vstupních datových struktur je graf datových toků, který i když neobsahuje tolik detailů jako navrhovaná reprezentace, je již součástí Manty a naopak je záměrně z důvodu přehlednosti redukován počet jeho vrcholů.
- Nesplnění podmínek na formát výstupu, které byly dodefinovány v počátcích práce, ale až po vzniku návrhu této reprezentace.

4.2.2 Textová reprezentace větami

Tento formát měl reprezentovat výstup textem, co možná nejblíže se podobající lidské řeči. Tedy přepis do lidsky čitelných vět.

Pokud by bylo možné generovat rozumně dlouhý a srozumitelný popis, pak by se tento formát dobře hodil pro netechnické uživatele. To je ovšem pro obecné SQL skripty a obzvláště pro komplikované transformace, na které se Manta převážně používá, nereálné. Z praxe víme, že takovýto popis, i když jej sestavuje člověk expert na danou problematiku, může být značně netriviální a ve výsledku hůře srozumitelný než například matematický zápis příslušné transformace.

Další důvody, které vedly k opuštění tohoto výstupního formátu:

1. Nevhodnost pro zpracování dalšími nástroji, kromě přímého zobrazení.
2. Nutnost lokalizace související odlišnou větnou skladbou v různých jazycích (toto je nevýhodou především vzhledem k rozsahu pracnosti a s ní spojeným nákladům na další rozvoj a údržbu).
3. Horší čitelnost pro technické uživatele (v obecném případě by popis byl méně strukturovaný a delší na čtení, tedy zorientování se v něm by trvalo déle něž v původním SQL skriptu).

4.2.3 Použití originálního SQL

Složení výsledného popisu transformace z částí vstupního SQL skriptu, odpovídající pouze zkoumanému atributu.

Tento formát má výhodu pro technické uživatele, jelikož poskytuje snadnou referenci na původní zdrojový kód a tedy snazší lokalizování místa případné potřebné úpravy.

Pro netechnické uživatele ovšem nepřináší kromě odfiltrování zdrojového kódu, který se nepodílí na transformaci tvořící zkoumaný atribut další zjednodušení v čitelnosti. Zvýšení čitelnosti je možné řešit dalšími formátovacími úpravami.

K nim je ovšem nutné začít tvořit vlastní formát popisu z jednotlivých elementů SQL skriptu, ne nutně textově přítomných ve vstupním SQL skriptu.

Pro fungování je nutné vstupní SQL skript rozložit na jednotlivé tokeny a analyzovat jejich význam. Jestliže však již máme tuto analýzu hotovou v AST, není nutné se při tvorbě výstupního popisu transformace svazovat původní textací vstupního SQL skriptu, ale je možné definovat vlastní formát.

Tato reprezentace byla zamítnuta z důvodů:

1. Nesplnění podmínky číslo 3 z kapitoly číslo 4.1 na formát výstupu.
2. Nutnost dodělávat rozšířené formátování dodatečně způsobem neodpovídajícím tomuto zpracování.

5. Zvolený výstupní formát

Finálně zvolený způsob reprezentace byl inspirován následujícími vzory:

1. Definicí hodnoty atributu v programovacích jazycích, jako přiřazení hodnoty výrazu.

   Tento způsob zápisu je velmi přirozený pro technické i netechnické uživatele a tím usnadňuje orientaci ve výsledném popisu.

2. Jazyk SQL, který je analyzovaným vstupem, je použit i jako hlavní způsob popisu transformace.

   Důvodem je zachování jednotnosti výstupního formátu se vstupním, pro usnadnění orientace mezi nimi. Tato volba byla provedena primárně s ohledem na technické uživatele, ale i značná část netechnických uživatelů dokáže číst jednodušší SQL skripty.

3. Formátem zápisu syntaxe technických jazyků, jako je například Oracle SQL nebo jiné programovací jazyky. Ze kterých využívá složení popisu z elementů, které jsou definovány ve vlastním podpopisu.

   Rozdělení do menších částí umožnilo strukturování a lepší formátování výsledného popisu. Pro tvorbu výsledného popisu je ovšem nutné řešit spojení těchto již příliš krátkých podpopisů do vhodně dlouhých a čitelných bloků.

Ilustrativní příklad[1] očekávaného výstupu:

```
VYDAJE.CELKEM := SUM(PLATBY.CENA) * KURZ
  KURZ := 1.2
```

Formát vychází z textové reprezentace popisující jednotlivé zkoumané atributy za použití částí SQL skriptů. Atribut je reprezentován jako cíl přiřazení, do kterého je vkládán výsledek popisu. Jednotlivé části popisu mohou být dále reprezentovány svým vlastním podpopisem na novém odsazeném řádku. Tím je tvořeno strukturování formátu a zajištěna čitelnost výsledného popisu.

Plné oproštění se od SQL by bylo sice možné, ale vzhledem k cílové skupině uživatelů, pro kterou má být Expressions modul určen, není příliš vhodné. Techničtí uživatelé by tím přišli o reference na zdrojový SQL skript, ze kterého popis vznikl. Tím by přišli o možnost snadného lokalizování případných chyb/problémů, s jejichž identifikací má Expressions modul pomáhat. Pro netechnické uživatele by opuštění SQL mohlo být přínosem, ale u jednodušších transformací by měl být tento zvolený formát čitelný i pro ně. Zatím co v případě složitých transformací je typicky vyžadována konzultace s expertem znalým v dané problematice a implementaci.

Tento formát se postupem analýzy ukázal pro prezentaci uživatelům jako nejvhodnější. Dokonce v průběhu implementace prototypu Expressions modulu, nezávisle zaslal jako očekávaný výstup velmi obdobný formát jeden ze současných zákazníků Manty.

[1] Tento příklad slouží pouze pro ilustraci, jak by mohl výsledný popis vypadat. Oproti finálnímu formátu se liší tím, že `KURZ` mohl být pro lepší čitelnost nahrazen svojí hodnotou již v prvním řádku výsledného popisu.

5.1 Volba použitých vstupních dat

Jak bylo řešeno již v kapitole číslo 4.2.3, tak využití větších bloků originálního textu SQL skriptu není vhodné. Pokud ovšem začneme velikost bloků zmenšovat, dostáváme se postupně až na jednotlivé tokeny, ze kterých je tvořen SQL skript, tedy stejné tokeny které jsou použity jako vrcholy AST. AST navíc nabízí jejich analýzu významu a vazeb. Z těchto důvodů bylo AST zvoleno jako první zdroj dat pro Expressions modul.

V AST jsou hůře detekovatelné cíle pro uložení výsledku z některých částí SQL skriptu. Ty jsou naopak velmi dobře popsány v grafu datových toků, který je právě za tímto účelem tvořen. Pokud tedy pro část SQL skriptu nalezneme jeho obraz v grafu datových toků, lze z něj přímo vyčíst, kam je jeho výsledek ukládán. Dalším podstatným přínosem grafu datových toků je rozdělení vstupního SQL skriptu na samostatné části pro jednotlivé výstupní atributy tvořící jeho oddělené podgrafy. Každý podgraf popisuje pouze transformace, mající vliv na jeden výstupní atribut. Graf datových toků se pro tyto vlastnosti stal druhým zdrojem dat pro Expressions modul.

Samotný AST poskytuje kompletní informaci obsaženou v SQL skriptu a jako vstup by tedy byl postačující. Graf datových toků je využit, jelikož již existují standardní způsoby jeho tvorby z AST a pro fungování Expressions modulu přináší komfortnější přístup k cílům výstupů z jednotlivých částí SQL skriptu. Celkově tedy Expressions modul využívá dva zdroje dat: AST a graf datových toků.

5.2 Příklad formátu

Příklad vstupu a výstupů pro SQL skript mající na výstupu dva atributy:

```
INSERT INTO vydaje(datum, celkem)
SELECT datum, SUM(cena)
FROM platby
WHERE datum = TO_DATE(sysdate - 1, 'DD.MM.YYYY')
  AND typ = 'vydaj'
GROUP BY datum

UNION

SELECT datum, hodnota * zmena_kurzu
FROM kapital
WHERE datum = TO_DATE(sysdate - 1, 'DD.MM.YYYY')
  AND zmena_kurzu < 0
GROUP BY datum
;
```

```
VYDAJE.CELKEM := VAR_0 UNION VAR_6
  VAR_0 := SUM(PLATBY.CENA)
  VAR_6 := KAPITAL.HODNOTA * KAPITAL.ZMENA_KURZU
```

```
VYDAJE.DATUM := VAR_0 UNION VAR_6
  VAR_0 := PLATBY.DATUM
  VAR_6 := KAPITAL.DATUM
```

5.3 Informace obsažená ve výsledném popisu

Výsledný popis se zaměřuje na to z jakých zdrojových dat a jakým způsobem vznikají jednotlivé výstupní atributy. Pro popis je tedy podstatná logika použité transformace dat.

Jak je vidět z příkladu v kapitole číslo 5.2, tak výsledný popis neobsahuje veškeré informace obsažené v SQL skriptu. Především jsou vynechány jednotlivé podmínky. Toto je požadovaný stav vzhledem k využití pro Data Lineage, kde jsou tyto podmínky vynechány z důvodu zpřehlednění celkového datového toku. V globálním pohledu je podstatné především odkud kam jsou data přesouvána a podmínky jsou řešeny až v druhé řadě.

I když nejsou veškeré informace obsaženy v popisu transformace generované Expressions modulem, tak klient je má stále dostupné v rámci originálního vstupního SQL skriptu nebo v ostatních nástrojích, kterými je zpracovává. V případě prototypové implementace v Mantě, jsou tyto informace dostupné jednak v podobě filtračních hran v grafu datových toků i v atributu který obsahuje odpovídající původní SQL skript.

5.4 Formátování jednotlivých SQL bloků

Pro snadnou definici, implementaci i udržovatelnost by bylo ideální, pokud by existovala jedna obecně platná sada pravidel, podle které by byl celý SQL skript převeden do výsledného popisu transformace. Takovouto sadu pravidel ale vzhledem ke komplexnosti SQL jazyka a množství výjimek, které obsahuje, není možné definovat. Namísto toho vznikla co možná nejobecnější a minimalistická sada pravidel, která pokrývá co možná největší část SQL jazyka a je doplněna výjimkami/speciálními pravidly pro konstrukty, pro které není optimální.

5.4.1 Obecná pravidla

Pro přesnější vysvětlení jednotlivých pravidel budeme využívat příklad z kapitoly číslo 5.2.

1. Každá SQL operace je popsána pomocí původních klíčových slov a operátorů z SQL jazyka, jako přiřazení výsledku této operace do nějakého atributu/proměnné.

   Toto se projeví například tak, že výsledek násobení atributů `hodnota` a `zmena_kurzu`, je pomocí standardně používaného operátoru přiřazení „`:=`“ uložen do proměnné `VAR_6`.

   Načtení atributu je také chápáno jako samostatná operace, což je vidět například na přiřazení atributu `datum` do proměnné `VAR_0`.

2. Pojmenování všech vstupních atributů je zapisováno pomocí plně kvalifikovaného jména.

   Toto se projeví například u jména vstupního sloupce `hodnota` z tabulky `kapital`, který je převeden na `KAPITAL.HODNOTA`.

3. Každý popis se může odkazovat na podpopisy jiných podoperací, které jsou zapsány s odsazením pod ním a mají platnost pouze v takto odsazením ohraničeném bloku výsledného popisu. Odkaz je zapsán použitím výsledku podpopisu v rámci popisu zkoumané operace.

   Toto se projeví například při popisu výsledného atributu `celkem`, který vzniká množinovým spojením hodnot z proměnných `VAR_0` UNION `VAR_6`, které jsou definovány jako podpopisy na následujících řádcích.

4. Žádný samostatný popis (tedy jedno přiřazení bez svých přidružených podpopisů) nesmí obsahovat znak nového řádku.

   Toto pravidlo je důležité pro udržení struktury a čitelnosti formátování výsledného popisu obsahující příslušně odsazené podpopisy, aby nedocházelo k zaměnění podpopisu za část popisu zapsané na více řádcích.

5.4.2 SQL CASE

Tento výraz SQL jazyka je ze své podstaty rozsáhlejší a jeho zápis v konstruovaném popisu by tedy při zapsání do jednoho řádku textu podle pravidla 4 obecných pravidel (podle kapitoly číslo 5.4.1) byl špatně čitelný. Je tedy nutné pro něj utvořit výjimku a popisovat jej speciálním způsobem.

Při konstrukci formátu pro SQL `CASE` výraz byl hledán popis, který by byl víceřádkový, ale zároveň by bylo možné jej použít v kombinaci s ostatními popisy které obecná pravidla dodržují. Toto je důležité především při zvažování odsazení dalších řádků, aby nedocházelo k chybné interpretaci finálního popisu.

Zvolen byl formát, kde jsou jednotlivé větve vypsány na samostatné řádky s dvojnásobným odsazením a popis je zakončen klíčovým slovem „`END`“ na samostatném řádku bez odsazení. Dvojité odsazení pomáhá při odlišení větví `CASE` výrazu od jeho případných podpopisů a „`END`“ zajišťuje opticky přesné ohraničení utvořeného popisu.

Příklad SQL skriptu obsahující `CASE` výraz:

```
INSERT INTO platy(vyplata)
SELECT CASE zam.prodejce
  WHEN 'Ano' THEN zam.plat ELSE zam.plat * bon.bonus END
FROM zamestnanci zam,
  (
    SELECT pravidelny + premie AS bonus
    FROM bonusy
  ) AS bon
;
```

```
PLATY.VYPLATA := CASE ZAMESTNANCI.PRODEJCE
    WHEN 'Ano' THEN ZAMESTNANCI.PLAT
    ELSE ZAMESTNANCI.PLAT * BON.BONUS
END
  BON.BONUS := BONUSY.PRAVIDELNY + BONUSY.PREMIE
```

5.5 Zkrácení zápisu

Při dodržení pravidel pro tvorbu popisu podle kapitoly číslo 5.4.1 by vznikl strukturovaný popis, ale pro běžné čtení by typicky byl příliš dlouhý a tedy nepraktický. Důvodem je že každý řádek by obsahoval pouze jednu SQL operaci a ostatní operace by následovaly jako podpopisy. Výsledný popis by tedy byl složen z příliš velikého množství jednoduchých popisů.

Pro člověka je ovšem takovéto dohledávání definice proměnné v podpopisu skákáním v textu nepřirozené a náročné na paměť, což vede ke špatné čitelnosti výsledného popisu. Hlavním cílem Expressions modulu je ovšem zjednodušit a zpřehlednit popis transformace. Pro dokončení formátování výsledného popisu, je tedy nutné provést ještě další krok, který zápis zkrátí.

Zkrácení zápisu, především co do počtu řádků a tedy nutných logických skoků při jeho čtení, je dosaženo pomocí „inliningu“. Při tomto procesu jsou jednotlivé podpopisy vkládány na místo jejich odkazu v nadřazeném popisu.

Inlining není možné použít pro všechny podpopisy, z několika různých důvodů:

porušení významu popisu Je nutné hlídat, že smysl popisu se po provedení inliningu nezmění z důvodu porušení precedence operací. Například není možné v popisu $a * b$ naradit b, pokud by jeho podpopis byl $c + d$.

Tento případ je možné řešit pomocí přidání závorek okolo vkládaného bloku. Popis ovšem vznikal z původního SQL skriptu, který musel buďto závorky obsahovat a pak je možné je zachovat i při tvorbě popisu. Nebo musela možnost takovéhoto inlinigu vzniknout některou z nepřímých cest (například vnořený `SELECT`).

Byla zvolena varianta zachování podpopisu bez inliningu a bez použití závorek. Důvodem rozhodnutí je že v tomto případě se typicky jedná o komplexnější případ popisu, který je primárně určeny pro technické uživatele. Těm varianta bez použití inlinigu nabízí snazší mapování na původní SQL skript a tím i možnost jeho úprav.

porušení pravidel formátování Je nutné hlídat, že při inliningu nedošlo k porušení obecných pravidel formátování definovaných v kapitole číslo 5.4.1. K tomuto může nejtypičtěji dojít pokud je podpopis víceřádkový viz výjimka pro `CASE` podle kapitoly číslo 5.4.2 nebo pokud má daná proměnná více možných definicí (k tomuto může dojít například v PLSQL skriptu, kde je hodnota některé proměnné běhově měněna a není možné statickou analýzou kódu zjistit její obsah).

příliš dlouhý řádek Je nutné hlídat délku vznikajícího řádku s popisem po provedení inliningu. Jinak by mohlo velmi snadno dojít ke sloučení celého popisu do jednoho dlouhého nestrukturovaného řádku textu, který je také špatně čitelný. Protože pro každého uživatele a analyzovaný systém může být vhodná délka řádku jiná, je vhodné mít konfiguračně nastavitelný parametr, který tuto délku omezí na definovanou mez. Například pro systém používající dlouhé identifikátory v rámci svých SQL skriptů, bude vhodná větší délka, aby popis tato jména mohl pobrat a zároveň měl možnost provést alespoň základní inlining.

6. Struktura Expressions modulu

Jedná se o prototyp implementace modulu do Manty pro potřeby ověření realizovatelnosti a vhodnosti zvolených postupů v této práci.

6.1 Použití modulu

Celý modul vystupuje v implementaci pod třídou `ExpressionsTask`. Jedná se o samostatný modul, který je možno zapojit do zpracování Manty, za účelem obohacení výstupu o prezentaci transformací jednotlivých atributů z Oracle SQL.

Předpokladem použití Expressions modulu je jeho spuštění až po modulech tvořících AST a graf datových toků, protože tyto dvě struktury jsou využity jako vstupní data pro jeho fungování. Dalším předpokladem je spuštění před modulem Filter Task (viz kapitola číslo 3.3), který má za úkol odstranění značné části vrcholů grafu. Tyto vrcholy jsou ale pro fungování Expressions modulu nutné, protože obsahují potřebné informace pro jeho fungování a mohou být cílem pro vnitřní odkazy vzniklé v první fázi zpracování Epressions Modulu (viz kapitola číslo 7).

Expressions modul se do zpracování Manty vkládá projektově standardním způsobem, definováním modulové základní Beany[1] a jejím zapojením do Spring konfigurace[2]. Toto přidání modulu se provádí v XML souboru s konfigurací Manty pro konkrétní analyzovaný systém (každý systém má typicky své specifické požadavky na konfiguraci a požadavky na výstup), kde je také možné upřesnit konfiguraci Expressions modulu jako takového (viz kapitola číslo 6.4).

6.2 Základní princip fungování

Modul pracuje ve dvou základních fázích:

1. Sestavení popisu jednotlivých částí transformace z AST a jejich uložení k příslušným vrcholům grafu datových toků (viz kapitola číslo 7).

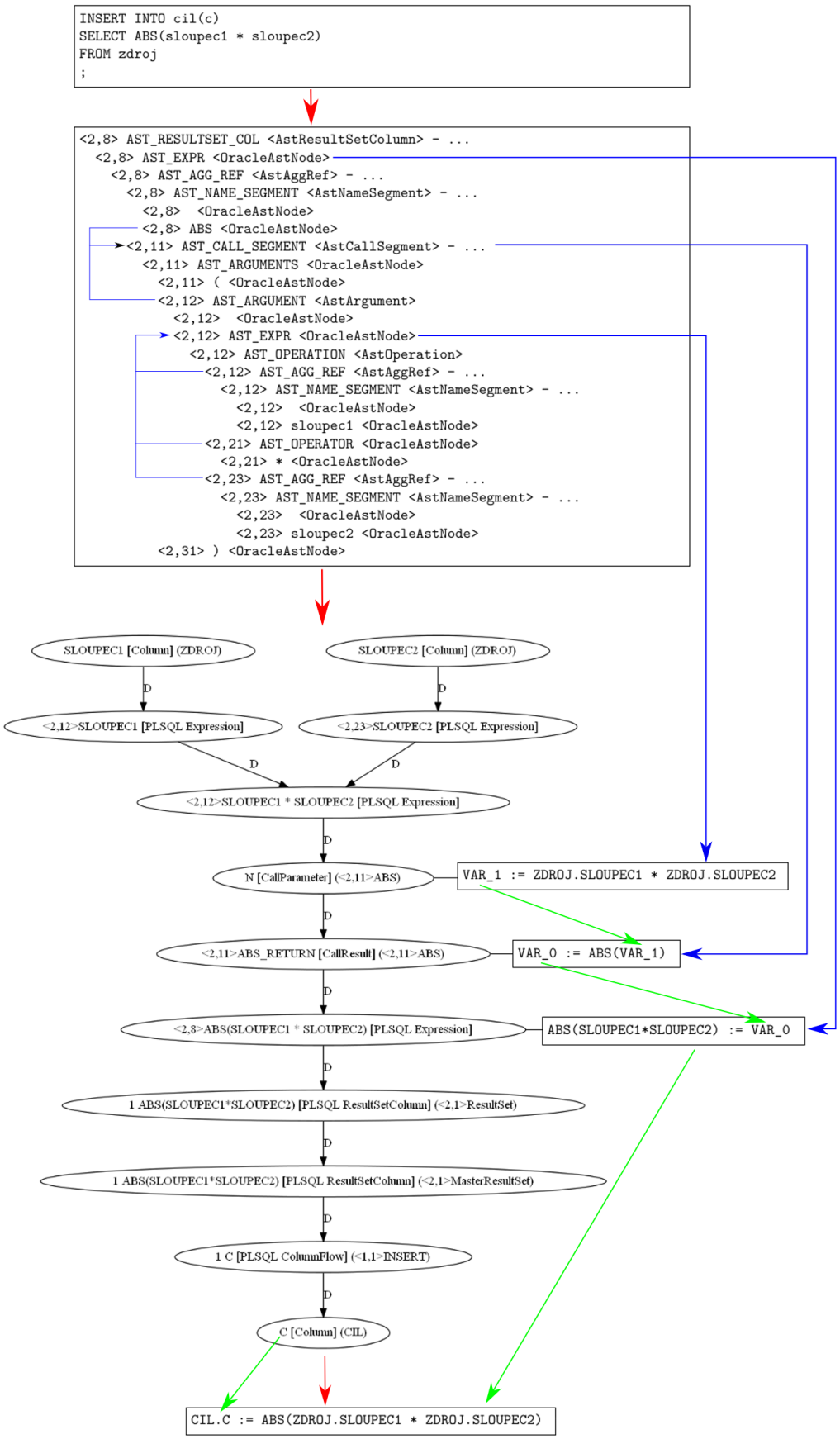
   Z AST není vždy přímo patrné do kterého SQL elementu (atribut, proměnná, ...) je výsledek transformace ukládán. Proto se tato informace doplňuje v druhé fázi.

2. Spojení jednotlivých částí popisu transformace vygenerovaných v první fázi, do výsledného popisu z grafu datových toků (viz kapitola číslo 8).

   Nejprve jsou jednotlivé části transformace posbírány z vrcholů grafu datových toků a následně dochází ke zkrácení popisu pomocí nahrazení lokálních proměnných za jejich skutečný popis (tzv. inlining).

[1] Beana je standardní Java třída, která splňuje požadavek na serializovatelnost, existenci bezparametrického konstruktoru a přístup k jednotlivým atributům pomocí get/set metod. Podrobnější definice viz Literatura [5].

[2] Spring je framework použitý ke správě životního cyklu a propojení jednotlivých definovaných Bean. Podrobnější definice viz Literatura [6].



Obrázek 6.1: Přehled fungování Expressions modulu a toku informací v rámci jeho běhu.

Tento obrázek číslo 6.1 zachycuje kompletní průběh zpracování Expressions modulu od vstupního SQL skriptu po výsledný popis transformace (z důvodu množství dat nebylo možno použití rozsáhlejšího příkladu).

Červené šipky Popisují přechod mezi jednotlivými datovými strukturami a fázemi zpracování Expressions modulu. Význam jednotlivých šipek od vrchu obrázku:

1. převod vstupního SQL skriptu do AST (zajišťuje poskytnuté prostředí viz kapitola číslo 3.1)
2. první fáze zpracování (předzpracování pro druhou fázi, viz kapitola číslo 7) a převod AST do grafu datových toků (zajišťuje poskytnuté prostředí viz kapitola číslo 3.2)
3. druhá fáze zpracování (tvorba výsledného popisu transformace, viz kapitola číslo 8)

Modré šipky Popisují průběh první fáze zpracování (viz kapitola číslo 7).

Šipky na levé straně znázorňují sběr podpopisů bez jejich uložení o které se stará některý z nadřazených AST vrcholů.

Šipky na pravé straně znázorňují ukládání jednotlivých utvořených podpopisů k vrcholům grafu datových toků.

Zelené šipky Popisují průběh druhé fáze zpracování (viz kapitola číslo 8), tedy skládání jednotlivých podpopisů a jmen pro uložení výsledku z grafu datových toků do výsledného popisu transformace zkoumaného atributu.

Největší komplikací při implementaci se ukázala velmi volná vazba mezi vrcholy AST a vzniklého grafu datových toků. Manta totiž tuto vazbu standardně nepotřebuje, protože se k AST po vygenerování grafu datových toků již nevrací. V rámci generování grafu je potřeba za určitých okolností z AST určit již vygenerovaný vrchol znovu, ale toto neplatí obecně pro všechny vrcholy. Graf je sice generován přímo z AST, ale po jeho vygenerování není ve všech případech jasné mapování mezi jejich jednotlivými vrcholy. V ideálním případě by pro všechny významové vrcholy AST (využívané v Expressions modulu) mohlo existovat mapování $1 : 1$ na vrcholy grafu, ale reálně tomu tak není.

6.3 Určení cíle popisu transformace

Jméno atributu do kterého se výsledek podvýrazu nebo celého popisu transformace zapisuje se určuje v obou fázích fungování Expressions modulu. V první fázi se určují jména lokálních atributů jako jsou dočasné proměnné (podrobněji viz kapitola číslo 7.7), zatím co v druhé fázi se detekují finální jména atributů, kterými se nahrazují jména z první fáze.

Rozdělení detekce do obou fází je nutné, protože z AST není vždy triviálně patrné do jakého atributu se bude výsledek finálně ukládat. Tato informace je sice z AST dostupná, ale její získání by duplikovalo činnost Manta modulu pro tvorbu grafu datových toků z AST.

Příklad dvou případů, kdy není jméno cílového atributu patrné z AST:

- Zápis do SQL tabulky prostřednictvím `INSERT INTO` přes pořadí atributu.

```
INSERT INTO zamestnanci(jmeno, prijmeni)
VALUES('Jan', 'Novák');
```

- Zápis do proměnné v PLSQL bloku skriptu.

  Dáno tím že Expressions modul vyhodnocuje pouze samotné výrazy a cíl přiřazení tedy není zpracován. Toto je požadovaný stav z důvodu požadavků na výstupy z Expressions modulu s ohledem na stávající funkčnost Manty.

Mohlo by se zdát, že všechna jména cílových atributů tedy budeme získávat až ve druhé fázi zpracování z grafu datových toků. To ovšem také není možné, protože v něm se již nevyskytují dočasné proměnné, použité při tvorbě popisu. Popis je generován z co možná nejmenších částí (viz kapitola číslo 7.3), aby poskytl dostatečný prostor pro fungování inliningu (viz kapitola číslo 8.5).

6.4 Konfigurace

Proměnné prostředí (využity pouze pro testovací scénáře):

manta.test.saveFiles Pokud je nastavena zapíná, že se v rámci běhu testu mají ukládat do souborů jednotlivé stavy vnitřních datových struktur. Ovlivňuje vznik souborů: `TEST_tree.txt`, `TEST_flow.txt`, `TEST_expressions.txt` a `TEST_expected.txt` (viz kapitola číslo 9.3).

manta.test.savePng Pokud je nastavena zapíná, že se v rámci běhu testu má generovat obrázek grafu datových toků, do maximálního počtu 500 vrcholů (větší graf je již příliš náročný na vygenerování a kvůli jeho rozsahu se snižuje jeho užitečnost). Ovlivňuje vznik souborů: `TEST_graph.png` a `TEST_graph.png.dot`.

Konfigurace Expressions modulu ve Spring XML konfiguraci:

stopOnVariables Ovlivňuje průchod algoritmu přes proměnné. Zda má obsahovat i způsob vzniku použité proměnné při skládání popisu atributu (viz kapitola číslo 8.4). Pokud není popis vzniku proměnné připojován do místa jeho použití, pak je potřebné zajistit vznik jeho samostatného popisu (viz kapitola číslo 8.3).

maxQualifiedNameLevel Určuje délku kvalifikovaného jména atributu[3], jako maximální počet kvalifikátorů včetně jeho jména (plně kvalifikované jméno je ve většině případů příliš dlouhé a zhoršovalo by čitelnost výsledného popisu transformace).

maxInliningLength Určuje maximální délku, kterou nesmí přesáhnout inline podvýrazů transformace v jednom řádku (nebere v potaz odsazení tohoto řádku v rámci formátování výsledné reprezentace transformace).

[3]Kvalifikované jméno atributu je pojmenování obsahující přesné označení podle hierarchie jeho umístění (například „DATABÁZE.SCHÉMA.TABULKA.SLOUPEC“).

genVariableFormat Vzor pro generování jmen dočasných proměnných vznikajících při analýze SQL v první fázi zpracování, které ze své podstaty mít jméno nemusí (například viz kapitola číslo 7.8.4).

Příklad konfigurace použitý pro automatické testovací scénáře:

```
<!-- ExpressionsTask -->
<bean
 id="expressionsTask"
 class="eu.profinit.manta.dataflow.generator.oracle.expressions.ExpressionsTask"
 scope="prototype">
    <property name="nodeCreator" ref="nodeCreator" />
    <property name="dbResource" ref="dbResource" />
    <property name="scriptResource" ref="scriptResource" />
    <property name="expressionsUtils" ref="expressionsUtils" />
    <property name="stopOnVariables" value="false" />
</bean>
<bean
 id="expressionsUtils"
 class="eu.profinit.manta.dataflow.generator.oracle.expressions.ExpressionsUtils"
 scope="prototype">
    <property name="maxQualifiedNameLevel" value="2" />
    <property name="maxInliningLength" value="80" />
    <property name="genVariableFormat" value="VAR\_\%d" />
</bean>
```

7. První fáze zpracování

Třída `ExpressionsVisitor` implementuje první fázi zpracování v Expressions modulu. Vstupem je AST generovaný z Oracle SQL skriptů v Mantě. A na výstupu jsou jednotlivé části transformace namapované na příslušné vrcholy grafu datových toků.

Tato kapitola je popsána na konkrétní implementaci v nástroji Manta (prototypová implementace). Popisované postupy je ovšem možné využít obecně pro libovolnou implementaci. Je pouze nutné vzít v úvahu specifika AST a grafu datových toků generovaných Mantou.

7.1 Výběr zdroje dat

Při průchodu AST se z jeho jednotlivých vrcholů staví samotný popis transformace. I když na první pohled vypadá skládání popisu v podobě velmi podobné původnímu SQL skriptu z AST místo ze vstupního SQL skriptu jako zbytečná práce a nevyužití vhodných dat ze vstupní textové reprezentace SQL, ukázalo se být nutné. Důvodů je několik:

- V původním SQL skriptu jsou jednotlivé atributy pojmenovány v kontextu jejich použití. Tedy typicky bez plného jména.
- Výsledný popis je požadováno přeformátovat do nějakého standardního tvaru. Zdrojové SQL skripty jsou standardně formátovány vzhledem k celému skriptu a ne pouze k sekci, týkající se jednoho atributu.
- Potřebujeme detekovat které části SQL skriptu jsou významné pro požadovaný výsledný popis. Toto je řešitelné i pomocí pozičních referencí, které obsahují jednotlivé vrcholy AST, ale při jejich využití se již velmi blížíme sestavování přímo z AST a vyhneme se případným komplikacím, které by mohly nastat.

7.2 Průchod AST

AST je procházen pomocí standardního visitor paternu (viz Literatura [7]), kdy se vyhodnocuje pouze podmnožina typů vrcholů zastoupených v AST. Důvodem je že AST reprezentuje kompletní Oracle SQL skript, ale pro Expressions modul jsou významné pouze vrcholy reprezentující operace, týkající se datových transformací, tedy vrcholy které se mohou vyskytovat uvnitř SQL výrazů. Přehled a popis zpracování jednotlivých typů uzlů je popsán v samostatné kapitole číslo 7.8.

7.3 Uložení podvýrazů do grafu vs. skládání

Při iniciální analýze a dekompozici modulu před samostatným vývojem, bylo počítáno v první fázi zpracování pouze se sběrem jednotlivých podvýrazů přímo

z příslušného AST vrcholu a jejich ukládáním na vrcholy grafu pro pozdější skládání ve druhé fázi. V průběhu vývoje se ovšem ukázalo, že toto není pro všechny AST vrcholy možné, protože k nim neexistuje příslušný grafový vrchol. Bylo tedy nutné tyto podvýrazy pouze vrátit z vyhodnocování aktuálního AST vrcholu a skládat je v jejich nadřazeném vrcholu již v první fázi zpracování.

Vracet a skládat podvýrazy již v první fázi je výkonově výhodnější, jelikož není nutné dohledávat příslušný grafový vrchol a ušetří se režie na samotné uložení podvýrazu. Nevýhodou je ovšem omezení prostoru pro inlining ve druhé fázi zpracování a lze jej provést pouze v případě, kdy je jasný datový tok. Není tedy žádoucí kompletně vypustit druhou fázi zpracování a skládat kompletní popis transformace již z AST. Celkově se ukázalo, že skládání menších podvýrazů již při zpracování AST, kromě omezení možností inliningu (které může být chápáno i jako výkonová optimalizace) je korektním zpracováním již v první fázi. Pokud nelze určit grafový vrchol odpovídající AST vrcholu, je podvýraz propagován zpět ke kořeni AST, dokud není nalezen typ vrcholu, ve kterém je uložen.

7.4 Odkazy se skoky na jiný zdroj dat v grafu

Jednotlivé podvýrazy načtené z AST jsou ukládány k příslušným vrcholům grafu, pro poskládání do finálního popisu transformace ve druhé fázi zpracování, při které se podvýrazy skládají při průchodu grafu podle místa jejich nalezení. Tento postup není možné aplikovat vždy a to z důvodu toho, jak je sestaven graf datových toků pro Mantu.

Jedná se o parametry při volání funkcí/procedur/přetypování, spojování výsledků několika `SELECT`ů v rámci množinových operací a všech podvýrazů SQL `CASE`. V těchto případech by zpracování standardním průchodem podle hran grafu vedlo k chybnému vyhodnocení, protože graf datových toků zohledňuje i deklarace příslušných operací, jako jsou například formální parametry funkce. Podrobněji bude rozepsáno u jednotlivých typů zpracovávaných AST vrcholů v kapitole číslo 7.8.

Ke každému ukládanému podvýrazu je tedy možné uložit kromě samotného popisu i referenci na grafový vrchol, a tím nahradit standardní průchod grafem za skok do místa určeného již při vyhodnocování AST. Znamená to ale, že je potřeba dohledat nejen grafový vrchol odpovídající aktuálně zpracovávanému AST vrcholu, ale také vrchol(y), ze kterých se mají získat jednotlivé podvýrazy.

7.5 Precedence

Při skládání a především inliningem ve druhé fázi zpracování je nutné znát význam jednotlivých podvýrazů, aby nedošlo k chybnému složení popisu, který by měl jiný význam. Konkrétně: není nutné znát kompletní význam jednotlivých podvýrazů, ale postačí jejich precedence. Například pro následující vstupní SQL skript:

```
insert into tbl(column)
select tmp * c
from (
```

```
    select a + b as tmp
    from src
)
;
```

Bez určení a dodržování precedence jednotlivých podvýrazů nebo preventivního uzávorkování při každém inliningu, by byl takovýto výstup nekorektní, protže má jiný význam:

```
TBL.COLUMN := SRC.A + SRC.B * C
```

Správný výstup musí vypadat například následujícím způsobem:

```
TBL.COLUMN := TMP * C
  TMP := SRC.A + SRC.B
```

Precedence operátorů je definována strukturou gramatiky pro stavbu AST, ale z již postaveného AST není k dispozici. Z tohoto důvodu se precedence jednotlivých operátorů a z nich složených podvýrazů určuje při zpracovávání AST vrcholů znovu a je ukládána spolu s popisem transformace do vrcholu grafu. Odtud je využita při inliningu v rámci druhé fáze fungování Expressions modulu.

Při implementaci byla zvolena reprezentace precedence jako číselná hodnota, rostoucí s klesající precedencí, jinými slovy jako pořadí vyhodnocení operátoru v rámci výrazu (například pro operátor „+“ byla zvolena hodnota 30, zatím co pro operátor „∗“ hodnota 20).

7.6 Ukládaná struktura popisu transformace

Při průchodu AST je sbírán především podvýraz popisu transformace, ale spolu s ním jsou ke každému ukládány i další metadata. Výsledná struktura získaná z AST a ukládaná k jednotlivým vrcholům grafu datových toků obsahuje následující atributy:

expression seznam tokenů reprezentující textový popis výrazu

inlineType informace zda je možné provádět inline do a z tohoto podvýrazu

target jméno atributu, do kterého se ukládá výsledek výrazu/podvýrazu

precedence precedence tohoto podvýrazu (viz kapitola číslo 7.5)

sources odkazy na vrcholy grafu datových toků ze kterých mají být získány podvýrazy použité v tomto výraze(viz kapitola číslo 7.4)

propagateThroughGraph zda se jedná o podvýraz který je nutné přenést do potomků grafového vrchu na kterém je definováno (jedná se o technický atribut vzniklý kvůli technické struktuře grafu datových toků použité v Mantě, viz kapitola číslo 8.2)

Seznam tokenů v atributu `expression` může být jednoho ze dvou typů:

proměnná slouží k uložení proměnných, které je možno při inliningu (viz kapitola číslo 8.5) nahradit za podvýraz

textový slouží pro uložení textové části popisu (například závorky nebo klíčová slova)

7.7 Zásobník jmen cílů podvýrazů

Při průchodu AST jsou detekována jména cílových atributů a proměnných v nadřazených vrcholech k aktuálně zkoumanému vrcholu. Z tohoto důvodu je vždy při nalezení nového cílového jména pojmenování uloženo do zásobníku, pro využití při zpracování jeho podstromu.

Způsob plnění:

1. AST obsahuje na některých svých vrcholech jména cílů podvýrazů, které neobsahují samotný popis podvýrazu. Jméno je pouze uloženo do zásobníku, je zpracován podstrom onoho vrcholu a jméno je opět ze zásobníku odstraněno, protože bylo platné pouze pro příslušný podstrom.

2. Některé AST vrcholy obsahují popis komplexnějšího výrazu. V takovémto případě jsou jednotlivé podvýrazy vyhodnocovány samostatně průchodem jejich příslušných AST podstromů, kdy je pro každý podstrom vygenerováno unikátní jméno dočasné lokální proměnné, která je použita v popisu komplexního výrazu a vložena na vrchol zásobníku na dobu jejich zpracování. Tyto lokální proměnné se nevyskytují nikde ve zdrojovém SQL skriptu a mohou tedy představovat pro koncového uživatele zhoršení čitelnosti popisu transformace. Protože tyto proměnné ovšem vznikají jako dočasné, jsou ve většině případů odstraněny v druhé fázi zpracování při inliningu. Výjimku tvoří případy, kdy by byl výsledek bez těchto lokálních proměnných hůře čitelný z důvodu formátování nebo příliš dlouhých řádků finálního popisu (konkrétní podmínky jsou dány procesem inliningu).

3. Do zásobníku je již při jeho vzniku vloženo první jméno, které se nikdy neodstraňuje. Slouží jako doraz v případě neočekávaného stavu a vyhnutí se pádu aplikace z důvodu čtení z prázdného zásobníku. Usnadňuje také detekci chyby Expressions modulu již pohledem na výstup, bez zkoumání aplikačního logu.

Použití specifického zásobníku do jisté míry duplikuje zásobník volání funkcí použitý pro samotný průchod AST. Místo předávání posledního nalezeného jména jako parametru průchodové funkce, které by nás při implementaci oprostilo potřeby při návratu ho ručně odstranit. Samostatný zásobník je ovšem nutné použít z důvodu využití v Mantě standardní implementace visitor paternu, která neumožňuje jméno předávat.

7.8 Zpracování jednotlivých typů operací

Tato kapitola popisuje zpracování jednotlivých AST vrcholů v první fázi zpracování Expressions modulu pomocí visitoru. Implementuje takto jednotlivá pra-

vidla tvorby popisu podle kapitoly číslo 5.4.1.

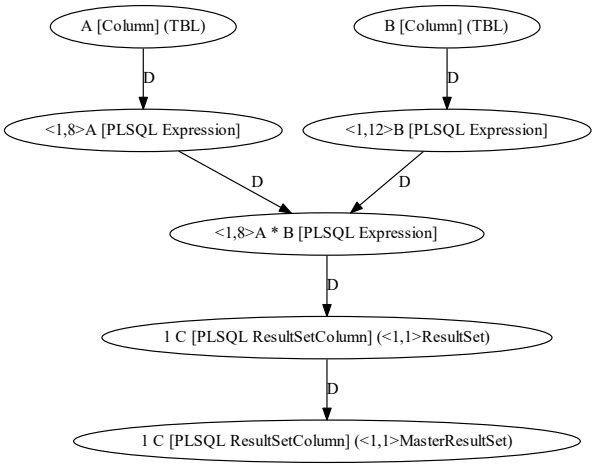
Jak bude vidět z následujícího příkladu, je značná část typů zastoupena i v základním Oracle SQL skriptu. Tento skript bude použit jako obecný příklad i pro část následujících podkapitol podle jednotlivých typů AST vrcholů.

Příklad základního Oracle SQL skriptu:

```
SELECT a * b AS c
FROM tbl
;
```

Příklad části exportu AST vygenerovaného z předchozího Oracle SQL skriptu:

```
...
<1,8> AST_RESULTSET_COL <AstResultSetColumn> - ...
  <1,8> AST_EXPR <OracleAstNode>
    <1,8> AST_OPERATION <AstOperation>
      <1,8> AST_AGG_REF <AstAggRef> - ...
        <1,8> AST_NAME_SEGMENT <AstNameSegment> - ...
          <1,8>  <OracleAstNode>
          <1,8> a <OracleAstNode>
      <1,10> AST_OPERATOR <OracleAstNode>
        <1,10> * <OracleAstNode>
      <1,12> AST_AGG_REF <AstAggRef> - ...
        <1,12> AST_NAME_SEGMENT <AstNameSegment> - ...
          <1,12>  <OracleAstNode>
          <1,12> b <OracleAstNode>
  <1,14> AS <OracleAstNode>
  <1,17> AST_ALIAS <AstAlias>
    <1,17> c <OracleAstNode>
...
```



Obrázek 7.1: Graf datových toků pro základní Oracle SQL skript.

7.8.1 Začátek výrazu

Začátek libovolného výrazu je reprezentován AST vrcholem `AST_EXPR`, který je v Mantě implementován pomocí datového typu `IOracleAstNode`.

Z obecného pohledu by se mělo jednat o AST vrchol reprezentující kořen podstromu, který bude analyzován. Jelikož některé důležité informace (jako jméno návratové hodnoty) se vyskytují v AST dříve, není tento vrchol brán jako ohraničení zpracovávaných AST podstromů.

Tento vrchol sám o sobě neobsahuje pro Expressions modul zajímavé informace, ale je možno jej namapovat na vrchol grafu. Proto se používá se jako hlavní vrchol ukládající data pro druhou fázi fungování Expressions modulu z vrcholů, které pouze skládají podvýraz, ale samy jej na vrcholy grafu nemapují.

Jak je vidět ze základního příkladu na začátku této kapitoly, tak AST vrchol `<1,8> AST_EXPR <OracleAstNode>` je mapován na grafový vrchol `<1,8>A * B [PLSQL Expression] (AST_EXPR.sql)`.

7.8.2 Návratová hodnota

Návratová hodnota z SQL `SELECT`u je reprezentována AST vrcholem `AST_RESULTSET_COL`, který je v Mantě implementován pomocí datového typu `IAstResultSetColumn`.

Z vrcholu je jméno vraceného atributu načteno do zásobníku jmen a zpracování AST pokračuje standardním průchodem do hloubky pomocí visitoru.

Jméno vraceného atributu je také vráceno jako token popisu podvýrazu ke složení ve zpracování některého předka. Tato hodnota je využita pouze pokud se jednalo o skalární `SELECT`, kdy výsledná hodnota výsledku byla použita v rámci nadřazeného výrazu.

Jak je vidět ze základního příkladu na začátku této kapitoly, tak pro návratovou hodnotu ze `SELECT`u vzniká AST vrchol `<1,8> AST_RESULTSET_COL <AstResultSetColumn>`.

7.8.3 Operátor

Operátory mezi jednotlivými SQL výrazy jsou reprezentovány AST vrcholem `AST_OPERATOR`, který je v Mantě implementován pomocí datového typu `IOracleAstNode`.

Jedná se o základní operátory, jako jsou například + nebo $*$.

Pro každý operátor je určena precedence (viz kapitola číslo 7.5). Protože však unární operátory (+ a −) mají stejné označení jako jejich binární verze, je nejprve nutné z AST určit, jestli se jedná o unární nebo binární verzi příslušného operátoru.

Pro tento typ AST vrcholu není dohledáván grafový vrchol, ale textová reprezentace operátoru se vrací jako token popisu podvýrazu ke složení ve zpracování některého předka.

Jak je vidět ze základního příkladu na začátku této kapitoly, tak pro AST vrchol `<1,10> AST_OPERATOR <OracleAstNode>` neexistuje odpovídající grafový vrchol.

7.8.4 Závorky

Uzávorkovaná část SQL výrazu je reprezentována AST vrcholem `AST_BRACKET` který je v Mantě implementován pomocí datového typu `IOracleAstNode`.

Pro tento typ AST vrcholu se skládá popis z textových tokenů závorek „()“, generovaných dočasných proměnných a případně textových tokenů, oddělovacích čárek (Oracle SQL může v závorce obsahovat seznam podvýrazů). Jména jednotlivých dočasných proměnných jsou po dobu vyhodnocování jejich příslušných podstromů standardním visitorem přidávána na vrchol zásobníku jmen.

Výsledný popis je vkládán k vrcholu grafu pod nově vygenerovanou dočasnou proměnnou, která je také vrácena jako návratová hodnota zpracování, aby bylo možno při skládání v některém z předků identifikovat generovaný popis a zajistit provázání.

Příklad Oracle SQL skriptu obsahující závorky:

```
SELECT ('a', 'b')
FROM DUAL
;
```

Část exportu AST vygenerovaného z předchozího Oracle SQL skriptu. Pro potřeby vyhodnocení těchto závorek jsou podstatné následující AST vrcholy:

<1,8> AST_BRACKET zpracovávaný vrchol, obaluje blok uzavřený v závorkách

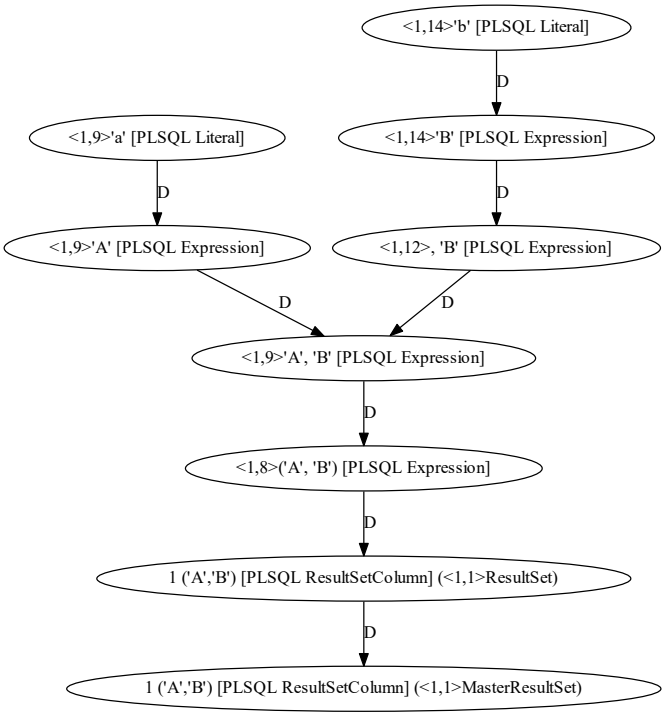
<1,8> (tento vrchol v rámci zpracování je tento vrchol vynechán, protože je dán gramatikou závorek a text „()“ je generován přímo

<1,9> AST_EXPR_LIST obalový vrchol pro případ listu parametrů, pokud by závorky byly okolo vnitřního skalárního `SELECT`u pak by tento vrchol nevznikl (jedná se spíše o okrajový případ)

<1,9> AST_EXPR_LIST_ITEM všechny podvýrazy ze seznamu v závorkách jsou obaleny tímto vrcholem

```
...
<1,8> AST_EXPR <OracleAstNode>
  <1,8> AST_BRACKET <OracleAstNode>
    <1,8> ( <OracleAstNode>
    <1,9> AST_EXPR_LIST <OracleAstNode>
      <1,9> AST_EXPR_LIST_ITEM <OracleAstNode>
        <1,9>  <OracleAstNode>
        <1,9> AST_EXPR <OracleAstNode>
          <1,9> AST_CHAR_LITERAL <AstCharLiteral>
            <1,9> 'a' <OracleAstNode>
      <1,12> AST_EXPR_LIST_ITEM <OracleAstNode>
        <1,12> , <OracleAstNode>
        <1,14> AST_EXPR <OracleAstNode>
          <1,14> AST_CHAR_LITERAL <AstCharLiteral>
            <1,14> 'b' <OracleAstNode>
    <1,17> ) <OracleAstNode>
...
```

V grafu datových toků vygenerovaném z předchozího Oracle SQL skriptu jsou vidět jednotlivé vstupy do listu prvků v závorkách, ale nejsou zde patrné vrcholy na které by bylo možné popis zapsat. Proto je výsledný popis pouze vracen k uložení v nadřazeném `AST_EXPR` vrcholu.



Obrázek 7.2: Graf datových toků při použití závorek.

7.8.5 Časový výraz

Časový výraz Oracle SQL je reprezentován AST vrcholem `AST_DATETIME_EXPR`, který je v Mantě implementován pomocí datového typu `IOracleAstNode`.

V rámci vývoje bylo nutno tomuto typu vrcholu věnovat speciální pozornost, protože využívá klíčová slova se standardně nevracela do výsledného popisu. Po rozšíření zpracování jednotlivých klíčových slov Oracle SQL (viz kapitola číslo 7.8.7) již nebylo potřeba tento vrchol ošetřovat zvláštním způsobem a vyhodnocuje se tedy jako ostatní nespecifikované vrcholy (viz kapitola číslo 7.8.15). Jediným rozdílem zůstalo jeho specifické zalogování pro potřeby dalšího rozvoje nebo hledání chyb.

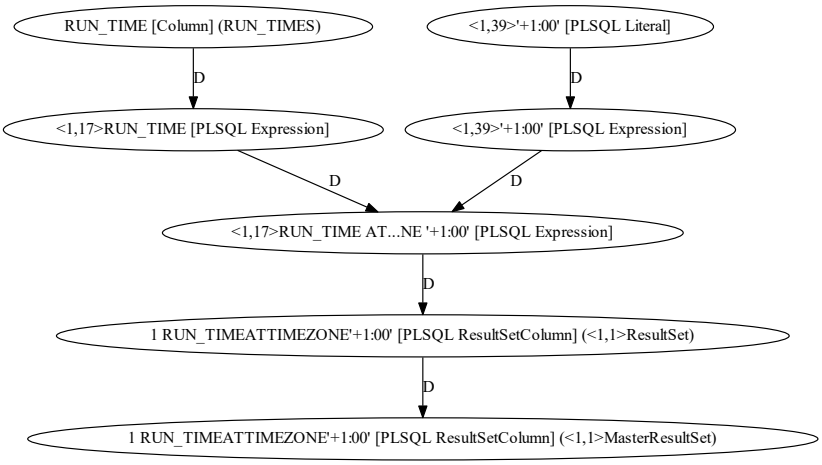
Příklad Oracle SQL skriptu obsahující časový výraz:

```
SELECT run_time AT TIME ZONE '+1:00'
FROM run_times
;
```

Příklad části exportu AST vygenerovaného z předchozího Oracle SQL skriptu, kde je vidět výskyt `<1,8> AST_DATETIME_EXPR <OracleAstNode>`:

```
<1,8> AST_EXPR <OracleAstNode>
  <1,8> AST_DATETIME_EXPR <OracleAstNode>
    <1,8> AST_AGG_REF <AstAggRef> - ...
      <1,8> AST_NAME_SEGMENT <AstNameSegment> - ...
```

```
      <1,8>  <OracleAstNode>
      <1,8> run_time <OracleAstNode>
    <1,17> AT <OracleAstNode>
    <1,20> TIME <OracleAstNode>
    <1,25> ZONE <OracleAstNode>
    <1,30> AST_CHAR_LITERAL <AstCharLiteral>
      <1,30> '+1:00' <OracleAstNode>
```



Obrázek 7.3: Graf datových toků pro časový výraz.

7.8.6 Speciální konstanty

Speciální konstanty Oracle SQL (např.: `NULL`, `SYSDATE`, ...) jsou reprezentovány AST vrcholem `AST_SPEC_CONST`, který je v Mantě implementován pomocí datového typu `IOracleAstNode`.

Z pohledu popisu výrazu je lze chápat jako speciální případ literálu (viz kapitola číslo 7.8.9). Zpracování je stejné až na zalogovanou zprávu z důvodu případného zkoumání běhu Expressions modulu.

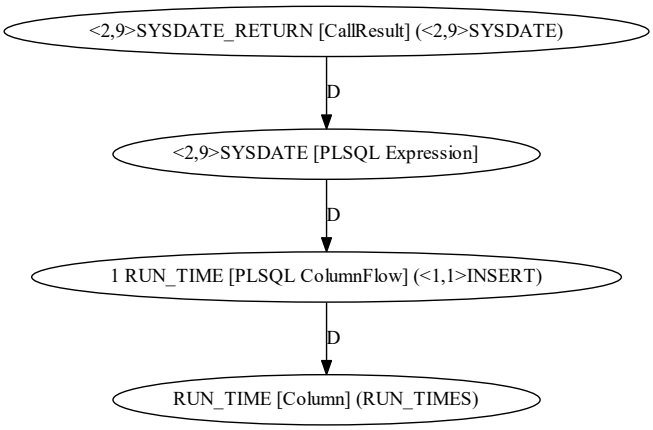
Příklad Oracle SQL skriptu obsahující `SYSDATE`, jako příklad speciální konstanty Oracle SQL:

```
INSERT INTO run_times(run_time)
VALUES (sysdate)
;
```

Z části exportu AST vygenerované z předchozího Oracle SQL skriptu je vidět, že AST vypadá obdobně jako pro volání bezparametrické funkce. To že není podstrom tohoto vrcholu již dál vyhodnocován, nepředstavuje žádnou komplikaci, protože pro Expressions modul je podstatné pouze pojmenování konstanty (obdobně jako u literálů).

```
<2,9> AST_EXPR <OracleAstNode>
  <2,9> AST_SPEC_CONST <OracleAstNode>
    <2,9> AST_AGG_REF <AstAggRef> - ...
```

```
<2,9> AST_NAME_SEGMENT <AstNameSegment> - ...
  <2,9>  <OracleAstNode>
  <2,9> sysdate <OracleAstNode>
<2,9> AST_CALL_SEGMENT <AstCallSegment> - ...
  <2,9> AST_ARGUMENTS <OracleAstNode>
    <2,9>  <OracleAstNode>
```



Obrázek 7.4: Graf datových toků pro speciální konstanty.

7.8.7 Klíčová slova

Jedná se o skupinu typů vrcholů AST reprezentující literály klíčových slov.

Klíčová slova pro Oracle dialekt SQL se v AST vyskytují ve třech typech vrcholů podle jména jejich tokenu:

specifické tokeny Jedná se o rezervovaná slova pro Oracle SQL (například `DISTINCT`, viz Literatura [8]).

V AST jsou reprezentovány svými vlastními typy tokenů podle textace rezervovaného slova (například `DISTINCT`), které jsou v Mantě implementovány pomocí datového typu `IOracleAstNode`.

REGULAR_ID Jedná se o skupinu nerezervovaných slov pro Oracle SQL, pro které Manta negeneruje z historických důvodů speciální typ tokenu (například `ZONE`).

Klíčová slova která nejsou gramatikou určena jako rezervovaná, mohou vystupovat jako klíčová slova i jako klasické identifikátory. Je tedy nutné rozlišit v jaké roli jsou v aktuálně zpracovávaném vrcholu použity.

V AST jsou reprezentovány vrcholem `AST_SPEC_CONST`, který je v Mantě implementován pomocí datového typu `IOracleAstNode`.

KW_* Jedná se o skupinu nerezervovaných slov pro Oracle SQL, pro které již má Manta speciální typ tokenu (například `TIME`).

Stejně jako v případě `REGULAR_ID` je nutno rozlišovat jejich roli použití.

V AST jsou reprezentovány svými vlastními typy tokenů podle textace klíčového slova s prefixem „KW_“ (například KW_TIME), které jsou v Mantě implementovány pomocí datového typu IOracleAstNode.

Textaci klíčových slov je potřeba sbírat z důvodu jejich propsání do výsledného popisu. Klíčová slova mají svůj význam a při jejich vynechání by popis ztratil význam nebo měl jiný než v původním SQL skriptu.

Klíčová slova jsou ve výsledném popisu transformace pouze vkládány obdobně jako literály a v grafu datových toků neexistuje vrchol, na který by bylo možno jejich popis propsat. Proto je jejich textová hodnota vrácena jako návratová hodnota zpracování, ke složení do popisu v některém z předků.

Ukázkou jejich reprezentace jsou například z kapitoly číslo 7.8.5: <1,25> ZONE <OracleAstNode> pro REGULAR_ID a <1,20> TIME <OracleAstNode> pro KW_*.

7.8.8 Pojmenované prvky SQL

Libovolný pojmenovaný prvek SQL skriptu (může se jednat o jména proměnných, sloupců tabulek, funkcí, ...) je reprezentován AST vrcholem AST_AGG_REF, který je v Mantě implementován pomocí datového typu IAstAggRef.

Určené jméno je přidáno na vrchol zásobníku jmen po dobu standardního zpracování podstromu AST. Standardně tedy tento typ AST vrcholů negeneruje novou informaci pro žádný grafový vrchol.

Při určování jména je nutné rozlišovat, jestli pro konkrétní typ AST vrcholu je zapotřebí generovat kvalifikované jméno prvku, na který odkazuje nebo nové unikátní jméno dočasné lokální proměnné.

Kvalifikovaná jména se generují pro jména sloupců, proměnných a vstupních parametrů příslušného bloku skriptu. Používají se ještě pro jména neatomických datových typů, u kterých je ovšem nutné navíc přidat na odpovídající grafový vrchol popis identity, která bude při inliningu odstraněna, ale umožní získání jména, které by jinak nebylo již dostupné. Pro ostatní typy je generováno unikátní jméno dočasné lokální proměnné.

Příklad jména sloupce je vidět ze základního příkladu na začátku této kapitoly, kde <1,8> AST_AGG_REF <AstAggRef> reprezentuje obalový AST vrchol pro sloupec TBL.A.

7.8.9 Literály

Literály použité v SQL skriptu (může se jednat například o textovou konstantu „abc“ nebo číselnou konstantu „5“) jsou reprezentovány jedním z AST vrcholů AST_*_LITERAL které jsou v Mantě implementovány pomocí datového typu IAstLiteral.

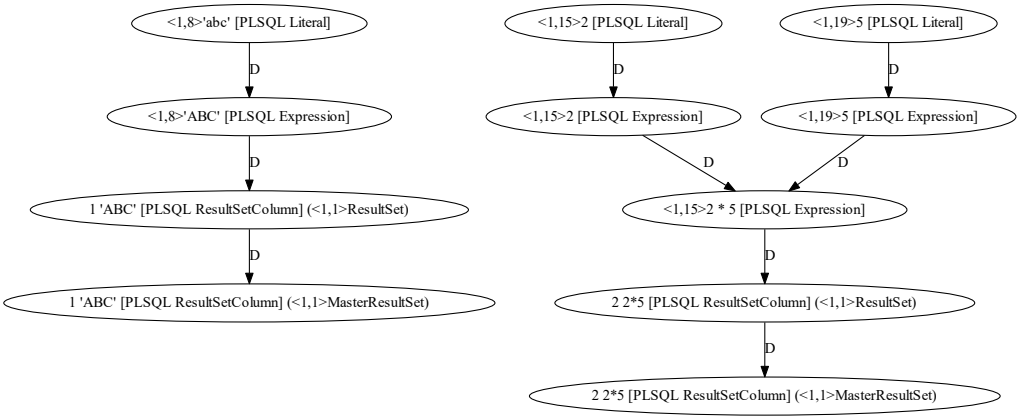
Pro jednotlivé literály je možné v grafu datových toků nalézt odpovídající vrchol reprezentující nový vstup do datového toku. Při jejich zpracování jejich hodnotu nevkládáme ke grafovému vrcholu, ale je vrácena jako návratová hodnota zpracování, ke složení do popisu v některém z předků. Důvodem je že literály vystupují jako součást podstromu vrcholu AST_EXPR (viz kapitola číslo 7.8.1), který zajistí jejich uložení k vrcholu grafu datových toků. Typicky se také vyskytují jako součást nějakého výrazu, který očekává návrat popisu jednotlivých podčástí pro jejich složení.

Příklad Oracle SQL skriptu obsahující textový i číselný literál:

```
SELECT 'abc', 2 * 5
FROM DUAL
;
```

Z části exportu AST vygenerované z předchozího Oracle SQL skriptu, je vidět že textové literály jsou identifikovány jako `AST_CHAR_LITERAL`, zatím co číselné literály jako `AST_NUM_LITERAL`:

```
<1,8> AST_SELECT_LIST <OracleAstNode>
  <1,8> AST_SELECT_ITEM <OracleAstNode>
    <1,8>  <OracleAstNode>
    <1,8> AST_RESULTSET_COL <AstResultSetColumn> - ...
      <1,8> AST_EXPR <OracleAstNode>
        <1,8> AST_CHAR_LITERAL <AstCharLiteral>
          <1,8> 'abc' <OracleAstNode>
  <1,13> AST_SELECT_ITEM <OracleAstNode>
    <1,13> , <OracleAstNode>
    <1,15> AST_RESULTSET_COL <AstResultSetColumn> - ...
      <1,15> AST_EXPR <OracleAstNode>
        <1,15> AST_OPERATION <AstOperation>
          <1,15> AST_NUM_LITERAL <AstNumericLiteral>
            <1,15> 2 <OracleAstNode>
          <1,17> AST_OPERATOR <OracleAstNode>
            <1,17> * <OracleAstNode>
          <1,19> AST_NUM_LITERAL <AstNumericLiteral>
            <1,19> 5 <OracleAstNode>
```



Obrázek 7.5: Graf datových toků pro literály.

7.8.10 Výsledek SQL dotazu

Výstup ze `SELECT`u nebo množinové operace implementované v SQL jazyce (jako je například `UNION` nebo `INTERSECT`) nad vícero `SELECT`y je reprezentován AST vrcholem `AST_MASTER_SELECT`, který je v Mantě implementován pomocí datového typu `IAstMasterSelect`.

Pokud tento AST vrchol obaluje pouze jediný SELECT, pak není potřeba jej nijak speciálně vyhodnocovat, protože o jeho vyhodnocení se postará zpracování ostatních AST vrcholů. Pokud ovšem obsahuje množinovou operaci, pak je nutné vygenerovat popis této operace spolu s odkazy na jednotlivé skládané SELECTy a uložit jej k příslušnému vrcholu v grafu datových toků.

Popis množinové operace se skládá z generovaných unikátních jmen dočasných lokálních proměnných, které reprezentují jednotlivé skládané SELECTy, mezi kterými je příslušná množinová operace. Z důvodu čitelnosti výsledného popisu transformace je zakázán inlining do tohoto popisu (nastaven atribut `inlineType`).

Pro jednotlivé skládané SELECTy je dále nutné nastavit odkazy na vrcholy grafu datových toků, kde má být pokračováno s jejich vyhodnocováním. Jinak by byly jednotlivé SELECTy vyhodnocovány nezávisle na jejich zasazení do množinové operace a nedošlo by k provázání na vygenerované jméno dočasné proměnné, reprezentující je v popisu množinové operace.

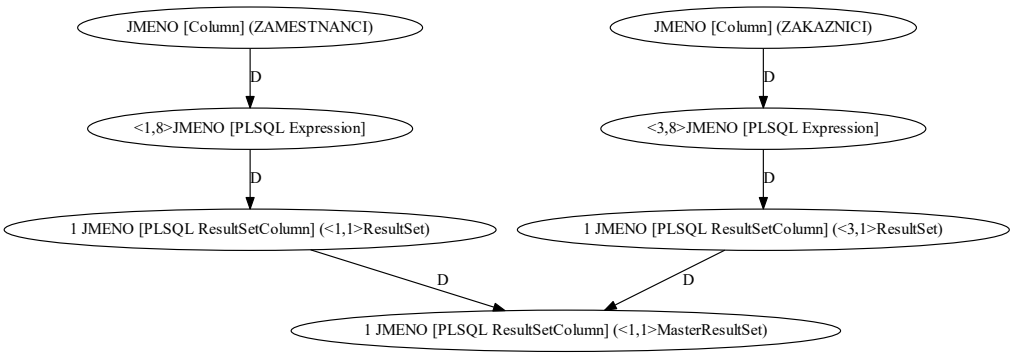
Posledním příznakem, který je nutné nastavit je `propagateThroughGraph`, protože tento AST vrchol může reprezentovat operaci mající vliv na transformaci více různých atributů. Z AST je komplikované určit o které všechny atributy se jedná, ale v grafu datových toků, mají všechny jejich odpovídající vrcholy společného předka. Proto je popis z tohoto AST vrcholu zapsán do grafu datových toků na vrchol společného předka, odkud bude v druhé fázi zpracování přenesen do příslušných vrcholů jednotlivých transformací (viz. kapitola číslo 8.2). V následujícím příkladu je toto vidět na tom, že jeden UNION se uplatňuje nezávisle (vzhledem k Data Lineage a řešeným popisu datových toků) na skládání atributů JMENO i NAROZENI.

Příklad Oracle SQL skriptu obsahující množinovou operaci sloučení:

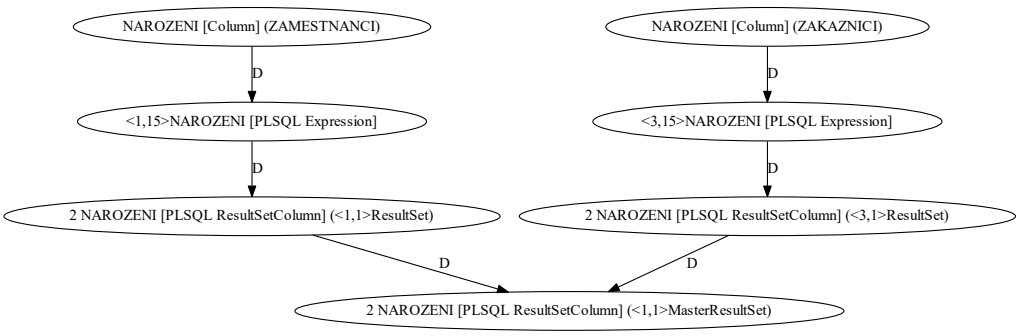
```
SELECT jmeno, narozeni FROM zamestnanci
UNION
SELECT jmeno, narozeni FROM zakaznici
;
```

Z části exportu AST vygenerované z předchozího Oracle SQL skriptu, je vidět že AST_MASTER_SELECT je obalový vrchol pro jednotlivé AST_SELECT_UNIT mezi kterými jsou spojující AST_SET_OPERATOR:

```
<1,1> AST_MASTER_SELECT <AstMasterSelect> - ...
  <1,1> AST_SELECT_UNIT <OracleAstNode>
    <1,1> AST_SELECT <AstSelect> - ...
      ...
  <2,1> AST_SET_OPERATOR <OracleAstNode>
    <2,1> UNION <OracleAstNode>
  <3,1> AST_SELECT_UNIT <OracleAstNode>
    <3,1> AST_SELECT <AstSelect> - ...
      ...
```



Obrázek 7.6: Graf datových toků pro množinové operace pro sloupec JMENO.



Obrázek 7.7: Graf datových toků pro množinové operace pro sloupec NAROZENI.

Výsledný popis generovaný z předchozího Oracle SQL skriptu, ze kterého je vidět, že generovaná dočasná proměnná `VAR_0` je zpropagována a tedy duplikována pro popisy obou atributů `JMENO` i `NAROZENI`:

```
1 JMENO := VAR_0 UNION VAR_1
  VAR_0 := ZAMESTNANCI.JMENO
  VAR_1 := ZAKAZNICI.JMENO
-----
2 NAROZENI := VAR_0 UNION VAR_1
  VAR_0 := ZAMESTNANCI.NAROZENI
  VAR_1 := ZAKAZNICI.NAROZENI
```

7.8.11 Přiřazení

Přiřazení hodnoty do proměnné v rámci PLSQL skriptu je reprezentováno AST vrcholem `AST_ASSIGNMENT_CMD`, který je v Mantě implementován pomocí datového typu `IAstAssignmentCmd`.

Pro potřeby získávání popisů jednotlivých transformací pro nástroj Manta není požadováno řešení samotného Data Lineage, nýbrž pouze popisy jednotli-

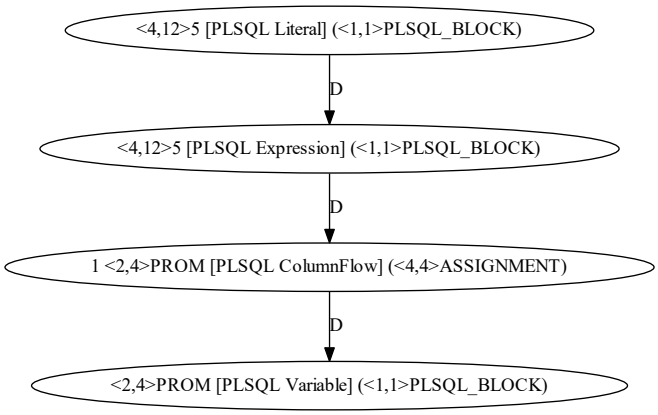
vých transformací. Z tohoto důvodu nejsou v Expressions modulu záměrně vyhodnocovány cíle přiřazení. Tento typ AST vrcholu je proto zpracováván pouze z důvodu modifikace standardního zpracování vrcholů (viz kapitola číslo 7.8.15). Oproti němuž nevyhodnocuje veškeré podstromy zpracovávaného vrcholu, ale pokračuje ve zpracování pouze pro „pravou“/výrazovou stranu výrazu.

Příklad Oracle PLSQL skriptu obsahující přiřazení:

```
DECLARE
   prom NUMBER;
BEGIN
   prom := 5;
END;
```

Část exportu AST vygenerovaná z předchozího Oracle SQL skriptu:

```
<4,4> AST_ASSIGNMENT_CMD <AstAssignmentCmd>
  <4,4> AST_AGG_REF <AstAggRef> - ...
    <4,4> AST_NAME_SEGMENT <AstNameSegment> - ...
      <4,4>  <OracleAstNode>
      <4,4> prom <OracleAstNode>
  <4,9> := <OracleAstNode>
  <4,12> AST_EXPR <OracleAstNode>
    <4,12> AST_NUM_LITERAL <AstNumericLiteral>
      <4,12> 5 <OracleAstNode>
```



Obrázek 7.8: Graf datových toků pro přiřazení.

7.8.12 Volání funkcí a procedur

Volání funkce nebo procedury z SQL skriptu je reprezentováno AST vrcholem `AST_CALL_SEGMENT`, který je v Mantě implementován pomocí datového typu `IAstCallSegment`.

Tento typ AST vrcholu je použit také pro přístup k položkám polí. Manta ovšem pracuje s polem jako s celkem. Hodnota indexu z toho důvodu není záměrně vyhodnocována a zpracování je ukončeno. Druhým případem zrychleného

ukončení zpracování je případ, kdy volaná funkce/procedura nemá žádnou návratovou hodnotu. Pak je zpracován AST podstrom tohoto vrcholu, ale bez ostatního specifického zpracování, které se uplatňuje pro standardní volání funkcí a procedur.

Zpracování nejtypičtějšího volání funkce/procedury kvůli kterému je primárně tento typ AST vrcholu zpracováván, má tři hlavní úlohy:

1. zjištění jména funkce/procedury — Jméno není součástí zpracovávaného AST vrcholu, ale je nutné jej načíst z jeho předchozího sourozence.

2. vyhodnocení jednotlivých parametrů — Každému parametru je vygenerováno unikátní jméno dočasné lokální proměnné a každý je vyhodnocován standardním způsobem jako samostatný podvýraz. Jediným rozdílem je, že zpracování je určeno, na který vrchol grafu datových toků má být uložen výsledný popis, aby bylo zajištěno korektní provázání na požadovaný parametr.

3. odkazy na jednotlivé parametry — Graf datových toků generovaný v Mantě při volání funkce neodkazuje na jednotlivé parametry, pouze na referenci na její návratovou hodnotu. Proto je nutné připojit reference na podgrafy reprezentující vyhodnocení jednotlivých parametrů.

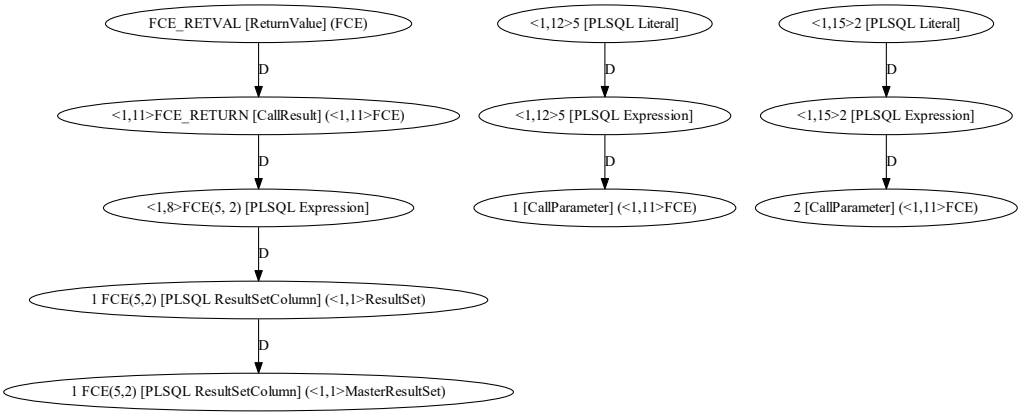
Výsledný popis složený ze jména funkce/procedury a výčtu parametrů je spolu s odkazy na podvýrazy jednotlivých parametrů uložen na příslušný vrchol grafu datových toků.

Příklad Oracle SQL skriptu obsahující volání funkce `fce`:

```
SELECT fce(5, 2)
FROM DUAL
;
```

Část exportu AST vygenerovaná z předchozího Oracle SQL skriptu:

```
<1,8> AST_EXPR <OracleAstNode>
  <1,8> AST_AGG_REF <AstAggRef> - ...
    <1,8> AST_NAME_SEGMENT <AstNameSegment> - ...
      <1,8>  <OracleAstNode>
      <1,8> fce <OracleAstNode>
    <1,11> AST_CALL_SEGMENT <AstCallSegment> - ...
      <1,11> AST_ARGUMENTS <OracleAstNode>
        <1,11> ( <OracleAstNode>
        <1,12> AST_ARGUMENT <AstArgument>
          <1,12>  <OracleAstNode>
          <1,12> AST_EXPR <OracleAstNode>
            <1,12> AST_NUM_LITERAL <AstNumericLiteral>
              <1,12> 5 <OracleAstNode>
        <1,13> AST_ARGUMENT <AstArgument>
          <1,13> , <OracleAstNode>
          <1,15> AST_EXPR <OracleAstNode>
            <1,15> AST_NUM_LITERAL <AstNumericLiteral>
              <1,15> 2 <OracleAstNode>
        <1,16> ) <OracleAstNode>
```



Obrázek 7.9: Graf datových toků pro volání funkce.

7.8.13 Přetypování

Přetypování atributu nebo proměnné na jiný datový typ je reprezentováno AST vrcholem `AST_CAST_SEGMENT`, který je v Mantě implementován pomocí datového typu `IAstCastSegment`.

Zpracování tohoto typu AST vrcholu má obdobně jako u AST vrcholu pro volání funkce tři hlavní úkoly (viz kapitola číslo 7.8.12).

Vyhodnocení parametrů a k nim příslušné odkazy v grafu datových toků se vyhodnocují stejně jako u parametrů volání funkce. Rozdíl je, že u přetypování je zapotřebí určit cílový datový typ, na místo určení jména funkce/procedury.

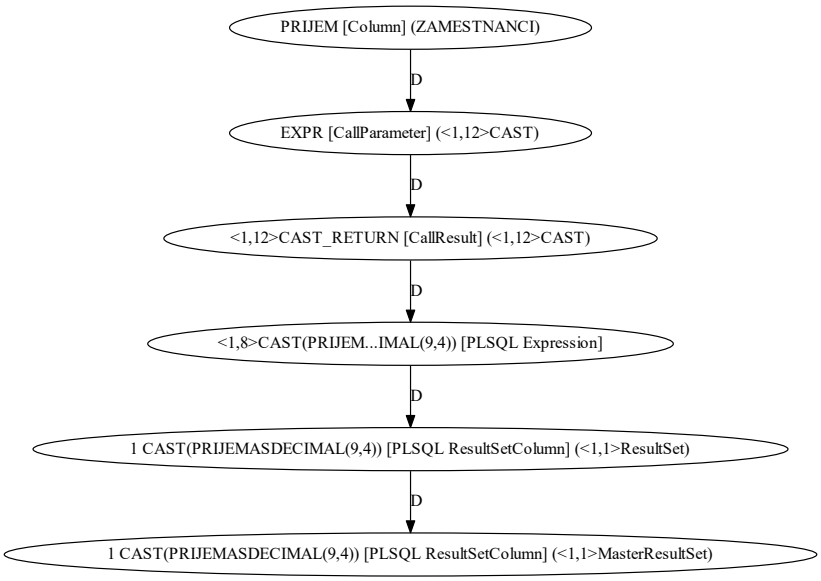
Příklad Oracle SQL skriptu obsahující přetypování:

```
SELECT CAST(prijem AS DECIMAL(9,4))
FROM zamestnanci
;
```

Část exportu AST vygenerovaná z předchozího Oracle SQL skriptu:

```
<1,8> AST_EXPR <OracleAstNode>
  <1,8> AST_AGG_REF <AstAggRef> - ...
    <1,8> AST_NAME_SEGMENT <AstNameSegment> - ...
      <1,8>  <OracleAstNode>
      <1,8> CAST <OracleAstNode>
    <1,12> AST_CAST_SEGMENT <AstCastSegment> - ...
      <1,12> AST_ARGUMENTS <OracleAstNode>
        <1,12> ( <OracleAstNode>
        <1,13> AST_ARGUMENT <AstArgument>
          <1,13>  <OracleAstNode>
          <1,13> AST_EXPR <OracleAstNode>
            <1,13> AST_AGG_REF <AstAggRef> - ...
              <1,13> AST_NAME_SEGMENT <AstNameSegment> - ...
                <1,13>  <OracleAstNode>
                <1,13> prijem <OracleAstNode>
        <1,20> AS <OracleAstNode>
        <1,23> AST_DATA_TYPE <OracleAstNode>
```

```
          <1,23> AST_BUILTIN_TYPE <AstBuiltInType>
            <1,23> DECIMAL <OracleAstNode>
            <1,30> ( <OracleAstNode>
            <1,31> 9 <OracleAstNode>
            <1,32> , <OracleAstNode>
            <1,33> 4 <OracleAstNode>
            <1,34> ) <OracleAstNode>
        <1,35> ) <OracleAstNode>
```



Obrázek 7.10: Graf datových toků pro přetypování.

7.8.14 SQL CASE výraz

Manta AST rozlišuje dva různé typy `CASE` výrazů a k nim příslušících vrcholů:

základní Pro vyhledávání hodnot odpovídajících zadanému parametru.

V AST je reprezentováno vrcholem `AST_CASE_MATCHED`, který je v Mantě implementován pomocí datového typu `IAstCaseMatched`.

vyhledávaný Pro posloupnost podmínek, kde je výsledek dán první pravdivou podmínkou.

V AST je reprezentováno vrcholem `AST_CASE_SEARCHED`, který je v Mantě implementován pomocí datového typu `IAstCaseSearched`.

Z pohledu vyhodnocování popisu v Expressions modulu se oba typy liší pouze přítomností úvodního výrazu pro základní verzi `CASE`. Je tedy možné oba typy zpracovávat stejným způsobem.

Ze své podstaty se `CASE` SQL výraz skládá z několika podvýrazů, které lze vyhodnocovat stejným způsobem nezávisle na sobě. Při vzniku popisu postupujeme tak, že jednotlivé podvýrazy nahradíme vygenerovanými dočasnými unikátními

proměnnými, ze kterých sestavíme popis operace a uložíme odkaz na jejich vyhodnocení. Tímto zajistíme rozložení jinak komplexního a potenciálně dlouhého a nepřehledného popisu do menších bloků, které budou později ve druhé fázi zpracování podléhat opětovnému inliningu. Tímto způsobem je implementován speciální formátovací požadavky popsaný v kapitole číslo 5.4.2.

Na rozdíl od zpracování většiny ostatních typů AST vrcholů, je zde nutno jednotlivé podvýrazy dohledávat v AST. Gramatika, kterou je ovšem AST generováno definuje pevnou strukturu, kde je možné jednotlivé podvýrazy nalézt a je možné přistupovat na konkrétní předem známé podvrcholy.

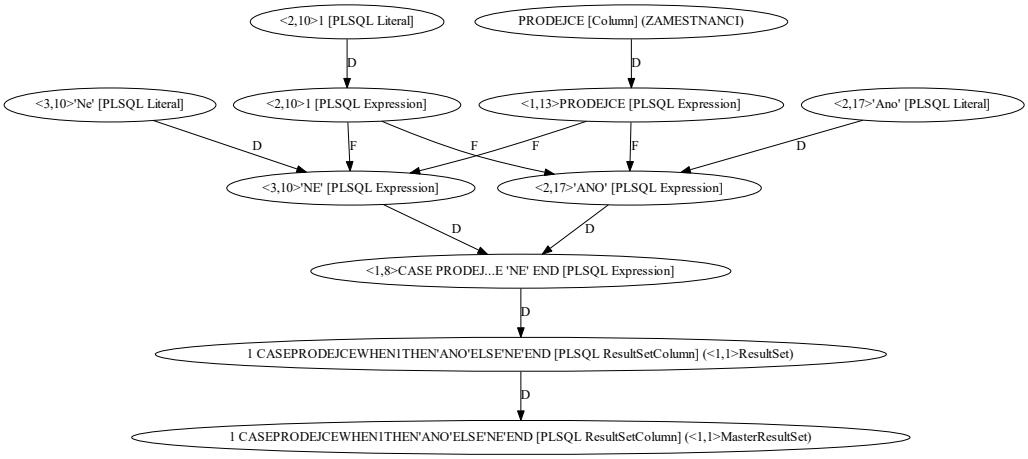
Odkazy na vrcholy grafu datových toků s popisy jednotlivých podvýrazů (reprezentující vyhledávaný parametr, podmínky a k nim náležící hodnoty) je nutné uložit, protože ve výsledném grafu již tyto vazby nejsou jasně dané a nebylo by jak složit korektní výsledný popis.

Příklad Oracle SQL skriptu obsahující základní CASE výraz:

```
SELECT CASE prodejce
    WHEN 1 THEN 'Ano'
    ELSE 'Ne'
  END
FROM zamestnanci
;
```

Část exportu AST vygenerovaná z předchozího Oracle SQL skriptu:

```
<1,8> AST_CASE_MATCHED <AstCaseMatched>
  <1,8> CASE <OracleAstNode>
  <1,13> AST_EXPR <OracleAstNode>
    <1,13> AST_AGG_REF <AstAggRef> - ...
      <1,13> AST_NAME_SEGMENT <AstNameSegment> - ...
        <1,13>  <OracleAstNode>
        <1,13> prodejce <OracleAstNode>
  <2,5> AST_CASE_ITEMS <OracleAstNode>
    <2,5> AST_CASE_ITEM <OracleAstNode>
      <2,5> AST_CASE_WHEN <OracleAstNode>
        <2,5> WHEN <OracleAstNode>
        <2,10> AST_EXPR <OracleAstNode>
          <2,10> AST_NUM_LITERAL <AstNumericLiteral>
            <2,10> 1 <OracleAstNode>
      <2,12> AST_CASE_THEN <OracleAstNode>
        <2,12> THEN <OracleAstNode>
        <2,17> AST_EXPR <OracleAstNode>
          <2,17> AST_CHAR_LITERAL <AstCharLiteral>
            <2,17> 'Ano' <OracleAstNode>
  <3,5> AST_CASE_ELSE <OracleAstNode>
    <3,5> ELSE <OracleAstNode>
    <3,10> AST_EXPR <OracleAstNode>
      <3,10> AST_CHAR_LITERAL <AstCharLiteral>
        <3,10> 'Ne' <OracleAstNode>
  <4,3> END <OracleAstNode>
```



Obrázek 7.11: Graf datových toků pro CASE výraz.

7.8.15 Ostatní typy AST vrcholů

Pro ostatní typy vrcholů není potřeba provádět žádné speciální vyhodnocení. Pouze spojíme popisy získané z jednotlivých AST podstromů zpracovávaného vrcholu a vrátíme je.

Při skládání jednotlivých podpopisů je nutné pouze určit výslednou precedenci celého složeného popisu. Tu určíme jako minimální precedenci ze skládaných podpopisů, protože při složení má celý výraz precedenci danou operátorem s minimální precedencí. Z důvodu reprezentace precedence (viz kapitola číslo 7.5) se při skládání výsledná hodnota určí jako maximální hodnota precedencí všech podpopisů.

8. Druhá fáze zpracování

Třída `ExpressionsTask` implementuje druhou fázi zpracování v Expressions modulu. Vstupem je graf datových toků generovaný v Mantě a popisy jednotlivých podvýrazů mapované na jednotlivé vrcholy tohoto grafu vznikající v první fázi zpracování. Na výstupu jsou popisy transformací pro jednotlivé atributy vyskytující se ve vstupním SQL, uložené do příslušných vrcholů grafu datových toků.

8.1 Princip fungování

Druhá fáze zpracování funguje v několika po sobě jdoucích krocích. Rozdělení vyplývá z logických a technických kroků, které je nutné udělat pro získání výsledného popisu transformace.

Rozdělením úlohy na menší části je možno kód lépe strukturovat, díky čemuž je kód srozumitelnější a tím i udržovatelnější a modifikovatelnější.

Následuje seznam kroků, ze kterých se skládá druhá fáze zpracování se základním popisem, co každý krok dělá (podrobný popis následuje vždy v samostatné kapitole):

1. Propagace popisu do podvrcholů grafu (viz kapitola číslo 8.2).

   Jedná se o technický krok daný specifiky grafu datových toků generovaným v Mantě. Jeho úkolem je přenést popisy z první fáze zpracování do korektních vrcholů grafu. V případech kde nebylo možno je již v první fázi uložit korektně.

2. Startovní uzly grafu (viz kapitola číslo 8.3).

   Z vrcholů grafu datových toků je nutné určit vrcholy, pro které se bude skládat popis transformace pro některý atribut. Tyto vrcholy slouží jako kořeny průchodu v následujícím kroku.

3. Průchod grafem a sběr podvýrazů (viz kapitola číslo 8.4).

   V tomto kroku je určen celkový popis transformace atributu tak, jak byl určen v první fázi zpracování z AST. Výsledný popis tedy může obsahovat značné množství generovaných dočasných proměnných, které mohou zhoršovat čitelnost výsledného popisu.

4. Inlining (viz kapitola číslo 8.5).

   Zlepšuje čitelnost výsledného popisu transformace tím, že podle definovaných pravidel vkládá jednotlivé podvýrazy do jejich místa použití, místo použití odkazu na ně.

5. Složení výsledného popisu transformace (viz kapitola číslo 8.6).

   Utvoří z jednotlivých podvýrazů výsledný textový popis transformace.

6. Uložení výsledku (viz kapitola číslo 8.7).

   Výsledný popis transformace získaný z předchozích kroků je zapsán do vrcholu grafu datových toků pro jeho uložení a následné zobrazení v Mantě nebo další zpracování.

8.2 Propagace popisu do podvrcholů grafu

V grafu datových toků generovaném Mantou mají jednotlivé vrcholy datového předka, který určuje jejich přesnější umístění (vazba není definována pomocí hran grafu).

Například pro `UNION` dvou `SELECT`ů je pro vrchol, reprezentující vracený sloupec (`ResultSetColumn`) jeho předkem `MasterResultSet` reprezentující sloučení výsledků obou `SELECT`ů prostřednictvím množinové operace `UNION`. Další předci určují přesnější umístění, kde byl tento výraz definován v závislosti na jeho umístění ve vstupním SQL a struktuře analyzované databáze.

Propagace se provádí plošně přes celý graf datových toků, v závislosti na nastavení příznaku `propagateThroughGraph` (viz kapitola číslo 7.6) na uložené struktuře popisu výrazu z první fáze zpracování. Tato propagace byla implementována obecně pro potřeby dalšího rozšiřování algoritmu, ale v průběhu implementace se ukázalo, že je aktuálně potřebná pouze pro zpracování množinových operací (viz kapitola číslo 7.8.10).

Propagace se provádí kopírováním popisu ze zpracovávaného vrcholu grafu do všech jeho datových potomků. Při kopírování je pouze nutné stejným způsobem aktualizovat případné reference na zdrojové grafové vrcholy (atribut `sources` viz kapitola číslo 7.6). Tedy původně odkazovaný vrchol nahradit za všechny jeho případné datové potomky.

Jinými slovy jsou při tomto procesu popisy v grafu datových toků propagovány o jednu úroveň níž podle vazby předek/potomek.

8.3 Startovní vrcholy grafu

Z první fáze zpracování jsou k jednotlivým vrcholům grafu datových toků uloženy popisy podvýrazů. Ty je nyní potřeba složit do výsledných popisů transformací pro jednotlivé atributy vyskytující se ve vstupním SQL skriptu. Skládání se provádí průchodem grafu, ale k tomu je nutné určit vrchol ze kterého bude průchod začat.

Výběr vrcholů je určen strukturou grafu datových toků generovaným Mantou a požadavky na atributy, pro které má být sestaven popis transformace. Protože jednotlivé typy vrcholů mohou vystupovat jako vstupy i výstupy, je nutné u všech vrcholů kontrolovat aby se jednalo o výstupy. Toto lze z grafu poznat tak že do vrcholu vede hrana z technického vrcholu `PLSQL ColumnFlow`.

Z důvodu různých požadavků systémů, na které může být Expressions modul aplikován, je umožněno konfiguračně (viz parametr `stopOnVariables` popsaný v kapitole číslo 6.4) měnit zda jsou proměnné v PLSQL blocích chápány jako startovní vrcholy.

Pro potřeby Manty byly jako startovní vrcholy v grafu datových toků zvoleny vrcholy reprezentující:

- sloupce tabulek
- návratové hodnoty z procedur/funkcí
- proměnné (v závislosti na konfiguraci, vždy nebo pouze v případě, kdy vystupuje jako nelokální výsledek datového toku)
- výsledný sloupec `SELECT`u, pokud není použit uvnitř jiného SQL příkazu (jedná se o samostatný `SELECT`, jehož výstup se dále nepoužívá)

8.4 Průchod grafem a sběr podvýrazů

V tomto kroku druhé fáze zpracování je procházen graf datových toků a jsou sbírány jednotlivé popisy podvýrazů, které zde byly připraveny v první fázi zpracování. Výsledkem je stromová struktura nesoucí v jednotlivých vrcholech části popisů posbírané z grafu.

Graf je procházen od cílového vrcholu určeného podle předchozího kroku zpracování (viz kapitola číslo 8.3). K průchodu je použit standardní DFS algoritmus (viz Literatura [9]), pro který jsou další vrcholy k průchodu z aktuálního vrcholu určovány:

1. Z odkazů v popisu aktuálního vrcholu.

   Jak již bylo popsáno v první fázi zpracování (viz kapitola číslo 7.8), některé popisy vyžadují ke svému korektnímu zpracování při průchodu grafem odkaz na jiný vrchol, než by byl určen hranami. Primárně se postupuje po těchto explicitně definovaných odkazech.

2. Z hran samotného grafu.

   Graf datových toků obsahuje dva druhy hran (datové a filtrační) a je orientovaný tak, že se postupuje v protisměru hran. Filtrační hrany odkazují na vrcholy grafu, které ve výrazu slouží jako podmínky (například celá sekce reprezentující SQL `WHERE` podmínku). Pro Data Lineage nejsou tyto sekce primárně podstatné a záměrně jsou proto filtrační hrany při průchodu grafem ignorovány.

Graf může obsahovat cykly a je proto nutné udržovat seznam již prozkoumaných vrcholů, protože další průchod by již nepřinesl žádné nové informace do výsledného popisu, ale došlo by k zacyklení algoritmu.

Pokračování v průchodu grafu pomocí DFS algoritmu je také ukončeno v případě pokud je nastaven parametr `stopOnVariables` (viz kapitola číslo 6.4) a aktuálně procházený vrchol reprezentuje proměnnou. Tím že je průchod grafu přerušen, dojde ve finálně vygenerovaných popisech k vygenerování samostatných popisů pro proměnné.

Důležitým úkolem průchodu grafem je také aktualizace cílů popisů získaných v první fázi zpracování. Z AST nejsou vždy přímo určitelná jména atributů do kterých bude výsledek výrazu uložen. Například určení parametrů `INSERT`u pořadím

v zápisu nebo zápis do proměnné v PLSQL z důvodu nevyhodnocování levé strany přiřazení (viz kapitola číslo 7.8.11).

8.5 Inlining

V tomto kroku jsou jednotlivé podvýrazy skládány do finálního popisu transformace.

Vstupem je strom popisů získaný z předchozího kroku (viz kapitola číslo 8.4), který je procházen pomocí standardního DFS algoritmu. Podvýrazy jsou skládány od listů procházeného stromu z důvodu dodržení pravidel formátování (viz kapitola číslo 5.4.1) a inline podmínek (atributy `precedence` a `inlineType` viz kapitola číslo 7.6).

Nejprve je nutné zkontrolovat zda více podvýrazů neurčuje stejný token (jinými slovy, zda některý token nemá několik možných definicí). Toto může nastat například v případě kdy se jedná o proměnnou, do které bylo provedeno v PLSQL skriptu několik přiřazení. Jelikož při statické analýze kódu se neví, kterou cestou se průchod ubíral, je nutné brát v potaz všechny možnosti. Výsledkem je, že pokud je některý token definován více podvýrazy, nemůže být inlinován (není jasné kterým z podvýrazů). Místo toho musí být zachovány všechny možnosti a uvedeny do výsledného popisu.

Při inliningu využíváme toho, že popis každého podvýrazu je uložen jako posloupnost tokenů, které rozlišují zda se jedná o proměnnou (kterou leze nahradit při inliningu) nebo o textový token (který nahradit nelze). Rozlišení na proměnné a textové tokeny vzniká již při prvotním sběru popisu z AST v první fázi zpracování Expressions modulu. Kromě rozlišení, které tokeny je možno nahradit, nám uložení popisu jako posloupnost tokenů umožňuje provádět inlining přímým nahrazením jednoho tokenu za posloupnost tokenů z podvýrazu, bez nutnosti vyhledávání v dlouhém textovém řetězci.

Podmínky inliningu, aby bylo možné nahradit token proměnné v popisu za popis z podvýrazů, který do této proměnné vede:

- Zpracovávaný výraz i podvýraz musí splňovat podmínku na možnost použití inliningu podle svých atributů `inlineType` (viz kapitola číslo 7.6), které byly nastaveny při jejich vzniku v první fázi zpracování.
- Musí být dodržena precedence, kdy zpracovávaný výraz nesmí mít vyšší precedenci než vkládaný podvýraz. Jinými slovy, ve výrazu sčítajícím dvě proměnné, je možné proměnnou nahradit pokud obsahuje násobení.
- Není povoleno inlinovat podvýraz pokud ve svém popisu obsahuje odřádkování. Toto pravidlo je dáno formátováním výsledného popisu, kde odřádkování primárně odděluje jednotlivé podvýrazy a takovýto inlining by mohl velmi snadno narušit formátování. Výjimku tvoří technický případ, kdy je zpracovávaný výraz tvořen pouze tokenem proměnné a bude tedy kompletně nahrazen.
- Výsledný popis nesmí po inliningu přesáhnout maximální definovanou délku textové reprezentace. Maximální povolená délka je definována parametrem

`maxInliningLength` (viz kapitola číslo 6.4). Výjimku tvoří případ, kdy inlining délku textové reprezentace zkrátí (může nastat, pokud je jméno proměnné delší než její popis), pak je povolen i pokud je výsledek stále příliš dlouhý.

Pokud došlo k inliningu tokenu některé proměnné, pak je nutno upravit vazby na zdrojové podvýrazy ve stromu podvýrazů. Na místo inlineného podvýrazu získáme všechny jeho podvýrazy. Jinými slovy, ze stromu byl odstraněn vrchol příslušného podvýrazu a je tedy místo hrany do něj nutné přepojit všechny hrany do vrcholů do kterých měl hranu on.

Příklad výstupu bez použití a s použitím inliningu, pro jednoduchý umělý příklad:

```
SELECT a * (b + c) AS d
FROM tbl
;
```

```
1 D := TBL.A * VAR_0
  VAR_0 := (VAR_1)
    VAR_1 := TBL.B + TBL.C
```

```
1 D := TBL.A * (TBL.B + TBL.C)
```

8.6 Složení výsledného popisu transformace

Předchozí kroky připravily finální popis ve stromové struktuře a nyní je potřeba pouze složit výsledný popis v textové podobě.

Pro uložení připravovaného výsledného popisu byla zvolena stromová struktura, protože odpovídá požadovanému formátu výsledného popisu transformace (viz kapitola číslo 5). Díky této volbě bylo možné provést převod implementačně jednoduchým průchodem stromu pomocí standardního DFS algoritmu, kde je aktuálně zpracovávaný vrchol vypsán před průchodem do dalších vrcholů. Jediná nutná úprava se týká odsazení popisu jednotlivých podvýrazů, kdy je vždy přidáno odsazení před všechny řádky získaného textového popisu podvýrazu vráceného z DFS průchodu.

8.7 Uložení výsledku

Výsledný popis transformace získaný z předchozích kroků, je nutné uložit do vrcholu grafu datových toků, aby byl dostupný pro standardní zobrazení v Mantě a dalších nástrojích.

Pro každý atribut je samostatný popis transformace, kterou je tvořen a každý je nutné uložit k příslušnému vrcholu grafu.

Popis nemůže být uložen přímo na vrchol reprezentující příslušný atribut (sloupec tabulky, ...), protože popis se vztahuje ke konkrétní transformaci, která je také reprezentována v grafu datových toků a následně v Mantě. Tento vrchol

je ovšem společný pro všechny transformace, kde se tento atribut vyskytuje, aby bylo možno provádět komplexní analýzu datových toků.

Výběr vrcholů je převážně ovlivněn principem fungování Filter Tasku, aby nedošlo k odfiltrování vrcholu na který byl popis uložen. Z principu jeho fungování, je pro uložení místo vrcholu odpovídajícího atributu vyhledán první vrchol typu `PLSQL ColumnFlow` pro zápis do atributu nebo `PLSQL ResultSetColumn` pro případ `SELECT`u bez zápisu výsledku, do kterého je výsledný popis zapsán.

9. Testování

Nedílnou součástí vývoje byla příprava testů. Testy se staly při vývoji natolik důležité, že nebyly využívány pouze ke kontrole výstupů, ale i v rámci standardního vývoje jako náhrada spouštění kompletní Manty. Bez jejich využití by byla implementace řádově pomalejší a náročnější.

9.1 Rozdělení implementace

Testy jsou primárně rozděleny na dvě části:

programová Není součástí výsledného Expressions modulu, ale slouží pouze pro účely spuštění vyvíjeného Expressions modulu v požadované konfiguraci Manty.

testovací scénáře Páry souborů propojené jmennou konvencí obsahující SQL skript scénáře a očekávaný výsledek, které jsou předkládány programové části ke zpracování.

Toto rozdělení umožnilo výrazně snazší přípravu nových testů, kdy bylo možno pouze nachystat nový pár souborů, obsahující testovaný scénář bez nutnosti zásahu do zdrojového kódu. Programová část testů byla nachystána na začátku vývoje a dále již kromě drobných úprav do ní nebylo nutné zasahovat.

Programová část se stará o postupné spuštění všech testovaných scénářů z celého adresáře. Ke každému scénáři utvoří jednotlivé výstupní soubory (viz kapitola číslo 9.3), sloužící ke zjištění, co se při běhu testu v Mantě a Expressins modulu dělo.

9.2 Využití

Testovací scénáře byly využívány v průběhu celého vývoje a budou se využívat i nadále v rámci údržby a případného dalšího rozvoje Expressions modulu.

Využití se liší nejen účelem, ale také samotnou přípravou testovacího scénáře. Ne pro všechny případy je vyžadován komplexní SQL skript s očekávaným výstupem. Naopak například pro vývoj nové funkcionality by toto byla zbytečná práce.

9.2.1 Vývoj nové funkcionality

V iniciální fázi vývoje, byla připravena základní struktura Expressions modulu a struktura pro spouštění testů. Dále již probíhal vývoj primárně v cyklu, kdy pro každý nový konstrukt SQL jazyka byl nejprve napsán test, podle kterého byla implementována obecná logika, tak aby byl test splněn.

Důvodem tohoto postupu vývoje byla především potřeba zjišťování konkrétního způsobu reprezentace SQL konstruktu v AST a grafu datových toků, které pro něj Manta tvoří, aby bylo možno se na ně napojit. Pro fungování první fáze zpracování (viz kapitola číslo 7) je nutné především vědět, které vrcholy v AST pro daný SQL konstrukt vznikají, ale i jejich strukturu. Druhou otázkou, kterou

bylo nutné pomocí těchto testů zodpovědět, bylo především nalezení mapování mezi AST vrcholem a vrcholem grafu datových toků, což se ukázalo jako netriviální.

Tento postup, ale kromě analýzy fungování Manty umožnil urychlení vývoje tím, že nebylo po každé implementační změně nutné spouštět kompletní nástroj Manta nad kompletní databází. Pro odzkoušení specifické úpravy bylo možno spustit pouze jeden testovací scénář pokrývající řešený problém a během okamžiku vidět výsledek úpravy ve výstupním souboru.

Pro potřeby vývoje vznikl adresář „`scripts/develop`“, ve kterém byl primárně jeden test „`test1.sql`“, jehož obsah se měnil podle aktuálně vyvíjené funkcionality Expressins modulu. Tento test nesloužil k ověření zda výstup odpovídá očekávání a soubor s očekávaným výsledkem tedy v tomto případě použití testů nebyl podstatný. O to podstatnější však byly jednotlivé výstupní soubory, které pomáhaly s analýzou pro další řešení.

Díky tomu, že jeden skript může obsahovat více SQL příkazů, bylo možno v jednu chvíli testovat i více variant SQL konstruktu.

Tyto testovací scénáře sloužily primárně k vývoji a nebyly přímo verzovány v SVN, ale sloužily jako podklady pro vznik sady testovacích scénářů uložených v „`scripts/mix`“ pro automatické testování popsané v následujících dvou kapitolách.

9.2.2 Úprava stávající funkcionality

Úpravy zdrojového kódu Expressions modulu v rámci vývoje je možné rozdělit do dvou kategorií. Lokální, které mají vliv pouze na aktuálně vyvíjenou funkcionalitu a globální úpravy které je nutné provést ve společné části kódu a mají dopad i na již vyvinutou a otestovanou funkcionalitu.

Lokální změny byly z pohledu testů již pokryty testy pro vývoj nové funkcionality (viz kapitola číslo 9.2.1). Nyní je tedy potřeba se zaměřit na testy podporující globální úpravy ve zdrojovém kódu. Globálními jsou myšleny úpravy mající plošný vliv na celý Expressins modul nebo větší část jeho fungování, ale ne na celou Mantu.

Při globálních úpravách je podstatné zachovat funkční již vyvinuté a otestované bloky. Proto pro každý již hotový SQL konstrukt a blok funkčnosti byla připravena sadu testů v adresáři „`scripts/mix`“, kontrolující jeho korektní funkčnost při různých variantách vstupu. Tyto testy je poté možné pravidelně a rychle spustit vždy po provedení globální úpravy a velmi rychle je ověřeno, že již hotová funkčnost nebyla aktuální úpravou porušena. Pokud by ovšem k porušení došlo, je rychle detekováno v jakém případě a je možno řešit pouze tento případ, místo kompletní kontroly veškerého již hotového zdrojového kódu.

Globální úprava může mít za následek i požadovanou změnu výstupu. Ta se projeví na většině testovacích scénářů, u kterých je následně nutné opravit požadované výstupy. Příkladem může být změna generování jmen dočasných proměnných, které některé ve výsledném popisu přetrvávají a je to požadovaný stav. Takováto oprava většího počtu testovacích scénářů je ovšem časově náročná. Výstupem testů je proto i soubor `TEST_expected.txt` (viz kapitola číslo 9.3), kte-

rým lze vstupní očekávané soubory opravit. Je pouze nutné zkontrolovat, že nově vygenerované vstupy jsou korektní a splňují naše požadavky.

9.2.3 Automatické regresní testy

Jedná se o stejnou sadu testů jako v případě testů pro úpravu stávající funkcionality (viz kapitola číslo 9.2.2). Liší se pouze způsobem spouštění a jejich úlohou.

Úlohou těchto testů je zajistit korektnost Expressins modulu v kontextu celé Manty nejen v rámci jeho vývoje, ale i do budoucna. Z tohoto důvodu jsou testy spouštěny na integračním serveru v rámci pravidelných buildů celé Manty jako součást „Continuous Integration“ (používá se Jenkins server, viz Literatura [10]) pro zajištění, že nová změna kdekoliv v Mantě nerozbila některou z funkcionalit Expressions modulu.

9.2.4 Příprava podkladů pro text práce

Posledním využitím testovacích scénářů byla příprava podkladů pro text této práce, především pro první fázi zpracování (viz kapitola číslo 7).

Testovací scénáře byly psány jako minimalistické SQL skripty (z důvodu rychlého růstu počtu vrcholů grafu datových toků) v adresáři „`scripts/text`“, obsahující v textu popisovaný SQL konstrukt. V případě tvorby podkladů byly podstatné výstupy testu a především generované obrázky grafu datových toků (viz kapitola číslo 9.3). Naopak nebylo nutné kontrolovat očekávaný výstup, který proto není součástí scénářů.

9.3 Výstupy

Každý testovací scénář při svém běhu generuje sadu výstupních souborů, sloužících k analýze a vývoji Expressins modulu. Jednotlivé soubory jsou pojmenovány podle jména souboru testovacího scénáře s příponou podle obsahu:

TEST_expected.txt Soubor obsahuje generované popisy transformací pro jednotlivé atributy ze vstupního SQL skriptu. Jeho obsah je porovnáván se vstupním očekávaným výsledkem.

TEST_expressions.txt Výpis finálních výstupů jako u `TEST_expected.txt`, rozšířen o všechny podpopisy vzniklé v první fázi zpracování. Jednotlivé popisy i podpopisy jsou označeny vrcholem grafu, ke kterému náleží pro usnadnění orientace při vývoji.

TEST_flow.txt Textový zápis grafu datových toků, tak jak jej generuje Manta.

TEST_graph.png Obrázek grafu datových toků jak je generován knihovnou `dot`.

TEST_graph.png.dot Vstupní soubor pro generování obrázku knihovnou `dot`, tak jak jej generuje Manta.

TEST_tree.txt Textový zápis AST, tak jak jej generuje Manta.

Závěr

Cílem této práce bylo nalezení způsobu prezentace transformace jednotlivých atributů obsažených ve vstupních Oracle SQL skriptech a implementace prototypu do nástroje Manta.

Navržený výsledný formát použitý pro popis jednotlivých transformací splňuje očekávání a požadavky které pro něj byly definovány. Toto bylo prokázáno pomocí prototypové implementace v nástroji Manta, kde se zvolený formát shledal s kladným přijetím.

V průběhu práce bylo nutno řešit výběr vhodných vstupních dat pro běh analýzy předcházející tvorbě výsledného popisu. Použití čisté textové reprezentace vstupního SQL skriptu není vhodné, protože by bylo nutné provádět analýzu textu, která je již řešena jinými běžnými nástroji tohoto typu. Finálně se ukázaly jako vhodné vstupní datové struktury AST a graf datových toků odpovídající vstupnímu SQL skriptu. Z těchto struktur je možné získat všechny potřebné informace, bez nutnosti provádět duplicitně logiku použitou při jejich tvorbě.

Na podobu výsledného popisu transformací má zásadní vliv krok inliningu. Díky volbě skládání popisu z co možná nejmenších částí a následnému inliningu, bylo možné dosáhnout dobře strukturovaného a čitelného popisu pro uživatele.

Celkově je možné výsledky této práce využít pro tvorbu přehledného a uživatelsky dobře čitelného popisu transformace jednotlivých atributů obsažených ve vstupních Oracle SQL skriptech. A poskytnutou prototypovou implementaci je (po úpravách specifických pro nástroj Manta) možné po dodání vlastních nástrojů pro tvorbu AST a grafu datových toků, využít jako základ pro implementaci do vlastního systému.

Možnosti dalšího rozvoje

Ačkoliv návrh řešení i jeho prototypová implementace splnily očekávání této práce, naskýtají se další možnosti jejich rozšíření.

Mezi hlavní témata patří rozšíření podle specifik dalších dialektů jazyka SQL kromě Oracle. Protože jednotlivé dialekty mají různé odlišnosti (především ve své AST reprezentaci), je pro ně potřebné upravit první fázi zpracování.

Dalším možným rozšířením by mohly být parametrizovatelné úpravy formátu výsledného popisu podle požadavků jednotlivých uživatelů. Tyto nebyly zahrnuty do výsledků této práce, protože mohou být velmi specifické a nedá se dopředu určit jejich povahu.

Seznam použité literatury

[1] Manta Tools s.r.o., Manta Flow. Dostupný na: https://getmanta.com/, 2018.

[2] Wikipedia, Data lineage. Dostupný na: https://en.wikipedia.org/wiki/Data_lineage, 2018.

[3] Torben Ægidius Mogensen. *Introduction to Compiler Design*. Druhé vydání. Springer International Publishing, 2017. ISBN 978-3-319-66965-6.

[4] graphviz.org, Graphviz. Dostupný na: https://www.graphviz.org/, 2018.

[5] Pivotal Software Inc., Bean overview. Dostupný na: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html#beans-definition, 2018.

[6] Pivotal Software Inc., Spring Framework Reference Documentation. Dostupný na: https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/index.html, 2018.

[7] Vaskaran Sarcar. *Java Design Patterns*. Apress, 2015. ISBN 978-1-484-21802-0. Strany 149-154.

[8] Oracle, Oracle SQL Reserved Words. Dostupný na: https://docs.oracle.com/database/121/SQLRF/ap_keywd001.htm, 2018.

[9] Even, Shimon. *Graph Algorithms*. Druhé vydání. Cambridge University Press, 2011. ISBN 978-0-521-73653-4. Strany 46–48.

[10] jenkins.io, Jenkins. Dostupný na: https://jenkins.io/, 2018.

Seznam obrázků

Seznam použitých zkratek

AST Abstract syntax tree

JVM Java virtual machine

XML eXtensible Markup Language

PLSQL Procedural Language/Structured Query Language

SQL Structured Query Language

Spring Spring Framework 3.2.8

SVN Apache Subversion

Manta softwarový balík Manta Flow do kterého je integrována prototypová implementace této práce

DFS Depth-first search