FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Využití metodiky Data vault pro ETL procesy
Student:	Peter Kolárovec
Vedoucí:	Ing. Lukáš Hrnčíř
Studijní program:	Informatika
Studijní obor:	Znalostní inženýrství
Katedra:	Katedra aplikované matematiky
Platnost zadání:	Do konce letního semestru 2019/20

Pokyny pro vypracování

Prostudujte metodiku data vault z pohledu využitelnosti na BI/DWH projektu. Zaměřte se na využití této metodiky pro datové modelování, ETL procesy, auditovatelnost a efektivitu rozvoje celého řešení. Navrhněte změny v ETL nástroji, který se aktuálně využívá na BI/DWH projektech tak, aby byly v souladu s touto metodikou.
Na základě těchto návrhů změn proveďte změny v implementaci ETL nástroje a proveďte otestování nových funkcionalit.

Teoretickým výstupem bakalářské práce je studie využitelnosti metodiky data vault při implementaci datových skladů a v praktické části budou tyto závěry ověřeny na základě provozu upraveného ETL nástroje.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Karel Klouda, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 10. prosince 2018

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalárska práca

Využití metodiky Data vault pro ETL procesy

Peter Kolárovec

Katedra aplikované matematiky
Vedúci práce: Ing. Lukáš Hrnčíř

13. mája 2019

Poďakovanie

V prvom rade, sa chcem poďakovať mojim rodičom za ich podporu počas celého bakalárskeho štúdia. Rovnako tak, aj mojej sestre za dodávanie inšpirácie a motivácie. Poďakovanie patrí aj mojím spolužiakom/kolegom, za ich ochotu vždy nezištne pomôcť pri osobných alebo študijných problémoch. Ďakujem aj môjmu vedúcemu Ing. Lukášovi Hrnčířovi a celému tímu mojich kolegov v Neit Consulting s. r. o. za túto tému na bakalársku prácu a ich dôveru vloženú do mňa na jej vypracovanie.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k užívaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo užívať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela (vrátane komerčného využitia). Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Prahe 13. mája 2019 ……

Odkaz na túto prácu

Kolárovec, Peter. *Využití metodiky Data vault pro ETL procesy.* Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Abstrakt

V práci sa zaoberám metodikou modelovania Data Vault v spojení s budovaním dátového skladu a prenosmi dát – ETL procesmi. Analyzujem možnosti rozvoja interného ETL nástroja firmy Neit Consulting s. r. o. Ako jej zamestnanec predstavujem nové, nezávislé riešenie pre dynamické generovanie spustiteľného SQL kódu na prenos dát. Generovanie prebieha na základe dát uložených v relačnej databáze a znalostných modulov, ktoré obsahujú volania API funkcií pre dynamické dopĺňanie informácií. Toto rozšírenie je poskytnuté voľne a môže slúžiť komukoľvek ako základ pre jeho ETL nástroj.

Kľúčové slová dátový sklad, automatizácia, ETL, Data Vault, Python, SQL, metadáta, dynamická interpretácia

Abstract

In this thesis, I deal with the Data Vault modeling technique in conjunction with building a data warehouse and transferring the data – ETL processes. I analyze the development opportunities of intern ETL tool made by Neit Consulting Ltd. As its employee, I present a new, independent solution for dynamically generated runnable SQL code transferring the data. The generating is based on metadata stored in a relational database and knowledge modules, which contain API calls for dynamical insert of the data. This extension is free to use, and it may serve to whomever as a base for their ETL tool.

Keywords data warehouse, automatization, ETL, Data Vault, Python, SQL, metadata, dynamic interpretation

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Metodika Data Vault ako alternatívna ku metodikám Kimball alebo Inmon začína byť čím ďalej, tým častejšie používaná v praxi pri implementácií dátových skladov. Vďaka jej elementárnym princípom existuje priestor na určitý spôsob automatizácie.

Praktickým výstupom tejto práce je rozšírenie interného nástroja používaného na ETL (extract, transform, load) procesy, ktorý je využívaný na projektoch pri budovaní dátových skladov. Tieto rozšírenia budú zvyšovať efektivitu, použiteľnosť a kompatibilitu spomínaného nástroja.

Téma mi bolo ponúknuté zamestnávateľom z dôvodu zvýšenie konkurencieschopnosti tohoto nástroja pri nasadení u zákazníkov, ktorý uvažujú o novom dátovom sklade.

V práci sa zaoberám možnosťami rozšírenia uvedeného nástroja a ich implementáciou v spojitosti so zásadami metodiky Data Vault.

Táto práca pokračuje v nasledujúcej štruktúre. V prvej kapitole začínam pojmom data warehouse – ako najviac používaním výrazom v tejto práci. V druhej kapitole vysvetľujem základné princípy a stavebné jednotky modelovacej metodiky Data Vault. Tretia kapitola zas uvádza do problematiky ETL procesov. Interný nástroj sa zaoberá práve ETL procesmi a tak súčasťou tejto kapitoly je aj analýza nedostatkov súčastného riešenia a návrh zmien. Zvyšok práce popisuje implementáciu týchto navrhnutých zmien a následne ich testovanie.

Ako autor vypracovávam túto prácu v spolupráci s firmou Neit Consulting s. r. o. ako jej zamestnanec.

Ciele práce

Teoretickým cieľom práce je zoznámenie sa s metodikou Data Vault v spojení s vývojom dátových skladov. Práca sa zameriava na základné princípy dátového modelovania, použiteľnosť v ETL procesoch, auditovateľnosť a efektivitu vývoja celkového riešenia v spojení s touto metodikou.

Praktická časť zahŕňa návrh a implementáciu rozšírenia v internom podnikovom nástroji pre ETL procesy, ktorý sa aktuálne využíva pre BI/DWH (business intelligence/data warehouse) projekty, tak aby bol v súlade s touto metodikou. Súčasťou implementácie je aj vývin ETL modulov slúžiacich na prenos dát do/z jadra dátového skladu. Vyvinuté zmeny je následne nutné otestovať. Testovaním je požadované potvrdiť korektne fungujúcu implementáciu zdrojového kódu a správne vytvorené ETL moduly na prenášanie dát.

Kapitola 1

Data warehousing

Je ťažké nezačať túto publikáciu takou zaužívanou vetou, akou je: „*Jedným z najdôležitejších majetkov akejkoľvek spoločnosti sú jej dáta.*“[1] (preklad autora) Táto veta uvádza, myslím si, že legendárnu knihu celého odvetvia DWH (data warehouse).

Cieľom tejto kapitoly je zoznámiť čitateľa so vznikom a modelovaním dátových skladov. Vo všetkých nasledujúcich častiach práce pod pojmom DWH uvažujem v zmysle troch oddelených vrstiev, ktorých význam uvádzam ďalej v tejto kapitole.

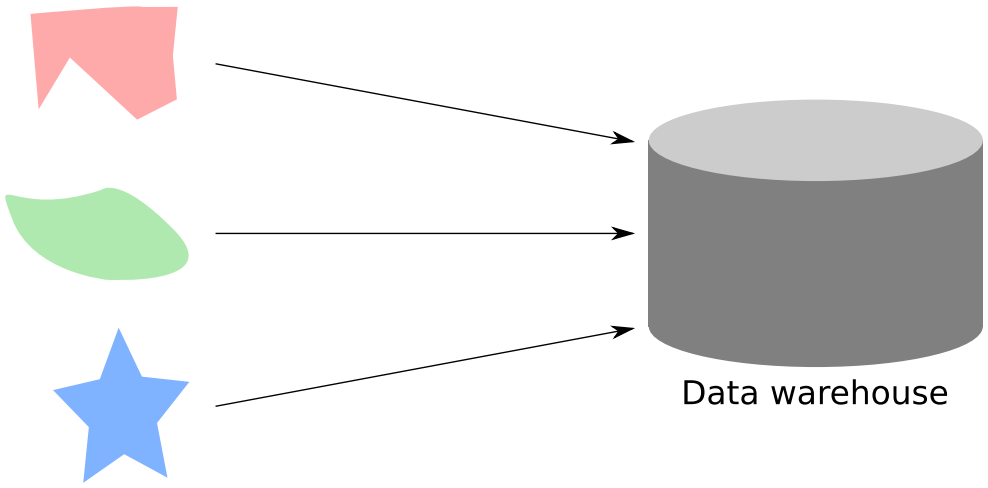
1.1 Vznik dátových skladov

S vývojom informačných technológií koncom 20. storočia dochádzalo aj k efektívnejším spôsobom, akými je možné ukladať dáta. Počnúc diernymi štítkami, magnetickými páskami až k ukladaniu na disk. Podľa [2] to všetko viedlo k vzniku DBMS (database management system), ktoré nad dátami zaobstaráva tieto operácie :

- ukladanie
- prístup
- úpravu
- mazanie

Vývojom procesnej sily s použitím DBMS bol umožnený vznik nového typu systému, ktorým je OLTP (online transaction processing). Dôsledkom bol začiatok interaktívneho využitia počítača, ktoré dovtedy nebolo možné. Začali sa objavovať prvé rôznorodé rezervačné, bankové a iné systémy. [2]

Technológie však nie sú všetko a k dátam bolo potrebné vytvoriť jednotný prístup. A práve tu začína éra dátových skladov s ich jedinou verziou pravdy s podporou historizácie.



Obr. 1.1: Zdrojové systémy vstupujúce do dátového skladu. [2]

1.2 Modelovanie dátových skladov

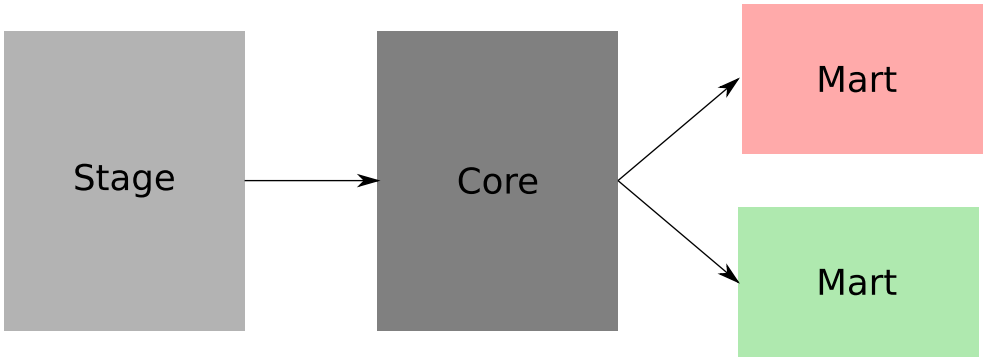
Dátové sklady sa od seba navzájom líšia – niektoré viac, iné menej – ale hlavným rozoznávacím príznakom je metodika modelovania. V praxi sa využívajú tri hlavné modely:

- dimenzionálny
- relačný
- Data Vault

O výhodách/nevýhodách a porovnaniach jednotlivých modelov by sa dala napísať celá ďalšia bakalárska práca, preto v nasledujúcich podsekciách predstavujem abstraktnejší pohľad na dátový sklad po jeho jednotlivých vrstvách bez špecializácie na modelovaciu metodiku.

1.2.1 Vstupná vrstva

Táto vrstva sa bežne označuje/pomenováva ako *stage*. Pod týmto pojmom si môžete predstaviť „pristávaciu plochu“ pre dáta, ktoré ukladáte. Štandardne do stage – ako 1. vrstvy dátového skladu – vstupujú dáta rôznej povahy, obrázok 1.1. Nemusia pochádzať len z rôznych databáz, ale napríklad aj zo súborov. K zdrojovým dátam sa často pripájajú metadáta , napr. označenie zdroja, dátum a čas vloženia/zmeny,...



Obr. 1.2: Vrstvy dátového skladu

Dáta v tejto vrstve môžu byť tiež uložené ako textové súbory (so štruktúrou riadkov a stĺpcov) alebo vo formáte XML (extensible markup language) a nemusí to byť len relačná databáza. [3]

1.2.2 Business jadro

Podstata celého konceptu a procesu tvorby dátových skladov leží práve v tejto vrstve. Z praktických dôvodov teda označujem túto vrstvu ako *core*, teda jadro DWH. Úspech celého projektu závisí na správnej implementácii – technickej, ale aj businessovej – core vrstvy. Zaručuje totižto jednotný pohľad na podnikové dáta.

Predpokladom je samozrejme ucelená definícia sémantiky, za účelom správnej integrácie dát zo zdrojových systémov. Tú ďalej, podľa [3], dosiahneme normalizáciami:

- mennej konvencie
- kódových hodnôt
- dátových typov
- klasifikácie dát

Na jadro dátového skladu môžeme mať aj ďalšie požiadavky:

- historizácia a časové rezy
- agilita
- úplnosť
- overiteľnosť [4]

Konceptuálne modelovanie jadra, tak isto ako iné procesy modelovania a dizajnu, sú tvorivé činnosti. Na počiatku stojí koncept a požiadavky. Iteratívnym spôsobom sa teda prepracovávame k správnemu a konečnému návrhu, ktorý je na prvý pokus skôr nemožný ako možný. [4]

1.2.3 Reportovacia vrstva

Náročná cesta budovania jadra dátového skladu sa nepodstupuje len tak pre nič za nič. Ako zadosťučinenie môže poslúžiť, keď sa dáta dostanú k ich používateľom, ktorí ich preberajú práve z poslednej, tzv. *(data)martovej* vrstvy. Pre uspokojenie rozličných pohľadov na agregované dáta, vznikajú jednotlivé datamarty špeciálne pre konkrétnych užívateľov/oddelenia. Ako som v obrázku 1.2 naznačil, datamarty sú navzájom vnímané ako oddelené „databáze“. V praxi si napríklad zamestnanec marketingového oddelenia nemôže prezerať datamart finančnej sekcie.

Kapitola 2

Metodika Data Vault

Metodika databázového modelovania Data Vault (konkrétne jej verzia 2.0) ako taká, bola zadefinovaná a zdokumentovaná v roku 2013. Jej pôvodný koncept však siaha už do roku 2000. Za jej zakladateľa je označovaný Dan Linstedt.

V tejto kapitole predstavím jej hlavné vlastnosti a výhody. Okrem toho vysvetlím aj jej základné modelovacie princípy – ktoré sú stavebnou jednotkou jadra dátového skladu – a ktorých pravidlá modelovania sú postavené nad ktorékoľvek iné.

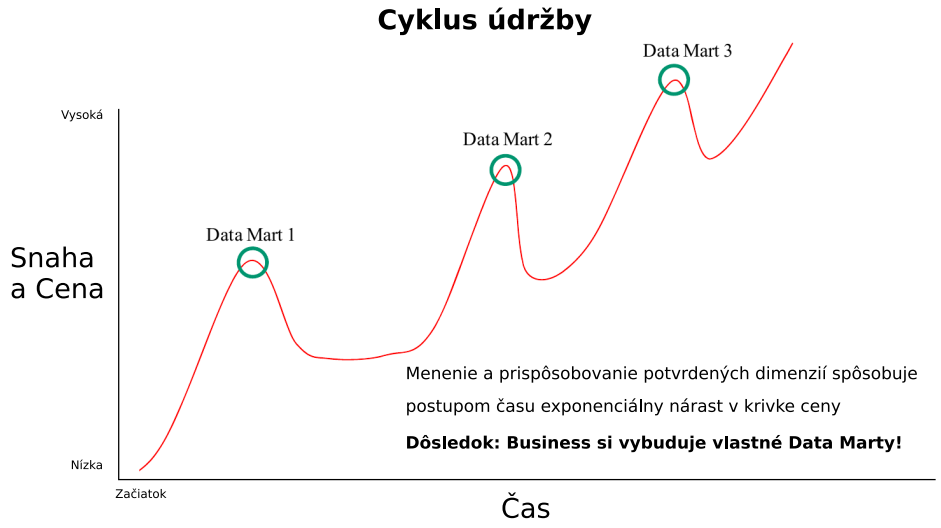
2.1 Využiteľnosť metodiky v dátovom sklade

„*Metodika Data Vault je veľmi efektívna pre budovanie dátových skladov, pretože je optimalizovaná na integráciu, historizáciu a agilný vývoj.*“ [4] (preklad autora) Tieto výhody môžu projekty veľmi zefektívniť. Práve preto popíšem tieto jej výhody v nasledujúcich podsekciách.

2.1.1 Integrovanosť

Základom správne prevedenej integrácie podnikových dát je zjednotená predstava businessových konceptov. Jednotlivé atribúty v rôznych zdrojových systémoch môžu mať iný názov alebo mierne odlišný význam. Cieľom integrácie je tieto rozdiely zjednotiť.

Metodika Data Vaultu je výborne pripravená pre najvyšší stupeň integrácie – jedna tabuľka obsahuje dáta zo všetkých zdrojových systémov bez zbytočnej redundancie dát. Umožňuje to fakt, že každý záznam obsahuje metadáta o jeho pôvode – ID zdrojového systému. ETL (extract trasform load) proces core-core 3.2 tak môže záznamy deduplikovať a zjednotiť cez všetky zdrojové systémy a priradiť im interné, integračné ID zdrojového systému.



Obr. 2.1: Vývoj ceny neagilného projektu. [5] (preklad autora)

2.1.2 Historizácia

S predchádzajúceho textu je jasné, že historizácia je jedna z najdôležitejších vlastností každého funkčného dátového skladu. Podstatným znakom metodiky Data Vault je tzv. *all data all time* [4] – konkrétne v konštrukcií satelitov 2.2. Tento stav dosiahneme obmedzujúcou podmienkou DML (Data Manipulation Language) príkazov v core vrstve – povolený je jedine `INSERT`. V jadre sa teda nachádzajú všetky údaje, ktoré boli kedy nahrané. Tak môžeme dostať platné údaje ku ktorémukoľvek časovému rezu nad jadrom dátového skladu. Vylepšenia na šetrenie miesta a zmenšenie počtu uložených/vkladaných záznamov samozrejme umožňujú dáta vkladať len ak bol zmenený ktorýkoľvek údaj.

2.1.3 Agilný vývoj

Bez agilného prístupu k problému so stúpajúcou cenou ako na grafe 2.1, zákazník začne obchádzať dodávateľa a z kedysi krásneho fungujúceho dátového skladu vznikne moloch odsúdený k záhube. [5]

Ako som už v časti 1.2 prvej kapitoly spomínal, implementácia dátového skladu je inkrementálny proces, výhodou teda môže byť, ak je od začiatku pripravená na zmeny. Pribudnúť môžu zdrojové systémy, pravidlá, transformácie, atribúty,...

Výhoda metodiky Data Vault sa v spojení s touto vlastnosťou prejavuje pomocou jej stavebných jednotiek, ktoré sú takzvanou *unified decomposition* businessových entít. Každá stavebná jednotka má svoj jednoznačný účel. Pri-

dať nový businessový koncept alebo zdrojový systém a previazať ho so zvyškom skladu – tak aby sa to prejavilo na výstupe – môže byť tým pádom otázkou hodín miesto dní. [5]

2.2 Základné princípy modelovania

Modelovanie jadra dátového skladu s použitím metodiky Data Vault je zamerané hlavne na dizajn. Podnikové entity sú rozdelené do týchto základných štruktúr:

- Hub
- Satellite
- Link

Na dodržiavaní tejto fundamentálnej dekompozície je postavený úspech dátového skladu. Konzistencia pri používaní týchto konštruktov je nadradená ktorýmkoľvek iným pravidlám tejto metodiky a iným ovplyvňujúcim faktorom. [4]

Ukážkovú dekompozíciu businessovej entity je možné vidieť na obrázku 2.2. Konštrukty majú niekoľko spoločných technických atribútov:

Surrogate key (SK) - unikátne ID generované databázou

Hash key (HK) - ID entity transformované hash funkciou

LoadDatetime - časová značka, kedy bol záznam nahraný

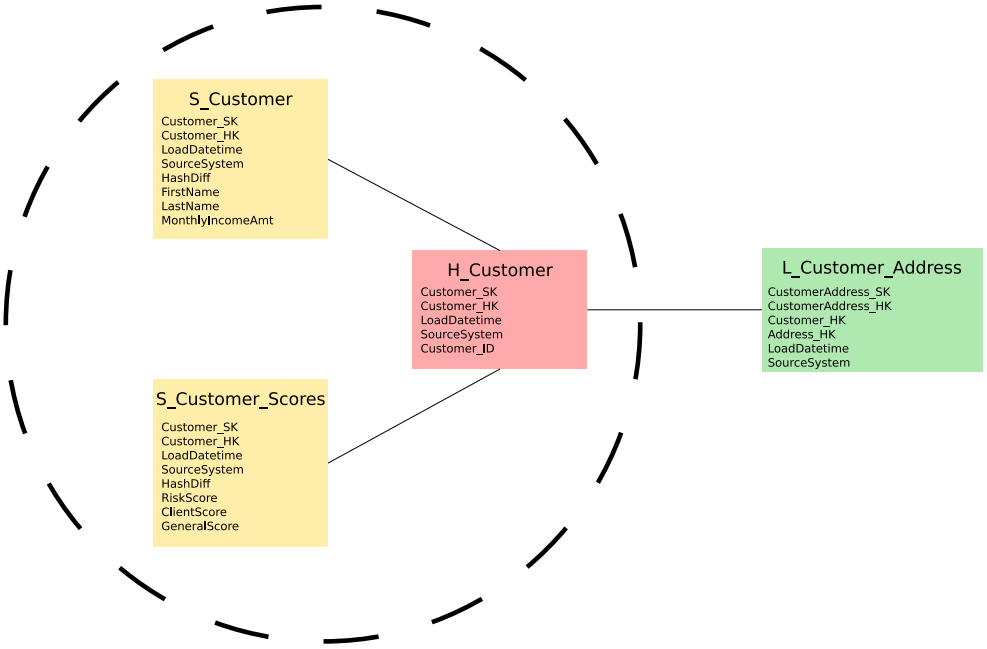
SourceSystem - ID zdrojového systému odkiaľ záznam pochádza

HashDiff - agregovaný obsah atribútov transformovaný hash funkciou

2.2.1 Hub

Tento typ tabuľky reprezentuje businessovú entitu ako napríklad zmluva, majetok, platba,...Okrem technických atribútov nesie záznam o *Natural key* teda prirodzenom unikátnom identifikátore konkrétnej entity. Kardinalita tabuľky hub je $1 : 1$. Tým je dosiahnuté, že v technických atribútoch sú údaje o prvom príchode entity do dátového skladu. [4]

Často využívanou technikou je na tento prirodzený identifikátor aplikovať hash funkciu a výsledok uložiť v samostatnom atribúte. Tento hash potom slúži pre referencovanie businessového konceptu v ostatných tabuľkách jadra dátového skladu. Okrem toho, použitie hash ako identifikátoru zefektívňuje spájanie tabuliek. [6]



Obr. 2.2: Businessová entita zákazníka a jej okolie. [4]

2.2.2 Satelit

Keďže hub udržuje len prirodzený kľúč, všetky ostatné kontextové informácie o entite musia byť uložené v satelitoch naň pripojených. Pri vytváraní satelitov nastáva práve spomínaná tvorivá činnosť. Príčinou je fakt, že hub môže mať viacero satelitov a rozdelenie jednotlivých atribútov po satelitoch je len na dátovom analytikovi. Satelit nemusí rozvíjať svojimi informáciami len hub, môže byť napojený aj na záznamy tabuľky typu link.

Kardinalita satelitov je $1 : N$. Tabuľka tohoto typu ako jediná udržiava históriu. Zachytáva všetky zmeny na svojich atribútoch a preto súčasťou kľúča je aj časová značka, čo umožňuje v princípe rovnakú historizáciu ako v pomaly meniacej sa dimenzií typu 2 z dimenzionálneho modelu. [4]

2.2.3 Link

Poslednou ale taktiež podstatnou Data Vault konštrukciou je link. Tabuľka zabezpečuje prepojenie medzi dvoma (alebo aj viacerými) business entitami väzbou $N : M$. Podobne ako hub neuchováva žiadne kontextové informácie a zachytáva výhradne prvé objavenie väzby medzi entitami. Ďalšie údaje sú teda uložené na satelite jednotlivých entít, ktoré link spája alebo na satelite, ktorý patrí linku.

Implicitne stanovená kardinalita $N : M$ prináša svoje výhody ale aj nevýhody. Medzi hlavné výhody patrí pripravenosť na zmenu väzby bez nutnosti zmien v dátovom modele, tým pádom sa architekt dátového skladu môže viac sústrediť na identifikovanie prirodzených business väzieb. Medzi nevýhody zas patrí určite to, že na prvý pohľad nie je jasná skutočná kardinalita väzby. Ako ďalšie mínus môže pôsobiť pokušenie modelovať túto prirodzenú väzbu ako samostatnú entitu. Link ako taký sám o sebe nemá význam a tak naopak pojmy ako udalosť, transakcia alebo predaj nemodelujeme pomocou linku ale ako samostatný business koncept. [4]

Kapitola 3

ETL procesy

V úvodnej kapitole 1 som spomenul trojvrstvové zloženie dátového skladu. Dáta sa do vrstiev však nedostanú samé a preto sa tam nejakým spôsobom musia preniesť. Pre tieto potreby existujú rôzne komerčné nástroje, ale viacero firiem zaoberajúcich sa touto problematikou má pre tieto účely svoj vlastný nástroj. V tejto kapitole popisujem štruktúru ETL procesu ako takého – keďže ide o skratku zo slov *extract, transform, load* (získavanie, transformácia, ukladanie – preklad autora). Tieto procesy môžeme ďalej rozdeliť podľa toho, medzi ktorými vrstvami dáta prenášajú.

Keďže táto práca ďalej pokračuje kapitolami patriacimi do praktickej časti, tak v poslednej sekcií tejto kapitoly zanalyzujem nedostatky aktuálneho riešenia nástroja pre ETL procesy, ktoré poslúžia ako miesta pre moje rozšírenie funkčnosti v nasledujúcich kapitolách.

3.1 Štruktúra ETL procesu

Ako bolo v úvode tejto kapitoly spomenuté, ETL je skratka z anglických slov *extract, transform, load*, čo značí 3 hlavné fázy tohoto procesu – získavanie, transformácia, ukladanie. Jedna s druhou sú previazané a pokiaľ ktorákoľvek z nich neskončí správne, môžme považovať celý proces za chybný.

3.1.1 Získavanie dát

Akékoľvek prenášané dáta musíme nejakým spôsobom načítať. Spôsob akým to vykonáme, záleží aj na implementácií vrstvy, odkiaľ dáta načítame. Dôležité je však to, že preberáme dáta tak, ako sú. Pokiaľ sa jedná o dáta uložené v dátabázových objektoch, môžte si túto fázu predstaviť ako DML príkaz `SELECT`.

Niekedy takýto pokyn môže trvať dlhšie, preto častým trikom býva tieto nahrané dáta uložiť do dočasného databázového objektu. Tým zabezpečíme

to, že pri nesprávnom prevedení alebo páde procesu môžeme túto prvú fázu preskočiť – pokiaľ bola vykonaná správne a chyba nenastala práve tu.

3.1.2 Transformácia dát

Správne načítané dáta v dočasnom dátovom objekte v druhej fáze transformujeme do podoby, ktorá je požadovaná v cieľovej tabuľke. Základnými transformáciami sú teda zmeny dátových typov, napríklad číselné vyjadrenie dátumu do databázovo špecifického dátového typu pre dátum. Ďalšími transformáciami môžeme dáta čistiť, teda zbavovať zbytočných znakov, doplňovať chýbajúce hodnoty a podobne. Inými transformáciami zas dosahujeme businessovo požadované významy nahrávaného atribútu. Môže ísť o rôzne podmienky, orezávania, spájania,...

3.1.3 Ukladanie dát

Po správnom načítaní a transformácií dát je možné dáta nahrať do cieľovej tabuľky. Tá môže mať spoločnú štruktúru pre viacero zdrojových systémov a tým umožňovať najvyšší stupeň integrácie.

Záznam sa uloží podľa typu cieľovej tabuľky. Pri procesoch smerom do jadra dátového skladu sa uplatňujú princípy ukladania a historizácie, ktoré boli vysvetlené v kapitole 2.2. Ak ide o ETL proces medzi jadrom a reportovacou vrstvou, využijú sa zas hlavne princípy dimenzionálneho modelovania.

3.2 Data vault v spojení s ETL procesmi

Vrstva modelovaná metodikou Data Vault – keďže figuruje v prostrednej vrstve tejto architektúry – má špecifické ETL procesy smerujúce do nej ako aj z nej. Špeciálnym prípadom môže byť ešte ETL proces nad samotným jadrom dátového skladu. Cieľom tohoto procesu je integrovať dáta v „surovom“ jadre a vytvoriť tak jednotné business jadro. Iným použitím je zas predpočítanie rôznych hodnôt z už obsiahnutých atribútov pre zjednodušenie procesov smerujúcich z jadra.

3.2.1 Prenos záznamov Stage - Core

Záznamy nahrávané týmto typom ETL procesu majú za cieľ naplniť Data Vault stavebné štruktúry, ktoré sú popísané v časti 2.2. Okrem požadovaných prenášaných atribútov pridáva tento proces do týchto tabuliek taktiež spomínané meta údaje. Niekoľko krát bolo spomenuté, že historizácia prebieha v jadre dátového skladu. Z tohoto dôvodu chceme dáta ukladať a historizovať najskôr v stave v akom prišli zo zdrojového systému. Tým pádom dochádza len k transformácií dátových typov a orezávaniu zbytočných znakov (biele znaky zľava alebo sprava).

Priebeh nahrávania tabuľky hub a link je určitým spôsobom podobný. V tabuľkách sa objavuje len jeden unikátny záznam pri prvom výskyte ich jednoznačného identifikátoru (dvojice identifikátorov v prípade tabuľky link), dáta už obsiahnuté v týchto tabuľkách teda prejdú prvými dvoma fázami ETL procesu, no pri tretej budú odfiltrované a nebudú duplicitne uložené.

Nahrávaním dát do satelitov udržiavame históriu v našom dátovom sklade. Nový záznam je pre optimalizáciu počtu záznamov nahraný pri zmene ktoréhokoľvek preberaného atribútu. Za aktuálne platný považujeme najnovší záznam.

3.2.2 Prenos záznamov Core - Core

Tento typ ETL procesu je špecifický a pri niektorých (jednoduchších) implementáciách vôbec nie je nutné ho použiť. Využiteľnosť môže byť nasledujúca:

- Integrácia
  - deduplikácia záznamov naprieč zdrojovými systémami
  - zjednotenie businessového pohľadu na danú entitu
  - dopĺňanie chýbajúcich hodnôt
- Agregácia
  - predpočítavanie závislých hodnôt
- Iné požadované transformácie

Jadro dátového skladu môže byť teda aplikáciou týchto ETL procesov rozdelené na takzvané „surové“ a businessové. V prvej časti ukladáme históriu atribútov ako prišli, zatiaľ čo v druhej časti na tieto atribúty aplikujeme transformácie a agregácie alebo pracujeme s deduplikovanými záznamami.

3.2.3 Prenos záznamov Core - Mart

Užívatelia dátového skladu majú zvyčajne pristúp len do poslednej (reportovacej) vrstvy. Preto štruktúry v tejto vrstve sú vytvárané pre ich uspokojenie a plnené presne podľa ich požiadavkov. Všetky atribúty by už mali byť obsiahnuté v jadre a preto pri ETL procesoch tohoto typu je dôležité aplikovať požadované transformácie a dodržať spôsob ukladania do (väčšinou dimenzionálneho) modelu tejto vrstvy.

Zdrojom dát v jadre môže byť výhradne businessové jadro, ak v ňom však nie sú obsiahnuté všetky entity, tak jeho kombinácia s tým „surovým“.

3.3 Analýza súčastného stavu ETL Tools

V tejto sekcií priblížim nástroj vyvinutý zamestnancami – zároveň mojími kolegami – Neit Consulting s. r. o. slúžiaci na riadenie ETL procesov nazvaný „ETLTools“. Jedná sa o pomerne zložitú Python aplikáciu, ktorá sa stará o kompletné riadenie spomenutých procesov. Táto verzia pracuje aktuálne nad databázou Microsoft SQL Server.

Funkcionalita aplikácie spočíva vo vytvorení sady DML príkazov slúžiacich na prenos dát medzi vrstvami dátového skladu, na základe metadát. Tieto príkazy tak isto nad databázou spúšťa a ďalej ich riadi.

Nástroj však má svoje nedostatky, ktoré zhrniem ďalej v tejto sekcii a nájdem k nim efektívnejšie riešenie, ktoré v praktickej časti – nasledujúcej kapitole tejto práce – implementujem. Nutno podotknúť, že v mojej práci sa zaoberám spôsobom prenášania dát v ETL procesoch, teda časťou generovania sady príkazov – s využitím metodiky modelovania Data Vault – a nie ich riadením.

3.3.1 Nedostatky súčastného riešenia

Ako najväčší nedostatok aplikácie vidím vysokú previazanosť s databázou, kde sa tieto ETL procesy spúšťajú. Aj keď Microsoft SQL Server je v korporátnom prostredí populárny, pri použití iných technológií by bolo potrebné znovu implementovať niekoľko 100 - 1000 riadkových metód, ktoré generujú príkazy na prenos dát.

S tým je previazané aj ďalšie negatívum. Pri implementácii nových spôsobov prenášania dát musí dátový vývojár rozumieť pokročilej syntaxi programovacieho jazyka Python a nevystačí si so znalosťou SQL (Structured query language).

Pri súčasnom riešení je tak isto vyššia šanca zavedenia chyby do spôsobu prenášania dát, keďže tento spôsob nie je moc intuitívny a zo zdrojového kódu nie je vôbec na prvý pohľad jasné, akým spôsobom je tento proces vykonávaný.

Výhodou nového generovania príkazov by určite oproti starému riešeniu bola aj jeho nezávislosť na nasledujúcom riadení a spúšťaní ETL procesov.

3.3.2 Návrh zmien riešenia

Spôsob prenášania dát by mal byť vyňatý z ktoréhokoľvek zdrojového kódu – starého riešenia alebo môjho rozšírenia – a mal by byť jednoducho čitateľný a teda podobný jazyku SQL. Pomocou volania API (application programming interface) funkcií v tomto pseudo SQL kóde bude zaručená interpretácia tohoto kódu pre konkrétny typ prenosu dát. Táto interpretácia bude prebiehať pomocou môjho rozšírenia, ktoré bude mať zadefinované podporované volania týchto funkcií. Toto rozhranie by malo byť jednoducho rozšíriteľné pre potrebu pridania nových funkcionalít.

Týmto riešením je možné dosiahnuť stav, kde dátový vývojár nebude musieť rozumieť syntaxi jazyka Python a stačí mu porozumieť vystavenému rozhraniu, z ktorého poskladá pseudo SQL kód. Takýto SQL kód je následne aj veľmi jednoduché prepísať pre inú cieľovú databázu, za podmienky, že podporuje všetky použité funkcionality. Zvolené riešenie sa javí ako efektívnejšie, prehľadnejšie, jednoduchšie na správu a prípadnú opravu chýb.

Týmito vylepšeniami sa dá dosiahnuť nezávislosť nového riešenia a preto som sa so zamestnávateľom zhodol na publikácii tohoto riešenia voľne k použitiu. Moje riešenie môže poslúžiť komukoľvek ako základ pre jeho vlastný ETL nástroj.

Kapitola 4

Implementácia navrhnutých zmien

V tejto kapitole predstavím praktické riešenie, ktorým dosiahnem navrhnuté zmeny z konca predchádzajúcej časti. V jednotlivých sekciách popíšem dielčie časti riešenia, ktoré som implementoval v jazyku Python. Aj keď je moje riešenie na predchádzajúcej implementácií ETL nástroja kompletne nezávislé, rozhodol som sa pre zachovanie tohoto programovacieho jazyka. Na ukladanie informácií a metadát som sa rozhodol použiť databázu MySQL – hlavne pre jej jednoduchosť. Keďže moja implementácia bude publikovaná ako verejná, každý si ju môže prispôsobiť podľa svojich osobných preferencií.

4.1 Metadáta ETL procesov

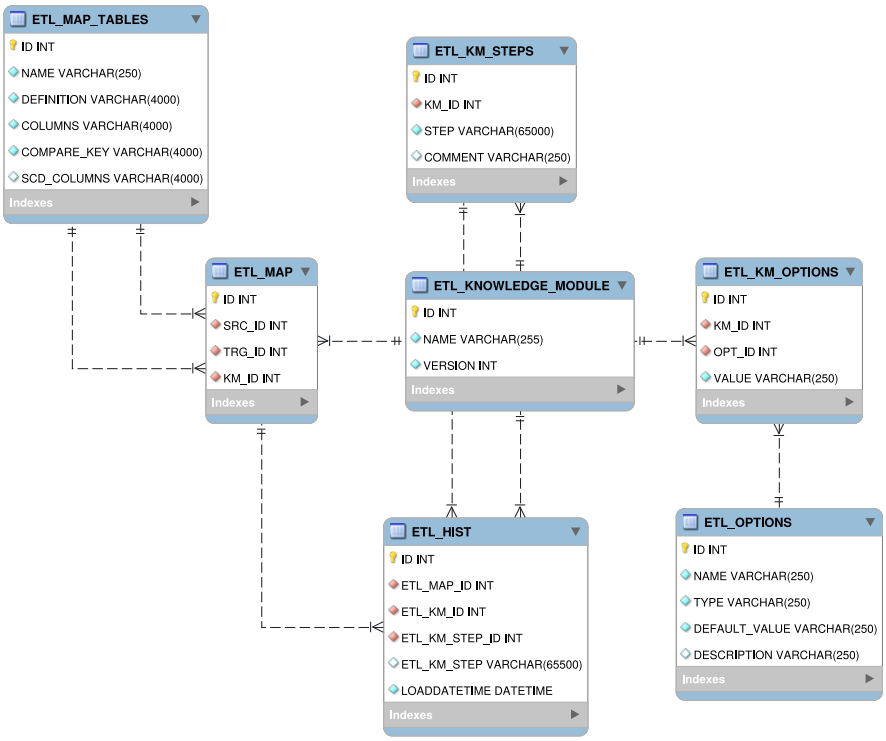
Metadáta v kontexte ETL nástroja slúžia pre uchovávanie informácií o zdrojových a cieľových tabuľkách, medzi ktorými prebiehajú spomínané procesy, spôsob akým sa tieto dáta budú prenášať a ďalšie nastavenia tohoto prenosu.

4.1.1 `ETL_MAP`

Tieto metadáta spája riadiaca tabuľka `ETL_MAP`, ktorá obsahuje cudzie kľúče do ostatných štruktúr. Riadky tejto tabuľky sú vstupom pre interpretér dynamických príkazov `SELECT` – ETLPreter 4.3. Štruktúra tabuľky `ETL_MAP` je nasledujúca:

- `ID` - automaticky generovaný identifikátor záznamu
- `SRC` - cudzí kľúč do tabuľky `ETL_MAP_TABLES` 4.1.2 označujúci zdrojovú tabuľku
- `TRG` - cudzí kľúč do tabuľky `ETL_MAP_TABLES` 4.1.2 označujúci cieľovú tabuľku

- `KM` - cudzí kľúč do tabuľky `ETL_KNOWLEDGE_MODULE` 4.1.3 označujúci zvolený spôsob prenášania dát



Obr. 4.1: Relačné schéma udržiavaných metadát.

4.1.2 `ETL_MAP_TABLES`

Informácie o zdrojových alebo cieľových tabuľkách sú udržiavané v tabuľke `ETL_MAP_TABLES`. Pre tabuľky zvolené ako zdrojové požadujem, aby poskytovali dávku prenášaných dát, tj. DML príkaz `SELECT` nad tabuľkami obsahujúcimi zdrojové dáta. Pri tabuľkách, ktoré považujem za cieľové, je nutné udržiavať názov databázového objektu, do ktorého majú byť zdrojové dáta vložené. Tieto informácie sú uchované v atribúte `DEFINITION`. V oboch prípadoch je nutné udržiavať informácie o používaných zdrojových/cieľových atribútoch v jednotnom poradí a oddelené čiarkami vo všetkých stĺpcoch tejto tabuľky, ktoré sú označené príponou `COLUMNS`. Celková štruktúra je nasledujúca:

- `ID` - automaticky generovaný identifikátor záznamu
- `NAME` - voliteľné meno zdrojovej/cieľovej tabuľky

- `DEFINITION` - zdroj/cieľ prenášaných dát
- `COLUMNS` - zdrojové/cieľové stĺpce
- `COMPARE_KEY` - zdrojový/cieľový primárny kľúč
- `SCD_COLUMNS` - zdrojové/cieľové stĺpce, ktoré sa porovnávajú pri prenášaní do dimenzionálnej tabuľky

4.1.3 `ETL_KNOWLEDGE_MODULE` a `ETL_KM_STEPS`

Okrem informácií, metadáta zachytávajú spôsob akým tieto dáta budú prenášané a to v podobe pseudo SQL kódu rozdeleného do krokov podľa expertného usúdenia dátového vývojára. Tento takmer SQL kód v tejto práci ďalej nazývam „Znalostný modul“ 4.2. `ETL_KNOWLEDGE_MODULE` je tabuľka, ktorá uvádza dostupné možnosti spôsobu prenosu, obsahuje tieto atribúty:

- `ID` - automaticky generovaný identifikátor záznamu
- `NAME` - názov znalostného modulu
- `VERSION` - verzia znalostného modulu

Nad kombináciou názvu a verzie znalostného modulu je vynútená unikátnosť.

Samotná definícia znalostného modulu je uložená v tabuľke `ETL_KM_STEPS`, kde sa nachádzajú jednotlivé kroky spôsobu prenášania dát. Do týchto M krokov rozdelí znalostný modul 4.2 dátový vývojár, používajúci toto rozšírenie. Kroky sú vkladané v poradí, v ktorom chceme aby boli vykonávané. Definícia tabuľky vyzerá následovne:

- `ID` - automaticky generovaný identifikátor záznamu
- `KM` - cudzí kľúč do tabuľky `ETL_KNOWLEDGE_MODULE` 4.1.3 označujúci, ktorému znalostnému modulu krok patrí
- `STEP` - textová definícia v podobe pseudo SQL (viac v sekcii 4.2)

4.1.4 `ETL_OPTIONS` a `ETL_KM_OPTIONS`

Spôsob prenosu dát môže ďalej byť ovplyvnený možnosťami, ktoré môže taktiež definovať dátový vývojár. Na tento účel slúžia tabuľky `ETL_OPTIONS` a `ETL_KM_OPTIONS`. Do štruktúry `ETL_OPTIONS` sú ukladané ponúkané možnosti, ktoré môžu byť zakomponované do ktoréhokoľvek znalostného modulu použitím príslušnej API funkcie. Ak tento znalostný modul nemá priradenú k tejto možnosti vlastnú hodnotu v `ETL_KM_OPTIONS`, použije sa prednastavená hodnota z tabuľky `ETL_OPTIONS`. Štruktúra tabuľky `ETL_OPTIONS`:

- `ID` - automaticky generovaný identifikátor záznamu

- NAME - názov možnosti, ktorý predávame do volania API funkcie v znalostnom module 4.2
- TYPE - typ možnosti
- DEFAULT_VALUE - prednastavená hodnota
- DESRIPTION - popis možnosti

ETL_KM_OPTIONS je teda väzobnou tabuľkou medzi dvomi tabuľkami, konkrétne ETL_OPTIONS a ETL_KNOWLEDGE_MODULE.

Štruktúra tabuľky ETL_KM_OPTIONS:

- ID - automaticky generovaný identifikátor záznamu
- KM - cudzí kľúč do tabuľky ETL_KNOWLEDGE_MODULE 4.1.3 označujúci, ktorému znalostnému modulu možnosť patrí
- OPT - cudzí kľúč do tabuľky ETL_OPTIONS 4.1.4 označujúci možnosť, ktorej hodnotu nastavujem
- VALUE - nastavovaná hodnota

4.1.5 ETL_HIST

Poslednou štruktúrou do ktorej ukladám dáta, je tabuľka ETL_HIST. Do tejto tabuľky dátový vývojár vkladá záznamy a to nepriamo, pomocou spúšťania samotného generovania dynamických príkazov. Samotné fungovanie tohoto nástroja popisujem v sekcii 4.3. Tento nástroj interpretuje postupne kroky znalostného modulu 4.1.3 a z technickými metadatami ich vloží práve do tejto tabuľky. Jej štruktúra je následovná:

- ID - automaticky generovaný identifikátor záznamu
- ETL_MAP_ID - cudzí kľúč do tabuľky ETL_MAP 4.1.1 označujúci záznam, ktorý bol interpretovaný
- ETL_KM_ID - cudzí kľúč do tabuľky ETL_KNOWLEDGE_MODULE 4.1.4 označujúci použitý znalostný modul
- ETL_KM_STEP_ID - cudzí kľúč do tabuľky ETL_KM_STEPS 4.1.4 označujúci, ktorý krok znalostného modulu bol interpretovaný
- ETL_KM_STEP - interpretácia kroku znalostného modulu
- LOADDATETIME - časová značka interpretácie

4.2 Znalostné moduly

Znalostné moduly tvoria hlavnú časť logiky celého rozšírenia ETL nástroja. Jasne definujú, akým spôsobom sa dáta medzi zdrojom a cieľom budú prenášať. Zhotovený znalostný modul dátový vývojár rozdelí do krokov, ktoré následne nahrá do príslušnej metadátovej tabuľky 4.1.3.

Využívaný je jazyk SQL, ktorý je doplnený o volania API funkcií. Moduly sú tvorené pre konkrétnu implementáciu jazyka SQL a kvôli používaniu vstavaných funkcií nemusia byť medzi sebou kompatibilné. V tejto časti implementujem znalostné moduly v jazyku T-SQL, ktorý využíva Microsoft SQL Server.

4.2.1 API volania

Pomocou volania API funkcií je umožnené dynamické generovanie DML príkazov. Znalostný modul slúži ako šablóna, do ktorej sa práve týmito volaniami doplnia potrebné informácie o zdroji, cieli, atribútoch, možnostiach,...

Každé API volanie začína špeciálnymi znakmi `<%` a končí `%>`. Čokoľvek medzi týmto ohraničením je považované za volanie API funkcie. Interpretéru 4.3 nezáleží, či sú volania funkcií uvedené malými alebo veľkými písmenami. V súčasnej verzii tohoto nástroja sú podporované nasledujúce volania funkcií:

- `GET.SRC.DEFINITION` - Doplní `SELECT` na zdrojové dáta, ktoré budú prenášané.
- `GET.TRG.DEFINITION` - Doplní databázový objekt (tabuľka), ktorá je cieľom prenášania dát.
- `GET.SRC.COLUMNS(alias)` - Doplní čiarkou oddelené stĺpce zdrojového DML príkazu `SELECT` uvedené v rovnakom poradí. Nepovinný parameter `alias` pridá každému atribútu uvedený alias – `alias.stlpec1,alias.stlpec2`.
- `GET.TRG.COLUMNS` - Doplní čiarkou oddelené stĺpce cieľovej tabuľky, do ktorých sa budú vkladať dáta. Nepovinný parameter `alias` pridá každému atribútu tento alias – `alias.stlpec1,alias.stlpec2`.
- `GET.SCD_COLUMNS(alias_src, alias_trg)` - Vytvorenie porovnania medzi údajmi, ktoré sú uvedené v zdrojovej/cieľovej tabuľke v atribúte `SCD_COLUMNS` 4.1.2 – `alias_src.stlpec1 <> alias_trg.stlpec1`.
- `GET.SRC.COMPARE_KEY` - Doplní primárny kľúč zdrojovej tabuľky.
- `GET.TRG.COMPARE_KEY` - Doplní primárny kľúč cieľovej tabuľky.
- `GET.TMP.SRC.NAME` - Vytvorí unikátny názov pre dočasný objekt zdrojovej tabuľky.

- `GET.TMP.TRG.NAME` - Vytvorí unikátny názov pre dočasný objekt cieľovej tabuľky.
- `GET.TMP.SRC.COLUMNS` - Doplní čiarkou oddelené stĺpce zdrojového DML príkazu `SELECT` aliasované názvom dočasného zdrojového objektu
- `GET.TMP.TRG.COLUMNS` - Doplní čiarkou oddelené stĺpce cieľovej tabuľky aliasované názvom dočasného cieľového objektu.
- `GET.ETL.OPTION(option)` - Doplní hodnotu možnosti `option`. Ak je táto možnosť definovaná v tabuľke `ETL_KM_OPTIONS`, vráti nastavenú hodnotu, inak vráti prednastavenú hodnotu definovanú v `ETL_OPTIONS` 4.1.4.

4.2.2 Data Vault moduly

Pre ukážku fungovania môjho rozšírenia ETL nástroja som vyvinul základné Data Vault znalostné moduly na prenášanie dát do jadra dátového skladu, ktoré spĺňajú metodiku opísanú v sekcii 2.2 Každá stavebná štruktúra metodiky predstavuje jeden znalostný modul. Menovite sú to tieto znalostné moduly:

- `dv_hub` - modul tabuľky typu hub
- `dv_link` - modul tabuľky typu link
- `dv_sat` - modul tabuľky typu satelit

Aby som sa v jednotlivých moduloch neopakoval – základom – a teda prvými dvomi krokmi ďalej spomenutých znalostných modulov sú:

1. DDL (Data Definition Language) príkaz `DROP TABLE` na dočasnú cieľovú tabuľku (ak existuje) 4.1
2. Vloženie zdrojových dát do dočasnej cieľovej tabuľky DML príkazom `SELECT ... INTO` 4.2

```
1 IF (OBJECT_ID('tempdb..<%get.tmp.trg.name%>') IS NOT NULL)
2 DROP TABLE <%get.tmp.trg.name%>;
```

Výpis kódu 4.1: `DROP TABLE` na dočasnú cieľovú tabuľku

Nasledujúce výpisy kódov znalostných modulov sú rôzne obalené v samostatných transakciách a ochránené proti výskytu chýb, tie však neuvádzam, pretože by som ich tak isto opakoval. Plný rozsah uvedených znalostných modulov je možné nájsť v prílohe tejto práce.

```
1  SELECT *
2  INTO <%get.tmp.trg.name%>
3  from (<%get.src.definition%>) src
```

Výpis kódu 4.2: Vloženie zdrojových dát do dočasnej cieľovej tabuľky

4.2.2.1 Modul `dv_hub`

Nasledujúci modul slúži na vkladanie dát do tabuľky typu hub. Keďže tento typ tabuľky reprezentuje businessovú entitu kardinalitou 1 : 1, je záznam na základe jeho kľúču vložený len pri jeho prvom výskyte. To je vo výpise kódu 4.3 zaručené riadkami 5–15. Pri budúcom objavení sa záznamu s rovnakým prirodzeným kľúčom zostane vložený záznam nezmenený.

```
1  INSERT INTO <%get.trg.definition%>
2  (<%get.trg.columns%>)
3  SELECT <%get.trg.columns(tmp)%>
4  FROM <%get.tmp.trg.name%> AS tmp
5  LEFT JOIN   (SELECT b.<%get.trg.compare_key%>, b.SourceSystem
6               FROM <%get.tmp.trg.name%> b
7               JOIN <%get.trg.definition%>
8               ON b.<%get.trg.compare_key%> =
9                  <%get.trg.definition%>.<%get.trg.compare_key%>
10              AND b.SourceSystem = <%get.trg.definition%>.SourceSystem
11             ) b
12 ON tmp.<%get.trg.compare_key%> = b.<%get.trg.compare_key%>
13 AND tmp.SourceSystem = b.SourceSystem
14 WHERE b.<%get.trg.compare_key%> IS NULL;
```

Výpis kódu 4.3: Logika vkladania do tabuľky typu hub

4.2.2.2 Modul `dv_sat`

Tento modul ako jediný vkladá s prihliadnutím na historizáciu atribútov. Dochádza teda k tomu, že pre jeden kľúč – v tomto prípade je to atribút s príponou `_HK` (hash key 2.2),ktorý predstavuje referenciu na entitu nachádzajúcu sa v tabuľke typu hub – existuje viacero záznamov. Len jeden však môže byť ten platný a to väčšinou býva práve posledný vložený záznam. Vo výpise kódu 4.4 je na riadkoch 5–12 výber posledného záznamu, podľa kľúča tabuľky. Ak kľúč ešte v tabuľke neexistuje, alebo sa medzi atribútmi niečo zmenilo, záznam sa vloží. Inak sa riadky 13–16 postarajú o jeho filtráciu.

```
1  INSERT INTO <%get.trg.definition%>
2  (<%get.trg.columns%>)
3  SELECT  <%get.trg.columns(tmp)%>
4  FROM <%get.tmp.trg.name%> AS tmp
5  LEFT JOIN (SELECT *
6             FROM
7             (SELECT b.*, row_number() over(
8                                 partition by <%get.trg.compare_key%>
9                                 ORDER BY b.LoadDatetime desc
10                             )rn
11             FROM <%get.trg.definition%> as b )s
12             WHERE rn = 1) AS b
13 ON tmp.<%get.trg.compare_key%> = b.<%get.trg.compare_key%>
14 AND tmp.SourceSystem = b.SourceSystem
15 WHERE b.<%get.trg.compare_key%> IS NULL
16 OR tmp.HashDiff <> b.HashDiff;
```

Výpis kódu 4.4: Logika vkladania do tabuľky typu satelit

4.2.2.3 Modul `dv_link`

Posledným ale tak isto dôležitým znalostným modulom metodiky Data Vault, ktorý uvádzam v tejto práci, je modul `dv_link`. Z názvu teda vyplýva, že slúži na plnenie tabuliek typu link. Podobne ako modul hub vkladá len jeden záznam, ale pre kombináciu dvoch prirodzených kľúčov. Tým je dosiahnutá kardinalita $N : M$, keďže jeden prirodzený kľúč môže mať M záznamov s informáciou o druhom prirodzenom kľúči a opačne. Vo výpise kódu 4.5 sa o filtráciu záznamov starajú riadky 5–15, ktoré sú zhodné s výpisom kódu 4.4 až na podmienku atribútu `HashDiff` pri tabuľkách satelitu.

4.2.3 Dimenzionálne moduly

Napriek tomu, že sa celú prácu zaoberám hlavne jadrom dátového skladu a teda metodikou Data Vault, je moje rozšírenie aplikovateľné aj na klasické dimenzionálne modelovanie. To bolo tak isto aj cieľom, a tak je tento nástroj použiteľný ako na prenos Stage-Core 3.2.1, ale aj Core-Mart 3.2.3. Ako ukážkový typ prenosu dát som si vybral plnenie pomaly sa meniacej dimenzie typu 2 ako jedno z najviac používaných v dátových skladoch. Som presvedčený, že nástroj je pripravený na rozšírenie tak, aby sa typy dimenzií dali v znalostnom module kombinovať, tak ako to v praxi tiež býva bežné.

```
1  INSERT INTO <%get.trg.definition%>
2  (<%get.trg.columns%>)
3  SELECT  <%get.trg.columns(tmp)%>
4  FROM <%get.tmp.trg.name%> AS tmp
5  LEFT JOIN (SELECT *
6             FROM
7             (SELECT b.*, row_number() over(
8                                partition by <%get.trg.compare_key%>
9                                ORDER BY b.LoadDatetime desc
10                           )rn
11 FROM <%get.trg.definition%> as b )s
12 WHERE rn = 1) AS b
13 ON tmp.<%get.trg.compare_key%> = b.<%get.trg.compare_key%>
14 AND tmp.SourceSystem = b.SourceSystem
15 WHERE b.<%get.trg.compare_key%> IS NULL;
```

Výpis kódu 4.5: Logika vkladania do tabuľky typu link

4.2.3.1 Modul `dim_scd2`

Tento typ prenosu je podobný tomu, ako sa vkladajú dáta do Data Vault konštruktu typu satelit. Avšak namiesto toho, aby sme záznamy len vkladali ako je to pri module `dv_sat` 4.2.2.2, tak sa postaráme o to, aby sme staré záznamy uzatvorili a zneplatnili. Tým dostaneme jednoznačne jediný platný záznam. Okrem požadovaných prenášaných atribútov sú teda tabuľky, ktoré sa plnia týmto modulom doplnené o:

ValidFrom - dátum a čas odkedy je záznam platný

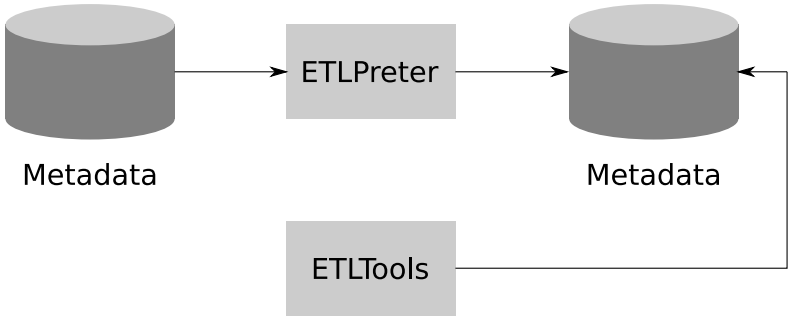
ValidTo - dátum a čas dokedy je záznam platný

CurrentFlag - Označuje posledný platný záznam s daným kľúčom ako True inak False

Nové záznamy, ktoré ešte podľa kľúča neexistujú v cieľovej tabuľke sa vložia pomocou riadkov 24–37 4.6. Detekcia zmeny záznamu zo zdroja prebieha na riadkoch 11–18. Následne sú tieto staré záznamy v cieľovej tabuľke uzatvorené, riadky 19–22 a ako výstup – riadok 38 – sú poskytnuté nové platné záznamy pre vloženie do cieľovej tabuľky. Pri platných záznamoch je uvádzaný ako koncový dátum platnosti záznamu 01.01.3000. Čo predstavuje dostatočne dlhú dobu, aby sa záznam predtým uzavrel.

```
1  INSERT INTO <%get.trg.definition%>
2  SELECT
3  <%get.src.columns%>
4  ,getdate() as ValidFrom
5  ,'30000101' as ValidTo
6  ,1 as CurrentFlag
7  FROM
8  (
9      MERGE <%get.trg.definition%> AS Target
10     USING <%get.tmp.trg.name%> AS Source
11     ON Source.<%get.src.compare_key%> =
12         Target.<%get.trg.compare_key%>
13     AND Target.CurrentFlag = 1
14 WHEN MATCHED AND
15 (
16 <%get.scd_columns(Source,Target)%>
17 )
18 THEN UPDATE
19 SET
20 Target.CurrentFlag = 0,
21 Target.[ValidTo] = getdate()-1
22 WHEN NOT MATCHED
23 THEN INSERT
24 (
25     <%get.trg.columns%>
26     ,ValidFrom
27     ,ValidTo
28     ,CurrentFlag
29 )
30 VALUES
31 (
32     <%get.src.columns(Source)%>
33     ,getdate()
34     ,'30000101'
35     ,1
36 )
37 OUTPUT $action AS Action, Source.*
38 ) as MergeOutput
39 WHERE MergeOutput.Action = 'UPDATE';
```

Výpis kódu 4.6: Logika vkladania do dimenzie typu scd2



Obr. 4.2: Diagram priebehu dynamického interpretovania.

4.3 ETLPreter - Interpretér dynamických selectov

Nástroj je naprogramovaný v jazyku Python, konkrétne v jeho verzii Python3. Na riešenie využívam minimálne množstvo knižníc tretích strán. V rámci riešenia vznikla aj dokumentácia, ktorá je súčasťou prílohy, podobne ako všetky zdrojové kódy.

Práve nástroj ETLPreter 4.2 je miesto, kde sa znalostné moduly spájajú s metadátami uloženými v relačných tabuľkách. Jeho úlohou je interpretovať volania API funkcií v znalostných moduloch a nahradiť ich skutočnými údajmi z metadátových tabuliek.

Vstupom pre interpretáciu sú záznamy v riadiacej tabuľke `ETL_MAP` 4.1.1. Pri každom spustení nástroja ETLPreter sú interpretované postupne všetky záznamy v tejto tabuľke. Táto tabuľka uvádza, čo je zdroj, cieľ a akým spôsobom sa medzi nimi dáta budú prenášať. Do krokov znalostného modulu sa teda dopĺňajú informácie uvedené v ostatných relačných tabuľkách. Po úspešnej interpretácií všetkých krokov jedného záznamu riadiacej tabuľky je ich interpretácia zapísaná do `ETL_HIST` 4.1.5 spolu s kľúčmi tabuliek, ktoré tento záznam vygenerovali a časovou značkou generovania.

Z tabuľky 4.1.5 si teda akákoľvek nezávislá aplikácia – po pripojení sa na databázový zdroj – vie podľa časovej značky vybrať najnovšie vložené záznamy k danému ETL prenosu. Ak tieto kroky následne zoradí podľa poradia v akom boli zložené, a teda podľa umelo generovaného ID záznamu má k dispozícií kompletnú, spustiteľnú sadu DML, DDL prípadne TCL (Transaction Control Language) príkazov. Spúšťanie a riadenie týchto procesov však nie je obsahom tejto práce.

Kapitola 5

Testovanie

V tejto kapitole predvediem akým spôsobom som otestoval správnosť mojej implementácie. Testovanie prebiehalo v dvoch rovinách. Ako prvé uvádzam testovanie samotného Python kódu a to pomocou knižnice `unittest`. V druhej podkapitole sa zameriam na správnu funkčnosť znalostných modulov. Moduly sú testované formou akceptačných kritérií.

5.1 Unit testy

Kľúčovými funkcionalitami kódu sú moduly `km_api` a `parser`. Tieto moduly obsahujú výhradne jednu triedu s rovnakým pomenovaním. Pri testovaní sa teda zameriavam práve na tieto triedy a ich metódy.

Testovacie scenáre obsahujú prípravu testovacích dát. Nad týmito dátami sa následne vykonajú príslušné metódy a ich návratová hodnota je porovnávaná s očakávanou návratovou hodnotou. Ak sa tieto dve hodnoty rovnajú, metóda prechádza testom.

Trieda `KM_API` spracovávaná volania API funkcií zo znalostných modulov. Okrem funkcie `get_tmp_name`, ktorá vracia hash, sú testované všetky metódy obsiahnuté v tejto triede. Spustením testovacieho prostredia nad triedou s definovaným scenárom testovania, ktorý je súčasťou prílohy dostávame nasledujúce výsledky 5.1.

Trieda `Parser` sa stará – ako už z jej názvu vyplýva – o parsovanie, čiže syntaktickú analýzu. Jej hlavnou úlohou je teda nájsť volania API funkcií a oddeliť ich od ostatného „textu“. Testovací scenár testuje správnosť tohoto rozdelenia textu voči rôznym kombináciám pozícií API funkcie a zvyšku textu. Podobne ako v predchádzajúcom prípade, kompletný testovací scenár je súčasťou prílohy a výsledky uvádzam vo výpise 5.2.

```
$ python -m unittest -v test/test_api.py

test_columns_with_alias (test.test_api.TestAPI) ... ok
test_columns_without_alias (test.test_api.TestAPI) ... ok
test_compare_keys (test.test_api.TestAPI) ... ok
test_definition (test.test_api.TestAPI) ... ok
test_options (test.test_api.TestAPI) ... ok
test_scd_columns (test.test_api.TestAPI) ... ok

----------------------------------------------------------------------
Ran 6 tests in 0.145s

OK
```

Výpis kódu 5.1: Výsledok testu triedy `km_api`

```
$ python -m unittest -v test/test_parser.py

test_boundaries (test.test_parser.TestParser) ... ok

----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```

Výpis kódu 5.2: Výsledok testu triedy `km_api`

5.2 Testovanie znalostných modulov

Znalostné moduly som sa rozhodol otestovať formou akceptačných kritérií. Pre každý modul som stanovil určité druhy požiadavkov, ktoré musia spĺňať. Pri určovaní týchto kritérií som sa riadil definíciou cieľovej tabuľky 2.2 a požiadavkami na modul 4.2.

V prílohe práce sú uvedené kompletné testovacie scenáre, skripty na vytvorenie testovaných objektov, vloženie testovacích dát a výstup z interpretácie pomocou nástroja ETLPreter 4.3. Testovanie prebiehalo na databáze SQL Server 2017 Express za pomoci SQL Server Management Studio vo verzii 18.

Pri testovaní pozorujem nasledovné počty záznamov:

- Src - počet záznamov v zdrojovej tabuľke
- Init - počet záznamov v cieľovej tabuľke po prvotnom spustení
- Reload - počet záznamov v cieľovej tabuľke po opakovanom spustení

- Inserted - počet záznamov v cieľovej tabuľke po pridaní testovacieho záznamu v zdrojovej tabuľke
- Updated - počet záznamov v cieľovej tabuľke po úprave atribútu v zázname v zdrojovej tabuľke (pri moduloch, ktorých úlohou je historizácia)

5.2.1 Test modulu `dv_hub`

Pre testovanie modulu `dv_hub` boli stanovené nasledujúce akceptačné kritéria:

1. Znovuspustiteľnosť - znovuspustením skriptu vygenerovaného pomocou nástroja ETLPreter sa bez pridania záznamu v zdrojovej tabuľke nič nevykoná, tj. zachovanie kardinality 1 : 1
2. Počet vložených záznamov - počet vložených záznamov je zhodný s počtom záznamov, ktoré sú obsiahnuté na zdroji a nie sú obsiahnuté v cieľovej tabuľke

Testovací scenár začína zmazaním dát v cieľovej tabuľke. Pred spustením ETL skriptu skontroluje počet záznamov v zdrojovej tabuľke. Následne je spustený skript a po jeho vykonaní je uvedený počet záznamov v cieľovej tabuľke. Bez zásahu do zdrojovej tabuľky sa opakovaným spustením skriptu skontrolujú akceptačné kritéria 1 a 2. Pridaním testovacieho záznamu do zdrojovej tabuľky a znovuspustením skriptu sa tieto akceptačné kritéria potvrdia. Tabuľka 5.1 uvádza počty záznamov v jednotlivých krokoch scenára.

Tabuľka 5.1: Počty záznamov v krokoch testovacieho scenára modulu `dv_hub`

Src	Init	Reload	Inserted
100	100	100	101

5.2.2 Testovanie modulu `dv_sat`

Pre testovanie modulu `dv_sat` boli stanovené nasledujúce akceptačné kritéria:

1. Historizácia - ak boli zmenené niektoré údaje v zdrojovej tabuľke, tak v cieľovej tabuľke vznikne záznam s novými hodnotami
2. Počet vložených záznamov - počet vložených záznamov je zhodný s počtom záznamov, ktoré sú obsiahnuté na zdroji a nie sú obsiahnuté v cieľovej tabuľke plus počet zmenených záznamov v zdrojovej tabuľke

Testovací scenár začína zmazaním dát v cieľovej tabuľke. Pred spustením ETL skriptu skontroluje počet záznamov v zdrojovej tabuľke. Následne je iniciálne spustený skript a po jeho vykonaní je uvedený počet záznamov v cieľovej

tabuľke. Bez zásahu do zdrojovej tabuľky sa opakovaným spustením skontroluje časť akceptačného kritéria 2. Pridaním testovacieho záznamu do zdrojovej tabuľky a znovuspustením skriptu sa potvrdí ďalšia časť 2. akceptačného kritéria. Zmenou hodnoty atribútu v zdrojovej tabuľke a ďalším spustením skriptu sa potvrdia obe akceptačné kritéria. Tabuľka 5.2 uvádza počty záznamov v jednotlivých krokoch scenára.

Tabuľka 5.2: Počty záznamov v krokoch testovacieho scenára modulu `dv_sat`

Src	Init	Reload	Inserted	Updated
100	100	100	101	102

5.2.3 Testovanie modulu `dv_link`

Pre testovanie modulu `dv_link` boli stanovené nasledujúce akceptačné kritéria:

1. Znovuspustiteľnosť - znovuspustením skriptu vygenerovaného pomocou nástroja ETLPreter sa bez pridania záznamu v zdrojových tabuľkách nič nevykoná

2. Počet vložených záznamov - počet vložených záznamov je zhodný s počtom záznamov, ktoré sú obsiahnuté v zdrojových tabuľkách a nie sú obsiahnuté v cieľovej tabuľke

Ide o rovnaké akceptačné kritéria ako pri testovaní modulu `dv_hub`. Avšak tabuľka typu link zachytáva väzbu medzi dvomi entitami tabuľky typu hub, tým pádom je jej kardinalita $N : M$ a testujem ju samostatne.

Testovací scenár začína zmazaním dát v cieľovej tabuľke. Pred spustením ETL skriptu sa skontroluje počet záznamov v zdrojových tabuľkách. Následne je prvotne spustený skript a po jeho vykonaní je uvedený počet záznamov v cieľovej tabuľke. Bez zásahu do zdrojových tabuliek sú opakovaným spustením skontrolované akceptačné kritéria 1 a 2. Pridaním testovacích záznamov do zdrojových tabuliek a znovuspustením skriptu sa tieto akceptačné kritéria potvrdia. Tabuľka 5.3 uvádza počty záznamov v jednotlivých krokoch scenára.

Tabuľka 5.3: Počty záznamov v krokoch testovacieho scenára modulu `dv_link`

Src	Init	Reload	Inserted
100	100	100	101

5.2.4 Testovanie modulu `dim_scd2`

Pre testovanie modulu `dim_scd2` boli stanovené nasledujúce akceptačné kritéria:

1. Historizácia - ak boli zmenené niektoré údaje v zdrojovej tabuľke, tak v cieľovej tabuľke vznikne záznam s novými hodnotami a starému záznamu je uzatvorená jeho platnosť

2. Počet vložených záznamov - počet vložených záznamov je zhodný s počtom záznamov, ktoré sú obsiahnuté na zdroji a nie sú obsiahnuté v cieľovej tabuľke plus počet zmenených záznamov v zdrojovej tabuľke

Testovací scenár začína zmazaním dát v cieľovej tabuľke. Pred spustením ETL skriptu sa skontroluje počet záznamov v zdrojových tabuľkách. Keďže ide o modul, ktorý prenáša dáta medzi vrstvami *core* a *mart* 3.2 vyberám posledný vložený záznam vo vrstve *core* ako platný. Následne je spustený skript a po jeho vykonaní je uvedený počet záznamov v cieľovej tabuľke. Bez zásahu do zdrojovej tabuľky sa opakovaným spustením skontroluje časť akceptačného kritéria 2. Úpravou testovacieho záznamu pridaného v predchádzajúcich krokoch testovania a znovuspustením skriptu sú potvrdené obe akceptačné kritéria. Bol vložený nový platný záznam a platnosť starého sa ukončila. Tabuľka 5.4 uvádza počty záznamov v jednotlivých krokoch scenára.

Tabuľka 5.4: Počty záznamov v krokoch testovacieho scenára modulu `dim_scd2`

Src	Init	Reload	Updated
101	101	101	102

5.3 Výsledky testovania

Implementácia kódu bola podrobená unit testovaniu. Boli testované dva hlavné moduly, na ktorých je postavená celá funkcionalita. V moduloch boli otestované metódy, ktoré sú využívané v implementácii riešenia. Z výpisov 5.1 a 5.2 vyplýva, že riešenie prešlo všetkými stanovenými testmi.

Znalostné moduly boli podrobené testovaním pomocou akceptačných kritérií. Testy ako aj testovacie dáta sú súčasťou prílohy a umožnené k nahliadnutiu. Znalostné moduly vyhoveli vo všetkých stanovených akceptačných kritériách.

Keďže riešenie vyhovelo obom stanoveným druhom testovania považujem moju implementáciu za správnu.

Záver

Teoretickým cieľom tejto práce bolo zoznámiť sa s metodikou Data Vault v spojení s vývojom dátových skladov. Preto, som túto prácu začal teoretickou časťou v ktorej som ako prvé rozoberal dátové sklady ako také. V rámci modelovania dátových skladov som vysvetlil základné princípy metodiky Data Vault. Spomenul som jej výhody a využiteľnosť v spojení s ETL prenosmi dát.

Súčasťou praktickej časti práce mala byť implementácia rozšírenia interného podnikového nástroja pre ETL procesy a ETL moduly slúžiace na prenos dát. Na základe analýzy predchádzajúceho riešenie nástroja pre ETL procesy, som prišiel s novým, nezávislým riešením. Toto riešenie využíva metadáta uložené v relačnej databáze a spolu so znalostnými modulmi, ktoré definujú spôsob prenosu dát, produkujú dynamické generovanie spustiteľného SQL kódu na prenos dát – ETLPreter. Toto riešenie je riadne otestované z dvoch hľadísk. Prvým testovaním prešiel samotný spustiteľný Python kód, ktorý bol testovaný knižnicou `unittest`. Znalostným modulom bola otestovaná ich funkcionalita správne vkladať a upravovať záznamy podľa požadovanej ETL funkcionality formou akceptačných kritérií.

Zdrojové kódy, ktoré sú obsiahnuté v prílohe tejto práce, sú pripravené na rozšírenie. Publikovaná aplikácia môže slúžiť komukoľvek ako základ pre vlastný nástroj na riadenie a správu ETL procesov.

Literatúra

[1] Kimball, R.; Ross, M.: *The data warehouse toolkit: The definitive guide to dimensional modeling*. John Wiley & Sons, 2013, ISBN 978-1-118-53077-1.

[2] Inmon, W. H.; Linstedt, D.: *Data architecture: a primer for the data scientist: big data, data warehouse and data vault*. Morgan Kaufmann, 2014, ISBN 978-0-12-802044-9.

[3] Kimball, R.; Caserta, J.: *The data warehouse ETL toolkit: practical techniques for extracting, cleaning, conforming, and delivering data*. John Wiley & Sons, 2011, ISBN 0-764-57923-1.

[4] Hultgren, H.: *Modeling the agile data warehouse with data vault*. New Hamilton, 2012, ISBN 978-0615723082.

[5] Linstedt, D.; Graziano, K.; Hultgren, H.: *The New Business Supermodel: The Business of Data Vault Data Modeling: DW2. 0 TM Compliant, 2nd Generation Data Warehouse Architecture*. Lulu.com, 2009, ISBN 978-1-4357-1914-9.

[6] Linstedt, D.; Olschimke, M.: *Building a scalable data warehouse with data vault 2.0*. Morgan Kaufmann, 2015, ISBN 978-0-12-802510-9.

Dodatok **A**

Zoznam použitých skratiek

SQL Structured query language

DWH Data warehouse

ETL Extract transform load

API Application programming interface

SCD Slowly changing dimension

DML Data manipulation language

DDL Data definition language

BI Business intelligence

SK Surrogate key

HK Hash key

XML eXtensible markup language

Dodatok **B**

Obsah priloženého CD

- `readme.txt` ..... odkaz na fakultný gitlab repozitár
- `src`
  - `etlpreter` ..... zdrojové kódy implementácie
    - `test` ..... testovacie scenáre a výsledky
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce vo formáte PDF