FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Paralelizácia ETL procesov DW ČVUT s využitím nástroja Pentaho
Student:	Adam Marhefka
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2023/2024

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. prosince 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Pokyny pro vypracování

Cílem práce je navrhnout vhodnou paralelizaci aktuálních ETL procesů datového skladu ČVUT (dále DW ČVUT) a na vhodně zvoleném příkladu (části ETL procesu) návrh ověřit formou POC (Proof of Concept). Vypsány jsou dvě souběžné bakalářské práce, každá z nich se snaží problém řešit pomocí jiné technologie. V závěru prací bude zhodnocení silných a slabých stránek příslušného řešení, což umožní vybrat vhodnější cestu pro další rozvoj DW ČVUT.

1. Společně se souběžnou bakalářskou prací specifikujte požadavky na paralelizaci ETL procesů DW ČVUT.

2. Společně se souběžnou bakalářskou prací zvolte vhodný příklad (část ETL procesu DW ČVUT), který budete paralelizovat ve zvolené technologii.

3. Analyzujte možnosti nástroje Pentaho z pohledu naplnění požadavků z bodu 1.

4. Navrhněte a implementujte paralelizaci části ETL z bodu 2 pomocí technologie Pentaho. V případě, že tento nástroj neposkytuje dostatečné prostředky, rozšiřte své řešení na aplikaci napsanou nad nástrojem Pentaho.

5. Zhodnoťte realizované řešení z hlediska naplnění požadavků z bodu 1 a také z hlediska budoucího použití pro správu celého ETL procesu DW ČVUT.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 6. prosince 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Paralelizácia ETL procesov DW ČVUT s využitím nástroja Pentaho

Adam Marhefka

Katedra softwarového inženýrství
Vedúci práce: Ing. Michal Valenta, Ph.D.

11. mája 2023

Poďakovanie

Chcel by som poďakovať vedúcemu tejto bakalárskej práce Ing. Michalovi Valentovi, Ph.D. za jeho odborné rady a kvalitné vedenie práce a taktiež za to, že mi umožnil túto prácu spracovať. Ďakujem patrí taktiež vedeniu Dátového skladu ČVUT za možnosť vypracovania tejto práce a získania ďalších cenných znalostí. V neposlednom rade ďakujem mojej rodine a kamarátom, ktorí mi boli dôležitou oporou počas celého môjho štúdia a bez ktorých by to bolo celé o poznanie ťažšie.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval samostatne a že som uviedol všetky použité informačné zdroje v súlade s Metodickým pokynom o dodržiavaní etických princípov pri príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, obzvlášť skutočnosť, že České vysoké učení technické v Prahe má právo na uzavretie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 citovaného zákona.

V Prahe 11. mája 2023 . . . . . .

Odkaz na túto prácu

Marhefka, Adam. *Paralelizácia ETL procesov DW ČVUT s využitím nástroja Pentaho*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Táto bakalárska práca sa zaoberá paralelizáciou ETL procesov Dátového skladu ČVUT. Cieľom je analyzovať možnosti aktuálne používaného nástroja Pentaho Data Integration z pohľadu splnenia požiadaviek a v prípade potreby rozšíriť prostriedky nástroja pomocou vlastnej aplikácie. Požiadavky sú definované spolu so súbežnou bakalárskou prácou, ktorá skúma riešenie pomocou iných nástrojov. V praktickej časti sa na základe analýzy využívajú vhodné funkcionality aktuálneho nástroja na dátovú paralelizáciu a taktiež správu logovania. Pre implementáciu úlohovej paralelizácie a ostatných požiadaviek je vytvorená aplikácia v jazyku Java s využitím Kettle Java API. Aplikácia ukladá komponenty potrebné k nahrávaniu vo forme orientovaného acyklického grafu, čím zaručuje správne poradie vykonávania ETL procesov. Na uloženie potrebných závislostí a informácií o komponentách či na správu metadát o nahrávaniach sú použité databázové tabuľky. Na interakciu s aplikáciou je poskytnuté API rozhranie. V závere práce je zhodnotený prínos realizovaného riešenia z pohľadu budúceho využitia pre správu ETL procesov Dátového skladu ČVUT.

Kľúčové slová aplikácia, dátový sklad, ETL, Java, Kettle API, paralelizácia, Pentaho Data Integration, POC, Spring

Abstract

This bachelor thesis deals with the parallelization of ETL processes of the CTU Data Warehouse. the aim is to analyze the capabilities of the currently used Pentaho Data Integration tool from the point of view of meeting the requirements and, if necessary, to extend the tool's resources by means of a custom application. the requirements are defined together with a parallel bachelor thesis that explores the solution using other tools. In the practical part, based on the analysis, appropriate functionalities of the current tool for data parallelization and also log management are used. For the implementation of task parallelization and other requirements, a Java application is developed using the Kettle Java API. the application stores the components required to load the data warehouse in the form of a directed acyclic graph, thus guaranteeing the correct order of execution of ETL processes. To store the necessary dependencies and component information or to manage metadata about ETL loads, database tables are used. An API interface is provided to interact with the application. In the conclusion of the thesis, the contribution of the implemented solution is evaluated in terms of its future use for the management of ETL processes of the CTU Data Warehouse.

Keywords application, data warehouse, ETL, Java, Kettle API, parallelization, Pentaho Data Integration, POC, Spring

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam výpisov kódu

Úvod

Dátový sklad je základnou súčasťou tvorby strategických rozhodnutí v rámci akéhokoľvek typu podniku či spoločnosti. V dnešnej bleskurýchlo fungujúcej dobe je potrebné rozhodnutia robiť vecne, často expresne a s čo najväčšou kvalitou. Na to je potrebný dobre fungujúci dátový sklad, ktorý sa používa ako podpora pre takéto činnosti. České vysoké učení technické v tejto oblasti nezaháľa a už v roku 2013 na Fakulte informačních technologií vznikla prvá verzia Dátového skladu ČVUT. Na vývoji pracovalo počas rokov aj viacero študentov vo svojich bakalárskych a diplomových prácach.

Základným stavebným blokom každého dátového skladu sú ETL procesy. Bez nich by dátový sklad svoje dáta nikdy nezískal – ide totiž o procesy spojené s nahrávaním a historizáciou dát. S pribúdajúcim objemom spracovávaných dát sa dôležitosť takejto technológie zvyšuje. Tlak je vytváraný hlavne na rýchlosť nahrávania dát, ktorá úzko súvisí s aktuálnosťou dát v dátovom sklade. Keďže sú zo získaných dát vytvárané výstupy na základe požiadaviek koncových používateľov, aktuálnosť je nevyhnutná. Tento tlak sa odzrkadľuje v snahe skracovať dobu spracovania čo najviac. To je dosiahnuteľné spracovávaním viacerých úloh alebo viacerých menších častí dát paralelne.

Spolu so súbežnou bakalárskou prácou kolegyne Kristiny Zolochevskaiej bude v oboch prácach riešený práve problém paralelizácie ETL procesov v Dátovom sklade ČVUT. Ten aktuálne na beh ETL procesov využíva open-source nástroj Pentaho Data Integration, ktorý je súčasťou platformy Pentaho. Keďže ide o výraznejšie zmeny vo fungovaní dátového skladu, bol zvolený postup dvoch súbežných bakalárskych prác. Táto bakalárska práca preskúma a navrhne možnosti paralelizácie využívajúce aktuálne používaný nástroj a v bakalárskej práci kolegyne sa preskúmajú iné dostupné nástroje. Požiadavky a aj časti ETL, ktoré budú v oboch prácach spracovávané budú nadefinované spoločne. Cieľom oboch bakalárskych prác je dôkladne preskúmať možnosti týchto prístupov a následne podľa jedného z riešení smerovať ďalší vývoj Dátového skladu ČVUT.

V tejto bakalárskej práci teda budú preskúmané a analyzované možnosti aktuálne využívaného ETL nástroja s dôrazom na paralelizáciu ETL procesov. V prípade, že funkcionality nebudú dostatočné, bude navrhnuté vlastné riešenie nad rámec ETL nástroja. Hlavným prínosom pre Dátový sklad ČVUT by malo byť skrátenie doby nahrávania, ktorým sa umožní pravidelnejšie nahrávanie a teda aj aktuálnejšie dodávanie dát koncovým používateľom. Momentálne je nahrávanie príliš časovo náročné a s pribúdajúcim množstvom spracovaváných tabuliek a dát by sa doba nahrávania mohla predlžovať do neúnosných dĺžok.

Ciele práce

Hlavným cieľom tejto bakalárskej práce je navrhnúť paralelizáciu ETL procesov Dátového skladu ČVUT s využitím aktuálne používaného nástroja Pentaho Data Integration (PDI), prípadne navrhnúť a implementovať riešenie pomocou aplikácie, ktorá bude nástroj PDI využívať. Tento návrh je potrebné následne overiť na zvolenej časti ETL procesov pomocou Proof of Concept (POC).

Spoločným cieľom so súbežnou bakalárskou prácou je špecifikovať požiadavky na paralelizáciu ETL procesov a zvoliť ich vhodnú časť tak, aby na tejto časti mohlo byť riešenie prezentované.

Naväzujúcim cieľom je analyzovať možnosti nástroja PDI na základe špecifikovaných požiadaviek a v prípade nedostatočných prostriedkov tohto nástroja navrhnúť a implementovať riešenie pomocou aplikácie nad týmto nástrojom.

V neposlednom rade je cieľom zhodnotiť realizované riešenie z hľadiska naplnenia definovaných požiadaviek a z hľadiska budúceho použitia tohto riešenia pre správu celého ETL procesu. Zhodnotenie silných a slabých stránok riešení v súbežných bakalárskych prácach umožní vybrať vhodnejšiu alternatívu pre ďalší rozvoj Dátového skladu ČVUT.

Prínosom bakalárskej práce bude urýchlenie ETL procesov, ktoré sú zodpovedné za nahrávanie dát do dátového skladu. Toto zrýchlenie vyústi do možnosti pravidelnejšieho nahrávania a teda dodávania aktuálnejších dát. Aktuálnosť dát je prioritou, keďže tieto dáta sú používané na dôležité rozhodovacie procesy v rámci univerzity aj mimo nej.

Kapitola 1

Dátové sklady a ETL procesy

Táto kapitola zahŕňa popis dátových skladov a s nimi spojených ETL procesov. V prvej časti sú priblížené dve hlavné definície dátových skladov a popísaná ich moderná architektúra a bežne využívané typy historizácie dát. V druhej časti je vysvetlené, ako ETL procesy fungujú a sú priblížené aj iné typy nahrávania dát. V závere kapitoly sú vysvetlené možnosti paralelizácie ETL procesov.

1.1 Dátové sklady

„*Dátový sklad je informačný systém, ktorý dlhodobo ukladá dáta z informačných systémov spoločnosti za účelom potencionálneho analytického využitia. Tieto dáta bývajú často integrované do jedného dátového modelu a bývajú uchovávané aj historické hodnoty jednotlivých dátových entít.*“ [1, s. 3]

Najznámejšie pohľady na architektúru dátového skladu vyslovili **William Bill Inmon** a **Ralph Kimball**. Kým Inmon pojem dátový sklad vysvetľuje ako ucelenú vrstvu nad používanými systémami, z ktorej sa poskytujú dáta vo forme dátových trhov[1], Kimball používa prístup, ktorý najskôr predpokladá stanovenie cieľov na výsledné dáta a až potom hľadanie zdrojových dát pre tento cieľ. Inmonov popis je považovaný za tzv. „top-down“ prístup a Kimballov naopak za tzv. „bottom-up“ prístup. [2]

1.1.1 Definícia podľa Inmona

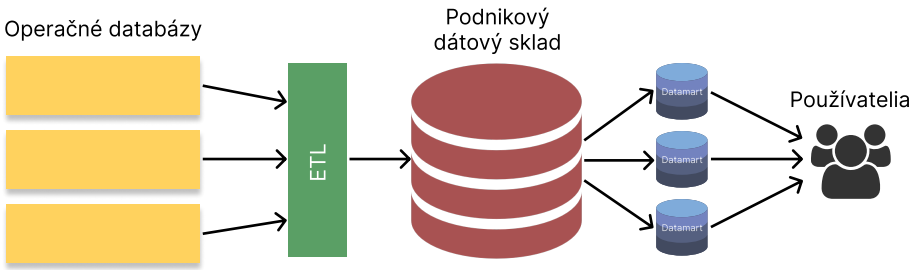
Definícia podľa Inmona znie – „*A data warehouse is a subject–oriented, integrated, nonvolatile, and time–variant collection of data in support of management's decisions.*“ [3, s. 31]

Pojmom **subjektovo-orientované** sa myslí zameranie sa na konkrétne subjekty firmy alebo spoločnosti. Ako príklad sa uvádzajú konkrétne subjekty

[1] súhrn údajov zameraných na jeden predmet alebo funkčnú oblasť spoločnosti

v dátovom sklade – zákazník, predaje, tovar a pod. **Integrácia** je považovaná za najdôležitejšiu charakteristiku dátového skladu. Dáta sú do dátového skladu čerpané z rôznych zdrojov a ich konvertovaním, úpravou a zosumarizovaním sa vytvára jednotný obraz. Integráciou sa teda zmažú rôzne princípy ukladania dát v rôznych aplikáciách a vytvorí sa normalizovaný pohľad na dáta. **Nemennosť** dát tkvie v tom, že kým operačné dáta v aplikáciách sú upravované často, dáta v dátovom sklade sú načítavané v pravidelných dátových snímkach, je k nim pristupované, ale podstatou je, že sa neupravujú. Pri akejkoľvek zmene operačných dát je potrebné vytvoriť novú dátovú snímku. S tým súvisí posledná charakteristika – **časová odlíšiteľnosť** dát. Pri zmenách v operačných dátach sa vytvorí nová dátová snímka s definovaným časovým momentom. Všetky dáta v dátovom sklade sú aktuálne k nejakému časovému momentu. To môže byť implementované pomocou časovej pečiatky alebo dátumu integrovania do dátového skladu. Tým sa v sklade vytvára historizácia v časovom horizonte 5 až 10 rokov, oproti operačným dátam, ktoré sú bežne aktuálne len v horizonte 60 až 90 dní. [3, s. 31 – 35]

Niekedy sa tejto architektúre hovorí aj „hub and spoke“ – normalizované dáta sú uložené v centrálnej databáze a nad touto databázou sú postavené dátové trhy. Ide o časovo náročnejšiu architektúru, ale poskytuje jednoduchú možnosť rozšírenia o nové dátové zdroje a následný proces tvorby nových dátových trhov. [1, s. 8]



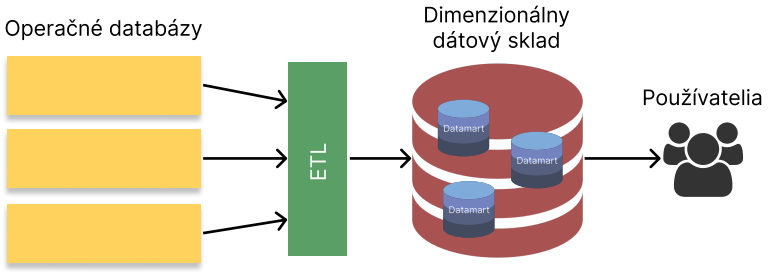
Obr. 1.1: Architektúra dátového skladu podľa Inmona [4]

1.1.2 Definícia podľa Kimballa

Kimballova definícia vytvára dátové zdroje, resp. dátové trhy na základe biznisových požiadaviek spoločnosti. Podľa biznisových požiadaviek sú teda zistené potrebné zdrojové dáta a v dátovom sklade je vytvorený dimenzionálny model. Tento model delí dáta na faktové, ktoré obsahujú transakčné dáta a na dimenzionálne dáta, ktoré obsahujú dáta na podporu faktov. Takáto schéma sa nazýva star schema a je základom Kimballovho prístupu k architektúre dátového skladu.

Transformované dáta prúdia priamo do dátových trhov a neukladajú sa do jedného zložitého celku. Tieto dátové trhy potom môžu byť jednoducho distri-

buované medzi používateľov. Výhodou je rýchlejšia tvorba nových výstupov alebo pridanie nových dátových zdrojov na základe požiadaviek používateľov. Naopak, nevýhodou je strata jednotného pohľadu na dáta ako v prípade Inmonovho prístupu a taktiež obmedzenie spracovania niektorých biznisových požiadaviek, keďže sa nezameriava na celú spoločnosť, ale hlavne na jej procesy. [5] Táto architektúra je tiež nazývaná ako „datamart bus“. [1, s. 12]



Obr. 1.2: Architektúra dátového skladu podľa Kimballa [4]

1.1.3 Súčasná architektúra dátových skladov

Moderné architektúry dátových skladov vychádzajú z vyššie vysvetlených základných pohľadov. Architektúru súčasných dátových skladov je možné rozdeliť na viacero vrstiev (layer) a to:

- **source systems layer** – nie je priamou súčasťou dátového skladu, ale ide o základnú vrstvu architektúry. Ide o zdrojové systémy, z ktorých sú čerpané dáta do dátového skladu. Takýmito systémami môžu byť rôzne operačné systémy spoločnosti. Dáta sú získavané podľa definície získavania dát a to napríklad cez súbor, webovú službu, priame pripojenie na zdrojovú databázu alebo inými spôsobmi.

- **landing layer** – táto vrstva prijíma, validuje a archivuje dáta vo forme súborov. Väčšinou sa používa na spracovanie jednorázových dátových vstupov, prípadne vstupov, ktoré nie sú uložené vo fyzickej databáze.

- **staging layer** – ide o prvú databázovú vrstvu architektúry dátových skladov. Obsahuje dáta získané zo zdrojových systémov alebo súborov (landing layer) v nezmenenom stave, iba s pridanými exekučnými metadátami. Táto vrstva slúži hlavne pre základné kontroly dátovej kvality ako napríklad chýbajúce záznamy, duplicitné záznamy, chýbajúce cudzie kľúče atď.

- **integrated data layer (IDL)** – hlavná komponenta dátového skladu, niekedy označovaná aj ako „dátový sklad“ alebo „centrálna databáza“. Obsahuje dáta z viacerých dátových zdrojov a informačných systémov

(integrácia), ktoré majú vysokú granularitu a sú oddelené od biznisovej logiky. Tento jednotný dátový model popisujúci celú inštitúciu obsahuje aj historické hodnoty (časová odlíšiteľnosť) a je subjektovo-orientovaný. [1, s. 19 – 24]

- **semantic layer** – vytvára tzv. „jednotnú pravdu“ pomocou definovania jednotných biznisových pravidiel a obmedzení. Tieto biznisové definície sú stanovené nad dátami centrálnej databázy a eliminujú riziko nekonzistentných výstupov v rámci access vrstvy. Väčšinou sú vytvárané pomocou databázových pohľadov, ktoré sú následne používané ako podklad pre tvorbu dátových trhov.
- **access layer** – v preklade prístupová vrstva, slúži na prístup užívateľov k dátam dátového skladu a to buď v priamej forme (pomocou sql) alebo nepriamej forme (reportingový nástroj). Zväčša ide o spojenie viacerých dátových trhov. Dáta sú agregované s obchodných účelom – pomocou obchodných podmienok na dáta. [1, s. 28 – 29]
- **information delivery layer** – slúži na biznisové využitie dát s konkrétnym cieľom a to napríklad tvorbou prediktívnych modelov, dátových zostáv pre potreby dátových analýz alebo využitie dát v rámci reportingových nástrojov. Vytvára rôzne prístupy k využitiu dát, t.j. k získaniu cenných informácií, ktoré tieto dáta poskytujú. V tejto vrstve sú tiež poskytované komplexnejšie informácie o dátach vo forme dátových katalógov alebo popisu biznisových pravidiel. [1, s. 33]

1.1.3.1 Historizácia dát

Historizácia je kľúčovou časťou každého dátového skladu. Keďže jeho primárnou funkciou je analýza historických dát, je dôležité udržovať rôzne stavy dát. Na to slúži SCD (Slowly changing dimensions) – ide o dimenzie schopné udržovať zmeny dát. [6] Základné typy SCD sa rozdeľujú do typov 0 až 7:

- SCD0 – záznam nie je možné v čase meniť, t.j. sú udržiavané iba originálne dáta,
- SCD1 – pri zmene záznamu na zdroji dochádza k prepisu na nové hodnoty, historizácia neexistuje,
- SCD2 – zmena záznamu na zdroji spôsobí pridanie nového riadku,
  - takéto riešenie vyžaduje úpravu primárneho kľúča na umelý technický kľúč a pridanie atribútov date_from (začiatok platnosti), date_to (koniec platnosti) a version (verzia záznamu),
  - pomocou identifikátoru záznamu je možné získať kompletnú históriu zmien, [1, s. 26 – 27]

- SCD3 – história je vytvorená pridaním atribútu,
  - napríklad pre atribút profesia pridáme atribút predošlá profesia, do ktorého budeme ukladať pôvodnú hodnotu v prípade zmeny,
  - oproti SCD2 umožňuje, ale udržať len poslednú verziu dát, [6]
- SCD4 - pre často sa meniaciu dimenziu sa vytvorí nová mini dimenzia, v ktorej sa udržuje historizácia,
  - rapídne sa meniaci atribút alebo skupina atribútov teda nevytvára veľké množstvo duplicitných záznamov v pôvodnej dimenzii,
- SCD5 až SCD7 – ide o rôzne kombinácie predošlých typov SCD. [7]

1.2 ETL procesy

ETL procesy sú neoddeliteľnou súčasťou dátových skladov. Skratka ETL značí tri základné časti a tými sú:

1. **extract**,
2. **transform**,
3. **load**.

Ide o integračný proces dát, ktorý kombinuje dáta z rôznych zdrojov do jedného konzistentného úložiska, ktorým je dátový sklad alebo iný cieľový systém. Predstavuje základ pre dátovú analýzu a strojové učenie. Pomocou rôznych biznisových pravidiel, čistení a organizovania dát umožňuje adresnejšie riešiť špecifické biznisové problémy spoločnosti. [8]

Rýchlosť dodania dát koncovým používateľom má veľký dopad na architektúru ETL procesov. Väčšinou je preferované dávkové nahrávanie dát, ktoré ale nevyhovuje rýchlosti, ktorá sa začína postupne vyžadovať. To sa dá vyriešiť paralelným spracovaním, rozumnejšími algoritmami spracovania alebo pridaním výpočtovej sily. Postupne ale dochádza k bodu, kedy ani takéto riešenie nie je dostatočné a namiesto dávkového nahrávania sa vyžaduje priebežný tok dát zo zdroja do dátového skladu a následne do cieľového systému. [9, s. 7]

Nasledujúci text sa bude zaoberať bližším popisom základných častí ETL procesov. Vykonávanie jednotlivých krokov je niekedy nazývané aj ako staging – zapisovanie dát na disk. Staging takmer vždy vyžaduje fyzické snímky dát, ktoré sú ukladané po vykonaní kroku v spracovaní, aby mohli byť použité v ďalšom kroku spracovania. [9, s. 17 – 18]

1.2.1 Extrahovanie dát

Prvou základnou časťou každého ETL procesu musí byť extrahovanie dát zo zdroja. Bez tohto kroku by nemal celý proces a ani stavba dátového skladu žiaden zmysel. Dáta zo zdrojových systémov sú bez zmeny ukladané do databázy dátového skladu. Keďže tento krok neobsahuje zložité transformácie dát, je väčšinou veľmi rýchly a v prípade potreby umožňuje pravidelnú extrakciu dát pre následujúce kroky. Zložité úpravy dát sa vykonávajú v časti transformácie, avšak v časti extrahovania je vhodné vykonávať konverzie nízkoúrovňových dát.

Prvotne nahraté dáta môžu byť po vykonaní následujúceho kroku (čistenie) buď zmazané alebo ponechané pre prípad porovnania zmien v dátach medzi dvoma rôznymi extrakciami.

1.2.2 Transformácia dát

Pre zaručenie kvality dát v dátovom sklade je potrebné extrahované dáta transformovať tak, aby spĺňali požiadavky dané dátovým skladom. Krok transformácie sa zväčša delí na čistenie a úpravu dát.

Čistenie dát (cleaning) slúži hlavne na elimináciu nevyhovujúcich dát a zahŕňa napríklad kontrolu validity dát (či atribút obsahuje len povolené hodnoty), kotrolu duplikátov, kontrolu konzistencie naprieč dátami alebo kontrolu definovanej biznisovej logiky. Pri nájdení nezrovnalostí sú takéto dáta odstránené. Vyčistené dáta sú znova separátne uložené pre využitie v ďalších krokoch. Nájdené nezrovnalosti môžu byť reportované späť do zdrojových systémov pre elimináciu chybných dát pri ďalšom extrahovaní.

Úprava dát (conforming) zaručuje, že v prípade spájania dát z rôznych dátových zdrojov dáta spĺňajú definované biznisové pravidlá, domény a obmedzenia. Dáta sú upravované do takej podoby, aby výstup z nich bol konzistentný. Môže ísť o zmenu dátových typov, úpravy textových hodnôt do požadovanej formy atď. [9, s. 18 – 19]

1.2.3 Nahrávanie dát

Posledným dôležitým krokom je nahratie výsledných dát (load), ktoré boli získané extrahovaním a následnou transformáciou, do centrálnej databázy dátového skladu. Zväčsa sa dátový sklad nahráva prvotným nahratím záznamov a následne sa využíva periodické nahrávanie inkrementálnych zmien, čo vytvára históriu dát. Tento proces je automatizovaný a nahráva sa vždy po väčších kusoch dát (tzv. batch loading). Z dôvodu časovej náročnosti sa nahrávanie vykonáva v čase, kedy sú zdrojové systémy aj dátový sklad menej vyťažené. [8]

1.2.4 Alternatívy k ETL

V Dátovom sklade ČVUT je využívané ETL spracovanie, ale pre úplnosť budú uvedené aj iné známe alternatívy k ETL. V nasledujúcej časti preto budú v skratke popísané aj ELT a ETLT procesy.

1.2.4.1 ELT

Oproti ETL, proces ELT prehadzuje kroky nahrávania do cieľového systému a transformácie dát. Skratka ELT totiž znamená: extract, load, transform. Hlavným rozdielom je to, že zdrojové dáta sú nahrávané priamo do cieľovej databázy, v ktorej sú až následne transformované. Výpočtová náročnosť transformácie dát sa tak prenesie na databázový stroj.

Hlavnou výhodou je rýchlejšia dátová priepustnosť a taktiež to, že transformácie môžu byť vykonávané paralelne počas nahrávania do cieľového systému. To, že sa transformácia prevedie na databázový stroj ale môže vyústiť aj do nižšej časovej odozvy na iných dotazoch (napr. reportingové dotazy). Ďalšou nevýhodou je aj strata monitoringu transformačných procesov, ktorá je v jazyku SQL zložitejšia, či malý výber ELT nástrojov na trhu (hlavne open-source nástrojov). Z pohľadu bezpečnosti (napr. spracovanie citlivých údajov v transformácií) a dátovej kvality vyhráva ETL oproti ELT. [10], [11]

1.2.4.2 ETLT

Skratka ETLT značí extract, transform, load, transform a spája dokopy funkcionality ETL a ELT. Najskôr nastane proces extrahovania dát rovnako ako v ETL/ELT procesoch. Potom nasleduje prvá transformácia dát, ktorá je rýchla a transformuje len jednotlivé dátové zdroje. Netransformuje sa viacero zdrojov zároveň, to sa ponecháva až na druhú transformáciu, ktorá prebieha na databázovom stroji. Keďže prvá transformácia sa vykonáva mimo databázy, je vhodná na odstránenie citlivých dát. Po prvej transformácií nastáva nahratie dát do cieľovej databázy (load). Niektoré takéto dáta už môžu byť použité pre analýzu a reporting. Pre zložitejšie transformácie sa vykonáva druhá transformácia, ktorá využíva výpočtovú silu databázového stroja. [12]

1.3 Paralelné spracovanie ETL

ETL spracovanie má zväčša komplikovaný návrh s pomerne vysokým množstvom operácií na veľkom objeme dát. Spracovanie musí byť mnohokrát ukončené v určitom časovom okne, čo zvyšuje potrebu jeho optimalizácie. Niektoré optimalizačné procesy sa využívajú na úrovni databázového stroja. To ale nie je dostatočné, pretože databázové optimalizácie sú len jednou z častí celého procesu. [13, s. 7]

Jednou z možností optimalizácie ETL spracovania je paralelizácia jednotlivých ETL úloh. Úlohy môžu bežať na viacerých jadrách procesoru alebo

viacerých počítačoch, čo môže signifikantne zvýšiť efektivitu spracovania. Paralelné spracovanie umožňuje využitie všetkých dostupných zdrojov pre rýchlejšie ukončenie ETL úloh. [14, s. 41]

Paralelizácia je proces spracovania viacerých operácií súčasne buď na jednom alebo viacerých počítačoch. Hlavnými dvoma prístupmi k paralelizácií sú úlohová a dátová paralelizácia. [15]

Ani paralelizácia a zrýchľovanie výpočtu však nie je nekonečné. Jeden z problémov približuje Amdahlov zákon. Tento zákon hovorí, že od určitého momentu už nie je možné výpočet paralelizáciou zrýchľovať. Problémom je to, že stále existuje nejaká časť problému, ktorú je nutné vykonávať sekvenčne. Rýchlosť spracovania je teda vždy limitovaná najdlhšie trvajúcou sekvenčnou časťou problému. Pridávanie ďalších vlákien, procesov alebo počítačov už rýchlosť dokončenia celého problému od nejakého bodu neovplyvní. Voči tomuto zákonu sa ale ako protipríklad spomína Gustafsonov zákon. Ten hovorí, že s pribúdajúcim množstvom výpočtovej sily by mal byť pridávaný aj počet spracovávaných úloh. Z tohto pohľadu sa teda spracovaním väčšieho množstva dát množstvo paralelizácie nezastavuje. [16], [17, s. 45 – 46]

1.3.1 Úlohová paralelizácia

V tomto type paralelizácie sú súčasne vykonávané úlohy, ktoré spracovávajú rôzne typy operácií. Tieto operácie sú na sebe väčšinou nezávislé a preto môžu byť vykonávané naraz. V prípade závislostí sa vytvára graf úloh, ktorý závislosti definuje a na základe nich je spúšťanie jednotlivých úloh plánované. [17, s. 24]

Príkladom môže byť počítanie rôznych výsledkov z jedného dátového setu. Napríklad potrebujeme zistiť maximum, minimum a priemer čísel z dátového setu. Takéto úlohy sú na sebe nezávislé a preto je možné ich vykonávať paralelne s tým, že každá úloha využíva rovnaký zdroj dát. [15], [16]

1.3.2 Dátová paralelizácia

Dátová paralelizácia naopak oproti úlohovej vykonáva rovnaké operácie súčasne, ale na rozdielnych kolekciách dát. V ideálnom prípade sú dáta rozdelené na rovnomerné časti, aby sa zabezpečilo rovnomerné rozdelenie výpočtovej sily medzi procesormi. Dátová paralelizácia má najväčší zmysel v prípade spracovania veľkého množstva dát, na ktorom je potrebné vykonávať rovnaké úkony alebo transformácie. [16], [17, s. 22]

Ak časti dát vyžadujú na spracovanie rozdielne množstvo času, výkon celého procesu je obmedzený rýchlosťou najpomalšieho z nich. Tento problém sa dá zmierniť tak, že sa dáta rozdelia na veľké množstvo menších častí. V tom prípade rýchly proces vyrieši viacero menších častí. Jedným z príkladov dátovej paralelizácie je paradigma Single-program-multiple-data (SPMD), kde procesy spúšťajú rovnaký kód, ale operujú na rôznych častiach dát. [18, s. 211]

Kapitola 2

Nástroj Pentaho Data Integration

V tejto kapitole bude popísaný nástroj Pentaho Data Integration (PDI), jeho základná stavba, komponenty a funkcionality, ktoré by mohli byť využité v riešení tejto práce.

Nástroj PDI je súčasťou nástrojov pod spoločným označením Pentaho. Skrátene, ale nesprávne, sa niekedy PDI nazýva tiež ako Pentaho. V tejto práci bude kladený dôraz na nezamieňanie týchto pojmov.

„*Pentaho products are a comprehensive platform used to access, integrate, manipulate, visualize, and analyze your data.*“ [19] Okrem PDI, Pentaho ponúka aj iné produkty, ako napríklad:

- Pentaho Analyzer – vizualizácia a analýza reportov,
- Pentaho User Console – webový klient pre vytváranie integrovaných nástenieek a zdieľanie s ostatnými účastníkmi, administrácia pre konfiguráciu Pentaho Server,
- Pentaho Metadata Editor – tvorbu metadátových modelov a domén,
- Pentaho Report Designer – tvorba detailných reportov vhodných na tlač s využitím rôznych zdrojových databáz,
- a iné...

Aktuálne sú všetky Pentaho nástroje spravované a vyvíjané spoločnosťou Hitachi Vantara. Mnohé z nástrojov sú dostupné vo verziách Community a Enterpise. Community verzie sú open-source a aj DW ČVUT používa vo svojom spracovaní túto verziu. [19]

Niekedy je PDI inak nazývané aj ako Kettle, pretože v minulosti išlo o samostatný open-source nástroj práve pod týmto názvom. Kettle je skratkou

pre dlhší popis – Kettle Extraction Transformation Transport Load Environment. Aj z tohto popisu vyplýva, že úlohou PDI je spracovávanie ETL procesov. Názov Kettle, v preklade kanvica, bol súčasťou kulinárskej terminológie používanej aj pre iné súčasti tohto nástroja. [20] Inými dôležitými komponentami PDI sú:

- Spoon (lyžica) – grafické prostredie na tvorbu a správu PDI úloh a transformácií, taktiež je v ňom možné robiť monitorovanie a debug úloh,
- Kitchen (kuchyňa) – program na spúšťanie PDI úloh v príkazovom riadku,
- Pan (panvica) – poskytuje funkcionality ako Kitchen, ale pre PDI transformácie,
- Carte – server pre spúšťanie PDI úloh a transformácií, je používaný pre distribúciu spúšťaných úloh na viacerých počítačoch alebo spúšťanie na vzdialenom serveri. [21, s. 54]

Táto terminológia je nadsádzkou k tomu, že ETL procesy sa dajú prirovnať ku kulinárstvu s dátami. Bola ponechaná aj po premenovaní nástroja Kettle na Pentaho Data Integration. [20]

2.1 Stavebné bloky PDI

V PDI sa na tvorbu komplexných ETL procesov používajú rôzne časti, kde každá z častí má svoje preferované použitie. Základnými stavebnými blokmi v PDI nástroji sú úlohy a transformácie. Môžu byť reprezentované v XML formáte, uložené v repozitári[2] alebo používané formou Kettle Java API. [21, s. 36]

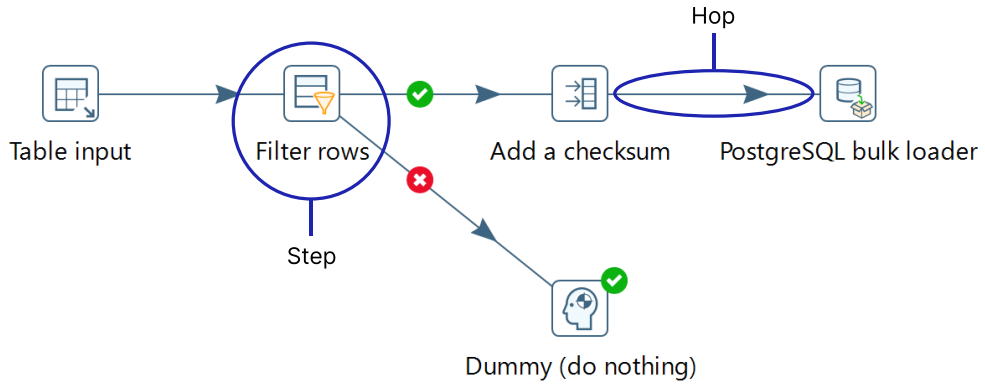
2.1.1 Transformácia

Transformácia (*transformation*) je základnou zložkou PDI, ktorá vykonáva všetky manipulácie s riadkami dát. Transformácie sú zložené z viacerých *krokov* (*steps*), ktoré sú zodpovedné za konkrétne úpravy dát. Kroky sú prepojené pomocou *hops*, ktoré reprezentujú jednosmerný dátový tok z jedného kroku do druhého. Názvy súborov transformácií sú označované koncovkou `.ktr`.

2.1.2 Krok

Krok (*step*) je stavebným blokom transformácie. Kroky sú esenciálnou časťou PDI, bez ktorej by nebolo možné vykonávať potrebné transformácie dát. Môžu mať rôzne typy podľa toho, akú majú funkcionalitu. Napríklad krok „Table

[2] PDI repozitár



Obr. 2.1: Ukážka a popis častí transformácie [20, Data integration perspective]

input“ umožňuje spúšťať SQL dotazy a získavať tak dáta z databázy. „Text file output“ zasa umožňuje zapisovať prichádzajúce riadky dát do textového súboru. V PDI je tiež vyžadované, aby boli názvy krokov v jednej transformácií unikátne.

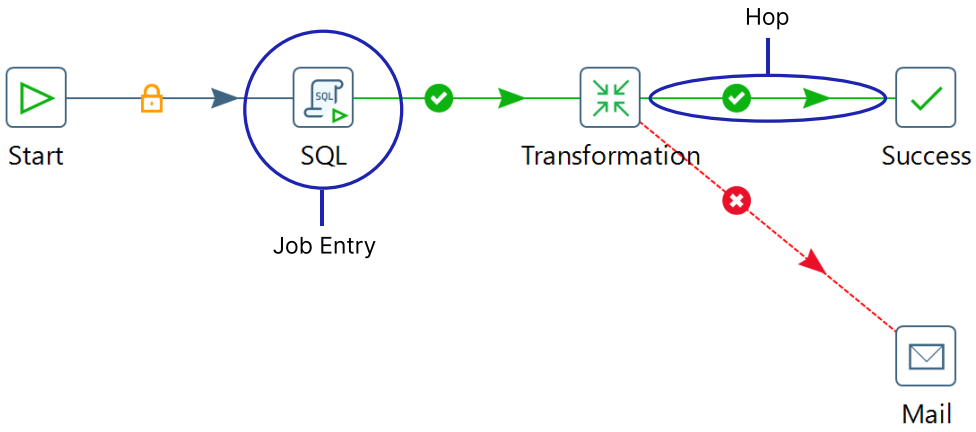
Každý krok je po spustení transformácie spustený vo vlastnom vlákne a dáta medzi krokmi postupne prechádzajú podľa definovaných dátových tokov. Z tohto dôvodu by sa dalo povedať, že transformácia nemá definovaný začiatok a koniec. Pomocou tvorby dátových tokov je vlastne definované, kde transformácia „začína“ a „končí“.

Kroky môžu mať viacero vstupných aj výstupných dátových tokov, označovaných ako *incoming hops* a *outgoing hops*. Za základnú jednotku dát je v PDI považovaný jeden riadok dát. V prípade viacerých výstupných tokov môžu byť dáta distribuované buď kopírovaním každého riadku dát do všetkých nasledujúcich krokov alebo formou *round robin* – pre každý riadok sa postupne priraďuje len jeden výstupný tok a toky sa dookola alternujú. Dátový tok medzi krokmi má obmedzenú kapacitu, ktorá ale môže byť pre každú transformáciu zmenená. V prípade naplnenia dátového toku sa krok, ktorý doň zapisuje dáta postupne zastavuje (*halting*). Naopak, ak je dátový tok prázdny, tak krok prijímajúci dáta čaká na nové dáta. [21, s. 25 – 27]

2.1.3 Úloha

Úloha (*job*) slúži na správu transformácií a ich orchestráciu, na definovanie závislostí medzi procesmi v ETL alebo rôzne iné procesy, ktoré nemusia priamo súvisieť s transformáciou dát. Ako príklad je možné uviesť kontrolu pripojenia k databázam, zadefinovanie nutného poradia spracovania jednotlivých transformácií alebo spúšťanie externých skriptov a služieb. Názvy súborov úloh sú označované koncovkou `.kjb`.

Každá úloha sa skladá z jedného alebo viacerých *job entries*. Job entries možno prirovnať ku krokom v transformácií, preto ich budem rovnako pomenovávať ako kroky úlohy aj keď exaktnejším prekladom by bolo skôr pomenovanie *vstup úlohy*. Poradie exekúcie úlohy sa definuje pomocou *job hops* medzi jednotlivými krokmi úlohy.



Obr. 2.2: Ukážka a popis častí úlohy [20, Data integration perspective]

2.1.4 Krok úlohy

Krok úlohy, vo voľnom preklade z *job entry*, je stavebným blokom úlohy. Hlavným rozdielom medzi krokom úlohy a krokom transformácie je, že kroky úlohy nebežia vo vlastnom vlákne ani paralelne, ale majú určené poradie spúšťania. Z tohto dôvodu musí mať každá úloha daný svoj začiatok a môže ho mať len jeden.

Dátové toky medzi krokmi úlohy sú obdobou dátových tokov používaných v transformáciách. Nejde ale o konštantný prúd dát. Dáta sú zasielané vo forme objektu výsledkov nazvaného *result object* vždy po ukončení exekúcie kroku úlohy. Objekt výsledkov môže obsahovať riadky dát, zoznam názvov súborov, ktoré načítal predošlý krok, informácie o počte načítaných, zapísaných a aktualizovaných riadkov alebo aj návratovú hodnotu spúšťaných externých skriptov. Dátový tok medzi krokmi môže byť označený ako:

- bezpodmienečný – tento smer je nasledovaný bez vplyvu na stav ukončenia predošlého kroku (v Spoon tmavomodrá farba),
- nasleduj ak bol stavom ukončenia kroku úspech (v Spoon zelená farba),
- nasleduj ak bola stavom ukončenia kroku chyba (v Spoon červená farba).

[21, s. 30 – 35]

2.2 Premenné, parametre, argumenty

V PDI je možné na parametrizáciu využívať premenné, parametre a argumenty. Argumentami sa myslia argumenty pridané pri spúšťaní úloh alebo transformácií pomocou Kitchen a Pan skriptov. Obsah argumentov je možné získať v transformácií pomocou kroku *Get System Info* a ďalej ich využívať v spracovaní.

Parametre je možné definovať v nastaveniach úlohy alebo transformácie a následne ich pomocou parametrov skriptov Kitchen a Pan špecifikovať. Taktiež je možné ich špecifikovať pri spúšťaní priamo v grafickom editore Spoon. Parametre sa v úlohach a transformáciách používajú pomocou špeciálneho uzátvorkovania „${názov_parametru}“ alebo „%%názov_parametru%%“.

Premenné je možné definovať v hlavnom konfiguračnom súbore *kettle.properties*, definovaním environmentálnych premenných v prostredí, v ktorom je úloha/transformácia spúšťaná alebo pri spúšťaní cez grafický editor Spoon. Premenné sa v PDI používajú rovnako ako parametre. Konfiguračný súbor *kettle.properties* obsahuje viacero premenných, na základe ktorých sa mení správanie pri spúšťaní PDI komponent. Je v ňom možné nastaviť východzie nastavenia niektorých krokov, nastavenie pripojenia k tabuľkám pre logovanie, adresárové cesty ku pluginom atď. Tento súbor je možné upravovať priamo v operačnom systéme alebo v Spoon pomocou dialogového okna „Edit“ a následne možnosti „Edit the kettle.properties file...“, v ktorej sa nachádza aj popis jednotlivých premenných. Taktiež je umožnené vytvárať si v tomto súbore vlastné premenné.

PDI ponúka aj interné premenné, ktoré slúžia na získanie informácie o aktuálnom priečinku [3], získanie počtu kópií kroku[4], získanie aktuálneho čísla kópie kroku[5] atď. [21, s. 43 – 45, 325 – 326]

2.3 Paralelné spracovanie

PDI umožňuje vykonávať dátovú aj úlohovú paralelizáciu. Paralelizácia je umožnená buď pomocou „scaling up“, čo je paralelizácia vo vláknach na jednom serveri, alebo pomocou „scaling out“, čo značí rozdelenie úloh na viacero serverov alebo počítačov. Hlavným zameraním je paralelizácia transformácií, resp. jednotlivých krokov pomocou vytvárania kópií krokov. Ku „scaling up“ sa zaradzuje vytváranie kópií krokov, ale taktiež beh viacerých úloh paralelne alebo paralelné spracovanie dát rozdelených pomocou *partitioning*. *Clustering* predstavuje „scaling out“ pomocou master a slave Carte serverov.

[3] Internal.Job.Filename.Directory
[4] Internal.Step.Unique.Count
[5] Internal.Step.Unique.Number

2.3.1 Paralelizácia krokov v transformácii

Ako už bolo spomenuté vyššie, každý krok v transformácií sa spracováva vo vlastnom vlákne. PDI umožňuje pre každý krok definovať počet kópií, kde každá kópia beží vo vlastnom vlákne. Je tak možné, aby sa paralelne spracovávalo viac riadkov. Na to sa používa možnosť „Change Number of Copies to Start...“, kde sa v dialógovom okne vloží počet vlákien (počet kópií), v ktorých sa má tento krok vykonávať naraz.

Základne má každý krok počet kópií nastavený na 1. Kópie sú v názve kroku označované pomocou postfixu „.x“, kde x je číslo kópie (čísluje sa od 0). Graficky v Spoon je krok majúci štyri kópie označený označením „x4“ vľavo hore ako je možné vidieť na obrázku 2.3. [21, s. 403 – 404]

2.3.2 Partitioning

Partitioning je proces, ktorý rozdeľuje dáta na vopred definovaný počet častí (*parts*) pomocou stanoveného delenia dát (*partitioning method*). Jednotlivé časti sú následne spracovávané paralelne. Ide o obdobu vytvárania kópií kroku avšak dáta sú delené podľa pravidiel. Pri klasickom delení v prípade paralelizácie sa používa distribúcia technikou *round robin* – vysvetlená v sekcii 2.1.2. Partitioning je dôležitý v prípade paralelizácie krokov, ktoré agregujú dáta. Príkladom môže byť paralelizácia v transformácii s krokom *Group by*. Delenie dát do kópií krokov pomocou *round robin* by spôsobilo nesprávne výsledky pri počítaní hodnôt.

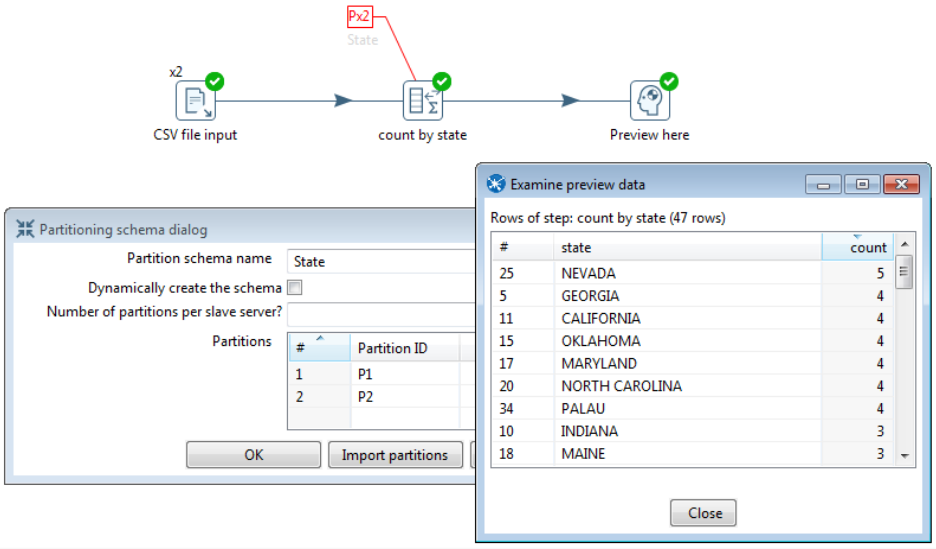
Dáta sa delia do častí podľa *partitioning schema*, kde sa definuje počet a názvy jednotlivých partícií a následne sa na kroku pomocou možnosti „Partitions...“ zvolí jedna z dostupných *partitioning method*, podľa ktorej sú dáta delené pri vstupe do kópií kroku. Taktiež sa zvolí *partitioning schema* keďže ich môže byť v transformácií definovaných viacero. Počet vytvorených vlákien pri behu odpovedá počtu partícií použitej *partitioning schema*.

Na obrázku 2.3 je ukážka použitia *partitioning* a kópií krokov. Konkrétne sa z dvoch kópií kroku, ktoré načítavajú paralelne vstup z CSV súboru, sa následne dáta v kroku *count by state* delia do dvoch častí podľa *partitioning schema* s názvom *State*.

Delenie sa dá vykonať aj na celom databázovom zdroji. Databázové partície sú označované ako *shards*. Definujú sa na konkrétnom databázovom pripojení a dá sa pomocou nich nahrávať do rôznych častí databázy paralelne. [21, s. 425 – 429], [20, Partitioning data]

2.3.3 Clustering

Pre spúšťanie transformácií paralelne na rôznych serveroch je možné použiť *clustering*. V tomto type paralelizácie sú používané Carte servery, čo sú webové servery, ktoré umožňujú vzdialenú exekúciu úloh a transformácií. [20, Use Carte Clusters] Pre využitie je potrebné nadefinovať *clustering schema*,



Obr. 2.3: Použitie partitioning a kópií kroku [20, Partitioning data]

čo je zloženie viacerých Carte serverov, ktoré sú označované ako *master* a *slave* servery. Môže existovať iba jeden master server, ktorý je zodpovedný za distribúciu dát na spracovanie do svojich podriadených slave serverov. Slave servery následne vykonávajú konkrétne úlohy.

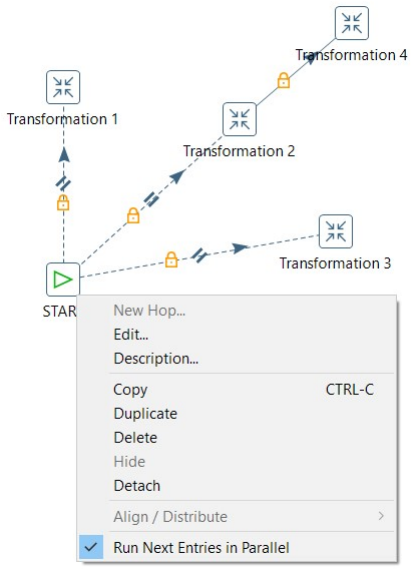
Jednotlivé kroky môžu byť označené, aby bežali paralelne na viacerých serveroch pomocou možnosti „Clusters...“. Následne sa v dialógovom okne zvolí, aká *clustering schema* sa má používať. Transformácia s aspoň jedným krokom označeným ako *clustered* sa nazýva *clustered transformation*. Graficky v Spoon je takýto krok označený, podobne ako pri kópiách kroku, označením „Cx4“ vľavo hore.

Kroky označené ako *clustered* sa vykonávajú na jednotlivých slave serveroch a ostatné kroky, ktoré sa neparalelizujú, sú vykonávané na master serveri. Preto si PDI vytvára kópie transformácie upravené podľa toho, na ktorom serveri sa vykonávajú. Týmto upraveným typom transformácií sa hovorí *metadata transformations*. [21, s. 417 – 419, 421 – 422]

Bežne je potrebné vopred definovať počet slave serverov a ide tak o statické rozdelenie podľa konkrétneho behu transformácie. *Dynamic clustering* umožňuje to, že jednotlivé slave servery sa ku master serveru samovoľne prihlasujú a tým pádom je možné paralelizovať dáta podľa počtu aktuálne dostupných slave serverov. [20, Use Carte Clusters]

2.3.4 Paralelizácia v úlohe

Kroky úlohy majú presne dané poradie vykonávania, ale môžu byť taktiež paralelizované. Keďže kroky úlohy väčšinou nemajú tendenciu byť opakovateľné, tak sa neparalelizujú vytvorením kópií jedného kroku, ale paralelne sa spúšťajú rozdielne kroky. Kroky v úlohe vieme spustiť naraz tak, že na predošlom kroku zvolíme možnosť „Run Next Entries in Parallel. . .“. Pri spustení hlavnej úlohy je potom každý paralelne spúšťaný krok úlohy vykonávaný v odlišnom vlákne. Príklad na obrázku 2.4 ukazuje grafické zobrazenie paralelizácie viacerých krokov úlohy v nástroji Spoon. Transformácie č. 1, 2 a 3 sú spúšťané paralelne tak, ako je to naznačené prerušovanou čiarou a znakom rovná sa na tejto čiare. Transformácie č. 2 a 4 sú spúšťané v rovnakom vlákne. [21, s. 411]



Obr. 2.4: Ukážka paralelizácie v úlohe [21, s. 411]

2.4 Logovanie

Takmer každá komponenta PDI dokáže logovať priebeh spracovávania vo forme riadkov textu. Každý riadok logu obsahuje dátum a čas, názov komponenty a logovaciu informáciu oddelenú pomlčkami.

```
2023/02/28 10:36:29 - Step name.0 -
    Finished processing (I=0, O=0, R=0, W=25, U=0, E=0)
```

Výpis kódu 2.1: Jeden riadok PDI logu

Je možné zvoliť rôzny level logovania postupne od zákazu logovania, cez minimálne a štandardné logovanie až po detailné alebo debug logovanie. Tento level sa dá nastaviť v každej transformácii alebo úlohe osobitne, prípadne pri spúšťaní je možné zvoliť globálny level logovania.

Na logovanie je využívaný centrálny buffer. V ňom sa pre každú komponentu udržuje *log channel ID*. Toto ID predstavuje náhodne generovaný (kvázináhodne) reťazec, ktorý identifikuje PDI komponentu, v ktorej bola logovacia informácia vytvorená. Centrálny buffer si udržuje informácie aj o rodičovských *log channel ID*, čím je umožnené vytvárať logovaciu hierarchiu, resp. hierarchiu spúšťania komponent. Logy je možné vypisovať na štandardný výstup, ukladať do súboru alebo do tabuliek. [21, s. 363 – 366]

2.4.1 Logovanie do tabuliek

V prípade veľkého množstva spúšťaných úloh sa môže logovací text veľmi predĺžovať a začína byť ľahko neprehľadný. Aj z tohto dôvodu je v PDI možné logovacie informácie ukladať aj do databázových tabuliek. Sú dostupné tieto typy logovacích tabuliek

transformation logovanie informácií z transformácie

step logovanie jednotlivých krokov transformácie

job logovanie informácií z úlohy

job entry logovanie jednotlivých krokov úlohy

logging channels informácie o log channel ku každej spustenej komponente

performance informácie o výkone transformácie pre ďalšiu analýzu

metrics ukladanie informácií o vopred sledovaných metrikách transformácie

Pre každú tabuľku je potrebné mať definovanú fyzickú tabuľku v databáze a pre každú úlohu a transformáciu je potrebné nastaviť pripojenie do logovacej databázy a názov fyzických tabuliek. Taktiež je možné si pre každú tabuľku zvoliť, aké atribúty budú do fyzických tabuliek ukladané počas behu úlohy/transformácie. Tieto metadáta je možné nastaviť aj globálne v premenných definovaných v *kettle.properties* súbore (2.2). Napríklad pre nastavenie logovacej tabuľky na logovanie transformácií stačí nastaviť premenné:

- *KETTLE_TRANS_LOG_DB* – názov databázového pripojenia obsahujúci fyzickú tabuľku,
- *KETTLE_TRANS_LOG_SCHEMA* – názov schémy v ktorej sa nachádza fyzická tabuľka,
- *KETTLE_TRANS_LOG_TABLE* – názov fyzickej tabuľky.

Obdobne je možné tieto metadáta nastaviť aj pre ostatné typy tabuliek. V názve premennej sa zmení reťazec *TRANS* na potrebný názov (*STEP*, *JOB*, *JOBENTRY*, *CHANNEL_LOG*, *TRANS_PERFORMANCE*, *METRICS*). V globálnych premenných sa dá nastaviť aj interval zápisu do tabuliek, obmedzenie počtu uchovávaných riadkov v tabuľke, obmedzenie dĺžky uchovávania logov atď. [21, s. 367 – 369]

2.4.1.1 Atribúty logovacích tabuliek

Pre každú logovaciu tabuľku existuje v Spoon rozdielne dialógové okno, v ktorom je možné zaškrtnúť, aké atribúty budú pre konkrétnu komponentu získavané. Atribúty obsahujú počet spracovávaných riadkov dát, status spracovania, čas začiatku a konca spracovania, informácie potrebné k vytváraniu *job log channel*, názov komponenty, logovací text atď.

Je popisované nastavenie spojené s logovaním transformácie. Ukážka nastavení je na obrázku 2.5. Dialógové okno obsahuje polia na definovanie pripojenia, obmedzení logovania a následne jednotlivé atribúty tejto komponenty s ich názvom (*Field name*), vysvetlením (*Field description*) a pri niektorých atribútoch je možné zvoliť aj konkrétny krok, ku ktorému sa atribút bude vzťahovať (*Step name*). Keďže ide o logovanie transformácie tak napr. pri *Field name* s názvom *LINES_READ* je potrebné špecifikovať, ktorý krok je reprezentatívnym krokom tejto transformácie a majú z neho byť zapisované informácie do logovacej tabuľky. Podobne vyzerajú aj nastavenia logovania úlohy. Pri logovaní kroku sa nastavenie *Step name* vypúšťa, keďže nemá opodstatnenie. Nastavenie „Include ?“ umožňuje zaškrtnutím vybrať, ktoré atribúty sa majú pri spustení logovať.

Atribúty, ktoré si vyžadujú presnejšie vysvetlenie sú:

- *ID_BATCH/ID_JOB* – číslo, ktoré je inkrementálne generované pri spustení úlohy/transformácie. Všetky kroky, ktoré boli v tejto úlohe/transformácií spustené si toto číslo prevezmú. Je tak možné spojiť, kde boli aké kroky spustené. Neexistuje ale spojenie tohto ID medzi úlohami a transformáciami. Je možné spojiť len transformáciu s krokmi transformácie a úlohu s krokmi úlohy.

- CHANNEL_ID – ide o kvázi-náhodný reťazec, ktorý si každá komponenta pri spustení sama vygeneruje. V jednotlivých logovacích tabuľkách má každý záznam svoje *CHANNEL_ID* a zároveň sú všetky informácie o log channel ukladané do logovacej tabuľky *logging channels*. Pomocou nej a jej atribútov *ROOT_CHANNEL_ID* a *PARENT_CHANNEL_ID* je možné zostavovať hierachiu spúšťaných komponent.

- *STATUS* – stav spracovania, ktorý môže nadobúdať stavy start (spustené), end (úspešne ukončené), stop (ukončené s chybou).

- *LOGDATE* – čas úpravy tohto logovacieho záznamu. Ak má transformácie status „end“ tak ide o čas ukončenia.
- *REPLAYDATE* – ide o čas začatia spracovania úlohy/transformácie. [21, s. 369 – 370, 372 – 373]

Niektoré informácie boli doplnené z informácií v nástroji Spoon, prípadne z pozorovaní logovaných informácií.

Obr. 2.5: Dialógové okno pre nastavenie logovania transformácie

2.5 Kettle Java API

Java Application Programming Interface (Java API) predstavuje kolekciu balíkov, tried a rozhraní, ktoré je možné využívať vo vlastných Java programoch. Výhodou je znovuvyužívanie kódu a riešení problémov, ktoré sa často opakujú z iných softvérových projektov. [22]

PDI na tvorbu a beh úloh alebo transformácií v iných Java aplikáciách poskytuje Kettle Java API. Pre využitie preddefinovaných tried je potrebné si do svojho projektu vložiť potrebné závislosti. Ide o .jar súbory, ktoré používa aj samotné PDI pri spúšťaní úloh a transformácií. Na použitie ich stačí prekopírovať z oficiálneho inštalačného priečinku[6] do vlastného projektu a pridať do classpath[7]. V článku v PDI dokumentácií [20, Embed and Extend PDI] sú k tejto funkcionalite dostupné všetky informácie. Taktiež je možné využiť,

[6] `https://www.hitachivantara.com/en-us/products/dataops-software/data-integration-analytics/pentaho-community-edition.html`

[7] cesty v ktorých Java kompilátor a Java Virtual Machine hľadajú závislosti programu

že PDI je open-source nástroj distributovaný pod licenciou LGPL[8], ktorá umožňuje ľubovoľne rozširovať zdrojový kód na vytvorenie vlastných verzií tohto nástroja. [20, Embed and Extend PDI]

Kettle Java API sa delí na štyri základné časti, kde každá z častí je reprezentovaná vlastným .jar súborom. Týmito časti sú:

- Core (kettle-core) – jadro PDI – definícia úloh a transformácií, úpravy krokov, dynamické vytváranie PDI súborov,
- Database (kettle-dbdialog) – nastavenie a používanie databázových pripojení,
- Engine (kettle-engine) – triedy spojené s behom aplikácie (runtime),
- Graphical User Interface (kettle-ui-swt) – triedy spojené s grafickým rozhraním Spoon. [21, s. 571]

Konkrétna definícia balíkov a tried je dostupná v javadoc dokumentácií[9].

[8] `https://www.gnu.org/licenses/old-licenses/lgpl-2.0.en.html`
[9] `https://javadoc.pentaho.com/`

Kapitola 3

Dátový sklad ČVUT

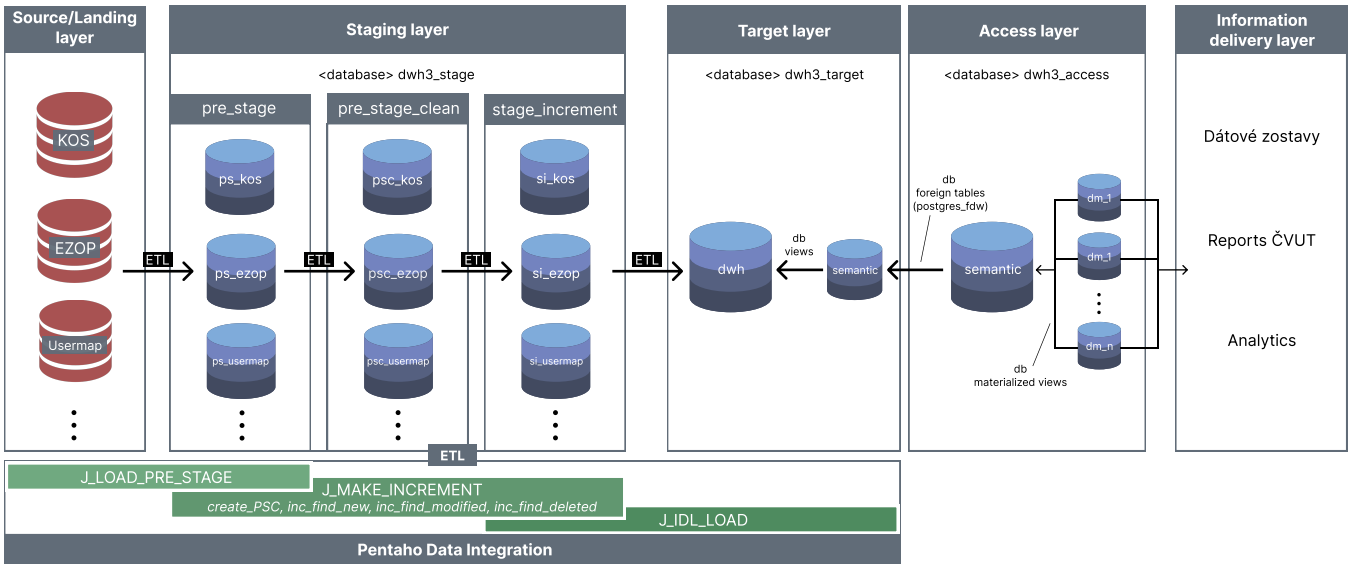
Táto kapitola sa bude zaoberať analýzou aktuálneho stavu Dátového skladu ČVUT a to konkrétne popisom využívanej architektúry databázy a popisom ETL procesov dátového skladu.

Dátový sklad ČVUT (DW ČVUT) slúži ako podpora pre rozhodovacie procesy univerzity ČVUT a jej súčastí. V súčasnosti integruje viacero zdrojových systémov a poskytuje výstupy na základe požiadaviek zamestnancov univerzity. V postupe času existovalo viacero implementácií architektúr DW ČVUT. Od prvej architektúry definovanej v diplomovej práci Ing. Stanislava Kuznetsova [23] až po aktuálnu verziu, interne označovanú ako DWH3, ktorá vznikla v diplomovej práci Ing. Roberta Kotláře [24]. Postupným vývojom bola ale architektúra upravovaná a najaktuálnejším popisom dátového skladu je ten z bakalárskej práce Bc. Adama Makaru [25].

Aktuálne DW ČVUT integruje tieto zdrojové systémy: EZOP[10], V3S[11], KOS[12], UserMap[13], Projects[14], Anketa[15], Grades[16] a Úvazky. Integrácia v source systems layer prebieha pomocou priameho pripojenia na zdrojové databázy. Hlavnou doménou, ktorú sklad integruje je doména štúdium. Často sledované dáta sú napríklad počty prihlášok, úspešnosť študentov v predmetoch, úspešnosť študijných programov v ukazateľoch pre VZOČ[17], informácie o záverečných prácach atď. Ďalším dôležitým výstupom je aj tvorba vedeckého hodnotenia na základe bodov za vedeckú činnosť alebo aj export informácií na fakultné webové stránky pre Fakultu informačních technologií a Fakultu elektrotechnickú.

[10] `https://ezop.cvut.cz`
[11] `https://v3s.cvut.cz`
[12] `https://kos.cvut.cz`
[13] `https://usermap.cvut.cz`
[14] `https://projects.fit.cvut.cz`
[15] `https://anketa.cvut.cz`
[16] `https://grades.fit.cvut.cz`
[17] Výroční zpráva o činnosti

Na obrázku 3.1 je znázornená architektúra DW ČVUT v spojení architektúry databázy a využívaných ETL procesov. Jednotlivé časti architektúry budú podrobnejšie vysvetlené v nasledujúcom texte.



Obr. 3.1: Model architektúry Dátového skladu ČVUT [24]

3.1 Databázová architektúra

Architektúra DW ČVUT vychádza zo súčasnej architektúry dátových skladov, popisovanej v sekcii 1.1.3, upravenej historickým vývojom dátového skladu. [26] Postupne budú jednotlivé časti DW ČVUT začlenené do častí súčasnej architektúry dátových skladov. Databáza dátového skladu beží na databázovom systéme PostgreSQL a je rozdelená na tri hlavné vrstvy – stage, target, access. Každá vrstva je reprezentovaná oddelenou databázou. [25]

3.1.1 Stage vrstva

Táto vrstva je reprezentovaná databázou *dwh3_stage* (pre testovacie prostredie ekvivalentná *dwh3_test_stage*). V zmysle súčasnej architektúry dátových skladov ide o staging layer. Databáza obsahuje tieto typy schém:

- pre_stage (PS) – obsahuje odtlačok dát zo zdrojového systému, ktorý sa vytvára pri extrahovaní v prvej časti ETL. Názvy tabuliek sú rovnaké ako názvy v zdrojovom systéme. V prípade, že sa na tabuľke vykonáva čistenie v pre_stage_clean, tak je názov doplnený o postfix *_orig*.

- pre_stage_clean (PSC) – obsahuje dáta vyčistené o nezrovnalosti akými sú napr. už na zdroji neplatné dáta alebo duplikáty (nie každá integrovaná tabuľka je obsiahnutá v pre_stage_clean).
- stage_increment (SI) – obsahuje posledný odtlačok dát pred aktuálnym nahrávaním dátového skladu. Na základe porovnania dát v stage_increment a pre_stage sa vytvárajú príznaky podľa, ktorých sa následne riadok upravuje aj v centrálnej databáze (target vrstve). Na to sú pridané technické atribúty:
  - *md5* – md5 hash celého záznamu, ktorý sa používa na porovnávanie záznamu medzi pre_stage/pre_stage_clean a stage_increment (atribút *md5* je prítomný aj v tabuľkách v pre_stage a pre_stage_clean),
  - *insertion_date* – dátum prvého vloženia záznamu,
  - *last_update* – posledná zmena záznamu,
  - *active* – označuje, či je záznam ešte aktívny – ak bol v zdrojovom systéme zmazaný, je označený hodnotou 0, inak 1,
  - *state* – záznam môže byť po vykonaní porovnania s pre_stage/pre_stage_clean buď nový, upravený alebo zmazaný. Podľa toho sa pre každý záznam vyplní do tohto atribútu príznak N (nový), M (upravený) alebo D (zmazaný).

Popis rozdelenia dátového skladu bol prevzatý z [26] a ďalšie informácie boli doplnené zo znalostí autora.

V databáze *dwh3_stage* sa tieto typy schémat reprezentujú prefixami ps_, psc_ a si_ a to pre každý zdrojový systém samostatne. Pre príklad, zdrojový systém KOS má v stage vrstve schémy: *ps_kos*, *psc_kos* a *si_kos*.

3.1.2 Target vrstva

Integrated data layer je v DW ČVUT reprezentovaný v target vrstve, konkrétne v schéme *dwh*. Target vrstva má taktiež vlastnú oddelenú databázu a tou je *dwh3_target* (pre testovacie prostredie ide o *dwh3_test_target*). Nachádzajú sa v nej aj schémy *ciselniky* a *semantic*, kde schéma *semantic* predstavuje semantic layer databázových skladov. Jednotlivo:

- schéma *dwh* – predstavuje tzv. centrálnu databázu a obsahuje všetky hlavné tabuľky dátového skladu. Atribúty jednotlivých tabuliek môžu využívať historizáciu typu SCD0, SCD1 alebo SCD2. Pre zabezpečenie SCD2 je pridaných 5 technických atribútov, a to:
  - *tk* – technický kľúč, ktorý predstavuje primárny kľúč tabuľky,
  - *version* – verzia záznamu spojená s konkrétnym biznis kľúčom tabuľky, inkrementuje sa od 1,

  - *date_from* – začiatok platnosti záznamu – pri prvotnom nahratí celej tabuľky by sa mala nastavovať na dátum 1. 1. 1900 (tzv. záporné nekonečno), potom už podľa hodnoty *insertion_date* v prípade nového záznamu alebo *last_update* v prípade zmeny záznamu (hodnoty zo stage_increment),
  - *date_to* – koniec platnosti záznamu – bežne sa nastavuje na 31. 12. 2199 (tzv. kladné nekonečno), pri zmazaní záznamu sa vyplní aktuálnou časovou pečiatkou, pri zmene sa vyplní hodnotou *last_update* zo stage_increment,
  - *last_update* – posledná zmena záznamu, pri novom zázname sa nastavuje rovnaká hodnota ako hodnota atribútu *date_from*, pri zmene sa nastaví ako hodnota atribútu *date_to*.

- schéma *semantic* – slúži pre agregáciu viacerých tabuliek a pre vytvorenie tzv. jednotnej pravdy používanej v semantic layer. Sú tu vytvorené databázové pohľady (views), ktoré predstavujú obchodné entity nad tabuľkami centrálnej databázy (schéma *dwh*).

- schéma *ciselniky* – je používaná k upresneniu dát používaných v rámci dátového skladu. Nachádzajú sa tu napríklad kódy štátov a miest, kódy používané v školstve atď. V týchto tabuľkách nedochádza k častým zmenám a preto nie je nutné ich nahrávať s pravidelným intervalom.

Popis bol prebraný z [25] a informácie upresnené podľa aktuálneho stavu dátového skladu. V texte budú tabuľky zo schémy *dwh* nazývané buď ako IDL tabuľky, alebo ako target tabuľky.

Medzi databázami *dwh3_target* a *dwh3_access* (target a access vrstvy) existuje prepojenie pomocou technológie postgres_fdw[18]. Pomocou používateľa *semantic_agent* sú prepojené rovnomenné schémy *semantic* v týchto databázach. Databázové pohľady z target vrstvy sú v access vrstve reprezentované ako cudzie tabuľky (foreign tables). Výhodou takéhoto prepojenia je väčšia bezpečnosť dát v centrálnej databáze, pretože externí používatelia sa pripájajú len do vrstvy access, t.j. nepripájajú sa priamo ku všetkým dátam dátového skladu. [26]

3.1.3 Access vrstva

Táto vrstva predstavuje prístupovú vrstvu dátových skladov (access layer). Pripájajú sa k nej používatelia z dôvodu čerpania dát vo forme SQL dotazov alebo vo forme reportingových nástrojov. V DW ČVUT je reprezentovaná oddelenou databázou *dwh3_access*, ktorá obsahuje schémy tvoriace dátové trhy (datamarts). [25]

[18] Postgres Foreign Data Wrapper

Dátové trhy sú uložené pomocou materializovaných pohľadov, ktoré čerpajú z cudzích tabuliek uložených v schéme *semantic*. Dáta v dátových trhoch sú teda čerpané priamo z centrálnej databázy. Z povahy fungovania materializovaných pohľadov vzniká kratšia doba odozvy pri dotazovaní sa na dáta. [26]

3.1.4 Information delivery layer

Pre dodanie dátových výstupov DW ČVUT aktuálne používa buď výstupy pomocou Excelu, alebo dodanie dát pomocou endpointov. Endpoint je tzv. koncový bod, ktorý slúži pre prístup k dátovým trhom skladu. Jeden endpoint môže poskytovať aj viacero dátových trhov. Endpointy sú reprezentované aj na webovej stránke Reports[19] vyvíjanej tímom dátového skladu. [25] Na webe Reports dátový sklad poskytuje výstupy vo formáte CSV, ale aj rôzne iné výstupy dostupné pomocou HTTP požiadaviek na konkrétne URL. Taktiež sú dostupné reporty vo forme grafov alebo tabuliek. Tieto dáta sú sprístupňované podľa roly osoby na ČVUT a sú čerpané z dátových trhov dostupných v access vrstve.

Dátový sklad taktiež poskytuje výstupy hodnotenia vedecko-výskumnej činnosti pre aplikáciu Bobr[20]. Sú vytvárané aj výstupy pre výročnú správu o činnosti univerzity alebo fakúlt, inštitucionálne akreditácie alebo rôzne na mieru tvorené analýzy a reporty podľa zadaných požiadaviek.

3.2 ETL procesy

V DW ČVUT sa pre procesy spojené s ETL používa open-source nástroj PDI popisovaný v kapitole 2. V tejto sekcii bude analyzovaný a popísaný aktuálny stav ETL procesov.

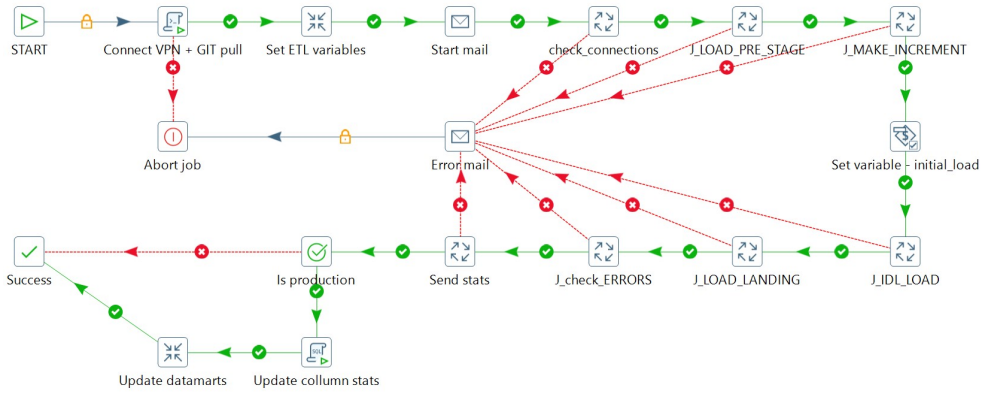
Aktuálne je nahrávanie dátového skladu spúšťané automaticky podľa určeného harmonogramu. Pri spustení nahrávania je spustená hlavná PDI úloha s názvom *J_DWH_routine*. Hlavná úloha postupne obsahuje:

1. stiahnutie zmien z git repozitára (`git pull`),
2. nastavenie premenných používaných v nahrávaní,
3. overenie pripojení na zdrojové systémy
4. skopírovanie dát zo zdrojových systémov do pre_stage (úloha *J_LOAD_PRE_STAGE*)
5. tvorba pre_stage_clean (prvotné čistenie dát) a následne identifikácia zmenených záznamov v stage_increment (úloha *J_MAKE_INCREMENT*)

[19] `https://reports.fit.cvut.cz`
[20] `https://bobr.fit.cvut.cz`

6. nahratie dát do IDL (target vrstva) s využitím logiky pre historizáciu záznamov (úloha *J_IDL_LOAD*)
7. nahratie tabuliek bez historizácie, t.j. skopírovanie zdrojových dát priamo do IDL (úloha *J_LOAD_LANDING*)
8. kontrola chýb počas nahrávania a odoslanie štatistík z nahrávania
9. v prípade produkčného nahrávania, obnovenie dát v dátových trhoch (dostupných v access vrstve)

V jednotlivých úlohách sú vnorené ešte ďalšie úlohy, ktoré postupne nahrávajú dáta podľa zdrojových systémov a podľa nahrávaných tabuliek. Počas celého procesu nahrávania sa používajú informačné e-maily pre oznámenie postupu v procese nahrávania alebo upozornenia na neúspešné nahrávanie v prípade chyby. E-maily sú odosielané celému tímu dátového skladu. Hlavná úloha je zobrazená na obrázku 3.2 v grafickom zobrazení v PDI.



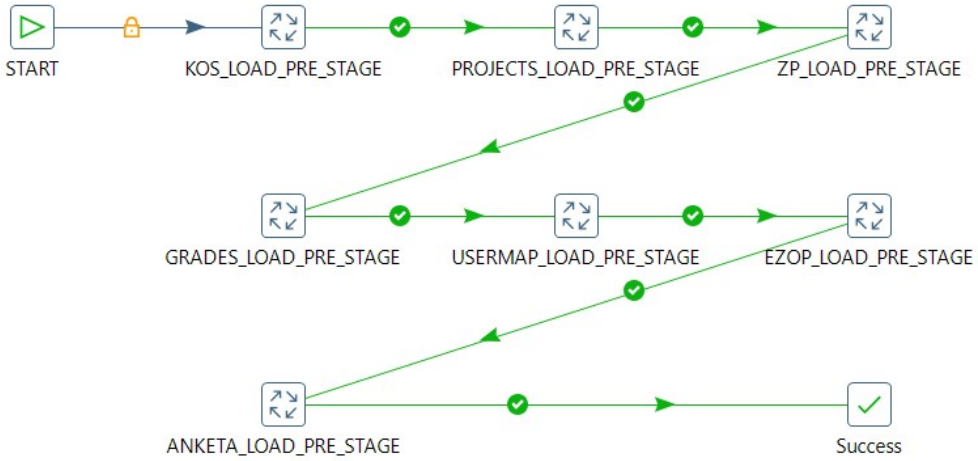
Obr. 3.2: Graf úlohy *J_DWH_routine* v PDI

Z pohľadu nahrávania sú najdôležiejšie úlohy: *J_LOAD_PRE_STAGE*, *J_MAKE_INCREMENT* a *J_IDL_LOAD*. Nahrávanie týchto úloh nesmie byť v inom poradí než je definované v ETL grafe z dôvodu procesov, ktoré súvisia s historizáciou dát v tabuľkách. Tieto úlohy v sebe obsahujú transformácie na kopírovanie a zmeny dát a taktiež používajú PL/pgSQL procedúry definované v schémach DW ČVUT, ktoré majú na starosti hlavne procesy spojené s historizáciou tabuliek, ale spracovávajú aj štatistiky nahrávania alebo iné úlohy.

3.2.1 Obsah J_LOAD_PRE_STAGE

Táto úloha zodpovedá za extrahovanie dát z tabuliek v zdrojových systémoch do tabuliek v schémach dátového skladu s prefixom ps_ (pre_stage), ktoré sú

odtlačkom dát zo zdrojového systému. Pre každý zdrojový systém je vytvorená samostatná úloha (s názvom *J_<názov-systému>_LOAD_PRE_STAGE*), v ktorej je pre každú tabuľku definovaná úloha (s názvom *<názov-tabuľky-v-zdroji>_CHECK_JOB*) vykonávajúca transformácie potrebné ku skopírovaniu dát zo zdroja. Úlohy extrahujúce dáta z tabuliek sú spúšťané jednotlivo za sebou. Pre príklad je popísané nahratie dát z tabuľky *tekns* zo zdrojového systému KOS a celý proces nahrávania pre túto tabuľku do jej reprezentatívnej tabuľky v target vrstve – *t_orgj_organizacni_jednotka*.



Obr. 3.3: Graf úlohy *J_LOAD_PRE_STAGE* v PDI

TEKNS_CHECK_JOB je úloha extrahujúca dáta zo zdrojovej tabuľky do tabuľky v dátovom sklade v schéme *ps_kos*, konkrétne *tekns_orig*[21]. Táto úloha obsahuje dve transformácie:

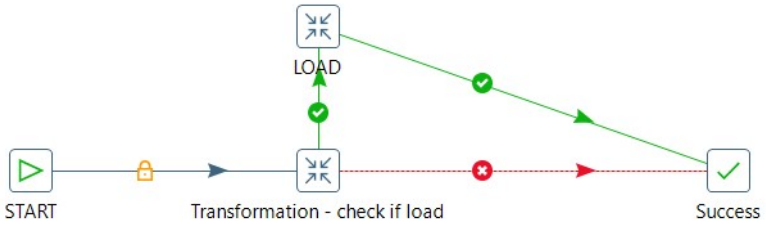
- *TENKS_CHECK* – overí pomocou databázovej funkcie *fc_should_table_be_loaded_to_ps()* to, či sa v tomto nahrávaní má spúšťať transformácia pre extrahovanie dát do pre_stage (transformácia s postfixom *_LOAD*) – je totiž možné niektoré tabuľky nahrávať menej pravidelne než pri každom spustení celého nahrávania,
- *TEKNS_LOAD* – transformácia pre extrahovanie dát tejto zdrojovej tabuľky do jej odpovedajúcej pre_stage tabuľky
  - pomocou SQL dotazu do zdrojového systému sú získané aktuálne dáta z tabuľky *TEKNS*,
  - pre každý riadok je spočítaný md5 hash spojením obsahu všetkých atribútov do jedného textového reťazca a následným zahashovaním,

[21] postfix _orig sa pridáva ak sú dáta čistené pomocou pre_stage_clean

– je pridaný nový atribút *md5*, do ktorého je pre každý riadok vložený vypočítaný hash,

– pôvodné dáta v tabuľke *tekns_orig* sú nahradené aktuálnymi dátami zo zdroja

– niektoré transformácie ešte upravujú dátové typy alebo formáty (napr. formát dátumu)



Obr. 3.4: Graf úlohy *TEKNS_CHECK_JOB* v PDI

3.2.2 Obsah J_MAKE_INCREMENT

Táto úloha spracúva nahrávanie dát do schém pre_stage_clean a stage_increment popisovaných v sekcii 3.1.1. Nejde o transformácie vykonávané priamo v PDI, ale sú použité PL/pgSQL procedúry definované v dátovom sklade.

Najskôr je spustený krok *clean stage*, ktory zaradom spúšťa procedúry:

- *create_clean_pre_stage()* – zabezpečí spustenie všetkých procedúr pre vytváranie pre_stage_clean tabuliek, ktoré sú definované v dátovom sklade,
- *inc_clear_state_flag()* – pre všetky tabuľky v schémach s prefixom *si_* nastaví všetkým riadkom tabuľky atribút *state* na hodnotu `NULL`.

Pre lepšiu predstavu bude popísané, čo sa stane s tabuľkou *tekns*. Zavolanie prvej procedúry spôsobí zavolanie procedúry *psc_kos.create_clean_pre_stage()* (a tiež ostatných rovnakých procedúr pre iné zdrojové systémy) a táto procedúra zavolá všetky procedúry na vytvorenie odpovedajúcich pre_stage_clean tabuliek zdrojového systému KOS. Medzi nimi aj *psc_kos.create_tekns()*, ktorá vytvorí tabuľku vyčistených dát z *ps_kos.tekns_orig* do tabuľky *psc_kos.tekns*. Nie všetky zdrojové tabuľky musia používať čistenie dát v pre_stage_clean. Po tomto kroku môže byť zavolaná druhá procedúra.

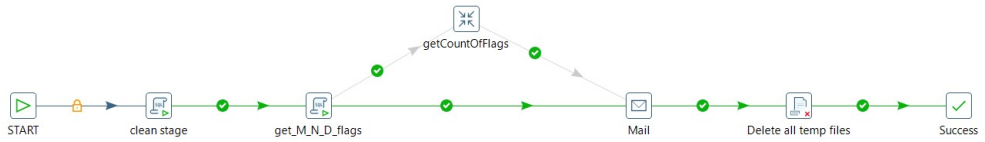
Nasleduje krok *get_M_N_D_flags*, ktorý spúšťa tri procedúry:

- *inc_find_new_in_pre_stage()* – nájdenie nových záznamov,

- *inc_find_modified_in_pre_stage()* – nájdenie upravených záznamov,
- *inc_find_deleted_in_pre_stage()* – nájdenie zmazaných záznamov.

Hľadanie zmien v zdroji spočíva v porovnávaní záznamov vždy pre konkrétnu tabuľku medzi dátami v pre_stage alebo pre_stage_clean (ak sa tabuľka čistí, tak je na porovnanie použitá pre_stage_clean) a dátami v stage_increment. Postupne sú jednotlivé záznamy označované tak ako to bolo popísané v sekcii 3.1.1. Každá z procedúr najskôr nájde názvy tabuliek, ktoré má porovnávať (všetky v schémach s prefixom si_) a až následne vykonáva porovnávanie. Momentálne teda nie je možné nájsť zmeny len pre jednu tabuľku. Podrobná implementácia procedúr je popísaná v diplomovej práci Ing. Roberta Kotláře [24].

V tabuľke *si_kos.tekns* sa po vykonaní týchto troch procedúr nachádzajú nájdené zmeny oproti aktuálnemu odtlačku dát reprezentované označením zmien v atribúte *state*. Pomocou tohto atribútu sú zmeny následne zanesené do tabuľky v IDL (*t_orgj_organizacni_jednotka*) počas nahrávania úlohy *J_IDL_LOAD*.



Obr. 3.5: Graf úlohy *J_MAKE_INCREMENT* v PDI

3.2.3 Obsah J_IDL_LOAD

Nahrávanie dát do IDL je rozdelené do nahrávania tabuliek jednotlivých domén, ktoré sú definované v dátovom sklade. Domény uľahčujú orientáciu v centrálnej databáze podľa základných dátových entít. [26] Nahrávanie domény je pre každú doménu reprezentované úlohou s názvom *J_<skratka-domény>_LOAD*. Domény sú reprezentované štvorpísmenovými skratkami a každá tabuľka v IDL z konkrétnej domény má túto skratku v názve. Domény a aj úlohy nahrávajúce tabuľky v konkrétnej doméne sú nahrávané sériovo a na poradí nahrávania väčšinou nezáleží.

Pre príklad, tabuľka *tekns* zo zdrojového systému sa nahráva v IDL do tabuľky *t_orgj_organizacni_jednotka*, kde skratka *ORGJ* predstavuje doménu *organizační jednotka*. Počas nahrávania je teda spustená úloha *J_LOAD_ORGJ*, ktorá v sebe obsahuje transformácie nahrávajúce jednotlivé tabuľky tejto domény. Transformácie nahrávajúce zmeny zistené v stage_increment sú vždy spojené z názvu tabuľky, ktorú nahrávajú a postfixu *_LOAD*. Pre doménu *ORGJ* ide o transformácie:

Obr. 3.6: Graf úlohy *J_IDL_LOAD* v PDI

- *t_orgj_organizacni_jednotka_LOAD*,
- *t_orgj_organizacni_jednotka_externi_LOAD*.

Tieto PDI transformácie transformujú dáta získané zo stage_increment vrstvy a pomocou technického atribútu *state* vyplňovaného v transformáciách v *J_LOAD_INCREMENT* nahrávajú zmeny do tabuľky v IDL. Transformácie dát môžu zahrňovať zmeny formátu, spájanie dát z viacerých stage_increment tabuliek do jednej IDL tabuľky, vyhľadávanie dát v iných stage_increment tabuľkách atď.

Na začiatku transformácie sú vybrané záznamy zo stage_increment tabuľky (napr. *si_kos.tekns*), pri ktorých bola detekovaná zmena (atribút *state* nie je `NULL`). Následne sú vykonané rôzne transformácie, ak je to vyžadované. Po transformáciách sa jednotlivé záznamy rozdeľujú do troch vetiev podľa obsahu atribútu *state*. Atribút môže nadobúdať hodnoty:

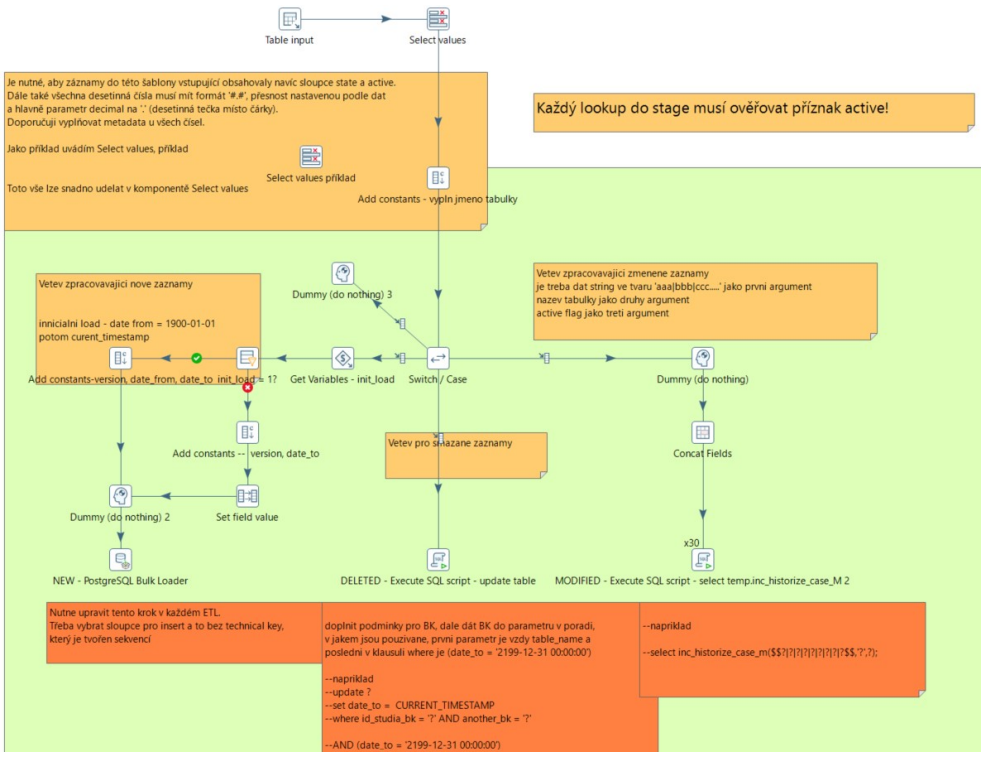
- N (new) – sú pridané hodnoty technických atribútov popísaných v sekcii 3.1.2 a dáta sú nahraté pomocou bulk loading procesu[22],

- M (modified) – pomocou databázovej procedúry *inc_historize_case_m()* sú vykonané zmeny v IDL tabuľke, ktoré predstavujú, že záznam s týmto biznis kľúčom bol v zdrojovom systéme zmenený

  – tento krok je zduplikovaný 30-krát, čiže sa nahráva 30 zmenených záznamov zároveň,

- D (deleted) – podľa biznis kľúča (kľúčov) tabuľky je nájdený odpovedajúci záznam v IDL tabuľke a jeho platnosť je ukončená nastavením atribútu *date_to* na aktuálnu časovú pečiatku.

Detailnejší popis týchto procesov je popísaný v diplomovej práci Ing. Roberta Kotláře [24]. Na obrázku 3.7 je ukážka tejto transformácie.



Obr. 3.7: Graf transformácie *t_orgj_organizacni_jednotka_LOAD* v PDI

[22]nahratie väčšieho množstva záznamov naraz

3.2.4 Pomocné procesy v J_DWH_routine

Úlohy a transformácie spúšťané v PDI sú verzované na fakultnom gitlabe[23]. Tím dátového skladu pri úpravách nahrá zmeny do vzdialeného repozitára a pred každým spustením nahrávania sú zmeny načítané v kroku *Connect VPN + GIT pull*. Tento krok obsahuje aj pripojenie pomocou VPN, aby bolo možné sťahovať dáta zo zdrojových systémov v internej sieti ČVUT. Konkrétnou implementáciou tohto kroku je shell skript uložený na serveroch dátového skladu.

V kroku *Set ETL variables* sú nastavené premenné používané v procese nahrávania – napríklad na aké e-maily sa majú odosielať notifikácie o nahrávaní alebo uloženie aktuálneho dátumu nahrávania. Počas nahrávania sa používajú aj premenné prostredia, v ktorých sú zadefinované prihlasovacie údaje ku zdrojovým databázam, ku databázam dátového skladu, ale aj to, pre aké prostredie je aktuálne nahrávanie spustené (testovacie/produkčné).

Pre zasielanie štatistík z nahrávania v kroku *Send stats* sú použité python skripty spolu s databázovými procedúrami. Databázové procedúry vygenerujú štatistiky o počte záznamov označených ako zmenených zo stage_increment tabuliek a o reálnom počte dotknutých záznamov v IDL. Python skript následne porovná tieto čísla a vytvorí zoznam tabuliek, pre ktoré sa čísla líšia. Všetky výstupy sú pridané k e-mailu o úspešnom dokončení nahrávania.

3.3 Iné procesy a funkcionality

Webová aplikácia pre monitoring Dátového skladu ČVUT slúži na automatické spúšťanie nahrávania, monitorovanie ETL procesov dátového skladu, zobrazuje aktuálny stav nahrávania a log nahrávania. Taktiež umožňuje konfigurovať používané premenné prostredia, spúšťať vlastné ETL transformácie a zobrazuje aj graf doby trvania predošlých nahrávaní. Bola vytvorená v rámci bakalárskej práce Ing. Ondřeja Pletichu [27] a je pravidelne využívaná tímom dátového skladu.

Rôzne iné procesy ako napríklad generovanie záloh dátového skladu alebo generovanie bodového hodnotenia vedeckej činnosti sú spúšťané automaticky pomocou nástroja *crontab*[24] na serveroch dátového skladu.

Na popis závislostí zdrojových tabuliek a tabuliek v IDL a popis stavby tabuliek a sémantických pohľadov je používana dátová logická mapa (DLM).

[23] `https://gitlab.fit.cvut.cz`
[24] `https://crontab.guru/`

Kapitola 4

Analýza možností paralelizácie ETL procesov DW ČVUT

V tejto kapitole sú spolu so súbežnou bakalárskou prácou kolegyne Kristiny Zolochevskaiej zadefinované požiadavky na paralelizáciu ETL procesov a vybraná časť ETL procesu, ktorá bude paralelizovaná. Ďalej sú analyzované možnosti paralelizácie v nástroji PDI, či možnosti rozšírenia o program napísaný nad týmto nástrojom. Taktiež sú zhodnotené možné prístupy k riešeniu a odôvodnené voľby finálneho riešenia.

4.1 Požiadavky na paralelizáciu

Aby obe bakalárske práce vychádzali z rovnakých požiadaviek, je potrebné si ich spoločne definovať. Taktiež je potrebné si vopred ujasniť čo je cieľom oboch prác, aby bolo možné nájdené riešenia dôkazov koncepcie (POC[25]) v závere porovnať. Preto sú zadefinované následujúce funkčné a nefunkčné požiadavky (funkčné – F, nefunkčné – N).

4.1.1 Funkčné požiadavky

F1. Správa závislostí medzi úlohami

Riešenie musí poskytovať správu závislostí medzi ETL úlohami, aby bolo dodržané správne poradie nahrávania, ktoré je vyžadované databázovou architektúrou Dátového skladu ČVUT. Riešenie musí klásť dôraz na dodržiavanie závislostí, pretože v prípade nedodržania hrozia nekonzistencie v histórií dát dátového skladu.

[25] Proof of concept

F2. Nahratie jednej IDL tabuľky s historizáciou

Aktuálne je možné spúšťať iba celé nahrávanie dátového skladu, ktoré nahráva všetky tabuľky obsiahnuté v IDL. Riešenie musí umožňovať nahrať len samostatnú tabuľku alebo výčet tabuliek. Umožní sa teda oddelené nahratie pre_stage/pre_stage_clean tabuliek a historizačné procesy len pre stage_increment tabuľky potrebné ku korektnému nahratiu dát do zvolenej IDL tabuľky či tabuliek.

F3. Nahrávanie z rôznych zdrojových systémov

Riešenie musí umožňovať nahrávanie z rôznych typov zdrojových systémov. Aktuálne sú využívané iba databázové systémy PostgreSQL a Oracle, ktoré v riešení musia byť obsiahnuté. Ostatné databázové systémy musia mať možnosť byť jednoducho doplnené v prípade potreby v budúcnosti.

F4. Paralelizácia komponent s využitím dátovej alebo úlohovej paralelizácie

Jednotlivé ETL procesy, ktoré na sebe nie sú závislé by mali byť vykonávané paralelne, čomu odpovedá definícia úlohovej paralelizácie. Možnosťou je taktiež dátová paralelizácia v prípade procesov, ktoré spracovávajú veľké množstvo dát, prípadne tam, kde to dáva zmysel. Paralelizácia by mala byť vykonávaná na jednom serveri, ktorý má DW ČVUT dostupný pre ETL nahrávanie.

F5. Prehľadnosť logovania

Pri spúšťaní procesov paralelne nie je vhodné zapisovať logovacie informácie do jedného súboru z dôvodu prepínania kontextu rôznych vlákien či procesov. Je potrebné, aby riešenie zabezpečilo jednoduché dohľadanie logovacej informácie konkrétneho nahrávania jednej alebo viacerých tabuliek, prípadne kompletného nahrávania dátového skladu.

4.1.2 Nefunkčné požiadavky

N1. Škálovateľnosť paralelizácie

Riešenie bude umožňovať škálovať paralelizáciu, či už pôjde o dátovú alebo úlohovú paralelizáciu. Škálovateľnosť môže byť menená v zmysle priradzovania väčšieho alebo menšieho množstva prostriedkov pre paralelizačné procesy.

N2. Prenositeľnosť riešenia

Vytvorený POC musí byť možné spúšťať na rôznych typoch systémov z dôvodu jednoduchosti pri používaní inými používateľmi v budúcnosti.

4.2 Zvolenie paralelizovanej časti ETL

Momentálne ETL procesy nahrávajú pri každom nahrávaní väčšinu tabuliek z target databázy. Ide o zhruba 180 tabuliek v databáze *dwh3_target*, ktoré využívajú podobné množstvo tabuliek z databáze *dwh3_stage*. Keďže by implementácia celého ETL procesu bola príliš časovo náročná, bolo zvolené menšie množstvo tabuliek (časť ETL procesu), na ktorých budú prezentované navrhnuté riešenia oboch bakalárskych prác. V tabuľke 4.1 sú popísané závislosti medzi vybranými tabuľkami v stage a target databázach. Výber bol učinený tak, aby pokrýval všetky možné vzťahy v nahrávaní medzi stage a target tabuľkami - 1:1, 1:N, N:1, N:M. Taktiež boli zvolené dve tabuľky s veľkým objemom dát a častými zmenami v zdrojovom systéme (tabuľky *tcitation_affiliations, tcitation_authors*). Pre úplnosť bola pridaná aj jedna tabuľka bez historizácie, ktorá má iný režim nahrávania - nepoužíva stage tabuľky (tabuľka s koncovkou *_nohist*).

Tabuľka 4.1: Popis závislostí medzi zvolenými tabuľkami

Typ databázy zdroja	Zdrojový systém	Tabuľka v stage databáze	Tabuľka v target databáze	Využitie PSC
PostgreSQL	Grades	classification_text	t_klas_klasifikace	Nie
		classification_user classification	t_klas_klasifikace, t_klas_klasifikace_student	
		student_classification boolean_student_classification number_student_classification string_student_classification	t_klas_klasifikace_student	
Oracle	Usermap	osoby	t_osob_osoba	Áno
		tusers		Nie
	KOS	tkontakt	t_koud_adresa t_koud_email t_koud_telefonni_cislo	Áno
		tekns	t_orgj_organizacni_jednotka	
			t_orgj_organizacni_jednotka_externi	
	EZOP	torganizations		
		tcitation_affiliations tcitation_authors -	t_externiorganizacnijednotka_externicitaceautor_rel t_vvvs_externi_citace_autor t_vvvs_vedecky_vysledek_bibl_indik_nohist	-

4.3 Možnosti nástroja PDI

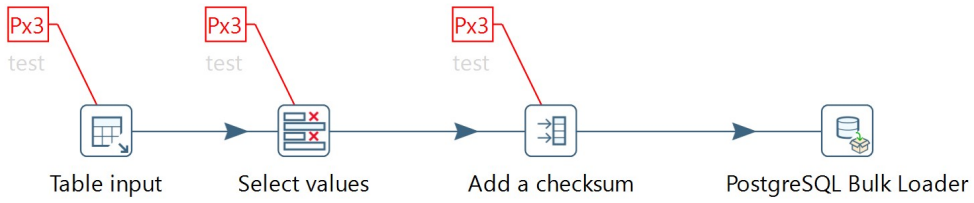
Aktuálne DW ČVUT využíva PDI vo verzii 6, pričom najnovšou verziou je verzia 9.4. Vo fungovaní nástroja sa ale medzi verziami nič výraznejšie nezmenilo. Aspoň nie čo sa týka aktívne používaných funkcionalít v dátovom sklade. Novinkou je ale napríklad prechod na vyššiu verziu jazyka Java, zvýšenie bezpečnosti či lepšia integrácia s inými Pentaho produktmi. Taktiež boli pridané nové PDI kroky pre lepšiu prácu s dátami. Tie sa ale veľakrát nachádzajú len v platenej verzii, prípadne nemajú pre DW ČVUT aktuálne pridanú hodnotu. Postupným vývojom ale bola určite zaručená vyššia stabilita nástroja a aj preto boli funkcionality analyzované v novej verzii, aby mohla byť vykonaná aktualizácia nástroja. V prípade problémov boli chyby porovnávané s nižšími verziami.

4.3.1 Dátová paralelizácia

Využitie dátovej paralelizácie v PDI bolo analyzované získavaním dát tabuľky *tcitation_affiliations* zo zdrojového systému EZOP a extrahovaním týchto dát do rovnomennej stage tabuľky v dátovom sklade. V tejto tabuľke je pri každom nahrávaní vytvorená kópia aktuálnych dát zo zdroja. Obsahuje veľké množstvo záznamov a sú v nej časté zmeny dát. Aktuálny počet záznamov v zdrojovej tabuľke sa pohybuje okolo 65,5 milióna. Postupne boli vyskúšané možnosti paralelizácie PDI transformácie vysvetlené v sekcii 2.3.1. Nahrávanie sa analyzovalo na lokálnom počítači so stabilným internetovým pripojením. Následne bola najvhodnejšia možnosť dátovej paralelizácie testovaná na serveroch dátového skladu, na ktorých bežne prebieha ETL spracovanie a porovnaná s aktuálnou dĺžkou nahrávania tejto tabuľky na serveri.

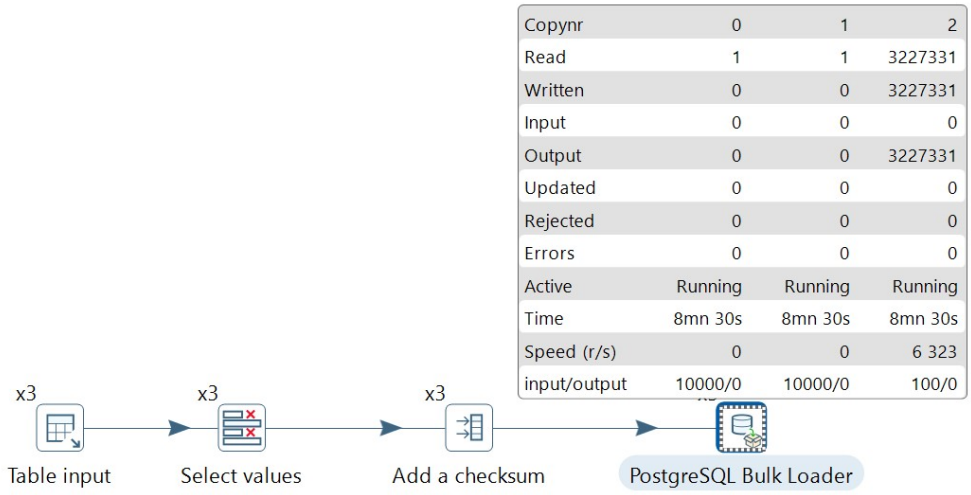
Najskôr bola pre porovnanie spustená pôvodná transformácia bez paralelizácie, ktorej nahratie by hrubým odhadom na lokálnom počítači trvalo približne 2 h 44 min. Následne bolo vyskúšané použitie partitioning (2.3.2). Pomocou partitioning schémy o troch partíciách sa nastavila možnosť delenia do partícií tak, ako je ukázané na obrázku 4.1. Takýto prístup však nepomohol, pretože delenie do partícií PDI nevykonávalo na kroku *Table input* a dochádzalo k trojnásobnému nahratiu dát. Jednotlivé kópie kroku *Table input* totiž spúšťali rovnaký SQL dotaz. Taktiež, využitie partitioning v tomto prípade nemá opodstatnenie, pretože sa tu nepoužívajú žiadne agregačné kroky, pri ktorých by jednoduché vytvorenie kópií krokov spôsobovalo problémy vo výpočtoch.

Ďalšou možnosťou bolo vytvorenie jednoduchých kópií kroku. Konkrétne boli pre každý krok zvolené tri kópie. V tomto prípade sa vyskytli hneď dva problémy. Dáta zo zdroja sa znova načítavali viackrát ale vytvorenie kópií pre krok *PostgreSQL Bulk Loader* (PBL) spôsobilo aj to, že pracovala vždy iba jedna kópia každého kroku. To je možné vidieť na obrázku 4.2, kde pri kópiách číslo 0 a 1 (copynr) je rýchlosť krokov nula riadkov za sekundu (speed (r/s)).



Obr. 4.1: Využitie partitioning pri nahrávaní tabuľky *tcitation_affiliations*

PBL totiž otvára pripojenie do databázy na skopírovanie štandardného vstupu pomocou príkazu `COPY`. Keďže každá kópia chcela vytvoriť takéto spojenie, nastalo blokovanie na úrovni databázového systému. To spôsobilo, že sa naplnili jednotlivé dátové toky medzi krokmi a postupne sa zastavovali aj predošlé kroky s rovnakým číslom kópie. Z tohto dôvodu a zároveň preto, že bulk loading je rovnako rýchly aj pre väčšie množstvo záznamov, nemá zmysel vytvárať kópie kroku PBL.



Obr. 4.2: Ukážka zaseknutia kópií kroku *PostgreSQL Bulk Loader*

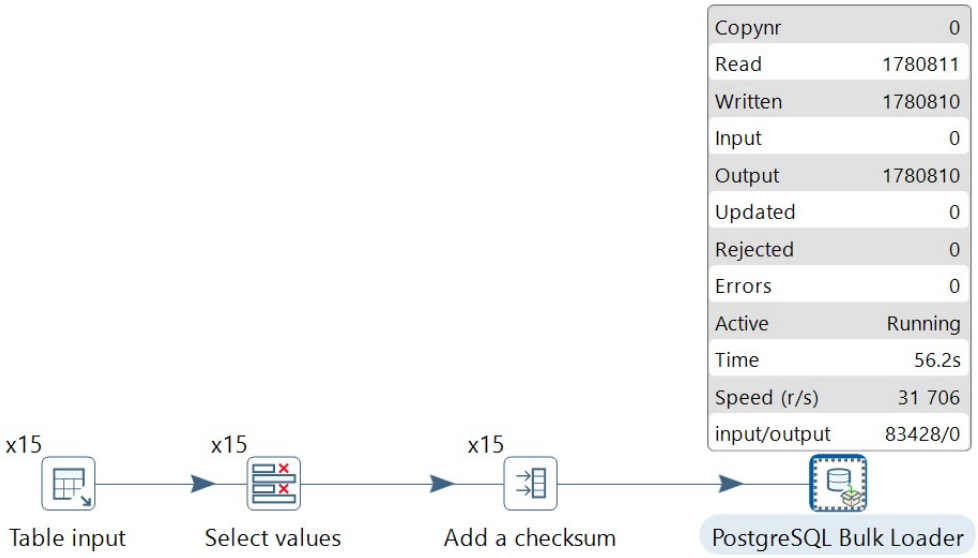
Duplikátne nahrávanie zdrojových dát bolo potrebné vyriešiť priamo v *Table input*. Bolo potrebné obmedziť SQL dotaz tak, aby každá kópia získavala unikátne dáta a zároveň, aby všetky kópie dokopy získali celý obsah zdrojovej tabuľky. Na to je vhodné deliť dáta pomocou modula, ktoré zaručí, že všetky dáta budú získané a zároveň pri vhodnom zvolení atribútu tabuľky a module vytvorí rovnomerné delenie dát do kópií. SQL dotaz bol upravený pomocou klauzule `WHERE` spolu s použitím interných PDI premenných, ktoré

```
WHERE COALESCE(
   MOD(CITATION_AFFILIATION, ${Internal.Step.Unique.Count}), 0
) = ${Internal.Step.Unique.Number}
```

Výpis kódu 4.1: Podmienka pre selektovanie dát do rôznych kópií kroku

odkazujú na počet kópií kroku a pre každú kópiu na jej poradové číslo. Klauzula `COALESCE` je prítomná pre prípad, že by funkcia `MOD` vrátila `NULL` hodnotu. Vtedy sa zamení za hodnotu 0 aby sa nestratili žiadne riadky dát pri porovnávaní s poradovým číslom kópie. Podmienka je na výpise 4.1. V kroku *Table input* je na dopĺňanie premenných potrebné zaškrtnúť možnosť „Replace variables in script?“.

Pri použití ukázanom na obrázku 4.3 interná premenná *Internal.Step.Unique.Count* obsahuje hodnotu 15, pretože ide o celkový počet kópií a premenná *Internal.Step.Unique.Number* obsahuje pre každú kópiu svoje číslo. Modulo je determinované počtom kópií kroku *Table input*. Tým pádom sa hodnoty v atribúte *citation_affiliation* modulia číslom 15. Napríklad len tie riadky dát, pri ktorých bude hodnota po zmodulení rovná 1 budú načítané v *Table input* kópií s číslom 1 (označuje sa ako *Table input.1*).

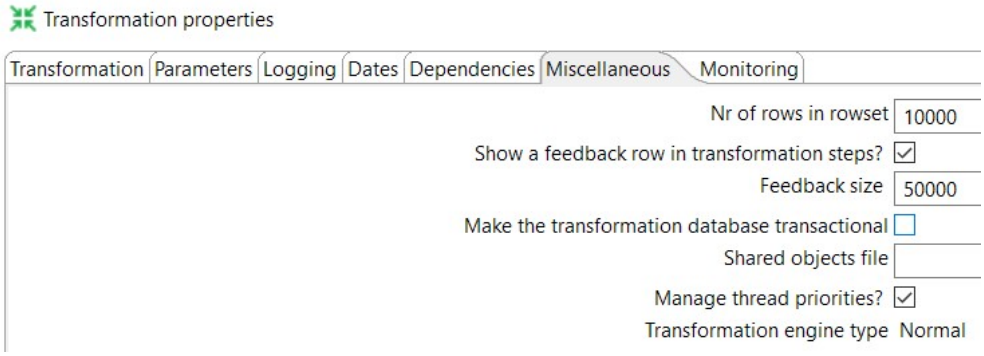


Obr. 4.3: Využitie 15 kópií a upraveného kroku *Table input* pri nahrávaní

Postupne boli takto vyriešené viaceré prekážky a jediné, čo ostávalo bolo zvoliť vhodný atribút zdrojovej tabuľky a vhodnú hodnotu modulo tak, aby každá kópia mala takmer rovnaký počet priradených riadkov, t.j. maximalizovalo sa využitie jednotlivých vlákien. To sa dosiahlo skúšaním rôznych

kombinácií atribútov a modula. V prípade ak by tabuľka neobsahovala číselné atribúty, je možné na delenie dát použiť dátumy, prípadne dĺžky reťazcov atď.

Výrazné zrýchlenie nastalo ale až po vypnutí nastavenia „Make the transformation database transactional“, ktoré spúšťa všetky pripojenia v jednej transakcií. Ak je totiž nastavenie zapnuté, tak sa znižuje priepustnosť pripojenia a pridávaním kópií krokov (t.j. vlákien na výpočet) sa objem spracovávaných dát nezvyšuje. Po vypnutí tohto nastavenia nastalo reálne zvýšenie objemu tečúcich dát a zrýchlenie extrahovania celej zdrojovej tabuľky. Stráca sa tým ale možnosť využiť v prípade chyby transformácie databázový `ROLLBACK`. Nahrávanie do pre_stage ale vytvára iba kópiu dát zo zdroja, ktorá by z predošlého úspešného nahrávania mala byť uložená v stage_increment kde sa uchováva aj história. Prípadnú chybu v transformácií stačí opraviť a zdrojovú tabuľku extrahovať znova. Vypnutie transakčného spracovania v tomto prípade nespôsobuje zásadné problémy a umožňuje urýchlenie spracovania. Ukážka tohto nastavenia v Spoon je na obrázku 4.4.



Obr. 4.4: Možnosť „Make the transformation database transactional“ v nastaveniach transformácie

V tabuľkách 4.2 a 4.3 sú porovnané rôzne prístupy k paralelizácií. Najskôr boli vykonané testy na lokálnom počítači zachytené v tabuľke 4.2 a následne najrýchlejšie prístupy overené na stabilnejšom prostredí, t.j. na serveroch dátového skladu. Keďže nahratie takého množstva záznamov by na lokálnom počítači trvalo príliš dlho, nahrávanie bolo zastavené skôr a časy celého nahrávania sú často odhadované. Pri lokálnom spúšťaní je vidno, že spomínané vypnutie transakčného spracovania spôsobilo požadované zrýchlenie. Taktiež možno pozorovať, že vyššie množstvo kópií, resp. vlákien nezaručuje rýchlejšie nahratie dát. Obmedzenie je pravdepodobne spôsobené priepustnosťou internetového pripojenia.

V tabuľke 4.3 sú zistenia overené extrahovaním všetkých záznamov zdrojovej tabuľky. Server dátového skladu má väčšie výpočtové prostriedky, preto je dĺžka nahrávania kratšia. Zo získaných dĺžok extrahovania na serveri je

Tabuľka 4.2: Pokusy dátovej paralelizácie pri lokálnom nahrávaní

Popis pokusu	Dĺžka nahratia 1 mil. záznamov (s)	Odhad nahratia 65,5 mil. záznamov (min)	Poznámky
Bez paralelizácie	150	164	
Partitioning	150	164	Chyba – duplicitné nahrávanie dát
3 kópie, aj pre PBL	150	164	Chyba – zaseknutie kópií PBL
12 kópií, okrem PBL, s transakciou	180	196,5	bez zrýchlenia
12 kópií, okrem PBL, bez transakcie	31	34	
30 kópií, okrem PBL, bez transakcie	36,5	40	

pozorovateľné, že v najrýchlejšom prípade (30 kópií) sparalelizovaním transformácie došlo k 2,7-násobnému zrýchleniu oproti riešeniu bez paralelizácie. Rozdiel predstavoval skrátenie nahrávania o 26 min 38 s, čo predstavuje takmer 64 % zrýchlenie. Je možné, že pri podrobnejšom testovaní by sa dalo dopracovať aj ku rýchlejšiemu výsledku. Išlo by avšak už iba o desiatky sekúnd. Dĺžka nahrávania je ovplyvňovaná mnohými faktormi, ktoré nie je možné zmeniť a aj preto sú tieto zistenia považované za dostatočné.

Tabuľka 4.3: Pokusy dátovej paralelizácie na serveri

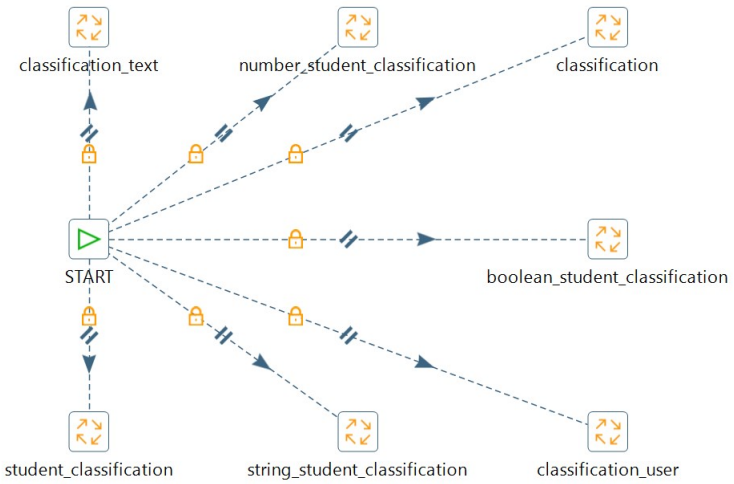
Popis pokusu	Dĺžka nahratia 65,5 mil. záznamov
Bez paralelizácie	41 min 56 s
12 kópií, okrem PBL, bez transakcie	16 min 43 s
20 kópií, okrem PBL, bez transakcie	16 min 7 s
30 kópií, okrem PBL, bez transakcie	15 min 18 s
50 kópií, okrem PBL, bez transakcie	15 min 43 s

Paralelizácia pomocou funkcionality clustering (2.3.3) nebola analyzovaná, pretože DW ČVUT aktuálne neumožňuje využiť viacero serverov na ETL spracovanie. Ide ale o dostupné riešenie, ktoré umožňuje jednoduché zvýšenie výpočtových prostriedkov v prípade budúcich inovácií.

4.3.2 Úlohová paralelizácia

Úlohová paralelizácia v PDI je reprezentovaná len mechanizmom paralelizovania za sebou následujúcich úloh z konkrétneho bodu. Ukážka na obrázku 4.5 zobrazuje paralelizáciu nahrávania pre_stage tabuliek zo zdrojového systému Grades. Tento mechanizmus ale neumožňuje komplikovanejšiu správu spúšťaných vlákien alebo procesov ani riešenie závislých chýb, ktoré môžu pri paralelnom spracovaní vzniknúť. Počas viacerých skúšok takéhoto nahrávania nastávali rôzne problémy, ktoré nemali žiadne logické prepojenie. Niekedy zlyhalo pripojenie do zdrojovej databázy pri rôznych tabuľkách, niekedy zlyhalo pripojenie k logovacím tabuľkám. Pri spustení samostatnej úlohy ale problémy nepretrvávali.

Ďalší problém, ktorý súvisí s úlohovou paralelizáciou je maximálny počet pripojení k databázam. Niektoré databázové pripojenia môžu mať obmedzený počet konkurentných pripojení, čo v prípade paralelizácie môže vyústiť do pádu nahrávania, ak sa počet konkurentných pripojení prekročí a databáza odmietne vytvoriť nové pripojenie. V PDI je možné na správu počtu otvorených pripojení využívať *database pooling*. Počas analýzy ale bolo zistené, že neobmedzuje maximálny počet pripojení. Je možné nastaviť minimálny a maximálny počet pripojení pre pool. Ak však úloha alebo transformácia používa viac pripojení, než je nastavená maximálna hodnota, tak pool čaká na vytvorenie nového pripojenia, aby ho doplnil. Nové pripojenie ale nikdy nepríde, pretože ho databáza nepovolí vytvoriť. Úlohy a transformácie tým pádom ostávajú čakať navždy. Aj pre tento problém teda PDI funkcionality úlohovej paralelizácie nie sú dostatočné a bude ich potrebné rozšíriť.



Obr. 4.5: Nahrávanie pre_stage tabuliek pomocou paralelizácie úloh v PDI

4.4 Analýza rozšírenia pomocou programu nad PDI

Na splnenie všetkých požiadaviek definovaných v 4.1 nástroj PDI nie je dostatočný. Napríklad tvorba paralelizácie jednotlivých úloh a definovanie závislostí v prípade zložitejších väzieb medzi stage a target tabuľkami by bola uskutočniteľná aj v PDI, ale jednoducho by sa komplikovala. Nemožnosť viac ovplyvňovať alebo nastavovať paralelizáciu úloh by vo väčšom merítku mohlo spôsobiť problémy s výpočtovou silou. Taktiež by bolo značne komplikované umožniť nahratie rôzneho množstva target tabuliek podľa požiadaviek používateľa. Boli teda analyzované možnosti rozšírenia pomocou aplikácie, ktorá by spravovala závislosti medzi tabuľkami a spúšťala úlohy a transformácie namodelované v PDI.

4.4.1 Možnosti využitia Kettle Java API

PDI poskytuje na rozšírenie funkcionalít alebo integrovanie do vlastného programu Kettle Java API, ktoré bolo popísané v sekcii 2.5. Na využitie je potrebné implementovať aplikáciu v jazyku Java alebo jazyku jemu podobnému, aby mohli byť využité poskytnuté Java triedy.

Bol vytvorený nový projekt v jazyku Java a do neho priložené potrebné .jar súbory, aby bolo možné triedy volať v programe. Prvotne bola použitá verzia 9.4, v ktorej bol ale spozorovaný problém s krokom *PostgreSQL Bulk Loader*, ktorý je v ETL procesoch dátového skladu často využívaný. V prípade, že sa spustila transformácia, kedy do tohto kroku neprúdili žiadne dáta (napr. zdrojová tabuľka žiadne neobsahovala), krok a ani samotná transformácia sa nikdy neukončili. Z tohto dôvodu bolo zostúpené na verziu 9.3 a následne 9.2. V oboch týchto verziách ale nastávali viaceré problémy pri inicializácií Kettle prostredia v programe.

Ako jediné riešenie sa javilo opraviť chybu vo verzii 9.4. V oficiálnom github repozitári[26] bola nájdená oprava tejto chyby. Keďže ide o samostatnú komponentu Kettle Java API tak boli opravené Java triedy prekompilované, vytvorený nový .jar súbor a ten zamenený za pôvodný .jar súbor. Po tejto oprave bolo skontrolované, či fungujú všetky aktuálne používané kroky v ETL procesoch DW ČVUT. Žiadne iné chyby neboli nájdené takže sa nakoniec bude využívať verzia 9.4.

Kettle Java API umožňuje extenzívnu prácu s PDI úlohami a transformáciami. Je možné ich spúšťať, zastavovať, zisťovať ich stav, ale napríklad je možné aj nastavovať premenné alebo parametre, zisťovať channel ID[27] či zisťovať rôzne iné informácie o komponentách. Taktiež je možné vytvárať úlohy a transformácie dynamicky, či robiť zmeny v grafickom prostredí Spoon.

[26] `https://github.com/pentaho/pentaho-kettle`

[27] potrebné pre získanie informácie o konkrétnej PDI komponente z logovacích tabuliek

4.4.2 Možnosti využitia internej DW ČVUT aplikácie

DW ČVUT aktuálne na automatické spúšťanie a správu ETL procesov využíva webovú aplikáciu implementovanú v bakalárskej práci Ing. Ondreja Pletichu [27]. Z informácií získaných z textu práce a z analýzy zdrojového kódu aplikácie je zrejmé, že aplikácia je napísaná v jazyku Python a využíva aj viacero framework riešení.

Do tejto aplikácie by bolo možné dopísať logiku, ktorá by implementovala závislosti medzi PDI úlohami a paralelizačný algoritmus. Na spúšťanie úloh a transformácií je v tomto prípade možné využiť len spúšťanie z príkazového riadku pomocou skriptov Kitchen alebo Pan (vysvetlené na začiatku kapitoly 2). Tieto skripty ale neumožňujú intenzívnejšiu prácu s parametrami alebo stavom PDI komponent. Skripty umožňujú pri spustení nadefinovať parametre, prípadne cestu ku logovaciemu súboru ale neumožňujú po spustení žiadne nové informácie spätne získať. Integrácia s PDI preto nie je taká blízka ako pri použití Kettle Java API.

4.5 Zhrnutie analýzy

Zo zistení analýzy sa rozhodlo pre tvorbu paralelizačného riešenia pomocou aplikácie napísanej nad nástrojom PDI. Využité budú ale aj niektoré funkcionality PDI. Na dátovú paralelizáciu je vhodné riešenie popísané v sekcii 4.3.1. Na správu logovania je tiež možné použiť PDI a to konkrétne logovacie tabuľky popísané v 2.4.1. Ostané požiadavky ale nie sú splniteľné len pomocou nástroja PDI.

Na rozšírenie funkcionalít nad PDI pomocou aplikácie sa využije Kettle Java API. Oproti riešeniu, ktoré bolo načrtnuté v sekcii 4.4.2 je výhodou väčšie množstvo ovplyvňovania behu PDI úloh a transformácií. Skripty Kitchen a Pan sú oproti Kettle Java API značne obmedzené v interakcií s už bežiacou komponentou. V jazyku Java a s pomocou tried získaných z API bude vytvorený algoritmus paralelizácie úloh s ohľadom na závislostí definované ETL procesmi a s ohľadom na architekúru databázy DW ČVUT.

Kapitola 5

Návrh aplikácie nad nástrojom PDI

Táto kapitola popíše návrh aplikácie, ktorá rozširuje funkcionality nástroja PDI tak, aby boli splnené všetky požiadavky definované v analytickej časti práce.

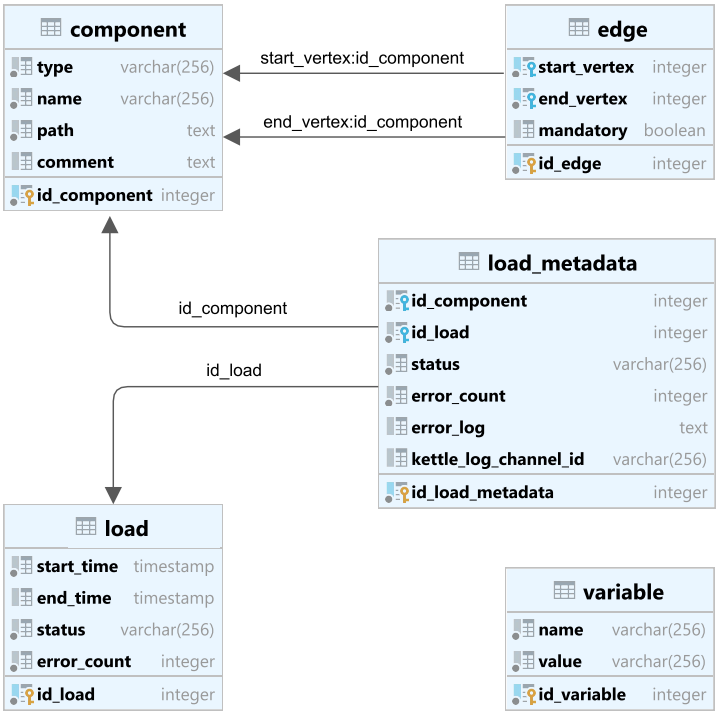
Pre splnenie požiadaviek je potrebné vytvoriť aplikáciu, ktorá bude dbať na závislosti úloh v ETL nahrávaní, mať možnosť prehľadu rôznych nahrávaní a správu logovania. Ako najvhodnejšie riešenie sa ukazuje tvorba aplikácie, ktorá bude schopná za pomoci vhodne navrhnutých databázových tabuliek tieto požiadavky naplniť. Z tohto dôvodu bola zvolená tvorba backend časti systému, ktorý bude poskytovať API pre interakciu s aplikáciou. Na API bude možné v budúcnosti nadviazať tvorbou frontend časti.

5.1 Návrh databázových tabuliek aplikácie

Databázové tabuľky budú jadrom celého riešenia. Musia ukladať informácie o spustených nahrávaniach, používaných úlohách a transformáciách, informácie pre získanie logov z PDI či premenné používané v nahrávaniach. Taktiež musí byť myslené na závislosti medzi nahrávanými úlohami a transformáciami. Je teda potrebné vytvoriť graf závislostí, ktorý to bude kontrolovať. Návrh databázových tabuliek je zobrazený na obrázku 5.1 a v následujúcom texte bude bližšie popísaný.

5.1.1 Komponenty

V návrhu sú PDI úlohy a transformácie reprezentované pod spoločným názvom komponenta V databáze ide o tabuľku *component*. Komponentou môže byť v budúcnosti aj iný typ objektu, ako napríklad spustiteľný skript či iný súbor, než len PDI špecifické súbory. Typ komponenty bude preto definovaný v atribúte *type*, ktorý bude v aplikácií reprezentovaný výčtovým typom enum, aby



Obr. 5.1: Návrh databázového modelu

bola zaručená jednotnosť názvov naprieč aplikáciou. V atribúte *path* bude definovaná cesta k súboru pre načítanie a spustenie súboru.

5.1.2 Nahrávanie a metadáta nahrávania

Pre zachovanie informácií o spustených nahrávaniach komponent bude využívaná tabuľka *load*. V tejto tabuľke bude upravovaný stav nahrávania, uvedený čas začiatku a konca nahrávania a počet chýb počas nahrávania. V tabuľke *load_metadata* budú ukladané presnejšie informácie ku konkrétnym komponentám, ktoré boli spustené v danom nahrávaní a to:

- *status* – stav nahrávania komponenty v priebehu nahrávania,
- *error_count* – počet chýb počas nahrávania komponenty,
- *error_log* – logovacia informácia v prípade chyby počas behu ak log nie je dostupný v PDI,
- *kettle_log_channel_id* – ID používané k nájdeniu logovacej informácie v logovacích tabuľkách PDI (vysvetlené v sekcii 2.4.1).

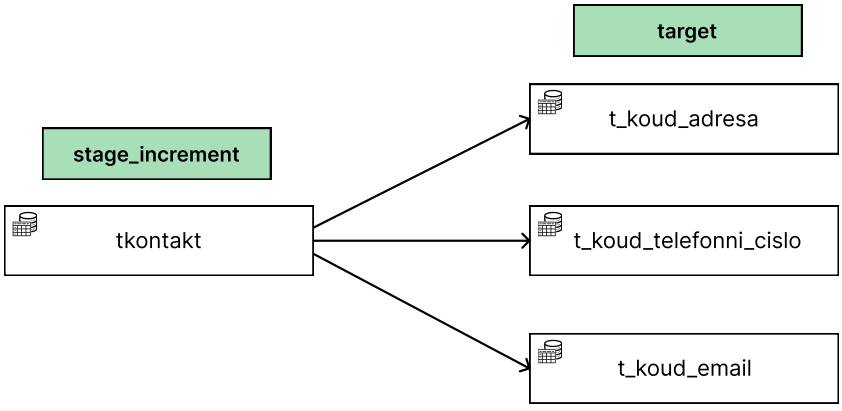
Pomocou týchto dvoch tabuliek bude možné získať, aké komponenty boli v akom nahrávaní spustené, priebeh nahrávania a stav ukončenia a taktiež vytvoriť kompletný log nahrávania pomocou prístupu ku PDI logovacím informáciám.

V komponentách môžu byť využívané aj premenné – na ich uloženie a následné nastavenie počas behu aplikácie bude slúžiť tabuľka *variable*, kde atribút *name* je názov premennej a atribút *value* jej hodnota.

5.1.3 Graf závislostí komponent

Pre určenie závislostí medzi nahrávanými komponentami bude využítá tabuľka *edge*, v ktorej jeden záznam predstavuje jednu hranu grafu. Z podstaty ETL a procesov DW ČVUT musí byť tento graf orientovaný a acyklický, bežné označovaný ako DAG[28]. Bude musieť byť zaručené, že pridávaním hodnôt do tabuľky sa nevytvorí cyklus. Jednotlivé hrany sú reprezentované atribútmi *start_vertex* a *end_vertex*, v smere zo začiatočného vrcholu (start) do koncového vrcholu (end). Vrcholmi sú komponenty z tabuľky *component*.

Atribút *mandatory* bude používaný na zaručenie konzistencie nahrávania vychádzajúceho z architektúry DW ČVUT popísanej v kapitole 3. Problém pri nahrávaní jednej target tabuľky by nastal napríklad v prípade na obrázku 5.2. Ak by požiadavkou bolo nahrať target tabuľku *t_koud_adresa*, tak zo závislostí musí byť nutne nahratá aj stage_increment tabuľka *tkontakt*. V tejto stage_increment tabuľke ale budú vypočítané zmeny dát, ktoré by sa mali prepísať aj do ďalších dvoch target tabuliek z obrázku. Ak by sa tak nestalo a v tabuľke *tkontakt* by sa pri ďalšom nahrávaní zmeny prepísali novými zmenami, tak by v tabuľkách *t_koud_telefonni_cislo* a *t_koud_email* zmeny neboli nikdy zachytené. Vznikla by nekonzistencia v histórii dát a chyby pri ďalších nahrávaniach.



Obr. 5.2: Ukážka závislosti stage_increment a target tabuliek

[28]Directed acyclic graph

Pomocou povinných (mandatory) hrán je dosiahnuté, že ak sa nahráva nejaká komponenta tak musia byť nahraté aj všetky komponenty, ktoré sú konečným vrcholom povinných hrán grafu, kde začiatočným vrcholom je nahrávaná komponenta. V prípade obrázku 5.2 by všetky tri hrany boli označené ako povinné a nahrali by sa tak aj ostatné dve target tabuľky.

5.2 Využitie PDI v návrhu

Komponentami v aplikácií budú úlohy a transformácie aktuálne používané v ETL nahrávaní dátového skladu. Keďže procesy na tvorbu pre_stage_clean a stage_increment tabuliek sú aktuálne PL/pgSQL procedúry, ktoré upravujú dáta vo všetkých pre_stage tabuľkách zaradom, bude potrebné upraviť tieto procedúry tak, aby spracovávanie vykonávali len pre jednu tabuľku, pomocou jej názvu a schémy v ktorej sa nachádza. Upravené procedúry *inc_find_new_in_pre_stage*, *inc_find_modified_in_pre_stage* a *inc_find_deleted_in_pre_stage* sú dostupné v priloženom archíve. Vytvoria sa taktiež úlohy v PDI, ktoré budú pomocou premenných spúšťať procedúry na vytvorenie pre_stage_clean (tie má každá tabuľka vlastné) a na výpočet zmien v stage_increment (spomínané *inc* procedúry). Bližšie budú popísané v ďalších sekciách o implementácií.

Pre logovanie sa využijú logovacie tabuľky dostupné v PDI popísané v sekcii 2.4.1. Konkrétne tabuľky *job*, *transformation* a *logging channels*. Pomocou atribútu *kettle_log_channel_id* v databázových tabuľkách bude možné získať logovací text z PDI tabuliek. V aplikácií stačí získavať logy len z týchto dvoch tabuliek podľa typu PDI komponenty, avšak pre využitie mimo aplikácie je možné integrovať aj ostatné tabuľky. Prístup k tabuľkám bude nastavený pomocou globálnych premenných nastaviteľných v súbore *kettle.properties*.

V rámci dátovej paralelizácie sa využijú znalosti a postupy popísané v sekcii 4.3.1 spolu s teoretickými znalosťami zo sekcie 2.3. Takýto postup sa uplatní na nahrávanie tabuliek, pri ktorých je bežne extrahované veľké množstvo dát z databázového zdroja a teda využitie dátovej paralelizácie dáva zmysel.

Na správu databázových pripojení budú využité taktiež funkcionality poskytované v PDI, keďže databázové pripojenia sa definujú a používajú v konkrétnych krokoch úloh a transformácií prípadne v globálnom súbore pripojení. Pre kontrolu maximálneho počtu pripojení počas nahrávania je ale potrebné implementovať vlastnú logiku s využitím metód z Kettle API pomocou ktorých bude obmedzované množstvo aktuálne využívaných pripojení tak, aby sa nepresiahol počet maximálnych pripojení.

5.3 Návrh architektúry aplikácie

Pre implementáciu aplikácie sa zvolila trojvrstvová architektúra softvérovej aplikácie. Trojvrstvová architektúra obsahuje prezentačnú, aplikačnú a dátovú

vrstvu. Prezentačná vrstva slúži na spracovanie vstupných požiadaviek od klientskej časti aplikácie a následnú prezentáciu výsledkov. [28] Aplikačná vrstva obsahuje logiku implementovaných domén, zaisťuje doménové pravidlá a prevádza výpočty a dátové manipulácie na základe požiadaviek prezentačnej vrstvy. Dátová vrstva má za úlohu zaisťovať komunikáciu s databázou a poskytuje objekty na prístup k dátam (Data Access Object). Zaisťuje tiež mapovanie databázových štruktúr na triedy v programe. Vždy platí, že vyššia vrstva využíva len služby nižších vrstiev. Tým sa vytvárajú jednoduchšie možnosti na zmenu implementácie jednotlivých vrstiev a znižuje sa množstvo závislostí v programe. [29]

Bude implementovaná serverová časť aplikácie s tým, že v prezentačnej vrstve bude navrhnuté API rozhranie, ktoré môže byť v budúcnosti využité na tvorbu klientskej časti aplikácie. Implementácia bude v jazyku Java, keďže sa využíva Kettle Java API poskytované od PDI. Na tvorbu jednotlivých vrstiev a funkcionalít sa využije Java framework Spring[29] spolu s nástrojom na automatickú kompiláciu a stavbu programu – Gradle[30].

5.3.1 API rozhranie

API bude obsahovať štyri hlavné domény a to:

- /edges – definícia hrán grafu závislostí (DAG),
- /components – doména pre správu komponent,
- /loads – správa nahrávaní a získavanie logov,
- /variables – doména pre správu premenných.

Implementácia API sa bude riadiť RESTful princípmi a teda bude implementovať CRUD[31] koncové body pre každú doménu (ak to v implementácií dáva zmysel). Celá špecifikácia API je dostupná v priloženom archíve. Pár dôležitých koncových bodov:

/loads spustenie nahrávania so špecifikovaným zoznamom komponent, ktoré majú byť spustené,

/loads/all spustenie nahrávania všetkých komponent, ktoré sú obsiahnuté v grafe,

/loads/stop zastavenie aktuálne bežiaceho nahrávania,

/loads/{loadId}/log získanie logu konkrétneho nahrávania,

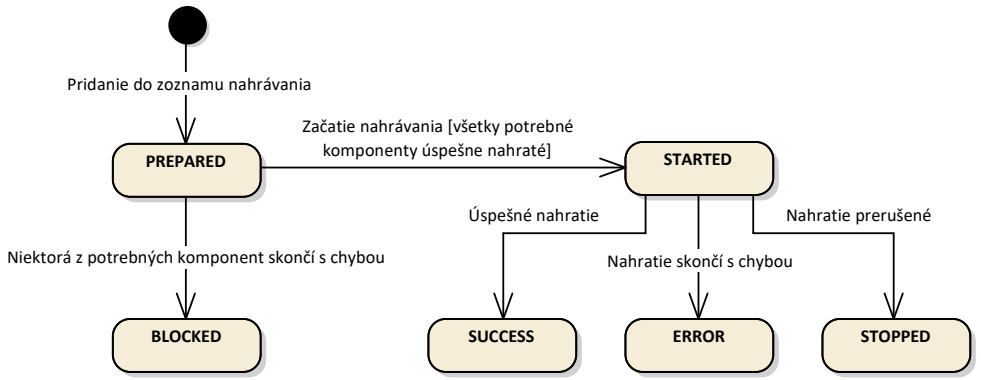
[29] `https://spring.io/`
[30] `https://gradle.org/`
[31] create, read, update, delete

/loads/{loadId}/components získanie všetkých komponent, ktoré boli nahraté v konkrétnom nahrávaní,

/loads/{loadId}/components/{componentId}/log
získanie logu konkrétnej komponenty v konkrétnom nahrávaní.

5.3.2 Algoritmus nahrávania komponent

Pre vytvorenie požiadavky na začatie nahrávania bude potrebné zavolať koncový bod */loads* a v tele požiadavky zaslať id komponent, ktoré sú vyžadované k nahratiu. Keďže komponenty majú definované závislosti v grafe závislostí, pre každú komponentu sa zistí, aké komponenty je potrebné vopred úspešne nahrať pre nahratie požadovaných komponent. Algoritmus berie na vedomie aj povinné hrany navrhnuté v sekcii 5.1.3 a to tak, že ak je potrebné nahrať komponentu, z ktorej idú nejaké povinné hrany, tak do nahrávania musia byť zahrnuté aj komponenty na konci povinných hrán. Komponenty nutné k nahratiu sú pridávané do spoločného zoznamu, z ktorého sa následne postupne vyberajú a nahrávajú. Toto všetko je potrebné vykonať rekurzívne, keďže všetky pridané komponenty k nahrávaniu môžu mať ďalšie komponenty, na ktorých sú závislé.



Obr. 5.3: Stavový diagram komponenty počas nahrávania

Po získaní zoznamu nahrávaných komponent sú všetky komponenty v stave *prepared*. Zo zoznamu sú postupne vyberané vopred definovaným počtom vlákien a začínajú sa nahrávať. Po vybratí zo zoznamu a začatí nahrávania sa komponenta dostane do stavu *started*. Nahrávanie môže začať len ak všetky komponenty potrebné k nahratiu tejto komponenty skončili úspechom a ak je voľný dostatočný počet pripojení k databázam, ktoré komponenta používa. Ukončením nahrávania sa komponenta dostáva buď do stavu *success* v prípade bezchybného nahratia alebo do stavu *error* v prípade chýb počas nahrávania. Špeciálnym stavom je stav *blocked*. Do tohto stavu sa komponenta dostane

v tom prípade, ak nejaká z komponent, ktoré je potrebné úspešne nahrať pred touto komponentou skončí s chybou. Všetky komponenty závislé na chybne nahratej komponente nie je možné nahrávať a preto sa ani nikdy nespustia – ostanú zablokované neúspešným nahratím predošlých komponent. Do stavu *stopped* sa môže komponenta dostať, ak príde požiadavka na zastavenie nahrávania. V prípade takejto požiadavky sa vyšle signál na zastavenie aktuálne nahrávaných komponent. Tie prejdú do tohto stavu a všetky ostatné komponenty ostanú v stavoch, v ktorých aktuálne boli. Zmeny stavov sú zachytené aj v stavovom diagrame na obrázku 5.3.

Celé nahrávanie (*load*) využíva stavy *started*, *success*, *error* a *stopped*. Nahrávanie mení svoj stav analogicky ku komponentám, avšak až po zmene stavov všetkých komponent. Podľa súčtu počtu chýb v nahrávaných komponentách je rozhodnuté či nahrávanie skončí úspešne, alebo s chybou (*success*/*error*). Do stavu *stopped* sa dostane v prípade požiadavky na zastavenie nahrávania po zastavení všetkých aktuálne nahrávaných komponent.

5.4 Správa maximálneho počtu pripojení do databázy

Ako bolo vysvetlené v sekcii 4.3.2 v PDI je možné ovplyvňovať connection pooling ale ten sám o sebe nekontroluje maximálny počet pripojení do databázy. Maximálny počet pripojení sa bežne definuje na používateľské konto databázy, pomocou ktorého sa ETL procesy pripájajú. Nie každá zdrojová databáza má definované takéto obmedzenie, ale niektoré databázy používané v ETL áno (napr. Grades).

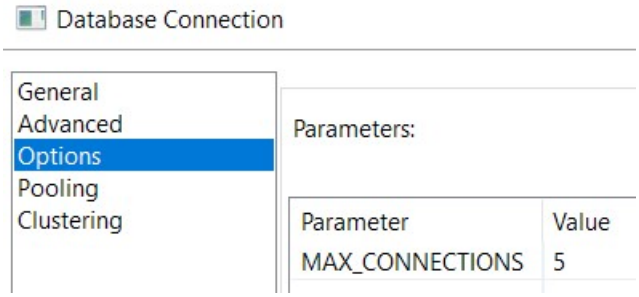
Riešenie si pred spustením nahrávania zistí nastavený maximálny počet pripojení z konfiguračného súboru *shared.xml*, ktorý sa v PDI používa na ukladanie zdieľaných objektov. Maximálny počet pripojení bude musieť byť pre databázu definovaný v nastaveniach databázového pripojenia v PDI pomocou parametru *MAX_CONNECTIONS*. Ukážka nastavenia pre zdieľané databázové pripojenie v PDI je na obrázku 5.4. Toto nastavenie je možné tiež zmeniť v súbore *shared.xml* XML kódom na ukážke 5.1. V aplikácií bude udržiavaná informácia o aktuálnom počte voľných pripojení pre každé databázové pripojenie, ktoré má definované maximálny počet pripojení vo svojich nastaveniach.

Pred spustením komponenty v nahrávaní aplikácie bude zistené, aké databázové pripojenia komponenta používa a aký najväčší počet ich bude používať v jeden moment počas svojho behu. Počet bude zisťovaný pre každé rôzne definované pripojenie. Pripojenia sa líšia podľa svojho názvu nastaveného pri vytváraní v PDI, ktorý musí byť unikátny ak ide o zdieľané pripojenie.

V prípade, že je počet používaných pripojení vyšší než hodnota maximálnych pripojení, ide o chybu. V prípade behu takejto komponenty v PDI

```
<connection>
<name>TARGET</name>
...
<attributes>
     <attribute>
          <code>EXTRA_OPTION_POSTGRESQL.MAX_CONNECTIONS</code>
          <attribute>120</attribute>
     </attribute>
     ...
</attributes>
</connection>
```

Výpis kódu 5.1: Nastavenie *MAX_CONNECTIONS* v *shared.xml* pre PostgreSQL pripojenie



Obr. 5.4: Nastavenie maximálneho počtu pripojení pre databázu v PDI

by totiž došlo v jednom momente ku otvoreniu väčšieho množstva pripojení než je povolené cudzím databázovým strojom a to by vyústilo v chybu počas behu, čím by sa ukončil celý beh komponenty. Komponenta s takouto konfiguráciou teda nemôže byť nikdy spustená.

Ak ale počet použitých pripojení nie je vyšší než maximálna hodnota, je možné komponentu v nahrávaní spustiť, ak je v celom nahrávaní v aplikácií aktuálne voľný aspoň takýto počet pripojení. V prípade, že počet voľných pripojení nie je dostatočný je komponenta odložená na neskoršie spustenie po tom, čo sa v celom nahrávaní uvoľnia používané pripojenia k potrebným databázam.

5.4.1 Získanie počtu použitých pripojení v transformácií

Keďže v transformácií každý krok beží v oddelenom vlákne naraz, tak platí, že každá kópia kroku vytvára exkluzívne pripojenie do databázy (vždy do databáz, ktoré sú v kópií kroku využívané). Ak sa teda napríklad v transformácií využíva databázové pripojenie do Grades v dvoch krokoch, kde každý z krokov

má dve kópie, tak dokopy po spustení transformácie môžu naraz byť otvorené štyri pripojenia do Grades databázy.

Ak má transformácia zaškrtnuté nastavenie „Make the transformation database transactional“, tak sa v celej transformácií k jednému dátovému zdroju vytvorí vždy iba jedno fyzické pripojenie. V tomto prípade počet krokov a počet kópií krokov, v ktorom je databázový zdroj využitý nehrá rolu. Nastavenie bolo zobrazené v analytickej časti v sekcii 4.3.1 na obrázku 4.4.

5.4.2 Získanie počtu použitých pripojení v úlohe

Beh úlohy je postupný a jednotlivé kroky úlohy sú spúšťané podľa definovaného poradia. Počet používaných pripojení v jednom momente je teda iba počet pripojení, ktoré používa krok úlohy, ktorý sa aktuálne exekuuje. Je ale možné na kroku úlohy nastaviť možnosť „Run Next Entries in Parallel“, ktorá bola popisovaná v sekcii 2.3.4. To spôsobí, že následujúce kroky úlohy sú spustené vo vlastných vláknach, t.j. je možné v úlohe vytvoriť pripojenia z viacerých krokov naraz.

Počet použitých pripojení bude zisťovaný rekurzívne tak, že bude zistený počet pripojení v následujúcich krokoch konkrétneho kroku a pripojenia budú sčítané po vrátení rekurzívnych hodnôt v prípade, že boli tieto kroky spustené paralelne. Ak neboli spustené paralelne, bude vždy vybraný počet najväčšieho množstva použitých pripojení spomedzi všetkých vrátených hodnôt. Pre zistenie počtu pripojení v celej úlohe sa začne od kroku úlohy *START*, ktorý musí mať každá úloha definovaný.

Ak je krokom úlohy iná vnorená úloha alebo transformácia, bude zistený počet použitých databázových pripojení v týchto vnorených úlohách a bude rovnako braný do úvahy najväčší počet pripojení.

Kapitola 6

Implementácia a testovanie riešenia

V tejto kapitole bude popísaná implementácia paralelizačného riešenia pomocou aplikácie nad nástrojom PDI a taktiež budú popísané ostatné zmeny potrebné pre implementáciu ako úpravy PDI súborov a využitie dátovej paralelizácie v PDI komponentách. V druhej časti kapitoly bude riešenie otestované spustením pôvodného sériového a nového paralelného riešenia. Budú popísané vzniknuté zistenia a porovnané rýchlosti nahrávaní.

6.1 Použité technológie

Riešenie je implementované v jazyku Java. Pre jednoduchšiu implementáciu API a perzistenciu dát do databázy je použitý framework Spring a konkrétne jeho závislosti *spring-boot-starter-web* (tvorba API rozhrania) a *spring-boot-starter-data-jpa* (perzistencia dát). Pre automatické generovanie API dokumentácie podľa anotácií definovaných v kóde pri jednotlivých API koncových bodoch je využitá závislosť *springdoc-openapi-ui*. API dokumentácia je generovaná vo formáte OpenAPI[32] a taktiež ako HTML stránka zobraziteľná v Swagger editore[33].

Tabuľky navrhnuté v sekcii 5.1 sú uložené v PostgreSQL databáze, konkrétne v internej databáze DW ČVUT vo vlastnej schéme. Tabuľky v databáze je potrebné namapovať na doménové objekty v aplikácií, aby mohli byť perzistované pomocou API. Na to slúži ORM (Object-relational mapping). V Jave je možné ho implementovať s využitím Java Persistence API (JPA), čo je jednotné rozhranie pre objektovo-relačné mapovanie. *Hibernate* je implementácia JPA, ktorá bola použitá. Ide o základnú implementáciu JPA vo frameworku Spring. Na perzistenciu dát sa používa JpaRepository, ktoré

[32] `https://www.openapis.org/`
[33] `https://swagger.io/`

poskytuje základné SQL operácie na úpravu dát, prípadne umožňuje vytvárať vlastné komplikovanejšie SQL dotazy.

Pre pridanie závislostí Kettle API sú priamo do projektu skopírované .jar súbory potrebné k použitiu podľa PDI dokumentácie. Súbory sa nachádzajú v oficiálnej distribúcií PDI, ktorú je možné stiahnuť zo stránok Hitachi Vantara[34]. Konkrétne boli použité priečinky /classes, /lib, /libswt a /plugins. Závislosti sú pridané podľa oficiálneho postupu[35]. V spojení s frameworkom Spring bolo potrebné zmazať niektoré súbory z priečinka /lib, pretože dochádzalo k duplicitnému pridávaniu závislostí do Java classpath.

Na získanie všetkých závislostí, kompiláciu a stavbu programu je použitý nástroj Gradle. Pomocou závislostí a konfigurácie definovaných v súbore *build.gradle* je možné projekt spustiť. Po spustení projektu sa vytvorí server, na ktorý je možné posielať API požiadavky. V základnom nastavení je dostupný na doméne *localhost:8080*. Celý zdrojový kód a implementácia aplikácie je dostupná v priloženom archíve. Ukážky spúšťania aplikácie a testovanie chybových výstupov sú dostupné v prílohe C.

6.2 Implementácia paralelného nahrávania komponent

Logika nahrávania je implementovaná podľa návrhu popísaného v predošlých sekciách. Hlavným bodom riešenia je paralelné nahrávanie jednotlivých komponent podľa poradia definovaného v grafe závislostí so správov stavov nahrávania komponent a celého nahrávania pomocou databázových tabuliek.

Graf závislostí je definovaný ako orientovaný acyklický graf. V implementácií bola táto podmienka vynútená kontrolou cyklov v tabuľke *edge*. Na implementáciu sa využíva databázový trigger, ktorý pri vkladaní alebo úprave riadkov v tabuľke skontroluje, či by vykonané zmeny nevytvorili v grafe cyklus. Na kontrolu cyklu je použitý algoritmus topologického usporiadania vrcholov grafu (Top Sort). Ten vie detekovať cyklus v prípade, že nie je možné určiť poradie vrcholov. Ak nie je možné určiť poradie vrcholov v grafe, znamená to, že na sebe cyklicky závisia. Topologické usporiadanie je implementované pomocou SQL a v prípade, že je detekovaný cyklus pri zmenách v tabuľke, je zabránené, aby sa vykonali. Implementácia je dostupná v prílohe A.

Správa nahrávania je v aplikácií implementovaná v triedach *LoadService* a presahuje aj do tried *KettleService* a *LoadMetadataService* a balíka *business.runnables*. Sú využívané aj ostatné triedy aplikačnej vrstvy nachádzajúce sa v balíku *business*. Nahrávanie komponent je možné začať volaním endpointu */loads* s telom požiadavky, v ktorom sú špecifikované komponenty pre nahratie. Ukážka tela je vo výpise 6.2. Po kontrole či poslané komponenty existujú

[34] `https://www.hitachivantara.com/en-us/home.html`

[35] `https://help.hitachivantara.com/Documentation/Pentaho/9.4/Developer_center/Embed_and_extend_PDI_functionality`

```
raise exception
     'Trying to create cycle in directed acyclic graph '
     'structure by edge \%->\%, id_edge: \%',
          new.start_vertex, new.end_vertex, new.id_edge;
```

Výpis kódu 6.1: Vyhodenie výnimky v prípade nájdenia cyklu

```
{
    "componentIds": [
        51, 55, 57
    ]
}
```

Výpis kódu 6.2: JSON objekt pre požiadavku na nahratie komponent

je zistený zoznam komponent, ktoré je potrebné nahrať spolu s komponentami špecifikovanými v požiadavke. To je vykonané rekurzívnym dotazom do tabuľky *edge*, ktorý berie do úvahy závislé komponenty a povinné hrany podľa definície v návrhu (5.1.3). Volanie získavajúce komponenty je v prílohe B.

Po získaní komponent je v triede *LoadService* začaté nahrávanie, ak aktuálne nebeží iné nahrávanie. V jednom momente totiž nemôže v aplikácií bežať viacero nahrávaní. Nahrávanie prebieha podľa návrhu v sekcii 5.3.2. Paralelné nahrávanie komponent je implementované asynchrónne bežiacimi vláknami, ktoré sú vytvorené cez *ExecutorService*. V konfiguračnom súbore *application.properties* je definovaný počet vlákien, ktorý sa spustí pri každom nahrávaní. Komponenty vybrané k nahrávaniu sú uložené v spoločnom zozname. Ten je synchronizačným objektom všetkých vlákien. Vlákna postupne vyberajú komponenty zo zoznamu a snažia sa spustiť ich exekúciu. Zároveň pred spustením kontrolujú, či sú všetky komponenty, na ktorých sú závisle úspešne nahraté a taktiež, či majú dostatočný počet voľných pripojení na spustenie ak používajú databázové pripojenia, ktoré majú definovaný maximálny počet pripojení. Počet pripojení sa zisťuje a kontroluje v triede *KettleService* pomocou Kettle API metód a v prípade nedostatočného rozhrania sa využíva zisťovanie hodnôt priamo z XML súborov PDI komponent. Ak podmienky nie sú splnené, vyberie sa ďalšia komponenta zo zoznamu, až kým sa nenájde komponenta, ktorú je možné nahrať. Ak všetky komponenty v zozname zatiaľ nemôžu byť nahraté, vlákno sa uspí a čaká na prebudenie. Prebudenie nastane po ukončení nahrávania nejakej komponenty iným vláknom. Po vyprázdnení zoznamu sa vlákna ukončia a nastaví sa celkový stav nahrávania.

Exekúcia komponent je implementovaná v triedach *KettleJobRunnable* a *KettleTransRunnable*, ktoré implementujú Java interface *Runnable* používaný pri exekúcií konkurentných vlákien. Ukážka implementácie exekúcie PDI úlohy je vo výpise kódu 6.3. Podobne je implementovaná aj exekúcia transformácie, iba používa iné metódy podľa definície rozhrania Kettle API.

```
JobMeta jobMeta = new JobMeta(
        loadMetadata.getComponent().getPath(), null);
Job job = new Job(null, jobMeta);

// get used connections in job
Map<String, Integer> usedConnectionsCount =
        loadMetadata.getKettleUsedDBConn();
// or calculate them if not stored already
...
// check if amount of used connections
// does not prevent job from running
kettleService.checkKettleUsedConnections(load,
        usedConnectionsCount);

job.start();
loadMetadata.setStatus(LoadStatus.STARTED);
loadMetadata
        .setKettleLogChannelId(job.getLogChannelId());
loadMetadataService.update(loadMetadata);
// check for stop request
...
kettleService.releaseKettleUsedConnections(load,
        usedConnectionsCount);
loadMetadata
        .setErrorCount(job.getResult().getNrErrors());
loadMetadataService.update(loadMetadata);
```

Výpis kódu 6.3: Spustenie PDI úlohy v triede *KettleJobRunnable*

6.2.1 Dátová paralelizácia v PDI

Podľa analýzy v sekcii 4.3.1 je dátová paralelizácia použitá pri nahrávaní pre_stage tabuliek *tcitation_affiliations* a *tcitation_authors*. V transformáciách je vypnuté nastavenie „Make the transformation database transactional“, aby bol vytvorený taký počet pripojení do zdrojového systému, aký je počet definovaných kópií kroku *Select values*.

6.3 Prístup cez API

V aplikácií je vytvorené rozhranie API pre správu databázových objektov. Jeho návrh bol popísaný v sekcii 5.3.1 a celá dokumentácia API je dostupná pri zdrojovom kóde v priloženom archíve. Implementácia koncových bodov je dostupná v *Controller* triedach v balíku *api.controllers* vždy podľa názvu

API domény. Na prijímanie API požiadaviek a odosielanie odpovedí sú použité Data Transfer Objects (DTO). Výhodou týchto objektov je skrytie vnútornej implementácie objektov aplikácie nazývaných Data Access Objects (DAO). Na konvertovanie medzi DTO a DAO objektami sú využívané konvertory implementované v balíku *api.converters*. DTO objekty sú dostupné v prezentačnej vrstve v balíku *api.dtos* a DAO objekty sú súčasťou dátovej vrstvy aplikácie v balíku *domain*.

V DAO aj DTO objektoch, ako aj v celej implementácií aplikácie, sú využívané výčtové typy *enum* pre udržanie konzistencie názvov a prehľadnosti kódu. Konkrétne sú definované dva výčtové typy v balíku *enums*:

- *ComponentType* – typy komponent, hodnotami sú momentálne *job* a *trans*,
- *LoadStatus* – typy stavu nahrávania, ktorých hodnoty boli popísané v sekcii 5.3.2.

6.4 Log nahrávania

Logovaciu informáciu je možné získať pre celé nahrávania podľa id nahrávania alebo pre konkrétnu komponentu v nahrávaní podľa id nahrávania a id komponenty. Logy sú pre konkrétne komponenty získavané z tabuľky *loadMetada*, z ktorej sa získava stav behu komponenty a počet chýb. Podľa atribútu *kettleLogChannelId* z tejto tabuľky sú získavané kompletné logovacie texty z PDI logovacích tabuliek definovaných v internej databáze DW ČVUT, do ktorých PDI procesy počas nahrávania logy zapisujú s nastaveným logovacím intervalom. Ten je možné zmeniť v konfiguračnom súbore *application.properties*. Z tabuľky *load* sú doplnené informácie o priebehu celého nahrávania.

Po zavolaní požiadavky na API koncový bod */loads*{*loadId*}*/log*, kde *loadId* je id nahrávania je vrátený DTO objekt *LoadLogDto*. Ukážka jeho JSON formátu v API odpovedi je na výpise 6.4.

6.5 Úprava PDI komponent používaných pri ETL procesoch

Na nahrávanie ETL procesov sú v riešení využívané PDI úlohy a transformácie. Ich pôvodná štruktúra je popísaná v analýze v kapitole 3. Pre podporu paralelného nahrávania bolo potrebné zmeniť úlohu *J_MAKE_INCREMENT* tak, aby bolo možné nahrávať pre_stage_clean a stage_increment ETL transformácie len pre samostatnú tabuľku. Súbežná bakalárska práca kolegyne mala rovnaký problém a boli v nej upravené používané PL/pgSQL procedúry tak, že prijímajú parametre s názvom schémy a tabuľky.

```
{
    "load": {
        "id": 155,
        "startTime": "2023-04-30 19:42:27.027",
        "endTime": "2023-04-30 19:42:46.046",
        "status": "success",
        "errorCount": 0
    },
    "componentLogs": [
        {
            "componentId": 44,
            "componentType": "job",
            "componentName": "tekns_LOAD",
            "startTime": "2023-04-30 19:42:29.029",
            "endTime": "2023-04-30 19:42:45.045",
            "status": "success",
            "errorCount": 0,
            "logMessage":
                "2023/04/30 19:42:29 - tekns_LOAD
                    - Start of job execution
                 2023/04/30 19:42:30 - tekns_LOAD
                    - Starting entry [Set vars for PS..."
        },
        ...
    ]
}
```


Výpis kódu 6.4: JSON objekt odpovede na požiadavku o log nahrávania

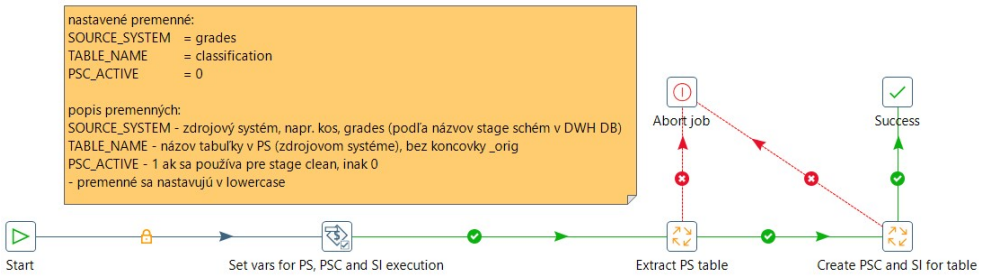
Keďže by v tejto práci išlo o rovnakú úpravu, boli využité procedúry upravené kolegyňou.

Hlavnými zmenami, ktoré boli vykonané bolo zjednodušenie a zníženie počtu súborov v stage časti nahrávania (3.2.1). Ukážka novej štruktúry PDI súborov pre nahrávanie Grades je popísaná vo výpise 6.1. Pôvodné transformácie s koncovkou *_LOAD* na extrahovanie dát zo zdrojových systémov boli premenované na výstižnejšiu koncovku *_EXTRACT*. Transformácie s koncovkou *_CHECK*, v ktorých sa pre rôzne tabuľky menil iba názov tabuľky a schéma pri volaní PL/pgSQL funkcie boli zmazané a vytvorená bola jedna parametrizovaná transformácia *PS_table_extract_CHECK.ktr*, ktorú využívajú všetky stage PDI úlohy. Na spustenie PSC a SI transformácií som vytvoril parametrizovanú úlohu *create_PSC_and_SI_for_table.kjb* a pre nastavenie premenných pre kontrolu nahratia pre_stage tabuľky (pôvodné *_CHECK* transformácie) zasa úlohu *extract_PS_table.kjb*.

```
STAGE ..... úlohy nahrávajúce stage vrstvu
  LOAD_JOBS_grades ..... stage procesy pre zdrojový systém Grades
    Extract_transformations ... extrahovanie zo zdrojových tabuliek
      classification_EXTRACT.ktr
      classification_user_EXTRACT.ktr
      student_classification_EXTRACT.ktr
      ...
    classification_LOAD.kjb
    classification_user_LOAD.kjb
    student_classification_LOAD.kjb
    ...
  create_PSC_and_SI_for_table.kjb ..... nahratie PSC a SI
  extract_PS_table.kjb ..... príprava extrahovania
  PS_table_extract_CHECK.ktr ..... kontrola pred extrahovaním
TARGET ..... súbory nahrávajúce target vrstvu
  NOHIST ..... nahrávanie tabuliek bez historizácie
  STAGEtoTARGET ..... nahrávanie tabuliek, ktoré využívajú stage
    KLAS ..... transformácie pre nahratie tabuliek v doméne KLAS
      t_klas_klasifikace_LOAD.ktr
      t_klas_klasifikace_student_LOAD.ktr
```

Obr. 6.1: Súborová štruktúra PDI súborov používaných pri nahrávaní Grades

Pre nahratie stage tabuľky je teda potrebné vytvoriť len transformáciu *_EXTRACT*, kde sa nastaví extrahovanie dát zo zdroja pomocou bulk loading procesu a úlohu s koncovkou *_LOAD*, v ktorej sa nastavujú premenné potrebné pre vykonanie procesov. Premenné sú potrebné pre spustenie parametrizovaných úloh, v ktorých sa používajú princípy názvoslovia definovaného v DW ČVUT. Taktiež je potrebné dodržať definovanú súborovú hierachiu. Ukážka súboru *classification_LOAD* pre zdrojovú tabuľku *classification* je na obrázku 6.2. Celá štruktúra aj s PDI súbormi je dostupná v priloženom archíve.

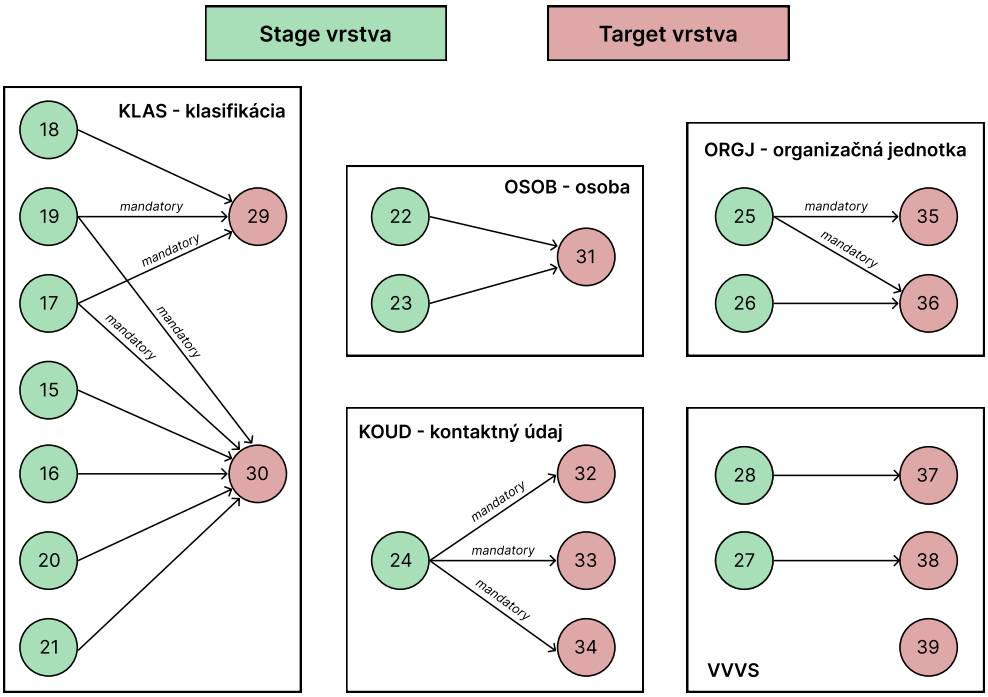


Obr. 6.2: Extrahovanie tabuľky *classification* v úlohe *classification_LOAD*

6.6 Testovanie nahrávania

Pre otestovanie riešenia bolo potrebné porovnať čas sériového nahrávania pomocou pôvodného riešenia a čas paralelného nahrávania pomocou novovytvoreného riešenia. Pre čo najbližšie výsledky bola obnovená záloha databázy dátového skladu na jeden z interných serverov, v ktorej boli vytvorené kópie testovaných tabuliek pre jednotlivé testy. Testovacie nahrávania tak vychádzali z rovnakého bodu v histórií dát a obsahovali rovnaké záznamy.

Sériové aj paralelné nahrávanie používali novú hierarchiu PDI súborov zo sekcie 6.5. V sériovom nahrávaní bol vytvorený PDI súbor, ktorý zaradom nahrával jednotlivé pre_stage, pre_stage_clean, stage_increment a na koniec target tabuľky vždy postupne jednu tabuľku za druhou. Tak je totiž aktuálne vykonávané ETL nahrávanie. V aktuálnom stave je v nahrávaní používaná dátová paralelizácia pri nahrávaní zmenených záznamov do target tabuľky. Tá bola aj počas testu sériového nahrávania zachovaná. V paralelnom nahrávaní bol vytvorený graf nahrávania v aplikácií, v ktorom boli definované závislosti tabuliek. Graf je možné vidieť na obrázku 6.3. Tabuľky na obrázku sú zoskupené podľa štvorpísmenných target domén a sú reprezentované číslami komponent v aplikácií počas behu paralelného nahrávania. Závislosti korešpondujú aj s tabuľkou 4.1. Bola využitá úlohová paralelizácia v implementácií aplikácie a zároveň dátová paralelizácia v niektorých PDI transformáciach.



Obr. 6.3: Graf komponent v aplikácií počas paralelného nahrávania

6.6.1 Porovnanie sériového a paralelného nahrávania

Nahrávania boli spúšťané na lokálnom počítači, ktorý disponuje štyroma jadrami a dokopy ôsmimi vláknami so základnou rýchlosťou procesora 1,80 GHz s operačnou pamäťou o kapacite 16 GB. Počas celého nahrávania mal lokálny počítač stabilné pripojenie o rýchlosti okolo 95 Mbps. Typom databázového systému dátového skladu v čase nahrávania bol PostgreSQL 14.

Tabuľka 6.1: Dĺžky jednotlivých nahrávaní

Nahrávanie	Dĺžka nahrávania	Súčet dĺžok nahrávania jednotlivých tabuliek
Sériový test	8:07:19	8:07:19
Paralelný test č. 1	4:22:16	8:40:07
Paralelný test č. 2	3:15:44	7:44:57

V tabuľke 6.2 je možné vidieť dĺžky nahrávania jednotlivých tabuliek vo vrstvách. Čas nahrávania tabuliek v stage vrste v tejto tabuľke zahŕňa extrahovanie dát zo zdroja, čistenie dát (pre_stage_clean) a zistenie zmien v dátach oproti poslednej verzii dát v dátovom sklade (stage_increment). Nahrávanie tabuliek v target vrstve predstavuje nahrávanie dát do IDL a tvorbu historizácie v IDL. Paralelný test č. 1 aj č. 2 využíval dátovu paralelizáciu pri extrahovaní dát zo zdroja do pre_stage tabuliek a to konkrétne na tabuľkách *tcitation_affiliations*, *tcitation_authors*, *tkontakt* a *osoby*. V paralelných testoch bola taktiež pri väčších target tabuľkách vypnutá možnosť „Make the transformation database transactional“, čím sa umožnilo využitie viacerých pripojení do databáz. Bola ale zaručená jednoduchá oprava dát v prípade chyby počas behu transformácie a to pridaním *Blocking step* krokov, ktoré zaručujú poradie nahratia zmien v poradí nové, zmazané a upravené. V target ETL bola pridaná dátová paralelizácia aj pre kroky *Database lookup*, ktoré v target transformáciách vyhľadávajú dáta v stage vrstve. V paralelnom teste č. 2 nebola tabuľka *t_klas_klasifikace_student* nahrávaná pretože v transformácií bolo definované väčšie množstvo pripojení než maximálny počet povolených pripojení.

V tabuľke 6.1 sú dĺžky jednotlivých pokusov nahrávaní, z ktorých je možné pozorovať, že pri paralelnom nahrávaní s využitím dátovej aj úlohovej paralelizácie došlo k signifikantnému zrýchleniu nahrávania. Zaujímavým zistením spolu s dĺžkami z tabuľky 6.2 je aj to, že ak sú porovnávané len dĺžky nahrávania jednotlivých tabuliek tak je ich nahrávanie pri paralelnom behu väčšinou dlhšie. Keby dĺžky nahrávania jednotlivých tabuliek z tabuľky 6.2 v paralelných testoch spočítame ako keby bežali sériovo, t.j. vytvoríme súčet dĺžok, dostaneme hodnotu v stĺpci „Súčet dĺžok nahrávania jednotlivých tabuliek“ v tabuľke 6.1. Na týchto hodnotách je možné vidieť, že by šlo o podobne dlhý proces ako pri sériovom nahrávaní. Úsporov času je spustenie nahrávania tabuliek paralelne pomocou úlohovej paralelizácie a aj paralelné extrahovanie

Tabuľka 6.2: Dĺžky nahrávania tabuliek v sériovom a paralelných nahrávaniach

	Názov tabuľky	Sériový test	Paralelný test č. 1	Paralelný test č. 2
Stage vrstva	boolean_student_classification	0:00:04	0:01:56	0:01:17
	classification	0:00:03	0:01:45	0:00:57
	classification_text	0:00:01	0:01:37	0:00:57
	classification_user	0:00:01	0:01:06	0:01:11
	number_student_classification	0:00:16	0:02:48	0:02:32
	string_student_classification	0:00:03	0:01:49	0:01:27
	student_classification	0:00:26	0:04:23	0:04:03
	osoby	0:01:05	0:05:10	0:05:12
	tusers	0:00:30	0:07:50	0:12:01
	tkontakt	0:02:18	0:07:52	0:10:36
	tekns	0:00:03	0:00:48	0:01:18
	torganizations	0:00:04	0:04:50	0:04:38
	tcitation_affiliations	3:16:47	3:20:14	2:42:14
	tcitation_authors	3:46:58	4:19:46	3:14:53
Target vrstva	t_klas_klasifikace	0:02:25	0:00:11	0:13:55
	t_klas_klasifikace_student	0:16:41	0:00:23	-
	t_osob_osoba	0:00:11	0:00:03	0:00:45
	t_koud_adresa	0:17:40	0:00:52	0:13:14
	t_koud_email	0:09:41	0:01:08	0:05:25
	t_koud_telefonni_cislo	0:09:50	0:00:33	0:15:44
	t_orgj_organizacni_jednotka	0:00:01	0:00:45	0:00:16
	t_orgj_organizacni_jednotka _externi	0:00:01	0:00:03	0:00:09
	t_externiorganizacnijednotka _externicitaceautor_rel	0:00:27	0:04:43	0:01:19
	t_vvvs_externi_citace_autor	0:01:15	0:02:23	0:00:45
	t_vvvs_vedecky_vysledek _bibl_indik_nohist	0:00:28	0:07:09	0:10:09

dát pomocou dátovej paralelizácie. Pri porovnaní celkovej doby nahrávania je paralelné nahrávanie výrazne kratšie. Predĺženie nahrávania jednotlivých tabuliek môže byť spôsobené prepínaním kontextu vlákien ale aj priepustnosťou internetového pripojenia. Napríklad pri extrahovaní dát zo zdroja pre tabuľku *tcitation_affiliations* sa prejavilo obmedzenie rýchlosti pripojenia keďže extrahovanie v analytickej časti (4.3.1) s rovnakou dátovou paralelizáciou bolo odhadnuté na dĺžku okolo 40 min. Počas paralelného nahrávania, kedy sa dáta extrahovali naraz z tabuliek *tcitation_affiliations* aj *tcitation_authors* (spoločný zdroj EZOP), sa extrahovanie tabuľky *tcitation_affiliations* predĺžilo na 3 h 5 min v paralelnom teste č. 1 a na skoro dve hodiny v paralelnom teste č. 2 (v

tabuľke 6.2 sú uvedené dĺžky aj s nahrávaním PSC a SI). Avšak dĺžka výpočtu zmien v stage_increment, čo nie je operácia náročná na internetové pripojenie, medzi testami nebola výnimočne ovplyvnená. Z tohto dôvodu je možné očakávať, že pri navýšení rýchlosti pripojenia, prípadne výpočtových prostriedkov, sa môže paralelné nahrávanie ešte viac urýchliť. Dôležitým faktorom je ale aj rýchlosť získavania dát zo zdroja, kde bolo pozorované, že pri získavaní dát zo zdrojových systémov EZOP a KOS je extrahovanie výrazne pomalšie než pri extrahovaní napr. zo systému Grades. Extrahovanie približne 500 000 riadkov dát z tabuľky *tkontakt* (KOS) trvalo v sériovom nahrávaní 2 min 18 s a extrahovanie približne 1 000 000 riadkov dát z tabuľky *student_classification* (Grades) trvalo v sériovom nahrávaní 26 s. Pri paralelnom nahrávaní bola dĺžka extrahovania zo systému Grades taktiež kratšia, konkrétne o polovicu.

Dĺžka nahrávania paralelného testu č. 2 bola o hodinu kratšia než dĺžka paralelného testu č. 1. Je to spôsobené nahrávaním tabuliek *tcitation_affiliations* a *tcitation_authors*. Analyzovaním informácií o extrahovaní tabuľky *tcitation_authors* z logu bolo zistené, že v paralelnom teste č. 1 trvalo extrahovanie každých 50 000 riadkov rôzne dlho. Dĺžka kolísala postupne medzi jednou minútou, dvoma minútami až troma minútami. V strede extrahovania sa interval skrátil na jednu minútu ale následne sa znova predĺžil na tri minúty až do ukončenia celého extrahovania. Pri paralelnom teste č. 2 išlo o rýchlejší proces a extrahovanie 50 000 riadkov zabralo zo začiatku približne jednu minútu, za polovicou extrahovania krátko vystúpilo na tri minúty ale následne sa držalo stabilne na dĺžke jednej a pol minúty. To prispelo k tomu, že sa extrakcia oproti prvému testu skrátila z 3 h 46 min na 2 h 45 min, V paralelnom teste č. 1 je zaujímavé kolísanie rýchlosti nahrávania. Extrahovanie v teste č. 1 bolo vykonávané konkrétne od 22:36 h do 02:22 h, oproti testu č. 2, ktorý bol vykonávaný od 11:43 h do 14:28 h. Pravdepodobne sa v noci v systéme EZOP vykonávali iné procesy, ktoré skrátili rýchlosť získavania dát a vytvárali kolísanie v dĺžkach extrahovania. Pozorovateľné je aj to, že v paralelnom teste č. 2 sa dĺžky nahrávania jednotlivých tabuliek oproti paralelnému testu č. 1 zväčšili. To môže byť spôsobené zvýšenou záťažou na spracovanie z dôvodu rýchlejšieho extrahovania EZOP zdrojových tabuliek v druhom teste.

Posledným dôležitým zistením je urýchlenie spracovania stage_increment procesov pre jednotlivé tabuľky. V aktuálnej implementácií sú totiž tieto procesy spúšťané naraz pre všetky tabuľky v jednej veľke databázovej transakcii. Z dôvodu paralelného nahrávania boli procesy rozdelené na spracovanie samostatných tabuliek. Stage_increment proces sa pre tabuľku *tcitation_affiliations* skrátil z bežnej dĺžky 50 min (aktuálne pravidelne v nahrávaniach dátového skladu) na 8 min 20 s v prvom paralelnom teste a na 7 min 1 s v druhom paralelnom teste. V sériovom teste trval proces tiež len 8 min 23 s. Takéto zrýchlenie môže byť spôsobené rozdelením jednej príliš veľkej transakcie na viacero malých transakcií, čím sa naraz nevyužíva tak veľké množstvo pamäte potrebnej na udržanie databázovej transakcie. Podobné správanie je možné pozorovať aj pre ostané tabuľky v nahrávaní.

Kapitola 7

Zhodnotenie riešenia a jeho možného využitia

Cieľom práce bolo definovať požiadavky na paralelizáciu systému a na vhodne zvolenej časti ETL procesu vytvorené riešenie demonštrovať. Požiadavky aj časť ETL procesu boli definované v kapitole 4 spolu so súbežnou bakalárskou prácou kolegyne Kristiny Zolochevskaiej. Jednotlivé požiadavky boli splnené takto:

F1. splnená reprezentovaním jednotlivých komponent ako vrcholov orientovaného acyklického grafu s využitím doplnkového označenia povinných hrán pre vyriešenie závislostí tabuliek. Riešenie je implementované v rámci navrhnutej aplikácie nad nástrojom PDI.

F2. splnená možnosťou nahratia zoznamu komponent v aplikácií pomocou API požiadavky s využitím databázových tabuliek aplikácie na správu komponent a závislostí medzi komponentami, úpravou PL/pgSQL procedúr používaných pre výpočet zmien v stage_increment tabuľkách tak, aby upravovali len jednu tabuľku a zjednodušením PDI súborov pre nahrávanie tabuliek v stage vrstve

F3. splnená využitím nástroja PDI na tvorbu ETL procesov s aktualizáciou na najnovšiu verziu 9.4, ktorý umožňuje nahrávanie pomocou rôznych typov databázových systémov na základe vstavaných pluginov, prípadne umožňuje tvorbu vlastných pluginov pre prípad nepodporovaného typu databázového systému

F4. splnená úlohovou paralelizáciou v rámci implementácie aplikácie nad nástrojom PDI s využitím viacerých vlákien na spracovávanie nezávislých úloh a dátovou paralelizáciou v PDI transformáciách podľa analýzy zo sekcie 4.3.1. Riešenie je pripravené aj na použitie na interných serveroch Dátového skladu ČVUT.

F5. splnená možnosťou získania logovacej informácie o celom nahrávaní alebo o konkrétnej komponente v konkrétnom nahrávaní pomocou API požiadavky v aplikácií spolu s využitím vstavanej funkcionality nástroja PDI na logovanie priebehu nahrávania do databázových tabuliek

N1. splnená možnosťou úpravy použitého množstva vlákien pri dátovej paralelizácií v konfigurácií aplikácie a v prípade dátovej paralelizácie je úprava množstva využívaných prostriedkov súčasťou paralelizačných funkcionalít v nástroji PDI

N2. splnená pomocou využitia jazyka Java a využitia Kettle Java API na spúšťanie PDI v riešení, ktoré je možné kompilovať a spúšťať nezávisle od typu operačného systému, a splnená pomocou nástroja Gradle pre automatické stiahnutie závislostí v programe, ktorý nevyžaduje žiadne dodatočné inštalácie od používateľa pred spustením aplikácie a je taktiež spustiteľný nezávisle od operačného systému

Otestovanie riešenia bolo prevedené spustením paralelného nahrávania na vybranej časti ETL procesu. Dĺžka paralelného nahrávania bola následne porovnaná so sériovým nahrávaním. V sekcii 6.6 boli zhrnuté poznatky z porovnania jednotlivých nahrávaní.

Riešenie prináša signifikantné zrýchlenie v dĺžke nahrávania a umožnuje rozdelenie nahrávania na menšie časti. Keďže sa ale nástroj PDI ukázal ako nedostatočný, bolo ho potrebné rozšíriť vlastnou aplikáciou, čo do ETL procesu prináša ďalšie relatívne komplikované softvérové riešenie. To je potrebné pravidelne udržovať a vyvíjať, a keďže ide o vlastné riešenie existuje aj väčšia tendencia vytvárania chýb pri úpravach v budúcnosti. Navrhnuté riešenie taktiež zatiaľ neposkytuje grafické rozhranie, ktoré by bolo potrebné doimplementovať. Poskytované Java API od Kettle je niekedy nedostatočné a na zložitejšie problémy už nie je využiteľné. Keďže je využívané open-source riešenie, v prípade chýb je potrebné sa obrátiť na internetovú komunitu, ktorá sa ale počas riešenia tejto bakalárskej práce ukázala ako neaktívna. Ďalším z nedostatkov využívania open-source PDI distribúcie je aj nedostatočná dokumentácia, prípadne chyby, ktoré sú opravované menej často než v prípade platenej distribúcie.

Využiteľnosť riešenia je nesporná a riešenie by prinieslo zrýchlenie a zjednodušenie niektorých procesov. Avšak s ohľadom na znižujúcu sa aktivitu vývoja PDI a komplikácie, ktoré prináša vlastný softvérový projekt takéhoto rázu sa javí ako nevhodné. Sú dostupné kompletné riešenia, ktoré by odbremenili dátový sklad od nutnosti údržby a prevádzky vlastného ETL orchestračného nástroja prezentovaného v tejto bakalárskej práci. Jedným z príkladov je nástroj *Apache airflow* bližšie skúmaný v súbežnej bakalárskej práci kolegyne, ktorý je možné integrovať aj s nástrojom PDI. Aktuálnejším ETL nástrojom, ktorý by bolo možné tiež využiť a vychádza z nástroja PDI je napríklad nástroj *Apache Hop*.

Záver

Táto práca mala za cieľ pomocou analýzy nástroja Pentaho Data Integration (PDI) navrhnúť paralelizáciu ETL procesov Dátového skladu ČVUT. Bolo potrebné využiť možnosti tohto nástroja a ak boli možnosti nedostatočné, tak navrhnúť aplikáciu, ktorá doplní chýbajúce funkcionality tak, aby boli splnené požiadavky definované v analytickej časti práce. Požiadavky na paralelizáciu a tabuľky dátového skladu, na ktorých bolo riešenie predvedené boli špecifikované spolu so súbežnou bakalárskou prácou, ktorá riešenie predviedla na inom type nástroja.

Vytvorené riešenie sa zameriava na dva typy paralelizácie a to na dátovú a úlohovú paralelizáciu so zreteľom na špecifikované požiadavky. Dátová paralelizácia bola analyzovaná a implementovaná pomocou nástroja PDI a to konkrétne rozdelením dátového toku na viacero častí a transformáciou jednotlivých dátových blokov kópiami krokov v nástroji PDI. Taktiež boli analyzované iné možnosti pre dátovú paralelizáciu v nástroji PDI, ktoré môžu byť použité v budúcnosti, ak by boli vyžadované. Úlohová paralelizácia a ostatné požiadavky boli implementované pomocou trojvrstvovej aplikácie v jazyku Java s využitím Kettle Java API. Aplikácia poskytuje správu nahrávaní, správu závislostí nahrávaných komponent či možnosti získania logovacích informácií z nahrávania. Riešenie je postavené na databázových tabuľkách, ktoré tvoria jadro celej implementácie. Na správu tabuliek a interakciu s aplikáciou je poskytnuté API rozhranie. V riešení je taktiež umožnené nahrávanie jednotlivých tabuliek dátového skladu, čo doteraz s ohľadom na ETL procesy dátového skladu nebolo možné. Jednotlivé úlohy sú nahrávané paralelne podľa definovaného poradia nahrávania, ktoré je obsiahnuté v grafe závislostí. Riešenie taktiež zaručuje integritu dát po ukončení nahrávania definovaním povinných hrán grafu. Zjednodušili sa aj niektoré PDI úlohy a transformácie spojené s nahrávaním, čím sa predišlo duplikovaniu súborov s rovnakou funkcionalitou a znížilo sa množstvo potencionálnych chýb počas nahrávania. V poslednej kapitole bolo zhodnotené riešenie z pohľadu splnenia požiadaviek a zhodnotené možnosti využitia v ETL procesoch Dátového skladu ČVUT.

Na základe ďalšej diskusie bude v rámci tímu dátového skladu zhodnotené, či sa využije riešenie tejto bakalárskej práce alebo riešenie súbežnej bakalárskej práce. Prínosom tohto riešenia bude urýchlenie nahrávacích procesov a zjednodušenie nahrávania jednotlivých tabuliek dátového skladu. To vytvorí priestor pre pravidelnejšie nahrávanie dát a tým pádom aj možnosť poskytovať kvalitnejšie výstupy z dátového skladu v kratšej dobe. V budúcnosti by bolo vhodné na toto riešenie nadviazať tvorbou frontendovej časti aplikácie, keďže väčšina ETL alebo orchestračných nástrojov obsahuje aj grafické rozhranie. Taktiež by bolo vhodné optimalizovať ETL procesy pre jednotlivé tabuľky analyzovaním množstva spracovávaných dát, rýchlosti databázového pripojenia k databáze či analyzovaním pravidelnosti a objemu zmien dát v zdrojovej tabuľke.

Literatúra

[1] KOTLÁŘ, Robert a Jakub KREJČÍ. *Podnikové datové sklady (NI-EDW): 2. přednáška* [online]. 2022-03-09. [cit. 2023-02-05]. Dostupné z: `https://courses.fit.cvut.cz/NI-EDW/@B212/lectures/02_prednaska.pdf`.

[2] PEDAMKAR, Priya. Kimball vs Inmon. In: *EDUCBA* [online]. [cit. 2023-04-11]. Dostupné z: `https://www.educba.com/kimball-vs-inmon/`.

[3] INMON, William H. *Building the Data Warehouse*. 3. vydanie. John Wiley & Sons, Inc., 2002. 412. ISBN 0-471-08130-2.

[4] TAMIMI, Naser. Fundamentals of Data Warehouses for Data Scientists. In: *Towards Data Science* [online]. [cit. 2023-04-24]. Dostupné z: `https://towardsdatascience.com/fundamentals-of-data-warehouses-for-data-scientists-5314a94d5749`.

[5] NAEEM, Tehreem. Data Warehouse Concepts: Kimball vs. Inmon Approach. In: *Astera Software* [online]. [cit. 2023-04-11]. Dostupné z: `https://www.astera.com/type/blog/data-warehouse-concepts/`.

[6] ASANKA, Dinesh. Implementing Slowly Changing Dimensions (SCDs) in Data Warehouses. In: *SQLShack* [online]. [cit. 2023-02-07]. Dostupné z: `https://www.sqlshack.com/implementing-slowly-changing-dimensions-scds-in-data-warehouses/`.

[7] ROSS, Margy. Design Tip #152 Slowly Changing Dimension Types 0, 4, 5, 6 and 7. In: *Kimball Group* [online]. [cit. 2023-02-07]. Dostupné z: `https://www.kimballgroup.com/2013/02/design-tip-152-slowly-changing-dimension-types-0-4-5-6-7/`.

[8] ETL (Extract, Transform, Load). In: *IBM* [online]. [cit. 2023-04-11]. Dostupné z: `https://www.ibm.com/topics/etl`.

[9] KIMBALL, Ralph a Joe CASERTA. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley Publishing, Inc., 2004. 491. ISBN 0-764-57923-1.

[10] BARTLEY, Kevin. ETL vs ELT: What's the Difference?. In: *Rivery* [online]. [cit. 2023-02-08]. Dostupné z: `https://rivery.io/blog/etl-vs-elt/`.

[11] FRIEDLAND, David. ETL vs ELT: We Posit, You Judge. In: *Innovative Routines International* [online]. [cit. 2023-02-08]. Dostupné z: `https://rivery.io/blog/etl-vs-elt/`.

[12] DEARMER, Abe. What Is ETLT? Merging the Best of ETL and ELT Into a Single ETLT Data Integration Strategy. In: *Integrate.io* [online]. [cit. 2023-02-08]. Dostupné z: `https://www.integrate.io/blog/what-is-etlt/`.

[13] SELLIS, Timos K. a Alkis SIMITSIS. ETL Workflows: From Formal Specification to Optimization. In: IOANNIDIS, Yannis, Boris NOVIKOV a Boris Rachev, eds. *Advances in Databases and Information Systems*. Berlin: Springer-Verlag, 2007, pp. 1-11. ISBN 978-3-540-75184-7.

[14] SEENIVASAN, Dhamotharan. ETL (Extract, Transform, Load) Best Practices. In: *International Journal of Computer Trends and Technology* [online]. January 2023, Vol. 71, issue 1, pp. 40-44. [cit. 2023-04-24]. ISSN 2231–2803. Dostupné z: doi: 10.14445/22312803/IJCTT-V71I1P106 Dostupné tiež z: `https://www.researchgate.net/publication/368300449_ETL_Extract_Transform_Load_Best_Practices`.

[15] Task Parallelism vs Data Parallelism. In: *All Programming Tutorials* [online]. [cit. 2023-03-07]. Dostupné z: `https://www.allprogrammingtutorials.com/tutorials/task-parallelism-vs-data-parallelism.php`.

[16] REINDERS, James. Understanding task and data parallelism. In: *ZDNET* [online]. [cit. 2023-03-07]. Dostupné z: `https://www.zdnet.com/article/understanding-task-and-data-parallelism-3039289129/`.

[17] TVRDÍK, Pavel. *Paralelní a distribuované programování (NI-PDP): 1. přednáška* [online]. [cit. 2023-03-07]. Dostupné z: `https://courses.fit.cvut.cz/MI-PDP/media/lectures/NI-PDP-Prednaska01-Introduction-HandOut.pdf`.

[18] VITOROVIĆ, Aleksandar, Milo V. TOMAŠEVIĆ a Veljko M. MILUTINOVIĆ. Manual Parallelization Versus State-of-the-Art Parallelization Techniques. In: *Advances in Computers* [online]. 2014, Vol. 92, pp. 203–251.

[cit. 2023-04-24]. ISSN 0065-2458. Dostupné z: doi: 10.1016/b978-0-12-420232-0.00005-2.

[19] Hitachi Vantara. Pentaho products. In: *Hitachi Vantara* [online]. [cit. 2023-02-28]. Dostupné z: `https://help.hitachivantara.com/Documentation/Pentaho/9.4/Products`.

[20] Hitachi Vantara. Pentaho Data Integration. In: *Hitachi Vantara* [online]. [cit. 2023-02-28]. Dostupné z: `https://help.hitachivantara.com/Documentation/Pentaho/9.4/Products/Pentaho_Data_Integration`.

[21] CASTERS, Matt, Roland BOUMAN a Jos van DONGEN. *Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration*. Wiley Publishing, Inc., 2010. 674. ISBN 978-0-470-63517-9.

[22] ROUSE, Margaret. Cross-Cutting Concern. In: *techopedia* [online]. [cit. 2023-03-23]. Dostupné z: `https://www.techopedia.com/definition/25133/application-programming-interface-api-java`.

[23] KUZNETSOV, Stanislav. *Datový sklad fakulty*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

[24] KOTLÁŘ, Robert. *Datový sklad ČVUT - způsoby datové integrace*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

[25] MAKARA, Adam. *Návrh a implementace klientské části a rolí systému Reports*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

[26] KREJČÍ, Jakub. *Návrh datových vrstev pro datový sklad ČVUT*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

[27] PLETICHA, Ondřej. *Webová aplikace pro monitoring datového skladu ČVUT*. Diplomová práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

[28] Třívrstvá architektura (Three-tier architecture). In: *Management mania* [online]. [cit. 2023-04-23]. Dostupné z: `https://managementmania.com/cs/trivrstva-architektura-three-tier-architecture`.

[29] DRESLER, Robert. *Vícevrstvé architektury aplikací* [online]. [cit. 2023-04-23]. Dostupné z: `http://www.robertdresler.cz/2011/04/vicevrstve-architektury-aplikaci.html`.

Dodatok A

Validácia acyklickosti grafu pomocou algoritmu Top sort

```
create function executor_app.fc_dag_validation_top_sort()
returns trigger as $$
declare
    nodes_without_incoming_edge integer[];
begin
-- implementation of top sort algorithm to check
-- if the insertion or update of an edge
-- doesn't create cycle in the directed acyclic graph

-- temporary copy of the original table
drop table if exists edges_tmp;
create temp table edges_tmp as
    select distinct start_vertex, end_vertex
    from executor_app.edge;

-- selection of nodes which don't have incoming edges
nodes_without_incoming_edge := array(
        (select distinct start_vertex
         from edges_tmp
         union
         select distinct end_vertex
         from edges_tmp)
         except
         select distinct end_vertex
         from edges_tmp
    );
```

```
-- while there are nodes without incoming edges, these nodes
-- are removed from the graph and then the selection of nodes
-- is done again
-- function cardinality returns the total number of elements
-- in the array
while cardinality(nodes_without_incoming_edge) != 0 loop
    delete from edges_tmp where start_vertex =
        any(nodes_without_incoming_edge);

    nodes_without_incoming_edge := array(
        (select distinct start_vertex
         from edges_tmp
         union
         select distinct end_vertex
         from edges_tmp)
         except
         select distinct end_vertex
         from edges_tmp
    );
end loop;

-- if there are no nodes without incoming edges but there
-- are still nodes in the graph that means that the graph
-- does have a cycle -> insertion/update of new edge would
-- create a cycle thus it is forbidden to do so
if exists(select * from edges_tmp) then
    raise exception
        'Trying to create cycle in directed acyclic graph '
        'structure by edge %->%, id_edge: %',
            new.start_vertex, new.end_vertex, new.id_edge;
end if;

return new;

end; $$ language plpgsql;

create trigger tr_validate_dag after insert or update
    on executor_app.edge
    for each row
    execute procedure executor_app
            .fc_dag_validation_top_sort();
```

Dodatok B

Volanie pre získanie komponent potrebných k nahratiu

```
@Query(nativeQuery = true, value =
    "with recursive vertexesToLoad(vertex, id_edge) as ( " +
    "    select start_vertex as vertex, array[id_edge] " +
    "    from executor_app.edge " +
    "    where end_vertex in (:vertexIds) " +
    "    union " +
    "    select end_vertex as vertex, array[id_edge] " +
    "    from executor_app.edge " +
    "    where start_vertex in (:vertexIds) and mandatory " +
    "    union all " +
    "    select * from ( " +
    "    with vertexes as (select * from vertexesToLoad) " +
    "        select e1.start_vertex, array_append( " +
    "                    cte.id_edge, e1.id_edge) " +
    "        from executor_app.edge e1 " +
    "            join vertexes cte on e1.end_vertex " +
    "                                 = cte.vertex " +
    "        where not e1.id_edge = any(cte.id_edge) " +
    "        union " +
    "        select e2.end_vertex, array_append( " +
    "                    cte.id_edge, e2.id_edge) " +
    "        from executor_app.edge e2 " +
    "            join vertexes cte on e2.start_vertex " +
    "                                 = cte.vertex " +
    "        where e2.mandatory = true " +
    "          and not e2.id_edge = any(cte.id_edge)) t " +
    ") " +
```

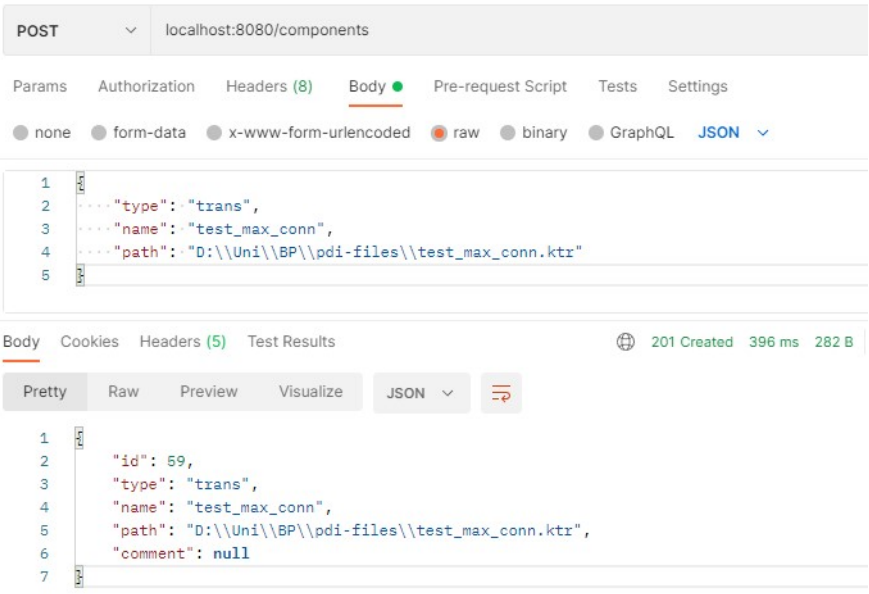
```
    "select distinct vertex from vertexesToLoad " +
    "union  " +
    "select id_component from executor_app.component " +
        "where id_component in (:vertexIds)")
Set<Integer> findComponentsRequiredToLoad(
    @Param("vertexIds") Set<Integer> vertexIds);
```

Dodatok C

Ukážka použitia riešenia a nevalidných požiadaviek

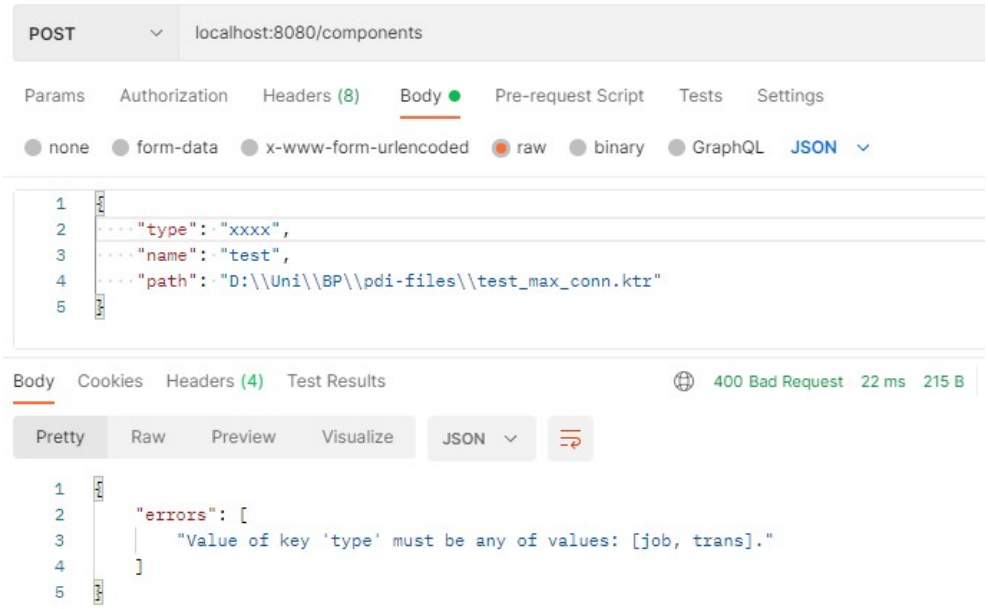
V tomto dodatku je ukázané použitie implementovanej aplikácie. Ako je možné spravovať aplikáciu pomocou API rozhrania a taktiež ukážka fungovania aplikácie, či reakcií na nevalidné požiadavky.

Pridanie komponenty je zobrazené na obrázku C.1 volaním POST požiadavky cez program Postman. Takto boli pridané aj ostatné komponenty a to č. 59 až 64.



Obr. C.1: Pridanie novej komponenty

Na obrázku C.3 je možné vidieť pridanie hrán do grafu nahrávania pomocou POST požiadavky. Hrana z komponenty č. 59 do komponenty č. 61 je

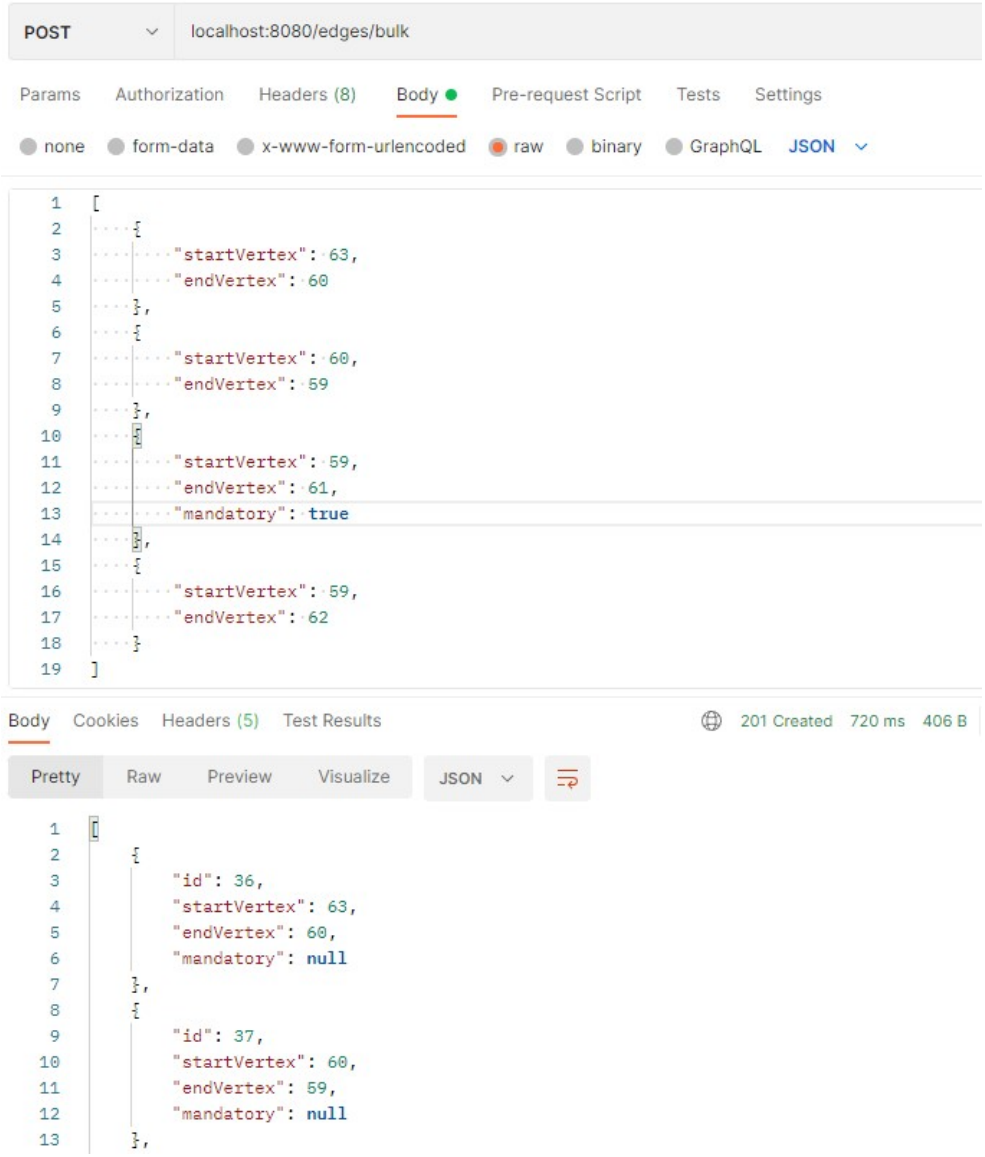


Obr. C.2: Chybová správa pri nesprávnom type komponenty

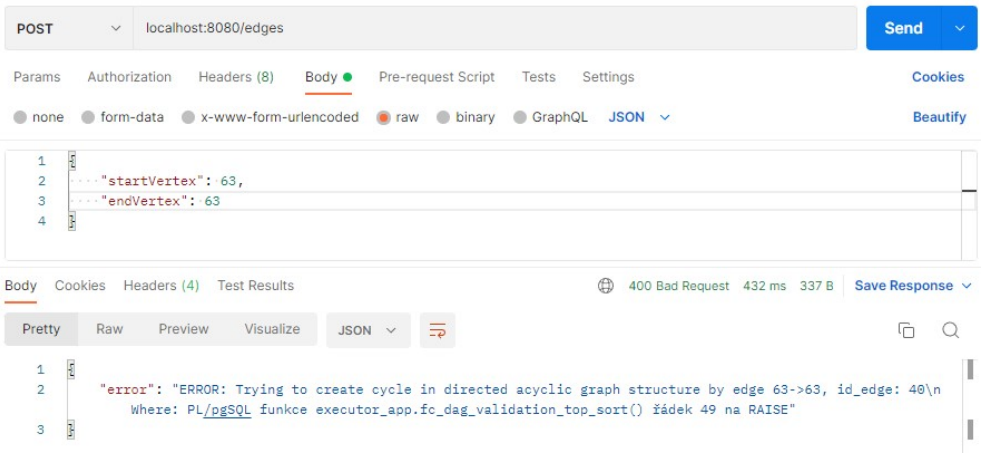
označená ako povinná. Komponenta č. 64 nie je prítomná v grafe. Na obrázku C.4 je ukázaná chybová hláška ak sa používateľ pokúsi vytvoriť v grafe cyklus.

Na obrázku C.5 je zobrazené volanie POST požiadavky na začatie nahrávania komponenty č. 59. Počas testovania komponenty č. 59 a č. 61 využívali väčšie množstvo pripojení než bolo povolené atribútom *MAX_CONNECTIONS* nastavenom na pripojení. To by malo vyústiť ku chybe v nahrávaní a zablokovať nahrávanie komponenty č. 61. Komponenta č. 61 by mala byť zablokovaná ešte predtým než by bol zisťovaný počet použitých pripojení.

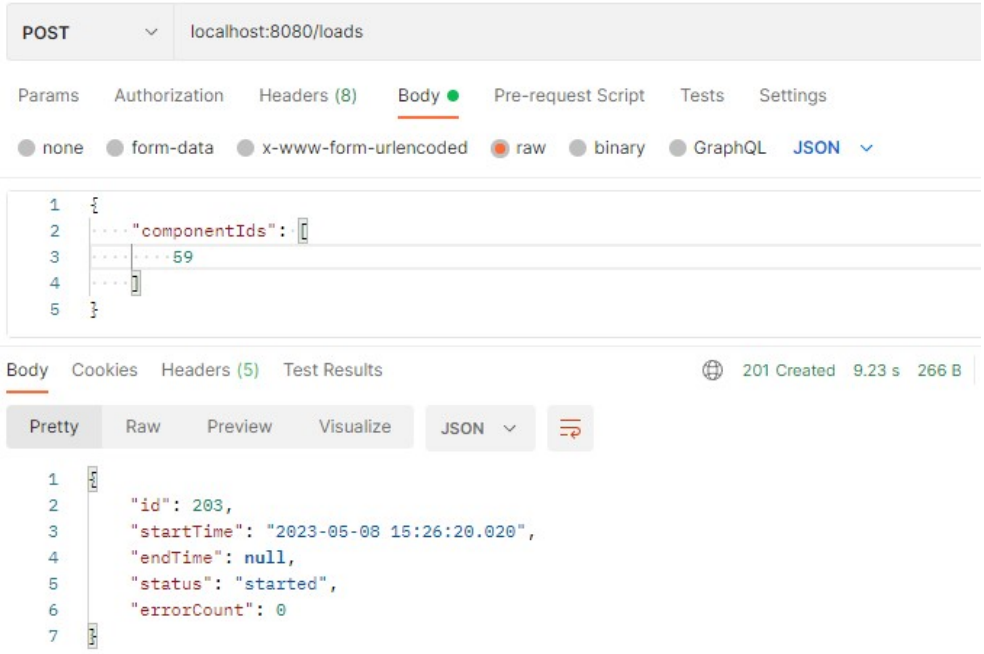
V interných logoch aplikácie je možné vidieť, že pri komponente č. 59 bolo zistené väčšie množstvo pripojení než je povolené a teda jej nahrávanie bolo skončené v stave *error*. Bola teda zablokovaná komponenta č. 61, čo je možné vidieť na obrázku C.8. V logu na obrázku C.7 je ukázané, že do nahrávania boli pridané komponenty č. 63, 60, 59 a 61 čo odpovedá logike grafu a aplikácie. Na obrázku C.9 sú dostupné logy po ukončení nahrávania. Na ďalších obrázkoch sú zobrazené rôzne chybové správy pri nesprávnom používaní.



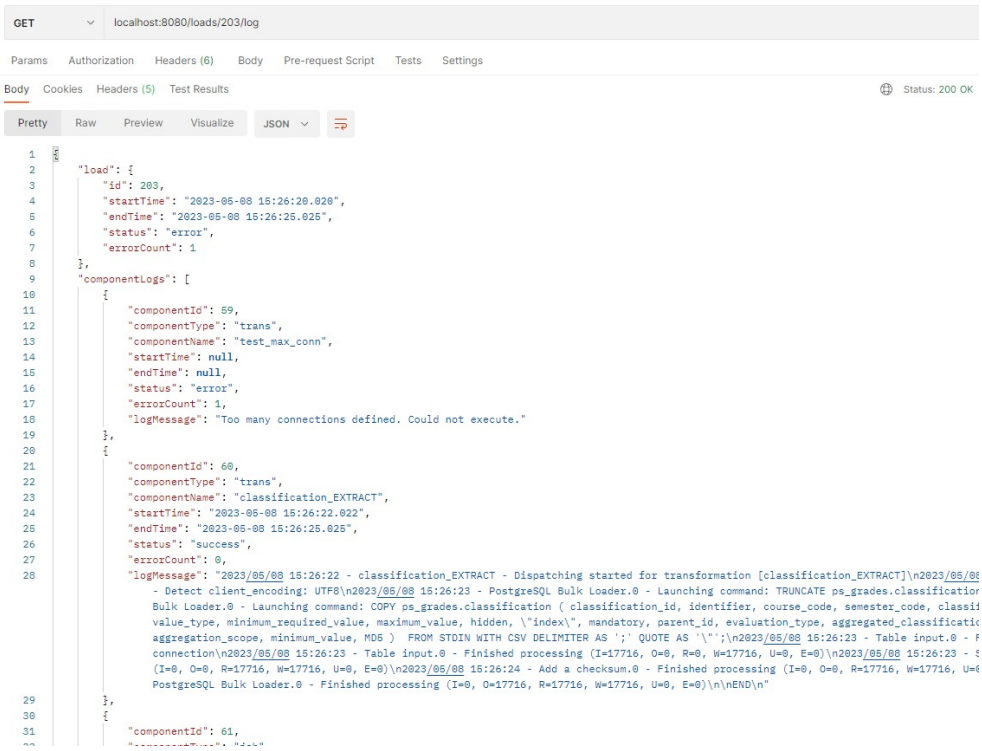
Obr. C.3: Pridanie hrán do grafu komponent



Obr. C.4: Chybová hláška v prípade vytvorenia cyklu v grafe



Obr. C.5: Spustenie nahrávania pre komponentu č. 59



Obr. C.6: Ukážka logovacej informácie z nahrávania

```
2023-05-08 15:26:20.361  WARN --- c.c.d.e.business.services.LoadService: LOAD: STARTED, load ID: 203
2023-05-08 15:26:20.363  WARN --- c.c.d.e.business.services.LoadService: PREPARED, component ID: 59
2023-05-08 15:26:20.409  WARN --- c.c.d.e.business.services.LoadService: PREPARED, component ID: 60
2023-05-08 15:26:20.432  WARN --- c.c.d.e.business.services.LoadService: PREPARED, component ID: 61
2023-05-08 15:26:20.450  WARN --- c.c.d.e.business.services.LoadService: PREPARED, component ID: 63
```

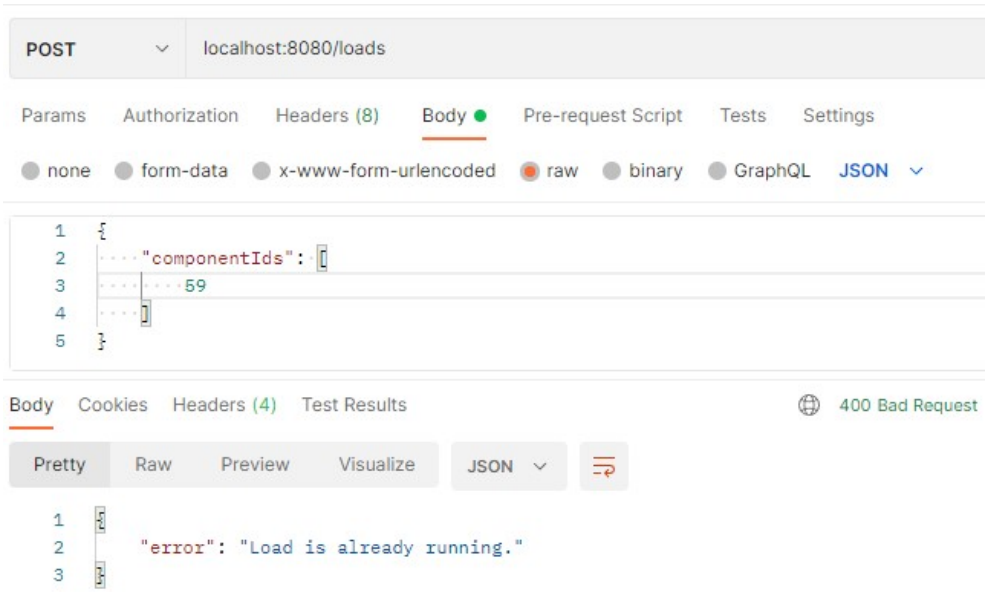
Obr. C.7: Ukážka logu aplikácie po začatí nahrávania

```
2023-05-08 15:26:25.340 ERROR --- c.c.d.e.business.services.LoadService: ERROR, Too many connections defined, component ID: 59
2023-05-08 15:26:25.363  WARN --- c.c.d.e.business.services.LoadService: BLOCKED, component ID: 61
2023-05-08 15:26:25.396 ERROR --- c.c.d.e.business.services.LoadService: ERROR, component ID: 59
```

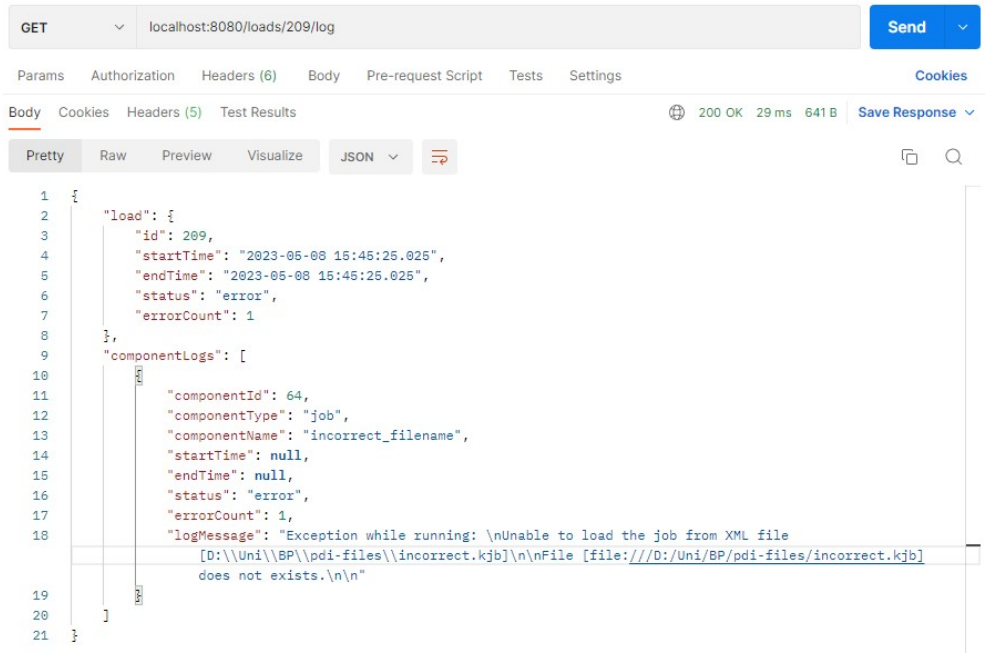
Obr. C.8: Ukážka logu aplikácie s chybou a blokovaním komponenty

```
2023-05-08 15:26:25.437  WARN --- c.c.d.e.business.services.LoadService: Shutdown of Kettle environment.
2023-05-08 15:26:25.506  WARN --- c.c.d.e.business.services.LoadService: LOAD: error count = 1, load ID: 203
2023-05-08 15:26:25.507  WARN --- c.c.d.e.business.services.LoadService: LOAD: ERROR, load ID: 203
2023-05-08 15:26:25.579  WARN --- c.c.d.e.business.services.LoadService: Ended load and cleared used resources.
```

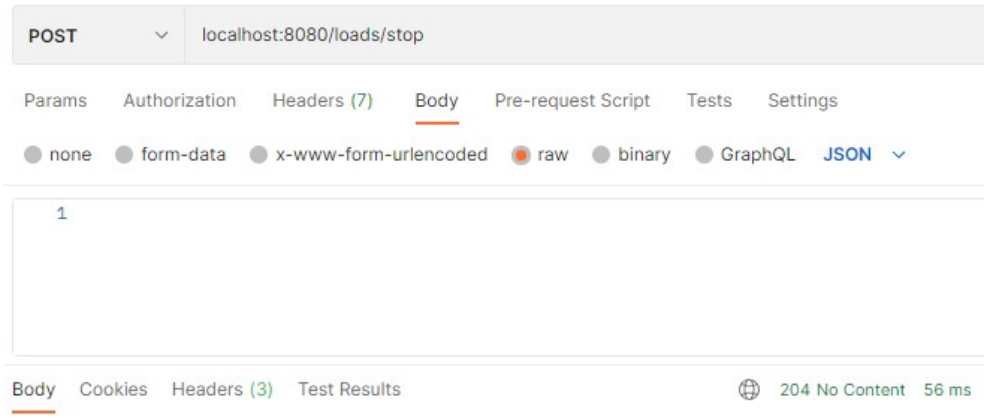
Obr. C.9: Ukážka logu aplikácie po ukončení nahrávania



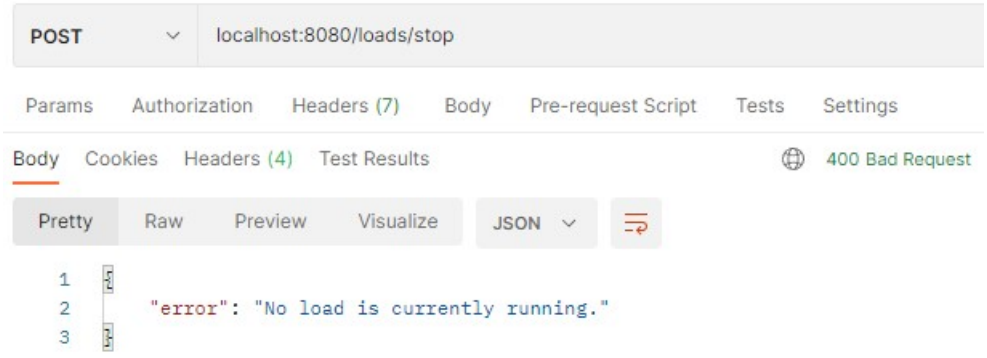
Obr. C.10: Chybová správa pri spustení nahrávania ak už beží iné nahrávanie



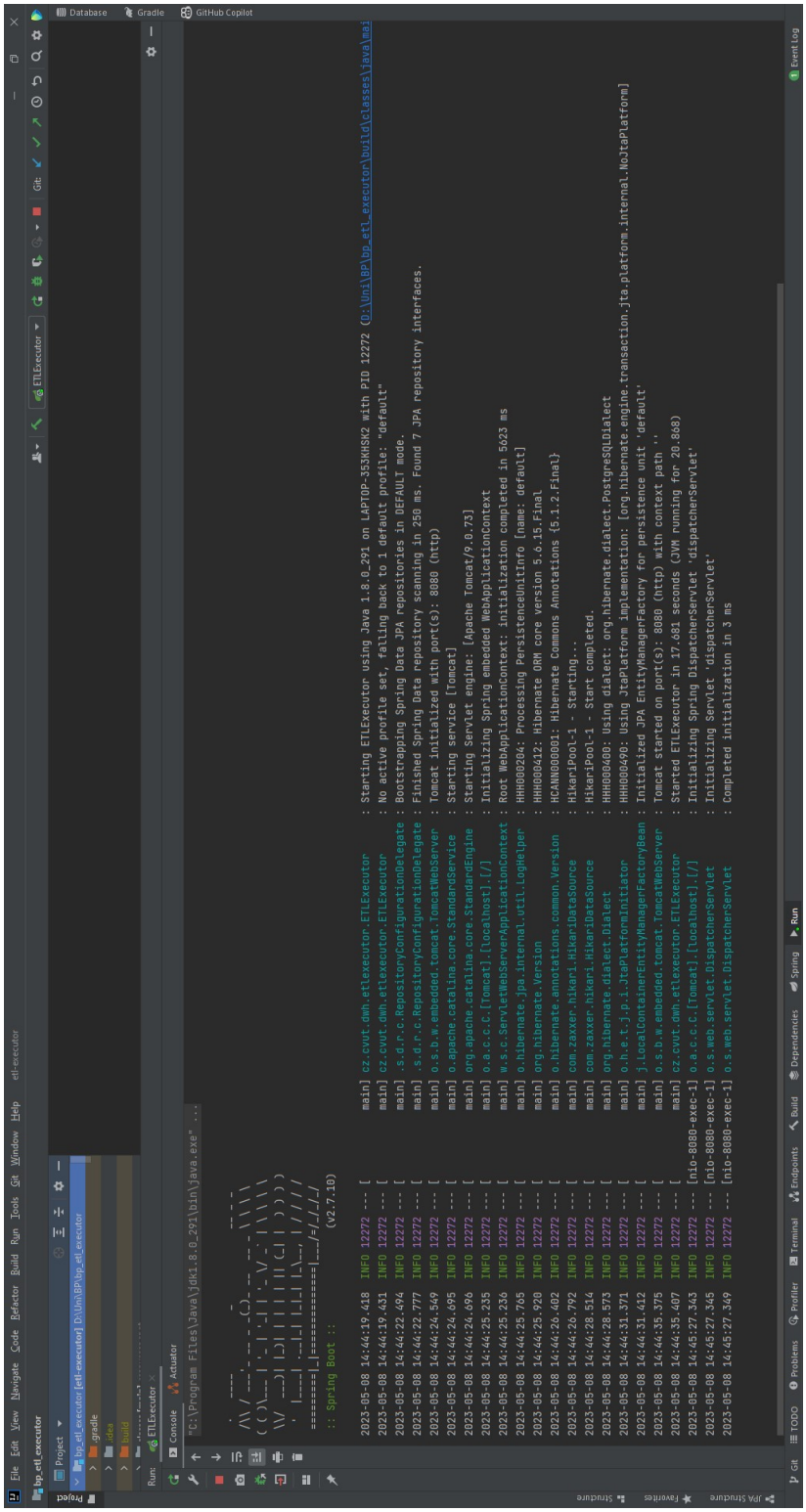
Obr. C.11: Chybová správa pri nahrávaní súboru s nesprávnou adresárovou cestou



Obr. C.12: Úspešné volanie o zastavenie nahrávania



Obr. C.13: Chybová hláška pri pokuse o zastavenie nahrávania ak žiadne nebeží



Obr. C.14: Ukážka spustenia riešenia v IntelliJ IDEA

Dodatok D

Zoznam použitých skratiek

API Application programming interface

CSV Comma-separated values

ČVUT České vysoké učení technické

DAG Directed acyclic graph

DAO Data access objects

DLM Dátová logická mapa

DTO Data transfer objects

DW Data Warehouse

ETL Extract, Transform, Load

ELT Extract, Load, Transform

ETLT Extract, Transform, Load, Transform

HTTP Hypertext transfer protocol

IDL Integrated data layer

JSON JavaScript Object Notation

JPA Java persistance API

LGPL Lesser General Public License

ORM Object-relational mapping

PBL PostgreSQL bulk loader

PDI Pentaho Data Integration

POC Proof of concept

PS Pre stage

PSC Pre stage clean

SCD Slowly changing dimensions

SI Stage increment

SQL Structured query language

URL Uniform resource locator

VPN Virtual private network

XML Extensible markup language

Dodatok **E**

Obsah priloženého archívu

- `readme.md` ..... stručný popis obsahu prílohy
- `bp_etl_executor/` ..... zdrojové kódy implementácie
- `ETL_executor_jobs/` ..... PDI súbory spúšťané pri paralelizácií
- `stage_increment_procedury/` ..... priečinok s procedúrami zo stage_increment
- `src_latex/` ..... zdrojové kódy práce vo formáte LaTeX
- `license.txt` ..... licencia pre distribúciu zdrojových kódov
- `BP_Marhefka_Adam_2023.pdf` ..... text práce vo formáte PDF