IT a anatomie firmy

(dbt)

(pracovní dokument)

Matěj Novák

VŠE Praha, 2025

[1] dbt (data build tool) a jeho přínosy ***(formy dbt, funkcionality dbt)***
[2] Technické charakteristiky dbt ***(formáty souborů, struktura dbt projektu, dbt příkazy, orchestrace)***
[3] Aplikace dbt při stavbě DWH pro batchové zpracování ***(založení dbt projektu, nastavení git repozitáře a projení s dbt projektem)***

Poznámky k textu:

- ***Účelem*** dokumentu je vymezit ***hlavní charakteristiky nástroje dbt*** a možnosti a postupy jeho využití.
- ***Dokument představuje doplnění*** k základnímu dokumentu orientovanému na podnikovou analytiku: [Podniková analytika]

Obsah

1. dbt (data build tool) a jeho přínosy

dbt je ***nástroj určený ke stavbě datových skladů*** a pokrývá ***několik důležitých faktorů*** při jejich stavbě:

- vývoj (SQL ulehčený o DDL),
- zajištění datové kvality přímo v projektu,
- zajištění dokumentace přímo v projektu,
- verzování kódu,
- modularita,
- používání balíčků.

Obrázek 1-1 Využití dbt při stavbě datových skladů (Byrum 2022)

1.1 Formy dbt

Dvě základní formy nástroje dbt, které jsou využívány v rámci moderních datových infrastruktur slouží k transformaci dat a správě datových pipeline, ale liší se v možnostech, funkcionalitách a přístupu k infrastruktuře. Jedná se o ***2 formy***:

- dbt Core,
- dbt Cloud.

dbt Cloud funguje jako cloudová nástavba nad dbt Core, tudíž pro analytiky poskytuje určité služby navíc.

1.1.1 dbt Core

Dbt Core je ***open source CLI forma*** způsobu práce s dbt. Díky tomu, že je nástroj open source[1], je zdarma a pokrývá ***základní funkcionality pro transformaci dat*** (a stavbu datových skladů), ho může používat v podstatě jakýkoliv analytik se zkušenostmi s SQL.

1.1.2 dbt Cloud

Dbt Cloud je ***placená nadstavba nad dbt Core***. Jeho součástí je i uživatelské rozhraní, dále funkce pro automatizaci ***pipelines (orchestrace), definování environmentálních proměnných***, ***stavbu sémantické vrstvy či data explorer.*** Díky funkcionalitě Data Explorer dostane uživatel ještě mnohem větší vhled do dat, například v podobě DAGu na úrovni sloupců, nikoliv jen finálních tabulek.

Jednou ***z plánovaných funkcí*** je uživatelské rozhraní pro práci s daty pomocí drag & drop. Toto rozhraní bude umožňovat jednoduše vizuálně manipulovat s daty, zatímco na pozadí bude automaticky generován potřebný SQL kód. Díky této funkci se dbt Cloud stane přístupnějším i pro uživatele, kteří

[1] Open source je software, který má veřejně dostupný kód, takže jakýkoliv vývojář může nastudovat jeho funkcionality a případně je podle svých potřeb upravit (Opensource.com [b.r.]).

nejsou zcela zdatní v psaní SQL dotazů, což může přispět k širšímu využití nástroje i mimo technicky zaměřené týmy.

Další významnou výhodou je ***plánovaná schopnost vytvářet grafické rozhraní*** na základě existujícího SQL kódu. Uživatelé tak budou moci vidět vizualizaci svého kódu a v tomto prostředí jej dále upravovat. To umožní lepší pochopení datových procesů a zvýší efektivitu práce tím, že poskytne vizuální náhled na to, jak jednotlivé části SQL dotazů interagují s daty.

1.1.3 Rozdíly dbt Core vs. dbt Cloud

Pro ***pokrytí rozdílů je uvedena tabulka*** a několik nejdůležitějších parametrů pro stavbu datových skladů.

Tab. 1 Porovnání verzí dbt Core vs. dbt Cloud

Parametr	dbt Core	dbt Cloud
Vývoj modelů	✅	✅
Redukce opakování kódu	✅	✅
Dokumentace	✅	✅
Apikace testů	✅	✅
Cena	0 $	100 $/user (2+)
Sémantická vrstva	❌	✅
Column-level DAG	❌	✅
Orchestrace	❌	✅
IDE	❌	✅

dbt Cloud má kromě výše zmíněných samozřejmě i ***další výhody, jako security, audit logging či další***, což lze z praktických zkušeností na úkor vysokého pricing modelu vynechat či nahradit jinou formou. Podle Drewa Benina (2024) si lze ***dbt Core představit jako stavění modelů a testování***, ***kdežto dbt Cloud*** si můžeme ***představit jako zprovoznění funkční samostatné jednotky***.

1.2 Funkcionality dbt

Kapitola pokrývá hlavní funkcionality dbt z širšího, strategického pohledu, s důrazem na jejich přínos pro byznys a efektivitu datových procesů, tedy jak dbt umožňuje zlepšovat týmovou spolupráci, zajišťovat kvalitu dat a optimalizovat procesy.

1.2.1 Git – verzování kódu

Git – správa verzí. *„Správa verzi je systém, který zaznamenává změny souboru nebo sady souborů v průběhu času, a uživatel tak může kdykoli obnovit jeho/jejich konkrétní verzi“* (Chacon 2009, str. 17). Umožňuje (nejen) vývojářům ***sledovat změny v kódu***, pracovat ***na více verzích modelů*** současně a vracet se k předchozím stavům projektu v případě potřeby.

Díky Gitu mohou týmy efektivně ***spolupracovat a přispívat do jednoho dbt projektu***, aniž by docházelo ke konfliktům v kódu. V dbt je Git využíván pro správu verzí modelů, testů, dokumentace a celkového nastavení projektu, což zajišťuje ***transparentnost změn a auditovatelnost*** celého procesu datové transformace.

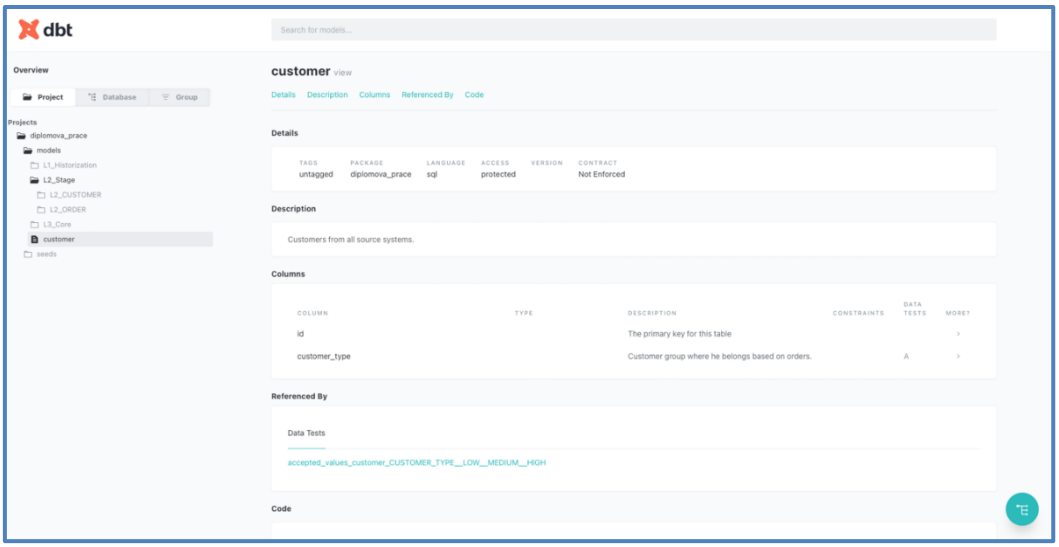
1.2.2 Testování a datová kvalita

V rámci všech procesů, kterými data prochází, mohou být poškozena. Na datovou kvalitu by měl být kladen ***největší důraz***. (Moses et al. 2022, str. 4). Z tohoto důvodu je dobré ***využít dbt testy***. Testování dat přináší nejen klid, ale také jistotu, že klíčové byznysové procesy jsou podporovány kvalitními a spolehlivými daty (Zagni 2023). Právě díky dbt testům může analytik ***podchytit velkou část problémů,*** které předpokládá, že mohou nastat, nebo s nimi již má zkušenost a chce předejít jejich opakování. To nejen snižuje riziko lidské chyby, ale také umožňuje rychleji reagovat na potenciální problémy, čímž se šetří čas a peníze. V momentě, kdy jsou testy nastavené, může s pravidelností zajistit ***mnohem vyšší spolehlivost dat***.

1.2.3 Dokumentace

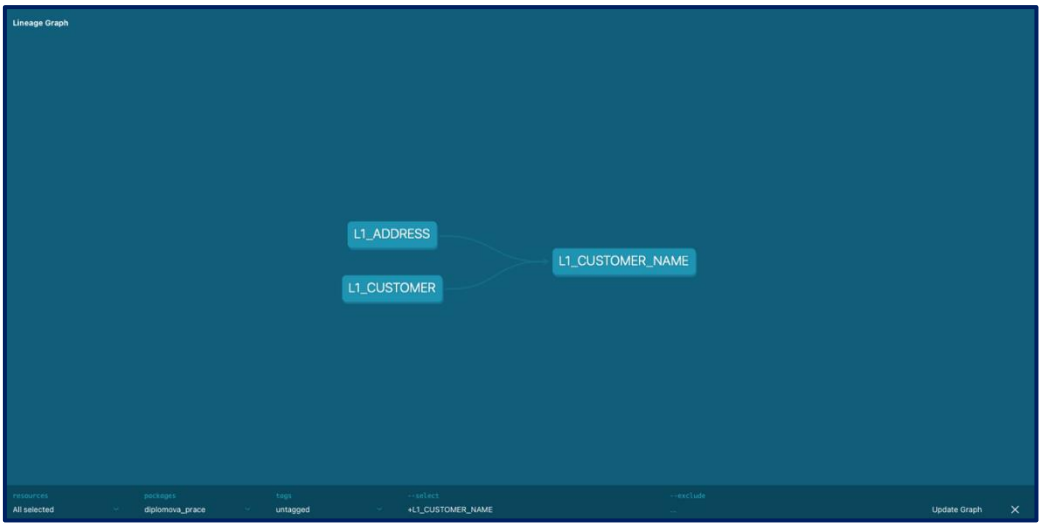
Vzhledem k časté nutnosti velkého tlaku na rychlé dodávání jde dokumentace do pozadí. V tomto případě přináší dbt velkou výhodu - ***dokumentace vedena přímo v projektu, hned vedle kódu*** (Machado a Russa 2023, str. 137). Další výhodou kromě toho, že je dokumentace ***vedena přímo v kódu, je vygenerování webového rozhraní***, kde si uživatel může dokumentaci proklikat a prohlédnout. V dokumentaci lze najít:

- interaktivní DAG2,
- dokumentaci tabulek,
- dokumentaci jednotlivých sloupců v tabulce,
- testy aplikované na tabulku,
- testy aplikované na sloupce,
- další libovolná metadata zadané vývojářem.



Obrázek 1-2 Ukázka vygenerované dokumentace dbt (vlastní zpracování)

[2] Pod pojmem DAG se rozumí přístup k nastavení sekvence úkolů tak, aby plynule navazovaly za sebou a nevznikaly v orchestrací cykly závislostí (Airflow [b.r.]). Tuto sekvenci si lze představit jako úkoly plynoucí zleva doprava (od nejzákladnějších, které společně tvoří jeden velký – složitý), aby úkol na konci sekvence nebyl primárním vstupem pro tuto sekvenci – v takovém případě by vznikl cyklus.



Obrázek 1-3 Ukázka vygenerovaného DAGu v rámci dbt dokumentace (vlastní zpracování)

1.2.4 Redukce opakování kódu

Tato funkcionalita je velmi ***důležitá pro modularitu a redukci chybovosti kódu***. Představte si jasně danou, nelehkou, byznys logiku konverze, kdy musíte počítat rozdíl dvou dat a převést je na roky, měsíce a dny. Tento přepočet se opakuje skoro v každém data martu a vy tuto logiku opakujete několikrát do týdne. S dbt a jeho kombinací Jinja[3] a maker logiku aplikujete na jednom místě a dále ji můžete přepoužívávat napříč projektem – stejně jako funkci v programování.

1.2.5 Dynamický výběr zdrojů a cílové databáze

V rámci vývoje datových skladů výzkum ukazuje, že až 71 % firem používá svoje produkční nebo subset produkčních dat ve vývojovém prostředí, což z pohledu ochrany dat není nutně nejvhodnější přístup z pohledu údržby a ochrany (Ltd a Hilbert 2024).

dbt umožňuje řešení, kdy je ***schopné pro různé typy dat zapisovat do různých databází***, či ***z různých databází čerpat.*** Tato vlastnost je vhodná pro oddělení různých prostředí – například vývojové a produkční. Jednoduchým příkladem řešení může být situace, kdy pro:

- vývojové prostředí využíváme testovací data a také výstupy zapisujeme do databáze s testovacími daty,
- produkční prostředí využíváme produkční data (která jsou uložená na jiném místě než data testovací) a obdobně jako v příkladu výše, výstupy zapisujeme do produkčního prostředí.

1.2.6 Přepoužití kódu napříč databázemi

dbt Labs se zaměřilo i na ***pokrytí nutností migrací***. Vzhledem k tomu, že analytik píše konkrétní SELECT SQL dotaz a o DDL[4] se dbt postará za vás. Migrace mezi databázemi (například z databáze Snowflake do BigQuery) je tak mnohem jednodušší. Je však potřeba myslet na to, že ***ne všechny databázové funkce jsou napříč databázemi stejné.***

Další možností přepoužitím je dbt adaptér, který umožňuje práci s databázemi pomocí metod, které jsou napříč podporovanými databázemi všechny stejné. Pokud budete chtít například vygenerovat základní tabulku se všemi sloupci oproti vaším zdrojovým datům, adaptér vám může pomoci bez ohledu

[3] Jinja je šablonovací jazyk syntaxí podobný Pythonu (Jinja [b.r.]). Právě pomocí Jinjy jsme schopni využívat modularitu v dbt a to například tak, že si pomocí for cyklů a if podmínek generujeme kód.

[4] DDL je akronym pro data definition language, což je skupina SQL příkazů umožňující tvorbu a úpravu strukturu databázových objektů, jako je například vytvoření tabulky či její smazání (Awati 2022).

na to, o jakou z podporovaných databází se jedná (getdbt 2024a). Stejně tak vám udrží vaši tabulku vždy aktuální a vývojář nemusí sloupce spravovat ručně.

1.2.7 Model governance v dbt

V novějších verzích dbt přišlo dbt Labs s ***řešením i pro model governance v dbt***. Jedná se o možnost zavedení následujících 4 faktorů do vašich projektů:

- přístupy k modelům,
- datové kontrakty,
- verze modelů,
- závislosti modelů napříč projekty.

Výše uvedené možnosti vám dávají prostor zajistit v projektech vyšší bezpečnost, datovou kvalitu a plynulejší publikaci změn v modelech s možností definice, o jaké změny se jedná (getdbt 2024c).

Přístupy k modelům

Přístupy k modelům slouží k určité ochraně dat a redukci chybovosti dat vzhledem k nekompletním tabulkám. Z tohoto důvodu přichází s řešením, které se nazývá „***Model access***“. Jedná se o ***způsob tvorby skupin uživatelů a následnému označování modelů (tabulek) jako public, restricted a private***. Díky tomuto rozdělení dbt dokáže ochránit produkční tabulky od tabulek ve vývoji či přístupům k modelům napříč různými odděleními. Příkladem může být situace, kdy marketingové oddělení nebude mít možnost propojit modely (tabulky) s oddělením financí (getdbt 2024n).

Datové kontrakty

Datovým kontraktem se rozumí ***dohodou mezi poskytovatelem dat a jejich spotřebitelem***. Umožňuje ***nastavit formu a kvalitu dat***, která se od poskytovatele očekává, aby splňovala byznysové potřeby (Jones 2023). Příkladem může být definování:

- názvu sloupců,
- počet sloupců,
- datové typy jednotlivých sloupců,
- datovou kvalitu (entitní integrita, referenční integrita, …).

Datové kontrakty lze využít jako ***jeden z dalších přístupů k zajištění datové kvality***. S jejich nastavením by se při práci s daty nemělo stát, že v nahraných datech od poskytovatele chybí určitý počet sloupců či naopak nějaký přebývá. Zároveň je v kontraktu pevně definován datový typ, tudíž by neměla nastat situace, kdy ve sloupci analytik očekává hodnotu BOOLEAN (0 nebo 1, FALSE nebo TRUE) obdrží náhodný textový řetězec. Například při práci s dbt je analytik ***v případě porušení datového kontraktu automaticky upozorněn prostřednictvím selhání produkčních transformací***, což signalizuje problém s daty, který je třeba řešit.

Verze modelů

Verzování kódu a verzování modelů se v dbt rozumí 2 odlišným věcem. ***Verzování modelů nese vazbu na datové kontrakty***. V praxi se běžně stává, že je potřeba přejmenovat v tabulce sloupec, přidat nový, nebo naopak nějaký odebrat. Zde přichází dbt s funkcionalitou, která umožňuje monitorovat změny v čase tvorbou nových verzí modelů (tabulek) a zapisováním jejich rozdílů v YAML souborech. ***Vývojář tak může držet několik verzí modelů*** najednou a definovat, který z nich je „živý“ a který bude od kterého datumu zastaralý (getdbt 2024o).

Například můžeme definovat první verzi tabulky „customer“ obsahující 3 sloupce, časem se však ukáže, že je potřeba tabulku „customer“ rozšířit o 2 další sloupce, ale tuto změnu budeme chtít provést až za 3 týdny. Vývojář však může druhý (aktualizovaný) model připravit a definovat ho jako „prerelease“ verzi. Prvnímu modelu nastavíme informaci o jeho plánovaném přepnutí a v definovaný den jednoduše označí druhý model za „živý“.

Závislosti modelů napříč projekty

Kromě přístupů k modelům lze ***ke zvýšení ochrany přistupovat vytvořením několika dbt projektů***, které si do svého projektu můžete nahrát jako balíčky (viz. další kapitoly). dbt vám tak umožní odkazovat na modely z různých projektů.

Pokud se vrátíme k příkladu s různými odděleními, situace by vypadala tak, že marketingové oddělení bude mít svůj vlastní projekt, stejně tak oddělení pro zákaznickou spokojenost. V případě, že bude chtít marketingové oddělení pracovat s daty o zákaznické spokojenosti, naimportuje si jejich projekt do vlastního projektu a může využít jejich připravené tabulky (modely) bez možnosti jejich úprav – tzn. read-only přístup.

1.2.8 Import knihoven od jiných vývojářů ve formě dbt packages

Jako poslední z hlavních výhod dbt bych zmínil možnost využití dbt packages. ***Dbt packages si lze představit jako knihovny v programování.*** Jedná se o ***soubor funkcionalit***, které v základních verzích dbt nelze najít a tyto funkcionality si lze legálně nahrát do vlastních projektů.

2. Technické charakteristiky dbt

Kapitola představí technický pohled na dbt projekt, jednotlivé ***formáty a struktury souborů***, které jsou pro dbt klíčové. Kromě struktury pokrývá také příkazy dbt a ***principy orchestrací***, což umožňuje efektivní automatizaci datových transformací.

2.1 Formáty souborů v rámci dbt

Před tím, než budou zmíněné konkrétní struktury dbt projektu, je důležité zmínit ***tři základní formáty souborů***, které hrají v rámci dbt zásadní roli. Bez nich by nebylo možné s tímto nástrojem efektivně pracovat:

- SQL soubory,
- markdown soubory,
- YAML soubory.

SQL soubory

Do SQL souborů se vkládají, jak už je jasné z přípony, SQL dotazy. Tyto ***soubory jsou ve struktuře 4 složek:***

- macros,
- models,
- analyses,
- tests.

SQL dotazy nemusí být nutně pouze SELECT, ale také ***část kódu, která má být přepoužitelná*** – stejně jako u metod v rámci programovacích jazyků.

MD soubory

MD soubory slouží k vedení dokumentace primárně pro 2 účely:

- overview,
- opakující se dokumentace.

Overview je specifické klíčové slovo, které se využívá ***při tvorbě hlavní dokumentační stránky*** – tzn. vše, co bude součástí dokumentace pod „blokem“ overview se zobrazí na hlavní stránce. Co se týče ***opakující se dokumentace***, zde se jedná o popisek tabulky, sloupce, či makra, který je využitelný napříč projektem vícekrát. Z tohoto důvodu ji ukládáme do .md souboru, aby její úprava byla na jednom místě, nicméně byla použitelná napříč celým projektem.

YAML soubory

Soubory ***s příponou yml*** jsou v dbt využity jako ***konfigurační soubory***. dbt z nich je poté schopno ***čerpat několik klíčových věcí***:

- nastavení projektu,
- dokumentace,
- testy.

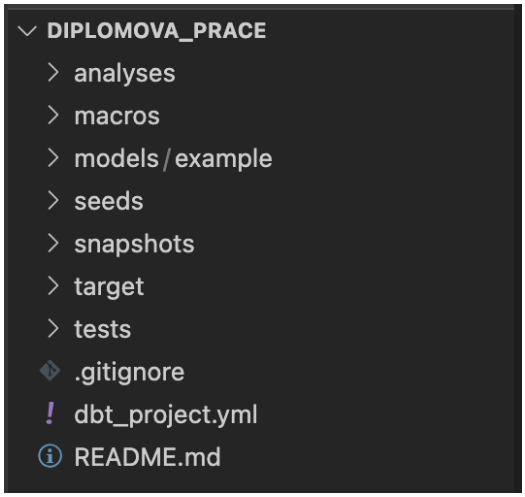
Díky těmto souborům je vývojář schopen ***dokumentovat a testovat přímo v projektu*** – na jednom místě. Zároveň umožňují ***nastavení pravidel pro celý projekt*** (ať už co se týče zápisu dat do různých databází, schémat, či názvů, do kterých mají být uložené), konkrétní model, či skupinu modelů.

2.2 Struktura dbt projektu

Při založení nového projektu se uživateli ***vygeneruje základní dbt projekt***, který obsahuje několik složek a souborů sloužících pro různé účely. Základní strukturu je možné ***obohacovat o další složky či soubory,*** nicméně je potřeba mít na paměti, že bez některých souborů dbt projekt nebude fungovat. Na dalším obrázku je možné tuto strukturu vidět. Vzhledem k tomu, že je dbt napojeno na git, na obrázku níže se vyskytují 2 soubory:

- .gitignore (soubor sloužící k nastavení souborů vynechání při publikaci kódu),

- README.md (základní dokumentační soubor při zobrazení Git repozitáře).



Obrázek 2-1: Základní struktura dbt projektu po vygenerování

Strukturu lze měnit například ***přejmenováním složek, či přidáním složek vlastních.*** V případě těchto změn je nutné aktualizovat čtení ze složek, aby dbt rozpoznalo, co se ve které složce nachází. Toto nastavení bude zmíněno v kapitole dbt_project.yml.

2.2.1 Profiles.yml

V případě lokálního vývoje a práci s dbt Core je důležité zmínit ještě jeden soubor – profiles.yml (getdbt 2024d). ***Profiles.yml*** defaultně není součástí struktury dbt projektu, ale ***vzniká jako soubor ve skryté složce*** na uživatelském počítači při zakládání prvního projektu. Skrytý je primárně z toho důvodu, že obsahuje (v některých případech citlivé) informace související s připojením k různým databázím – v případě databáze Snowflake:

- název accounta,
- název databáze (do které chce uživatel zapisovat),
- přihlašovací jméno uživatele,
- přihlašovací heslo uživatele,
- role (se kterou se kód spouští),
- název schématu (do kterého chce uživatel zapisovat),
- typ databáze,
- typ warehouse (který se má aktivovat při zpracovávání dotazů),
- maximální počet vláken (kolik dotazů najednou může uživatel spustit).

V momentě, kdy dochází na „***překlopení“ lokálního vývoje na produkční prostředí v cloudu***, kde je nutno dbt orchestrovat, se profiles.yml přidává i do struktury dbt projektu. Na tento profiles.yml je pak při spouštění dbt běhu ve formě cesty k souboru přidán odkaz. V praxi je běžné veškeré citlivé údaje nahrazovat skrytými proměnnými jako jsou například ENV_VAR při práci s Gitem či secretEnv při práci se Secret Managerem na GCP, aby nedošlo k úniku dat.

Při zakládání dalších projektů se tento soubor pouze aktualizuje (veškeré projekty jsou v rámci základního nastavení na jednom místě a rozšiřují základní profiles.yml). Pokud se ale uživatel rozhodne jinak, může si těchto souborů založit vícero. V takovém případě je ale při vývoji vždy nutné zadefinovat cestu ke správnému profiles.yml. Ukázku struktury je možné vidět na dalším obrázku.

```yaml
diplomova_prace:
  outputs:
    dev:
      account: x
      database: x
      password: x
      role: x
      schema: x
      threads: 4
      type: snowflake
      user: x
      warehouse: x
  target: dev
```

Obrázek 2-2 Ukázka struktury profiles.yml pro připojení k databázi Snowflake

Profiles.yml umožňuje zjednodušení práce s různými formami prostředí (testovacích či produkčních), ale tato část je rozebrána v samostatné kapitolách týkajících se dynamických výběrů a zápisů.

2.2.2 Analyses

První zmíněnou složkou, která se v dbt struktuře nachází, je analyses. Princip této složky slouží k tomu, aby si vývojář mohl ***vyzkoušet různé analytické SQL dotazy***, které ale úplně nutně nemusí být materializovány do databáze (jako soubory ve složce models). Stejně jako u již zmíněných modelů je principem vytváření textových .sql souborů, ze kterých lze pomocí funkcí {{ ref() }} či {{ source() }} odkazovat na různé modely. Princip je ale takový, že tato analytická query se nikdy nepropíše do databáze jako fyzická tabulka, nýbrž se z ní pouze vytvoří zkompilovaný kód (getdbt 2024h).

Práce s těmito soubory je mnohem pohodlnější v dbt Cloud, kde má vývojář možnost rovnou zobrazit výsledky dotazu, ale nechce si nutně zahlcovat složku s modely. V dbt Core je tato funkcionalita omezenější, vzhledem k tomu, že bez speciálního IDE (či běžným IDE s nainstalovanými doplňky) výsledky dotazu nelze spustit.

2.2.3 Macros

Další důležitou součástí dbt projektu jsou makra (macros). Makra slouží k ***zjednodušení a znovupoužití kódu v SQL dotazech.*** V podstatě fungují jako funkce v programovacích jazycích – vývojář si může definovat opakovaně využitelný kód, který je následně volán na různých místech v projektu. Makra se píší v jazyce Jinja, který je integrován s dbt a umožňuje dynamicky generovat SQL kód (getdbt 2024l).

Typickým příkladem je situace, kdy chceme provést stejnou transformaci dat ve více modelech – místo toho, abychom stejný kód psali opakovaně, vytvoříme makro, které bude přijímat parametry a tuto transformaci provede. Makra jsou velmi flexibilní a je možné je využít pro různé účely, včetně automatizace generování SQL dotazů, vytváření dynamických filtrů nebo podmínek, či složitějších výpočtů. Při správném návrhu mohou výrazně zjednodušit správu, údržbu projektu a velmi redukovat chybovost při úpravě kódu.

2.2.4 Models

Modely jsou ***jednou z klíčových složek dbt***. Vývojář píše SQL kód do souborů uložených ve složce models, které dbt následně zkompiluje a spustí v cílové databázi. Jsou základními stavebními bloky transformací v dbt, kde se surová data z různých zdrojů transformují do formátu vhodného pro další analýzu nebo reportování.

Jedná se o .***sql soubory, které vytvářejí fyzické tabulky nebo views v databázi***. Tabulky se fyzicky uloží do databáze a obsahují statická data, zatímco pohledy jsou dynamické dotazy, které se generují při každém jejich spuštění. Třetí možnost, jak můžeme ***modely materializovat*** je *ephemeral*, tato

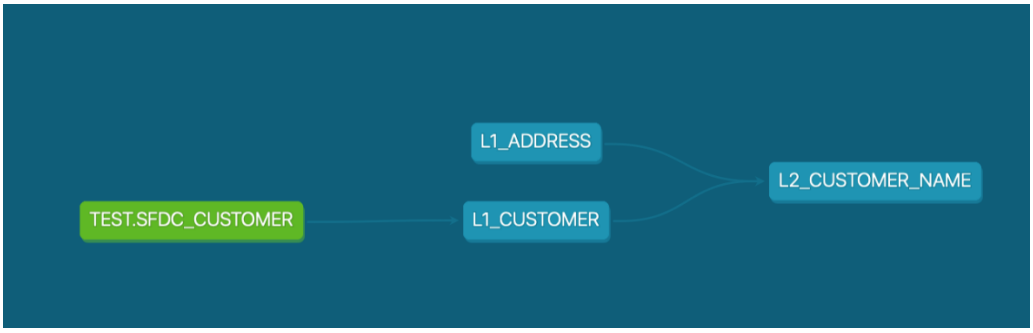
forma materializace může existovat jako samostatný model, nicméně není materializován v databázi jako tabulka či view. Ephemeral model je využit jako CTE, které se při odkazu na něj automaticky vloží do SQL dotazu jiných modelů. O tom, zda bude model materializován jako tabulka nebo pohled, rozhoduje nastavení materializace v souboru dbt_project.yml nebo přímo v jednotlivém modelu (getdbt 2024b).

Díky funkcím jako {{ ref() }} a {{ source() }} je možné jednoduše odkazovat na další modely a zdroje, což zajišťuje, že dbt dokáže automaticky spravovat závislosti mezi modely a zajistit, že jsou spouštěny ve správném pořadí. Jak může takový model vypadat lze vidět na ukázce níže.

Jak již bylo výše uvedeno, mezi modely na sebe lze odkazovat dvěma způsoby, které se od sebe logicky silně liší:

- source funkcí = {{source(‚nazev_zdroje‘,‘tabulka_na_zdroji‘)}},
- ref funkcí = {{ref(‚nazev_modelu‘)}}.

Source funkce by měla být zpravidla využívána pouze ***pro první vrstvu datových skladů***, která odkazuje přímo na zdrojová data v databázi. Na ref funkce se přechází v momentě, kdy se propojují jednotlivé modely mezi sebou (již se neodkazuje přímo na zdrojová data v databázi). Na obrázku níže lze vidět rozdíly – zdroje (sources) jsou uvedené zeleně, kdežto reference mezi modely jsou uvedené modře.



Obrázek 2-3 Ukázka source() a ref() funkcí v DAGu

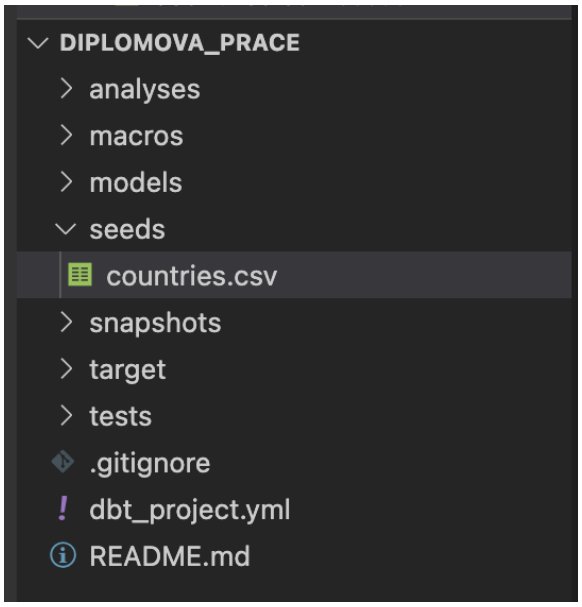
Výpis 1 Ukázka modelu L2_customer_name.sql

```
SELECT
          Customer.ID
          ,customer.NAME
          ,customer.CUSTOMER_TYPE
          ,address.ADDRESS
FROM {{ ref(“L1_CUSTOMER“) }} customer
JOIN {{ ref(„L1_ADDRESS“) }} address ON customer.ID = address.PERSON_ID
```

2.2.5 Seeds

Seed soubory (seeds) jsou ***statické CSV soubory***, které lze načíst do databáze jako tabulky pomocí dbt. Tato funkcionalita umožňuje vývojářům importovat menší množství dat, která mohou sloužit jako ***referenční nebo pomocná data pro transformace*** (getdbt 2024f). Příkladem může být číselník států se standardem ISO 3166-1. Takový seed by byl prakticky uložen ve složce seeds s názvem „countries.csv“.

Praktická ukázka



Obrázek 2-4 Struktura dbt projektu s přidaným souborem countries.csv

Výpis 2 Ukázka seedu countries.csv souboru

```
alpha_2, Country
AF, Afghanistan
AX, Alandy
```

Z praktických zkušeností lze ale říci, že tuto možnost ale používáme pouze v minimální míře (spíše vůbec). To primárně z toho důvodu, že je v případě změn nutné udělat release (v rámci Git repozitáře), což neumožňuje business uživatelům rychle reagovat na změny v datech či v číselnících.

2.2.6 Snapshots

Snapshoty (snapshots) v dbt slouží k ***uchovávání historických verzí dat*** v průběhu času (jedná se o SCD2 formu) (getdbt 2024g). SCD2 forma ukládání dat je nejčastější typ SCD. Umožňují zachytit změny v tabulkách, které se mohou dynamicky měnit, a ukládat jejich stavy do databáze, což je klíčové pro sledování historických datových změn, jako jsou například změny ve stavu adres, zákazníka nebo objednávek (Kimball a Margy 2011, str. 311).

Vývojář definuje snapshot pomocí SQL dotazu a určí sloupce, na jejichž základě dbt automaticky sleduje změny (např. ID zákazníka). Pokud se záznamy změní, dbt snapshot aktualizuje a uloží novou verzi dat.

Dbt umožňuje sledovat změny 2 možnostmi:

- časová značka (timestamp),
- hodnoty ve sloupcích.

Obě možnosti mají svůj specifický účel – na základě praktických zkušeností je lze rozlišit poměrně jednoduše. ***Možnost časových značek je vhodnější při využití inkrementálním vstupu dat*** – tzn. zdrojová data obsahují pouze řádky, na kterých proběhla změna. Co se týče kontroly dat dle hodnot ve sloupcích, tu je vhodnější využít pro práci s kompletním vstupem dat (tzn. na zdroji jsou vždy veškerá data). Je potřeba ale zmínit, že kontrolu dat dle hodnot ve sloupcích je občas obtížnější nastavit, obzvlášť pokud je tabulka na vstupu opravdu široká.

Snapshoty jsou užitečné pro analýzu historických trendů, audity dat a jakékoli scénáře, kde je potřeba uchovávat a analyzovat změny v čase. Tento přístup výrazně zjednodušuje správu verzí dat bez nutnosti složitých vlastních mechanismů v databázi.

Výpis 3 Ukázka nastavení snapshotu s kontrolou časových značek (docs.getdbt.com)

```
{% snapshot orders_snapshot_timestamp %}

    {{
        config(
          target_schema='snapshots',
          strategy='timestamp',
          unique_key='id',
          updated_at='updated_at',
        )
    }}

    select * from {{ source('jaffle_shop', 'orders') }}

{% endsnapshot %}
```

2.2.7 Target

Adresář target slouží jako ***metadatové úložiště*** – z těchto souborů se generuje například dokumentace, nebo se zde ukládají informace o posledním proběhlém běhu. Na co by se měl vývojář zaměřit při vývoji jsou primárně 2 složky:

- compiled,
- run.

V adresáři compiled lze nalézt zkompilovaný kód z vytvořených modelů, lze z nich poměrně dobře debugovat vývoj modelů. Veškeré reference a zdroje ({{ref()}} a {{source()}}) jsou nahrazeny za cesty v databázi podle nastavení v YAML souborech a makra jsou nahrazena za vygenerovaný kód. Jedná se tedy o finální SELECT statement, který bude provolán v databázi.

Výpis 4 Model L1_customer_name.sql v dbt

```
SELECT
    *
FROM {{ ref('L1_CUSTOMER')}}
JOIN {{ ref('L1_ADDRESS')}}
```

Výpis 5 Zkompilovaný model pro databázové spuštění

```
SELECT
    *
FROM DBT_TEST.DBT.L1_CUSTOMER
JOIN DBT_TEST.DBT.L1_ADDRESS
```

Adresář run obsahuje kompletní dotaz včetně DDL, který je z dbt spouštěn v databázi. Soubor L1_customer_name by v našem případě obsahoval i například CREATE OR REPLACE TABLE. Jedná se o finální query, která je v databázi provolána.

Dalšími důležitými soubory jsou již zmíněné metadatové:

- catalog.json,
- manifest.json,

- run_results.json.

Catalog.json

Tento soubor obsahuje ***informace o struktuře dat ve projektu*** a využívá se pro vygenerování dbt dokumentace (getdbt 2024i).

Manifest.json

Manifest obsahuje ***kompletní informace informace o projektu***, všech modelech, závislostech mezi modely, testech a zdrojích na jednom místě v JSON formátu. Stejně jako catalog.json se využívá při generování dokumentace (getdbt 2024m).

Run_results.json

V posledním ze 3 důležitých souborů lze nalézt ***data o posledním spuštěném běhu a jeho výsledcích***. Je možné zjistit, které modely se spustily, které ani spuštěny nebyly či které testy proběhly v pořádku nebo naopak nalezly v datech chybu (getdbt 2024q).

2.2.8 Tests

Datovou kvalitu lze zajistit pomocí dbt testů, jejichž základní rozdělení je následovné:

- generic (obecné testy vhodné na testování integrity),
- singular (testy zaměřující se na byznysovou logiku).

Aplikací těchto testů můžete zajistit, že vaše data obsahují pouze správná data. Základní chování je takové, že pokud dbt ***objeví na základě definovaných testů datovou chybu, celý běh zastaví*** a veškeré tabulky, které na chybné tabulce závisí, nebudou aktualizovány. Toto základní chování je možné pozměnit. U veškerých testů lze nastavit tzv. „severity“, která definuje závažnost testu. Na základě nastavené závažnosti pak dbt pokračuje v běhu s tím, že zašle varování o datové chybě, nebo běh pro následující modely zastaví (getdbt 2024e).

Závažnost lze nastavit i na konkrétní počet řádků. Situace by vypadala tak, že by vývojář nastavil pro případ, kdy test odhalí méně než 30 špatných záznamů, pouze varování. Pokud by počet špatných záznamů překročil 30, změní se varování na chybu a běh se zastaví (getdbt 2024j).

Generic

Jak již bylo řečeno, ***generic testy jsou obecné testy primárně založené na kontrolu spíše z datového než byznysového pohledu.*** Dbt má od základu zabudované 4 typy generic testu:

- unique (kontrola unikátnosti),
- not_null (kontrola vyplněnosti),
- accepted_values (kontrola doménové integrity),
- relationships (kontrola referenční integrity).

Vzhledem k tomu, že generic testy jsou testy aplikované ve většině případů nad konkrétními sloupci „bez byznysové logiky“, dbt umožňuje vývojářům psát vlastní generic testy. Takový test může vypadat například jako kontrola, zda je e-mail či telefonní číslo zadáno ve správném formátu. V případě generic testů se jedná o implementaci do YML konfiguračních souborů, jak lze vidět na výpisu č.6 nad modelem „customer.sql“ níže (getdbt 2024e).

Výpis 6 Customer.sql model

```
SELECT
          ID
          ,NAME
          ,CUSTOMER_TYPE
          ,ORDER_ID
FROM {{ ref(“L1_SALESFORCE_CUSTOMER“) }}
```

Výpis 7 Customers.yml s dokumentací a testy (vlastní zpracování)

```
version: 2

models:
  - name: customer
    description: „Customers from Salesforce.“
    columns:
      - name: ID
        description: „Primary key of the table.“
      - name: CUSTOMER_TYPE
        description: „Customer group based on orders.“
        tests:
          -  accepted_values:
               values: [“LOW“,“MEDIUM“,“HIGH“]
```

Singular

Singular testy jsou testy které jsou ukládané jako SQL soubory právě do adresáře tests. Na rozdíl od základních dbt generic testů může analytik aplikovat libovolnou business logiku napříč jednou či více tabulkami – takovým příkladem může být to, že suma ceny produktů u objednávek není záporná.

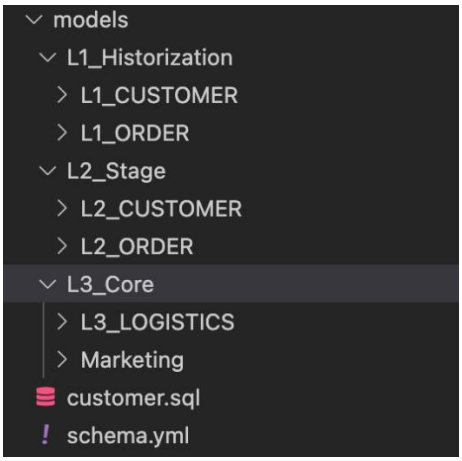
Výpis 8 Ukázka singular testu na byznysovou logiku

```
SELECT
         ORDER_ID
         ,SUM(PRODUCT_PRICE) as ORDER_PRICE
FROM {{ ref(“L2_ORDER“) }}
GROUP BY 1
HAVING ORDER_PRICE < 0
```

2.2.9 dbt_project.yml

Dbt_project.yml slouží jako ***primární konfigurační soubor pro celý projekt.*** Dbt má určité základní nastavení, nicméně zde je správné centrální místo, kde vývojář může nastavení modifikovat. Vývojář zde může nastavit nejen, ve kterých složkách se nachází, které struktury (např. models, docs, …). Zároveň také do kterých databází či schémat se mají různé modely zapisovat.

Dbt_project.yml je v hierarchii priorit na nejvyšší úrovni. V případě, že je potřeba dělat určité výjimky v pravidlech, lze tato pravidla zadefinovat v YAML souborech pro konkrétní modely či zdroje. V takovém případě získá pravidlo pro konkrétní model vyšší prioritu a pro tento model je konfigurace přepsána, přestože je v kolizi s centrálním nastavením.



Obrázek 2-5 Ukázka struktury modelů pro výpis níže

Výpis 9 Ukázka nastavení dbt_project.yml

```
name: 'diplomova_prace'
version: '1.0.0'
config-version: 2

profile: 'diplomova_prace'

model-paths: ["models"]         # adresar s ulozenymi modely
analysis-paths: ["analyses"]    # adresar s analyses modely
test-paths: ["tests"]           # adresar se singular / custom generic testy
seed-paths: ["seeds"]           # adresar s ulozenymi seedy
docs-paths: ["docs"]            # adresar s dokumentaci
macro-paths: ["macros"]         # adresar s makry
snapshot-paths: ["snapshots"]   # adresar se snapshoty

clean-targets: # adresare, ktere se promazou po pouziti prikazu 'dbt clean'
  - "target"
  - "dbt_packages"

models:
    L1_Historization:
      L1_CUSTOMER:
        +schema: L1_DOMAIN_CUSTOMER # modely ve slozce L1_CUSTOMER ukladat do schematu L1_DO-
MAIN_CUSTOMER
    L2_STAGE:
      L2_CUSTOMER:
        +schema: L2_CUSTOMER        # modely ve slozce L2_CUSTOMER ukladat do schematu
L2_CUSTOMER
    L3_Core:
      L3_LOGISTICS:
```

```
        +schema: L3_DORUCENI_DO_DRUHEHO_DNE # modely ve slozce L3_LOGISTICS ukladat do sche-
matu L3_DORUCENI_DO_DRUHEHO_DNE
```

2.2.10 Docs

Složka docs není součástí základní struktury dbt projektu, nicméně je v praxi relativně běžně implementována. ***V ní se nejčastěji nachází 2 typy souborů***:

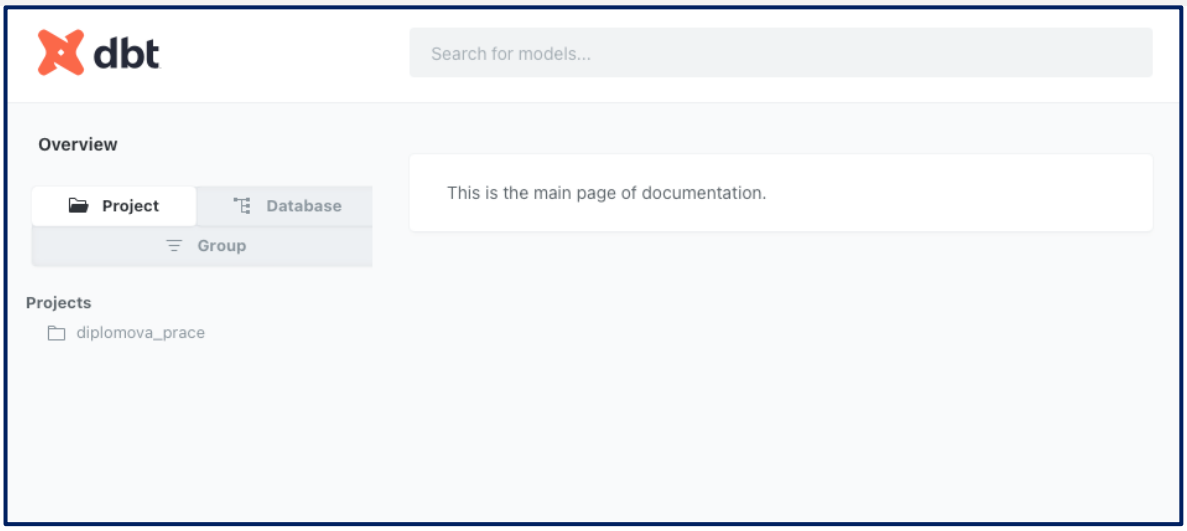
- obrázky (ty bývají uložené ve složce assets),
- dokumentace ve formátu markdown.

Dbt umožňuje uživatelům ***vytvářet markdownové soubory s dokumentací***, které jsou velmi dobře využitelné v případě, že vývojář využívá popisek na vícero místech (ať už se jedná o dokumentaci makra, modelu, sloupce či zdroje). Díky těmto souborům vývojář odkáže na jednu dokumentaci, kterou tak může spravovat na jednom místě s dopadem napříč projektem (getdbt 2024k).

Markdownové soubory lze využít k ***několika formám dokumentace – makra, tabulky, sloupce či hlavní stránka dokumentace.*** Pro hlavní stránku se v markdownovém souboru používá docs block s názvem __overview__. Na základě zkušeností si autor dovolí tvrdit, že je lepší držet zvlášť soubory pro hlavní stránku a zbylé prvky (tabulky, makra, …).

Výpis 10 Ukázka main_page.md sloužící pro hlavní stránku dbt dokumentace

```
{% docs __overview__ %}
    This is the main page of documentation.
{% enddocs %}
```



Obrázek 2-6 Ukázka vygenerované dokumentace pro hlavní stránku (vlastní zpracování)

Aby byla dokumentace plně funkční, je ***potřeba přidat nastavení, kde se složky s dokumentací nachází do projektového YAML souboru.*** Pokud budete chtít dokumentovat makra, tabulky či sloupce, v souborech je potřeba vytvořit tzv. docs block v Jinja syntaxi, na který v YAML souborech vývojář odkáže. Na výpisech níže lze vidět vytvoření docs block v markdownovém souboru a odkaz na něj v YAML souboru. Primární výhodou je, jak již bylo zmíněno, centrální místo pro údržbu dokumentace.

Výpis 11 Ukázka markdownového souboru documentation.md s docs bloky pro dokumentaci (vlastní zpracování)

```
{% docs CUSTOMER_TABLE %}
    Customers from all source systems.
{% enddocs %}
```

```
{% docs PK_DESC %}
    The primary key for this table.
{% enddocs %}
```

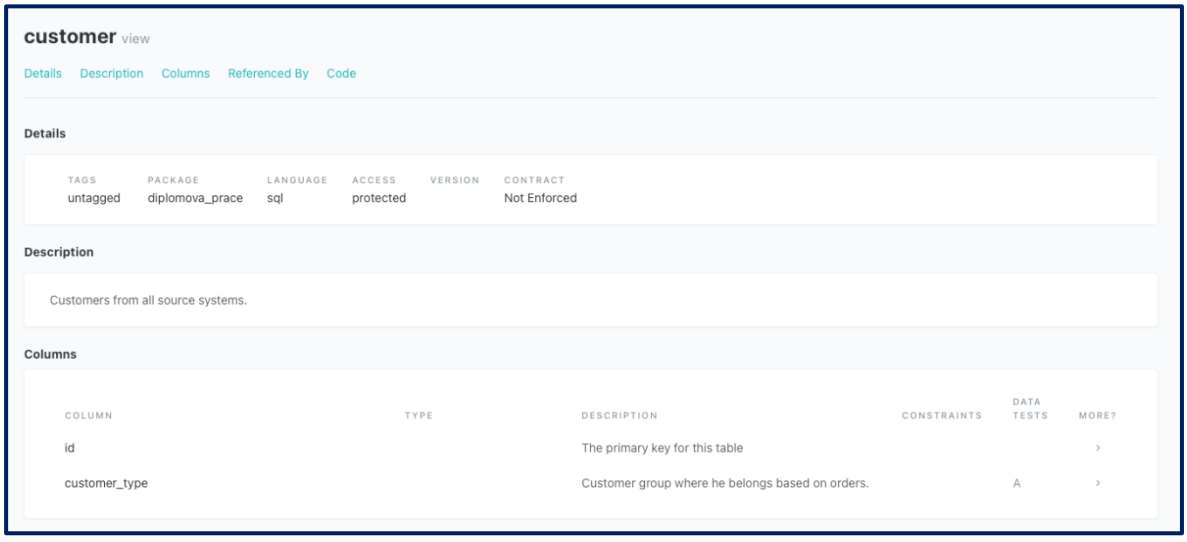
Výpis 12 Ukázka dokumentace s odkazem do markdown souboru v YAML souboru

```
version: 2

models:
  - name: customer
    description: "{{docs('CUSTOMER_TABLE')}}"
    columns:
      - name: ID
        description: "{{docs('PK_DESC')}}"
```

Používání markdownových souborů není jedinou možností, jak lze v dbt dokumentovat. Druhým způsobem dokumentace je i přímé dokumentování v YAML souborech – hned vedle testů.

Výpis 13 Ukázka statické dokumentace v YAML souboru

```
version: 2

models:
  - name: customer
    description: "Customers from all source systems."
    columns:
      - name: ID
        description: "The primary key for this table"
```



Obrázek 2-7 Vygenerovaná dokumentace pro model customer.sql (vlastní zpracování)

V neposlední řadě dbt Labs mysleli i na situaci, kdy by vývojář chtěl dokumentaci držet i přímo v databázi – nejen v projektových souborech a vygenerované dokumentaci. Pomocí parametru nastavení persist_docs v dbt_project.yml se ***veškerá dokumentace z dbt propíše do propojené databáze*** na všech úrovních – tabulek i sloupců (getdbt 2024p).

2.2.11 Packages.yml

Soubor packages.yml v dbt slouží ***k definování externích balíčků***, které chce vývojář integrovat do svého projektu. Tyto balíčky mohou obsahovat předdefinované modely, makra, testy nebo další dbt funkcionality, které rozšiřují možnosti práce s daty a usnadňují opakovaně využitelné transformace nebo datové procesy. Balíčky jsou spravovány prostřednictvím dbt Hubu nebo interních úložišť Git a jsou verzované, což umožňuje vývojářům snadno udržovat jejich projekty aktuální.

2.3 dbt příkazy

Dbt lze ovládat pomocí několika příkazů - viz tabulka, kde je přehledně vidět vazba: příkaz – význam (co příkaz dělá). Je důležité zmínit, že ***příkazy mohou být spouštěné s různými argumenty***, ty budou zmíněné mimo tabulku. V tabulce nejsou uvedené všechny příkazy, ale pouze ty z praxe nejčastěji používané. Veškerý seznam příkazů je k dispozici v oficiální dokumentaci.

Tab. 2 Seznam často používaných příkazů dbt (getdbt – dbt commands)

Příkaz	Vysvětlení
dbt init	Vytvoření dbt projektu.
dbt run	Spuštění všech modelů – reálné propsání do databáze.
dbt seed	Spuštění všech seedů – reálné propsání do databáze.
dbt snapshot	Spuštění všech snapshotů – reálné propsání do databáze.
dbt build	Spuštění modelů, seedů, snapshotů a aplikování nastavených testů.
dbt docs generate	Vygenerování souborů pro dokumentaci.
dbt docs serve	Lokální spuštění dokumentace.
dbt docs serve	Spuštění dokumentace (zobrazení webové stránky).
dbt retry	Spuštění uzlů, které v posledním běhu neproběhly úspěšně.
dbt run-operation	Spuštění makra.
dbt deps	Nainstalování balíčků.

Jak již bylo uvedeno výše, k některým příkazům ***je možné použít argumenty***. Velmi častým argumentem je příkaz *--select* či pouze *-s*. Tento argument slouží jako možnost spustit pouze konkrétní subset modelů, seedů, či snapshotů (getdbt 2024r). Jeho využití by bylo například v situaci, kdy jste udělali změnu v modelu L2_customer a chcete jeho změnu (společně s testy) propsat do závislých modelů (downstreamu). Příkaz by vypadal následovně:

Výpis 14 Příkaz pro spuštění downstreamu L2_customer

```
dbt build -s L2_customer+
```

Ve výše uvedeném příkladu + za modelem L2_customer spustí celý downstream. Naopak + před modelem by spustilo celý upstream.

2.4 Orchestrace

Orchestrace v datovém světě představuje ***proces, který automatizuje provádění datových transformací definovaných vývojáři.*** V dbt se stává klíčovou součástí při nasazení dbt do produkčního prostředí, kde je třeba zajistit, ***aby transformační pipeline běžela konzistentně***, bez přerušení a v pravidelných intervalech. Velkým přínosem orchestrací v dbt je ***schopnost dynamicky řídit závislosti mezi jednotlivými modely***, přičemž dbt na základě ref a source funkcí rozpozná, které modely je třeba spustit a v jakém pořadí, aby byly transformace provedeny úspěšně. To zjednodušuje celý proces a minimalizuje chybovost.

2.4.1 Orchestarční nástroje

GCP

GCP nabízí ***mnoho způsobů, jak pipeliny orchestrovat***, nelze vybrat jednu konkrétní, ale záleží na každém případu zvlášť. Orchestraci v rámci GCP autor vybral z důvodu, že s ní má nejvíce zkušeností v praxi. Z důvodu ulehčení představy o tom, jak tato praktická GCP orchestrace funguje, autor vytvořil obrázek. Než bude popsán konkrétní návod, jak orchestraci provést, budou v krátkosti zmíněny výhody a nevýhody využití této platformy pro orchestraci společně s teoretickým popisem jejího fungování.

Mezi výhody využití této platformy je rozhodně již obstaraná infrastruktura – běžně pak na úkor jednoduchosti správy přichází kámen úrazu, kterým i v případě GCP je cena. Pro malé projekty existuje šance, že vývojář nepřekročí limity služeb, v rámci kterých je GCP zdarma.

dbt Cloud

Kromě Google Cloud Platform existují další možnosti, jak orchestraci dbt řešit. Jednou z nich ***je dbt Cloud*** - dbt Cloud nabízí ***jednoduché nastavení, podporu pro týmovou spolupráci a snadnou integraci s Git repozitáři,*** což umožňuje snadné řízení datových transformací bez nutnosti spravovat vlastní infrastrukturu. Nevýhodou je, že je zdarma pouze pro jednoho vývojáře, následně je cena rovná 100 USD/vývojáře.

Keboola

Další možností je Keboola, ***datová platforma, která podporuje orchestraci dbt v širším kontextu datových procesů.*** Keboola umožňuje propojit dbt s dalšími částmi datové pipeline, jako je nahrání dat, transformace a export dat. Tato flexibilita dělá z Kebooly vhodný nástroj pro ty, kteří chtějí dbt integrovat do komplexních datových ekosystémů a zároveň těžit z automatizace a správy dat v rámci jedné platformy. Na úkor jejího používání je zde mínus, stejně jako u dbt Cloud, cena.

Apache Airflow

Třetí možností je Apache Airflow, ***populární open-source platforma pro orchestraci workflow***. Airflow poskytuje robustní nástroje pro plánování a řízení datových pipeline, včetně podpory pro dbt. Airflow je vhodný pro složitější datové operace a umožňuje detailní řízení závislostí mezi úlohami, což z něj dělá ideální nástroj pro pokročilé uživatele, kteří potřebují komplexní orchestraci datových procesů. Oproti možnostem dbt Cloud a Kebooly je Airflow sice zdarma, nicméně jeho obstarávání a nastavení vyžaduje pokročilejší znalosti programovacích jazyků a infrastruktur.

3. Aplikace dbt při stavbě DWH pro batchové zpracování dat

3.1 Založení dbt projektu

Založení dbt projektu je ***klíčovým krokem*** při práci s nástrojem dbt (Data Build Tool).

3.1.1 Instalace dbt core a adaptérů

Prvním krokem při založení dbt projektu je ***ověřit, že máme správně nainstalovaný nástroj dbt-core*** a příslušný adaptér pro databázi, kterou budeme v projektu používat. Pro kontrolu verze dbt a dostupných adaptérů otevřeme terminál a zadáme příkaz:

Výpis 15 Příkaz pro kontrolu nainstalované verze dbt

```
dbt --version
```

Pokud dbt-core či adaptér nejsou nainstalovány, ***může je analytik doinstalovat***. V této situaci existují 2 možnosti:

- instalace poslední verze,
- instalace konkrétní verze.

Pro instalaci poslední verze je nutné použít příkaz:

Výpis 16 Příkaz pro instalaci dbt Core

```
pip install dbt-core
```

Pokud chce vývojář nainstalovat určitou verzi dbt, nikoliv poslední, ***musí zvolit její specifikaci*** v příkazu – to následovně (příklad pro verzi 1.7.5):

Výpis 17 Příkaz pro nainstalování specifické verze dbt

```
pip install dbt-core==1.7.5
```

Po instalaci dbt je nutné ***vybrat adaptér (databázi),*** nad kterou projekt poběží. V daném případě se jedná o BigQuery[5]. Pro instalaci BigQuery je nutné zadat následující příkaz.

Výpis 18 Příkaz pro nainstalování adaptéru (konektoru pro databázi)

```
pip install dbt-bigquery
```

Při instalaci je potřeba mít na paměti, že v případě, kdy je verze adaptéru vyšší, než je verze samotného dbt-core, dbt nebude fungovat.

3.1.2 Vytvoření dbt projektu a propojení s databází

Po úspěšné instalaci je ***doporučené se přesunout do adresáře***, ve kterém chce vývojář dbt projekt vytvořit. To můžeme provést klasickým příkazem cd v terminálu. V autorovo případě se jedná o cestu:

[5] Seznam dalších podporovaných databází je možné nalézt v on-line dokumentaci na odkazu: https://docs.getdbt.com/docs/trusted-adapters.

Výpis 19 Cesta do složky pro nově vytvořený dbt projekt

```
cd /Users/matejnovak/Projects
```

V tento moment ***spustí příkaz dbt init, kterým inicializujeme nový dbt projekt.*** Tento příkaz následně provede základním nastavením projektu, jako je název projektu, volba databáze a nastavení jejího připojení:

Výpis 20 Příkaz pro založení projektu medical_project v dbt

```
dbt init medical_project
```

V rozhraní je potřeba ***vybrat databázi dle možnosti nainstalovaných adapterů*** (dbt samo nabídne očíslované adaptéry).

```
18:52:15  Setting up your profile.
Which database would you like to use?
[1] bigquery
[2] snowflake
[3] redshift
[4] postgres

(Don't see the one you want? https://docs.getdbt.com/docs/available-adapters)

Enter a number:
```

Pro BigQuery autor zvolí číslo 1. V dalším kroku je potřeba vybrat typ autentizace – v autorově případě je ***využit servisní účet*** (co je servisní účet a jak ho nastavit lze vyčíst z kapitoly 4.1.3):

- oauth,
- service-account.

Po zvolení možnosti servisního účtu ***je nutné ho stáhnout a zadat lokální cestu k souboru*** (viz. kapitola 4.1.3), kde se servisní účet nachází. Po nastavení cesty vývojář dále nastaví:

- projektové ID z GCP,
- název datasetu, pod kterým chcete ukládat modely,
- počet vláken, které mohou paralelně běžet,
- timeout databáze (ve vteřínách),
- lokaci.

Pro ověření, že vše správně funguje se autor v ***terminálu posune do adresáře*** s projektem a následně využije dbt příkaz pro kontrolu připojení.

```
cd medical_project


dbt debug
```

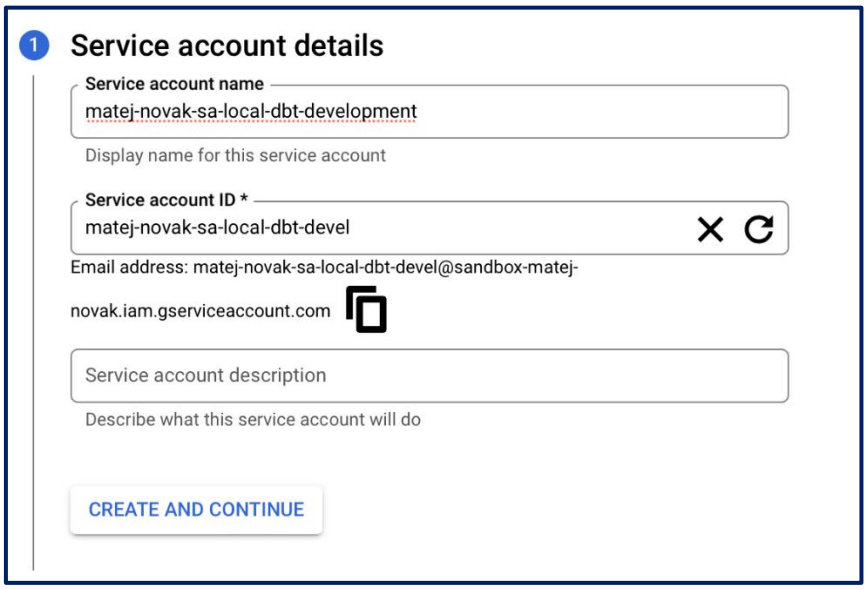
V případě, že vše proběhne v pořádku, uživatel obdrží informace o spojení s databází zakončené následující hláškou.

Obrázek 3-1: Ukázka výstupu při úspěšném dotázání na kontrolu propojení s databází

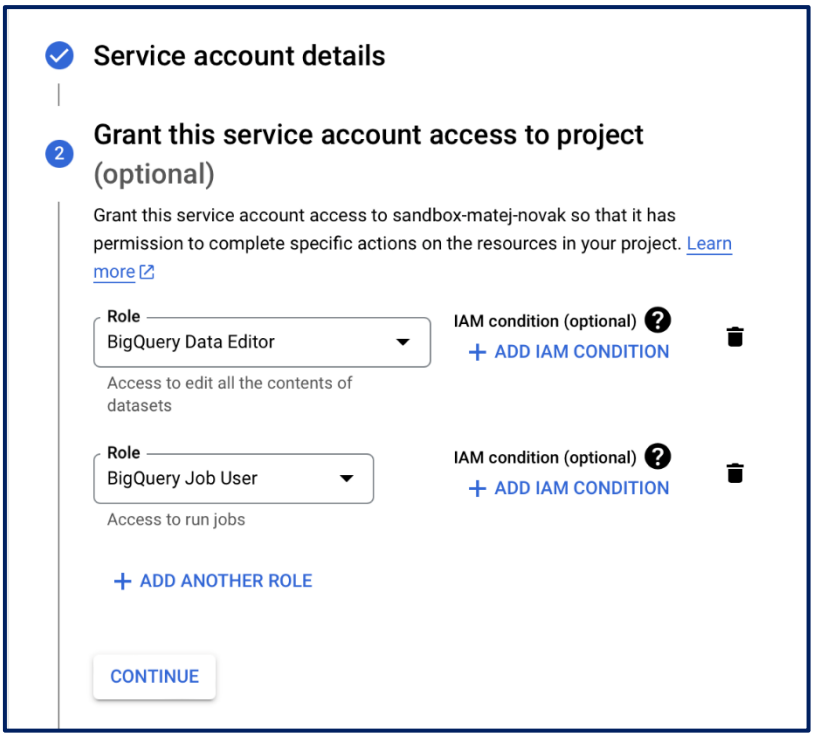
Je důležité zmínit, že ***toto nastavení a propojení s databází slouží pouze k lokálnímu vývoji***, nikoliv k automatickému orchestrovanému řešení (více o nastavení v kapitolách 4.1.3 a 4.8).

3.1.3 Servisní účet a jeho nastavení

Servisní účet v GCP je speciální typ účtu, který aplikace nebo služby používají ***k ověřování a získávání přístupu k API a službám GCP***. Servisní účet funguje jako „technický uživatel“ pro aplikace. Má přidělená oprávnění k přístupu k různým zdrojům v GCP podle nastavených rolí. Servisní účet se ***často konfiguruje s klíčem (JSON soubor), který aplikace využívá k autentizaci*** a získání přístupu k prostředkům v GCP. Takový servisní účet lze založit přímo v Google Cloud Consoli v sekci IAM > Servisní účty (Service Accounts) > Create Service Account.

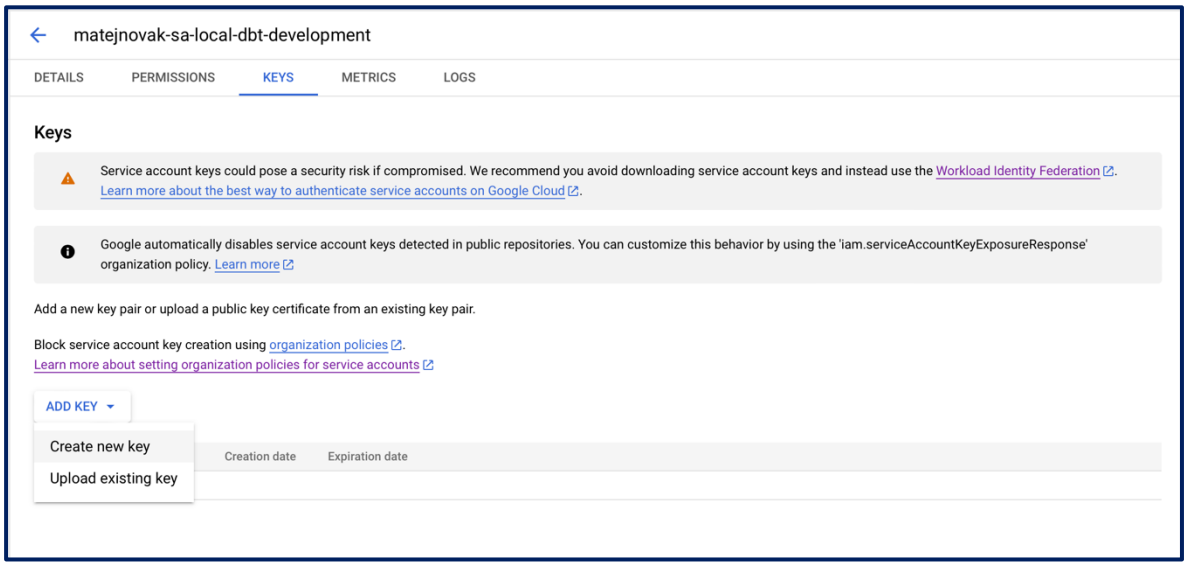


Obrázek 3-2 Pojmenování servisního účtu



Obrázek 3-3 Přiřazení vhodných rolí pro lokální vývoj dbt

V momentě, kdy je vytvořený servisní účet, v sekci Service Accounts se ***přes jeho název proklikneme a v horní části obrazovky zvolíme Keys, následně Add key a Create new key***. Ten stáhneme ve formátu JSON. Cestu ke staženému souboru pak zadáme do námi vytvořeného profiles.yml.

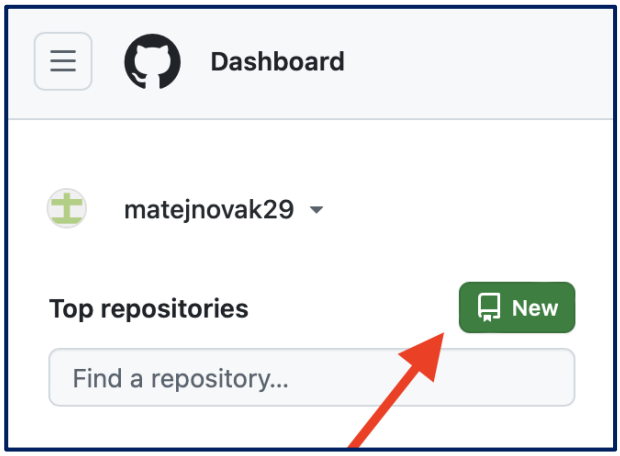


Obrázek 3-4 Vytvoření klíče k servisnímu účtu

3.2 *Nastavení git repozitáře a propojení s dbt projektem*

Pro správné verzování dbt projektu je nutné nejprve ***vytvořit prázdný repozitář*** v rozhraní GitHubu. Tento krok zajistí, že ***dbt projekt bude verzován a uložen na vzdáleném úložišti***.

V GitHubu v levé části vidíme „Top repositories“, zde stačí kliknout na tlačítko New. V průvodci pak zadáme ***název repozitáře*** (v autorovo případě medical_dbt_project), případně nastavíme viditelnost jako veřejnou nebo soukromou, a necháme repozitář prázdný bez README či licenčních souborů.



Obrázek 3-5 Ukázka vytvoření nového GitHub repozitáře

Tento prázdný repozitář medical_dbt_project ***následně propojíme s naším dbt projektem pomocí příkazů v terminálu.***

Výpis 21 Nastavení cesty v terminálu do dbt projektu

```
cd /Users/matejnovak/Projects/medical_project
```

Následně provede ***příkaz pro založení gitu*** a propojení nově vytvořeného repozitáře s vytvořeným dbt projektem.

Výpis 22 Založení git repozitáře na lokálním prostředí

```
git init
```

Výpis 23 Připojení dbt projektu k nově vytvořenému git repozitáři.

```
git remote add origin https://github.com/matejnovak29/medical_dbt_project.git
```

Nastaví nově ***vytvořenou branch na název main*** (či development/production), zde záleží na konkrétním případu.

Výpis 24 Přejmenování větve na main

```
git branch -M main
```

Připraví veškeré ***lokálně uložené soubory*** pro push (lze považovat za inicializaci projektu).

Výpis 25 První commit v git repoziáři - založení projektu

```
git add .
git commit -m "dbt init project"
```

Nahraje ***změny a branch*** do vzdáleného repozitáře.

Výpis 26 Propojení branche se vzdáleným repozitářem

```
git push -u origin main
```

Na základě poznatků kapitola poskytuje osvědčený postup, který může sloužit jako šablona či inspirace pro další projekty datových skladů s využitím nástroje dbt. Tento přístup zahrnuje klíčové metody, jako je strukturování datových vrstev, kde L2 vrstva využívá princip union relations k propojení dat, snapshoty pro zajištění sledování změn v čase a průběžná dokumentace pro zajištění konzistentního a jasného popisu všech metrik a procedur. Tyto přístupy umožňují efektivní správu datového skladu, zajišťují přesnost dat a přinášejí transparentnost pro spolupráci s byznysem.

4. Závěry

Text tvoří jeden z doplňujících textů k základnímu textu „AF_II_05_Podnikova analytika“ řady textů „IT a anatomie firmy“ v tomto případě zaměřený na produkt dbt. V tomto případě ***bylo cílem presentovat celkový pohled na uvedený systém*** i vytvořit předpoklady a podklady pro řešení aplikací na jeho základě.

Účelem tohoto textu bylo vymezení pouze hlavních principů dbt. V souvislosti s ostatními texty jsme uvedli, že smyslem uvedeného pojetí a přístupu k analýze je přispět ke ***zvyšování kvality a výkonu*** práce analytiků, manažerů a analytiků vývojářů v reálné praxi. V případě tohoto textu to platí nemalou měrou. Jestli i tento text takový příspěvek představuje, pak se jeho smysl podařilo naplnit.

5. Zdroje

BENIN, Drew. Online rozhovor o dbt. Praha, 17.10.2024.

AIRFLOW, [b.r.]. *DAGs — Airflow Documentation* [online] [vid. 2024-10-10]. Dostupné z: https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/dags.html

AWATI, Rahul, 2022. What is Data Definition Language (DDL) and how is it used? *WhatIs* [online] [vid. 2024-10-14]. Dostupné z: https://www.techtarget.com/whatis/definition/Data-Definition-Language-DDL

BYRUM, Amy, 2022. *dbt transform* [online]. 13. duben 2022. [vid. 2024-10-07]. Dostupné z: https://github.com/dbt-labs/dbt-core/blob/202cb7e51e218c7b29eb3b11ad058bd56b7739de/etc/dbt-transform.png

GETDBT, 2024a. *About adapter object* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/reference/dbt-jinja-functions/adapter

GETDBT, 2024b. *About dbt models* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/build/models

GETDBT, 2024c. *About model governance* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/collaborate/govern/about-model-governance

GETDBT, 2024d. *About profiles.yml* | *dbt Developer Hub* [online] [vid. 2024-08-05]. Dostupné z: https://docs.getdbt.com/docs/core/connect-data-platform/profiles.yml

GETDBT, 2024e. *Add data tests to your DAG* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/build/data-tests

GETDBT, 2024f. *Add Seeds to your DAG* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/build/seeds

GETDBT, 2024g. *Add snapshots to your DAG* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/build/snapshots

GETDBT, 2024h. *Analyses* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/build/analyses

GETDBT, 2024i. *Catalog JSON file* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/reference/artifacts/catalog-json

GETDBT, 2024j. *Configuring test `severity`* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/reference/resource-configs/severity

GETDBT, 2024k. *Documentation* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/build/documentation

GETDBT, 2024l. *Jinja and macros* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/build/jinja-macros

GETDBT, 2024m. *Manifest JSON file* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/reference/artifacts/manifest-json

GETDBT, 2024n. *Model access* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/collaborate/govern/model-access

GETDBT, 2024o. *Model versions* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/docs/collaborate/govern/model-versions

GETDBT, 2024p. *persist_docs* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/reference/resource-configs/persist_docs

GETDBT, 2024q. *Run results JSON file* | *dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/reference/artifacts/run-results-json

GETDBT, 2024r. *Syntax overview | dbt Developer Hub* [online] [vid. 2024-09-29]. Dostupné z: https://docs.getdbt.com/reference/node-selection/syntax

HANDY, 2020. A License to build the future. *dbt Labs* [online] [vid. 2024-10-19]. Dostupné z: https://www.getdbt.com/blog/fishtown-analytics-announces-29-5m-fundraise

HANDY, Tristan, 2017. One year in: we're still in business. *dbt Labs* [online] [vid. 2024-10-19]. Dostupné z: https://www.getdbt.com/blog/one-year-in-we-re-still-in-business

HANDY, Tristan, 2018. On two years of dbt. *dbt Labs* [online] [vid. 2024-10-19]. Dostupné z: https://www.getdbt.com/blog/on-two-years-of-dbt

CHACON, Scott, 2009. *Pro Git*. B.m.: CZ.NIC. ISBN 976-80-904248-1-4.

JINJA, [b.r.]. *Jinja — Jinja Documentation (3.1.x)* [online] [vid. 2024-10-10]. Dostupné z: https://jinja.palletsprojects.com/en/3.1.x/

JONES, Andrew, 2023. *Driving Data Quality with Data Contracts*. B.m.: Packt Publishing. ISBN 978-1-83763-624-2.

KIMBALL, Ralph a Ross MARGY, 2011. *The Data Warehouse Toolkit*. B.m.: Wiley. ISBN 978-1-118-08214-0.

LTD, Red Gate Software a Matt HILBERT, 2024. The risks – and rewards – of using production data for testing. *Redgate* [online]. [vid. 2024-09-29]. Dostupné z: https://www.red-gate.com/blog/database-devops/the-risks-and-rewards-of-using-production-data-for-testing

MACHADO, Rui Pedro a Helder RUSSA, 2023. *Analytics Engineering with SQL and dbt*. B.m.: O'Reilly Media, Inc. ISBN 978-1-09-814238-4.

MOSES, Barr, Lior GAVISH a Molly VORWERCK, 2022. *Data Qualify Fundamentals*. B.m.: O'Reilly Media. ISBN 978-1-09-811199-1.

OPENSOURCE.COM, [b.r.]. *What is open source? | Opensource.com* [online] [vid. 2024-10-10]. Dostupné z: https://opensource.com/resources/what-open-source

TRISTAN, Handy, 2019. Three years in: things are heating up. *dbt Labs* [online] [vid. 2024-10-19]. Dostupné z: https://www.getdbt.com/blog/three-years-in-things-are-heating-up

ZAGNI, Roberto, 2023. *Data Engineering with Dbt*. B.m.: Packt Publishing. ISBN 978-1-80324-188-3.