FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Tvorba stage vrstvy datového skladu (DWH) na bázi metadat v prostředí datové platformy Databricks v cloudu - případová studie
Student:	Vítězslav Hušek
Vedoucí:	Ing. Michal Valenta, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	do konce letního semestru 2024/2025

Pokyny pro vypracování

1) Uveďte představení cloudové datové platformy Databricks, k jakému účelu slouží, jaké jsou její výhody a nevýhody.
2) Vypracujte přehled aktuálních metod a přístupů k tvorbě stage vrstvy DWH v prostředí datové platformy Databricks. Zaměřte se zejména na postupy, které jsou založeny na generování a automatizaci pomocí metadat v rámci této platformy a proveďte jejich srovnání s klasickým vývojem.
3) Popište framework dbt, který využívá metadatový přístup a slouží pro automatizaci vývoje datových řešení v datové platformě Databricks.
4) Realizujte případovou studii tvorby stage vrstvy DWH v datové platfromě Databricks s použitím frameworku dbt na základě metadat dodaných zadavatelem.
5) Zhodnoťte přínosy a slabiny zvoleného přístupu s ohledem na rychlost vývoje, snadnost zapracování nových požadavků na změny a tvorbu dalších vrstev DWH.

Elektronicky schválil/a Ing. Magda Friedjungová, Ph.D. dne 9. prosince 2023 v Praze.

Bakalářská práce

TVORBA STAGE VRSTVY DATOVÉHO SKLADU (DWH) NA BÁZI METADAT V PROSTŘEDÍ DATOVÉ PLATFORMY DATABRICKS V CLOUDU - PŘÍPADOVÁ STUDIE

Vítězslav Hušek

Fakulta informačních technologií
Katedra aplikované matematiky
Vedoucí: Ing. Michal Valenta, Ph.D.
10. května 2024

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Chtěl bych poděkovat svému vedoucímu za ochotu a zpětnou vazbu během psaní této práce. Dále bych rád poděkoval všem, kteří se mnou konzultovali obsah práce z odborného hlediska. V neposlední řadě bych rád poděkoval své přítelkyni a blízké rodině za veškerou trpělivost a podporu.

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 10. května 2024

Abstrakt

Tato bakalářská práce se zabývá konceptem moderních datových skladů v cloudu se zaměřením na platformu Databricks a zkoumá ji z hlediska vývojáře a práce s metadaty. V rámci práce je vytvořena stage vrstva datového skladu s načítáním, transformací, historizací a kontrolou kvality dat s pomocí dostupných nástrojů a následně jsou zhodnoceny výhody a nedostatky oproti tradičním datovým skladům. Jsou porovnány frameworky dbt a DLT a jejich rozdílné vlastnosti.

Je zjištěno, že framework dbt poskytuje přehlednější prostředí a rozdělení funkcionalit do individuálních modulů, které napomáhá lepšímu porozumění ze strany vývojářů nezkušených s tímto frameworkem.

Výsledky této práce umožňují podnikům a vývojářům, kteří zvažují modernizaci datového skladu do oblasti cloudu, udělat informované rozhodnutí o vhodnosti, nákladech a přínosech takového kroku, případně zvolit vhodný framework pro své řešení.

V příloze práce se nachází zdrojové kódy, které vznikly v rámci implementace stage vrstvy datového skladu.

Klíčová slova případová studie, Databricks, dbt, DLT, cloud, datový sklad, lakehouse, big data, stage vrstva

Abstract

This thesis explores the concept of modern data warehouses in the cloud, focusing on the Databricks platform and examining it from a developer and metadata perspective. The thesis develops a staging layer of the data warehouse with data loading, transformation, historization and quality control using available tools and evaluates the advantages and disadvantages compared to traditional data warehouses. A comparison is made between the dbt and DLT frameworks and their different features.

In the final assessment, it is determined that the dbt framework provides a clearer environment and a division of functionality into individual modules, making it easier for developers unfamiliar with the framework to understand it.

The results of this work will enable organisations and developers considering moving their data warehouse to the cloud to make an informed decision about the suitability, costs and benefits of such a move, or to choose an appropriate framework for their solution.

The source code created during the implementation of the data warehouse stage layer is available in the appendix of the thesis.

Keywords case study, Databricks, dbt, DLT, cloud, data warehouse, lakehouse, big data, stage layer

Seznam zkratek

TB	Terabyte
SQL	Structured Query Language
DLT	Delta Live Table
DWH	Data Warehouse
dbt	Data Build Tool
SQL	Structured Query Language
ACID	Atomicita, Konzistence, Izolovanost, Durabilita
QoL	Quality of Life
LLM	Large Language Model
SaaS	Software as a Service
DBU	Databricks Unit
ETL	Extract, Transform, Load
ELT	Extract, Load, Transform
YAML	YAML Ain't Markup Language
IDE	Integrated Development Environment
VNET	Virtual Network
VPC	Virtual Private Cloud
TLS	Transport Layer Security
API	Application Programming Interface
XML	eXtensible Markup Language
OIM	Open Information Model
CWM	Common Warehouse Metamodel
UML	Unified Modeling Language
CLI	Command Line Interface
DBFS	Databricks File Storage
SCD	Slowly Changing Dimension
SCD2	Slowly Changing Dimension Type 2

Úvod

Data jsou dnes všude kolem nás. Datové nosiče, které dnes zvládají pojmout až stovky TB (Terabyte) se staly běžnou součástí každé větší firmy a rozšířil se zájem o ukládání dat v co největším možném měřítku. Pro společnosti, které data ukládají, nebo zpracovávají, následuje poté často velmi obtížná úloha velké objemy dat analyzovat a využít pro hledání rozličných vzorů, které nemusí být pouhým okem viditelné. Tomuto účelu dnes výrazně napomáhají datové sklady pro strukturovaná data, nebo datová jezera (data lakes) pro strukturovaná, polostrukturovaná a nestrukturovaná data. Tyto systémy jsou dnes esenciálními pilíři moderní práce s daty a zajišťují flexibilitu, škálovatelnost, větší efektivitu a rozšířené možnosti pro průzkum dat.

Na popularitě nabývají cloudová řešení, která poskytují řešení pro datové sklady jako službu (SaaS z anglického „Software as a Service“). Společnosti, které taková řešení poskytují, vyzdvihují především zjednodušení celého procesu vývoje datového skladu, úsporu nákladů, zabezpečení dat a flexibilitu škálovatelnosti, kterou klasické in-house (interní) datové sklady neposkytují. Mnoho firem se z tohoto důvodu začalo o cloudové technologie datových skladů zajímat a přemýšlet, zda by opravdu takový přístup nebyl vhodnější a modernější. Jelikož jde o velmi nové a aktuální téma, které se neustále vyvíjí, existuje celkem málo expertů, kteří jsou schopni na základě znalostí a zkušeností rozhodnout, které řešení je pro konkrétní firmu to správné. Z důvodu zvyšujícího se zájmu ze strany zákazníků a nedostatku literatury rozebírající téma datových skladů v cloudu, jsem se rozhodl vypracovat bakalářskou práci na dané téma.

Tato bakalářská práce si klade za cíl zaměřit se na analytickou platformu Databricks, představit její přednosti a nedostatky, ukázat na příkladech základní orientaci a práci v prostředí webového rozhraní, vytvořit v ní stage vrstvu datového skladu na konkrétních datech a zhodnotit výhody a nevýhody daného řešení v porovnání s klasickým vývojem. Stage vrstva bude vytvořena celkem dvakrát, poprvé s pomocí frameworku (rámce) dbt (data build tool) a podruhé s pomocí DLT (Delta Live Tables). Jedná se o možné přístupy zaměřené na práci s metadaty. Po jejich implementaci v rámci praktické části budou oba přístupy srovnány z hlediska požadavků na znalosti vývojářů a posouzení vhodnosti použití pro datové sklady s ohledem na rychlost vývoje, snadnost zpracování nových požadavků na změny a tvorbu dalších vrstev DWH (datového skladu, z anglického „Data Warehouse“). Celkovým přínosem by měla být znalost, na základě které bude možné porovnat přínosy a slabiny jednotlivých přístupů a pro konkrétní zadání zvolit nejlepší možnou variantu datového skladu a vývojových nástrojů.

Kapitola 1
Platforma Databricks

„Cloud je o tom, jak se provádí výpočty, ne kde se provádí výpočty.“
- Paul Maritz, CEO společnosti VMware [1]

1.1 Představení

Databricks je analytická platforma založená roku 2013 tvůrci Apache Spark, Delta Lake a MLflow. Jedná se cloudově agnostickou platformu, může být spuštěna například na službách Amazon AWS, Microsoft Azure a GCP (Google Cloud Platform). Slouží pro vytváření, nasazování, sdílení a údržbu dat, analytiku, práci s umělou inteligencí ve velkém měřítku a je optimalizovaná pro Microsoft Azure. Platforma jako taková běží v cloudu a je k ní nejčastěji přistupováno přes webové rozhraní (je však možné přistupovat i s pomocí Rest API, CLI nebo Terraform), samotná práce s kódem přes webové rozhraní probíhá ve formě notebooků. Tyto notebooky jsou následně spouštěny na jednotlivých clusterech, které je možné mít spuštěné ve dvou režimech. První režim, **Interactive**, též známý pod názvem **all-purpose cluster**, je využitelný pro různé dotazy v rámci notebooků, má vysokou flexibilitu a dle potřeby může být zapnut či vypnut. Druhým spustitelným režimem je **On Demand**, známý také jako **Job cluster**, který slouží především pro vykonání specifikované úlohy (jobu), nebo pro spuštění konkrétního kódu a po jeho provedení následuje je smazán. [2]

Platforma se primárně dělí na trojici prostředí:

- **Databricks Data Science and Engineering** je interaktivní pracovní prostředí umožňující spolupráci mezi data inženýry, data scientisty, machine learning inženýry a business analytiky a poskytuje možnosti vytváření data pipeline pro velká data.
- **Databricks SQL** slouží jako prostředí pro SQL dotazování se nad daty. Podporuje několik vizualizačních způsobů na procházení výsledků jednotlivých dotazů a k vyhodnocování dotazů používá specifické clustery, nazvané SQL Warehouse.
- **Databricks Machine Learning** poskytuje prostředí pro strojové učení. Umožňuje načtení dat a jejich předzpracování, feature engineering (výběr a vytváření nových atributů, které mohou zlepšit výkon modelu s využitím Spark SQL, MLib nebo knihovny scikit-learn), trénování modelů s pomocí AutoML, správu životního cyklu modelů v Unity Katalogu, nasazování modelů pro předávání získaných znalostí v reálném čase (*real-time*, též známý jako *stream*) nebo po dávkách (*batch*) a monitorování nasazených modelů.

Pracovat s metadaty je možné (v různé míře) v několika různých frameworcích (rámcích), které jsou v Databricks k dispozici. Pro tuto práci jsou nejzajímavější dbt (transformační nástroj

se zaměřením na metadata), DLT (nástroj pro procesy ETL, tedy procesy extrahování, transformace a načítání), případně DLT-META, knihovna umožňující definování DLT pipeline pomocí metadatových konfigurací.

1.2 Způsoby využití platformy

S pomocí nástrojů této platformy lze využít zdroje dat na různé procesy jako například zpracování, ukládání, sdílení, analýza, modelování a monetizace datových sad. Mnoho organizací v současné době pro tyto procesy provozuje složitou kombinaci datových jezer a datových skladů a využívá paralelní data pipelines pro zpracování dat v reálném čase nebo po jednotlivých dávkách. Následně často musí používat další nástroje pro analýzu, business intelligence nebo data science. Databricks se snaží ulehčit od této roztříštěnosti systémů a poskytnout všechny potřebné nástroje přehledně na jednom místě. Mezi základní výhody a poskytované funkcionality patří: [3]

- Snadná práce s daty ve formě v reálném čase i po dávkách.
- Transformace a organizování dat.
- Výpočty na datech.
- Dotazování se nad daty s pomocí SQL.
- Analyzování dat.
- Využití dat pro strojové učení a umělou inteligenci.
- Generování reportů a vizualizací.

Tento přístup, který dovoluje pracovat se strukturovanými, polostrukturovanými i nestrukturovanými daty bývá často nazýván pojmem „*data lakehouse*“ (spojení pojmů Data Warehouse a Data Lake) a snaží se poskytnout benefity obou přístupů bez nutnosti mezi nimi vybírat. Není však nutné podřizovat veškeré úkony právě pod Databricks. Je možné využívat je paralelně s jinými technologiemi, které má daná organizace již zavedené např. u jiných poskytovatelů cloudových služeb.

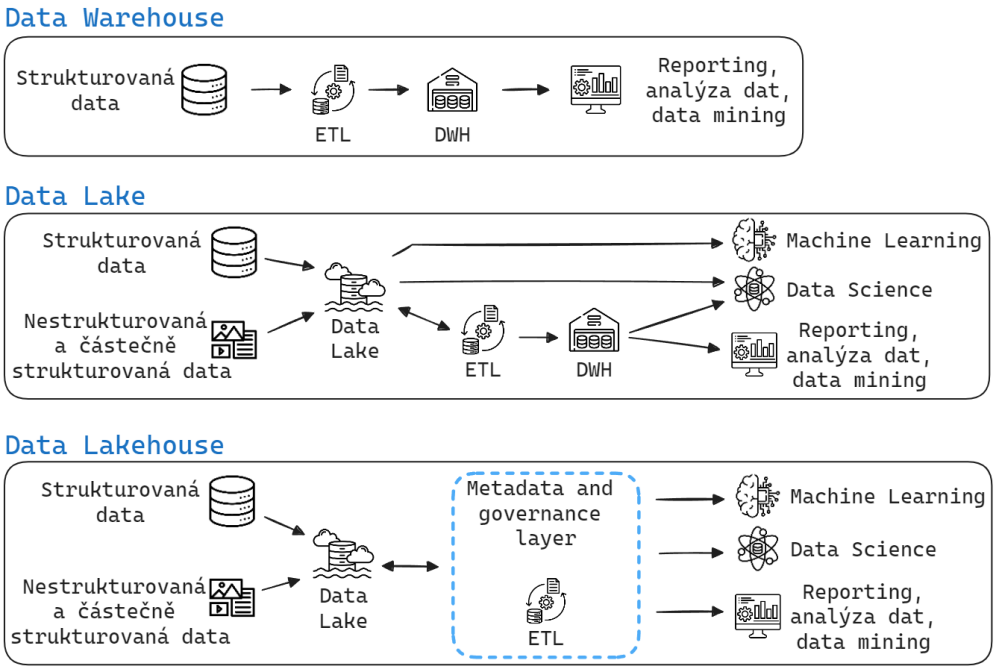
1.3 Data Lakehouse

Databricks Data Lakehouse představuje datovou platformu, která integruje nejlepší aspekty datových skladů a datových jezer do jednoho řešení pro správu dat. [7] Jelikož je stavěn na základu Data Lake, poskytuje ACID transakce (tedy transakce dodržující atomicitu, konzistenci, izolovanost a durabilitu) a značnou flexibilitu, kromě toho však poskytuje větší efektivitu při ukládání velkých dat a práci s nimi.

Zatímco datové sklady bývají výkonnější než datová jezera, mohou být nákladnější a omezené v možnostech škálování. Data Lakehouse se snaží tuto problematiku řešit využitím cloudového objektového úložiště k ukládání širší škály datových typů, tj. strukturovaných dat, nestrukturovaných dat a polostrukturovaných dat. Sjednocení pod jednu datovou architekturu přináší zrychlení zpracovávání dat při škálování a provádění pokročilejších analýz, protože už není potřeba přebytečné komunikace mezi dvěma odlišnými datovými systémy. [7]

1.4 Výhody

Následuje přehled předností, jak oproti klasickému přístupu, tak i v porovnání s ostatními rozšířenými platformami pro práci s DWH (datovými sklady) v cloudu, jakými je například Snowflake.



Obrázek 1.1 Porovnání DWH, Data Lake, Data Lakehouse. Obrázek vytvořen autorem s pomocí nástrojů [4][5]. Inspirace: [6].

1.4.1 Přístupnost

```
02:10 PM (38s)                                                    Cell 1

%python
df = spark.read.format("json").load("/databricks-datasets/nyctaxi/sample/json/")
df.createOrReplaceTempView("TaxiTable")
```

```
02:10 PM (2s)                                                     Cell 2

%sql
select fare_amount, passenger_count, tip_amount, trip_distance from TaxiTable
```

```
02:10 PM (< 1s)                                                   Cell 3

%python
df = _sqldf
df.describe()
```

Obrázek 1.2 Ukázka přepínání jazyku v navazujících buňkách. Nejprve je v pythonu načten dataset nyctaxi, následně v SQL vybrány chtěné sloupce a nakonec jsou vypsány informace o výsledných sloupcích v pythonu.

Jednou z hlavních výhod je Apache Spark, robustní open-source systém pro práci s velkými daty, na kterém celý Databricks stojí. Nad rámec Apache Spark využívá dalších komponent pro zlepšení výkonu, bezpečnosti a přístupnosti. Mezi předinstalované knihovny patří Python, Scala, R, Java, umožňující vytváření data pipelines v jakémkoliv jazyce. [2] V rámci vývoje jsou často využívány notebooky, dělené na jednotlivé samostatně spustitelné buňky (útržky kódu, také

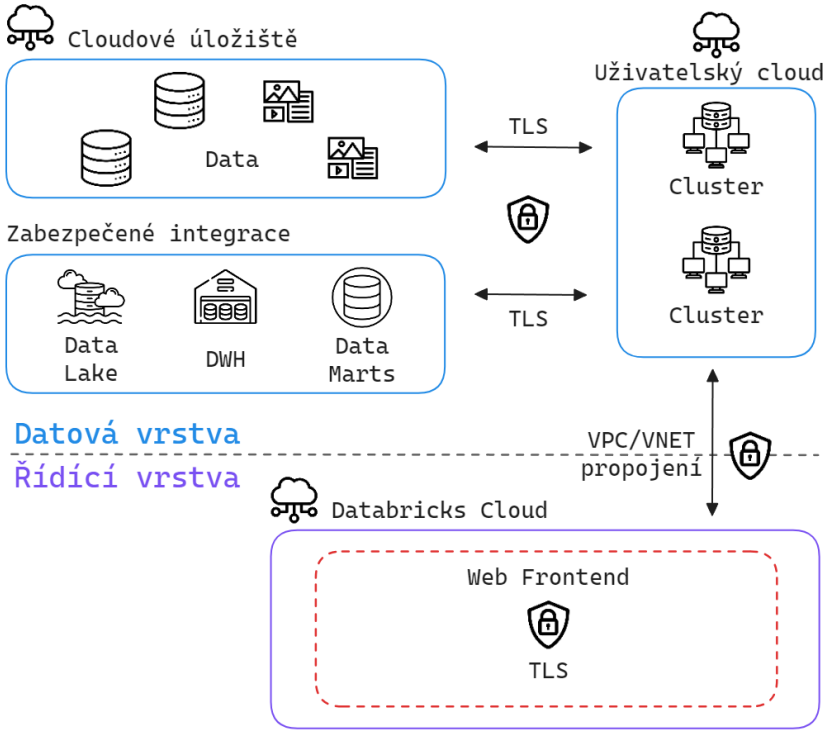
známé jako *cells*). Výhodou, kterou v Databricks tyto notebooky poskytují, je přepínání jazyků možné individuálně pro každou buňku. Stačí v každé buňce uvést na prvním řádku jazyk, ve kterém se má kód provést, společně s dekorátorem %, tedy například `%python`.

Jak lze z ukázky na obrázku 1.2 vidět, dotazy provedené v SQL se ukládají jako Spark dataframe do proměnné `_sqldf`, takže se výsledky dají jednoduše načíst a využít i v prostředí pythonu. Je také možné propojit vývojové prostředí Databricks s vlastním integrovaným vývojovým prostředím (IDE, například. Eclipse, PyCharm, Visual Studio Code) nebo notebookovým serverem (Zeppelin, Jupyter Notebook). Rozsáhlá dokumentace také napomáhá porozumění jak celkovému fungování, tak dílčím částem. Vše je podpořeno webovým uživatelským rozhraním, které dovoluje se pohybovat po celém systému myší a umožňuje tak lepší pochopení o fungování i méně technicky zdatným uživatelům.

1.4.2 Data, interoperabilita

Jak již bylo zmíněno v kapitole 1.1, Databricks je cloudově agnostickou platformou, což znamená že pro jeho používání nevyžaduje přesunutí dat pod soukromý systém. Místo toho se připojuje k účtu v cloudovém prostředí (Google, Azure, AWS) a je tak možné kdykoliv s daty manipulovat bez obav o vendor lock-in (situace, kde změna platformy by vedla k takové náročnosti, kdy je zákazník „uzamčen“ u stávající platformy). Pro sdílení dat je možné využít Delta Sharing protokol, umožňující bezpečné sdílení velkého objemu dat bez závislosti na prostředí, které společnosti používají.

1.4.3 Bezpečnost



Obrázek 1.3 Dvouúrovňová bezpečnost v Databricks. Obrázek vytvořen autorem s pomocí nástrojů [4][5]. Inspirace: [8].

Jednotlivé operace nad daty jsou děleny na dvojici cloudových prostředí. První, uživatelské (data plane), nazývané také jako datová vrstva, kde se shlukují veškerá data, výpočetní zdroje, výsledky spouštěných skriptů (notebooků) a druhé (control plane), nazývané též jako řídící vrstva, které spravuje notebooky, dotazy, úlohy (jobs) a clustery. Informace, které se ukládají na straně Databricks (konfigurační soubory, logy) jsou enkryptovány, stejně jako veškerá komunikace mezi těmito dvěma prostředími. Komunikace mezi vrstvami probíhá s pomocí VPC (Virtual Private Cloud) nebo VNET (Virtual Network) zabezpečeného propojení. V rámci vnitřní komunikace je v každé vrstvě využívána komunikace s pomocí protokolu TLS (Transport Layer Security). Clustery běží v odstíněném soukromém virtuálním cloudu.

Hierarchie vývojářských účtů a pracovních prostorů se zakládá především na oddělení pracovních prostorů od sebe navzájem (logicky i fyzicky) a rozdělení přístupů jednotlivým účtům vývojářů a dalších uživatelů mezi tyto prostory. Pro administrátory je dostupný administrátorský účet, který má dostatečná práva na správu různých skupin uživatelů (případně jednotlivce) a přiřazuje jim adekvátní přístupy a oprávnění.

Další vrstvy na podporu bezpečnosti lze získat z Unity Katalogu, který nabízí místo pro správu přístupu k datům, napříč všemi pracovními prostory. Tak je možné dosáhnout například zpřístupnění konkrétních sloupců nebo řádků v datech pouze specifickým uživatelům, nebo skupinám. Platforma poskytuje také auditing features pro monitorování aktivity uživatelů pro dodržování norem jako jsou HIPAA (Health Insurance Portability and Accountability Act, norma pro data ze zdravotnictví) nebo PCI DSS (Payment Card Industry Data Security Standard, norma pro data o platebních kartách).

1.4.4 Pravidelné inovace

Celá platforma se neustále vyvíjí a aktualizace přináší nové způsoby využití i zvýšení QoL (větší přívětivost, tzv. zlepšení úrovně života z anglického „Quality of Life“) pro uživatele. Na konferenci Data and AI Summit 2023 byl představen LakehouseIQ, engine na bázi umělé inteligence umožňující práci s daty s použitím přirozeného jazyka. [9] S rozvojem LLM (Large Language Model) jde o snahu využít tyto možnosti i nad vlastními daty, ať už pro účely vyhledávání, management, nebo vytváření nových aplikací. Předností LakehouseIQ je možnost naučení se na datech dostupných v pracovním prostředí a následného pochopení kontextu z uživatelských dotazů, vizualizací a komentářů a samotného kódu v noteboocích. [10] Může sloužit jako generátor SQL dotazů nad konkrétními daty, nebo asistent při psaní kódu a ladění chyb.

1.5 Nevýhody

Navzdory zmíněným výhodám se nejedná o jedinečné řešení, nahrazující stávající postupy nebo ostatní platformy. Pokud pomineme relativně malou komunitu oproti ostatním cloudovým řešením (a celkově velmi malou komunitu oproti stávajícím klasickým řešením), stále jsou zde daleko markantnější nevýhody, které je také potřeba zmínit.

1.5.1 Křivka učení

Vysoký počet nástrojů a integrací, které je možné využít, společně s konceptem Data Lakehouse, na první pohled přehlcuje uživatele a nastavuje vysokou úroveň na porozumění pro efektivní použití. Dostupná dokumentace do jisté míry snižuje tuto náročnost, ale orientaci v prostředí jako takovou ulehčit tolik nedokáže. Absenci vizualizací pro ne-programátory částečně řeší Dashboards, o které byla platforma rozšířena v roce 2023, stále však chybí možnost pracovat s většími objemy dat pomocí drag-and-drop, dnes již velmi rozšířeným standardem na přemisťování dat a takové operace, mezi které patří načtení dat v kódu a jejich ukládání tedy stále zůstávají součástí komplexního učícího procesu. Nahrání dat je sice možné s pomocí drag-and-drop do Databricks

úložiště nahrát, nicméně dále s nimi není možná žádná další manipulace tímto způsobem. Maximální počet takto nahrávaných souborů je momentálně (květen 2024) 10 s omezením celkové velikosti na 2GB.

1.5.2 Práce s clustery

Clustery v kontextu Databricks jsou skupinou virtuálních strojů nakonfigurovaných na Spark, resp. PySpark a nabízí možnost konfigurace po stránce HW (hardware), SW (software), i po stránce samotného módu spuštění. [11] Při špatném nastavení se lze setkat s vysokým poplatkem za běh clusteru i v nečinnosti (více k poplatkům v kapitole 1.5.3). To často vede k motivaci vypínat běžící clustery co nejdříve, což způsobuje zdlouhavé čekání při potřebě nový cluster nastartovat. Jakýkoliv běh kódu vyžaduje spuštěný cluster. Ten se do činnosti uvádí občas i několik minut a způsobuje tak zpomalení programátora, který musí počkat na zahájení spuštění kódu. Jedinou výjimkou je tzv. *serverless SQL warehouse*, který se spouští vždy pouze několik vteřin. Jeho použití je však vázáno na několik požadavků, jako je například placená verze Databricks Premium plan a vyšší, nebo podpora tohoto typu výpočetní jednotky pro daný region.

1.5.3 Cena

Poplatky za využití SaaS v cloudu jejich poskytovatelé uvádí jako jednu z výhod použití. Snížené nároky na vlastní zajištění datových center a jejich údržbu jsou výhodné především z hlediska flexibility a počáteční ceny, kdy by pro klasické in-house řešení bylo potřeba zakoupit veškerý hardware a nastavit ho pro účely datového skladu. To je vyváženo vyšší cenou za využívání clusterů, v jejichž ceně se promítá spotřeba, opotřebení fyzických komponent a marže.

Databricks pro výpočet výsledné ceny využívá DBU (Databricks Unit): [12]

$$dbu\_spotřeba * dbu\_hodnota = výsledná\_cena$$

Hodnota DBU se pohybuje mezi $0.08 a $0.50 v závislosti na několika faktorech, jako je typ clusteru, typ využívané edice Databricks, poskytovatel cloudu a na regionu. Následně nabízí také slevy pro pravidelné uživatele, kteří se zavážou k využití konkrétního množství prostředků na následující období.

Spotřeba DBU je závislá na objemu zpracovávaných dat, obtížnosti použitých algoritmů a četnosti běhů. I když nejsou zrovna zpracovávána žádná data, běžící cluster spotřebovává základní sazbu, je však možné jej nastavit, aby se při neaktivitě po konkrétní době sám vypnul a zabránit tím zbytečným poplatkům navíc.

Celkové náklady se skládají z nákladů za provoz clusterů na Databricks a z poplatků poskytovatele cloudových služeb (viz. kapitola 1.4.2).

Cena samotných clusterů na Databricks se díky její široké nabídce velmi liší a pro standartní plán se cena clusterů pro libovolné využití pohybuje od 0.16$/hodinu (nejmenší m4.large s 2x vCPUs a 8GB pamětí) do 9.728$/hodinu (největší m6in.32xlarge s 128x vCPUs a 512GB pamětí). [13] Kromě těchto univerzálních tarifů je možné vybírat i z instancí optimalizovaných pro paměť, úložiště, výpočet, nebo pro využití GPU, které mohou být pro daný typ práce optimalizovanější a jejich cena kvůli specifickému zaměření nižší.

Kapitola 2

Možné přístupy tvorby stage vrstvy datového skladu

Tato kapitola představuje obecné přístupy zahrnující inicializaci prostředí a integraci souborů a porovnává mezi sebou metody klasického vývoje (s pomocí databází, on-premise DWH) s možnostmi analytické platformy Databricks (cloudové řešení DWH). Hlavním cílem je zvýraznit hlavní rozdíly mezi jednotlivými způsoby práce s velkým objemem dat a poskytnout vhled do práce s platformou Databricks jako takovou, aby bylo možné dané znalosti využít v realizaci stage vrstvy v kapitole 4. Nejdříve je v této kapitole obecně vysvětlen pojem stage vrstva, poté se podkapitoly věnují klasickému vývoji a následně přístupu v Databricks, které se často odkazují na zbytek textu v této kapitole a přímo tyto dva přístupy porovnávají.

2.1 Problematika DWH

Vývoj datového skladu je složitá a nákladná záležitost. Je v mnoha ohledech podobný vývoji jakéhokoli softwaru a vyžaduje definici různých činností, které se týkají shromažďování požadavků, návrhu a implementace na cílové platformě. K tomuto vývoji však chybí metodický rámec, který by vývojáře vedl v různých fázích procesu vývoje datového skladu. Tato situace vyplývá ze skutečnosti, že potřeba budovat systémy datových skladů vznikla dříve, než byly definovány formální přístupy k vývoji datových skladů, stejně jako tomu bylo v případě operačních databází. [14]

Mezi hlavní výzvy, které zahrnuje vytvoření a správa datového skladu patří *složitost dat* a jejich různé formáty, *integrace dat* z různých zdrojů, *zajištění bezpečnosti dat*, *výkon a škálovatelnost* pro neustále narůstající objemy dat, *agilní vývoj a inovace*. Především v dnešním dynamickém prostředí je důležitá i *dokumentace a správa metadat*, nezbytná pro správné porozumění struktury dat, předávání znalostí dalším vývojářům a důvěryhodnost v analýzy prováděné nad daty z datového skladu. Zvládnutí všech výzev, se kterými se vývojáři při práci na datovém skladu setkají, vyžaduje zvládnutí technologických nástrojů, organizačních postupů a strategií jednotlivých procesů.

ETL z anglického „extract, transform, load“, tedy *extrakce* (vyjmutí dat z původního zdroje), *transformace* a *načtení* (do datového skladu) je jedním z hlavních používaných konceptů v datových skladech. Od dalšího často používaného konceptu ELT (extract, load, transform) se liší především pořadím, ve kterém jsou jednotlivé operace prováděny.

2.2 Metadata

Motivaci k vytvoření datového skladu často přispívá snaha o hlubší porozumění datům, která jsou organizacím dostupná. Toto porozumění je možné z části hlavně proto, že sbíraná data, resp. každý jednotlivý datový bod, vizualizovatelný jako záznam v tabulce, reprezentuje sadu konkrétních zaznamenaných hodnot. Taková data, o kterých můžeme říct, že v sobě enkapsulují další data, nazýváme metadata.

▶ **Definice 2.1** (Metadata)**.** *Data, poskytující informace o jiných datech. [15]*

Tato definice může působit neúplným nebo holým dojmem a proto jsou často uváděny další alternativy definice metadat pro lepší porozumění.

▶ **Definice 2.2** (Metadata, alternativní)**.** *Informace o objektu, ať už fyzickém nebo digitálním. [16]*

Tato definice na rozdíl od 2.1 evokuje v čtenáři konkrétnější představu a poskytuje širší kontext. Metadata lze také chápat jako popisná data, zachycující různé atributy a charakteristiky datových záznamů, která nám následně pomáhají pochopit chování zkoumaného objektu, jeho změny v čase, nebo jeho podstatu.

2.2.1 Metadata obecně

Využití metadat je možné nalézt v širokém spektru odvětví. Používají se v knihovnictví, galeriích, marketingu, při zpracování obrazu a videa, ve vědeckých výzkumech, finančních službách, bankovnictví a také například v průmyslové výrobě. Jejich základní rozdělení je na data *popisná*, *technická*, *uchovávací*, *právní*, *strukturální* a na *označovací jazyky* (*Markup Languages*). [17] Typickými způsoby ukládání a sdílení jsou relační databáze, XML (eXtensible Markup Language) a propojená data (linked data). Nejčastějším způsobem jsou relační databáze, ve kterých je možné pracovat s většinou typů metadat.

Pro zisk maximální hodnoty informací z metadat je nutné jim porozumět, což zahrnuje konstantní správu ukládaných dat. Navzdory snaze standardizovat formáty a konvenci ukládaných dat se však stále jedná o něco, co si organizace musí vynutit interně samy. V kontrolovaném prostředí se mohou zavést slovníky předdefinovaných hodnot, které jsou povolené, nastavit standardy pro značení desetinných míst (čárkou, tečkou, nebo jiným znakem), pravidla pro malá a velká písmena (např. slova musí začínat velkým písmenem, nebo musí být všechna písmena velká) a sjednocení zkratek, tedy zda slovo „výzkum“ má být psáno v celé délce, jako zkrácené „výzk.“, nebo dokonce „výz.“. Není pravidlem, že by organizace s využitím dat počítala od začátku jejich sběru. Některá data jsou již od počátku koncipována pouze jako záznam o jisté události s cílem mít možnost tuto událost v budoucnu vyhledat bez nutnosti chápat jí v kontextu s ostatními zaznamenanými událostmi. Ukládání dat je samo o sobě z finančního a časového hlediska nenáročné, a při malém počtu dat není nutné vynucování dodržení konkrétního formátu nebo struktury při jejich ukládání. V počáteční fázi ukládání buď není relevantní snažit se získat znalosti z takto malého vzorku dat, nebo je možné tyto znalosti získat pouhým pohledem lidského oka na datovou sadu. Problém nastane, když množství dat znatelně naroste a zvýší se i zájem data pochopit a odhalit možné skryté souvislosti. V takových případech se nabízí kompletní invalidace již uložených dat, konzultace definic a standardů formátu pro nově ukládaná data se systémovými návrháři a následné vynucení těchto pravidel pro nově sbíraná data. To by však znamenalo zahození velkého množství dat, která mohou potenciálně skrývat důležité informace. Častěji je tedy volena druhá varianta, využití datových skladů ke zpracování již uložených metadat a jejich následnému porozumění.

2.2.2 Metadata v datových skladech

Pro možnost analýzy dat jako celku v datovém skladu je pro vývojáře důležité porozumět původním datům, jejich struktuře, hodnotám, vztahům mezi různými atributy a pokud je to možné, i obecně kontextu, ve kterém byla data sbírána. Následně data očistit o nevhodné záznamy (chybné nebo chybějící hodnoty, duplikáty), vyřešit nekonzistence a transformovat data do vhodného jednotného formátu, který je snadno analyzovatelný. Transformace mohou obsahovat normalizaci, standardizaci formátů nebo rozdělení obsahu jednoho atributu do více. Takto zpracovaná data jsou lehčí na porozumění a využití k analýze, nebo jiným koncovým procesům.

Efektivní a spolehlivé využívání možností datového skladu je podmíněno dodržením několika následujících bodů:

- **Konzistence** a **dokumentace** pro efektivní pochopení dat a jejich využívání.
- **Sdílené úložiště** umožňující různým skupinám uživatelů přistupovat k metadatům.
- **Struktura** a **navigace** navržená tak, aby vyhovovala potřebám využívaných nástrojů a uživatelům.
- **Metamodel** popisující strukturu úložiště, typy prvků a jejich vztahy.

Tyto body v jisté míře zachycují standardy pro metadatově zaměřené datové sklady. Standardy, které se mohou stát rozhodujícím faktorem, odlišujícím úspěch a neúspěch při nasazení a správě datového skladu v závislosti na jejich dodržování nebo nedodržování. Dva významné standardy pro reprezentaci a výměnu metadat jsou metamodely OIM (Open Information Model) a CWM (Common Warehouse Metamodel). [18]

OIM se zaměřuje na podporu vzájemného propojení mezi různými komponentami (databáze, datové sklady, datová jezera, nástroje pro ETL). Vychází z abstraktních konceptů UML (Unified Modeling Language), které aplikuje na modely jednotlivých oblastí. Dochází tedy k rozdělení celku na menší části dle jejich významu, což následně způsobí, že tyto modely popisují specifická metadata. Základní dělení, ke kterému v rámci OIM dochází, je na *Analysis and Design Model*, *Business Engineering Model*, *Object and Component Model*, *Knowledge Management Model* a *Database and Warehousing Model*.

CWM cílí na umožnění snadné výměny společných metadat mezi nástroji v datovém skladu a jejich úložištěm. Je postaven na kombinaci standardů XMI (XML Metadata Interchange), UML a MOF (Meta Object Facility). CWM je metamodel tvořený z vrstev. Každá vrstva je tvořena z několika součástí a vrstvy jsou postaveny tak, aby vrstvy na vyšších úrovních byly závislé na vrstvách nižší úrovně, ale není zde žádná závislost mezi komponentami jedné vrstvy. Při postupu od nejnižší vrstvy po nejvyšší je jejich výčet následující: *Object Model Layer*, *Foundation Layer*, *Resource Layer*, *Analysis Layer* a *Management Layer*. [19]

OIM a CWM představují dva významné standardy v oblasti správy metadat, přičemž každý z nich má své vlastní charakteristiky a zaměření. Standard OIM je navržen tak, aby zahrnoval všechny fáze vývoje informačních systémů a podporoval různé výpočetní technologie. Na druhou stranu, CWM poskytuje rámec pro reprezentaci společných metadat skladu a je zaměřen na výměnu metadat mezi analytickými nástroji. OIM má širší rozsah a víc se zaměřuje na popis informací, zatímco CWM se soustředí na úložiště dat. Oba standardy používají jazyk UML, ale CWM je narozdíl od OIM v souladu s MOF. Výměna metadat probíhá u OIM s pomocí XML a vlastního enkódování, zatímco CWM využívá rozšířenou verzi XMI, standard pro výměnu metadatových informací prostřednictvím XML. [20]

Je zjevné, že správné porozumění a efektivní manipulace s metadaty jsou klíčové pro úspěšnou analýzu dat v datových skladech. Metadatově zaměřené standardy hrají klíčovou roli při zachycování a správě metadat. Jejich dodržování je nezbytné pro dosažení efektivity a spolehlivosti při využívání datových skladů pro analytické procesy.

2.3 Historizační metody v DWH

Z potřeby zachovat a analyzovat historická data v datových skladech byly definovány metody pro efektivní implementaci těchto procesů. Mezi další motivace může patřit dodržování některých předpisů a regulací, obnova stavu dat (pokud to daná metoda umožňuje) do konkrétního okamžiku v minulosti, nebo sledování vývoje trendu dat v průběhu času.

Nejvíce známé a používané metody jsou následující: [21]

- **Snapshotting** ukládá každou změnu v datech jako samostatnou verzi, tzv. „snapshot“. Tato technika je používána pro pravidelné pořizování snímků dat. Výhodou tohoto přístupu je jednoduchost při archivování a mazání starších dat, přehlednost a snadná realizace zapojení do ETL procesů datového skladu.
- **Slowly Changing Dimension** je metoda určená především pro data, jejichž charakter se mění velmi pomalu. Má několik typů (verzí), které je dále možné rozdělit na základní typy (nesoucí číselné označení 0-4) a hybridní techniky (označené číselně 5-7). Pro tuto metodu je v praxi používán akronym SCD.
  - **Typ 2** (nazýván také jako SCD2) je nejvíce používaným typem SCD. Jeho funkcionalita spočívá v přidání čtyřech nových sloupců: unikátní identifikátor sloužící k rozpoznání stejných záznamů napříč různými verzemi, časový sloupec s časovým razítkem začátku platnosti konkrétního záznamu, časový sloupec pro konec platnosti a binární sloupec označující jeho stav v danou chvíli (aktuální, neaktuální). Se změnou některého atributu stávajícího záznamu je vytvořen nový záznam v tabulce se stejným identifikátorem a starý záznam je invalidován, ale zůstává přítomen pro případné budoucí analýzy.

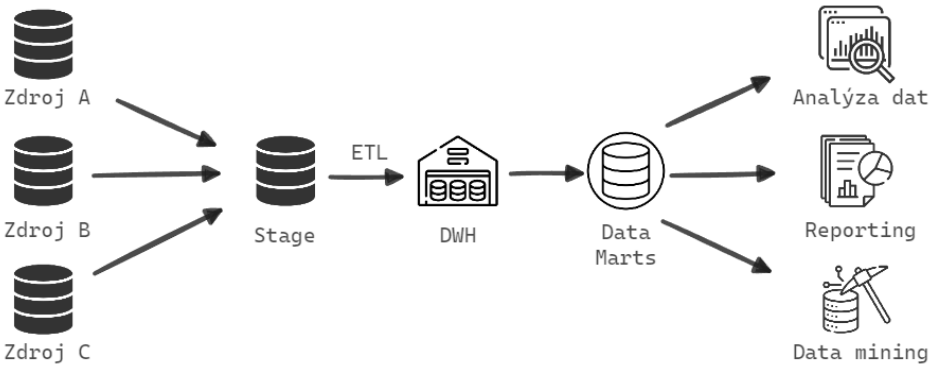
Hlavním rozdílem mezi oběma metodami je způsob ukládání změn v datech. SCD2 je více náročný na údržbu a nastavení a může mít potencionální dopady na výkon u datasetů s velkými rozměry. Snapshotting je naopak jednoduché nastavit a udržovat, nicméně ukládá velké objemy dat, což může vést k vyšším nákladům na úložiště a zesložitění analýzy historie změn dat. Zjednodušeně je možné každou metodu doporučit na základě několika rozhodovacích faktorů. Pro Snapshotting to jsou jednoduchost, rychlost a kompletní historie. Pro SCD2 jsou hlavními faktory sledování změn v datech, efektivní správa dat a komplexní analytické požadavky.

2.4 Upřesnění pojmu „stage vrstva“

„Stage vrstvu lze definovat jako úložiště a soubor procesů, které pročišťují, transformují, kombinují, zbavují se duplikací, spravují, archivují a připravují zdrojová data pro použití v datovém skladu.“ [22]

Jedná se o vrstvu nacházející se mezi zdrojem dat a hlavním jádrem datového skladu (to je znázorněno na obrázku 2.1 pod názvem „Stage“). Při načítání data ze zdroje prochází právě touto vrstvou, kde dochází ke kontrolám kvality dat, transformacím a vzniká přidaná hodnota celého skladu. Spoustu z těchto úprav má smysl provádět nad daty pouze jednou, při jejich načtení, například namapování na klíče. Tyto úpravy podporuje široké množství nástrojů pro práci s relačními databázemi. [22] Typický příklad použití je v případech, kdy musí být spojována data ze dvou různých zdrojů, které není možné spojovat ve stejný čas. Nejprve se tedy načtou data ze všech zdrojů odděleně, a až teprve ve stage vrstvě se spojí. Dalším příkladem použití je nutnost předzpracování načítaných dat. Posledním případem je nutnost rozdělení dat do několika různě velkých částí tak, aby byla umožněna paralizace ETL procesů. [24]

Základní vlastností rozlišující stage vrstvu od ostatních vrstev je chybějící možnost se v této části vývoje nad daty jakkoliv dotazovat. Vzhledem ke způsobu, jakým jsou data ukládána, postrádá taková možnost smysl a pokud by přece jen dotazování nad daty bylo umožněno, již by se tato vrstva dala řadit mezi prezentační prostředky, kam se typicky řadí jádro datového



Obrázek 2.1 Diagram ilustrující proud dat od původních zdrojů až ke koncové analýze, reportingu a data miningu. Obrázek vytvořen autorem s pomocí nástrojů [4][5]. Inspirace: [23].

skladu (DWH) a datová tržiště (data marts). Při dodržení této myšlenky jsou veškeré procesy včetně neupravených dat schované před koncovými uživateli a vnějším světem až do chvíle, než jsou data upravena dle potřeb a prezentována ve své finální formě ve vhodné vrstvě.

2.5 Metody klasického vývoje

Pro tuto práci se za klasický vývoj považuje především tvorba on-premise datového skladu. Před započetím vývoje je důrazně doporučeno vytvořit vysokoúrovňový plán implementace. Ten spočívá v získání relevantních informací pro daný projekt. Nejprve je vhodné porovnat, zda by dané transformační procesy bylo lepší provádět dříve nebo později. Následuje zhodnocení největších výzev, které se při implementaci očekávají a rozhodnutí se nad nástroji, které budou používány. Na závěr je na řadě zhodnocení efektivity použitých nástrojů. Při rozvržení tohoto plánu implementace je možné zamezit nečekaným změnám během vývoje.

Klasický vývoj dodržuje ETL, tedy postupně extrahuje data ze zdroje, transformuje je, a nahrává je do DWH, tedy vybraného cílového úložiště. Transformace nejsou limitovány pouze na stage vrstvu. Mohou se objevit i ve vrstvách předcházejících nebo následujících.

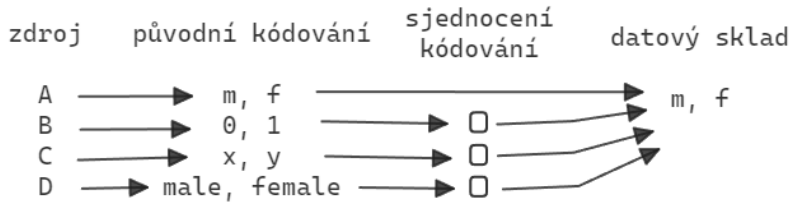
2.5.1 Inicializace prostředí

S prvními kroky vývoje datového skladu je vhodné nejprve vybrat technologie, poté připravit samotné vývojové prostředí, což zahrnuje vytvoření virtuálních serverů, obstarání softwaru (nákup licencí, instalace) potřebného pro vývoj a nastavení potřebné konfigurace. Na virtuálních serverech budou nainstalované nástroje, se kterými mohou vývojáři pracovat. Jmenovitě databázový systém, ETL nástroj, orchestrační nástroj a nástroj na vizualizaci dat. Následně je třeba vytvořit v databázi tabulky pro Stage vrstvu a volitelně také pro PreStage vrstvu. Je nutné vytvořit SQL skripty, které databázi a tabulky vytvoří. Nastavit export zdrojových dat do Stage, případně PreStage, vytvořit ETL procesy a orchestraci procesů pro Stage vrstvu a následně celou funkčnost otestovat. [25]

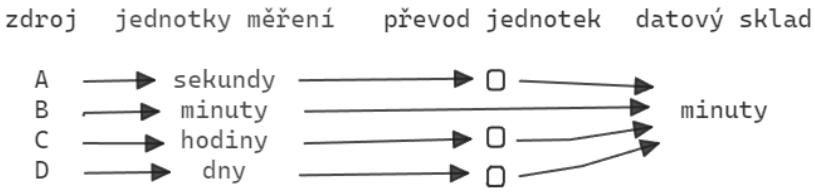
2.5.2 Integrace datových zdrojů

„Ze všech aspektů datového skladu je integrace nejdůležitější. Data jsou do datového skladu přiváděna z různých zdrojů. Jakmile jsou umístěna v datovém skladu, mají jedinou fyzickou podobu.“ [26]

V minulosti nebylo zvykem, že by data musela být někdy integrována s jinými daty. V důsledku toho napříč více aplikacemi neexistuje aplikační konzistence v kódování, konvencích pojmenování, fyzických atributech a způsobem měření atributů. Každý návrhář aplikace měl vždy volnou ruku při rozhodování o vlastním návrhu. Výsledkem je, že se vzájemně velmi liší a při integraci by mělo dojít k vkládání dat tak, aby se zároveň odstranily mnohé vzájemné nekonzistence na aplikační úrovni.



Obrázek 2.2 Nekonzistence v kódování stejné informace se řeší ve stage vrstvě před vstupem dat do datového skladu. Obrázek vytvořen autorem s pomocí nástrojů [4][5]. Inspirace: [26].



Obrázek 2.3 Data mohou být měřena v různých jednotkách. Hodnoty je nutné převést dle nové jednotky, aby vzájemné hodnoty byly jednoduše porovnatelné bez dalších převodů. Obrázek vytvořen autorem s pomocí nástrojů [4][5]. Inspirace: [26].

Bez ohledu na metodu nebo zdrojovou aplikaci je kódování skladu prováděno konzistentně. To samé platí také pro konvence pojmenování, jednotky atributů (u vzdálenosti např. cm, km, míle, palce) a strukturu klíčů (případně řešení vzájemných konfliktů). Výsledkem je schopnost poskytnout ucelený a provázaný pohled na data, která se v DWH nachází.

Integrace neprobíhá pouze jednou. Pojem *Warehouse Refreshment* popisuje důležitost aktuálních, respektive co možná nejaktuálnějších, dat v datovém skladu. Důležitosti aktuálnosti dat je věnována podkapitola v knize *Fundamentals of Data Warehouses*. [27] V této knize je uveden konkrétní příklad z reálného světa, kdy spousta dodavatelů WalMartu, nejúspěšnějšího prodejce (v roce 2013, dle [27]), má přímý přístup do DWH WalMartu a zasílá do konkrétních prodejen své produkty jakmile zjistí, že hrozí jejich vyprodání na konkrétních kamenných prodejnách. Tento přístup k aktuálním datům šetří oběma stranám nutné dohadování dodávek a přizpůsobuje se aktuální situaci. V tomto případě je důležité mít data aktuální, aby dodavatelé vždy věděli, po kterém zboží je v daný moment nejvyšší poptávka a flexibilně se mohli přizpůsobit a včas adekvátně zareagovat. V závislosti na využití lze rozlišit mezi aktualizacemi, které probíhají v rámci vteřin, hodin, nebo dnů. Platí, že čím častější aktualizace jsou, tím méně náročné transformace by měly nad daty být spouštěny.

2.6 Datový sklad v Databricks

Prostředí Databricks umožňuje nakonfigurovat pipeline pro veškeré procesy od načtení zdrojových dat, jejich zpracování a výsledné dotazování se nad takovými daty. Kromě tohoto tradičního způsobu je možné využít DLT framework. Ten si dává za cíl odebrat vývojářům nutnost řídit

úlohy, správu clusterů, monitorování, hlídání kvality dat a ošetřování chyb. V takovém případě stačí pouze definovat konkrétní transformace, které mají na datech proběhnout, volitelně i testy datové kvality, kterými mají transformovaná data být zkontrolována a framework DLT se postará o vše ostatní. Pro transformace používá pattern ETL. Speciálně pro práci s metadaty vznikl v rámci Databricks Labs framework DLT-META, usnadňující práci s metadaty v rámci DLT. Tento framework není však formálně podporován a je poskytován „as is“, tedy ve stavu bez garance jakýchkoliv úprav nebo vylepšení do budoucna.

2.6.1 Nastavení prostředí

Oba přístupy (Databricks Workflows, vyžadující vlastní nastavení ETL procesů a Databricks s využitím DLT, který se o orchestraci ETL procesů stará sám) se liší v konkrétní konfiguraci prostředí a pipelines na zpracování tabulek. Při základním přístupu k vývoji na Databricks je potřeba nastavit cluster, vytvořit skripty (například ve formě jednotlivých notebooků), nastavit pipeline (propojení se skripty, vzájemné závislosti, exekuční plán jednotlivých skriptů, připojit cluster). Uživatelské rozhraní je přehledné a jakmile se uživatel zorientuje v umístění jednotlivých součástí (clustery pod záložkou „Compute“, pipeline jako „job“ pod záložkou „Workflows“), není těžké vše nastavit. Co může působit větší obtíže je pochopení úložiště, kam data vlastně ukládat a odkud se na ně dotazovat. K tomu může alespoň trochu být nápomocné oficiální demo, [28] které je součástí dokumentace, ukazuje tvoření pipeline na konkrétním příkladu a poskytuje podrobný návod.

Naproti tomu využití DLT vyžaduje pouze vytvoření kódu, nad každou definici funkce vložení dekorátoru @dlt s akcí, které chce vývojář docílit (*@dlt.create_table* pro vytvoření tabulky, *@dlt.create_view* pro vytvoření pohledu), testy kvality a typ akce v případě, že test neprojde (*@dlt.expect* pro očekávané hodnoty, které však v případě nedodržení není nutné nijak omezovat a jsou zahrnuty do výsledků mezi validní hodnoty, *@dlt.expect_or_drop* pro hodnoty, které v případě nesplnění testu nejsou do výsledků zahrnuty a *@dlt.expect_or_fail*, který zastaví pipeline v případě, že se vyskytne neočekávaná hodnota). Není nutné nastavovat pořadí kódu, v jakém se bude spouštět. Framework se dle závislostí v kódu sám rozhodne o podobě exekučního plánu. V případě menší změny v kódu je možné vybrat jednotlivé tabulky, které se mají znovu načíst, není potřeba spouštět celý proces znovu.

Pro lepší ilustraci rozdílů mezi použitím čistého PySparku a využitím DLT je přiložena krátká ukázka kódu, která zahrnuje načtení ukázkových dat z dostupného demo datasetu Databricks do tabulky *sales_orders_raw* a následně jednoduchou transformaci s vytvořením tabulky *sales_orders_in_la*. Výpis kódu 2.1 zachycuje postup bez použití DLT a výpis 2.2 s použitím DLT. Cíl i výsledek obou ukázek kódu je stejný, ale přístupy se liší.

Rozdíl je také ve způsobu využití clusterů. Zatímco u manuálního vytváření pipeline je možné spustit vlastní cluster, nechat ho běžet a dle potřeby na něm spouštět části kódu (tedy mít ho v interaktivním režimu režimu jako all-purpose cluster), u DLT se vždy vytvoří cluster pouze pro tuto specifickou úlohu, který po typicky konci běhu zaniká (Job cluster), což vede ke zdržení při častějším spouštění (viz. podkapitola o práci s clustery 1.5.2).

Této nutnosti znovu startovat výpočetní jednotky po skončení úlohy je možné se v jisté míře vyhnout přenastavením u konkrétní DLT úlohy. V nastavení je možné přepínat mezi režimem vývoje *„Development“* a produkčním režimem *„Production“*. Ve vývojářském režimu clustery zůstávají po skončení úlohy zapnuté (ve výchozím nastavení dvě hodiny po jejich spuštění). V produkčním módu dochází po úspěšném skončení spuštěné úlohy k vypnutí všech clusterů. Pokud úloha není úspěšně dokončena, může dojít k automatickému restartu. Takové chování nastává v případech, kdy dojde k některým specifickým chybám, například úniku paměti, nebo při neúspěšném spuštění clusteru.

Výpis kódu 2.1 Vytvoření tabulky sales_orders_raw v prvním notebooku a následné vytvoření tabulky sales_orders_in_la v druhém notebooku po odfiltrování objednávek s chybějícím číslem objednávky a objednávek mimo Los Angeles.

```
# Kod je rozprostren do dvou odlisnych notebooku.
# Poradi spousteni je nutne nastavit rucne v pipeline.
# Notebook 1
def sales_orders_raw():
  return (spark.readStream
        .format("cloudFiles")
        .option("cloudFiles.schemaLocation", "/tmp/sales_schema")
        .option("cloudFiles.format", "json")
        .option("cloudFiles.inferColumnTypes", "true")
        .load("/databricks-datasets/retail-org/sales_orders/")
        .writeStream
        .toTable("sales_orders_raw")

# Notebook 2
def sales_order_in_la():
    df = spark.table("sales_orders_raw")
    return df.filter(df("order_number").isNotNull())
             .where("city == 'Los Angeles'")
             .select(df.city, df.order_date, df.customer_name)
             .write
             .format("parquet")
             .saveAsTable("sales_orders_in_la")
```

Výpis kódu 2.2 Vytváření tabulek s pomocí DLT s použitím @dlt.expect_or_drop pro kontrolu kvality a odstranění záznamů s hodnotami nesplňujícími zadané podmínky.

```
# Veskery kod je v jednom notebooku.
# DLT si samo udela exekucni plan a vytvori tabulky.
import dlt

@dlt.create_table
def sales_orders_raw():
  return (spark.readStream
    .format("cloudFiles")
    .option("cloudFiles.schemaLocation", "/tmp/sales_schema")
    .option("cloudFiles.format", "json")
    .option("cloudFiles.inferColumnTypes", "true")
    .load("/databricks-datasets/retail-org/sales_orders/"))

# dlt.expect_or_drop zde odstrani radky kde je order_number NULL.
# Ve vysledne tabulce se tyto zaznamy neobjevi.
@dlt.create_table(comment="Filtered sales orders in LA")
@dlt.expect_or_drop("valid order_number", "order_number IS NOT NULL")
def sales_orders_in_la():
  return dlt.read_stream("sales_orders_raw")
          .where("city == 'Los Angeles'")
          .select(df.city, df.order_date, df.customer_name)
```

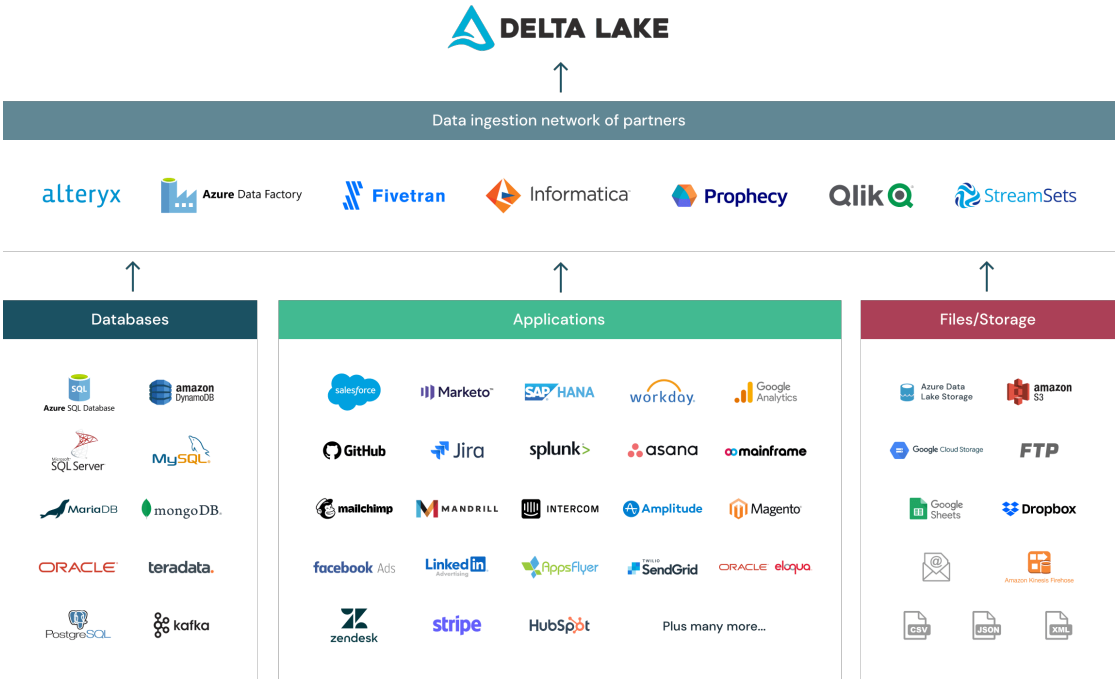
2.6.2 Integrace souborů

Soubory v rámci Databricks jsou uloženy ve vrstvě Delta Lake, která ukládá data ve formátu parquet na úložištích jako je HDFS, AWS S3 nebo Azure Data Lake Storage.

Soubory lze do databricks nahrát napřímo (v případě menších souborů) s využitím drag-and-drop, využít Apache Spark API na připojení ke cloudovým a on-premises zdrojům dat a načíst data z databází a podobných zdrojů. To umožňuje načítat data z lokálních databází, Kafky, lokálně hostovaného NFS, v případě doinstalování dalších knihoven lze data načíst i z Azure SQL databází.

Dalším způsobem je využití samotného Databricks API. To zahrnuje příkaz `COPY INTO`, který dovoluje inkrementálně načítat nové soubory. Následně byl představen Auto Loader, který funkčnost `COPY INTO` rozvíjí a přeměňuje příkaz na průběžný proces načítání a aktualizace dat, jakmile se objeví v zadané složce. Jde tedy vpodstatě o zjednodušení komplexnosti pipeline a snížení latence. Obě možnosti při načítání přeskakují soubory, které již byly načtené v minulosti a nedošlo u nich k žádným změnám.

Posledním způsobem je využití tzv. Data Integration Partners. Tato možnost je nejjednodušším přístupem v případě, kdy jsou data již součástí některého z partnerů Databricks, kteří mají implementované propojení a umožňují tak načtení a aktualizování změn bez nutnosti složitého kódu. Mezi hlavní partnery Databricks v době vzniku této práce (květen 2024) patří například Azure Data Factory, Fivetran, Informatica, Prophecy a Qlik. [29] Možná propojení, která jsou v Databricks k dispozici, ukazuje obrázek 2.4.



Obrázek 2.4 Příklad nejznámějších databází, aplikací, úložišť a typů souborů, které jsou součástí Databricks Data Integration Partners. [29]

2.7 Porovnání obou přístupů

Integrace dat z různých zdrojů do on-premise datového skladu (klasický přístup) je často značně komplexní a náročná úloha. Tento proces vyžaduje vytvoření a správu sofistikovaných ETL pro-

cesů. Tyto operace, zvláště v případě starších systémů nebo heterogenních datových prostředí (tedy takových, kde jsou data uložena a spravována pomocí různorodých technologií), mohou být obtížné na implementaci a údržbu. Kromě toho může dosažení konektivity být problematické, zejména v případech, kdy je potřeba integrovat se s uzavřenými systémy nebo takovými, které mají nedostatečně zdokumentovanou strukturu.

Naopak cloudové datové platformy, jako je Databricks, obvykle nabízejí rozšířené možnosti integrace dat a konektivity. Poskytují vestavěné konektory a API rozhraní, které umožňují snadnou integraci s různými datovými zdroji. To zjednodušuje proces integrace dat a často zkracuje dobu potřebnou pro implementaci. Kromě toho, cloudové platformy poskytují nástroje pro streamování dat, což umožňuje efektivní a rychlé zpracování dat v reálném čase. Celkově se tedy jeví řešení s pomocí cloudových datových skladů jako více optimální, modernější, a pro budoucí potřeby agilnější.

Kromě těchto rozdílů porovnávajících především integraci dat a nastavení samotné stage vrstvy je mezi klasickým přístupem a cloudovými platformami obecně spousta dalších rozdílů, z nichž velká část již byla rozebrána v kapitole 1 (jmenovitě provozní náklady, infrastruktura, škálovatelnost, správa a výkon).

Kapitola 3
Framework dbt

Tato kapitola představuje framework dbt (data build tool) a zkoumá jeho klíčové vlastnosti, možnosti využití v cloudovém DWH a důvody jeho stále rostoucí popularity v oblasti transformace dat v datových skladech. V úvodu kapitoly je rozebírán popis základních principů a funkcionality, postupně jsou poté rozebrány výhody použití dbt, jeho rostoucí popularita, rozdíly mezi verzemi dbt Cloud a dbt Core, přičemž je kladen důraz na jejich hlavní funkcionality a omezení. Jsou diskutovány vlastnosti obou verzí a strategie pro výběr správné verze podle potřeb konkrétního projektu. V samotném závěru kapitoly je dbt začleněn do kontextu platformy Databricks. Jsou diskutovány možnosti integrace dbt s Databricks a jeho role v celém ekosystému datového skladu.

3.1 Představení

Podobně jako Databricks kombinuje DWH a Data Lakes do platformy sjednocující výhody každé z nich (viz. obrázek 1.3), dbt se snaží o to samé a zavádí pojem inženýr analytik (Analytics Engineer), což je pojem spojovaný často právě a pouze s dbt a je na něj častěji nahlíženo spíše jako obecné označení pro uživatele tohoto frameworku.

Pro lepší srozumitelnost, co vlastně Analytics Engineer má mít jako pracovní náplň, je důležité vzájemné porovnání s pojmy, ze kterých vznikl, tedy spojením Data Engineer a Data Analyst. V knize *Unlocking dbt* autoři pro tyto účely používají následující popis: „*Datový analytik je obvykle vnímán jako odborník, jehož úkolem je sbírat a interpretovat data, vytvářet reporty a často využívat SQL dotazy ke zlepšení porozumění a modelování dat s cílem získat konkrétní znalosti. Naopak datový inženýr se spoléhá na nástroje navržené speciálně pro softwarové inženýrství, které automaticky aplikují osvědčené postupy.*“ [30]

Spojení obou rolí do jedné by mělo z každé role brát něco a Analytics Engineer by tak měl poskytovat očištěné, transformované a zdokumentované datové sady, mít k dispozici nástroje pro modelování dat, a mít možnost aplikovat osvědčené postupy softwarového inženýrství. Porovnání všech tří rolí je v bodech vypsáno i v tabulce 3.1.

Dbt, open source nástroj pro transformaci dat je od jeho vzniku v roce 2016 využíván při transformaci velkého množství dat jak v architektuře ETL, tak i ELT. Sám o sobě nezajišťuje načítání ani extrakci dat a neposkytuje mechanismus na samostatné výpočty. Při spuštění se napojí na výpočetní jednotku konkrétního DWH nebo databáze, které je součástí. Podporuje primárně zpracování dat po dávkách (*batch*). Do dbt je zapojen i nástroj pro vývoj šablon Jinja, který zde slouží jako dynamický generátor SQL kódu v závislosti na předdefinovaných šablonách a umožňuje využití maker. Ta jsou srovnatelná s funkcemi v objektově orientovaných a funkcionálních programovacích jazycích. I když na první pohled může jít o překážku v podobě nutnosti naučit se v rámci práce s dbt i s syntaxí Jinja, základní použití je velmi snadné a i v

Tabulka 3.1 Porovnání pojmů Data Engineer, Analytics Engineer a Data Analyst. [31]

Data Engineer	Analytics Engineer	Data Analyst
▪ Sestavuje vlastní datové integrace ▪ Spravuje organizaci pipelines ▪ Vyvíjí a nasazuje koncové body strojového učení ▪ Vytváří datovou platformu a zajišťuje její běh ▪ Optimalizuje výkon datového skladu	▪ Poskytuje čistá, transformovaná data připravená k analýze ▪ Aplikuje osvědčené postupy softwarového inženýrství (version control, testování, kontinuální integrace) ▪ Udržuje dokumentaci a definici dat	▪ Pracuje s hlubokými poznatky (např. „proč se minulý měsíc zvýšil odliv zákazníků?“) ▪ Pracuje s obchodem a uživateli, aby pochopil požadavky na data ▪ Vytváří dashboardy a prognózy ▪ Předává znalosti jak používat data vizualizačních nástrojů

případě komplexnějšího využití se nejedná o nic obtížného. Při práci s tímto frameworkem je dobré mít následující znalosti:

- **SQL**, především základní příkaz SELECT. Pokročilejší příkazy mají své využití především z hlediska efektivity, ale pro začátek je není nutné používat.
- **YAML** (YAML Ain't Markup Language), využívaný pro konfiguraci dbt projektu. Vzhledem k jeho lehké čitelnosti je velmi jednoduchý na pochopení a využití.
- **Python**, jehož podporu nabídlo dbt teprve během roku 2022, nabízí možnost nahrazení SQL v některých případech. Je možné, že se do budoucna stane esenciálním, prozatím však jeho znalost slouží spíše jako bonus navíc pro konkrétní případy využití, než jako žádoucí.
- **Data Modeling** slouží jako základ pro vytváření efektivních, spolehlivých a srozumitelných datových transformací a struktur. Předchozí znalosti v této oblasti vedou k efektivnějšímu rozvržení transformací a ke zlepšení kvality dat, což může vést k lepšímu porozumění datovým vzorcům, predikcím a rozhodování založenému na datech.
- **Source Control**, známý také jako **Version Control**, patří mezi osvědčené nástroje pro vývoj software v týmu a ukládání historie kódu. Dbt Cloud má integrován verzovací systém Git. Je tedy dobré znát pracovní postupy Gitu (workflow) a při vyvíjení je aktivně dodržovat.

Ve zkratce lze dbt prohlásit za nástroj pro transformaci dat, nejčastěji využíván pro modelování dat v datových skladech. Ve verzi dbt Core se navíc jedná o open source nástroj (pro dbt Cloud to však neplatí). Na pokročilejší úrovni je koncipován jako nástroj s development frameworkem, který spojuje modulární jazyk SQL s ověřenými postupy softwarového inženýrství. Tato kombinace umožňuje vytvářet efektivnější, rychlejší a spolehlivější transformace dat.

Hlavní omezení spočívá v tom, že toto řešení není určeno pro všechny dílčí části ETL (nebo případně ELT), nýbrž pouze pro transformační fázi. Někteří vývojáři preferují integraci jednoho nástroje, který by zahrnoval celý proces. V závislosti na podnikových procesech a kontextu organizace může být pro konkrétní projekty žádoucí mít k dispozici pouze jeden nástroj, který by zajistil pokrytí všech fází. [32]

3.2 Výhody

Vzhledem k velkému množství produktů na správu dat a tvorbě pipelines je dobré zmínit důvody, proč právě dbt stojí za povšimnutí.

Jedním z takových důvodů může být například poskytnutí řady funkcí, optimalizovaných pro transformace a analýzu. Jedná se o správu závislostí, inkrementální sestavování modelů, testování a další. Jeho základ stojí na SQL, což je jazyk široce rozšířený pro podobné úlohy, takže vyžaduje nízkou vstupní úroveň znalostí a nepředpokládá znalost jiných programovacích jazyků. Využití praktik modulárního SQL, tedy rozdělení kódu do menších, opakovaně použitelných částí, usnadňuje údržbu, testování a opakované využívání. Namísto hromadného přepisování SQL dotazů v případě změny ve vývoji je nutné ho přepsat pouze na jednom místě, pokud jsou správně využívány některé funkcionality dbt, v tomto případě funkce ref(), která dovoluje odkazovat se s pomocí proměnné na definici dotazu uloženou v jiném souboru. [33]

Výpis kódu 3.1 Ukázka použití funkce ref() v souboru model_b.sql pro odkázání se na select v souboru model_a.sql.

```
-- model_a.sql
        SELECT *
        FROM public.raw_data;

-- model_b.sql
        SELECT *
        FROM {{ref('model_a')}};
```

Jakožto prostředí zaměřené na spolupráci více lidí poskytuje integraci nástrojů pro verzování (GitLab, GitHub, Bitbucket, etc.) a automatickou tvorbu dokumentace s pomocí dokumentačních komentářů v kódu. Díky vysoké flexibilitě je možné ho využít při různých datových zdrojích a cílových databázích. S rostoucí popularitou se stal součástí nejznámějších cloudových DWH (Snowflake, Google BigQuery, Databricks, Amazon Redshift) a je možné ho v těchto datových skladech využívat nativně, tzv. „*out-of-the-box*“.

Vzhledem k tomu, že je jeho architektura primárně založena na SQL, není vyžadováno přenášení dat po síti, jak tomu bylo u některých starších nástrojů. Toto provedení se vyznačuje větší rychlostí a bezpečností. V souvislosti s tím dochází ke zpracování dat přímo na straně DWH, čímž se proces ETL proměňuje na proces ELT.

3.3 Rostoucí popularita

Jedním z rozhodujících faktorů, které se podílí na rostoucí popularitě dbt je právě míra ulehčení práce oproti množství úsilí nutného vynaložit na nastavení a propojení s ostatními systémy. Díky četným integracím s moderními cloudovými DWH je propojení velmi jednoduché a samotná křivka učení je strmá, takže i u méně zkušených uživatelů je možné dosáhnout rychlého ovládnutí všech využívaných konceptů. Míru ulehčení popisuje cloud a DevOps inženýr Toby Agboola v rozhovoru pro společnost Inawisdom následovně: „*Dříve, když analytik požádal datové inženýry o transformaci nějakých dat, trvalo to obvykle asi 3 dny, protože jsme museli vytvořit pipeline, otestovat ji a integrovat s architekturou. Ale s dbt to bylo možné udělat za půl dne. Někdy se požadavky v průběhu zkoumání dat měnily. Datový tým si například mohl uvědomit, že potřebuje tržby za celý rok místo pouhých tržeb, takže data musela projít další transformací. Díky dbt mohl datový tým spustit tolik dotazů nebo modelů, kolik potřeboval, aby získal potřebné odpovědi.*“ [32]

Framework dbt je vhodný pro všechny projekty, které v nějaké fázi potřebují data transformovat. Jedinou nevýhodou je nutnost mít data již před transformací načtená v DWH kvůli

neexistující možnosti extrahování dat ze zdrojových systémů s pomocí dbt (viz. kapitola 3.1). Univerzálnost nástroje dbt přispívá k jeho rostoucí popularitě, protože umožňuje přístup širokému spektru uživatelů napříč různými odvětvími. Oblast, ve které nástroj dbt vyniká nejvíce, jsou podniky, které již používají cloudové služby, protože se snadno začleňuje do jejich existující architektury.

3.4 dbt Cloud vs. dbt Core

Před samotným popisem dbt v rámci Databricks je dobré zmínit dvě rozdílné verze, se kterými je možné se setkat. Tou první je dbt Core, open source verze, kterou lze bezplatně používat na lokálním stroji v příkazové řádce. Poskytuje veškeré funkce týkající se transformace a analýzy dat, ale je potřeba tuto verzi propojit např. s IDE VScode, které poté přebírá roli řízení výpočtu a poskytovatele výpočetních prostředků.

Jednodušší možnost, než spouštět a spravovat vše lokálně s pomocí příkazové řádky nabízí dbt Cloud, placená verze dbt v cloudu, poskytující stejnou základní funkčnost rozšířenou o další, stále se rozrůstající komponenty. Tato verze je sice již placená, ale nabízí vlastní IDE (integrované vývojové prostředí, z anglického „Integrated Development Environment“) běžící v prohlížeči, job scheduling (plánování úloh), integraci s GitHubem a GitLabem, logování, alerty a hostovanou dokumentaci, což dělá z dbt Cloud verzi ideální především pro týmy, které chtějí čerpat výhod služby spravované ze strany dbt Labs (společnost stojící za frameworkem dbt), nebo chtějí využít nástroje pro spolupráci, které jsou zmíněné výše. [30][34]

3.5 Metadata v dbt

Metadata jsou nedílnou součástí frameworku dbt. Ať jde o metadata popisující aktuální stav projektu, specifikující datové typy sloupců modelů, nebo metadata o jednotlivých modelech a jejich konfigurace. Je také možné zahrnout do implementace projektu i row-level metadata (na úrovni řádků, jednotlivých záznamů) pro zjednodušení řešení problémů a optimalizaci zároveň se získáním nadhledu nad daty. Jde o monitoring nad standard artefaktů, které dbt u každé spuštěné úlohy ukládá. Motivace pro vlastní implementaci může být omezení artefaktů na modely samotné, bez využití hlubšího monitorování s využitím row-level metadat.

V dbt jsou po každé spuštěné úloze nasbírána metadata z produkčního prostředí a dbt Explorer dle typu spuštěných příkazů zpřístupní k nahlédnutí graf závislostí modelů, jejich detaily a sloupce, výsledky testů, statistické údaje o modelech, zdrojových tabulkách a snapshot tabulkách, případně jejich detaily.

Specifikování datových typů sloupců jednotlivých modelů může usnadnit případné modifikace v budoucnosti. Datový typ stačí přepsat na jednom místě a změna se do všech ovlivněných míst zpropaguje. K dosažení je možné využít dva způsoby, a to přetypování (cast), nebo definování datového typu v souboru *dbt_project.yml* v části *seeds* (tato druhá varianta však oficiálně není doporučována a v samotné dokumentaci je uvedeno, že se tento způsob konfigurace má používat pouze v nezbytných případech. [35] K přetypování je možné využít buď více rozšířený syntax (kód 3.2), nebo dvě dvojtečky (kód 3.3). [36]

Výpis kódu 3.2 Přetypování sloupců v dbt s použitím „cast()“. [36]

```
select
    cast(order_id as integer),
    cast(order_price as double(6,2))
from {{ ref('stg_orders') }}
```

Výpis kódu 3.3 Přetypování sloupců v dbt s použitím syntaxe „::“. [36]

```
select
    order_id::integer,
    order_price::numeric(6,2)
from {{ ref('stg_orders') }}
```

Pro modely je možné také definovat další různá metadata. Ta mohou být definována v souboru *models/schema.yml, dbt_project.yml*, nebo použitím Jinja makra *config()* přímo v SQL souboru daného modelu. Informace, které takto uložíme, jsou poté viditelné v automaticky generované dokumentaci proběhlých úloh. Charakter těchto ukládaných metadat není definován a je tak pouze na potřebách daného projektu a vývojářů, k čemu tuto možnost využijí (zaznamenání autora modelu, stav modelu, oblíbenou barvu a další). [37]

Ukázka kódu 3.4 zachycuje definování metadat *author*, *model_status* a *favorite_color* pro model *activities*.

Výpis kódu 3.4 Definování dalších metadat pro model activities. Inspirováno: [37]

```
--- soubor models/schema.yml
models:
  - name: activities
    meta:
      author: "@vitezslav"
      model_status: "in development"
      favorite_color: "black"
```

Do metadat můžeme počítat i dokumentační komentáře, které se berou v potaz především při generování dokumentace, kterou obohacují o informace k jednotlivým modelům a dalším součástem celého projektu. Pro implementační detaily lze nahlédnout do kódu 4.4, kdy je s pomocí parametru *description* přidán popis k modelům i k jednotlivým sloupcům. To kromě zlepšení porozumění kódu při vyvíjení zvyšuje i kvalitu samotné dokumentace. Celý projekt je poté lépe pochopitelný i pro nové vývojáře a lidi, kteří se přímo na dosavadní implementaci daného projektu nepodíleli.

3.6 Platformy podporující dbt

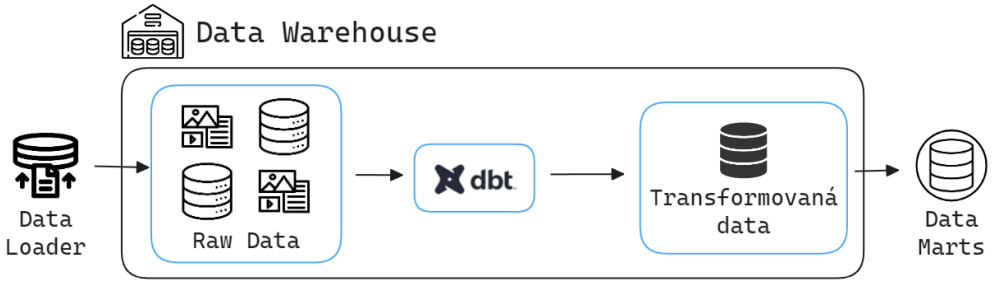
Díky možnostem konfigurace dbt je velké množství různých databází, datových skladů, datových jezer, dotazovacích vyhledávačů a analytických platforem, kde se s tímto frameworkem lze setkat. Daná propojení (adapters) poskytují rozličné úrovně podpory, které lze rozdělit do tří kategorií. První, verifikovaná propojení (verified adapters), jsou taková, která jsou spravována a podporována přímo dbt Labs a často podporují jak možnosti dbt Core, tak i dbt Cloud. Do této kategorie spadá například Postgres, Spark, Redshift, Snowflake a Databricks. Druhá kategorie, důvěryhodná propojení, jsou spravována a podporována ze strany dodavatele daného řešení a využívají pouze verzi dbt Core. Často se jedná o řešení se stabilní uživatelskou základnou, jako například Athena, Glue, Materialize, Oracle Autonomous Database. Zbývající skupina propojení je udržována komunitou uživatelů, kteří se o ně starají. Nelze však v takových případech zajistit stabilitu a aktuálnost všech řešení. Mezi nejznámější zástupce této rozsáhlé kategorie patří MySQL, SQLite, Hive, Impala, Teradata, AWS Glue a Amazon Athena. [38]

Důležité je zmínit fakt, že tyto kategorie a podporovaná propojení se stále vyvíjí a dochází k četným změnám. Aktuálnost informací uvedených výše si lze ověřit v oficiální dokumentaci, ze které pochází citace momentálního stavu propojení.

3.7 Propojení s Databricks

Připojení na Databricks je možné s pomocí Partner Connect, nebo manuálně, přičemž pro práci s dbt je doporučováno nastavit vše manuálně a seznámit se tak s celým procesem nastavení a integrace. [39] Manuální nastavení vyžaduje vytvořený cluster na nakonfigurovaný SQL Warehouse (typ výpočetní jednotky, viz. kapitola 1.1) a informace potřebné k napojení na něj (Server Hostname, Port, a HTTP). Pak už si stačí jen v uživatelském nastavení pod záložkou *developer* vygenerovat přístupový token a s pomocí návodu krok po kroku nastavit dbt projekt, napojený na Databricks cluster nebo SQL sklad (SQL Warehouse), repozitář se svým kódem a v případě potřeby několik vývojových prostředí (testovací, produkční). [40] Propojení s využitím Partner Connect je snazší, jelikož automaticky vybere vhodný cluster ze seznamu již existujících clusterů a konfigurační data jako Port, HTTP a Server Hostname nastaví automaticky. Uživateli tak pouze stačí vytvořit si u dbt účet, nechat si automaticky nastavit první demo projekt, a následně v Databricks Workspace nakonfigurovat pro dbt, které je vedeno jako další uživatel, práva k zobrazení dat a jejich možnou úpravu ve schématu, kde bude framework využíván. Výchozí verze použitá při tomto typu propojení je dbt Cloud.

Po provedení výše zmíněných kroků je možné vytvářet vlastní modely, psát SQL kód a dotazovat se nad tabulkami a jednotlivými daty přímo v dbt. Kód se kompiluje s pomocí příkazu `dbt build`, spouští příkazem `dbt run`, případně je možné za příkazem vždy specifikovat, pro jaký model má být proveden, tedy `dbt run --select <model_name>`.



Obrázek 3.1 Znázornění úlohy frameworku dbt v datovém skladu. Obrázek vytvořen autorem s pomocí nástrojů [4][5]. Inspirace: [41].

Jak již bylo zmíněno v kapitole 3.1, dbt slouží pouze pro transformaci dat a neumožňuje vykonávat žádné další procesy související s datovým skladem, kterými je například načtení dat (load) nebo jejich extrakce. Je tedy stále nutné data nějak načíst, uložit, a až následně transformovat s pomocí dbt (v případě využití ELT, jinak následuje jejich uložení až po zpracování v dbt). Výsledkem jsou transformovaná data, která mohou být dále využita například v datových tržištích. Zasazení dbt do ekosystému Databricks znázorňuje obrázek 3.1. Kombinace robustní infrastruktury pro práci s daty poskytované platformou Databricks a možností automatizace a správy transformací dat nabízených dbt umožňuje uživatelům vytvářet a provozovat komplexní datové pipeline a zlepšit řízení a monitorování datových toků. Tato integrace umožňuje uživatelům efektivně využít výhod obou platforem a optimalizovat tak své datové procesy.

Kapitola 4

Realizace tvorby stage vrstvy DWH

Kapitola obsahuje praktickou část této závěrečné práce, zabývající se implementací stage vrstvy. Na úvodu kapitoly je představen systém ProBIS, dále zdroj dat, která do něj vstupují a samotná stage vrstva již existujícího, stávajícího on-premise datového skladu. Následně jsou na konkrétních datech provedeny dvě realizace stage vrstvy, jedna s pomocí frameworku dbt, druhá s využitím DLT. Pro každý způsob jsou v příslušných sekcích rozebírány postupné kroky, ke kterým v rámci tvorby stage došlo. Tyto kroky zahrnují nastavení prostředí, nahrání dat, transformaci dat, validaci (testování datové kvality), historizaci, možnosti automatizace a další sekce specifické pro dané přístupy. Celý proces je následně zhodnocen v kapitole 5, kde dochází i k porovnání obou přístupů vzájemně.

4.1 Představení ProBISu

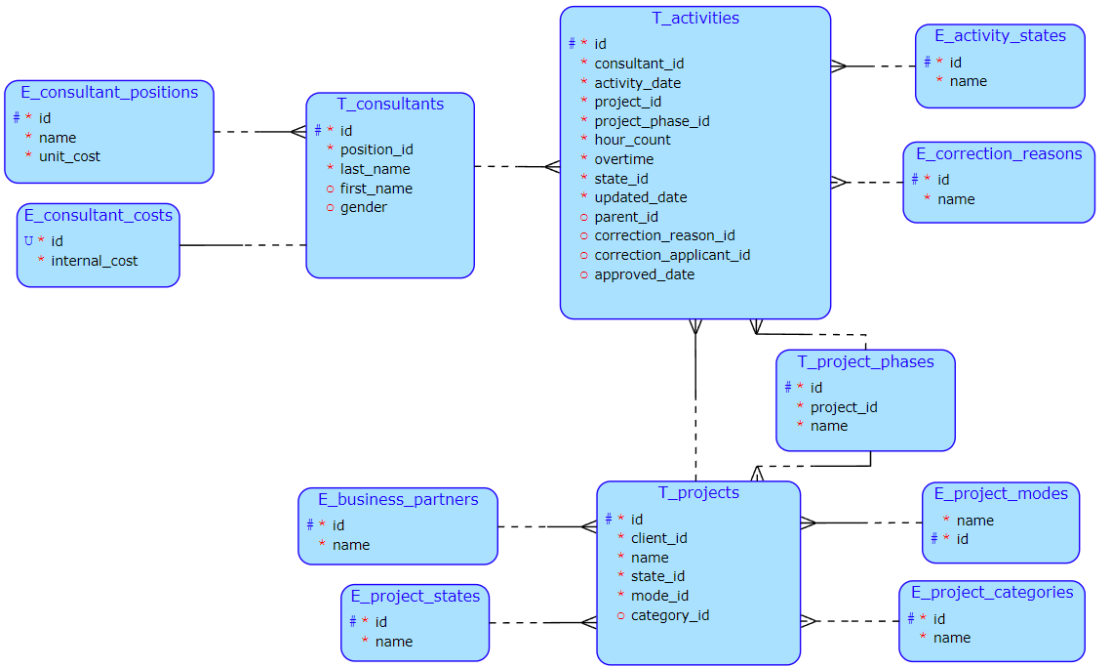
ProBIS je interní datový sklad, sbírající data z primárních systémů jedné IT firmy v Praze, kterou pro účely této bakalářské práce budeme nazývat fiktivním jménem ABC. Firma má několik zdrojových systémů jako například Profis, Navision, Bugzilla a další. Data z těchto systémů následně firma prezentuje ve formě reportů. Návrh skladu ProBIS vznikl v roce 2023 jako součást diplomové práce ([25]), která si kladla za cíl analyzovat požadavky, navrhnout architekturu řešení, postup řešení a realizaci vybraných kroků z postupu řešení. V současné době je implementována především stage vrstva, na implementaci dalších součástí se ve firmě stále aktivně pracuje. Stávající řešení používá databázový server Microsoft SQL Server a pro transformace používá pattern ELT. Celý tento datový sklad se skládá z několika databází (Profis, DWH_ADMIN, Stage, Core, DataMarts). Pro účely této bakalářské práce jsou důležité především databáze *Profis*, sloužící jako hlavní zdroj dat a *Stage*, která bude implementována v rámci praktické části.

4.2 Zdroj dat

Jak bylo zmíněno v podkapitole 4.1, hlavním zdrojem dat pro stávající DWH je Profis, hlavní informační systém pro podporu byznys procesů v organizaci ABC. Většina dat je transakčních, ale nalezneme zde i číselníky firmy a data z účetního systému Microsoft Dynamics NAV. Profis je vytvářen jako modulární aplikace. Obsahuje modul Vykazování, Fakturace, Obchod, CRM, BackOffice, Archiv, Administrace a Reporting. Pro účely této práce není potřeba věnovat se všem modulům.

Stage vrstva bude tvořena nad vybranými daty ze čtyř modulů. Těmito moduly jsou Vykazování (*Utilization*), BackOffice (*HR*), Obchod (*Sales*) a Administrace (*Finance*). Veškerá používaná transakční data byla předem anonymizovaná a není tak možné je namapovat na konkrétního uživatele nebo aktivitu ve firmě. [42] Jelikož nejde o kompletní data ze systému Profis (z důvodů jako například bezpečnost), ale pouze jistý anonymizovaný výběr, pohybují se počty záznamů v tabulkách řádově mezi tisíci až desítkami tisíc. Číselníky mají nižší počty záznamů, konkrétněji jednotky až desítky dat, zatímco tabulky obsahující konzultanty, projekty a aktivity mají počet záznamů řádově vyšší.

Celková provázanost tabulek (tedy i jednotlivých modulů), které budou v rámci této práce využity je znázorněna na obrázku 4.1, včetně kardinalit a typů vztahu.



Obrázek 4.1 Konceptuální schéma tabulek, které budou v praktické části této závěrečné práci využity. Tvůrce: autor, použitý software: [43].

Detailní popis modulů včetně příslušných tabulek je následující: [42]

- Modul **Vykazování** slouží pro podnikové činnosti spojené s evidencí, plánováním a schvalováním pracovních činností, dovolených a zákaznických podpor.

  Pod tento modul spadají tabulky **t_activities**, **e_activity_states**, **e_correction_reasons**.

- Modul **Obchod** slouží pro podporu obchodního cyklu ve firmě. Jeho hlavním zaměřením je evidence zakázek, obchodních výkazů a dokumentů.

  Pod tento modul spadají tabulky **t_project_phases**, **t_projects**, **e_business_partners**, **e_project_categories**, **e_project_modes**, **e_project_states**.

- Modul **BackOffice** má několik sekcí a slouží pro zaměstnance, kteří zajišťují veškerou administrativu pro chod společnosti. Evidují se zde zaměstnanci, externisté, vozový park, kniha jízd, karty CCS, telefony a další.

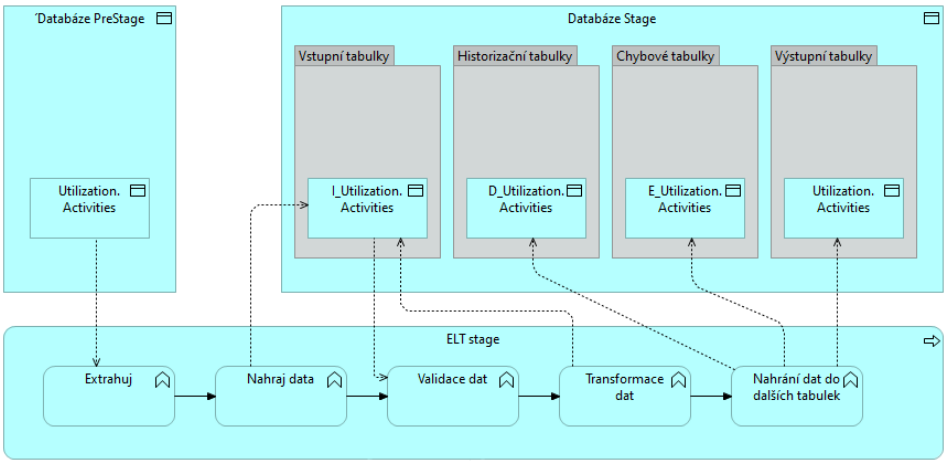
  Pod tento modul spadají tabulky **t_consultants**, **e_consultant_positions**.

- Modul **Administrace** slouží pro evidenci uživatelů, umožňuje spravovat jejich pracovní tarify a pozice. Obsahuje i správu číselníků aplikace a přehled pevných číselníků. Jeho součástí je i správa osobního nastavení každého uživatele.

  Pod tento modul spadá tabulka **e_consultant_costs**.

4.3 Existující Stage vrstva

Ve stage vrstvě dochází k ověření datové kvality, zálohám vstupních snímků, přidání technických sloupců (timestamp změny, autor záznamu) a obsahuje 4 typy tabulek, které rozlišuje na vstupní, historizační, chybové a výstupní.



Obrázek 4.2 Pohled na fungování Stage u stávajícího řešení DWH (na konkrétní tabulce Activities). [25]

Do **vstupních tabulek** se nahrávají data z Profisu. Nad vstupními tabulkami se provádí ověření datové kvality, a následně se data transformují a nahrávají do dalších tabulek.

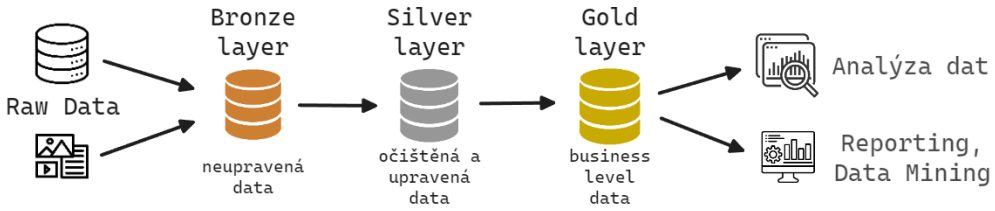
Historizační tabulky obsahují všechna data ze vstupních tabulek. Obsahují tedy validní i nevalidní data. V těchto tabulkách se historizuje 15 denních snímků dat.

Do **chybových tabulek** se nahrávají záznamy, které jsou označeny jako nevyhovující z hlediska kvality.

Mezi validace datové kvality patří kontrola formátu dat. Například zda hodnoty obsahující datum jsou ve správném formátu. Validní data se nahrávají do **výstupních tabulek**.

Tyto tabulky později poskytují zdroj pro databázi Core. Jak lze vidět na obrázku 4.2, tento datový sklad využívá PreStage vrstvu. Tato práce se bude odkazovat především na medailonovou architekturu, která logicky uspořádává data na různé úrovně kvality s cílem postupně během průchodu jednotlivými vrstvami zlepšovat jejich kvalitu a strukturu. Schéma této architektury je znázorněno na obrázku 4.3.

Vrstvy medailonové architektury se dělí na bronzovou, stříbrnou a zlatou (*bronze*, *silver*, *gold layer*). Bronzová vrstva je ekvivalentní s již zmíněnou PreStage a obsahuje neupravená data načtená ze zdrojů. Stříbrná vrstva slouží k filtrování, čištění a definování struktury dat pro poskytnutí pohledu na všechny klíčové entity a dá se přirovnat k Stage vrstvě. Zlatá vrstva poté obsahuje data specifická pro konkrétní projekty, tedy data optimalizovaná pro využití v Business Intelligence, Strojovém učení a Data Science a jedná se o téměř ekvivalent datového tržiště.



Obrázek 4.3 Schéma medailonové architektury. Obrázek vytvořen autorem s pomocí nástrojů [4][5]. Inspirace: [44].

4.4 Import dat do Databricks Unity Katalogu

Pro účely praktické části byl založen AWS (Amazon Web Services) účet, na který jsou nahrána zdrojová data ve formě csv souborů do úložiště S3 Bucket a je vytvořen Databricks Workspace v CloudFormation. Tento workspace slouží pro veškeré úkony v rámci praktické části této závěrečné práce. Prvním úkonem je nahrání dat uložených v AWS S3 Bucketu do Databricks a vytvoření tabulek. Tyto tabulky budou sloužit jako výchozí data pro stage vrstvu a nebudou proto při nahrání z AWS nijak kontrolována, nebo transformována. Kontrola kvality a transformace budou součástí až následujících kroků. Tato data jsou v medailonové architektuře označována jako bronzová vrstva.

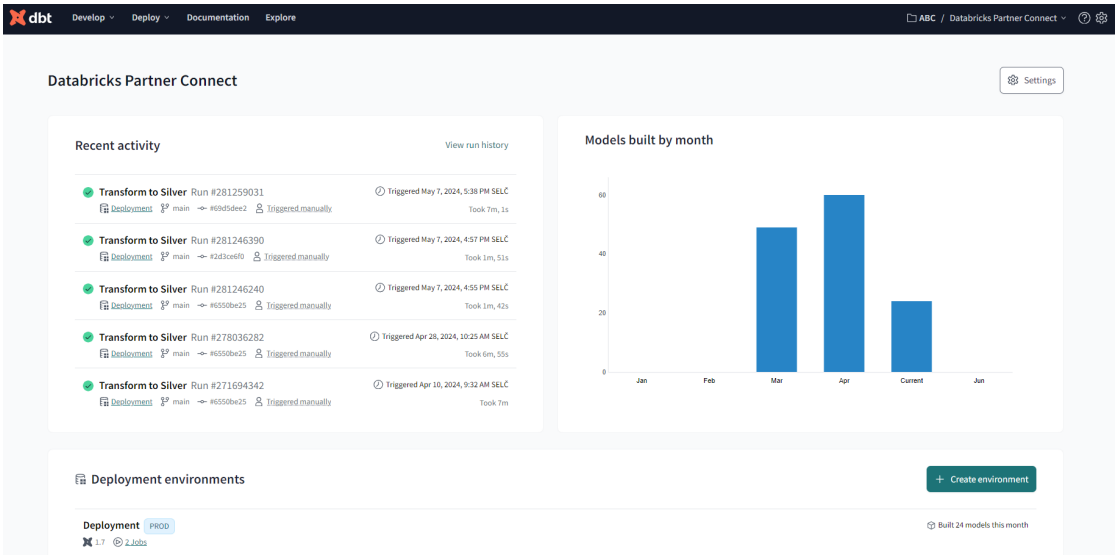
Výpis kódu 4.1 Kód pro načtení dat z S3 a vytvoření tabulky e_business_partners pod schéma *manual*.

```
CREATE OR REPLACE TABLE manual.e_business_partners
AS SELECT
*
FROM
  read_files(
    's3://databricks-workspace-stack-2e4e7-bucket/unity-
    catalog/3246982087025617/profisdata/manual/e_business_partners.csv',
    format => 'csv',
    delimiter => ';',
    schemaEvolutionMode => "none");
```

Na import dat je vytvořen job, který spouští notebook, obsahující SQL skripty pro vytvoření, případně aktualizování tabulek v Databricks. Ve výpisu kódu 4.1 je ukázán SQL skript pro jednu z vytvářených tabulek. Pro ostatní tabulky je kód obdobný. Veškeré tabulky, tedy tyto *raw* (původní, neupravené) tabulky, transformované stříbrné tabulky i případně zlaté tabulky, se budou nacházet v Unity Katalogu, jednom z úložišť na Databricks.

4.5 Implementace Stage vrstvy s dbt Cloud

V této podkapitole dojde k implementaci stage vrstvy v prostředí dbt Cloud s pomocí SQL. Následně v podkapitole 4.6 bude postup zreplikován pro DLT a postupy budou detailně srovnány v kapitole 5. Podkapitoly jsou seřazeny chronologicky dle postupu řešení, jak na sebe jednotlivé části navazují. Řazení podkapitol celého postupu v frameworku dbt slouží spíše jako ilustrativní či reprezentativní příklad, v jakém pořadí dílčí části řešit, než jako imperativní návod.



Obrázek 4.4 Úvodní stránka webového prostředí dbt po propojení s Databricks Partner Connect, zde již s předešlou historií běhu vytvořené úlohy *Transform to Silver*.

4.5.1 Nastavení prostředí

Pro implementaci byla zvolena možnost využít propojení přes Partner Connect. Stačí vyhledat „dbt“, vytvořit si dbt účet a od té doby je možné jednoduše se do prostředí dostat s pomocí tlačítka *Sign in*. Díky Partner Connect jsou veškeré potřebné konfigurační detaily automaticky předány. Není potřeba řešit PAT (Personal Access Token), Server Hostname, Port, ani HTTP cestu. Jak bylo zmíněno již v kapitole 3.4, vývoj je ve verzi Cloud možný s pomocí web IDE. Verzování může být realizováno pomocí konkrétního online repozitáře (například GitHub, GitLab) nebo tzv. „Managed“ repozitáře, který je spravován automaticky. Tento druh repozitáře nesplňuje veškeré doporučené postupy pro správu verzí, jelikož neposkytuje například možnost pull requestů. Nicméně lze ho využít pro projekty, u kterých vývojáři nechtějí řešit správu vlastního repozitáře, avšak stále chtějí využívat výhody verzování, integrovaného do vývojového prostředí. Takový repozitář může sloužit i jako základ pro projekty, které později přejdou k vlastnímu externímu repozitáři. To následně dovoluje možnost úplného exportu projektu.

Celé prostředí dbt je velmi minimalistické a na hlavní stránce se nachází pouze několik rozkliknutelných záložek. Ukázka hlavní stránky je na obrázku 4.4. Pod záložkou *Develop* se nachází Cloud IDE pro vývoj a nastavení Cloud CLI (Command Line Interface). Pod záložkou *Deploy* poté nastavení prostředí, job a historie předešlých úloh (jobů). Úplně vpravo na obrazovce je poté možnost dostat se k nastavením účtu a profilu.

4.5.2 Zpracování změn v datech - „snapshots“

Před samotnou transformací dat je zde dobré zmínit, jaké možnosti nabízí dbt pro práci s daty, která se s časem mohou v datovém skladu měnit. Typicky jde o vznik nových záznamů v databázi, zánik záznamů, nebo jakoukoliv jejich modifikaci. K dispozici jsou tzv. „snapshots“, tedy snímky dat, které načítají nezpracovaná data z bronzové vrstvy a přidávají k nim sloupečky s časovým razítkem, které poté slouží k rozpoznání, zda jde o nový záznam, nebo již byl záznam přítomen dříve. Jedná se v podstatě o předem implementované řešení transformace SCD2 (viz. kapitola 2.3 pro efektivní zachycení a správu aktuálních i historických změn dat, které je tak možné jednoduše v dbt využívat. Je tedy důležité si všimnout, že ačkoliv je historizace v dbt

Výpis kódu 4.2 Snapshot pro tabulku t_projects.

```
{% snapshot activities %}
    {{
        config(
            target_schema="snapshots",
            unique_key="id",
            strategy="timestamp",
            updated_at="updated_date")
    }}
    SELECT *
    FROM vitezslav_husek_bp.profis.t_activities;
{% endsnapshot %}
```

nazvána jako snapshoty, co se implementace týče nejde o Snapshotting, ale o SCD2. Typicky není potřeba „snapshotovat“ všechny tabulky. Některé mohou sloužit jako prosté číselníky a nepočítá se s tím, že by byly záznamy často, pokud vůbec, modifikovány. Jelikož i zdrojová data pro tuto práci obsahují některé číselníky, je snapshot prováděn pouze na některých vhodných tabulkách a zbytek je do stříbrné vrstvy načítán rovnou z Databricks.

Ve výpisu 4.2 je ukázán jeden ze snapshotů, který byl vytvořen. Je uložen v souboru *activities.sql* ve složce *snapshots*. Samotný kód obsahuje konfiguraci a `SELECT` statement, který má být uložen. Konfigurační část se může nacházet v odlišném souboru, nebo přímo v snapshotu. Obsahuje cílové schéma, název sloupečku s unikátními hodnotami, typ snapshotu, který má být použit a příslušný parametr. Jako kategorii lze zvolit *timestamp*, *check* nebo definovat vlastní způsob ověření modifikovaných záznamů. [45]

Timestamp vyžaduje předání parametru *updated_at*, podle kterého dbt rozlišuje nově modifikované záznamy. Do tohoto parametru je nutné předat sloupec, ve kterém je časové razítko informující o poslední aktualizaci záznamu. Pro data použitá v této práci je možné využít sloupec *updated_date* v tabulce *t_activities*, neboť nese informaci o poslední změně aktivity. Ostatní tabulky ale takové sloupce nemají, a proto je vhodnější použít další možnou kategorii snapshotu, *check*.

Check oproti *timestamp* nevyžaduje časové razítko, ale v parametru *check_cols* očekává seznam sloupců, které má k hledání změn využít. V tomto případě dojde k načtení všech hodnot v předaných sloupcích a jejich následnému srovnání se sloupci předchozího snapshotu. Jde tedy o jakési srovnávání listů se záznamy a zjišťování, zda na každém indexu je v každém listu stejná hodnota, jako u minulé verze. Tento přístup je obecně doporučován méně než *timestamp*, ale pro snapshotování tabulek bez sloupečku s časovým razítkem modifikace je využitelný.

4.5.3 Transformace dat

Transformace do stříbrné vrstvy probíhá s pomocí souborů umístěných ve složce *models*. V jednotlivých SQL souborech je možné využívat libovolné transformace, vlastní definovaná makra, nebo vestavěné funkce, ulehčující celkovou práci s daty. Toto dělení na jednotlivé moduly přispívá celkové modularizaci a znovupoužitelnosti částí kódu. Jedním takovým příkladem je předdefinovaná funkce *ref()*, která byla zmíněna již v ukázce 3.1. Kromě využití pro odkazování se na jiné tabulky (v terminologii dbt také modely) je možné navázat na předchozí kapitolu o snapshotech a použít právě je. V ukázce 4.3 je použita funkce *ref()* dvakrát, v obou případech jde o referenci na snapshot. Dbt to rozpozná a není potřeba nic víc, než dát do funkce název chtěného snapshotu. Při kompilaci se reference přeloží na absolutní cestu modelu a dochází tak k plynulému přechodu.

Transformace v ukázce 4.3 spočívá ve spojení více tabulek do jedné na základě jejich identi-

Výpis kódu 4.3 Transformace v dbt s využitím funkce ref() pro data ze snapshotů.

```
SELECT
    projects.id,
    partners.name AS client_name,
    projects.name,
    s.name AS state,
    m.name AS mode,
    c.name AS category
FROM {{ ref("partners") }}
INNER JOIN {{ ref("projects") }} ON partners.id = projects.client_id
INNER JOIN vitezslav_husek_bp.manual.e_project_states s ON s.id = projects.state_id
INNER JOIN vitezslav_husek_bp.manual.e_project_modes m ON m.id = projects.mode_id
LEFT JOIN
    vitezslav_husek_bp.manual.e_project_categories c
    ON c.id = projects.category_id;
```

fikátorů a následné kontroly kvality, které je dosaženo s pomocí data testů (viz kapitola 4.5.4). Při spojení tabulek jsou také přejmenovány nové sloupce, aby v nové tabulce lépe odpovídaly kontextu.

4.5.4 Validace dat

Po transformaci je vhodné zkontrolovat kvalitu výstupních dat, aby bylo jisté, že žádný záznam již neobsahuje neočekávané hodnoty. Dbt umožňuje pro tuto kontrolu tzv. *testy* (tests). Ty je možné libovolně definovat ve stejnojmenné složce, využít již předpřipravené základní typy testů, nebo využít různých balíčků (knihoven) pro dbt, poskytující další předvytvořené testy. Konfigurace testů pro jednotlivé tabulky probíhá v souboru *schema.yml*, umístěného ve složce *models*. Kromě toho tento soubor slouží jako definování schématu jednotlivých modelů a jejich popis, který lze využít pro automatické generování dokumentace. Tato možnost využití bude rozebrána více v kapitole 4.5.6.

Základní rozdělení pro testy je na *unit* a *data*. Historicky se pokrytí testů dbt omezovalo na data testy, které posuzovaly kvalitu vstupních dat nebo strukturu výsledných datových sad. Tyto testy je však možné provádět až po sestavení modelu a proto došlo k vzniku unit testů. [46] Podpora unit testů je poskytována pro SQL modely. Použití těchto testů je vhodné především, pokud modely využívají komplexnější SQL logiku, například regex (regulární výrazy) a window funkce, případně pokud chceme vyzkoušet SQL logiku pro edge case (okrajový případ) dat, který není ve stávajících datech reprezentován. Pro model v ukázce 4.4 byly použity pouze data testy.

Mezi základní data testy patří následující:

- **unique** testuje unikátnost hodnot v daném sloupečku
- **not_null** kontroluje chybějící hodnoty
- **accepted_values** umožňuje poskytnutím výčtu očekávaných hodnot zjistit, zda se ve sloupci nenachází nějaká nechtěná hodnota
- **relationships** testuje vztah s ostatními modely

V ukázce je testován vztah modelu *t_activities* s modely *t_consultants* a *t_projects*. Jelikož *t_activities* obsahuje cizí ID konzultantů a projektů, tento test kontroluje, zda ID nacházející se v sloupcích *consultant_id* a *project_id* mají záznam v příslušných modelech. Kromě toho jsou některé sloupce

Výpis kódu 4.4 Zkrácená ukázka ze schéma dbt modelu t_activities s popisky a testy v konfiguračním souboru v jazyce YAML.

```yaml
models:
  - name: t_activities
    description: "Model encapsulating activities, different timestamps,
                  their state, overtime, correction metadata and more."
    columns:
      - name: id
        description: "The primary key for this table."
        tests:
          - unique
          - not_null
      - name: consultant_id
        description: "Foreign key pointing to id in t_consultants."
        tests:
          - not_null
          - relationships:
              to: ref('t_consultants')
              field: id
      - name: activity_date
        description: "Activity date."
        tests:
          - not_null
      - name: project_id
        description: "Foreign key pointing to id in t_projects."
        tests:
          - not_null
          - relationships:
              to: ref('t_projects')
              field: id
      - name: project_phase
        description: "Phase of the project in time of the activity."
        tests:
          - not_null
      - name: hour_count
        description: "Number of hours of the activity."
        tests:
          - not_null
          - dbt_utils.accepted_range:
              min_value: 0.0
              max_value: 8.0
      - name: overtime
        description: "Number of overtime hours."
        tests:
          - not_null
      - name: state
        description: "Activity state."
        tests:
          - not_null
      - name: correction_reason
        description: "Reason of activity record correction."
      - name: correction_applicant_id
        description: "ID of corrector."
```

testovány také na *not_null* a sloupec s identifikátorem na unikátnost. Tabulka obsahuje i sloupce, které nemusí obsahovat vyplněné hodnoty. U takových případů není prováděn žádný test.

Jak je zmíněno výše, je také možné využít další balíčky pro širší spektrum předpřipravených testů. Jedním takovým je balíček `dbt_utils` od dbt_labs. Přidání vyžaduje vytvoření souboru *dependencies.yml* nebo *packages.yml* na stejné úrovni jako se nachází *dbt_project.yml*. Do něj je poté možné ve formátu YAML je vkládat jednotlivé závislosti, tedy názvy balíčků a jejich verzi, která je pro projekt žádoucí. V balíčku *dbt_utils* se nachází generické testy, makra, SQL generátory a další. Tento balíček byl pro názornost využit na generický test, konkrétně test hodnot v zadaném intervalu (`accepted_range`). V ukázce 4.4 je tento test použit pro sloupec *hour_count*, který v tabulce nese informaci o počtu odpracovaných hodin. Jelikož zaměstnavatel silně doporučuje nepsat do výkazu hodin více než 8 hodin za den a případné přesčasy si ve vykazování rozložit do více dnů, bude nás zajímat míra dodržování tohoto postupu. Pro názornost tuto chybu nebudeme považovat za zásadní a nebudeme ihned nevhodné záznamy odstraňovat, stále je však chceme odhalit a zaznamenat. K tomu je potřeba přenastavit způsob, jakým ve výchozím módu vnímá dbt testy.

Výchozí odpovědí dbt na neúspěšnou validaci dat (tedy některé ze záznamů nevyhovují naším představám) je *error*, který po provedení testů zastaví případné další příkazy ve frontě, například v úloze (jobu) selže pipeline během *dbt build*. Druhým, více vyhovujícím způsobem pro méně závažné testy je *warn*, který do logu zaznamená počet nevyhovujících záznamů a testy, kterých se varování týká, ale běh dalších příkazů nepřeruší. Způsob reakce na test je možné nastavit pro každý test zvlášť, nebo globálně pro celý projekt. Globální nastavení probíhá v souboru *dbt_project.yml*, jednotlivým testům jde přiřadit jinou reakci přímo po jejich definování v souboru *schema.yml*.

4.5.5 Možnosti automatizace

Pro zjednodušení veškerých výše zmíněných procesů lze nastavit *Job*. Toto nastavení je možné přímo v Databricks, kde se jako typ nastaví dbt, dále se vybere zdroj a příkazy, které mají být spuštěny, nebo v dbt cloudu, kde je možné vytvořit job spustitelný přes zavolání API (Application Programming Interface), ručně, nebo opakovaně ve zvolený čas. Jelikož se job v dbt napojuje na výpočetní jednotku v Databricks (ať už cluster, nebo SQL Warehouse), jedná se téměř o stejný job a volba je čistě na uživateli. Obě možnosti lze jednoduše zařadit do Databricks pipeline, včetně volání API, které se vloží do Python notebooku a bude spuštěno v rámci pipeline. Aby spouštění dávalo smysl, je potřeba jej sladit s načítáním dat do Databricks, jinak by se dbt spouštělo stále na stejných datech.

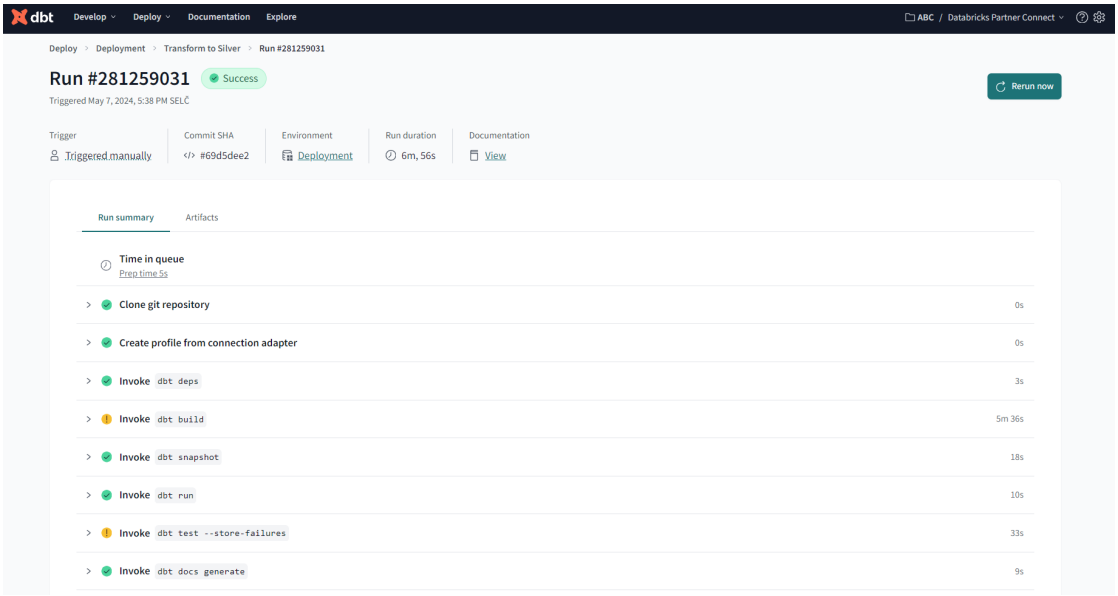
Volání API je možné pouze ve verzi dbt pro týmy. Tato verze je i při použití přes Databricks dále zpoplatněna. Ve stávající době (duben 2024) je možné používat plán *Developer*, který je zdarma a poskytuje přístup ke všem částem dbt Cloud pro jednoho uživatele, kromě přístupu k volání API . Přístupy zdarma zahrnují webové IDE, spouštění a správa úloh (jobů), dokumentace, ukládání logů, integrace s GitHubem, jeden projekt na účet a neomezené spouštění úloh. Plán *Team* povoluje přístup dalším lidem do dbt projektu až do maxima 8 lidí a umožňuje využití API přístupu. Cena této verze je momentálně (tzn. duben 2024) 100$ za měsíc za dalšího vývojáře na projektu. Poslední možností je plán *Enterprise*, který je poskytován na žádost a je uzpůsoben potřebám podniků, které ho chtějí využít, včetně nacenění. Pro potřeby této práce byl využit plán *Developer*.

4.5.6 Dokumentace

Součástí běhu jobu, který většinou provádí build, transformaci a validaci modelů, může být v dbt také dokumentace. V nastavení úlohy (jobu) je dostupné zakliknutí možnosti generovat dokumentaci automaticky, případně vygenerovat dokumentaci příkazem `dbt docs generate`. Kom-

pletní seznam spouštěných příkazů v rámci úlohy (jobu) je tvořen příkazy `dbt deps`, `dbt build`, `dbt snapshot`, `dbt run`, `dbt test` a `dbt docs generate` v tomto přesném pořadí.

Vzhledem k dřívějšímu popsání všech modelů v souboru *schema.yml*, včetně jejich sloupců a testů kvality je automaticky generovaná dokumentace obsáhlá a přehledně zobrazuje informace o celých tabulkách, sloupcích, testech, transformacích, referencích a závislostech. Navíc si lze kromě původního kódu transformací zobrazit i kompilovaný kód, kde jsou nahrazeny veškeré `ref()` funkce absolutními cestami modelů, které jsou při spuštění kódu využívány. U každého spuštěného jobu se zapnutým automatickým generováním dokumentace je po jejím vygenerování možné se proklikem přes odkaz u jobu k dokumentaci dostat a pročíst ji, včetně ukázek kódu. Jelikož je součástí projektu také soubor *dbt_project.yml*, ve kterém je možné nastavit název projektu, lze v dokumentaci vyhledávat i dokumentace dle názvu projektu.



Obrázek 4.5 Vytvořená úloha pro stage vrstvu v frameworku dbt.

Vytvořená úloha pro historizaci, transformaci a generování dokumentace je k nahlédnutí na obrázku 4.5. Je možné si všimnout oranžových vykřičníků u položek `dbt build` a `dbt test`. Nejde o žádnou chybu, pouze o varování z testů datové kvality. Jak bylo již dříve zmíněno, jedná se o varování, že některé záznamy neprošly datovou kvalitou, ale jsou v transformovaných datech přesto ponechány.

4.6 Implementace Stage vrstvy s DLT

Následuje implementace vrstvy stage pomocí DLT. Kapitoly jsou opět řazeny chronologicky podle postupu řešení. Řazení kapitol se liší od postupu s nástrojem dbt, neboť zde má smysl věnovat se kapitolám v jiném pořadí pro lepší názornost.

4.6.1 Import dat

Jelikož se data nachází již v Unity Katalogu, není nutně potřeba žádné složité načítání a stačí se na ně pouze odkázat s pomocí volání `spark.read.table()`. Pro účely této práce bylo rozhodnuto *raw* tabulky převést do DLT formátu, do odpovídající bronzové vrstvy, aby byla dodržena konvence medailonové architektury. Tento krok je možné provést už při prvotním vytváření tabulek

ze zdrojových dat v AWS S3, vzhledem k datům, která již dříve byla načtena v rámci praktické části s dbt, byly nakonec využity již existující vytvořené tabulky. V dalším kroku, transformaci, budou pro vytvoření stříbrné vrstvy využity tyto bronzové DLT tabulky.

Výpis kódu 4.5 Kód pro vytvoření bronzové tabulky bronze_activities ve formátu DLT.

```
@dlt.table(comment="Activities Raw Table")
def bronze_activities():
    return spark.read.table("vitezslav_husek_bp.profis.t_activities");
```

Speciálně u DLT je poté možnost využít tzv. *streaming table*, tabulku ve formátu Delta, která se zaměřuje na inkrementální zpracování nových dat. Tyto tabulky jsou optimální pro úlohy, které vyžadují aktuální data a nízkou latenci. Jsou také vhodné pro transformace velkých datových sad, jelikož mohou zpracovávat data postupně. Pro aktualizaci dat ve streamovací tabulce je možné použít příkaz `REFRESH`, který načte nejnovější data ze zdroje. Pro aplikaci změn na existující data je třeba použít příkaz `REFRESH TABLE FULL`, který provede celkovou aktualizaci. Je však důležité si uvědomit, že úplné obnovení může mít negativní dopad na existující data, zejména u zdrojů s krátkou dobou pro uchování dat. Pokud jsou data ve zdroji nedostupná, je možné, že nebude možné obnovit ani stará data.

4.6.2 Transformace

Je vytvořen notebook a jednotlivé buňky, ve kterých probíhají transformace tabulek bronzové vrstvy do stříbrné vrstvy. Podobně jako u výpisu kódu 4.7, je zde použit dekorátor `@dlt.table()`. Ve funkci, která definuje DLT tabulku, se načítají tabulky, které se budou spojovat. Díky předchozímu načtení do formátu DLT je možné se na ně odkazovat přes knihovnu *dlt*. Příkaz `dlt.read("<NAZEV_TABULKY>")` najde a vrátí tabulku se stejným jménem. Dalším krokem je přejmenování stejnojmenných sloupců.

V příkladu 4.6 mají všechny tabulky (*projects, partners, states, modes, categories*) sloupec *name*. Při následném propojení tabulek zaniká informace o tom, odkud daný sloupec pochází. Získáme tedy tabulku s několika identicky pojmenovanými sloupci, které bychom při následném přejmenovávání nebo referencování neměli jak identifikovat. Kromě toho samozřejmě nastává problém dalšího zpracování. Pokud bychom takovou tabulku použili ve zlaté vrstvě, nejsme efektivně schopni sloupce rozlišit. Jako nejjednodušší přístup bylo zvoleno přejmenování sloupců před napojením tabulek. Kromě sloupců *name* došlo také k přejmenování *id* vedlejších tabulek, z podobného důvodu. Ve výsledném napojení s použitím `<TABLE_NAME>.join()` tedy v druhém předávaném parametru není nutné specifikovat pro každou tabulku zvlášť, na jakých sloupcích má spojení proběhnout, protože jsou názvy *id* sjednocené podle hlavní tabulky (v ukázce 4.6 jde o tabulku *projects*).

Pro použití `@dlt.read()` je potřeba buňku, respektive notebook, spouštět v DLT pipeline. V opačném případě tabulka nepůjde najít. Pipeline je nutné nastavit na development mód. V tomto módu se opakovaně používá jeden cluster. Ve výchozím nastavení běží clustery po dobu dvou hodin. Tato hodnota může být změněna v nastavení. [47] V produkčním módu dochází k opakovaným spouštěním pipeline v případě specifických chyb, například při selhání spuštění clusteru. Ten je po ukončení pipeline (ať už úspěšného, nebo chybového) vypnut a při snaze o znovuspuštění je potřeba počkat na jeho inicializaci, která trvá (dle předchozích měření) okolo 7 minut, což při vývoji není zdaleka ideální a proto přenastavení pipeline na tento mód považuji při vývoji za nutnost.

Po běhu pipeline je k dispozici graf závislostí tabulek zobrazující počet záznamů v každé z nich. V případě dílčích úprav v kódu vytváření jednotlivých tabulek je možné spustit jen část

Výpis kódu 4.6 Vytvoření stříbrné tabulky silver_projects.

```
@dlt.table(comment="Silver Projects Table")
def silver_projects():
    projects = dlt.read("bronze_projects")
    partners = (dlt.read("bronze_business_partners")
        .withColumnRenamed("name", "client_name")
        .withColumnRenamed("id", "client_id")
    )
    states = (dlt.read("bronze_project_states")
        .withColumnRenamed("id", "state_id")
        .withColumnRenamed("name", "state")
    )
    modes = (dlt.read("bronze_project_modes")
        .withColumnRenamed("id", "mode_id")
        .withColumnRenamed("name", "mode")
    )
    categories = (dlt.read("bronze_project_categories")
        .withColumnRenamed("name", "category")
        .withColumnRenamed("id", "category_id")
    )
    return (projects.join(partners, ["client_id"], how="inner")
        .join(states, ["state_id"], how="inner")
        .join(modes, ["mode_id"], how="inner")
        .join(categories, ["category_id"], how="left")
        .drop("client_id", "state_id", "mode_id", "category_id")
    )
```

pipeline, a to zvolením tabulek v grafu závislostí a kliknutím na tlačítko *Refresh selected*. Není tak potřeba vytvářet všechny tabulky znovu při malé změně v části kódu.

4.6.3 Validace

Po vytvoření transformací v rámci stříbrné vrstvy je kontrola datové kvality v DLT přímočará. Nad dekorátor `@dlt.table()` se přidá dekorátor `@dlt.expect_all()` a do něj je vložena množina pravidel, která očekáváme, včetně jejich definic, ve formátu *json*.

Výpis kódu 4.7 Kontrola datové kvality, konkrétně hodnot sloupců ve stříbrné tabulce *projects*.

```
@dlt.expect_all(
    {
        "id_null": "id IS NOT NULL",
        "client_name_null": "client_name IS NOT NULL",
        "name_null": "name IS NOT NULL",
        "state_null": "state IS NOT NULL",
        "mode_null": "mode IS NOT NULL",
    }
)
```

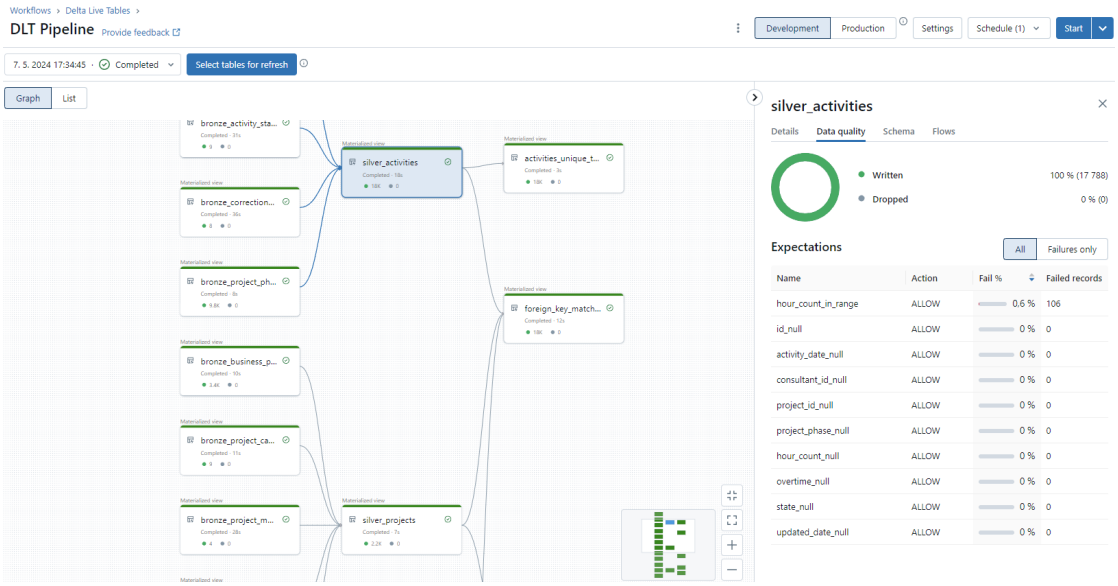
Jde tedy o páry `"<NAZEV_PRAVIDLA>":"<DEFINICE_PRAVIDLA>"`, kde definice jsou jednoduché SQL skripty, které je možné aplikovat na jednotlivé záznamy. Pokud chceme, aby pipeline v případě odhalení nedostačující datové kvality selhala, nebo zahodila nevhodné záznamy (výše zmíněné dlt.expect_all totiž pouze uloží varování a spočítá nevyhovující záznamy), nabízí se použití `@dlt.expect_all_or_drop()`, případně verze `@dlt.expect_all_or_fail()` pro zastavení celé pipeline v případě nevyhovujícího záznamu.

Nejsou zde žádná předdefinovaná generická pravidla, nebo testy. Definice vyžadovaných pravidel je přenechána uživatelům. Nejjednodušším a základním testem, který je v ukázce 4.7 je test na přítomnost hodnoty ve sloupci pro daný záznam. Jde o velice častý a jednoduchý test. Pro zakomponování ostatních testů datové kvality, které byly pro účely práce (vytvoření srovnatelných stage vrstev v DLT a dbt) potřebné, tedy jmenovitě testy na unikátnost hodnot některých sloupců, nebo návaznost na jinou tabulku, neexistuje prozatím u DLT přímá a jednoduchá cesta. Nejjednodušší možností je využití kombinace dočasné tabulky a aplikace potřebných testů na tuto dočasnou tabulku. Nevýhodou je větší počet tabulek v grafu závislostí DLT, ve výsledném schématu se však dočasné tabulky neobjeví. S tímto způsobem je možné otestovat jak existenci cizích klíčů v jejich hlavní tabulce, tak i unikátnost jednotlivých hodnot ve sloupci. V ukázce 4.8 jsou dvě ověření identifikátorů z cizích tabulek. Testu je docíleno využitím levého vnějšího spojení (`LEFT OUTER JOIN`), které v případě neexistence vhodného záznamu v druhé tabulce napojí původní záznam na záznam prázdný. V případě neexistujících hodnot je poté možné tuto skutečnost jednoduše zjistit s pomocí předpokladu `EXPECT (<COLUMN> IS NOT NULL)`. S pomocí klíčového slova `COMMENT` lze přidat popis tabulky, který se chová stejně jako `comment=` u vytváření dlt tabulek s pomocí Pythonu.

Výpis kódu 4.8 Pomocná dočasná tabulka pro kontrolu ID odkazujících na jiné tabulky. Kontroluje se, zda takové ID v příslušné tabulce opravdu existuje. Kontrola probíhá u sloupců *consultant_id* a *project_id* v tabulce *silver_activities*.

```
CREATE TEMPORARY LIVE TABLE foreign_key_match_test(
  CONSTRAINT match_consultant_id EXPECT (foreign_consultant_id IS NOT NULL),
  CONSTRAINT match_project_id EXPECT (foreign_project_id IS NOT NULL)
) COMMENT "Foreign key TEST" AS
SELECT
  sa.id,
  sc.id as foreign_consultant_id,
  sp.id as foreign_project_id
FROM
  LIVE.silver_activities sa
  LEFT OUTER JOIN LIVE.silver_consultants sc ON sa.consultant_id = sc.id
  LEFT OUTER JOIN LIVE.silver_projects sp ON sa.project_id = sp.id
```

Po dokončení procesu DLT pipeline je možné získat podrobný přehled výsledků testů pro každou tabulku. Tento přehled zahrnuje informace o počtu úspěšných a neúspěšných testů, stejně jako konkrétní záznamy, které nesplnily stanovená kritéria testů. Kromě toho je k dispozici kruhový diagram, který vizualizuje poměr mezi úspěšnými a neúspěšnými testy. Na obrázku 4.6 je zobrazeno webové rozhraní, které je vývojářům k dispozici po skončení DLT úlohy. V levé části je automaticky generovaný graf závislostí jednotlivých tabulek, včetně dočasných tabulek sloužící pouze pro testy datové kvality. V pravé části je možné získat detailnější informace o konkrétní tabulce a některých proběhlých testech, případně jejich úspěšnosti. Je zde důležité zmínit, že jde o pouze o některé testy a například kontrola unikátnosti klíčů probíhá v dočasné tabulce *activities_unique_test* (na obrázku 4.6 vpravo od *silver_activities*). Pro výsledky těchto testů je tedy nutné si rozkliknout i detaily této dočasné tabulky.



Obrázek 4.6 Delta Live Tables pipeline webové rozhraní s automaticky generovaným grafem závislostí.

Celkový proces zpracování dat s DLT umožňuje uživatelům přesně monitorovat stav dat a identifikovat potenciální problémy v datech. Poskytování přehledných statistik o kvalitě dat a výsledcích testů je klíčové pro efektivní správu datových toků a zajištění integrity a spolehlivosti datových souborů. Díky této funkcionalitě je možné rychle reagovat na problémy a provádět nezbytné úpravy nebo opravy, aby byla zajištěna konzistence a kvalita dat v databázích.

4.6.4 Historizace

Pro velká datová jezera existuje v Databricks Delta Time Travel, tedy takzvané cestování časem v Delta tabulkách. Nejedná se však o typickou metodu historizace, například již zmiňovaný Snapshotting nebo SCD2 (viz. kapitola 2.3). Namísto toho slouží spíše jako krátkodobá záloha dat s možností je obnovit při jejich poškození nebo smazání. Ve výchozím nastavení je délka uchování záznamů 7 dnů a jde přenastavit na 30 dnů. Také chybí možnost přímo se dotazovat na jednotlivé změny v datech mezi verzemi.

Pro využití Delta Time Travel je při načítání dat z externího zdroje (v případě této práce je zdrojem úložiště S3 AWS) potřeba vytvořit tabulky ve formátu delta. Následně je možné buď vytvořit DLT tabulky bronzové vrstvy, nebo za bronzovou vrstvu považovat již vytvořené delta tabulky a využít je jako zdroj pro transformace do stříbrné vrstvy. Při vytváření tabulky je možné specifikovat, že se má uložit tomto formátu s pomocí klíčové fráze `USING DELTA`. Při načítání dat do Unity Katalogu v kapitole 4.4 se tabulky vytvořily v tomto formátu, aniž by došlo k explicitní specifikaci. To je způsobeno tím, že veškeré vytvořené tabulky v Databricks mají delta formát nastavený jako výchozí. Každá operace provedená na tomto formátu tabulek je zaznamenána a verzována.

Různé verze jsou zpřístupnitelné dvěma způsoby. Prvním je použití časového razítka (nebo jako časový string ve formátu YYYY-MM-DD). Druhým je přístup s pomocí unikátního číselného identifikátoru verze. Verzování umožňuje zobrazit historii tabulek, nebo vrátit tabulku do starší verze. Verzování delta tabulek podporuje i příkaz `VACUUM`, který smaže tabulky, které jsou verzovány déle než je nastavená doba uchování. Tuto dobu je potřeba nastavit pro každou tabulku zvlášť dle jejího typu. U některých tabulek je dlouho uchovávaná historie nutná, u některých

to naopak není potřeba. Toto omezení pro délku uchovávání starých záznamů zajistí, že nebude nutné platit vyšší poplatky za úložiště dat kvůli vysokému objemu uložených dat.

4.6.5 Možnosti automatizace

Pro automatizaci úloh se v případě DLT využívá přímo Databricks Workflows. Pokud je splněn předpoklad, že pro vyvíjení již vznikla minimálně jedna DLT pipeline (která je pro spouštění komplexnějšího kódu s využitím knihovny dlt potřeba), pak je automatizace pipeline s ostatními procesy (nebo ostatními úlohami) velmi jednoduchá a spočívá ve vytvoření úlohy, napojení jednotlivých úkonů na sebe (například načtení dat z externího zdroje a následně DLT pipeline transformace do stříbrné vrstvy). Takto nastavenou úlohu je možné spouštět manuálně, v konkrétní nastavený čas, v případě nového souboru na konkrétním úložišti, nebo průběžně. Průběžný mód je spouštěn opakovaně ihned po dokončení předchozí úlohy.

Pro provádění samotné úlohy lze také automatizovat proces spouštění. V této situaci je možnost nastavit časové spouštění. Úloha se automaticky spustí v určený den a čas a připojí se k přiřazenému clusteru. Tato automatizace výrazně zlepšuje efektivitu správy úloh a umožňuje optimalizovat využití zdrojů.

4.6.6 AI asistent

Vývojové prostředí Databricks poskytuje ve webovém rozhraní integrovaný AI chat ve formě asistenta umožňujícího detekovat a opravovat chyby v kódu, poskytovat uživatelům významnou pomoc při úpravách a optimalizaci jejich programovacích projektů.

Jedním z klíčových aspektů této technologie je možnost přizpůsobit kód podle specifických požadavků uživatele. Asistent umožňuje nejen přepsat kód tak, aby vyhovoval novým potřebám, ale také vysvětlí určité funkce nebo syntaxi, což je užitečné zejména pro začátečníky v prostředí a speciální syntaxi Databricks. Tímto způsobem se zvyšuje přístupnost a umožňuje novým vývojářům lépe porozumět kódu a jeho struktuře.

Další významnou vlastností tohoto asistenta je schopnost navrhovat optimalizace a zjednodušení kódu. Tím může zlepšit celkovou efektivitu a výkon softwarového projektu. Uživatelé mají možnost přijmout navrhované změny nebo provést vlastní úpravy podle svých potřeb a preferencí.

Nicméně, je třeba mít na paměti, že i přes výhody, které AI chat asistent poskytuje, stále existují limity jeho schopností. Někdy může selhat v identifikaci složitých chyb v kódu nebo nedokáže zcela pochopit požadavky vývojáře. Výjimkou není ani situace, kdy asistent doporučí jistý postup s pomocí syntaxe, která v Databricks ani jinde neexistuje. Na doplňující otázku, zda například takové klíčové slovo existuje pak asistent přizná neexistenci předchozího navrhovaného řešení a zneplatní tak svoji předchozí odpověď. Je tedy důležité používat tuto technologii s uvážením a mít na paměti, že konečná odpovědnost za kvalitu a funkčnost kódu leží stále na vývojáři a zatím se nelze zcela spolehnout na podobné technologické inovace, i když jsou oficiální podporovanou součástí prostředí Databricks.

4.7 DLT-META

Tento metadata-driven (zaměřený na metadata) framework, který pomáhá s automatizací pipeline pro bronzové a stříbrné tabulky (tedy stage vrstvu), je možné využít jakožto experimentální neoficiální funkcionalitu Databricks a není v praktické části zahrnut. Jelikož se jedná o neoficiální framework, má omezenou podporu a malou uživatelskou základnu. I přes dokumentaci, popisující nastavení a první spuštění, se nepodařilo pro účely bakalářské práce framework zprovoznit. Vzhledem k primárnímu zaměření této práce na jiné frameworky nebylo cílem studie implementovat

stage vrstvu i v tomto frameworku, je však důležité zmínit jeho existenci a zmínit způsob jeho integrace do pracovního postupu.

K nastavení je možné využít Databricks Labs CLI (Command Line Interface), Python wheel job, nebo Notebook. Zvolený způsob dále volá DLT-META API, které už se stará o zbytek. Na vývojáři je vytvoření souborů, které budou do API poslány. Tyto soubory musí být následně uloženy na Databricks v úložišti DBFS (Databricks File Storage). Přístup do úložiště není ve výchozím stavu povolen, a jelikož nebylo pro práci s ostatními frameworky použito, musí se nejprve v nastavení povolit. Po zpřístupnění DBFS se v Unity Katalogu objeví nové tlačítko pro přístup do úložiště.

Konfigurační soubory, které je nutné do DBFS nahrát, jsou tři: *Onboarding*, *Data_quality_rules* a *Silver_transformations*. Všechny soubory mají formát JSON. První soubor je nejdůležitější, obsahuje informace o příchozích a odchozích metadatech, tedy o formátu načítaných dat, názvech schémat a tabulek, které se mají použít a další různá nastavení. Druhý soubor obsahuje kontrolu datové kvality, tedy seznam testů, které mají být provedeny, ve formátu DLT expectations. Poslední soubor pro každou tabulku obsahuje logické schéma transformací, kterými by data měla v rámci přechodu z bronzové do stříbrné vrstvy projít.

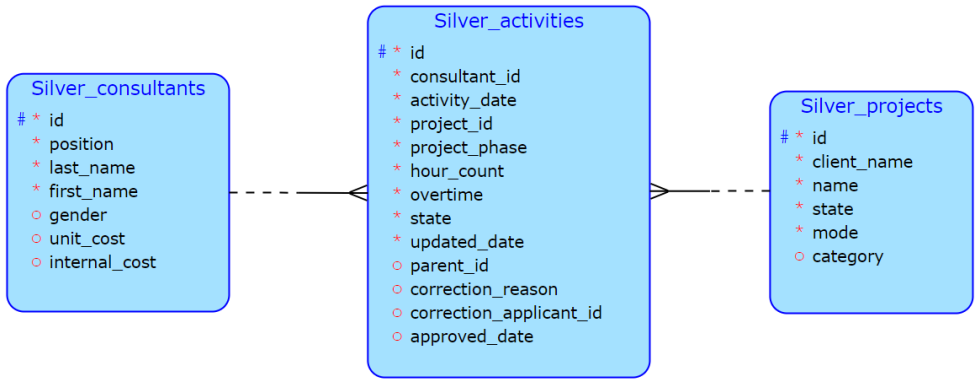
Při samotném vývoji je tedy potřeba udržovat pouze zmíněné tři soubory aktualizované a o vykonání zbytku procesů v bronzové a stříbrné vrstvě se postará samotný framework. To má své výhody především u velkého počtu tabulek, řádově u stovek až tisíců. Pro menší datasety může být míra benefitů méně výrazná.

Tento framework vznikl v březnu roku 2023 a jeho momentální verze je 0.0.7 (k datu 24.4.2024). Podpora Unity Katalogu a CLI byla přidána začátkem roku 2024 ve verzi 0.0.5, zatímco verze 0.0.6 a 0.0.7 přinesly kontrolu datové kvality pro tabulky ve stříbrné vrstvě a dokumentaci pro DLT-META CLI, společně s opravami chyb. Jedná se o stále se vyvíjející framework v počátečních fázích, na kterém se pracuje, jak sami vývojáři uvádí, „jak čas dovolí“. Lze očekávat že se s dalšími verzemi DLT-META bude stále vylepšovat a stane se v budoucnu lepším frameworkem z hlediska přehlednosti, jednoduchosti na nastavení i práci s ním, včetně zapojení větší podpory ostatních součástí platformy Databricks.

Kapitola 5
Zhodnocení výsledků

Kapitola obsahuje zhodnocení výsledků praktické části z kapitoly 4. V úvodu je rozebráno konceptuální schéma tabulek, které jsou výstupem stage vrstvy z praktické části. Následně dochází ke srovnání vývoje datového skladu klasickým způsobem, dbt a DLT z několika různých aspektů vývoje, jako je rychlost, flexibilita a náročnost. Na porovnání je nahlíženo jako na 3 různé způsoby vývoje datového skladu. Prvním je klasický způsob (bez specifikace konkrétního použitého softwaru), druhým Databricks a DLT a třetím je Databricks propojený s dbt. Porovnání frameworků dbt a DLT probíhá slovně v kapitole 5.2 a pro lepší přehlednost je následně srovnání znázorněno v tabulce 5.1.

5.1 Vytvořená stage vrstva



Obrázek 5.1 Konceptuální schéma popisující tabulky ve vzniklé stage (stříbrné) vrstvě a jejich vzájemné vztahy. Tvůrce: autor, použitý software: [43].

Pro obě implementace stage vrstvy, které vznikly v rámci kapitoly 4 byly pro zachování stejných podmínek z hlediska náročnosti jednotlivých procesů pro vývojáře použity stejné transformace a testy v rámci testové kvality. Z původních 12 tabulek (8 číselníků z části *manual* a 4 tabulky ze záznamů z části *profis*) v bronzové vrstvě (kde jsou v neupravené, původní podobě) vznikly celkem 3 tabulky. První, **silver_consultants**, obsahuje informace o jednotlivých konzultantech (zaměstnancích) a to včetně jejich finančního ocenění obsaženého v atributech *unit_cost* a *internal_cost*. Druhou tabulkou jsou **silver_projects**, kde atributy obsahující identifikátory do číselníků byly nahrazeny obsahem daných číselníků, tedy místo identifikátoru pro stav projektu

je nyní v tabulce přítomen slovně definovaný stav, ve kterém se projekt nacházel v době záznamu o projektu v poslední vzniklé tabulce v rámci stage vrstvy, **silver_activities**. Ta je, stejně jako v původním konceptuálním schématu 4.1, ústřední tabulkou, enkapsulující veškeré výkazy aktivit konzultantů. Konceptuální schéma pro výstupní tabulky je znázorněno na obrázku 5.1.

Testy datové kvality, které byly v rámci tvoření stage vrstvy prováděny, byly nastaveny na mód varování, aby bylo možné názorně je ukázat (například v obrázku 4.6). Tyto testy odhalily mimo jiné 106 nevyhovujících hodnot v sloupci *hour_count* tabulky *silver_activities*, kdy byly nalezeny hodnoty mimo povolený interval. Z nastavení procesů ve firmě ABC je povoleno mít v tomto sloupci pouze hodnotu mezi 0 a 8 (může se jednat i o neceločíselnou hodnotu). V testování datové kvality dosáhly oba frameworky stejných výsledků při odhalování nedostatků v zdrojových datech.

5.2 Porovnání frameworků

V konečném zhodnocení dojde k porovnání 3 způsobů tvorby stage vrstvy. Výchozím způsobem tvorby je klasický vývoj datového skladu on-premise, dalšími způsoby jsou DLT v Databricks a framework dbt, které zastupují cloudová řešení a můžou být s klasickým přístupem srovnávány buď individuálně, nebo jako cloudové řešení s podobnými vlastnostmi, které jim dává společný základ na platformě Databricks. Porovnány budou z hlediska rychlosti vývoje, flexibility (možnosti reagovat na případné modifikace), křivky učení a celkové zasazení do kompletního datového skladu, tedy jednoduchost vytváření dalších vrstev a jejich vzájemného propojení.

5.2.1 Rychlost vývoje

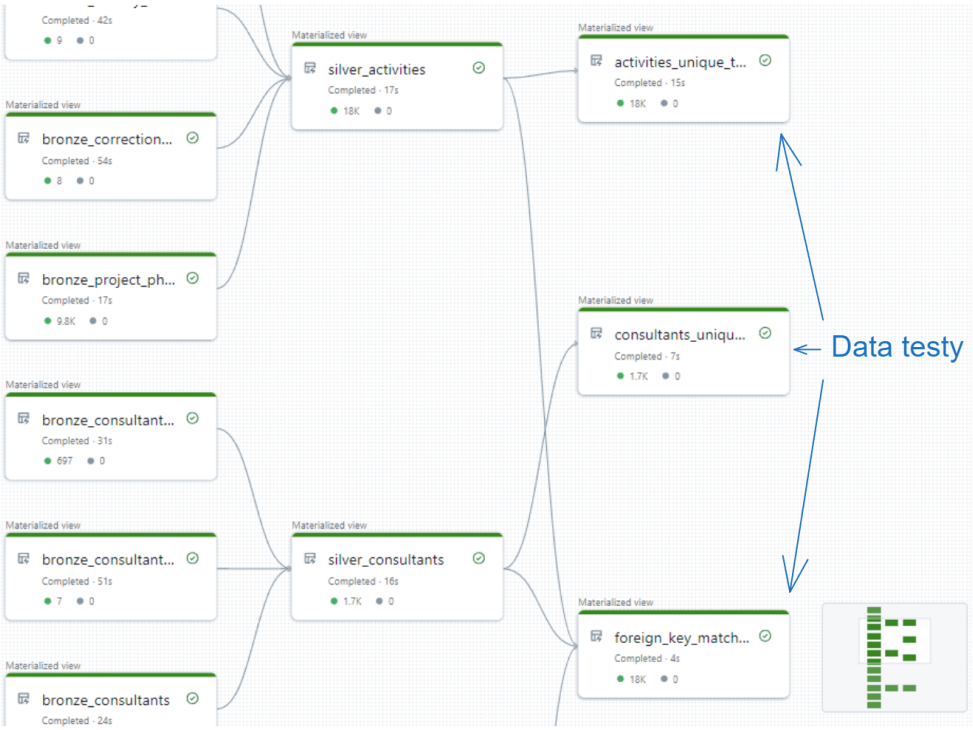
Rychlost vývoje se dá rozdělit do dvou kategorií, a to na přípravu zázemí pro datový sklad a na samotný vývoj. Příprava zázemí pro datový sklad on-premise zahrnuje nákup veškerého hardware a software, jeho nastavení a následné udržování a může způsobit zpomalení vývoje v případě, že je potřeba sklad do budoucna škálovat. Oproti tomu cloudové řešení v Databricks vyžaduje pouze nastavení samotné platformy (případně další platformy, která bude hostovat Databricks Workspace a bude poskytovat úložiště dat, jako například Amazon AWS nebo Microsoft Azure) a škálování je možné dle potřeby bez větší režie.

Při samotném vývoji je však u cloudových řešení potřeba počítat s jistým zpožděním ze strany clusterů. Vývojář tak stráví větší procento času pouze na čekání, i když se nutně nemusí jednat o dlouhé čekací doby. I tak ale dochází k jistému zpoždění od odeslání dotazu k výpočetní jednotce a po delší neaktivitě případně k nutnosti cluster znovu nastartovat, což je nevýhoda oproti on-premise datovém skladu.

Využití frameworků, které jsou vývojářům u jednotlivých přístupů k dispozici, může také rychlost velmi ovlivnit. DLT i dbt mají svoje silné stránky, ve kterých excelují, zároveň se ale u přímého porovnání v jednotlivých kategoriích ukážou i nedostatky, které v daném frameworku pro konkrétní kategorii chybí oproti druhému frameworku.

Příkladem toho je malé pokrytí testy kvality dat u DLT. Testy samotné jsou dobré a ulehčují práci, v porovnání s dbt však chybí široká nabídka vestavěných data testů, které je možné využít bez vlastní implementace. Mezi takové spadají testy na kontrolu unikátnosti hodnot v sloupci a provázanost hodnot sloupce s hodnotami sloupce z jiné tabulky (namapování identifikátoru na identifikátor v jiné tabulce). Chybějící testy využitelné „out-of-the-box“ není sice těžké implementovat ze strany vývojářů, problémem avšak v tomto případě je, že momentální způsoby, které se k tomu využívají, způsobují zanesení grafu závislostí tabulek (ten je vytvářen automaticky při spuštění DLT pipeline) spoustou virtuálních tabulek (pohledů, též používaným pojmem je *VIEW*), které slouží pouze k testování, nenesou v grafu závislostí žádnou významnou hodnotou a efektivně tak tvoří šum. Virtuální tabulky vznikající v grafu závislostí jsou znázorněny na obrázku 5.2. Pokud jsou navíc testy prováděny na složitějším spojení více tabulek a odhalí

se chyba v některém záznamu, je velmi obtížné tento záznam v spojené tabulce namapovat na původní záznam v původní tabulce a zjistit tak, kde je původ chybných dat. Kromě menšího zastoupení předpřipravených data testů navíc v DLT naprosto chybí jakékoliv unit testy na kontrolu korektnosti kódu.



Obrázek 5.2 Ukázka virtuálních tabulek vytvářených pro účely data testů v lineage grafu závislostí u DLT pipeline.

Dalším bodem k srovnání je historizace. Jak již bylo zmíněno, dbt má předimplementovanou historizaci s pomocí SCD2, zatímco u DLT tato možnost chybí a nahrazuje ji pouze částečně Delta Time Travel, sloužící spíše jako krátkodobá záloha než dlouhodobá historizace dat. SCD2 je sice možné pro DLT implementovat ručně, může to však být mnohonásobně těžší než využít již hotovou implementaci jako v případě dlt. V dokumentaci Databricks je popsán postup pro zapojení SCD2 historizace do celé workflow projektu. Nabízí ukázkový kód pro vygenerování testovacích dat a skript pro převedení dat do SCD2 formátu. Dále je zde popsáno, že pro zapojení do vlastní pipeline je potřeba vytvářet streaming tabulky (streaming tables) a následně použít `APPLY CHANGES INTO` k určení zdroje, klíče a sekvence pro přenos změn. [48]

Velkým nedostatkem je také možnost DLT zapisovat pouze do jednoho schématu. To může být značnou nevýhodou při potřebě data separovat do více databází, resp. schémat. To tedy znamená že s jednou pipeline není možné dosáhnout načtení dat ze schématu určeného pro bronzovou vrstvu (v medailonové architektuře) a po transformaci je uložit do schématu pro stříbrnou vrstvu. V terminologii klasického rozdělení vrstev datových skladů tak nelze pracovat s vrstvou Stage a Core v jedné DLT pipeline a je nutné pro takové operace vytvořit více úloh.

Další zajímavostí, která stojí za zmínku, je chybějící možnost manuálně změnit některé parametry v nastavení clusterů, které budou k provedení DLT pipeline použity. Mezi takové parametry může patřit i verze Sparku a dalších knihoven. Automaticky je zvolena nejnovější verze, což potencionálně může vést k problémům s nekompatibilitou kódu.

Na celkové srovnání frameworků lze nahlédnout v tabulce 5.1. Díky rozsáhlým možnostem

znovupoužitelnosti kódu, testům datové kvality, unit testům, možnostem historizace a dokumentaci generované na základě kódu by pro vývoj stage vrstvy datového skladu byl z časového hlediska a většímu počtu nástrojů, zjednodušujících celkový vývoj, vhodnější framework dbt. Jelikož je ale dbt nástroj pouze pro transformace dat, na rozdíl od DLT není soběstačný a musí být v kontextu celého datového skladu napojen k ostatním fázím zpracování dat, jakými jsou například načtení dat do Databricks ze zdrojových úložišť, což je naopak výhoda pro framework DLT, který je umístěn přímo v Databricks a je tak provázán s ostatními částmi tohoto prostředí bez nutnosti nastavování propojení a správy dalšího prostředí.

5.2.2 Flexibilita a cena

Flexibilita u klasických on-premise řešení je velice nízká. Rozšíření, zmenšení nebo jakékoliv další modifikace výpočetních jednotek je finančně a časově náročné. Využití výpočetních jednotek ve formě, která se u poskytovatelů datového skladu jako služby blíží v podstatě „pronájmu“, je velmi flexibilní a pro časté změny v potřebách výpočetního výkonu nebo úložiště je vhodnější než využívání vlastních clusterů. Pro podniky, které mají jasnou představu o nárocích na clustery a nepotřebují vysokou flexibilitu může být naopak vhodnější klasický on-premise sklad. Faktorem pro rozhodování může v této situaci být také cena vlastních clusterů v porovnání s cenou těch pronajímaných.

Jak bylo již zmíněno na konci kapitoly 5.2.1, dbt je narozdíl od DLT možné využít pouze na transformace dat, neobsahuje žádnou možnost data načítat. To jej značně limituje v možnostech využití, kde je potřeba vždy dbt zařadit do dalších procesů, které obstarají zbývající části datového skladu. DLT naproti tomu dokáže obstarat prakticky všechny části datového skladu, a to v jedné pipeline, kde se závislosti mezi jednotlivými procesy vytvoří automaticky a jsou modifikovatelné v jednom prostředí. Pro nastavení jobu v Databricks stačí pro DLT pouze vytvořit notebooky s kódem a přidat je do jobu, v případě dbt je nutné vytvořit část pipeline individuálně, nastavit závislosti jednotlivých úloh a spustit dbt, které je pro Databricks prakticky externím nástrojem, a jeho napojení na Databricks job nemusí být v závislosti na jeho nastavení triviální úlohou. Větší možnost flexibility nabízí DLT i z hlediska programovacích jazyků, které mohou vývojáři pro transformační logiku využít. Na rozdíl od dbt tak vývojáři mohou využít nad rámec SQL a Pythonu, který se v kontextu dbt začíná objevovat čím dál více, ještě jazyky Scala a Java.

5.2.3 Křivka učení

U všech tří způsobů je velmi podstatná i křivka učení a její strmost. U klasických datových skladů záleží na výběru software a znalostech programů mezi vývojáři. Technologie dbt a DLT jsou v současné době novými alternativami a i když jejich uživatelská základna roste každým dnem, stále jde o frameworky, které nejsou velmi rozšířené. U vývojářů bez zkušenostmi s Databricks a DLT, resp. dbt, je potřeba počítat s úvodním seznámením s prostředím a některými funkcionalitami. V každém případě nejde o komplikované frameworky, takže je křivka učení při seznámení s prostředím a dokumentací poměrně strmá. Vzhledem k tomu, že se DLT a dbt i nadále vyvíjejí a jsou vydávány nové verze, které mohou zásadně ovlivnit celý vzhled a pracovní postupy, je dobré se při jejich používání průběžně vzdělávat o nových verzích, což přidává mírnou průběžnou časovou náročnost.

Tabulka 5.1 Porovnání frameworků DLT a dbt v různých kategoriích.

	DLT	dbt
Testování	Expectations, možnost definovat vlastní testy	Široká nabídka vestavěných testů, možnost definovat vlastní testy
Druhy testů	Unit	Unit, Data
Graf závislostí	Automaticky generován, hrozí zanesení grafu virtuálními tabulkami pro expectations	Automaticky generován, neobsahuje vstupní tabulky, pokud nemají nastavenou historizaci
Historizace	Delta Time Travel	Snapshots (postavené na SCD2)
Programovací jazyky	SQL, Python	SQL, Python
AI asistent	ano	ne
Zapisování	Pouze v rámci jednoho schématu	Do libovolného počtu schémat
Využití	Ve všech procesech datového skladu	Pouze transformační nástroj
Dokumentace	Popisy tabulek v grafu závislostí z dokumentačních komentářů	Generována automaticky z dokumentačních komentářů v kódu
Výpočetní jednotky	Databricks Cluster (Job cluster)	Databricks Cluster (All-purpose cluster nebo SQL Warehouse)
Nastavení clusteru	Nelze konfigurovat, volí nejnovější verzi knihoven	Je možné konfigurovat
Cena	Součást Databricks, žádné další poplatky kromě cloud providera a provoz clusterů	Verze Core zdarma, verze Cloud zdarma pro plán Developer, placené plány Team a Enterprise, k tomu poplatky za cloud providera pro Databricks a provoz clusterů
Nativní GIT integrace	ano (v rámci Databricks)	ano
Spouštění úloh	Periodicky, manuálně, neustále, při načtení nových dat	Periodicky, manuálně, při skončení jiné úlohy, přes volání API

Kapitola 6
Diskuze

Kapitola obsahuje autorův pohled na osobní předchozí zkušenost s frameworkem DLT. Diskutuje možnosti využití webového prostředí pro jednotlivé nástroje v porovnání s lokálním prostředím. Kapitola jako celek slouží jako dovětek zhodnocení praktické části této práce z kapitoly 5.

Před vznikem celé této práce jsem měl předchozí zkušenosti s frameworkem DLT a jeho použitím jako součást datového skladu na platformě Databricks, a osobně jsem do učení se o frameworku dbt šel se skepsí, zda tento nástroj opravdu dokáže DLT konkurovat. Během praktické části se však můj původní názor změnil. Nejen že je dbt schopný DLT konkurovat, ale je schopný ho i v jistých aspektech doplňovat nebo svoji funkcionalitou převyšovat. Především využívání konceptu funkcí v kombinaci s jazykem SQL a výsledná modularita vede vývojáře k psaní kvalitnějšího a přehlednějšího kódu. Velkou nevýhodou je však stále jisté oddělení vývojového prostředí dbt od Databricks.

Výhodou DLT je přímé propojení s Databricks, v porovnání některých funkcionalit, např. kontrolních testů kvality dat, které lze využít „out of the box“, se zde ale nachází značně méně.

V práci byla využita možnost webového prostředí Databricks a dbt Cloud IDE, které jsou pro vývojáře dostupné bez nutnosti nastavení lokálního prostředí. Využít je možné i lokální propojení zvoleného IDE (VSCode, PyCharm, IntelliJ IDEA, Eclipse a další) k přístupu do prostředí Databricks. Je tak umožněno psát kód se zvýrazněním syntaxe v prostředí, které je uživateli známé a pohodlné, není nutné k vývoji využívat notebooky z prostředí Databricks. I tak je však potřeba interagovat s některými prvky právě a pouze přes webové rozhraní (kontrola grafu závislostí, výsledky testů datové kvality, pipeline, administrace prostředí). Pro dbt je možné nastavit lokální verzi dbt Core v prostředí příkazové řádky.

Žádný ze zkoumaných frameworků by se nedal označit za univerzálního vítěze, který by byl pro každý projekt tou nejlepší volbou. Na základě některých vlastností, důležitých pro vývojáře, rozebíraných v kapitole 5, je možné pozorovat a vyhodnotit vhodnost pro konkrétní datový sklad. dbt jakožto transformační nástroj je nutné vždy doplnit o další nástroje pro extrakci a načítání dat do datového skladu, a to je ne vždy na projektu žádoucí. Práce s ním velmi usnadňuje vývoj, ale v některých případech může jeho zapojení do transformací v datovém skladu vést pouze k minimální změně co se náročnosti vývoje, nákladů, nebo rychlosti zpracování dat týče.

Kapitola 7
Závěr

Hlavním cílem práce bylo v rámci případové studie ukázat možné způsoby vývoje datového skladu se zaměřením na stage vrstvu, představit online analytickou platformu Databricks a její framework DLT, metadata driven framework dbt, a poté vytvořit s těmito nástroji na konkrétních datech samotnou stage vrstvu. Následovalo zdokumentování různých aspektů vývoje s pomocí těchto nástrojů, především z obecného hlediska, a porovnání různých výhod a nedostatků jednotlivých přístupů.

Byly vytvořeny dvě varianty stage vrstvy, každá v jednom ze zmíněných frameworků, DLT a dbt. Každá varianta obsahuje všechny funkcionality, očekávatelné od stage vrstvy, jmenovitě načtení dat, transformaci, kontrolu kvality dat s pomocí kontrolních testů a možnost ukládat starší verze dat (historizaci). Veškeré procesy probíhající v rámci stage vrstvy byly taktéž automatizovány, aby bylo možné je jednoduše zařadit do komplexnější pipeline a využít výsledné tabulky pro další vrstvy datového skladu, dovolující případnou škálovatelnost celého řešení.

V rámci porovnání jednotlivých řešení bylo zjištěno, že každý způsob vývoje datového skladu má své silné a slabé stránky v porovnání s ostatními způsoby a žádný není univerzálním vítězem. I přes to však vytváření stage vrstvy v frameworku dbt pro účely této práce působilo nejlepším dojmem z hlediska přehlednosti, možností přizpůsobení vnitřních procesů a logického uspořádání jednotlivých modulů, do kterých jsou procesy stage vrstvy děleny.

Kompletní seznam zdrojových kódů a konfiguračních souborů je součástí bakalářské práce ve formě přílohy. Zdrojové kódy nejsou samy o sobě spustilené bez nastavených prostředí (Databricks Workspace, resp. dbt Cloud). Data, která byla pro vytvoření stage vrstvy použita, nejsou z hlediska jejich původu a neveřejného charakteru součástí příloh.

Je možné, že do budoucna bude framework DLT-META hrát významnější roli mezi frameworky pro práci s metadaty v datovém skladu v prostředí Databricks. I přes jeho menší zastoupení v této práci by se rozhodně neměl v kontextu s DLT a dbt opomínat. Momentálně je stále v rané fázi vývoje, ale není možné vyloučit, že by se v budoucnosti mohl stát klíčovou součástí procesů zaměřených na metadata v datových skladech na platformě Databricks.

Stanovené cíle práce byly splněny, jak v oblasti rešerše relevantní literatury, v praktické realizaci implementace stage vrstvy. Rovněž byla splněna část zabývající se zhodnocením a porovnáním zkoumaných přístupů. Zhodnocení výsledků praktické části má své využití jako návod pro vývojáře, kteří zvažují modernizaci datového skladu do oblasti cloudu a staví svá rozhodnutí na základě specifických požadavků s ohledem na nejvhodnější framework pro jejich konkrétní potřeby.

Bibliografie

1. MARITZ, Paul. *Paul Maritz Quotes And Sayings* [online]. 2024. [cit. 2024-01-02]. Dostupné z: https://quotlr.com/author/paul-maritz.
2. RAJ, Phani; JAISWAL, Vinod. *Azure databricks cookbook: accelerate and scale real-time analytics solutions using the apache spark-based analytics service*. Birmingham; Mumbai; Packt, 2021. ISBN 9781789809718;1789809711;
3. What is Databricks and what's it used for? In: *Cuusoo* [online]. 2022 [cit. 2024-01-04]. Dostupné z: https://www.cuusoo.com.au/learn/guides/what-is-databricks-and-whats-it-used-for.
4. CHEDEAU, Christopher. *Excalidraw* [software]. 2023. Ver. 0.17.3 [cit. 2024-04-05]. Dostupné z: https://excalidraw.com.
5. FREEPIK COMPANY S.L. *Flaticon* [software]. 2024. [cit. 2024-04-05]. Dostupné z: https://www.flaticon.com.
6. Data Lakehouse. In: *Databricks* [online]. 2024 [cit. 2024-01-10]. Dostupné z: https://www.databricks.com/glossary/data-lakehouse.
7. What is a data lakehouse? In: *IBM* [online]. 2021 [cit. 2024-01-04]. Dostupné z: https://www.ibm.com/topics/data-lakehouse.
8. Are you ready to scale your Data and AI initiatives? How will you scale your security? In: *Databricks* [online]. 2018 [cit. 2024-01-10]. Dostupné z: https://www.databricks.com/blog/2018/11/20/are-you-ready-to-scale-your-data-and-ai-initiatives-how-will-you-scale-your-security.html.
9. Databricks Data + AI Summit 2023 Keynote Recap: LakehouseIQ, Delta Lake 3.0, and More! In: *Monte Carlo* [online]. 2023 [cit. 2024-01-11]. Dostupné z: https://www.montecarlodata.com/blog-databricks-data-ai-summit-2023-keynote-recap/.
10. Introducing LakehouseIQ: The AI-Powered Engine that Uniquely Understands Your Business. In: *Databricks* [online]. 2023 [cit. 2024-01-11]. Dostupné z: https://www.databricks.com/blog/introducing-lakehouseiq-ai-powered-engine-uniquely-understands-your-business.
11. Types of Clusters in Databricks. In: *Spark By Examples* [online]. 2023 [cit. 2024-01-09]. Dostupné z: https://sparkbyexamples.com/spark/types-of-clusters-in-databricks/.
12. Databricks Pricing 101: A Comprehensive Guide (2024). In: *Chaos Genius* [online]. 2024 [cit. 2024-01-09]. Dostupné z: https://www.chaosgenius.io/blog/databricks-pricing-guide/.

13. Databricks pricing calculator. In: *Databricks* [online]. 2024 [cit. 2024-01-09]. Dostupné z: https://www.databricks.com/product/pricing/product-pricing/instance-types.

14. MALINOWSKI, Elzbieta; ZIMÁNYI, Esteban. *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. ISBN 3540744053; 9783540744054; 3540744045; 9783540744047;

15. *Metadata* [online]. Merriam-Webster.com Dictionary, 2024 [cit. 2024-04-07]. Dostupné z: https://www.merriam-webster.com/dictionary/metadata.

16. DUVAL, Erik. *Metadata Standards: What, Who Why.* Sv. 7. Journal of Universal Computer Science, 2001. Dostupné z DOI: 10.3217/jucs-007-07-0591.

17. RILEY, Jenn. *Understanding Metadata: What is Metadata, and What is it For?* National Information Standards Organization (U.S.), 2017. ISBN 978-1-937522-72-8.

18. VETTERLI, Thomas; VADUVA, Anca; STAUDT, Martin. *Metadata standards for data warehousing: open information model vs. common warehouse metadata*. Sv. 29. New York, NY, USA: Association for Computing Machinery, 2000. Č. 3. ISSN 0163-5808. Dostupné z DOI: 10.1145/362084.362138.

19. MEDINA, Enrique; TRUJILLO, Juan. *Symposium on Advances in Databases and Information Systems*. A Standard for Representing Multidimensional Properties: The Common Warehouse Metamodel (CWM). 2002. Dostupné také z: https://api.semanticscholar.org/CorpusID:6672685.

20. VETTERLI, Thomas. *A Comparison of OIM with CWM*. Information Systems Research, Swiss Life, Zürich, 1999. Dostupné z DOI: https://doi.org/10.1145/362084.362138.

21. KIMBALL, Ralph; MARGY, Ross. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling*. Third Edition. John Wiley Sons Inc, 2013. ISBN 978-1118530801.

22. KIMBALL, Ralph. *The data warehouse lifecycle toolkit: Expert methods for designing, developing, and deploying data warehouses*. Second Edition. New York: Wiley-Academy, 1998. ISBN 9780471255475; 0471255475;

23. PALMER, Matt; MIRJANKAR, Anish. The Complete Guide to Data Staging. In: *Zuar* [online]. 2022 [cit. 2024-04-05]. Dostupné z: https://www.zuar.com/blog/complete-guide-to-data-staging/.

24. INMON, William H.; LINSTEDT, Daniel. *Data Architecture: A Primer for the Data Scientist: Big Data, Data Warehouse and Data Vault*. Academic Press is an imprint of Elsevier, 2015. ISBN 9780128020449; 9780128020913; 012802044X; 0128020911;

25. VYVADIL, Martin. Vybudování datového skladu. In: *Diplomová práce (Ing.)* Praha: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky, 2023.

26. INMON, William H. *Building the data warehouse*. Second Edition. New York: John Wiley Sons, 1996. ISBN 0471141615; 978-0471141617.

27. JARKE, Matthias; LENZERINI, Maurizio; VASSILIOU, Yannis; VASSILIADIS, Panos. *Fundamentals of data warehouses*. Second Edition, rev. and extend. Springer, 2013. ISBN 3540653651; 9783540653653;

28. Build an end-to-end data pipeline in Databricks. In: *Databricks* [online]. 2023 [cit. 2024-01-19]. Dostupné z: https://docs.databricks.com/en/getting-started/data-pipeline-get-started.html.

29. Data ingestion. In: *Databricks* [online]. 2023 [cit. 2024-02-09]. Dostupné z: https://www.databricks.com/product/data-ingestion.

30. CYR, Cameron; DORSEY, Dustin. *Unlocking dbt: Design and Deploy Transformations in Your Cloud Data Warehouse*. Apress, 2023. ISBN 978-1484296998;1484296990.

31. What is analytics engineering? In: *dbt* [online]. 2024 [cit. 2024-01-15]. Dostupné z: https://www.getdbt.com/what-is-analytics-engineering.
32. MCFARLAND, Jade; AGBOOLA, Tobey. Ask the Experts: The Rise of DBT. In: *Inawisdom* [online]. 2022 [cit. 2024-04-06]. Dostupné z: https://inawisdom.com/ask-the-experts-the-rise-of-dbt/.
33. About ref function. In: *dbt* [online]. 2024 [cit. 2024-01-16]. Dostupné z: https://docs.getdbt.com/reference/dbt-jinja-functions/ref.
34. dbt cloud vs dbt core: a quick comparison. In: *castordoc* [online]. 2023 [cit. 2024-01-16]. Dostupné z: https://www.castordoc.com/blog/dbt-cloud-vs-dbt-core.
35. Resource configs and properties: column$_t$*ypes*. In: *dbt* [online]. 2024 [cit. 2024-04-29]. Dostupné z: https://docs.getdbt.com/reference/resource-configs/column_types.
36. How do I define a column type? In: *dbt* [online]. 2024 [cit. 2024-04-29]. Dostupné z: https://docs.getdbt.com/faqs/Models/specifying-column-types.
37. Resource configs and properties: meta. In: *dbt* [online]. 2024 [cit. 2024-04-29]. Dostupné z: https://docs.getdbt.com/reference/resource-configs/meta.
38. dbt: Supported data platforms. In: *dbt* [online]. 2024 [cit. 2024-01-16]. Dostupné z: https://docs.getdbt.com/docs/supported-data-platforms.
39. Quickstart for dbt Cloud and Databricks. In: *dbt* [online]. 2024 [cit. 2024-01-16]. Dostupné z: https://docs.getdbt.com/guides/databricks?step=4.
40. Databricks: Connect to dbt Cloud manually. In: *Databricks* [online]. 2024 [cit. 2024-01-16]. Dostupné z: https://docs.databricks.com/en/partners/prep/dbt-cloud.html#connect-to-dbt-cloud-manually.
41. What, exactly, is dbt? In: *dbt* [online]. 2017 [cit. 2024-03-18]. Dostupné z: https://www.getdbt.com/blog/what-exactly-is-dbt.
42. ZÝKA, Ondřej. *BP Víťa Hušek - ProFis Data* [elektronická pošta]. 2024. 28.3.2024 11:02, ondrej.zyka@profinit.eu.
43. ČVUT, FIT. *Portál BI-DBS* [software]. 2024. Ver. 9.1.2 [cit. 2024-04-05]. Dostupné z: https://dbs.fit.cvut.cz.
44. OLDHAM, Ed. Medallion Architecture: What is it? In: *Advancing Analytics* [online]. 2023 [cit. 2024-04-05]. Dostupné z: https://www.advancinganalytics.co.uk/blog/medallion-architecture.
45. Add snapshots to your DAG. In: *dbt* [online]. 2024 [cit. 2024-03-25]. Dostupné z: https://docs.getdbt.com/docs/build/snapshots.
46. Add data tests to your DAG. In: *dbt* [online]. 2024 [cit. 2024-03-25]. Dostupné z: https://docs.getdbt.com/docs/build/data-tests.
47. Run an update on a Delta Live Tables pipeline. In: *Databricks* [online]. 2024 [cit. 2024-03-26]. Dostupné z: https://docs.databricks.com/en/delta-live-tables/updates.html.
48. Simplified change data capture with the APPLY CHANGES API in Delta Live Tables. In: *Databricks* [online]. 2023 [cit. 2024-04-28]. Dostupné z: https://docs.databricks.com/en/delta-live-tables/cdc.html.

Obsah příloh