FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Datový sklad ČVUT - import dat v nástroji Apache Airflow
Student:	Jiří Lejsek
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Softwarové inženýrství 2021
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

Cílem práce je navázat na Proof of Concept (PoC) řešení nahrávání dat do datového skladu ČVUT (DW ČVUT), který je výsledkem bakalářské práce Kristiny Zolochevské. V této práci dojde k revizi PoC a pilotnímu nasazení v rámci běžného zpracování dat při provozu DW ČVUT.

1. Seznamte se s bakalářskou prací Kristiny Zolochevské a nástrojem Apache Airflow, se kterým pracovala.
2. Proveďte případnou revizi jejího řešení ve smyslu nových verzí a přístupů v nástroji Apache Airflow (jsou-li).
3. Navrhněte a implementujte pravidelné nahrávání dat ze zdrojových databází do DW ČVUT, podle dílčího zadání z oddělení VIC.
4. Zhodnoťte navrženou a realizovanou metodu z hlediska efektivity a udržitelnosti.

Výsledkem práce bude implementace celého procesu nahrávání dat (vybrané podmnožiny zdrojových databází) včetně jeho automatizace.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 2. února 2024 v Praze.

Bakalářská práce

DATOVÝ SKLAD ČVUT - IMPORT DAT V NÁSTROJI APACHE AIRFLOW

Jiří Lejsek

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Michal Valenta, Ph.D.
14. května 2024

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2024 Jiří Lejsek. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení, je nezbytný souhlas autora.

Odkaz na tuto práci: Lejsek Jiří. *Datový sklad ČVUT - import dat v nástroji Apache Airflow.* Bakalářská práce. České vysoké učení technické v Praze, Fakulta informačních technologií, 2024.

Obsah

Poděkování vi

Prohlášení vii

Abstrakt viii

Seznam zkratek x

1 Teoretický úvod 2
- 1.1 Business Intelligence . . . 2
- 1.2 Datový sklad . . . 2
  - 1.2.1 Architektura dle Inmona . . . 3
  - 1.2.2 Architektura dle Kimballa . . . 4
  - 1.2.3 Srovnání Inmonova a Kimballova pohledu . . . 5
  - 1.2.4 Základní vrstvy datového skladu . . . 6
- 1.3 ETL . . . 7
- 1.4 Datový sklad ČVUT . . . 7
  - 1.4.1 Architektura . . . 8
  - 1.4.2 Nahrávací ETL job . . . 10

2 Popis technologie Apache Airflow 15
- 2.1 Úvod do Apache Airflow . . . 15
- 2.2 Architektura . . . 16
  - 2.2.1 Jednotky práce . . . 17
- 2.3 Uživatelské rozhraní . . . 20
- 2.4 Další konstrukty . . . 21
  - 2.4.1 Task pools . . . 21
  - 2.4.2 Priorita tasků . . . 21
- 2.5 Plánování úloh – Scheduler . . . 22
- 2.6 Spouštění úloh – Executor . . . 22
- 2.7 Průzkum nových verzí . . . 23

3 Návrh řešení 24
- 3.1 Odchylka od PoC . . . 24
- 3.2 Hlavní výzvy paralelní implementace . . . 25
  - 3.2.1 Závislosti mezi úlohami . . . 25

3.2.2 Parametry paralelismu . . . 26
3.3 Hlavní důvody zrychlení . . . 27
3.4 Členění DAGů . . . 27
3.4.1 Hierarchie DAGů . . . 28
3.4.2 Scénáře nahrávání pro jednotlivá mapování . . . 29
3.5 Problémy se samostatným nahráváním jednotlivých tabulek . . . 29
3.5.1 Manuální nahrání tabulky po pádu automatického nahrávání 29
3.5.2 Mimořádné manuální nahrání tabulky mimo automatické nahrávání . . . 30
3.5.3 Nahrávání tabulek s různou frekvencí . . . 31
3.6 Nežádoucí výkyvy doby běhu . . . 31

4 Popis implementace 33
4.1 Nalezení závislostí . . . 33
4.1.1 Databázový model pro závislosti . . . 34
4.2 Listové DAGy . . . 35
4.3 Vnitřní DAGy . . . 36
4.4 Kořenový DAG . . . 37
4.5 Chybové scénáře . . . 37
4.6 Ostatní použité konstrukty . . . 38
4.6.1 Použití task pools . . . 38
4.6.2 Řešení nežádoucích výkyvů doby běhu . . . 39
4.6.3 Použití tagů . . . 39
4.6.4 Priorita tasků . . . 39
4.6.5 Optimalizace mimo Airflow . . . 40

5 Zhodnocení 42
5.1 Zhodnocení z hlediska efektivity . . . 42
5.2 Zhodnocení z hlediska udržitelnosti . . . 42
5.3 Silné stránky řešení . . . 43
5.4 Slabé stránky řešení . . . 44
5.5 Úkoly do budoucna . . . 44
5.6 Zhodnocení splnění zadání . . . 46

A Screenshoty z Airflow 48

Obsah příloh 54

Seznam obrázků

1.1 Inmonova architektura datového skladu 4
1.2 Kimballova architektura datového skladu 5
1.3 Vrstvy datového skladu dle Inmonovy architektury 6
1.4 Současná architektura DW ČVUT 8
1.5 Současný produkční ETL job 11
1.6 Check job 11
1.7 detail jobu *J_MAKE_INCREMENT* 12
1.8 Job *J_IDL_LOAD* 13
1.9 Transformace nahrávající tabulku do IDL 14

2.1 Architektura Apache Airflow 16
2.2 Jednoduchá ukázka DAGu 18
2.3 Detail DAGu 21

4.1 Databázové schéma pro reprezentaci závislostí 34
4.2 Listový DAG 36
4.3 Vnitřní DAG 37
4.4 Kořenový DAG 41

A.1 Přehled DAGů v uživatelském rozhraní Airflow 49
A.2 Ganttův diagram pro typický běh kořenového DAGu 50
A.3 Ganttův diagram pro největší tabulku (typický běh) 51
A.4 Ganttův diagram pro největší tabulku při vysokém počtu modifikovaných záznamů 51

Seznam tabulek

5.1 Porovnání dob běhu 43

Seznam výpisů kódu

1.1 SQL kód spouštěný krokem *clean_stage* 12
1.2 SQL kód spouštěný krokem *get_M_N_D_flags* 12

2.1 Příklad deklarace DAGu 17
2.2 Způsoby tvorby závislostí mezi tasky 18
2.3 Příklad vytvoření tasku přes BashOperator 19

4.1 SQL kód spouštěný taskem *get_M_N_D_flags* 35

Nejprve bych chtěl poděkovat ***Ing. Michalu Valentovi, Ph.D.*** *za kvalitní vedení práce, lidský přístup a vůbec všechno, co pro mě v tomto nelehkém období udělal.*
Mé velké díky patří také ***Bc. Adamu Makarovi*** *za všechny důležité rady ohledně datového skladu, které mi s obdivuhodnou ochotou neustále dává. Nedokáži si představit, že bych tuto práci dokázal vyhotovit bez nich.*
Děkuji také ***Mgr. Janu Anderlemu****,* ***Ing. Robertu Kotlářovi*** *a* ***Ing. Ondřeji Guthovi, Ph.D.*** *za dílčí, avšak neméně hodnotnou pomoc s textovou částí práce.*
V neposlední řadě bych chtěl poděkovat své rodině za její obrovskou trpělivost a bezpodmínečnou podporu, bez které by mé studium bylo nemyslitelné.

Prohlášení

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

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

V Praze dne 14. května 2024

Abstrakt

Práce se zabývá paralelizací ETL procesů datového skladu ČVUT, které nahrávají data z nejdůležitějších informačních systémů univerzity do centrální databáze datového skladu. Hlavním cílem této paralelizace je urychlení celého nahrávacího procesu. Pro vývoj výsledného řešení byla vybrána technologie Apache Airflow. Vybrána byla na základě průzkumu, který ve svých bakalářských pracích na fakultě informačních technologií ČVUT v minulosti provedli Kristina Zolochevskaia a Adam Marhefka.

V rámci práce bylo úspěšně navrženo a implementováno paralelní řešení nahrávání datového skladu ČVUT, které je aktuálně nasazeno v testovacím prostředí, kde je dále optimalizováno a připravováno na produkční nasazení.

Vytvořené řešení bylo podrobeno více než deseti testovacím běhům, které oproti původnímu řešení vykazují více než trojnásobné zrychlení nahrávacího procesu. Tyto výsledky otevírají možnost přechodu z týdenního nahrávání na denní nahrávání.

Klíčová slova paralelizace ETL, datový sklad ČVUT, Apache Airflow, DAG, PL/pgSQL

Abstract

The thesis deals with the parallelization of ETL processes of the CTU data warehouse that load data from the university's critical information systems to the central database of the data warehouse. The main goal of the parallelization is to speed up the whole process. The Apache Airflow technology was chosen for the development of the final solution. Its choice was based on research conducted by Kristina Zolochevskaia and Adam Marhefka in their bachelor's theses at the Faculty of Information Technology CTU.

A parallel solution of data input into the CTU data warehouse was successfully designed and implemented in the thesis. It is currently deployed in a testing environment, where it is being further optimized and prepared for the production deployment.

The solution underwent more than ten test runs that showed more than a triple increase in the speed of the loading process in comparison with the former standard. These results open the possibility of advancing from weekly loading to daily loading.

Keywords ETL parallelization, data warehouse CTU, Apache Airflow, DAG, PL/pgSQL

Seznam zkratek

3NF	3. Normální Forma
BFS	Breadth First Search (prohledávání do šířky)
BI	Business Intelligence
BK	Business Key
ČVUT	České Vysoké Učení Technické
DAG	Directed Acyclic Graph (orientovaný acyklický graf)
DDL	Data Definition Language
DLM	Datová Logická Mapa
DW	Data Warehouse (datový sklad)
ELT	Extract, Load, Transform
ETL	Extract, Transform, Load
FDW	Foreign Data Wrapper
IDL	Integrated Data Layer
JVM	Java Virtual Machine
PDI	Pentaho Data Integration
PL/pgSQL	Procedural Language/PostgreSQL
PoC	Proof of Concept
SCD	Slowly Changing Dimension
SQL	Structured Query Language
UDE	Uměle Definované Entity
VIC	Výpočetní a Informační Centrum

Úvod

Datový sklad ČVUT je projekt, který má po datové stránce ambici být „hlavním zdrojem pravdy“ na univerzitě. Do jednotného datového modelu integruje nejdůležitější informační systémy na univerzitě – například systém KOS, Usermap nebo Anketa. Důležitá data z těchto systémů pak historizuje (nejen) pro datově analytické účely. V současné době datové výstupy datového skladu ČVUT podporují rozhodování nejvyššího vedení univerzity, ale jsou používány i na dalších místech, například na webových stránkách některých fakult.

Datový sklad ke svému provozu potřebuje pravidelné nahrávání dat ze zdrojových systémů do své centrální databáze. V současnosti toto nahrávání probíhá pouze jednou týdně, protože je implementováno sekvenčně a zabírá příliš času na to, aby bylo možné data nahrávat každý den. Každodenní nahrávání by však přineslo výrazné zpřesnění dat, po kterém je velká poptávka.

Tato bakalářská práce se zabývá právě tímto problémem. Navazuje na bakalářské práce Kristiny Zolochevské a Adama Marhefky, které v roce 2023 vznikaly souběžně a zkoumaly možnosti, jak lze nahrávání vhodně paralelizovat (a tedy urychlit). Základy pro tuto práci však přinesla zejména práce Kristiny Zolochevské, neboť navrhla technologii Apache Airflow, ke které se vedení projektu nakonec přiklonilo. Práce má ambici toto paralelní nahrávání implementovat a ve funkční podobě ho nasadit do testovacího prostředí, kde bude testováno, dolaďováno a připravováno na nasazení do produkčního prostředí.

Teoretická část má za cíl nejprve zavést pojmy business intelligence, datových skladů a ETL procesů, dále přiblížit současný stav datového skladu ČVUT se zaměřením na podobu současného ETL řešení, které bylo vytvořeno v ETL nástroji Pentaho Data Integration a data do skladu nahrává sekvenčně. Dalším cílem teoretické části je popsat technologii Apache Airflow a mechanismy, které tato technologie pro implementaci paralelního řešení nabízí.

Cílem praktické části je navrhnout a implementovat samotné paralelní zpracování a následně ho nasadit do testovacího prostředí. Dalším cílem praktické části je zhodnotit výsledné řešení, porovnat ho s původním řešením a diskutovat další postup, který bude muset být proveden před nasazením řešení do produkce.

Kapitola 1
Teoretický úvod

Vzhledem k tomu, že se tato práce věnuje výhradně datovému skladu, je nejprve potřeba představit čtenáři ústřední pojmy a uvést ho do kontextu. V této kapitole bude představen pojem Business Intelligence a pojem datových skladů, které jsou spolu nerozlučně spjaty. Dále se čtenář dozví, co jsou to ETL[1] procesy, které jsou jsou pro datový sklad nepostradatelné. Nakonec bude stručně představena aktuální architektura a podoba DW ČVUT[2] se zaměřením na současně používané ETL řešení, které do DW ČVUT jednou týdně nahrává data ze zdrojových systémů. Popis stávajícího řešení bude sloužit především pro srovnání s paralelním řešením (popsaným v kapitole 4).

1.1 Business Intelligence

Pojmem Business Intelligence (BI) je rozuměn soubor postupů, úloh a technologií, sloužících téměř výlučně k podpoře analytických, plánovacích a rozhodovacích činností podniků a organizací. V dnešní době stále častěji tvoří běžnou součást řízení podniků a do značné míry ovlivňuje jejich konkurenceschopnost. [1]

Business Intelligence není žádná konkrétní technologie, ale je to celá businessová disciplína. Datové sklady jsou sice její důležitou komponentou, avšak Business Intelligence má smysl i bez datových skladů, byť má v takové podobě značně limitovaný rozsah analýz. Datový sklad (popsaný v následující podkapitole) by mohl být vytvořen i za jiným účelem, než je využití v Business Intelligence, v drtivé většině takových případů by však byl neobhajitelně drahý. [2]

1.2 Datový sklad

„*Datový sklad je informační systém, který dlouhodobě ukládá data z informačních*

[1] Extract, Transform, Load
[2] Datového skladu ČVUT

systémů společnosti za účelem potenciálního analytického využití. Tato data bývají často integrována do jednoho datového modelu a bývají uchovávány i historické hodnoty jednotlivých datových entit.“ [3]

Toto je jedna z mnoha možných definic datového skladu, která podle autora dobře slouží pro intuitivní pochopení tohoto pojmu.

S datovými sklady jsou od doby jejich vzniku až do dneška nerozlučně spojována dvě jména – William H. Inmon a Ralph Kimball. W. H. Inmon je považován za „otce datových skladů“. Pojem poprvé definoval ve své knize *Building the Data Warehouse* v roce 1992. Jeho definice je do dneška jedna z nejpopulárnějších. Ralph Kimball je považován za zakladatele dimenzionálního modelování, který také zadefinoval historizaci dat (SCD[3]). Ten svou definici datového skladu poprvé uvedl v roce 1996. Tito dva pánové mají ke tvorbě a architektuře datových skladů odlišné přístupy.

1.2.1 Architektura dle Inmona

Podle pana W. H. Inmona je Datový sklad subjektově orientovaná, integrovaná, stálá a časově rozptýlená kolekce dat sloužící k podpoře rozhodování vedení podniku. [4]

Následuje popis jednotlivých vlastností z této definice:

- **Subjektově orientovaná** Tato vlastnost je chápána tak, že datový sklad se na strukturu organizace dívá z pohledu konkrétních subjektů, které v organizaci figurují: V případě tovární výrobny by to mohl být například zákazník, objednávka a produkt, v případě vysoké školy například student, učitel, předmět a přihláška.

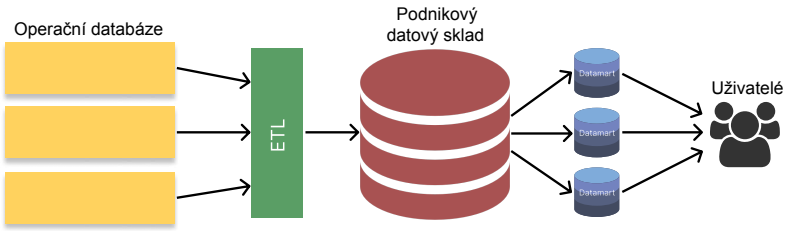
- **Integrovaná** Integrovanost je nejdůležitější charakteristikou dat v datovém skladu. Typická organizace ve svém běžném fungování používá velké množství informačních systémů, přičemž každý systém má svůj vlastní datový model a kontext využití. Když byly tyto systémy v minulosti vytvářeny, tak autoři jejich datových modelů obvykle nepočítali s možností, že by příslušná data mohla být v budoucnosti integrovaná s jinými daty z jiného informačního systému. Datový sklad je postaven nad organizací jakožto celkem, a musí tato data z různých kontextů konzistentně integrovat do sjednoceného kontextu, což je jednou z velkých výzev při jeho vytváření.

- **Stálá** Zde jde o fakt, že samotná data, která se do skladu již nahrála, se v pravém slova smyslu nemění. Data jsou do skladu nahrána v podobě časového snímku, který reflektuje přesný stav zdrojového systému v nějakém přesném, konkrétním čase. Tedy záznam, který byl ve zdrojovém systému například vymazán, je v datovém skladu pouze zneplatněn, ale stále je dohledatelný.

[3] Slowly Changing Dimension

- **Časově rozptýlená** Poslední charakteristická vlastnost dat v datovém skladu souvisí se stálostí dat a znamená, že ze skladu můžeme dostat nejen aktuální data, ale i ta historická. Záznamy jsou obvykle historizovány pomocí časových razítek, které uvádí časové rozmezí platnosti daného záznamu. [4]

Inmonově architektuře se také říká **Hub and Spoke** a jeho přístupu k celému vývoji datového skladu se říká **top-down** (shora dolů). Vývoj datového skladu dle Inmonovy filosofie probíhá tak, že jsou nejprve důkladně zanalyzovány všechny zdrojové systémy v organizaci, které mají být datovým skladem obsažené. Následuje náročný vývoj integrovaného datového modelu a až na konci jsou vytvořena datová tržiště, která definují podobu dat, jež jsou ze skladu poskytována odběratelům. [5]



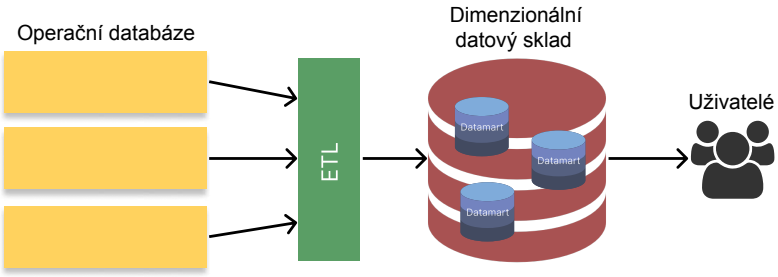
Obrázek 1.1 Inmonova architektura datového skladu. Původně z [6], upraveno v [7].

1.2.2 Architektura dle Kimballa

Pro srovnání a druhý pohled se sluší uvést konkurenční definici od Ralpha Kimballa. Ten ve své knize [8] říká, že datový sklad je dotazovatelný zdroj dat v podniku a není ničím jiným, než sjednocením všech obsažených datových tržišť.

Ralph Kimball nejprve popsal koncept architektury nezávislých datových tržišť, anglicky nazvaný **independent data marts**, kde jsou jednotlivá datová tržiště samostatně „životaschopná“. Později vzhledem k vývoji požadavků na BI řešení tento koncept přepracoval a zavedl nový koncept sběrnicové architektury, anglicky zvaný **data mart bus**. Oproti původnímu přístupu, kde jsou na sobě datová tržiště zcela nezávislá, se sběrnicová architektura odlišuje snahou tato datová tržiště do jisté míry integrovat, a to pomocí tzv. sdílených dimenzí. [1]

Kimballův přístup k vývoji datového skladu bývá nazýván **bottom-up** (zdola nahoru). Kimballův pohled na vývoj datového skladu spočívá v tom, že je nejprve potřeba analyzovat, která data podnik v datovém skladu potřebuje mít, a až potom se hledá místo, kde k těmto datům přijít. Kimball zastává myšlenku pro každý větší proces či subjekt v podniku vytvořit jedno nezávislé datové tržiště – datovým skladem potom nazývá sjednocení všech těchto datových tržišť. [8, 9]



Obrázek 1.2 Kimballova architektura datového skladu. Původně z [6], upraveno v [7].

1.2.3 Srovnání Inmonova a Kimballova pohledu

Mezi těmito dvěma architekturami je hodně rozdílů. Datový sklad postavený na Inmonově architektuře obsahuje centrální databázi, kde jsou data skutečně integrovaná. V této centrální databázi jsou data normalizovaná dle 3NF,[4] což zabraňuje redundanci dat a umožňuje se na datový sklad dívat jako na „jednotný zdroj pravdy“. Další výhodou této architektury je, že v ní lze snadněji integrovat nový zdrojový systém a také lze snadněji poskytovaná data přizpůsobit novým požadavkům. Je důležité připomenout, že centrální databáze je navržena nad celým podnikem.[5] Hlavní nevýhodou Inmonova přístupu je nutnost velké počáteční investice do tvorby datového skladu, protože návrh zmíněné centrální databáze vyžaduje expertní a dlouhotrvající analýzu.

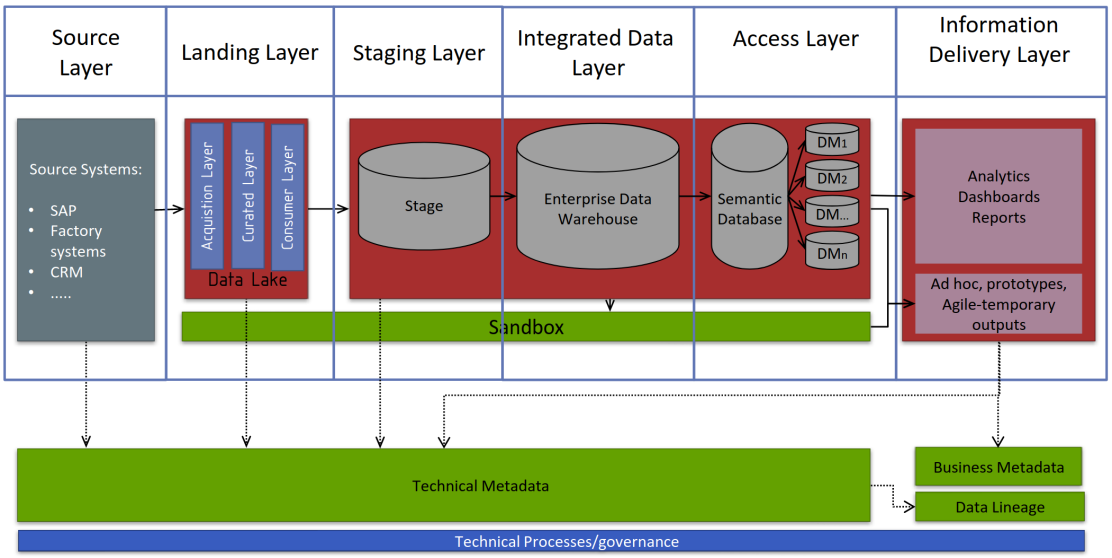
Datový sklad postavený na Kimballově architektuře vzniká po jednotlivých datových tržištích, kde se každé z nich zaměřuje na některý větší proces v podniku a je přizpůsoben konkrétním požadavkům např. z jednotlivých oddělení. Data jsou převážně integrovaná pouze v rámci svého datového tržiště, což vede k možným redundancím dat napříč datovými tržišti. Vzhledem k tomu, že zde není potřeba vytvořit centrální datový model nad celým podnikem, vyžaduje tato metoda oproti Inmonově metodě značně nižší počáteční náklady (finanční i časové), což je její hlavní výhodou. Datový sklad s touto architekturou je pak náročnější na změny a údržbu, nemusí nutně obsahovat „jedinou verzi pravdy“ a také má za důsledek náročnější datovou analýzu napříč datovými tržišti [5, 9]

Studie [9] z roku 2012 uvádí, že v tomto roce 35 % dotázaných subjektů používalo Inmonovu *Hub and Spoke* architekturu, 15 % dotázaných subjektů používalo architekturu nezávislých datových tržišť a 23 % dotázaných subjektů používalo Kimballovu *data mart bus* architekturu.

Je dobré zmínit, že DW ČVUT je postaven převážně na konceptech Inmonovy architektury.

[4] 3. Normální Forma
[5] Respektive jeho nejdůležitějšími částmi

1.2.4 Základní vrstvy datového skladu



Obrázek 1.3 Vrstvy datového skladu dle Inmonovy architektury [10]

Obrázek 1.3 vyobrazuje standardní podobu vrstev (**Layers**) dnešního typického datového skladu:

- **Source Layer:** Tato vrstva reprezentuje zdrojové systémy a další případné datové zdroje.
- **Landing Layer:** Volitelná vrstva, která slouží k přípravě extrahovaných dat do Staging Layer. Staging Layer je již obsažena přímo v databázi a Landing Layer je vhodné použít pouze v případě, že by data nebylo možné nahrát přímo do Staging Layer – například v případě nutnosti změny kódování.
- **Staging Layer:** První vrstva, která se nachází přímo v databázi datového skladu. Je sem nahráván přesný otisk dat ze zdrojových systémů. Tato vrstva slouží k validaci a případnému technickému očištění dat před jejich nahráním do IDL.
- **Integrated Data Layer (IDL):** Tuto vrstvu lze považovat za „srdce“ datového skladu. Obsahuje centrální databázi s integrovaným datovým modelem a tabulky v této vrstvě často obsahují historizovaná data.
- **Access Layer:** Zde se nachází jednotlivá datová tržiště, která slouží k poskytování dat odběratelům. Datová tržiště obvykle zobrazují pouze aktuálně platná data (nikoliv ta historická) a mohou obsahovat různé agregace dat z IDL. Access Layer také často obsahuje sémantickou databázi, která sjednocuje konvence pro reprezentaci dat v datových tržištích a zajišťuje tím „jednotnost pravdy“, kterou datová tržiště poskytují.

- **Information Delivery Layer**[6]**:** Slouží přímo pro poskytnutí přístupu uživatele k datům. Jedná se o reportingové nástroje (např. Power BI), různé analytické technologie a další možné formy datových výstupů. [3]

1.3 ETL

ETL[7] je proces, který extrahuje, transformuje a nahrává data z různých zdrojů do datového skladu. Podle různých pravidel čistí, reorganizuje a připravuje různorodá data do konsolidované podoby, která je vhodná pro uložení dat za účelem datové analytiky. [11]

Nejprve je na řadě **extrakce** – ETL nástroj zkopíruje surová data do tzv. **staging area**, což lze chápat jako dočasné úložiště extrahovaných dat, nad kterým je následně prováděna transformace. Toto úložiště je po transformaci a nahrání dat do cílového úložiště často zcela přemazáno, to ale záleží na implementaci konkrétního systému.

Dále přichází na řadu **transformace** dat. Podle potřeb konkrétního případu užití může docházet k různým typům transformace – typicky je zde zvyšována kvalita dat odstraňováním chyb a duplicitních dat nebo sjednocením jejich formátu. Také jsou zde data často propojována, nebo naopak rozdělována.

Nakonec přichází **nahrání** transformovaných dat do cílové databáze, typicky do IDL. Nahrána mohou být buďto všechna transformovaná data (**full load**), což se obvykle děje při prvním nahrávání dat do datového skladu, nebo jsou nahrávána pouze data, která se od posledního běhu ETL procesu změnila (**incremental load**). [11]

ETL procesy jsou zpravidla dávkově orientované a v drtivé většině organizací jsou řádně automatizované. V minulosti organizace často vyvíjely ETL procesy ručně, ale s vývojem této disciplíny se brzy objevilo velké množství ETL nástrojů, jak open-source, tak komerčních. ETL procesy mají své alternativy, například ELT[8] procesy, které nejprve nahrají surová data přímo do cílového úložiště a transformaci provádějí až v něm. ETL i ELT procesy mají v závislosti na povaze projektu svá pro a proti. [12]

1.4 Datový sklad ČVUT

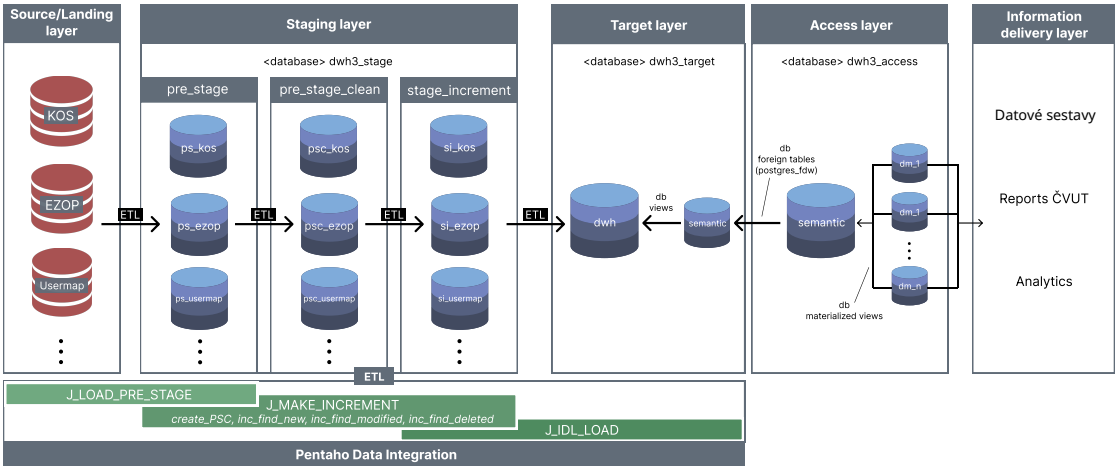
Datový sklad ČVUT následuje Inmonovu architekturu a integruje nejdůležitější informační systémy na univerzitě – v současnosti se jedná o systémy KOS, Usermap, Anketa, ProjectsFIT, grades a ezop. Díky jednotnosti dat poskytuje datové výstupy pro podporu rozhodování vedení univerzity, nebo poskytuje data například pro weby některých fakult. V poslední době neustále přibývají nové žádosti

[6] Neplést s Integrated Data Layer, pro kterou je používána zkratka IDL
[7] Extract, Transform, Load
[8] Extract, Load, Transform

o datové výstupy a také v minulosti vznikl požadavek na zpřesnění dat, který byl hlavním důvodem ke vzniku této práce. Data jsou do skladu totiž nahrávána jednou týdně, ale často jsou po DW ČVUT požadované datové výstupy k některému konkrétnímu dni – proto je potřeba nahrávání skladu urychlit a z týdenního nahrávání přejít na denní.

1.4.1 Architektura

Obrázek 1.4 znázorňuje současnou architekturu DW ČVUT. Následuje popis jednotlivých vrstev:



Obrázek 1.4 Současná architektura DW ČVUT. Původně z [2], upraveno v [7].

- **Source/Landing Layer** představuje zdrojové systémy a několik manuálně připravovaných datových exportů, které sem pravidelně přidávají pracovníci DW ČVUT (pro tuto práci nejsou důležité).
- **Staging Layer** (neboli „stage“) je tvořena samostatnou databází s názvem `dwh3_stage`. Ta slouží jako pomyslné „odkladiště“ dat ze zdrojových systémů, kde se označují změny, které jsou pak nahrávány do IDL. Dělí se na tři části, které v jistém smyslu odpovídají fázím transformace dat před nahráním do IDL:
  - **pre-stage (ps).** Sem jsou nahrávána data přesně v podobě, ve které se nacházejí ve zdrojových systémech. Jediná změna je přidání MD5 hashe, která dále slouží k rozpoznávání změn.
  - **pre-stage-clean (psc).** Sem přichází data z pre-stage, která jsou technicky očištěna. Tím je myšleno zejména zahození nepotřebných sloupců a neplatných či duplicitních řádků.

- **stage-increment (si).** Tato část slouží k rozpoznávání změn a dále plní úlohu „odrazového můstku“ pro nahrávání dat do IDL. Zejména kvůli auditovatelnosti záznamů obsahuje technické sloupce `active`, `insertion_date` a `last_update`. Klíčovým technickým sloupcem je pak sloupec `state`, který obsahuje příznaky: **N** (New – nový záznam), **M** (Modified – modifikovaný záznam) a **D** (Deleted – smazaný záznam).

Každý zdrojový systém má v každé z těchto částí své schéma – například zdrojový systém KOS má tedy v databázi `dwh3_stage` schémata `ps_kos`, `psc_kos` a `si_kos`. V pre-stage-clean mají svou obdobu pouze některé tabulky, protože u některých tabulek není technické čištění vyžadováno.

Ve staging layer se dále vyskytuje schéma UDE,[9] které má pouze svou `psc_` a `si_` variantu. Jedná se o tabulky, které v žádném ze zdrojových systémů nefigurují, ale pro potřeby ETL procesů DW ČVUT byly interně vytvořeny, a to obvykle propojením několika tabulek, které spolu ve zdrojových systémech nijak propojené nejsou. Důvodem je zejména zamezení několikanásobnému použití operace JOIN ve vybraných ETL procesech [2]. Existence schématu UDE bohužel zkomplikovala návrh grafu závislostí (viz obrázek 4.4).

- **Integrated Data Layer (IDL)** (neboli „target“), kde se nachází centrální databáze `dwh3_target` s integrovaným datovým modelem. Ve schématu `dwh` se nachází historizovaná data – jednotlivé tabulky jsou vybaveny technickým klíčem a technickými sloupci `version`, `date_from`, `date_to` a `last_update`. Sloupce `date_from` a `date_to` ohraničují dobu platnosti dané verze záznamu. Historizace probíhá následovně:
  - Pokud je záznam nový, pak je do IDL nahrán s `version` nastavenou na hodnotu `1`, `date_from` na hodnotu `1.1.1900` (pomyslné záporné nekonečno), `date_to` na hodnotu `31.12.2199` (pomyslné nekonečno) a `last_update` na čas, kdy byl daný záznam nahrán do datového skladu.
  - Pokud je záznam změněný, pak je původní verze záznamu zneplatněna nastavením `date_to` na čas nahrání modifikovaného záznamu. Na stejnou hodnotu je nastaven sloupec `date_from` v novém záznamu, `date_to` je pak opět nastaveno na pomyslné nekonečno. Sloupec `version` je pak v nově platném záznamu o jedna vyšší, než v původním.
  - Pokud je záznam smazán, dojde pouze k jeho zneplatnění ve sloupci `date_to`.

V IDL se dále nachází schéma `semantic`, které z integrovaného modelu ve schématu `dwh` pomocí databázových pohledů provádí selekci dat, která jsou potřebná v některém z datových tržišť. Tyto pohledy obvykle čerpají z více různých tabulek a definují formát dat, ve kterém jsou data poskytována v datových výstupech. Zároveň téměř výlučně filtrují historické záznamy a poskytují pouze ty aktuální.

[9] Uměle Definované Entity

- **Access Layer** je reprezentovaná samostatnou databází `dwh3_access`. Rovněž obsahuje schéma `semantic`, které pomocí mechanismu FDW[10] získává data ze stejnojmenného schématu v IDL. Data ve schématu `semantic` jsou pak čerpána jednotlivými datovými tržišti, a to konkrétně formou materializovaných pohledů. Datová tržiště obsahují data, která jsou poskytována koncovým uživatelům dle požadavků. Obvykle nad atomickými daty ze schématu `semantic` provádí různé agregace a výpočty, opět podle požadavků na datové výstupy.
- **Information Delivery Layer** je vrstva, na které se již pohybují koncoví uživatelé, kteří datové výstupy používají. Operují nad ní systémy pro reporting a vizualizaci dat,[11] které čerpají data přímo z datových tržišť.

1.4.2 Nahrávací ETL job

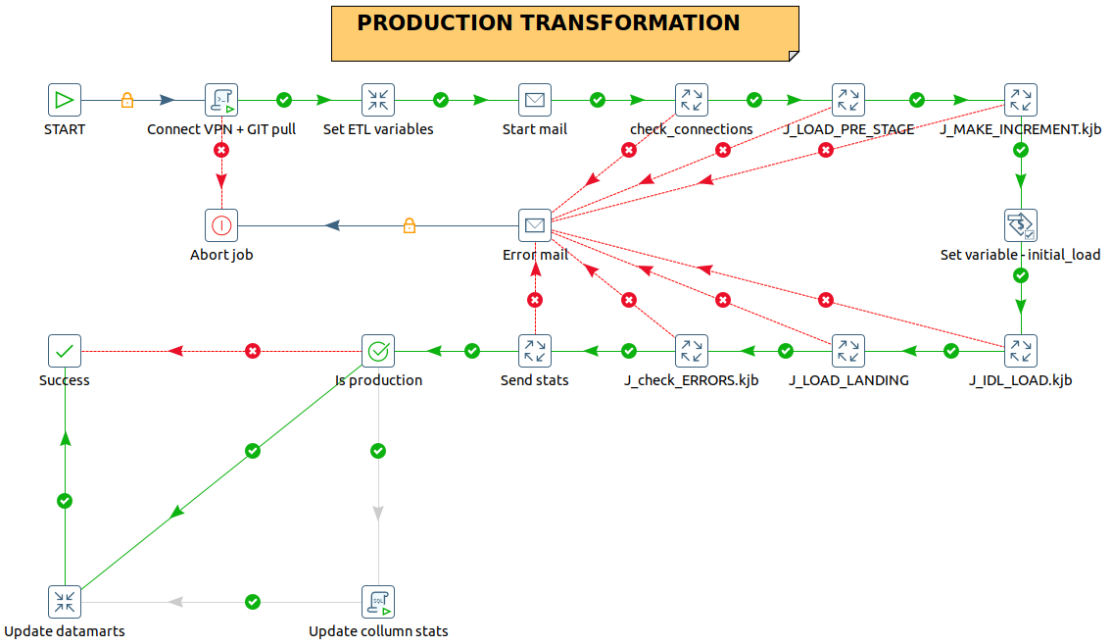
Data jsou do skladu nahrávána pomocí ETL nástroje PDI.[12] PDI je intuitivní open-source nástroj pro zpracovávání a transformaci velkých objemů dat. Je napsáno v javě, takže je kompatibilní s libovolnou platformou podporující JVM.[13] V PDI existují tři základní jednotky práce:

- **Job.** Orchestruje a spouští další joby a transformace, může obsahovat i jednotlivé kroky.
- **Transformace.** Obsahuje jednotlivé kroky a správně by měla mít na starosti jednu věc – například nahrání jedné tabulky nebo jednu část nahrávání jedné tabulky.
- **Krok.** Atomická operace s daty. Každý krok nad daty dělá jednu ze tří operací ze zkratky ETL – Extrakci (Extract), Transformaci (Transform), Nahrání (Load). Pomocí kroků lze také spouštět různé skripty, například SQL[14] skripty.

Na obrázku 1.5 se nachází současná podoba ETL jobu, který pokrývá celý proces nahrávání dat do DW ČVUT. Je nazván *J_DWH_routine*. Jeho první varianta vznikla v diplomové práci Roberta Kotláře [2], ve které se nachází detailnější popis částí, které jsou popsány i v této bakalářské práci.

Pro potřeby této práce jsou důležité pouze joby od *J_LOAD_PRE_STAGE* po *J_LOAD_LANDING* bez kroku *Set variable – initial_load*.

[10] Foreign Data Wrapper
[11] V současné době primárně systém Reports ČVUT
[12] Pentaho Data Integration
[13] Java Virtual Machine
[14] Structured Query Language

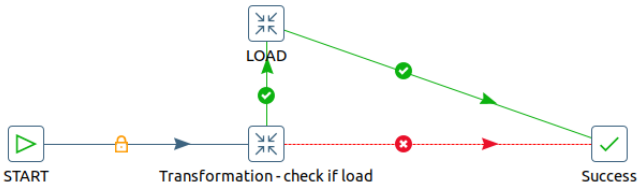


Obrázek 1.5 Současný produkční ETL job

1.4.2.1 Nahrání dat do pre-stage

Nahrání dat do pre-stage se odehrává v jobu *J_LOAD_PRE_STAGE*. V něm jsou postupně spouštěny další joby, které jsou rozdělené podle zdrojových systémů a pro každou příslušnou tabulku spouští takzvaný *check job*

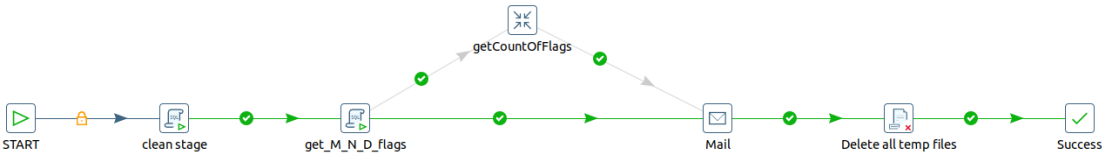
Check job se podívá do databáze a zkontroluje, jestli má tabulka být nahrána – datový sklad je totiž vybaven mechanismem, který pro jednotlivé tabulky umožňuje nastavovat frekvenci nahrávání. Pokud má tabulka být nahrána, spustí se transformace, která tabulku nahraje do pre-stage a přidá k ní textový sloupec obsahující MD5 hash. Rizika použití tohoto mechanismu budou široce diskutována v podkapitole 3.5.



Obrázek 1.6 Check job

1.4.2.2 Hledání změn

Na obrázku 1.7 jsou jednotlivé kroky, které obsahuje job *J_MAKE_INCREMENT*. Krok *clean_stage* spouští nad stage databází kód z výpisu 1.1.



Obrázek 1.7 detail jobu *J_MAKE_INCREMENT*

```
VACUUM;
ANALYZE;

BEGIN;
SELECT public.create_clean_pre_stage();
SELECT inc_clear_state_flag();
COMMIT;
```

Výpis kódu 1.1 SQL kód spouštěný krokem *clean_stage*

PL/pgSQL[15] funkce `create_clean_pre_stage` vytváří z jednotlivých pre-stage tabulek jejich pre-stage-clean varianty. Tento krok se netýká všech tabulek. Funkce `inc_clear_stage_flag` nuluje ve všech stage-increment tabulkách sloupec `state`, čímž umožní výpočet nové várky změn. Je důležité zmínit, že oba tyto plošné výpočty jsou vykonávány v jediné transakci.

Krok *get_M_N_D_flags* spouští procedury, které do stage-increment nahrají změny a označí je. Opět tak činí v jediné transakci:

```
BEGIN;
SELECT inc_find_modified_in_pre_stage();
SELECT inc_find_new_in_pre_stage();
SELECT inc_find_deleted_in_pre_stage();
COMMIT;
```

Výpis kódu 1.2 SQL kód spouštěný krokem *get_M_N_D_flags*

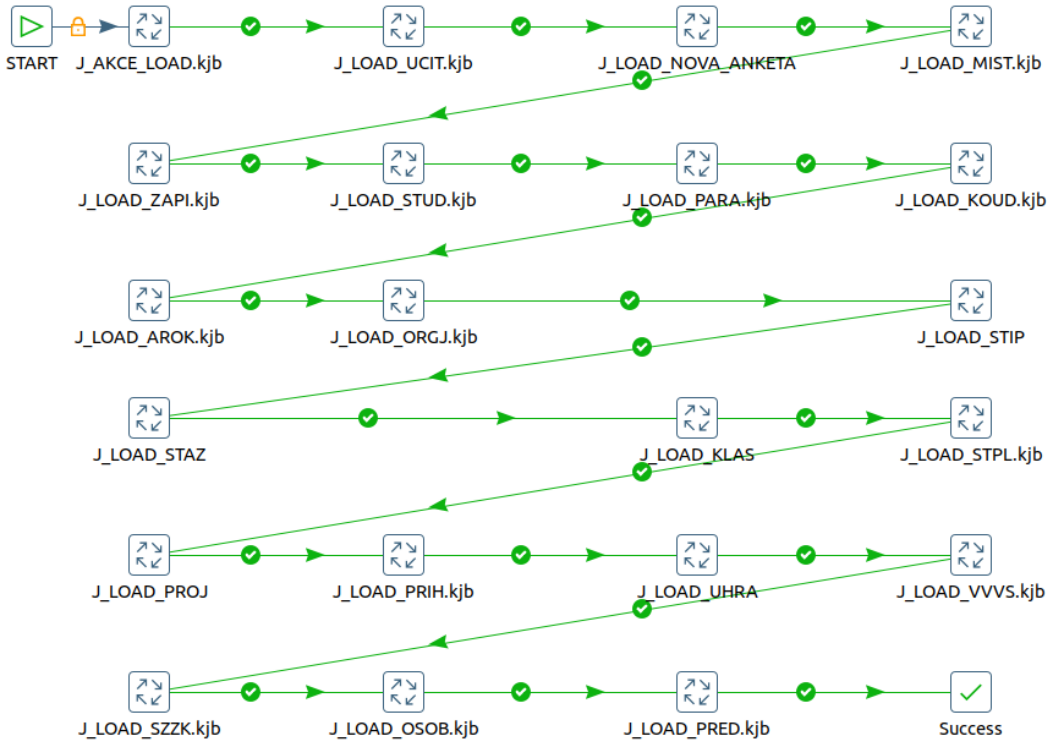
Tyto tři procedury k rozpoznávání změn používají MD5 hash a takzvané business klíče (BK). Detailnější popis tohoto mechanismu se nachází v [2].

1.4.2.3 Nahrání změn do IDL

Nahrávání změn do IDL má na starosti především job *J_IDL_LOAD*. Na obrázku 1.8 je vidět, že z pohledu IDL již nejsou tabulky děleny dle zdrojových systémů, ale dle logických celků, ke kterým v rámci IDL přísluší (toto dělení je reflektováno

[15] Procedural Language/postgreSQL

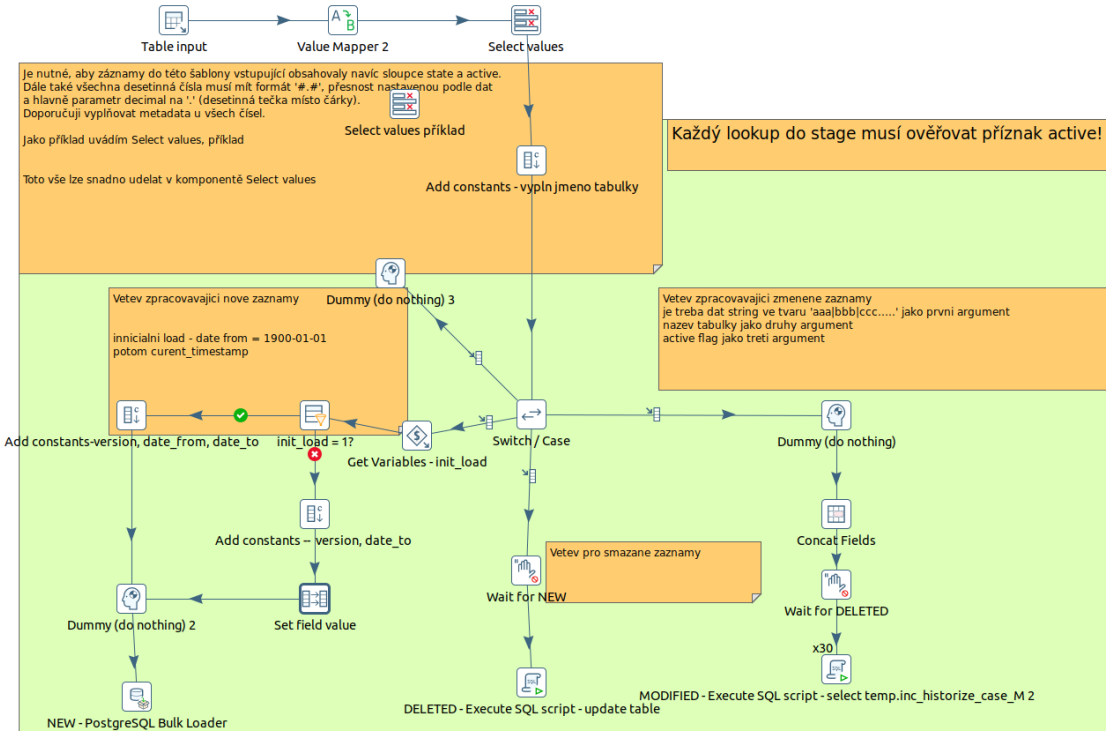
v návrhu paralelní implementace nahrávání). Každý logický celek má tedy svůj job, který sekvenčně spouští transformace pro všechny své tabulky.



Obrázek 1.8 Job *J_IDL_LOAD*

Jedna z těchto transformací se nachází na obrázku 1.9. Vychází z šablony, kterou ve své diplomové práci [2] vytvořil Robert Kotlář. Řádky jsou podle příznaku ve sloupci `state` rozdělovány do tří větví podle typu operace. Nejprve se čeká na zpracování všech nových řádků, potom na zpracování všech smazaných řádků. Nahrávání změněných řádků se provádí paralelně ve třiceti samostatných procesech.

Job *J_LOAD_LANDING* má na starosti nahrávání tabulek, které nejsou historizované. Tyto tabulky kopíruje přímo ze zdrojových systémů do IDL.



Obrázek 1.9 Transformace nahrávající tabulku do IDL

Kapitola 2
Popis technologie Apache Airflow

V této kapitole je popsána technologie Apache Airflow, která byla pro implementaci paralelního nahrávání vybrána. Technologie je popsána pouze v rozsahu, který je nutný k pochopení dalších kapitol této práce. Technologie byla vybrána na základě případových studií, které ve svých bakalářských pracích provedli Kristina Zolochevskaia a Adam Marhefka. Instalaci a potřebnou konfiguraci Airflow na serveru DW ČVUT již v rámci své práce provedla Kristina Zolochevskaia, autor se jí v této práci věnovat nebude. V této práci je používáno Airflow verze 2.6.3, ale nejnovější verze Airflow je v době psaní práce 2.9.0. Podkapitola 2.7 shrnuje funkcionality, které by upgrade Airflow na nejnovější verzi přinesl. Kapitola je převážně převzatá z oficiální dokumentace [13].

2.1 Úvod do Apache Airflow

Apache Airflow je open-source platforma pro vývoj, plánování a monitorování dávkově orientovaných **workflows**.[1]

Airflow je napsané v Pythonu a pyšní se impozantní škálovatelností. Lze ho nasadit téměř kamkoliv, od obyčejného počítače až po komplexní distribuované systémy.

Airflow také vyniká svým webovým uživatelským rozhraním, přes které lze procesy spouštět, sledovat a prohlížet jejich logy. [13]

Hlavní charakteristikou Apache Airflow je přístup **„Workflows as code“**, což znamená, že dané workflows jsou vytvářeny a konfigurovány kódem[2]. Tento přístup má několik žádoucích důsledků:

- **Dynamičnost** Umožňuje dynamické generování procesů.
- **Rozšiřitelnost** Framework Airflow obsahuje operatory, kterými lze Airflow propojit s velkým množstvím ostatních technologií. Komponenty používané

[1] Lze přeložit jako „pracovní postup“ nebo „tok úloh“.
[2] V případě Airflow je použit programovací jazyk Python

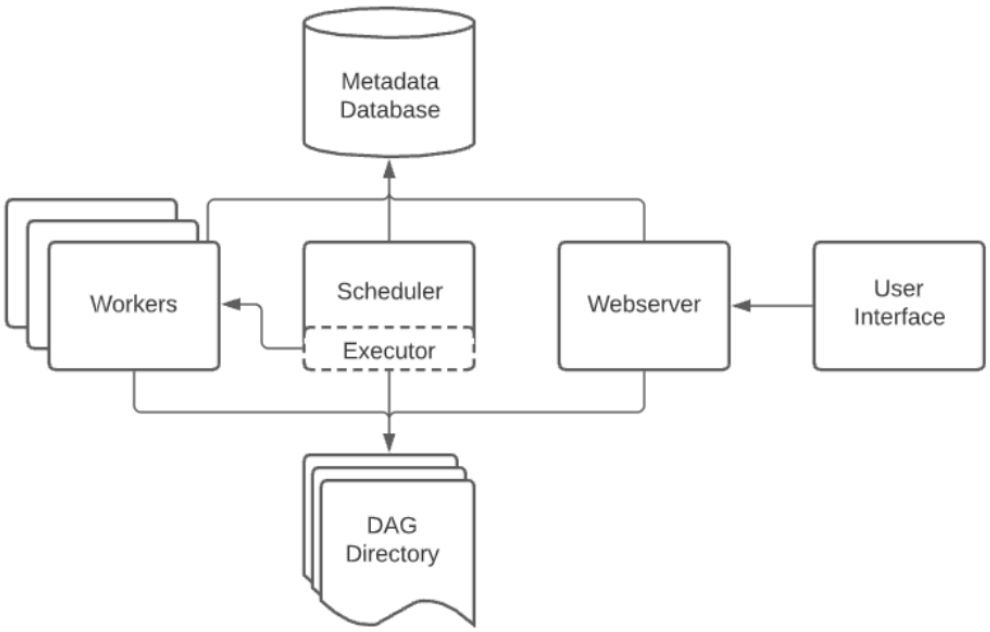
v Airflow je možné měnit, případně lze i implementovat své vlastní. K Airflow také existuje celá řada rozšíření (provider-packages), které je možné přiinstalovat.

- **Flexibilita** S využitím šablonovacího enginu jinja má Airflow vestavěnou podporu pro parametrizaci procesů.

Airflow **není** vhodné pro nekonečně běžící procesy reagující na externí akce. Airflow také není samo o sobě vhodné pro proudově orientované real-time procesy, ale často je používáno v kombinaci s proudově orientovanými systémy, jako je například Apache Kafka. [13]

2.2 Architektura

Na obrázku 2.1 je vyobrazena základní architektura Airflow.



Obrázek 2.1 Architektura Apache Airflow [13]

- **Scheduler** (plánovač) má na starosti spouštění naplánovaných workflows a odesílání jednotlivých **tasků** na **executor**, který tasky provádí.
- **Executor** (prováděč) provádí tasky, které mu přijdou od Scheduleru. V základní instalaci Airflow jsou tasky prováděny přímo ve Scheduleru, ale prakticky všechny Executory vhodné k produkčnímu nasazení delegují provádění tasků na své **Workery**.

- **Webserver** poskytuje uživateli přehledné uživatelské rozhraní (**User interface**) k procházení, spouštění a ladění DAGů[3] a tasků.
- **DAG Directory** (složka s DAGy) obsahuje veškeré DAGy a je čtena Schedulerem, Executorem a případně jeho workery.
- **Metadata Database** (Databáze s metadaty) je používána Schedulerem, Executorem a Webserverem pro čtení a ukládání stavu jednotlivých úloh. [13]

2.2.1 Jednotky práce

2.2.1.1 DAG

DAG je klíčovým konceptem Airflow. Obsahuje **tasky** (viz dále), organizuje je pomocí závislostí, které definují, kdy má který task být spuštěn. DAG může být deklarován více způsoby – zde je jeden z nich:

```
import datetime

from airflow import DAG
from airflow.operators.empty import EmptyOperator

with DAG(
    dag_id="my_dag_name",
    start_date=datetime.datetime(2021, 1, 1),
    schedule="@daily",
):
    EmptyOperator(task_id="task")
```

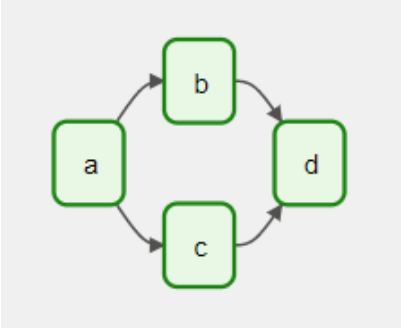
Výpis kódu 2.1 Příklad deklarace DAGu [13]

Je častým zvykem vytvořit pro každý DAG jeden soubor a autor se v této práci bude tohoto zvyku držet. V tomto souboru se potom nachází:

- **Konfigurace DAGu.** DAG musí mít nastaven unikátní identifikátor `dag_id`. Dále je zde `start_date`, které definuje, odkdy může být DAG plánován a spouštěn. Pro účely této práce postačí nastavit libovolné datum z minulosti. Parametr `schedule` určuje frekvenci, se kterou má být DAG pravidelně spouštěn schedulerem. Dále lze do DAGu předat velké množství volitelných parametrů, které zpravidla přepisují implicitní hodnoty získané z hlavního konfiguračního souboru `airflow.cfg`. Do DAGu lze přidat i vlastní parametry.

[3] Directed Acyclic Graph – orientovaný acyklický graf

- **Jednotlivé tasky.** Tasky budou vysvětleny v následující podkapitole. Každý task má na starosti různou agendu, na které z pohledu DAGu nezáleží – DAG se stará pouze o to, kdy má který task spustit, nebo například o jejich případný timeout. [13]
- **Závislosti mezi tasky.** Na obrázku 2.2 je vidět, jak uživatelské rozhraní vykresluje DAG. Tento ilustrativní DAG obsahuje 4 tasky: *a*, *b*, *c* a *d*. Nejprve se provede task *a*[4], potom se paralelně provedou tasky *b* a *c*, a po dokončení těchto obou je spuštěn task *d*. K terminologii: task *a* je *upstream task* tasků *b* a *c*, a tasky *b* a *c* jsou nazvány *downstream tasky* tasku *a*.



Obrázek 2.2 Jednoduchá ukázka DAGu [13]

Závislosti z obrázku 2.2 lze vytvořit několika způsoby. Například:

```
a >> b
a >> c
b >> d
c >> d
```

nebo

```
a >> [b, c]
[b, c] >> d
```

nebo

```
a.set_downstream(b, c)
d.set_upstream(b, c)
```

Výpis kódu 2.2 Způsoby tvorby závislostí mezi tasky, inspirováno v [13]

Důležitým pojmem je **DAG run** neboli běh DAGu. Je to objekt, který reprezentuje instanci DAGu v čase. Pokaždé, když je DAG spuštěn, se vytvoří DAG run, který provede všechny tasky, které jsou v DAGu nadefinované. DAG runů jednoho DAGu

[4]Obecně se provedou všechny tasky, které v DAGu nemají žádný upstream task.

může v jednu chvíli běžet více, ale v této práci je využití této možnosti nepřípustné. DAG run může mít 4 různé stavy – **queued**, **running**, **success**, **failed**. Koncové stavy jsou pouze **success** a **failed**. Vyhodnocují se na základě koncových stavů instancí tasků, které se v DAGu nachází. [13]

2.2.1.2 Task

Task je v Airflow základní a nedělitelnou jednotkou práce. Existují tři základní druhy tasků:

- **Operators (Operátory).** Jedná se o předdefinované šablony tasků a budou vysvětleny v podkapitole 2.2.1.3.
- **Sensors (Senzory).** Senzory jsou speciální podtřídou operátorů. Od běžných operátorů se liší tím, že nejsou plánovány DAGem, ale čekají na externí událost.
- **Vlastní Python funkce.** Pomocí *TaskFlow API* dekorátoru `@task` lze vlastní Python funkci reprezentovat jako task.

Task musí mít definované `task_id`. To musí být v rámci DAGu unikátní. Každý typ tasku má své povinné a volitelné parametry.

Stejným způsobem, jako DAG run reprezentuje instanci běhu DAGu, je task instancován do takzvané **task instance**. Ta funguje obdobně jako DAG run, ale má více možných stavů. Na základě koncových stavů task instances je rozhodováno o spouštění jejich downstream tasků a o koncovém stavu samotného DAGu. [13]

2.2.1.3 Operator

Operator je znovupoužitelná šablona pro task. Předdefinovává vnitřní logiku tasků tak, že pro vytvoření konkrétního tasku stačí vytvořit instanci operátoru a předat jí potřebné parametry. Tasky jsou pomocí nich vytvářeny deklarativně:

```
hello_world = BashOperator(
    task_id="hello_world",
    bash_command='echo "Hello World!"'
)
```

Výpis kódu 2.3 Příklad vytvoření tasku přes BashOperator

Airflow ve své základní instalaci obsahuje pouze několik základních operatorů. Obrovské množství operatorů lze do Airflow přidat instalací takzvaných **provider-packages**, což jsou rozšiřující balíčky. Airflow také nabízí možnost implementace vlastních operatorů. [13]

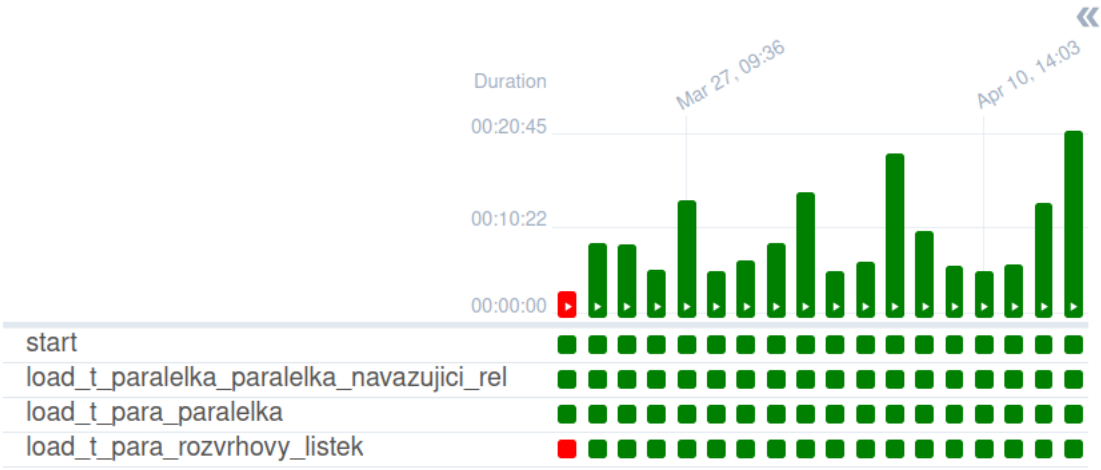
Následuje výčet operatorů, které jsou pro tuto práci relevantní:

- **EmptyOperator** nedělá doslova nic. Může být použit například k umělému sloučení kontrolního toku. Je vždy vyhodnocen schedulerem do stavu **success**, ale není posílán na executor.
- **BashOperator** vykoná bash příkaz nebo skript, který mu je předán povinném parametru `bash_command`.
- **BranchPythonOperator** slouží k větvení. V parametru `python_callable` přijímá název Python funkce, kterou spustí a od které očekává seznam `task_id` downstream tasků, které mají být spuštěny. Všechny ostatní downstream tasky jsou ignorovány a označeny stavem `skipped`, přičemž se tento postup tranzitivně aplikuje i na jejich downstream tasky.
- **PostgresOperator** slouží k vykonání daného SQL kódu v dané PostgreSQL databázi. V parametru `postgres_conn_id` dostává identifikátor předdefinovaného databázového připojení do cílové databáze, v parametru `sql` dostane vlastní SQL kód k vykonání. PostgresOperator je příkladem operatoru, který již není součástí základní instalace Airflow.
- **TriggerDagRunOperator** Spouští jiný DAG. Identifikátor spuštěného DAGu dostává v parametru `trigger_dag_id`. Implicitně cílový DAG pouze spustí a hned potom je mu přiřazen stav `success`. Toto chování lze změnit nastavením parametru `wait_for_completion` na hodnotu `True`. V takovém případě je spouštějící task po celou dobu běhu spuštěného DAGu ve stavu `running` a periodicky kontroluje, jestli se spuštěný DAG nedostal do některého z koncových stavů. [13]

2.3 Uživatelské rozhraní

Důležitou součástí Airflow je jeho webové uživatelské rozhraní, které umožňuje snadno monitorovat a ladit použité DAGy, tasky a další konstrukty. Ukázka detailu DAGu (tzv. grid view) je na obrázku 2.3, další screenshoty jsou v příloze A. Pomocí uživatelského rozhraní například **lze:**

- Prohlížet všechny existující DAGy, stavy jejich běhů a jednotlivých tasků
- Zobrazit detaily konkrétního DAGu, procházet logy z jeho běhů a přepínat mezi různými vizualizacemi jeho statistik
- Manuálně spouštět DAGy či opravné běhy tasků, případně je zastavovat
- Vytvářet a spravovat databázová a jiná připojení, proměnné prostředí, task pools a další pomocné mechanismy.



Obrázek 2.3 Detail DAGu (grid view)

Pomocí uživatelského rozhraní **nelze** vytvářet a měnit DAGy ani rozvrh jejich běhů.

Každému DAGu lze volitelným parametrem `tags` přiřadit libovolné množství tagů, pomocí kterých je pak možné DAGy v uživatelském rozhraní filtrovat. [13]

2.4 Další konstrukty

2.4.1 Task pools

Airflow má z globálního pohledu nějaký počet worker slotů, do kterých jsou přiřazeny běžící tasky. Tento počet závisí na použitém executoru a konfiguraci, ale je vždy konečný a tedy omezený. Tyto sloty je možné rozdělit mezi takzvané **task pools**, ke kterým jsou pak přiřazovány jednotlivé tasky. Dokud je v daném poolu místo, pak jsou tasky, které jsou do něj přiřazené, normálně spouštěny. Jakmile má však být spuštěn task, jehož pool je plný, pak je pouze zařazen do fronty a čeká na uvolnění některého ze slotů. Pro každý task lze také nastavit počet slotů, které při svém běhu zabírají. [13]

2.4.2 Priorita tasků

Každému tasku lze volitelným číselným parametrem `priority_weight` nadefinovat jeho prioritu, podle které může v globální frontě tasků a ve svém task poolu být upřednostněn nebo upozaděn. Implicitní priorita je 1 a může být nastavena na libovolné číslo. Priorita tasků se vyhodnocuje podle jedné ze tří metod:

- **Downstream.** Tato metoda je používána implicitně a reálnou hodnotu priority vytváří součtem nastavené priority tasku a všech jeho downstream tasků. Tasky

s vysokým počtem downstream tasků jsou tedy ve frontě „agresivnější“.

- **Upstream.** Tato metoda je přesným opakem předchozí metody – reálná priorita tasku je součtem jeho nastavené priority a priorit všech jeho upstream tasků. Upstream metoda je užitečná pro případy, kde je paralelně prováděno více běhů jednoho DAGu a je preferováno dokončení dřívějších běhů před zpracováním novějších.
- **Absolute.** Tato metoda prioritu tasků nijak nepočítá a přímo používá nastavené hodnoty priorit. Je vhodná, pokud uživatel přesně ví, jakou prioritu by daný task měl mít. Tato metoda šetří výpočetní prostředky, protože s jejím použitím odpadá režie za výpočet priorit – čím větší je DAG, tím znatelnější je výsledná úspora. [13]

2.5 Plánování úloh – Scheduler

Plánování a rozvrhování DAGů a tasků má na starosti **scheduler**. Standardně se rozvrh každého DAGu definuje pomocí cron syntaxe[5]. Pokud jsou pro daný případ užití možnosti cron syntaxe příliš limitující, lze DAGy rozvrhovat i pomocí komplikovanějších mechanismů, jako jsou vlastní rozvrhy (timetables) nebo datasety (datasets) [13]. Pro účely této práce bohatě postačí cron syntaxe.

2.6 Spouštění úloh – Executor

Za provádění tasků je zodpovědný **executor**. Existuje více executorů s různými vlastnostmi. Dělí se do dvou kategorií:

- **Lokální.** Tasky provádějí uvnitř procesu, kde běží samotný scheduler. Jsou jednoduché na konfiguraci, ale mají omezenou možnost paralelizace a škálování. Jejich příkladem je **SequentialExecutor** (který je ve výchozím nastavení po instalaci Airflow) a **LocalExecutor**.
- **Vzdálené.** Instance tasků jsou k provedení posílány na workery, které se nachází mimo scheduler. Obvykle jsou tvořeny několika samostatnými moduly, které je třeba nainstalovat, zprovoznit a nakonfigurovat. Příkladem je **CeleryExecutor**, který je použit během tvorby této práce. [13]

CeleryExecutor byl na serverech DW ČVUT zprovozněn již Kristinou Zolochevskou v [14], kde je toto téma popsáno detailněji.

[5] Intuitivní návrh cron výrazů umožňuje například stránka https://crontab.guru/

2.7 Průzkum nových verzí

Tato krátká podkapitola představuje několik důvodů k provedení upgrade Airflow předtím, než řešení vytvořené v této práci půjde do produkčního prostředí. Je dobré připomenout, že při tvorbě této práce je používána verze 2.6.3, která vznikla zhruba třičtvrtě roku před touto prací. Nejnovější verzí Airflow v době psaní této práce je verze 2.9.0.

Jedním z důvodů je zpětná kompatibilita. Na serveru DW ČVUT je v současnosti používán Python verze 3.9.2. Airflow ve verzi 2.7.0 zrušilo podporu Pythonu 3.7 a lze očekávat, že postupně bude rušit i podporu vyšších verzí. Vzhledem k množství užitečných funkcionalit, které v Airflow průběžně přibývají, doporučuje autor práce alespoň každoroční upgrade Pythonu a Airflow, aby jejich zastaráváním nedocházelo k rozšiřování technického dluhu (který se na DW ČVUT vyskytuje zejména kvůli tomu, že je stále používána verze PDI z roku 2015).

Hlavní důvody jsou ale mezi novými funkcionalitami uživatelského rozhraní. Bylo přidáno filtrování DAGů podle stavu **running**, což bude v případě DW ČVUT rozsáhle využito. Dále byla přidána nová stránka s monitoringem celého Airflow clusteru, což umožní rychlé zorientování uživatele například po pádu některé části nahrávání. Nakonec bylo přidáno několik možností vizualizace přímo nad historií běhů jednoho konkrétního tasku, což umožní snadnější analýzu výkonových problémů, které nejsou na straně Airflow. [13]

Kapitola 3
Návrh řešení

Tato kapitola nejprve vysvětluje značnou odchylku výsledné implementace od základů, které pro ni položila Kristina Zolochevskaia v [14]. Následně diskutuje a nasvětluje skutečnosti a okolnosti, které bylo potřeba při tvorbě řešení brát v úvahu. Úvaha je v rámci možností odproštěna od Apache Airflow, popis výsledného řešení v této technologii se nachází v kapitole 4.

3.1 Odchylka od PoC

Jak již bylo zmíněno, základy pro tuto práci položila Kristina Zolochevskaia se svou bakalářskou prací [14], kde vytvořila funkční PoC[1] k paralelní variantě nahrávání. Ve své práci používala poměrně pokročilé a pracné techniky – ETL procesy ručně implementovala ve formě Python pandas skriptů, pro jejichž spouštění si naprogramovala vlastní operatory.

Výsledná implementace se od zmíněného PoC značně odchýlí, protože z personálně-kapacitních důvodů padlo na oddělení VIC ČVUT[2] manažerské rozhodnutí Airflow sice používat, ale pouze pro spouštění dílčích PDI transformací/jobů a PL/pgSQL funkcí, které jsou již hotové a nasazené.

Tedy místo toho, aby bylo PDI opuštěno, nad ním bude vystavěná další vrstva, která bude jeho agendu efektivněji orchestrovat a monitorovat.

Kristina Zolochevskaia a Adam Marhefka, kteří PoC vytvářeli současně, společně zformulovali funkční (F) a nefunkční (N) požadavky, které by jejich PoC měly splňovat. Oba pak svá řešení s těmito požadavky konfrontovali.

Vzhledem k odchýlení řešení od PoC a k proběhnuvší analýze požadavku na nahrání jedné samostatné tabulky či výčtu tabulek je potřeba dva z požadavků uvést do kontextu:

- **F2. Nahrání jedné vybrané IDL tabulky s historizací:** Řešení umožňuje

[1] Proof of Concept
[2] Výpočetní a Informační Centrum ČVUT

nahrání jedné tabulky nebo seznamu tabulek, s možností samostatného nahrávání pre-stage/pre-stage-clean tabulek a historizačních procesů jen pro nutné stage-increment tabulky za účelem nahrání správných dat do vybraných IDL tabulek. [14]

Reálné řešení bude taktéž podporovat nahrání jedné celé tabulky, případně i spouštění jednotlivých fází nahrávání samostatně. Je to nutné pro případ manuálního dokončení neúspěšného nahrávání. Nabízí se také využití této funkcionality k náhodnému nahrání jedné tabulky mimo nahrávání celého skladu, ale zde je situace složitější. Detailnější rozbor této situace se nachází v podkapitole 3.5.

- **F3. Nahrávání z různých zdrojových systémů:** Řešení podporuje nahrávání z různých typů zdrojových systémů. Nyní podporuje databázové systémy PostgreSQL a Oracle a je navržené tak, aby bylo v budoucnosti v případě potřeby možné snadno přizpůsobit pro další databázové systémy. [14]

  Vzhledem k tomu, že reálné řešení bude spouštět PDI transformace, s čímž se v PoC nepočítalo, bude tento aspekt rozšiřitelnosti delegován zpět do PDI transformací. Airflow se napřímo bude připojovat pouze do DW ČVUT, kde běží databázový systém PostgreSQL.

3.2 Hlavní výzvy paralelní implementace

3.2.1 Závislosti mezi úlohami

Současný produkční ETL job (viz podkapitola 1.4.2) je koncipován tím způsobem, že vždy dělá jeden krok pro všechny tabulky, a až potom přistoupí k dalšímu kroku, což znamená, že je předem pevně dané pořadí operací, které je striktně dodržováno. Když se například nahrávají data do schématu UDE, kde příslušné tabulky závisejí na několika tabulkách v pre-stage, tak je z logiky věci garantováno, že všechny pre-stage tabulky jsou již nahrané. Nahrávání do IDL má obdobným způsobem garantováno, že před jeho spuštěním byly spočítány změnové příznaky ve stage-increment.

S příchodem paralelismu bohužel tento invariant přestává platit, protože snaha o pevnou definici pořadí všech úloh by měla za výsledek řešení, které by nutně muselo být opět sekvenční. Úkolem implementace je vytvořit graf závislostí mezi úlohami tak, aby byly dodrženy všechny závislosti, které jsou potřeba ke konzistenci dat, ale zároveň tak, aby pořadí úloh bylo kontrolováno co možná nejméně, aby jednotlivé úlohy mohly v klidu běžet a aby nevzniklo žádné „úzké hrdlo“ kvůli čekání na některou ze závislostí.

3.2.1.1 Mapování tabulek

IDL tabulky vždycky vznikají z některých zdrojových tabulek. Toto mapování má 4 varianty – 1:1, N:1, 1:N, M:N. Každá z těchto variant má jiný scénář nahrávání a jiné důsledky pro konstrukci grafu závislostí.

1:1 Nejjednodušší scénář. Ze zdroje se nahraje tabulka do pre-stage, spočítají se nad ní změny, které jsou následně nahrány do svého protějšku v IDL.

N:1 Do pre-stage se nahraje více tabulek. Samostatně se u nich spočítají změny, za použití některé varianty operace JOIN se spojí a množinové sjednocení změněných řádků se nahraje do jedné IDL tabulky. **Na příslušných zdrojových tabulkách nijak nezávisí žádná další IDL tabulka.**

1:N Do pre-stage se nahraje jedna tabulka, pro kterou se spočítají změny. Místo do jedné IDL tabulky se změny nahrají do více menších a specializovanějších tabulek, do kterých je zdrojová tabulka v IDL dekomponována. Příkladem je zdrojová tabulka `tkontakt`, ze které v IDL čerpají tabulky `t_koud_email`, `t_koud_telefonni_cislo` a `t_koud_adresa`. **Všechny dotyčné IDL tabulky v této situaci závisí právě na jedné zdrojové tabulce.**

M:N Tato situace je podobná situaci N:1 nebo 1:N. U situace N:1 ale minimálně na jedné zdrojové tabulce závisí jiná IDL tabulka, a u situace 1:N má minimálně jedna z IDL tabulek více než jednu závislost.

Nejjednodušší je varianta 1:1. Ta se navíc vyskytuje u většiny tabulek, což bude zužitkováno v podkapitole 3.4 při návrhu členění DAGů. Při pravidelném automatickém nahrávání, které je cílem této práce, pro jednotlivé tabulky stačí před nahráním do IDL nahrát všechny jejich závislosti, ať už mají jakékoliv mapování. V případě implementace případu užití s manuálním nahráním jedné tabulky se však (zejména pro mapování M:N) situace komplikuje, viz podkapitola 3.5.

3.2.2 Parametry paralelismu

V době psaní této práce obsahuje schéma `dwh` v IDL zhruba 200 tabulek, z nichž pravidelnému nahrávání podléhá více než 150. Bylo by nemyslitelné nahrávat takový počet tabulek skutečně najednou, a bude tedy potřeba najít vhodné parametry paralelismu tak, aby:

- Nevznikl problém s překročením povoleného počtu připojení k některé z databází (platí pro IDL, stage i zdrojové databáze).
- Nedošlo k přetížení databázového serveru či serveru, kde běží Airflow a tedy i PDI.
- Nenastala situace, ve které se paralelizace už ani nevyplatí, protože by většina výpočetních prostředků byla obětována na přepínání kontextu mezi běžícími vlákny.

3.3 Hlavní důvody zrychlení

Paralelní nahrávání v Airflow oproti původní variantě přináší trochu režie, například v podobě parsování DAGů a tasků, či v podobě toho, že se při spouštění každé transformace spouští celé PDI ve formě skriptu pan.sh. Výkonového zrychlení by však mělo být dosaženo ze tří hlavních důvodů:

- **Rozložení zátěže.** Když je tabulka kopírována ze zdrojového systému do pre-stage, nedějí se na databázovém clusteru DW ČVUT žádné výpočetní operace. Čeká se pouze na přenos dat, jehož rychlost může být omezena zdrojovým systémem či přenosovou kapacitou sítě. Když se zase například počítají změnové příznaky, tak je zaměstnán pouze databázový cluster a nic jiného. Doba sekvenčního nahrávání je součtem doby všech svých dílčích kroků, ale pokud paralelní nahrávání vhodně využije všechny zmíněné výpočetní prostředky najednou, může být doba nahrávání značně snížena.
- **Zmenšení transakcí.** Současná varianta zpracování má mimo jiné tu nevýhodu, že čištění dat v pre-stage a počítání změnových příznaků ve stage-increment probíhají ve velkých a monolitických databázových transakcích. To přináší zvýšené nároky na operační paměť a znatelné zpomalení. V paralelním zpracování se díky parametrizaci funkcí pro výpočet příznaků budou flagy počítat pro každou tabulku v samostatné transakci, což nároky na operační paměť sníží.
- **Nižší počet změn za jeden běh nahrávání.** Tento vedlejší efekt plyne ze zvýšení frekvence nahrávání – v současnosti se jednorázově nahrávají změny za celý uplynulý týden, po nasazení paralelního řešení to budou změny za jeden den. Nahrávání změn do IDL díky tomu proběhne v průměru rychleji.

3.4 Členění DAGů

Další výzvou je modularizace. Vytvořit jeden velký DAG pro celé nahrávání je možnost, která se ke zvážení nabízí jako první. Airflow nám nabízí mechanismus TaskGroups, který v uživatelském rozhraní dokáže schovat libovolný počet tasků (které spolu obvykle nějak souvisí) do jednoho „rozbalovacího“ tasku, ale autor práce v průběhu analýzy došel k závěru, že i s využitím tohoto mechanismu by nebyl dobrý nápad se vydat cestou jednoho monolitického DAGu. Řešení by totiž přišlo o možnost snadno spouštět části nahrávání manuálně – ta je kriticky důležitá při manuální nápravě neúspěšného nahrávání, ale také umožňuje při vývoji průběžně testovat malé části nahrávání.

Autor práce se v souladu s nepsanou konvencí rozhodl, že je vhodné brát nahrání jedné IDL tabulky jako odlišitelnou a spustitelnou operaci – a to promítnout do výsledné struktury celého nahrávání. **Nechť tedy nahrávání každé tabulky**

do IDL má svůj vlastní DAG. Ten totiž lze ve webovém rozhraní snadno dohledat a manuálně spustit. Uživatelské rozhraní Airflow navíc vykresluje téměř všechny statistické grafy pro právě jeden celý DAG, takže při dodržování konvence „1 soubor = 1 DAG“ lze snadno hledat a opravovat chyby. Většina tabulek má navíc mapování 1:1 (viz podkapitola 3.2.1.1). Tabulka s tímto mapováním pro své úspěšné nahrání potřebuje pouze obyčejnou sekvenci operací: nahrání do pre-stage, spočítání změn a nahrání změn do IDL. Autor usoudil, že dobrým řešením bude pro každou tabulku vytvořit DAG, který bude pokud možno obsahovat celý proces jejího nahrávání.

3.4.1 Hierarchie DAGů

Mezi DAGy jsou závislosti. TriggerDagRunOperator (viz podkapitola 2.2.1.3) nám umožňuje uvnitř DAGu spouštět jiný DAG a nabízí se jako dobrý způsob, jak dodržování těchto závislostí vynutit. Je tedy potřeba vytvořit hlavní DAG, jehož úkolem bude pouze spouštění všech ostatních DAGů tak, aby byly jejich závislosti dodrženy. Jeden DAG, který by přímo spouštěl všechny ostatní, by však byl příliš velký, což by ho připravilo o veškerou myslitelnou přehlednost.

Je tedy vhodné zavést nějaký mezistupeň. V současném nahrávání (viz podkapitola 1.4.2) plní roli tohoto mezistupně dílčí PDI joby, které jsou spouštěny hlavním jobem a spouští buďto další joby, či přímo transformace. Nahrávání do pre-stage je do dílčích jobů v současném řešení rozděleno podle zdrojových systémů, nahrávání do IDL je rozděleno podle logických celků z pohledu IDL. Autor se ve svém řešení rozhodl reflektovat pouze dělení z pohledu IDL a filtrování DAGů dle zdrojových systémů umožní použitím tagů.

Výsledným návrhem je následující tříúrovňová hierarchie:

- **Listový DAG** Je spouštěn zpravidla vnitřním DAGem, v ojedinělých případech přímo kořenovým DAGem. Obvykle je logickým ekvivalentem jedné IDL tabulky.
- **Vnitřní DAG** Je spouštěn kořenovým DAGem. Je ekvivalentem některého logického celku, například Osoby (OSOB), státní závěrečné zkoušky (SZZK) nebo Úhrady (UHRA). Spouští všechny listové DAGy, jejichž tabulky k danému logickému celku přísluší, případně přímo nahrává společné prerekvizity těchto listových DAGů.
- **Kořenový DAG** V době vývoje je spouštěn manuálně pomocí tlačítka v uživatelském rozhraní, **po nasazení do produkčního prostředí bude spouštěn automaticky pomocí Airflow scheduleru**, ideálně jednou denně. Reprezentuje celý proces nahrávání. Napřímo spouští všechny vnitřní DAGy a některé listové DAGy.

3.4.2 Scénáře nahrávání pro jednotlivá mapování

Scénáře pro jednotlivé typy mapování tedy budou vypadat takto:

- **1:1** Tabulka bude mít svůj vlastní DAG, který ji nahraje z pre-stage až do IDL.
- **N:1** V rámci DAGu se ze zdroje se do pre-stage nahrají všechny potřebné tabulky a spočítají se pro ně změny. DAG končí nahráním změn do cílové IDL tabulky.
- **1:N** Zde se situace mírně komplikuje. Navrhované řešení je takové, že se zdrojová tabulka nahraje do pre-stage a spočítají se pro ni změny už ve vnitřním DAGu (viz podkapitola 4.3), načež se paralelně spustí listový DAG pro každou související cílovou tabulku, který pouze nahrává změny do IDL.
- **M:N** Zde je situace nejsložitější. Pro každou individuální situaci s tímto mapováním je potřeba pro příslušné tabulky před nahráváním do IDL prověřit, že jsou uspokojeny všechny její závislosti, které mohou pocházet i z jiného vnitřního DAGu. Vzhledem k tomu, že četnost tohoto mapování je v DW ČVUT přijatelně nízká, bude tato situace řešena individuálně s možným porušením zmíněných konvencí, například vyextrahováním nahrávání zdrojových tabulek s velkým počtem závislostí mimo vnitřní DAGy.

3.5 Problémy se samostatným nahráváním jednotlivých tabulek

V podkapitole 3.1 je zmíněn funkční požadavek na možnost manuálně nahrát jednu či více tabulek. V souvislosti s ním je potřeba prodiskutovat následující tři situace.

3.5.1 Manuální nahrání tabulky po pádu automatického nahrávání

Zde je situace v pořádku, protože je k dispozici kořenový DAG, který udržuje správné pořadí nahrávání a závislosti pro všechny tabulky. Pokud nahrávání některé tabulky selže, tak některý z pracovníků problém lokalizuje, vyřeší a tabulku manuálně donahraje s tím, že všechny případné závislosti má nahrávaná tabulka již splněné – kdyby neměla, tak se nemohla začít nahrávat. Pokud na tabulce, kde se vyskytla chyba, závisí jiná tabulka či celý logický celek, lze je opět snadno nahrát manuálně, aniž by došlo k nekonzistencím v datech.

3.5.2 Mimořádné manuální nahrání tabulky mimo automatické nahrávání

Tato situace je výrazně složitější, než se zdá. Byla provedena její hluboká analýza. U tabulek s mapováním 1:1 a N:1 toto beztrestně provést lze. U tabulek s mapováním 1:N to lze provést také, ale všechny další IDL tabulky, které na dotyčné zdrojové tabulce závisí, musí být bezpodmínečně nahrány také – jinak by mohlo v IDL dojít k nekonzistenci v datech, která může mít hrozivé důsledky, viz dále.

U tabulek s mapováním M:N je situace zdaleka nejsložitější, protože tyto IDL tabulky obvykle závisí na tabulkách, na kterých závisí další IDL tabulky z úplně jiného logického celku.

Zde je zjednodušený odstrašující scénář pro ilustraci problému: IDL tabulka `t_anke_nova_student_hodnoceni` závisí na stage tabulce `toboryst` ze zdrojového systému KOS a na stage tabulce `student_evaluation` ze zdrojového systému Anketa. Na zdrojové tabulce `toboryst` navíc dále závisí IDL tabulka `t_stpl_studijni_obor`. Scénář vypadá následovně:

- **1.** Uživatel se rozhodne nahrát IDL tabulku `t_anke_nova_student_hodnoceni` manuálně. Do stage tedy nahraje tabulky `student_evaluation` a `toboryst`. Do obou tabulek na zdroji přibyl nový záznam – úspěšně se dostane až do stage-increment s nahozeným *NEW* příznakem.
- **2.** Změny se nahrají do tabulky `t_anke_nova_student_hodnoceni` – všechno zdánlivě funguje v pořádku. Je ale třeba mít na paměti, že změny se do tabulky `t_stpl_studijni_obor` nenahrály.
- **3.** Druhý den je spuštěno automatické nahrávání celého skladu. Před spočítáním změn pro tabulku `toboryst` se ve stage-increment všechny příznaky vymažou, což znamená, že nový záznam z kroku **1** ve stage-increment zůstane, ale do tabulky `t_stpl_studijni_obor` se nenahraje.
- **4.** V blízké či daleké budoucnosti se zmíněný záznam ve zdrojové tabulce `toboryst` změní. Při následujícím nahrávání skladu změna projde do stage-increment, avšak tentokrát s *MODIFIED* příznakem. Při nahrávání změn do `t_stpl_studijni_obor` pak nahrávání spadne s hláškou, že se ETL proces snaží změnit řádek, který neexistuje.

Tato chyba se může dotknout náhodného počtu řádků a projevit se v náhodný čas, nehledě na to, že tabulka `t_stpl_studijni_obor` ve skutečnosti závisí ještě na zdrojové tabulce `ttexty`, na níž závisí dalších 5 různých IDL tabulek. Bohužel se také jedná o chybu, jejíž oprava je extrémně náročná, protože data ve skladu musí být manuálně a zpětně uvedena do konzistentního stavu. Nepomůže zde ani obnova zálohy, protože se tato chyba typicky projevuje s větším časovým odstupem. Podobných scénářů končících podobnou chybou navíc může nastat více.

Možným řešením je vytvořit seznam všech závislostí mezi zdrojovými tabulkami a IDL tabulkami, uložit ho například do databáze a dívat se na něj jako na nesouvislý **neorientovaný** graf (nejedná se o graf nahrávání, ale pouze o seznam závislostí mezi tabulkami). Potom stačí v tomto grafu najít maximální souvislé komponenty a uložit je. Pokud chce uživatel manuálně nahrát některou z IDL tabulek, musí ve správném pořadí nahrát všechny tabulky v komponentě, ve které se daná IDL tabulka nachází. Je dobré připomenout, že v případě mapování 1:1 se jedná o triviální jednoprvkové komponenty.

Ukázkový scénář sice nepokrývá situace, kde manuálně nahrávaná tabulka závisí pouze na pre-stage variantě tabulky zdrojové, protože do ní dělá obyčejný databázový lookup, ale v tomto případě se vždy může stát, že do příštího nahrávání bude příslušný záznam smazán. V takovém případě by v jedné IDL tabulce figuroval jako cizí klíč, který neodkazuje nikam. Nekonzistence podobného charakteru mohou nastat vždycky, protože se tabulky nikdy do pre-stage nenahrávají přesně ve stejný okamžik, ale určitě je dobré pravděpodobnost těchto nekonzistencí minimalizovat.

3.5.3 Nahrávání tabulek s různou frekvencí

Současné nahrávání obsahuje mechanismus, kterým lze pro jednotlivé tabulky snížit frekvenci nahrávání, například z týdenní na měsíční (viz podkapitola 1.4.2.1). Tento mechanismus byl implementován pro případné budoucí použití a aktuálně používán není, při přechodu na každodenní nahrávání by však jeho použití stálo za zvážení.

Autor se rozhodl ho v testovací verzi paralelního nahrávání nezahrnout a v Airflow tedy místo check-jobů volá pouze nahrávající transformace. Problematika této situace lze totiž převést na předchozí situaci – kdyby chtěl uživatel snížit frekvenci nahrávání pro některou tabulku, musí ji stejnou měrou snížit pro všechny tabulky ze stejné souvislé komponenty.

Stávající řešení tyto změny frekvence nahrávání umožňuje pouze přes změnu odpovídajícího řádku přímo v databázi, a v případě použití není uživatel nijak upozorněn na možnou chybu, kterou mohl do nahrávacího procesu zanést. Podobné řešení lze navíc implementovat mnoha způsoby přímo v Airflow, včetně umožnění změn pouze pro celé komponenty.

3.6 Nežádoucí výkyvy doby běhu

Každá operace u nahrávání každé tabulky má nějaký obvyklý čas běhu, který se odvíjí od velikosti tabulky a obvyklého počtu změn, které ze zdrojových systémů přicházejí. Doba běhu samozřejmě do jisté míry kolísá, protože dostupné výpočetní prostředky jsou při každém nahrávání dané tabulky různě vytížené a počty změn jsou také různé.

Zásadní nárůst doby nahrávání může mimo naprosté výjimky nastat pouze při nahrávání změn do IDL, a to konkrétně ve dvou situacích:

- Pokud ve zdrojovém systému dojde ke změně DDL[3] nebo k plošné (byť kosmetické) úpravě dat, pak při nahrávání do pre-stage pravděpodobně dojde k masivním změnám v MD5 hashích. Tyto změny se pak dostanou do stage-increment a pokud je dotyčná tabulka dost velká, pak doba nahrávání změn do IDL drasticky vzroste, i kdyby v průběhu měla drtivá většina modifikovaných řádků být zahozena. Událost tohoto typu se na datovém skladu přihodila například v první polovině března roku 2024 a vedla k tomu, že se tabulka `t_vvvs_externi_citace_autor` do IDL nahrávala déle než tři dny.
- Pokud nastane plošná změna DDL nebo dat (obdobně jako u prvního případu), může dojít k zaseknutí nahrávání do IDL, ale z jiného důvodu. Většina PDI transformací, které nahrávají změny do IDL (viz obrázek 1.9) obsahuje kroky *WAIT FOR NEW* a *WAIT FOR DELETED*. Na úrovni celé transformace je pak nastavena velikost rowsetu, který lze chápat jako buffer čekajících řádků pro jednotlivé kroky. Pokud se buffery u *WAIT* kroků zaplní, začnou se plnit buffery předchozích kroků. Pokud se před dokončením nahrávání nových záznamů zcela zaplní rowset větvícího kroku *switch*, potom se nahrávání zasekne a nikdy se nedokončí.

Druhý scénář může nastat i v současném produkčním nahrávání a bude týmem DW ČVUT řešen, nicméně není předmětem této práce.

Předejít prvnímu scénáři je téměř nemožné, a je tedy potřeba brát v potaz, že může nastat. Kdyby při každodenní frekvenci nahrávání tento scénář nastal a prodloužil by dobu nahrávání na více než 24 hodin, nastal by konflikt mezi nedoběhlou a novou instancí nahrávání. Tento konflikt by měl za důsledek obrovské nekonzistence v datech, a je tedy nutné mu předcházet – více v podkapitole 4.6.2.

[3]Data Definition Language

Kapitola 4
Popis implementace

V této kapitole se nachází popis implementovaného testovacího řešení, doprovázený screenshoty pro čtenářovu lepší představu. Tato kapitola je spíše popisná, rozsáhlejší zhodnocení najde čtenář v kapitole 5.

4.1 Nalezení závislostí

Nalezení všech závislostí je první úkol, který stojí za to promyslet. DW ČVUT má svou DLM,[1] kde je na úrovni sloupců evidováno mapování ze zdrojového systému do IDL. Ta je však bohužel neaktuální a obsahuje i tabulky, které pro své nahrání potřebují nějakou formu manuálního zásahu, tudíž součástí automatického nahrávání nejsou. Další možností je manuálně procházet produkční PDI job a závislosti si poznamenávat. Tato možnost sice zabere více práce, ale je žádoucí, protože důkladné prostudování produkčního nahrávání nalezne skutečně všechny závislosti. To je kriticky důležité, protože nepodchycení byť jediné závislosti by mohlo vést k nekonzistentnímu stavu dat v datovém skladu, což je problém, který se opravuje velmi špatně.

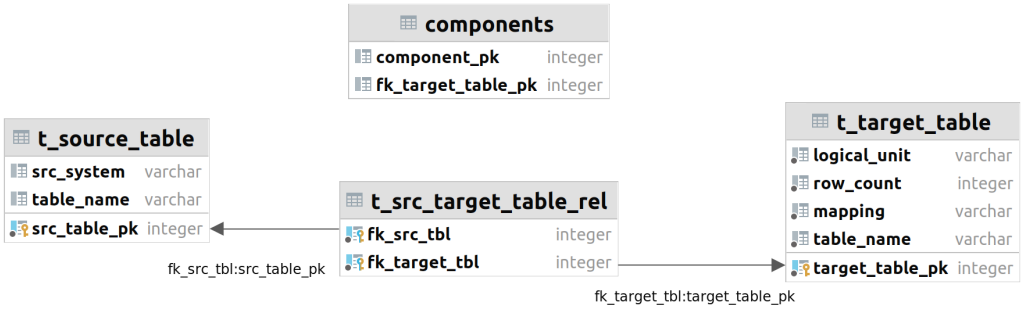
Autor práce si prostudoval všechny PDI transformace, které nahrávají data ze zdrojových systémů do pre-stage a ze stage-increment do IDL. Nalezené závislosti si poznamenal do tabulkového procesoru Google Sheets. Následně zjistil, že se do stage nahrává více tabulek, než se jich reálně podílí na datovém obsahu IDL. Jde zejména o tabulky, které jsou historickým reliktem – například tabulky ze starého systému Anketa či starého systému se závěrečnými pracemi, které dnes již nejsou v provozu a tím pádem se nemění. Na poradě týmu DW ČVUT bylo rozhodnuto, že tyto reliktní tabulky do paralelního nahrávání zahrnuty nebudou.

Dalším vhodným krokem je tyto závislosti přenést do databáze. Prvním důvodem pro tento krok je netriviální dotazování během konstrukce grafu nahrávání, které v tomto procesu ušetří hodně práce. Druhým důvodem je to, že při vhodné

[1] Datovou Logickou Mapu

reprezentaci závislostí je pak možné vytvořit PL/pgSQL proceduru, která nalezne souvislé komponenty pro případ samostatného nahrávání tabulek, který je popsán v podkapitole 3.5.

4.1.1 Databázový model pro závislosti



Obrázek 4.1 Databázové schéma pro reprezentaci závislostí mezi tabulkami

Na obrázku 4.1 je jednoduché databázové schéma obsahující čtyři tabulky sloužící k reprezentaci závislostí v nahrávacím procesu.

- **t_source_table** reprezentuje zdrojovou tabulku. Kromě primárního klíče a názvu reprezentované tabulky ještě obsahuje informaci o tom, ze kterého zdrojového systému tabulka pochází.
- **t_target_table** reprezentuje tabulku v IDL. Mimo primární klíč a název reprezentované tabulky obsahuje informaci o logickém celku, ke kterému přísluší (ten je vždy jen jeden), dále typ mapování a informaci o počtu řádků, která byla plošně pro všechny tabulky spočtena dne 29. 1. 2024 a slouží jako upřesňující informace pro plánování priorit nezávislých tabulek při nahrávání.
- **t_src_target_table_rel** je dekompozicí vazby typu M:N mezi předchozími dvěma tabulkami. Právě tato tabulka plní roli onoho seznamu závislostí.
- **components** pak reprezentuje jednotlivé souvislé komponenty. Komponenta má pouze svůj identifikátor a může obsahovat jednu nebo více IDL tabulek. Tabulka souvislých komponent je generována uloženou PL/pgSQL procedurou `find_continuous_components()`, kterou autor vytvořil za použití modifikace algoritmu BFS[2].

Vybaven takto reprezentovaným seznamem závislostí začal autor implementovat jednotlivé DAGy.

[2]Breadth-First-Search, neboli prohledávání grafu do šířky

4.2 Listové DAGy

Typický listový DAG je vyobrazen na obrázku 4.2. Téměř vždy nahrává jednu IDL tabulku. Mimo případ mapování 1:N a některé případy mapování M:N obsahuje i všechny předchozí kroky počínaje nahráním ze zdrojového systému.

Agenda jednotlivých tasků z obrázku je následující:

- **start (EmptyOperator)** Použití prázdného operátoru bylo zavedeno jako konvence pro všechny DAGy. U listových DAGů zatím žádné využití nemá, ale u ostatních DAGů, které jsou rozvětvené, slouží jakožto kořenový prvek pro přehledné vykreslování DAGů v uživatelském rozhraní.
- **load_to_ps (BashOperator)** Spouští existující PDI transformaci, která nahrává příslušnou tabulku do pre-stage.
- **clean_data (PostgresOperator)** Volá PL/pgSQL proceduru, která z pre-stage tabulky vytvoří její pre-stage-clean variantu a tím provede technické očištění dat. Tento task se provádí jenom u některých tabulek.
- **clean_stage(PostgresOperator)** Volá parametrizovanou obdobu PL/pgSQL procedury `inc_clear_state_flag`, která ve stage-increment vynuluje stavové příznaky pro zadanou tabulku ze zadaného schématu.
- **get_M_N_D_flags (PostgresOperator)** Spouští následující kód:

```
BEGIN;
SELECT inc_find_modified_in_pre_stage_schema_table_args('s','t');
SELECT inc_find_new_in_pre_stage_schema_table_args('s','t');
SELECT inc_find_deleted_in_pre_stage_schema_table_args('s','t');
COMMIT;
```

Výpis kódu 4.1 SQL kód spouštěný taskem `get_M_N_D_flags`, místo `'s'` je předáván název schématu, místo `'t'` název tabulky.

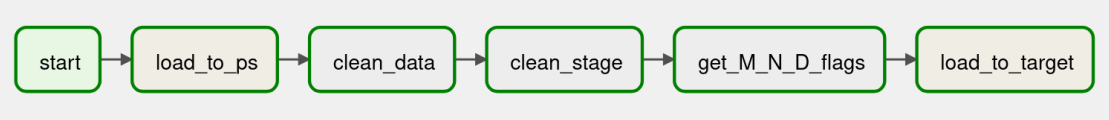
Tyto tři volané procedury jsou opět parametrizované obdoby původních procedur, jež počítají změny, které pak v podobě příznaků uloží do stage-increment. Nejprve modifikované (M), potom nové (N), a nakonec smazané (D) záznamy. stejně jako u procedury `inc_clear_state_flag` zde platí, že původně byly prováděny plošně a v jedné transakci, ale díky parametrizaci je lze provádět pro jednotlivé tabulky samostatně.

- **load_to_target(BashOperator)** Spouští (již existující) PDI transformaci, která nahrává data ze stage-increment do IDL a historizuje je.

Jak již bylo řečeno, v případě mapování 1:N a některých případů M:N obsahuje listový DAG pouze task `load_to_target`, protože předchozí operace jsou vykonané dříve – v některém z vnitřních nebo předcházejících listových DAGů. V případě mapování N:1 jsou v DAGu příslušné operace (mimo `start` a `load_to_target`) zduplikovány N-krát.

Řešení pak obsahuje několik listových DAGů, které nejsou přiřazeny k žádné IDL tabulce a zaštiťují pouze nahrávání zdrojové tabulky ze zdroje do stage-increment včetně spočítání změn. Tyto DAGy se týkají buďto tabulek ze schématu UDE, nebo tabulek, které jsou společnou závislostí pro hodně tabulek pocházejících z různých logických celků, a tudíž se je autor rozhodl vyextrahovat mimo vnitřní DAGy. V kořenovém DAGu na nich závisí vnitřní DAGy.

Nakonec je zde jeden listový DAG, který spouští celý PDI job nahrávající všechny tabulky s příponou `_nohist`. Jedná se o tabulky, které nejsou historizované, tudíž je stačí nahrát ze zdroje přímo do IDL bez potřeby jakýchkoliv závislostí a vzhledem k jejich malé velikosti si autor v testovací implementaci vystačí s tímto řešením.



Obrázek 4.2 Listový DAG

4.3 Vnitřní DAGy

Vnitřní DAGy jsou v rámci členění mezistupněm mezi kořenovým DAGem a listovými DAGy. Vždy přísluší k některému logickému celku, podle kterých je členěna IDL. Stejná logika členění je použita v současně používaném PDI jobu.

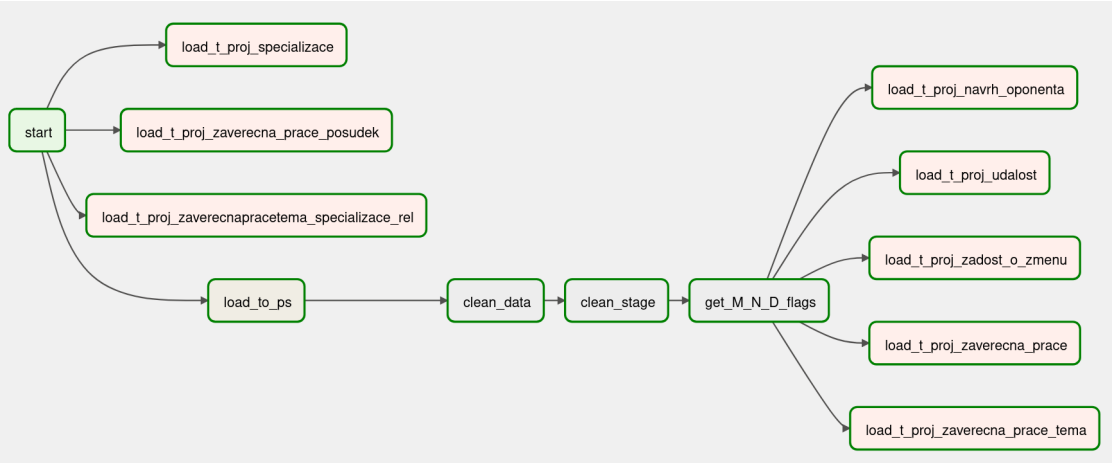
Mají dvě úlohy:

- Spouští všechny listové DAGy pro tabulky z příslušného logického celku
- Pokud daný logický celek obsahuje tabulky s mapováním 1:N, pak před spuštěním příslušných listových DAGů provede pro jejich zdrojovou tabulku operace od `load_to_ps` do `get_M_N_D_flags`, jako kdyby se jednalo o listový DAG.

Ke spouštění listových DAGů je použit TriggerDagRunOperator s parametrem `wait_for_completion` nastaveným na hodnotu `True`. Pro tyto tasky byla zvolena jmenná konvence `load_[název tabulky]`.

Vnitřní DAGy svou velikostí reflektují logické celky a díky tomu se ve velikosti velmi liší – nejmenším vnitřním DAGem je `load_arok` obsahující pouze jeden spouštěný listový DAG, naopak největší vnitřní DAG `load_vvvs` listových DAGů spouští 29.

Na obrázku 4.3 se nachází vnitřní DAG pro logický celek *PROJ*, kde jsou vidět oba jeho účely – spouští 3 listové DAGy, paralelně s nimi nahrává ze zdroje až do stage-increment tabulku `person`, po jejímž nahrání spouští dalších 5 listových DAGů, pro které je tabulka `person` upstream závislostí.



Obrázek 4.3 Vnitřní DAG pro logický celek *PROJ*

4.4 Kořenový DAG

Kořenový DAG je pomyslným srdcem celého řešení. Orchestruje spouštění všech vnitřních DAGů a několika listových DAGů, na kterých jsou různé vnitřní DAGy závislé. Jeho tvorba vyžadovala pečlivé rozmyšlení. Podoba výsledného kořenového DAGu je vidět na obrázku 4.4.

Je dobré zmínit, že kořenový DAG je v rámci Airflow rozvrhován jako jediný. Všechny ostatní DAGy jsou totiž spouštěny externě přes `TriggerDagRunOperator`.

4.5 Chybové scénáře

Nahrávání někdy skončí chybou různého druhu. Chyba se může vyskytnout na libovolném místě nahrávání a může být způsobena buď nedokonalostí nasazeného řešení, nebo například špatnými daty, která mohou ze zdrojových systémů přijít kdykoliv. Je velmi důležité tyto chybové scénáře předem promyslet, aby v případě jejich výskytu bylo možné chyby napravit. Vzhledem k rozmanitosti možných chyb si zde autor vystačí rozborem možných míst jejich výskytu:

- Pokud u některé z tabulek nastane chyba dříve, než dojde k dokončení tasku `get_M_N_D_flags`, pak po její opravě může uživatel beztrestně spustit celý listový DAG odznova, protože se žádné změny ještě nedostaly do stage-increment.

Nahrávání dat do pre-stage a pre-stage-clean totiž vždycky přemaže všechna původní data.

- Pokud je task `get_M_N_D_flags` již jednou dokončen a chyba nastane v nahrávání do IDL, pak je nutné chybu opravit a nahrávání do IDL manuálně spustit znovu. Před úspěšným nahráním do IDL ale **v žádném případě nesmí být spuštěn task** `clean_stage` (ideálně ani žádný předchozí). Změny by ve stage-increment zůstaly, ale byly by ztraceny jejich příznaky, kvůli čemuž by se změny do IDL nedostaly. V blízké budoucnosti by pak nastala obdobná chyba jako v odstrašujícím scénáři z podkapitoly 3.5.2. Před manuálním znovuspuštěním nahrávání do IDL je také potřeba zkontrolovat, že se do IDL nedostaly žádné nové záznamy, protože Postgres Bulk Loader občas vykazuje nepředvídatelné chování v transakčním zpracování.

V obou případech je také samozřejmě potřeba spustit všechny downstream závislosti dotčených DAGů, a to opět do doby dalšího automatického nahrávání.

4.6 Ostatní použité konstrukty

4.6.1 Použití task pools

Představený způsob implementace má své nesporné výhody, ale přináší i některé menší komplikace, které je potřeba vyřešit. Jedním z mnoha parametrů v konfiguračním souboru `airflow.cfg` je parametr `parallelism`, který určuje maximální povolený počet současně běžících tasků napříč celou instalací Airflow. Tento parametr by neměl být nastaven na příliš vysokou ani nízkou hodnotu a autor práce po nějaké době ladění přistoupil na hodnotu 32. Zde přichází problém, protože každý TriggerDagRunOperator s parametrem `wait_for_completion=True` je po celou dobu běhu spuštěného DAGu sám o sobě běžícím taskem, tudíž po celou tuto dobu zabírá jeden ze slotů.

Bez použití dalších omezujících opatření by takto nastala situace podobná deadlocku. Kořenový DAG by po svém spuštění pomocí TriggerDagRunOperatoru spustil vnitřní DAGy, které by to analogicky provedly pro DAGy listové – všech 32 slotů by bylo zabraných ještě předtím, než by se nahrála jakákoliv data, a tato data by se nikdy nahrávat nezačala, protože by nahrávající tasky čekaly na uvolnění slotů.

Elegantním řešením této situace je použití task pools (viz podkapitola 2.4.1), které jsou v Airflow vestavěné. Autor vytvořil pool pro:

- TriggerDagRunOperatory spouštějící vnitřní DAGy o kapacitě 6 tasků
- TriggerDagRunOperatory spouštějící listové DAGy o kapacitě 13 tasků
- Samotné nahrávající tasky o kapacitě 13 tasků

V jeden okamžik je tedy paralelně nahráváno až 13 tabulek. Po výkonové stránce jsou na straně datového skladu tyto parametry vyhovující. Vyskytuje se zde drobný problém s maximálním počtem připojení k target databázi – kvůli datové paralelizaci nahrávání modifikovaných záznamů může jedna instance nahrávání do IDL vytvořit až 32 spojení. V současné době je limit pro počet spojení nastaven na 100. V případě nejhoršího možného scénáře by řešení vyžadovalo až 416 databázových spojení, testovací běhy však ukázaly, že onen limit 100 spojení je překročen zhruba v jedné z pěti instancí celého nahrávání. Tento problém může být řešen buďto zvýšením počtu připojení do target databáze, vytvořením dalšího poolu pro nahrávání do IDL tabulek nebo kombinací obojího.

4.6.2 Řešení nežádoucích výkyvů doby běhu

Hrozbu problému s výkyvy dob běhu, která je popsána v podkapitole 3.6, navrhuje autor řešit nastavením parametru `wait_for_downstream` na hodnotu `True`, a to pro kořenový DAG. To zajistí, že běh nahrávání vždy začne až po skončení předchozího běhu. V nejhorším myslitelném případě tedy může nahrávání být v některý den vynecháno – autorovi není známo žádné rozumné řešení, které by toto riziko zcela odstranilo.

Autor by rád zdůraznil, že toto opatření je pouze prevencí proti špatným následkům těch největších výkyvů. Optimalizace, která má za cíl snížení výkonových dopadů všech výkyvů v počtech změn, nemá se zmíněným opatřením příliš společného a je popsána v podkapitole 4.6.5.

4.6.3 Použití tagů

Tagy nemají na samotný průběh nahrávání žádný vliv, slouží pouze ke zpřehlednění uživatelského rozhraní. Testovací řešení obsahuje 174 DAGů a je možné, že tento počet ještě poroste. Při takovém počtu je vhodné uživateli umožnit filtrování DAGů podle tagů. Autor v testovací verzi přiřadil tag každému zdrojovému systému a každému logickému celku v IDL, a vnitřní DAGy odlišil tagem `inner_node`. V této podobě má každý DAG minimálně dva tagy. Je možné, že v budoucnu bude logika přidělování tagů na základě porady změněna.

4.6.4 Priorita tasků

Vzhledem k tomu, že doba celého nahrávání je v současnosti určena dobou nahrávání dvou největších tabulek, je vhodné jim před ostatními tabulkami dát přednost. Toho autor docílil jednoduchým zvýšením parametru `priority_weight` pro vnitřní DAG `load_vvvs` a pro listové DAGy nahrávající příslušné dvě tabulky. Totéž autor učinil u tabulek `tusers`, `ttexty` a `tekns`, které jsou downstream závislostí pro velký počet vnitřních DAGů.

4.6.5 Optimalizace mimo Airflow

Řešení vytvořené v rámci této práce má na starosti pouze orchestraci nahrávání, samotné provádění jednotlivých atomických kroků je prováděno v PDI nebo přímo v databázi. Z toho vyplývá, že prostor pro optimalizaci nahrávání existuje nejen v Airflow, ale i na těchto dvou místech.

4.6.5.1 Optimalizace v databázi

Prostor pro optimalizaci přímo v databázi spočívá zejména v indexech. Při vývoji řešení se autor u několika náhodných tabulek potýkal s velmi pomalým nahráváním do IDL, které působilo dokonce jako zaseknutí. Bylo to způsobené tím, že parametrizovaná procedura `inc_clear_state_flag_schema_table_args` při svém běhu nevytvářela indexy pro stage-increment tabulky nad jejich BK a MD5 hashí. Tento problém byl sice vyřešen změnou procedury tak, aby klíče nad tabulkami ze systémů **KOS**, **Ezop** a **Usermap**, nicméně toto řešení zase vedlo k razantnímu prodloužení kroků `clean_stage` a `get_M_N_D_flags` u jiných tabulek, včetně dvou největších. V reakci na tuto skutečnost autor u dvou největších tabulek tvorbu indexů provizorně deaktivoval.

Je zde možnost vytvořit seznam tabulek, pro které se tato tvorba indexů vyplatí, a indexy vytvářet pouze pro ně. Jedná se ale o velmi pracnou úlohu, jejíž přínos by v současnosti byl u malých tabulek zanedbatelný.

4.6.5.2 Optimalizace v PDI

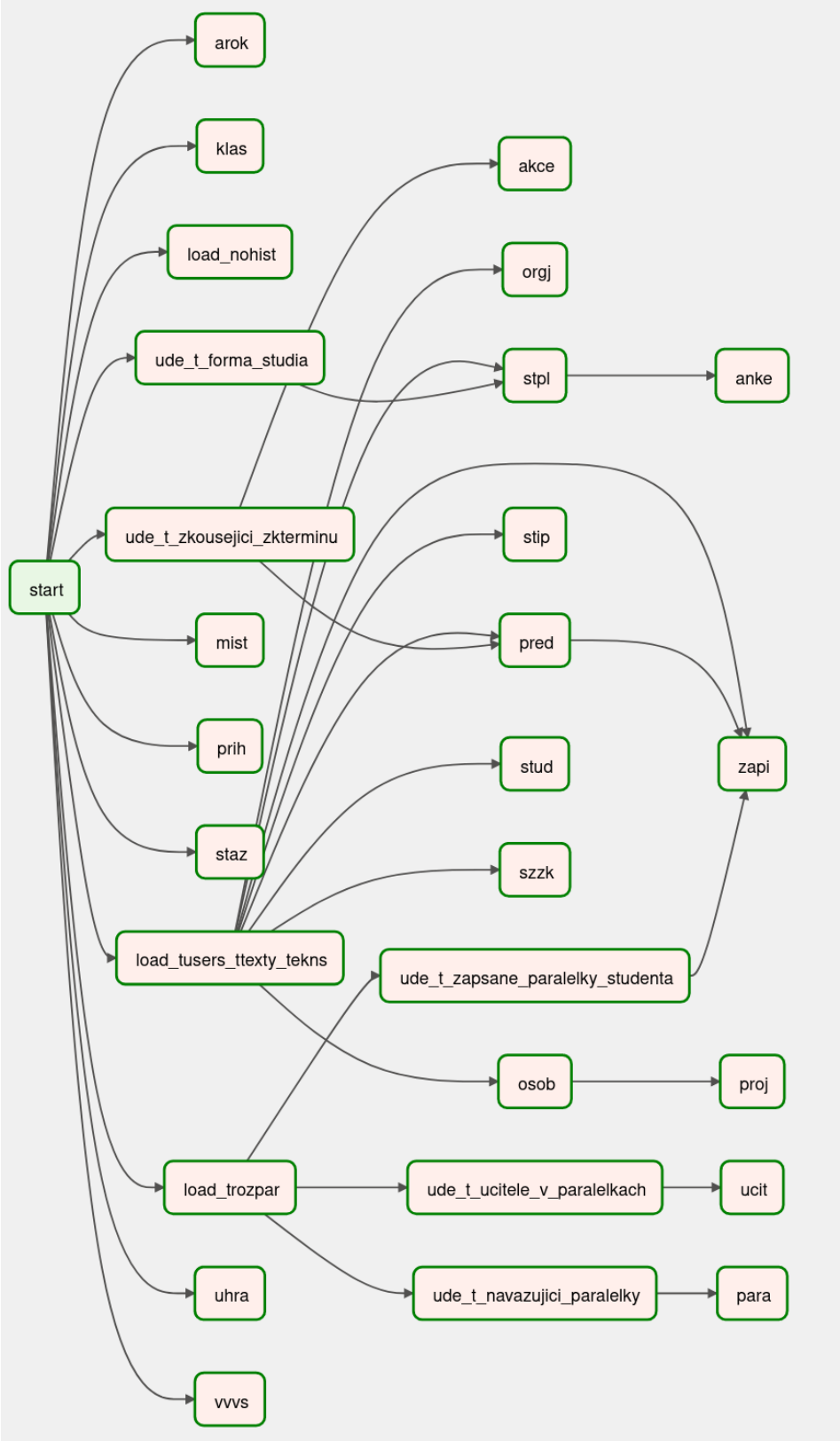
Jak již bylo zmíněno, doba běhu celého nahrávání je aktuálně zdola omezena dobou nahrávání tabulek `t_externiorganizacnijednotka_externicitaceautor_rel` a `t_vvvs_externi_citace_autor`. Drtivou většinu tohoto času zabírá jejich přesun ze zdrojového systému do pre-stage.[3] Tento problém vzniká převážně v PDI a vyžaduje hlubší analýzu, které se autor v tomto textu nevěnuje.

Mezi možnosti optimalizace v PDI patří například:

- Ladění parametrů datového paralelismu, který je používán v nahrávání změněných záznamů do IDL, či jeho zavedení na další místa.
- Zmenšení množiny atributů, ze kterých se počítá MD5 hash. Odebrat lze například atributy, které jsou zahazovány při technickém čištění tabulek. Tato optimalizace by zredukovala počet záznamů, které se ve stage-increment tváří jako modifikované, ale do IDL se stejně nedostanou.

Zmíněné optimalizace mohou mít za výsledek znatelné zrychlení, ale s přihlédnutím k omezeným lidským zdrojům v týmu DW ČVUT zatím nebyly provedeny. Může k nim však být přistoupeno v budoucnu.

[3] Toto platí pro běžný provoz, u větších výkyvů počtu změn nejdéle trvá nahrávání do IDL.



Obrázek 4.4 Kořenový DAG

Kapitola 5
Zhodnocení

V této kapitole najde čtenář nejprve srovnání doby běhu starého a nového řešení a zhodnocení udržitelnosti nového řešení. Následuje shrnutí výhod a nevýhod řešení a výčet toho, co musí ještě být zváženo, projednáno a doděláno před nasazením řešení do produkce. Na konci kapitoly se autor konfrontuje s jednotlivými body ze zadání a mírou jejich splněnosti.

5.1 Zhodnocení z hlediska efektivity

V levé části tabulky 5.1 jsou časy některých produkčních nahrávání, která proběhla za použití sekvenčního řešení zhruba v době vznikání této práce. Tyto časy jsou počítány od spuštění produkčního jobu do doběhnutí jobu *J_LOAD_LANDING* včetně (viz obrázek 1.5) – jsou zde tedy započítány i přípravné joby, které do paralelního řešení ještě nebyly zakomponovány, ale do doby celého běhu přispívají zanedbatelně.

V pravé části tabulky 5.1 jsou časy testovacích paralelních nahrávání. Na první pohled je zde vidět pozitivní dopad odstranění indexů u tabulek s citacemi ve stage-increment, ke kterému došlo 4. 4. 2024.

Výkyv, který nastal 20. 4. 2024, je způsoben velmi vysokým počtem modifikovaných záznamů v tabulce `t_vvvs_externi_citace_autor`. Proti výkyvům tohoto typu vzniklé řešení odolné bohužel není. Na obrázcích A.3 a A.4 se nachází Ganttovy diagramy pro čtenářovu lepší představu.

Autor práce je toho názoru, že **tyto výsledky lze považovat za úspěch**.

5.2 Zhodnocení z hlediska udržitelnosti

Implementované řešení má z hlediska udržitelnosti své problémy – Dříve byla pro manipulaci s ETL procesy DW ČVUT potřebná pouze znalost PDI a příslušných databázových mechanismů, po nasazení řešení do produkčního prostředí k potřeb-

Datum	Doba nahrávání
26. 2.	12:01
18. 3.	11:47
25. 3.	16:13
1. 4.	20:08
8. 4.	13:32
15. 4.	14:04
22. 4.	13:53

Datum	Doba nahrávání
2. 4.	06:00
3. 4.	05:22
4. 4.	03:44
6. 4.	03:32
9. 4.	03:48
10. 4.	03:30
20. 4.	08:14

Tabulka 5.1 Doby běhů původního řešení (vlevo) a nového řešení (vpravo). Hodnoty jsou uvedeny v hodinách a minutách, datumy jsou vázány k roku 2024.

ným znalostem přibyde znalost Apache Airflow a celého paralelního řešení, které bylo v této práci popsáno.

Řešení také trpí na nedostatečnou možnost plošných úprav. Některé důležité parametry je možné měnit v konfiguračním souboru `airflow.cfg`, ale velké množství plošných úprav je odsouzeno k nutnosti ručního zásahu do kódu všech dotčených DAGů či dokonce tasků.

Další nevýhodou je náročnější rozšiřování o nové tabulky. Pro přidání tabulky do procesu nahrávání je potřeba implementovat dvě PDI transformace jako dříve, ale nově je potřeba navíc implementovat nový DAG včetně jeho začlenění do hierarchie paralelního nahrávání, což typicky vyžaduje změnu ve vnitřním DAGu. V případě přidání celého logického celku je potřeba zmíněný postup uplatnit pro všechny dotyčné tabulky, k tomu vytvořit vnitřní DAG a začlenit ho do kořenového DAGu. Pro případ začleňování nového logického celku s nepraktickými upstream závislostmi (například z logického celku *vvs*) by autor rád zmínil existenci konstruktu `ExternalTaskSensor`, pomocí kterého lze s přimhouřeným okem obejít dosavadní způsob definování závislostí mezi částmi nahrávání.

Rozšiřitelnost ve smyslu použití nových mechanismů je naopak dobrá, a to díky zvolené technologii – Airflow nabízí násobně více pokročilých mechanismů, než bylo v této práci představeno. Oproti PDI má v tomto aspektu Airflow jasně navrch.

Autor je toho názoru, že tyto nevýhody sice nejsou zanedbatelné, ale v rámci potřeb DW ČVUT jsou snadno převáženy výhodami, zejména rychlostí běhu.

5.3 Silné stránky řešení

- **Rychlost běhu.** Ta je bezesporu hlavní výhodou řešení a snaha o její dosažení byla důvodem vzniku celé práce.
- **Snadnější orientace v logu.** Nové řešení umožňuje snadnou navigaci mezi logy na úrovni jednotlivých tasků. Ve starém řešení bylo možné pouze procházet jeden velký monolitický log soubor, což není příliš praktické.

- **Snadnější analýza průběhu**. V uživatelském rozhraní Airflow si může uživatel pro každý DAG snadno nechat vykreslit celou škálu různých grafů, což umožňuje výbornou orientaci v průběhu nahrávání a případné hledání slabých (pomalých) míst.
- **Rozšiřitelnost o nové mechanismy.** Pochází z rozhodnutí použít technologii Apache Airflow a může se do budoucna hodit.

5.4 Slabé stránky řešení

- **Udržitelnost.** Řešení přináší některé menší problémy s udržitelností. Popsány jsou v podkapitole 5.2.
- **Možnost výkyvů.** V podkapitole 5.1 je uvedeno, že řešení není výkonově odolné vůči výkyvům, kde v rámci jednoho běhu nahrávání přijde velké množství modifikovaných řádků pro jednu tabulku. Paralelní varianta nahrávání je v takovém případě stále rychlejší, než varianta sekvenční, nárůst doby běhu je nicméně výrazný. Potenciálním řešením tohoto problému by mohla být optimalizace na úrovni PDI či samotného SQL, na které již autor začal pracovat a jejíž možnosti popsal v podkapitole 4.6.5
- **Prostor pro nové chyby.** Staré řešení je plně funkční a je používáno již několik let. Při nahrávání se jednou za čas nějaká chyba vyskytne[1], ale chyby starého řešení jsou již poměrně dobře zmapované. Nové řešení přináší možné riziko nových, zatím neznámých chyb.
- **Nemožnost snadného testování.** Toto je spíše problém datových skladů obecně. Vždy lze snadno ověřit, jestli nahrávání proběhlo v pořádku či nikoliv, ale ověřit, že jsou data skutečně v pořádku a správná, je náročnější záležitost. Před nasazením řešení do produkčního prostředí bude tedy nutné vykonat velké množství manuálního testování.

5.5 Úkoly do budoucna

Řešení popisované v této práci je nasazené v testovacím prostředí DW ČVUT. Než bude nasazeno do produkčního prostředí, musí nad ním ještě být vykonáno několik dílčích úkolů. Autor práce předkládá jejich výčet:

- **Rozsáhlé testování.** Nejprve je potřeba ručně a extrémně důkladně prověřit, že jsou v grafu nahrávání správně podchyceny všechny závislosti, a že je každá

[1] Obvykle to není chyba na straně DW ČVUT, může se jednat například o výpadky zdrojových systémů či o překročení povolené délky textového řetězce (způsobené zdrojovými daty).

operace pro každou tabulku provedena právě jednou. Pak je třeba provést dostatek testovacích běhů nahrávání, protože chyby zákeřnějšího charakteru se zde projevují s odstupem.[2] Nakonec je potřeba otestovat správnost dat, a to opět ručně. Toto testování zabere velké množství času, ale je nezbytné.

- **Obnova datamartů.** Je to nedílná součást procesu aktualizace datového skladu, která následuje hned po nahrávání. V této práci se jí autor nevěnoval. Pro testovací účely postačuje spuštění staré, sekvenční obnovy datamartů hned po nahrání nových dat do IDL. Efektivnější by byla nová implementace obnovy datamartů využívající paralelismus obdobně jako nahrávání popsané v této práci. Vůbec nejefektivnější by bylo řešení, které by bylo úzce propojené se samotným nahráváním dat do skladu a obnovu jednotlivých datamartů by spouštělo hned po nahrání všech potřebných tabulek do IDL. Takové řešení by svou pracností však představovalo další samostatnou bakalářskou práci, možná i práci diplomovou. Implementována bude jedna ze tří zmíněných možností podle pracovních kapacit DW ČVUT.

- **Zprovoznění automatických emailových notifikací.** Je potřeba, aby pracovníci zodpovědní za provoz DW ČVUT byli o stavu produkčního nahrávání automaticky notifikováni, například e-mailem. V případě pádu některé části nahrávání musí být pracovníci notifikováni neprodleně. Airflow tuto funkcionalitu nabízí.

- **Pokračování v optimalizaci mimo Airflow.** Toto je potřeba zejména k minimalizaci výkonových dopadů v případě velkého množství změněných záznamů. V současné době je tato optimalizace potřebná u dvou největších tabulek. Jedná se primárně o nahrávání ze stage-increment do IDL, ale vhodná je i optimalizace nahrávání ze zdrojových systémů do pre-stage, například formou datové paralelizace, jejíž možnosti již zkoumal Adam Marhefka v [7].

- **Domluva se správci zdrojových systémů.** Než bude stanoven čas pravidelného nahrávání, je potřeba si u jednotlivých správců ověřit, že je tento čas vyhovující a že nahrávání v tuto dobu nepovede k přetížení žádného ze systémů. Taky bude vhodné se ujistit, že u žádného ze systémů nemůže dojít k překročení maximálního počtu databázových připojení.

- **Sepsání dokumentace.** Až se bude řešení nacházet v nasaditelné podobě, bude ho potřeba jasně, srozumitelně a přehledně zdokumentovat.

[2] Z vlastní zkušenosti autora během práce na řešení

5.6 Zhodnocení splnění zadání

- **1.** Popis technologie Apache Airflow se nachází v kapitole 2. Autor se s bakalářskou prací Kristiny Zolochevské seznámil, ale od jejího PoC se z důvodu manažerského rozhodnutí odchýlil. Tato odchylka je popsána v podkapitole 3.1.
- **2.** Přehled užitečných funkcionalit, které Airflow ve svých nových verzích nabízí, se nachází v podkapitole 2.7. Nové přístupy v nástroji Apache Airflow se vyskytují v kapitolách 3 a 4 – z těch podstatnějších lze uvést například externí spouštění DAGů pomocí operátoru TriggerDagRunOperator.
- **3.** Návrh a popis implementovaného řešení se nachází opět v kapitolách 3 a 4, implementované řešení se nachází v příloze. Na oddělení VIC došlo ke shodě na tom, že rozsah implementovaného řešení je pro potřeby této práce postačující a že další části popsané v podkapitole 5.5 budou doladěny později.
- **4.** Řešení je zhodnoceno v této kapitole.

Závěr

Cílem této práce bylo implementovat paralelní nahrávání datového skladu ČVUT, zejména s cílem tento proces urychlit oproti současnému sekvenčnímu řešení.

Pro toto řešení byla již před vznikem této práce zvolena technologie Apache Airflow, kterou ve své bakalářské práci v roce 2023 analyzovala Kristina Zolochevskaia.

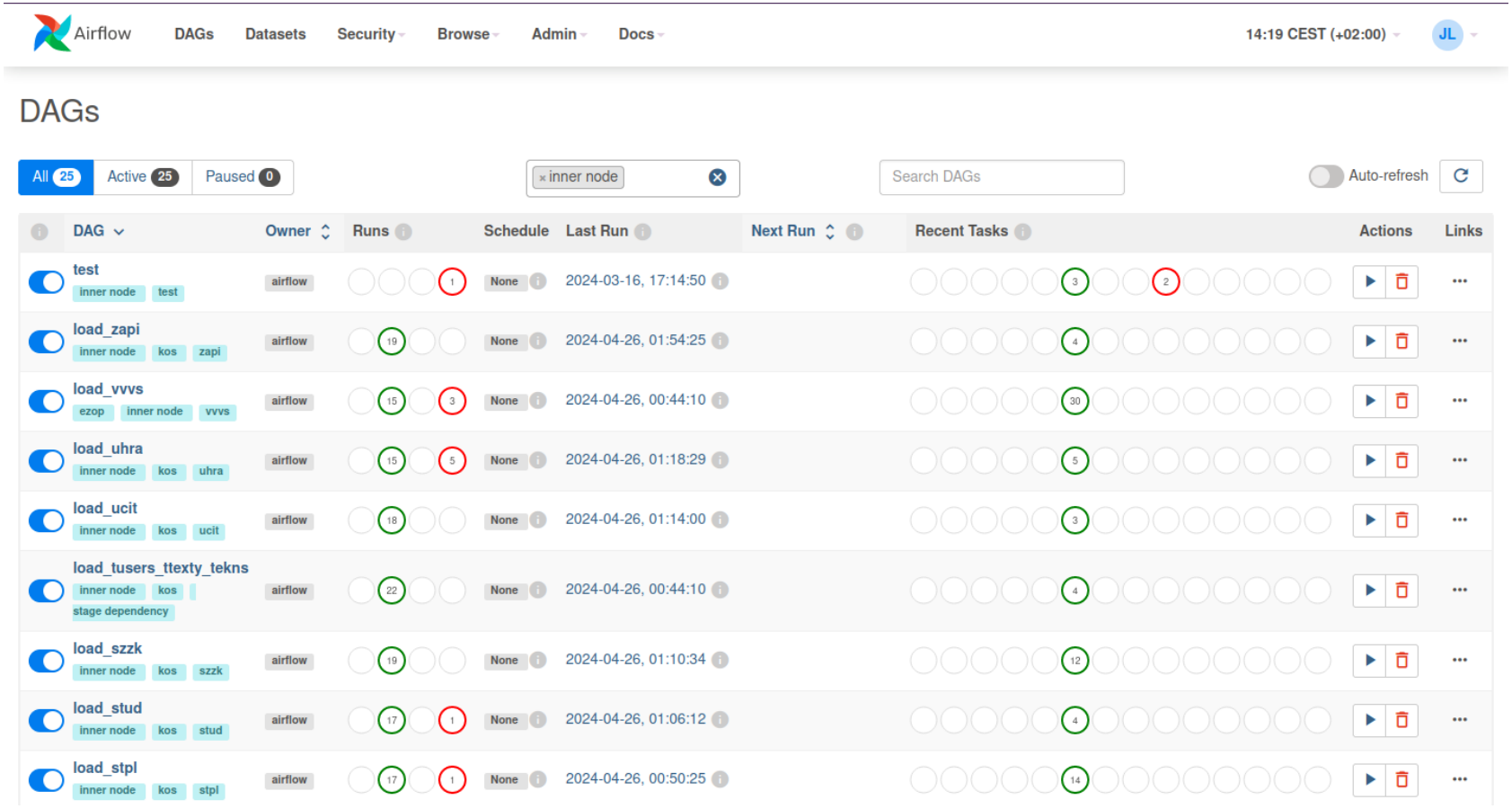
Testovací varianta tohoto řešení byla v této práci úspěšně navržena a implementována. Řešení bylo podrobeno více než deseti testovacím běhům, ze kterých je jednoznačně vidět, že požadovaného zrychlení bylo dosaženo.

Než bude řešení nasazeno do produkčního prostředí, je nad ním potřeba vykonat ještě několik úkolů menšího rozsahu – těm se autor bude nadále věnovat po dokončení této práce.

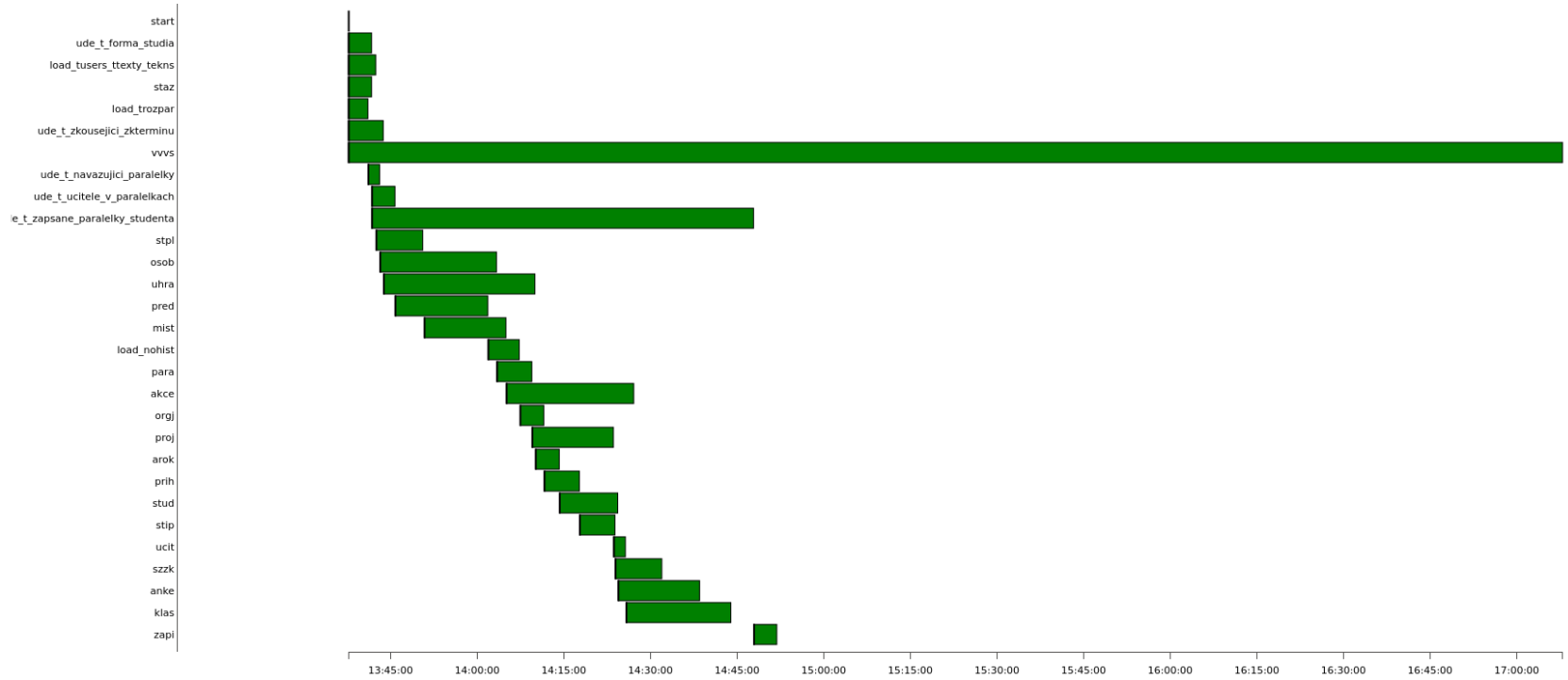
Příloha A

Screenshoty z Airflow

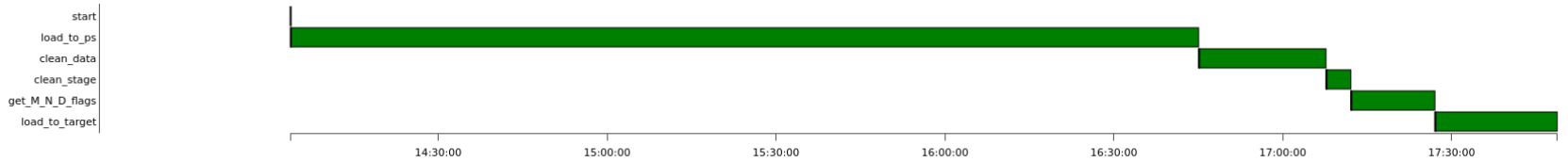
Tato příloha obsahuje vybrané screenshoty z Apache Airflow.



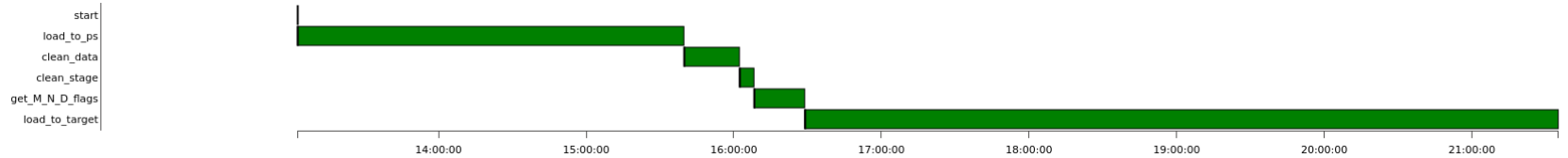
Obrázek A.1 Přehled DAGů v uživatelském rozhraní Airflow



Obrázek A.2 Ganttův diagram pro typický běh kořenového DAGu



Obrázek A.3 Ganttův diagram pro největší tabulku (typický běh)



Obrázek A.4 Ganttův diagram pro největší tabulku při vysokém počtu modifikovaných záznamů

Bibliografie

1. NOVOTNÝ, Ota; POUR, Jan; SLÁNSKÝ, David. *Business intelligence: jak využít bohatství ve vašich datech*. 1. vyd. Praha: Grada, 2005. ISBN 80-247-1094-3.
2. KOTLÁŘ, Robert. *Datový sklad ČVUT - způsoby datové integrace*. Praha, 2017. Diplomová práce. České vysoké učení technické v Praze, Fakulta informačních technologií.
3. KOTLÁŘ, Robert; KREJČÍ, Jakub. *Podnikové datové sklady (NI-EDW): 2. přednáška* [online]. 2024-03-07. [cit. 2024-05-08]. Dostupné z: `https://courses.fit.cvut.cz/MI-EDW/media/lectures/02_prednaska.pdf`.
4. INMON, William H. *Building the Data Warehouse,3rd Edition*. 3rd. USA: John Wiley & Sons, Inc., 2002. ISBN 0471081302.
5. ARIYACHANDRA, Thilini; WATSON, Hugh J. Technical opinion: Which data warehouse architecture is best? *Communications of the ACM*. 2008, roč. 51, č. 10, s. 146–147. ISSN 0001-0782. Dostupné z DOI: `10.1145/1400181.1400213`.
6. TAMIMI, Naser. *Fundamentals of Data Warehouses for Data Scientists* [online]. [cit. 2024-05-03]. Dostupné z: `https://towardsdatascience.com/fundamentals-of-data-warehouses-for-data-scientists-5314a94d5749`.
7. MARHEFKA, Adam. *Paralelizácia ETL procesov DW ČVUT s využitím nástroja Pentaho*. Praha, 2023. Bakalárska práca. České vysoké učení technické v Praze, Fakulta informačních technologií.
8. KIMBALL, Ralph; ROSS, Margy; THORNTHWAITE, Warren; MUNDY, Joy; BECKER, Bob. *The Data Warehouse Lifecycle Toolkit*. 2. vyd. Somerset: Wiley, 2011. ISBN 9780470149775.
9. ALSQOUR, Moh'd; MATOUK, Kamal; OWOC, Mieczysław L. A survey of data warehouse architectures — Preliminary results. In: *2012 Federated Conference on Computer Science and Information Systems (FedCSIS)*. 2012, s. 1121–1126.

10. ARNOŠT, Daniel. *Kontext BI v rámci komerční firmy, Framefork architektury BI a Konceptuální BI architektura [Přednášky MI-EDW]* [online]. 2017. [cit. 2024-05-08]. Dostupné z: `https://courses.fit.cvut.cz/NI-EDW/@B182/media/lectures/lecture1-2_thearchitectureofedw.pdf`.

11. AMAZON WEB SERVICES. *What is ETL (Extract Transform Load)?* [online]. [cit. 2024-04-30]. Dostupné z: `https://aws.amazon.com/what-is/etl/`.

12. INTERNATIONAL BUSINESS MACHINES CORPORATION. *What is ETL (extract, transform, load)?* [online]. [cit. 2024-04-30]. Dostupné z: `https://www.ibm.com/topics/etl`.

13. APACHE SOFTWARE FOUNDATION. *Apache Airflow Documentation* [online]. 2023-08-10. Ver. 2.6.3 [cit. 2024-04-30]. Dostupné z: `https://airflow.apache.org/docs/apache-airflow/2.6.3/index.html`.

14. ZOLOCHEVSKAIA, Kristina. *Parallelization of ETL processes DW CTU – case study*. 2023. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology.

Obsah příloh

- `src`
  - `impl` ..... zdrojové kódy implementace
    - `readme.txt` ..... Stručný popis obsahu zdrojových kódů
    - `airflow` ..... Zdrojové kódy DAGů a konfigurační soubor
    - `postgres` .... DDL, data a připravené dotazy pro schéma se závislostmi
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `BP_lejsejir_2024.pdf` ..... text práce ve formátu PDF