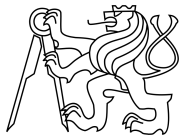
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ



# ZADÁNÍ DIPLOMOVÉ PRÁCE

| | |
|---|---|
| **Název:** | Datový sklad ČVUT - způsoby datové integrace |
| **Student:** | Bc. Robert Kotlář |
| **Vedoucí:** | Ing. Stanislav Kuznetsov |
| **Studijní program:** | Informatika |
| **Studijní obor:** | Webové a softwarové inženýrství |
| **Katedra:** | Katedra softwarového inženýrství |
| **Platnost zadání:** | Do konce letního semestru 2017/18 |

## Pokyny pro vypracování

1) Seznamte se s problematikou datového skladu a proveďte rešerši používaných architektur a různorodých způsobů získávání dat ze zdrojových systémů.
2) Navrhněte tzv. staging area datového skladu po stránce architektury i procesů.
3) Navrhněte a implementujte čištění dat ze zdrojových systémů tak, aby bylo možné data integrovat do navržené architektury datového skladu (tzv. integrated data layer).
4) Následně navrhněte a implementujte ETL procesy pro nahrání dat do navržené architektury datového skladu.
5) Pomocí výše implementovaných postupů realizujte automatické aktualizace dat v datovém skladu.
6) V práci popište použité způsoby monitoringu ETL procesů a navrhněte základní „reakční scénáře".
7) Proveďte testování prostřednictvím nahrání dat ze zdrojových systémů do datového skladu tak, že budou data dostupná v tzv. integrated data layer pro další zpracování. Testování prokažte výpisem všech potřebných statistik.

## Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 22. prosince 2016

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

# Datový sklad ČVUT - způsoby datové integrace

***Bc. Robert Kotlář***

Vedoucí práce: Ing. Stanislav Kuznetsov

9. května 2017

# Poděkování

Předně bych chtěl poděkovat vedoucímu mé práce **Ing. Stanislavu Kuznetsovovi** za cenné rady a jeho přístup k vedení této práce.

Dále bych chtěl poděkovat **Ing. Magdě Friedjungové** a **Bc. Jakubovi Krejčímu**, díky kterým bylo možné projekt datového skladu ČVUT nejenom začít, ale i rozšířit na současnou úroveň. Ze stejného důvodu děkuji i **Ing. Michalu Valentovi Ph.D** za jeho přístup a vedení projektu.

Velké poděkování patří i mé přítelkyni **Kateřině Huboňové** a mé rodině, jejichž opora pro mne byla neméně hodnotná.

# Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 9. května 2017 .....

### Odkaz na tuto práci

Kotlář, Robert. *Datový sklad ČVUT - způsoby datové integrace* . Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

# Abstrakt

Tato diplomová práce se zabývá návrhem a implementací integrace dat do datového skladu ČVUT, který vznikl rámci rozvojových projektů (DÚ č. 20 a č. 46). V první části práce je popsána teorie z oblasti datových skladů, architektur datových skladů a integrace dat. V implementační části je popsán návrh a implementace Stage vrstvy datového skladu ČVUT. Dále se práce zabývá integrací dat do databáze datového skladu a automatizací tohoto procesu.

**Klíčová slova** Datový sklad, CDC, SCD, Pentaho BI, Stage vrstva, Integrovaná datová vrstva, SQL

# Abstract

This master's thesis describes the design and implementation of data integration for the new data warehouse of CTU, which is implemented within development projects (DÚ no. 20 and no. 46). The first part of the thesis describes a theory of data warehouses, their architecture, and theory of data integration. In the implementation part of this thesis, author introduces the design and implementation of a Staging layer for the data warehouse of CTU. The last part defines the process of data integration into the data warehouse of CTU and its automatization.

**Keywords** Data warehouse, CDC, SCD, Pentaho BI, Stage layer, Integrated data layer, SQL

# Obsah

# Seznam obrázků

# Seznam tabulek

# Úvod

Tato diplomová práce obsahuje seznámení s tématem datových skladů a popis, jakým způsobem jsou integrována data do datového skladu ČVUT. V teoretické části jsou popsány všechny pojmy důležité k uchopení tématu a pochopení problematiky. Následně jsou vysvětleny jednotlivé základní prvky architektury datových skladů. Dále se teoretická část zabývá tématy blízkými datové integraci, jako je Master Data Management a Data Quality Management.

Praktická část této práce popisuje nejprve Stage vrstvu[1] datového skladu ČVUT, a to z pohledu architektury i procesů. Architektura této vrstvy je rozdělena na dvě části, které umožňují snadné rozpoznávání změn dat ve zdrojových databázích.

Dále je v praktické části popsáno, jakým způsobem jsou tyto změny rozpoznávány a zanášeny do databáze integrovaných dat datového skladu ČVUT.

Práce se také zabývá automatizací nahrávání datového skladu a umožněním sledování průběhu tohoto nahrávání. Této funkcionality jsem docílil pomocí bash skriptu, který je spouštěn pomocí démonu Cron. Tento skript spustí ETL procesy a následně odešle e-mail s logem transformace a průběžně zapisuje do systémového logu. Pokud některá z transformací selže nebo není nalezen její předpis, je odeslán chybový e-mail. Následně, pokud v logu transformace nalezne skript nějakou ze známých chyb, odešle v e-mailu i návod, jak tuto chybu opravit a co znamená.

## Cíl práce

Cílem práce je navržení a implementace procesu nahrání dat do datového skladu ČVUT tak, aby tento proces byl dostatečně rychlý, intuitivní na obsluhu a automatizovaný.

[1] Prostor, ve kterém jsou uložena data zdrojových systémů, než jsou nahrána do datového skladu.

Dalším cílem je navrhnout způsob monitoringu těchto procesů a navrhnout reakční scénáře pro případ chyby při nahrávání dat.

# Část I

# Teoretická část

Kapitola 1

# Zavedení pojmů

Tato práce se soustředí převážně na postupy a procesy datového skladu z pohledu implementace. V této kapitole bude představen celkový pohled na datový sklad a následně jsou některé pojmy rozepsány detailněji.

Datový sklad je možné si představit jako relační databázi, ve které jsou typicky integrována data z více systémů a uložena tak, aby se nad nimi dalo rychle a efektivně dotazovat. S dotazy přímo souvisí oblast Business Intelligence, neboli podpora rozhodování podniku.

Při implementaci datového skladu, je nutné tuto definici rozšířit, protože vznikne mnoho otázek ohledně integrace dat a jejich kvality, například pokud máme více systémů obsahující adresu zákazníka, tak která z informací je kvalitnější – která adresa je pravdivá. Odpovědí na tuto otázku se zabývá část 1.2.1.3. Další otázkou může být například jaká data a v jakém formátu budou správci zdrojových systémů dodávat. Také je třeba definovat, kdo a k jakým datům může přistupovat, jaká data můžeme ukládat a podobně. Tudíž se datovým skladem dá nazvat i sada procesů, technologií a podpůrných prostředků, které zajišťují automatickou integraci a vystavení dat tak, aby tato integrovaná data byla přístupná uživatelům.

Praktická část této práce je zaměřená převážně na získávání dat, jejich transformaci a nahrání do datového skladu. Tento proces se nazývá zkráceně **ETL** (Extract Transform Load), viz sekce 1.3. V jednoduchosti tento proces začíná získáním dat a to buď stažením dat z databáze zdrojového systému nebo obdržením exportů dodaných správcem systému (datově shodnými se zdrojovým systémem). Z těchto dat zrekonstruujeme obraz dat zdrojových systémů v naší databázi, která je celá v naší režii (tzv. Stage databáze). Díky tomu můžeme rozpoznávat změny, data předzpracovat nebo provádět analýzy, které by příliš vytěžovaly zdrojový systém (rychlá analýza dat jednoho zdrojového systému nad Stage databází namísto analýzy nad databází zdrojového systému). Tato data ze Stage databáze poté nahrajeme do integrovaného úložiště (databáze integrované datové vrstvy). Toto úložiště obsahuje sjednocená data všech zdrojových systémů co nejblíže třetí normální formě. Nahráním dat

končí zjednodušený ETL proces. Následně z těchto dat tvoříme datová tržiště, což jsou buďto jednotlivé tabulky nebo skupiny tabulek v normalizovaná či dimenzionální podobě, viz sekce 1.5.

Existuje několik přístupů k architektuře datového skladu, v této práci jsou uvedeny základní dva, které byly definovány již v devadesátých letech. Dále jsou zde popsány jednotlivé vrstvy datového skladu, jejich význam a využití. Na těchto vrstvách lze snáze vysvětlit rozdíl mezi jednotlivými přístupy a úskalí každého z nich, viz sekce 1.2.

Nehledě na architekturu má každý datový sklad důležitou vlastnost, a tou je možnost zobrazení historie změn záznamů. Toho je docíleno pomocí historizačních procedur a procesů. Základní myšlenky, rozčlenění a pojmenování jednotlivých typů historizace popisuji v sekci 1.4. Implementace historizačních procedur umožňuje zobrazit datový stav zdrojových systémů v kterémkoliv zvoleném čase od spuštění datového skladu a tím zjednoduší analýzy založené na sledování trendů.

Jelikož v datovém skladu je většinou velké množství záznamů, je nutné pro zpracování těchto dat využít jiné než typické databázové mechanismy. Z výkonnostních důvodů se pro interaktivní výstupy používají vystavené předpřipravené podmnožiny datového skladu – datová tržiště převážně v dimenzionální podobě, na rozdíl od klasických relačních modelů. Tento speciální způsob uložení dat využívají OLAP[2] technologie, které s daty v tomto formátu dokáží pracovat efektivněji. Dimenzionální schéma je také přirozenější a přijatelnější pro uživatele, jelikož je často již připravené v podobě datové kostky, se kterou může uživatel dále pracovat a provádět nad ní speciální operace umožňující dotazování. Více se tomuto tématu věnuje sekce 1.6.

Další částí této teoretické kapitoly jsou způsoby, jakými lze pohlížet na integraci dat. V následujících sekcích je heslovitě vysvětleno vše, co je potřebné k návrhu a implementaci mé diplomové práce.

## 1.1 Business Intelligence

Tato práce se zabývá datovým skladem, jeho architekturou a procesy jeho plnění, tudíž pojem Business Intelligence, dále jen BI, zde není příliš využívaný. Avšak spojení datových skladů a BI je tak zažité, že je třeba alespoň minimálně objasnit, jak v rámci této práce používám pojem BI.

> *„Business Intelligence (BI) je termín, označující celý komplex činností, úloh a technologií, které dnes stále častěji tvoří běžnou součást řízení podniku a jejich informačních systémů.“*

Tato věta pochází z knihy „Business Intelligence: jak využít bohatství ve vašich datech“[2] a je z ní zřejmé, že BI není konkrétní aplikace, architektura

[2]On-line Analytical Processing

databáze nebo datová sestava. Také z ní vyplývá, že BI existuje i bez datového skladu.

Avšak BI bez datového skladu má velice limitovaný rozsah analýz a datový sklad bez BI je neobhajitelně drahý. Z těchto důvodů jsou pojmy BI a datový sklad často svázané.

## 1.2 Datový sklad

Definic datového skladu je mnoho, avšak za základní jsou považovány dvě – Inmonova a Kimballova. Obě dvě ukazují, jakým způsobem lze přistupovat k architektuře a využití.

Myšlenky a postupy vyslovili již v letech 1992 a 1996, avšak ani po letech se nemění a jsou rozšiřovány. Důkazem, že jsou tyto přístupy použitelné, je i to, že datové sklady čistě dle jedné nebo druhé definice se stále používají. Dle dotazníku článek [3] bylo v roce 2012 15% používaných datových skladů postaveno podle definic Ralpha Kimballa a 35% podle definic Williama Inmona. Zbytek dotázaných používal kombinaci obou přístupů případně jinak modifikované definice.

Nejpopulárnější definici vyslovil **William Inmon** [4]. Tato definice ukazuje přístup k systému datového skladu jakožto k ucelené vrstvě nad používanými systémy.

> *„A data warehouse is a subject–oriented, integrated, nonvolatile, and time–variant collection of data in support of management's decisions.“*

**subject–oriented** Datový sklad je navržen pro zkoumání konkrétních subjektů jako jsou například zákazníci, prodeje, příjmy a podobně. Opakem by byl datový sklad zaměřený na procesy.

**integrated** Do datového skladu mají být integrována data z více systémů, aby tato data mohla být spojena a aby poskytovala celkový pohled na společnost. Pokud je datový sklad používán pouze na analýzy nad jedním systémem, nelze tento projekt nazývat datovým skladem. Integrací je tedy myšleno sjednocení záznamů z rozdílných systémů po informační i sémantické stránce.

**nonvolatile** Všechna data, které se do datového skladu nahrají, se již nadále nemohou měnit. Tvrzení o neměnnosti záznamů se však musí rozšířit tak, že data ve skladu uložená reflektují přesně stav zdrojových systémů v daném čase – pokud je nějaký záznam změněn, tak musí být nějakým způsobem zachován i původní záznam.

**time–variant** Z datového skladu můžeme získat nejen aktuální, ale i historická data. Dotazy nad datovým skladem tedy mohou přesně simulovat stav zdrojových systémů ve zvoleném čase.

Druhá definice je od **Ralpha Kimballa** [5] a jednoduše ukazuje, že architektura dle Kimballa je plně zaměřena na analýzu a BI. Tato definice se také označuje „bottom-up“, neboli odspoda nahoru, což znamená, že nejdříve se stanoví cíle – co a jak se bude analyzovat, a pro tyto cíle poté hledáme data.

> „*A data warehouse is a copy of transaction data specifically structured for query and analysis.*“

V této práci je jako definice datového skladu používaná ta dle Inmona, která je komplexnější a pro cíl této práce lépe vystihuje, čím by měl datový sklad být.

### 1.2.1 Základní vrstvy datového skladu

V této sekci jsou popsány jednotlivé základní vrstvy datového skladu a jejich vzájemné souvislosti. Definice použité v této sekci jsou převzaty z [6] a jsou doplněné o další informace. Celkový pohled na vrstvy datového skladu je na obrázku 1.4. Nejprve popisuji jednotlivé vrstvy a až poté celkové návrhy architektur, aby bylo vysvětlování jednotlivých rozdílů a problémů jednodušší a přehlednější.

#### 1.2.1.1 Source systems

Tato vrstva obsahuje všechny zdrojové systémy, které jsou následně integrovány do datového skladu. Tato vrstva se uvádí u datových skladů, i když do samotného skladu fyzicky nepatří. Jde pouze o abstraktní vrstvu, ve které se nacházejí všechny systémy, jejichž data jsou/mohou být integrována do datového skladu.

#### 1.2.1.2 Landing a Staging Area

**Landing area** je prostor, mezi zdrojovými systémy a Staging area, kam jsou nahrána data ze zdrojových systémů. Důvod, proč je důležité mít Landing area, je aby poskytovala strukturované prostředí pro přijetí a předzpracování dat ze zdrojových systémů. Má mimo jiné tyto vlastnosti:

- Umožňuje přesně rozdělit odpovědnost mezi dodavatelem dat a jejich zpracovatelem.
- Této vrstvě náleží jak relační, tak nerelační datové množiny – tím jsou myšleny jak ODS[3] databáze, tak exporty ze zdrojových systémů.
- Landing area také napomáhá k tomu, další zpracování dat je oddělené od zdrojových systémů a je plně v režii datového skladu, což je výhodné z výkonnostních důvodů.

[3] Operational Data Store

Jednodušší, ale shodná, definice se objevila na blogu společnosti Atunnity [7].

> „*There is a lot of talk in the private sector about the big data 'landing area,' though definitions offered by technology evangelists and corporate executives are often divergent or vague. In some cases, these areas are where information is aggregated, cleaned and analyzed. In other situations, these regions consist of where the assets were originally conceived. Decision-makers need to define this concept, and many are choosing to consider the data warehousing environment to be where all the magic happens.*“

**Staging Area** Staging area je převážně představována jako databáze, avšak jsou s ní spojeny procesy a postupy popsané dále. V této vrstvě probíhá kontrola a příprava dat, aby následné transformace proběhly bez potíží. Data v této vrstvě můžeme verifikovat. Následně můžeme provést změny dat, pokud máme explicitně zapsána pravidla změny a tyto změny nemění obchodní význam dat. Staging area má tyto vlastnosti:

- Všechna data ve Staging area jsou přesnou kopií současných dat ve zdrojovém systému.
- Všechny datové množiny mají stejnou strukturu jako ve zdrojovém systému.
- Všechny použité transformace jsou jednoduché a zdokumentované.
- Žádný uživatel do ní nesmí mít přístup.

Jsou však dva možné přístupy k implementaci Landing a Staging area:

**Asynchronní load** Nejdříve jsou data stažena/získána do landing area a až poté dále zpracovávána.

**Synchronní load** Přímé spojení zdrojových systémů a staging area – tudíž není použita Landing area.

Pokud uvažujeme asynchronní load, tak data získaná do landing area mohou nabývat více formátů (xml, json, csv apod) a typů (full, increment nebo delta export).

Také různé zdrojové systémy mohou poskytovat data v různých podobách. Základním cílem Staging area je zrekonstruovat tabulky do stavu, v jakém jsou ve zdrojovém systému. Ve Staging area používáme pro rekonstrukci zdrojových tabulek tři typy databázových exportů:

Tabulka 1.1: Typy databázových exportů [1]

| Type/Backup number | Full | Increment | Delta |
|---|---|---|---|
| Backup 1 | All data | – | – |
| Backup 2 | All data | Changes from backup 1 | Changes from backup 1 |
| Backup 3 | All data | Changes from backup 2 | Changes from backup 1 |
| Backup 4 | All data | Changes from backup 3 | Changes from backup 1 |

**Full** Jde o nejjednodušší způsob exportu. Jak již název naznačuje, jde o export všech dat z tabulky tak, jak jsou. Výhodou je jednoduché vytvoření a velice jednoduchá rekonstrukce těchto exportů. Stinnou stránkou jsou výkonnostní zátěž na straně zdrojového systému a velikost samotných exportů.

**Increment** Inkrementální export, je operace, jejímž výstupem je množina dat, která byla od posledního exportu změněna. Výhodou je rychlejší zpracování, jelikož se exportuje a posílá méně dat. Nevýhodou je složitější rekonstrukce dat.

**Delta** Diferenciální export, je takový typ exportu, který probíhá velice podobně jako inkrementální. Jeho výstupem je vždy množina dat, která byla změněna od zvoleného full exportu. Tudíž s opakováním těchto exportů narůstá jejich velikost oproti inkrementálnímu. Lépe vysvětlí tabulka 1.1.

Iniciální load musí být vždy full export. Dalším doporučením je tento full load nahrávat jednou týdně/měsíčně, aby se zmenšily velikosti delta exportů a aby se dříve objevily případné chyby. Tyto exporty jsou vlastně databázové zálohy. Konstrukt záloh je již hotový a zdokumentovaný a tudíž je snadné ho použít jako vstup datového skladu. Tyto exporty slouží k přesné rekonstrukci tabulek zdrojových systémů, tudíž podobnost s databázovými zálohami je významná. Podle [1] jsou 3 hlavní zálohovací strategie:

- Full denně.
- Full týdně + Delta každý den.
- Full týdně + Increment každý den.

V tabulce 1.2 je popsáno, jaké jsou nároky na datový prostor jednotlivých přístupů. Uvažujeme 20 TB velkou databázi s pětiprocentním denním přírůstkem a výpočet je za období jednoho měsíce. Tabulka ukazuje, že každý přístup má jiný poměr velikost/náročnost rekonstrukce zálohy. Je na každém správci, co je prioritou.

Tabulka 1.2: Srovnání backup strategií

| Common backup scenarios | Media Space Required for one Month | Media required for recovery |
|---|---|---|
| Full daily (weekdays) | (22 * 20 TB) = 440.00 TB | Most recent backup only |
| Full (weekly) + Differential (weekdays) | (5 * 20 TB) + (22 * 5%* 20 TB) = 124.23 TB | Most recent full + most recent differential |
| Full (weekly) + Incremental (weekdays) | (5 * 20 TB) + (22 * 5% * 20 TB) = 122.00 TB | Most recent full + all incrementals since full |

Hlavním úkolem při návrhu a implementaci Stage databáze je definování a zřízení DIA (data interface agreement), neboli pravidla způsobu dodávek extraktů / tabulek, a SLA (service-level agreement), ve které se poskytovatel dat zaváže k dodávání exportů v určitých časech a intervalech.

Některé systémy/správci však nemohou/nechtějí poskytovat exporty anebo se nechtějí zavázat k poskytování. Pak je nutné přímo spojit pomocí ETL nástroje zdrojový systém a stage vrstvu a stahovat celé tabulky.

#### 1.2.1.3 Integrated Data Layer

Význam této vrstvy je převážně v uchovávání dat v organizované podobě umožňující analýzu a výrobu jiných datových struktur s novým obchodním významem. Do této vrstvy mohou mít uživatelé přístup, avšak tato vrstva uchovává všechna data s celou její historií, tudíž se zpřístupňuje pouze podmnožina dat – převážně odfiltrované pouze aktuální záznamy. Avšak přístup do této vrstvy je vhodný jen pro zkušené uživatele, kteří pro to mají opodstatnění. Tato vrstva má 4 důležité charakteristiky:

**Centralizovaná databáze** Všechna data z různých zdrojových systémů, kde jsou uložena velmi rozdílnými způsoby (většinou vzájemně nekonzistentními) mají často rozdílný způsob definice stejných pojmů. Tato data jsou spojena do jedné logické struktury a uložena do jedné centralizované databáze. Tato struktura je jádrem celé architektury a bez ní není možné udržovat datový sklad.

**Detailní data s detailní historií** Jedním z hlavních přínosů datového skladu je jeho schopnost shromažďovat a ukládat historii dat. Tato vlastnost umožňuje datové analýzy dřívějších období, sledování trendů a mimo jiné také verifikaci datových modelů při statistické analýze.

**Škálovatelnost datových struktur** Možnost rozšířit datové struktury bez nutnosti reorganizovat ty, které již existují. I když tohoto v praxi nelze úplně dosáhnout, je třeba se při návrhu tímto pravidlem řídit a pokusit se tuto vlastnost naplnit.

**Nezávislost na užití** Z této vlastnosti vyplývá:

- Datový sklad slouží jako model vztahů mezi daty. ER modelování IDL tak nemá obchodní pravidla, ale datová [8].
- Datový model musí být co nejblíže 3NF[4] a to kvůli umožnění modelování obchodních pravidel v přístupové vrstvě, viz 1.2.1.5.
- Modelování musí být zaměřeno na jednotlivé subjekty. To znamená, že struktura datového skladu není přizpůsobována vnitřním procesům.
- Redundantní data nejsou ukládána.

**Master Data Management** Dalším pojmem, který se váže s IDL[5] je master data management (MDM). V IDL se spojují data z různých zdrojů a je třeba určit, který systém obsahuje „správná“ data.

Data z různých zdrojů mohou obsahovat duplicity nebo vzájemné inkonzistence. Tyto chyby se nesmí propagovat do IDL, aby se zaručila dostatečná kvalita informace, která je ze skladu předkládána.

Proces správy dat je nutné zaměřit i na data pouze z jednoho zdroje, jelikož mohou i tak obsahovat duplicity nebo jiné nesrovnalosti. Z těchto důvodů je nutné nastavit pravidla, podle kterých se z dat vstupujících do IDL vytvoří „jediná verze pravdy“, neboli „zlatá kopie“. Více se tomuto pojmu věnuje sekce 1.7.

#### 1.2.1.4 Access Layer

Access layer, neboli přístupová vrstva, je první vrstvou, do které mají obvykle přístup koncoví uživatelé. Přístupová vrstva obsahuje sémantickou vrstvu, která je běžně nepřístupná uživatelům a slouží k tvorbě datamartů podle požadavků konzumentů dat.

**Semantic layer** Sémantická vrstva poskytuje překlad zdrojových databázových struktur na obchodní pojmy podle obchodních pravidel a omezení. Často, v menších řešeních, je tato vrstva přímo zakomponovaná v reportovacím nebo dotazovacím nástroji [9].

Tato vrstva však neobsahuje přímo obchodní pojmy na nejnižší možné úrovni jako je příjem, hrubá marže, zákazník a podobně. Tato vrstva obsahuje pouze data, jejichž spojením lze zkonstruovat tyto základní obchodní pojmy pomocí datových struktur. Důležité je znovupoužití těchto základních pojmů tak, aby byla zaručena konzistence výstupů.

Struktura těchto dat není svázaná 3NF, a tak data mohou být i v denormalizované nebo dimenzionální podobě. Data z těchto struktur jsou následně použita pro tvorbu datových tržišť pro reportovací nástroje a analýzy.

[4] Třetí normální forma – metodika návrhu databáze vedoucí k optimálnímu využívání transakčních databází.

[5] Integrated data layer

Hlavním přínosem sémantické vrstvy je pevné definování stavebních bloků, aby byly stejné pro všechny analýzy a zároveň aby se nemusely vytvářet vždy znovu. Tímto přístupem zajistíme, že jednotlivé reporty a výstupy z datového skladu budou konzistentní.

**Data mart** Data mart, neboli datové tržiště, je datová struktura určená pro koncové uživatele, která je vytvořena dle jejich požadavků. Nejčastěji má dimenzionální podobu a je určena jen na specifický obchodní požadavek. Data mart je tvořen z databáze sémantické vrstvy, aby se předešlo opětovnému počítání a redefinici stejných pojmů.

#### 1.2.1.5 Information Delivery Layer

Tato vrstva umožňuje obchodním uživatelům přistupovat k datům. Většinou je tato vrstva rozdělena na dvě podvrstvy – aplikační a prezentační. Do aplikační podvrstvy přistupují nástroje na analytiku, které potřebují jinou, pro uživatele hůře čitelnou, strukturu. V případě prezentační podvrstvy jde o přesný opak.

### 1.2.2 Architektura dle Inmona

V této sekci používám informace z knihy Roberta Laberge [10]. V této knize poukazuje R. Laberge na to, že Inmonova architektura, stejně jako definice v sekci 1.2, je zaměřena na přípravu a konsolidaci dat a poté až na jejich využití. Na obrázku 1.1 je zakreslena takto navržená architektura, ve které jsou vidět všechny vrstvy popsané v sekci 1.2.1, jen není použita sémantická databáze.

Absence sémantické vrstvy může mít za následek obtížnou správu a údržbu rostoucího počtu datamartů. Pokud nejsou v sémantické vrstvě jasně definované obchodní pojmy, může se stát, že dva datamarty mohou mít dva rozdílné způsoby určování základních entit (například kdo je aktivní zákazník) a tudíž mohou poskytovat rozdílná data.

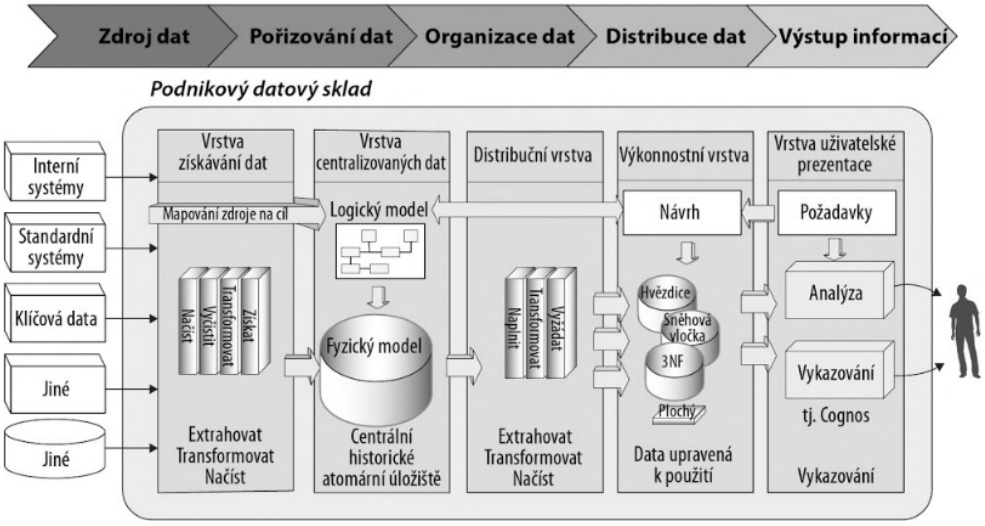
Hlavní cíl této architektury je v konsolidaci a až následném těžení dat, což je patrné z citace:

> „*Data warehouses are fundamentally different from data marts. The two do not mix—they are like oil and water.*“ [11]

### 1.2.3 Architektura dle Kimballa

Architektura dle Kimballa je již od začátku cílená na výtěžnost dat, což potvrzuje definice Ralpha Kimballa:

> „*The data warehouse is nothing more than the union of all the constituent data marts.*“ [12]




Obrázek 1.1: Architektura dle Williama Inmona

Podle této definice vypadá i architektura znázorněná na obrázku 1.3 [10]. Chybí zde nejen sémantická vrstva, ale i vrstva integrovaných dat. Datamarty se tvoří přímo ze staging area a tyto datamarty jsou následně vystaveny uživatelům.

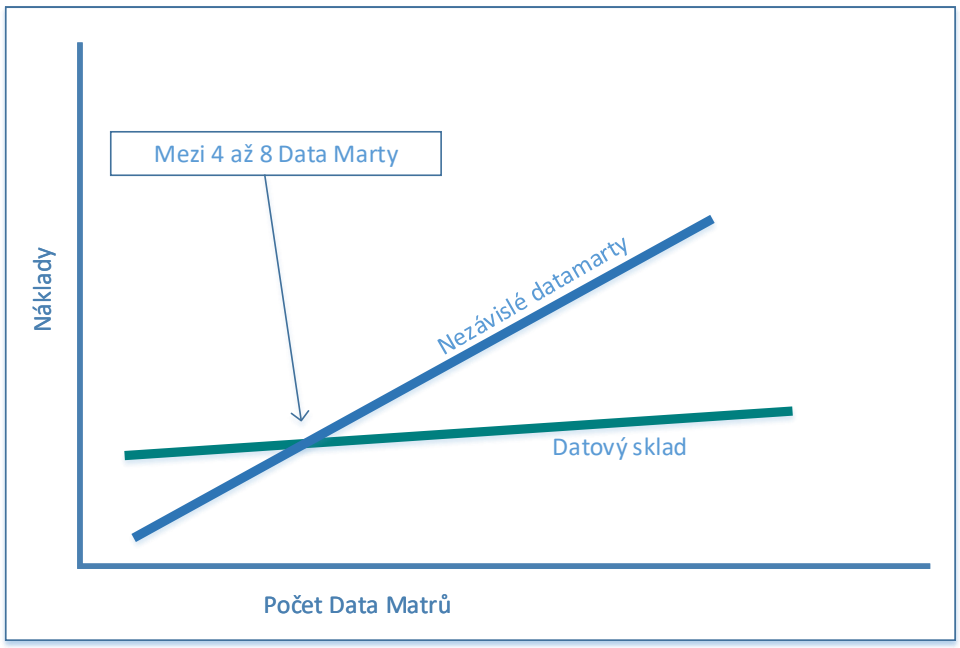
Tato architektura je vhodná pro malé projekty, které potřebují co nejrychlejší výstup – to také ukazuje obrázek 1.2. Každý dopředu definovaný data mart slouží určitému účelu a má jistou prioritu. Podle této priority přibývají data marty, které lze nezávisle na dalších okamžitě používat. Avšak tato nezávislost může být problémem s jejich přibývajícím počtem.

### 1.2.4 Současná architektura datových skladů

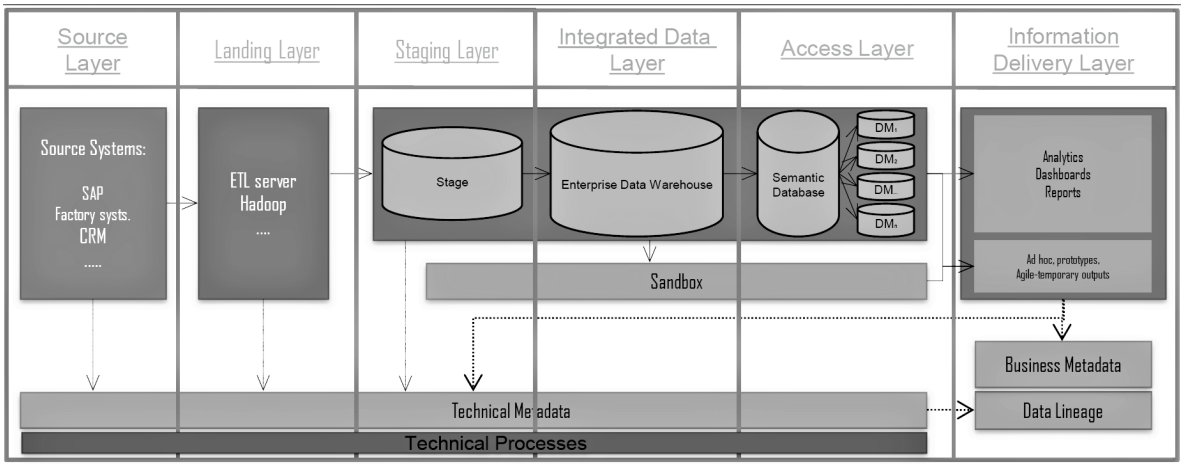
Dle přednášky Ing. Daniela Arnošta CSc. [6] je použitelný a udržitelný přístup k datovým skladům kompromisem mezi Kimballovou a Inmonovou architekturou. V současných datových skladech (viz 1.4) je použita i sémantická vrstva a architektura je velmi podobná Inmonově architektuře. K uživatelům se data mohou dostat i přímo z IDL přes sandbox[6], pokud jsou potřeba nějaké ad-hoc reporty – tato vlastnost umožňuje flexibilnější užívání datového skladu.

V souladu s potřebami a aktuálními trendy jsme vytvořili v rámci rozvojových projektů datový sklad ČVUT (DÚ č.20 a č.46).

[6]Testovací prostředí určené pro experimenty, oddělené od produkčního prostředí.




Obrázek 1.2: Růst ceny v závislosti na počtu data martů [6]

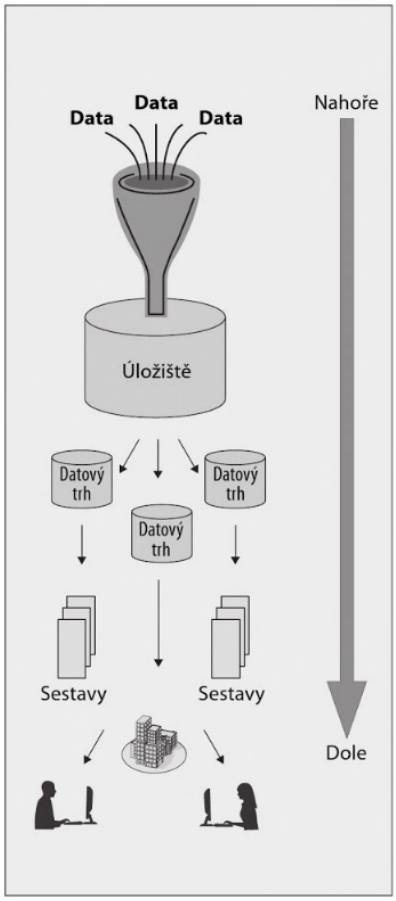



Obrázek 1.4: Základní vrstvy datových skladů

## 1.3 ETL

Zkratka ETL znamená získání dat ze zdrojových systémů – **extract**, tato data jsou nějakým způsobem transformována, čištěna, konvertována do jiné podoby – **transform** a následně nahrána do cílového úložiště – **load**.

Proces ETL je zpracováván buď nějakým nástrojem nebo pomocí skriptů.




Obrázek 1.3: Architektura dle Ralpha Kimballa

Nástrojů pro tvorbu ETL je na trhu nepřeberné množství, ať už open source (Pentaho DI[7]) nebo komerční (Informatica Integration[8]). Výhodou ETL nástrojů je snadný vývoj, do jisté míry abstrahovaný od samotné implementace. Tyto nástroje pracují převážně se záznamy (jednotlivými řádky), které jsou předávány různým modulům, které data transformují nebo s nimi jinak pracují. Tyto moduly jsou většinou konfigurovatelné a nastavitelné, avšak uživatel má jen dokumentaci a tudíž neví, jak je tento modul implementován. Neodborná tvorba nebo limity použité technologie způsobují problémy při hledání chyb. Na druhou stranu psaní vlastních skriptů je zdlouhavé a personálně náročnější.

**ELT** Dalším přístupem k nahrání dat do datového skladu je ELT (Extract – Load — Transform). Data jsou získána ze zdrojového systému, nahrána do

[7]http://http://community.pentaho.com
[8]https://www.informatica.com/products/data-integration.html

cílové databáze a až následně upravována a transformována. Dále jsou také používané různé kombinace jako například ETLT.

## 1.4 Historizace

Historizace je postup, jehož implemetace zaručuje možnost zobrazení historických dat zdrojových systémů. Historizaci se detailně věnovala má bakalářská práce [13]. V této práci stručně popíši rozdíl mezi changing data capture (CDC) a slowly changing dimension (SCD). Nejzákladnější rozdíl je, že se CDC zaměřuje především na rozpoznání změny, zatímco SCD na předpis, jak na změny reagovat.

### 1.4.1 Changing Data Capture (CDC)

> „Technologie changing data capture identifikují změny provedené nad zdrojovým systémem a ty následně zanesou do cílového systému.“

Jsou tedy zpracovávána jen ta data, která byla změněna, a tím je ušetřen výpočetní čas. CDC může probíhat buď dávkově, tzn. data se s určitou periodou stahují a porovnávají se starším otiskem dat nebo v reálném čase.

Implementace real-time CDC je netriviální úkol, protože vytěžování těchto dat musí mít minimální dopad na výkon dotčeného systému. Jako nejjednodušší, avšak nedostatečné jsou triggery v databázi, nepřetržitý streaming dat ze zdroje anebo posílání dávek dat v krátkých intervalech. Všechny tyto přístupy však velice nepříznivě ovlivňují výkon systému. Východiskem, i když složitým na implementaci, je řešení, které zpracovává transakční logy databáze, ze kterých se mimo zdrojový systém zpracovávají změnové vektory. [14]

### 1.4.2 Slowly Changing Dimension

Slowly changing dimension (zkráceně **SCD**) je proces, který definoval Ralph Kimball [15]. Tento proces zpracovává data ze zdrojových systémů podle určitých pravidel. Vyskytují se základní 3 typy (SCD0–SCD2) a další hybridní typy (SCD3–SCD6). Hlavní rozdíl oproti CDC je v tom, že SCD zpracovává všechna data při nahrávání do cílové databáze a ne jen ta změněná.

Každý z těchto typů je předpisem, jak upravit cílovou tabulku. Pokud se původní záznam změní, je založen nový záznam nebo je záznam v původní tabulce smazán. Původní význam SCD byl v historizaci záznamů, které se nahrávaly do dimenzionálního modelu datového skladu. Jelikož datové sklady nemusí být uloženy v dimenzionální podobě, ale předpis SCD je natolik známý, používá se tento název i pro historizaci záznamů, které se ukládají v normalizované podobě.

SCD v cílové tabulce vyžaduje pracovní sloupce, které určují platnost záznamu. Mezi tyto sloupce patří například čítač verze záznamu, technický klíč záznamu, datum začátku a konce technické platnosti záznamu, datum začátku a konce obchodní platnosti záznamu. Někdy se používá také termín jednoosá a dvouosá SCD, který označuje, kolik os (technická, obchodní ...) se ve skladu udržuje. Tyto osy se navzájem ovlivňují a je třeba důsledně navrhnout postupy a procesy jejich vzájemné konsolidace. Dále popíši základní typy jednoosé historizace, jak je definoval Ralph Kimball.

#### 1.4.2.1 SCD0

Tento typ SCD znamená „uzavření proti změně“. Tím je myšleno, že nový záznam se vloží, avšak jakákoliv změna záznamu je nežádoucí a nesmí být provedena.

#### 1.4.2.2 SCD1

Historizace tohoto typu je nejjednodušší na implementaci – jde o volně přepisovatelný atribut, u kterého neudržujeme žádnou historii.

#### 1.4.2.3 SCD2

Třetí z historizací je udržování celé historie záznamu, čehož je docíleno použitím pomocných sloupců. Minimum nutné pro správnou funkci jsou tři sloupce:

**Technický klíč** Náhradou za primární klíč tabulky je technický klíč. Primárnímu klíči musíme odstranit databázové omezení (jedinečnost záznamu), protože se záznam vkládá se stejným ID[9] vícekrát.

**Začátek a konec datové platnosti záznamu**

Udržování historie probíhá tak, že pro změněný záznam se vloží do cílové tabulky nový záznam a původnímu se upraví konec datové platnosti. Následující popis jsem převzal z mé bakalářské práce [13]:

**Přidání** nového záznamu znamená:

- Vytvoření nového záznamu s vlastním *Náhradním primárním klíčem*
- přiřazení aktuálního data do *LOAD_DATE*,
- přiřazení „nekonečna“ do *END_DATE*.

**Změna** v nějakém záznamu vyvolá:

- Změnu *END_DATE* původního záznamu na aktuální datum,

[9] původní primární klíč

- vytvoření nového záznamu s vlastním *Náhradním primárním klíčem* (dalším v řadě),
- přiřazení aktuálního data do *LOAD_DATE* nového záznamu,
- přiřazení „nekonečna“ do *END_DATE* nového záznamu.

**Smazání** záznamu znamená:

- nastavit *END_DATE* na aktuální datum

Pro snazší dotazování se mohou použít další pomocné sloupce, jako je:

- Verze – Číslo určující, o kolikátou verzi záznamu se jedná.
- Současný záznam – Příznak, určující, zda jde o současně platný záznam.
- Naposledy změněno – Každá změna záznamu vyvolá aktualizaci tohoto data.

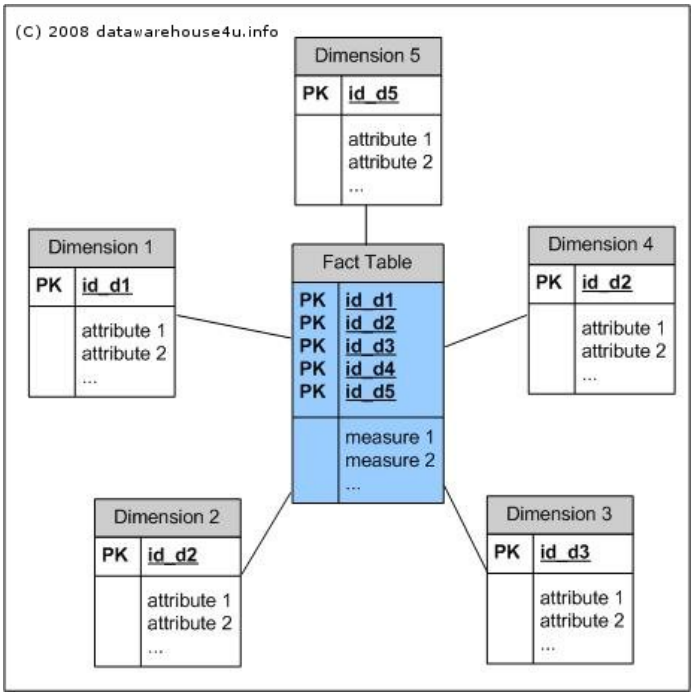
#### 1.4.2.4 Další typy SCD

Další typy, vycházející z těchto původních, přidávají další pomocné sloupce, například sloupec pro udržování minulé hodnoty (SCD3). Dalším způsobem udržování historie dle SCD je například vytvoření tabulky historie a aktivních hodnot nebo sloučení všech tří typů. Tyto typy v diplomové práci nevyužívám, proto se jimi nebudu dále zabývat.

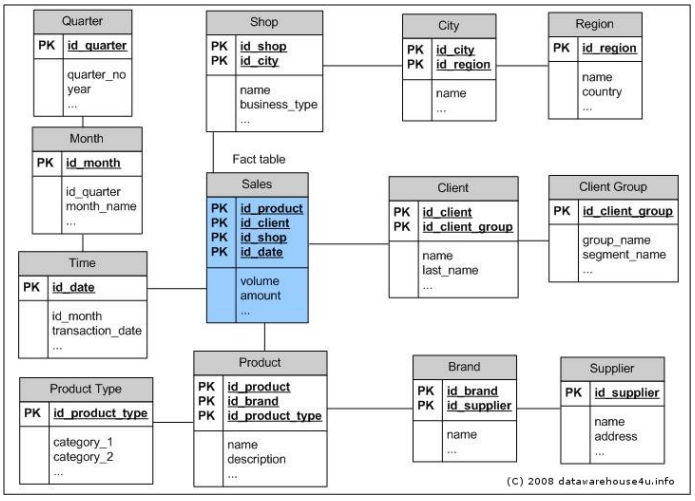
## 1.5 Dimenzionální modelování

V datovém skladu se využívá vedle modelování ve 3NF i dimenzionální modelování. Tento způsob návrhu je vhodný pro analytické a dotazovací nástroje, které díky již agregovaným datům mohou rychle a efektivně podávat výstupy. Základní typy modelů jsou star (obr. 1.5) a snowflake model (obr. 1.6), ze kterých se následně vytvářejí složitější struktury jako je fact constellation (obr. 1.7) a podobně. Dle návrhu Ralpha Kimballa je celý datový sklad v dimenzionální podobě, na rozdíl od Inmonovy architektury, kde je datový sklad ve 3NF a datová tržiště jsou tvořena (také) v dimenzionální podobě.

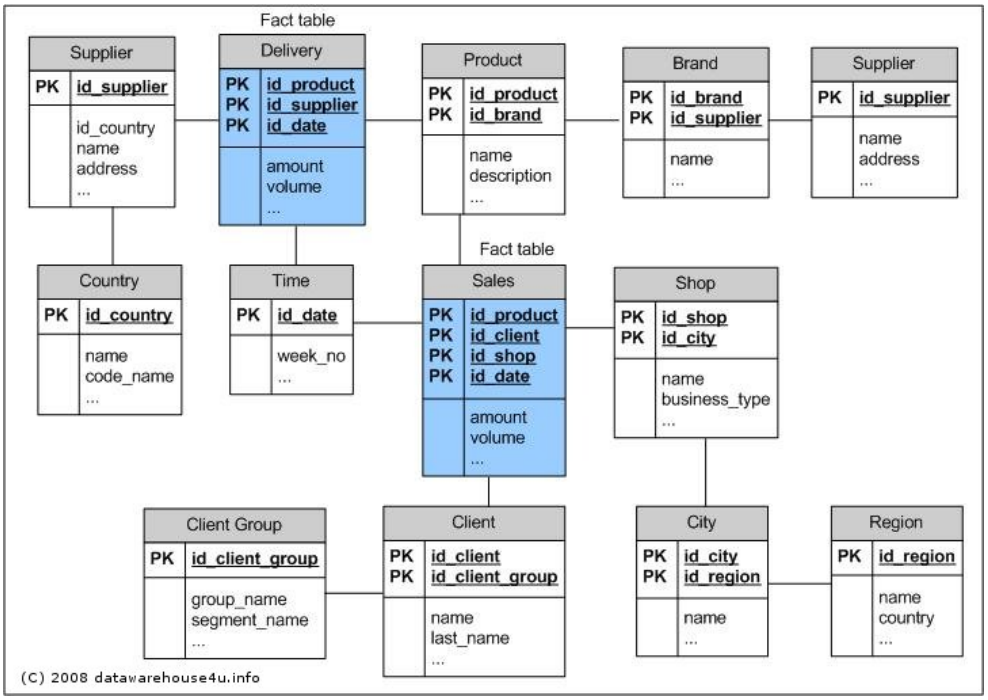
Dimenzionální modely se skládají z faktových a dimenzionálních tabulek. Faktová tabulka obsahuje obchodní metriky, v ideálním případě aditivní, a cizí klíče do dimenzí, podle kterých se napočítávají výsledné reporty. Tabulka dimenzí obsahuje data pro kategorizaci a agregaci metrik.




Obrázek 1.5: Star model[16]




Obrázek 1.6: Snowflake model[16]




Obrázek 1.7: Fact constellation model

## 1.6 Technologie OLAP

Analýza dat z datového skladu využívá převážně dimenzionální modelování, které je vstupem pro analytické nástroje. Tyto analytické nástroje se souborně nazývají OLAP [10].

OLAP systémy pomáhají analyzovat data z mnoha perspektiv a podporují komplexní analýzy velkých objemů dat. Existují 3 základní typy OLAP architektur:

**ROLAP** (Relational On-Line Analytical Processing) systémy používají dynamický server připojený k relační databázi.

**MOLAP** (Multidimensional OLAP) systémy využívají multidimenzionální databáze.

**HOLAP** (Hybrid On-Line Analytical Processing) je kombinací dvou předchozích přístupů. [17]

[10] on-line analytical processing system

## 1.7 Integrace dat

Integrace dat je ústředním úkolem datového skladu. V úvodu je ještě třeba říci, že integrace dat je jiný proces, než integrace systémů na datové úrovni. Není tedy nutné se zabývat tím, aby všechny systémy „viděly“ stejná data. Je ale nutné převzít data od zdrojových systémů a spojit je do smysluplného celku. Integrace dat je tedy proces, který data z více zdrojů konsoliduje pro použití v jednom kontextu.

Integrace dat z více systémů s sebou nese spoustu otázek – jak k datům přistupovat, jaká data uchovávat, určení validního záznamu, jak reagovat na změnu dat ve zdrojových systémech, jak data získávat a podobně. Ve zbytku této kapitoly jsou popsány jednotlivé pojmy nutné k naplnění praktické části této práce.

**Synchronní versus asynchronní integrace** Synchronní datová integrace systémů do datového skladu znamená, že se zpracovávají data, právě když vznikají. Toho lze docílit například pomocí technologie Oracle GoldenGate[11], která zpracovává transakční log databáze a tyto změny replikuje do cílového systému. Tento přístup se využívá u dat, jejichž zakomponování do datového skladu a vystavení v BI nástroji nemůže čekat do dalšího plánovaného nahrání datového skladu.

Naproti tomu se v kontextu datových skladů za asynchronní přístup označuje dávkové zpracovávání dat, ať je to jednou týdně, denně nebo každou minutu.

**Replikace dat versus ETL** Pokud uvažujeme způsob získání dat ze systému, tak za synchronní integraci lze považovat replikaci dat. ETL lze považovat za asynchronní (viz sekce 1.3). V tomto případě však řešíme pouze získaní dat do staging area. Nahrávání Stage -> IDL však převážně probíhá pomocí ETL procesů.

**ODS** Operational Data Store je systém, který slouží jako úložiště operačních dat. Převážně se ODS používá pro rychlou replikaci těchto operačních dat, která se často integrují z více zdrojových systémů. Hlavní výhodou je však možnost analytických dotazů nad těmito daty a také to, že **ODS** může sloužit jako sdílená cache (integrace systémů na datové úrovni). **ODS** se v kontextu datového skladu používá v Landing nebo Staging area jako vstup datové integrace, jelikož jsou v ní uložena aktuální data zdrojových systémů.

**MDM** Master Data Management označuje sadu procesů, které slouží k získání „správných“ dat a udržení jejich kvality. MDM tedy definuje pravdivou

[11] http://www.oracle.com/technetwork/middleware/goldengate/overview/index.html

verzi klíčových záznamů obchodních dat. Nejdříve je nutné tato data rozpoznat, definovat a zdokumentovat. Rozpoznávání pravdivých dat má 2 vrstvy, a to **obchodní** a **systémový** – obchodní pohled na správná data je převážně ze strany procesů, aby data měla správný obchodní význam. Systémový pohled je ten, který definuje požadavky na data z pohledu systémů, které tyto data využívají. Zjednodušeně existují 2 přístupy (a jejich kombinace) – centralizovaná MDM vrstva, odkud jsou data předána aplikacím, anebo jsou data z aplikací a systémů konsolidována do MDM vrstvy. Oba tyto přístupy mají repozitář rozdělený na 2 části – pracovní a produkční. Do pracovní oblasti jsou data nahrána, následně jsou na ně aplikována čistící a další pravidla. Následně se spustí schvalovací proces (ať automatický nebo se zásahem pracovníka), který uvolní data do produkční části. [18]

**DQM** Data Quality Management je proces, který zajišťuje, že data mají validní hodnoty, datové typy a kódy užité v obchodních slovnících (kód M u parametru pohlaví značí muže, kód X je chybný). DQM, pokud je správně implementován, je–li spolu s ním provedena standardizace názvů a je–li dodržována jmenná konvence, napomáhá k jednoznačné interpretaci dat a snazší práci s nimi. Samotný proces DQM vypadá následovně:

**Identifikace** Jde o proces zjištění a evidence metadat, zejména informací, ve kterých systémech se data o dané entitě vyskytují, definice tabulek, sloupců datových typů a podobné. Jsou nutné k tomu, aby bylo vůbec možné datovou kvalitu definovat a měřit. Většinou v kompetenci datové architektury.

**Profiling** Neboli poznání vlastních dat v jednotlivých systémech sběrem statistik o jednotlivých atributech. Ideálně se jeví využití nástrojů určených pro datový profiling. Většinou v kompetenci IT, je nutné řešit specifika způsobu uložení dat v daném systému.

**Měření a zavedení pravidel** Pravidla datové kvality by měla být řízena primárně obvhodními požadavky se zohledněním IT požadavků. Kromě kvalitativních metrik zahrnujících úplnost, validitu, přesnost, integritu, včasnost a konzistenci dat, je dobré implementovat kvantitativní metriky zahrnující např. spokojenost byznys uživatelů, zvýšení produktivity, byznys přínos kvalitních dat nebo reálná rizika způsobená nekvalitními daty. Kvantitativní metriky jsou klíčové pro klasifikaci závažnosti chyby z obchodního pohledu a nutnosti či ROI[12] její opravy.

**Monitoring a reporting** Má na starost kontinuální vyhodnocování výsledků kontroly DQ pravidel nad daty a jejich srovnání s definovanými limity. Obchodní uživatelé definují požadavky na monitoring a reporting,

[12]Return On Investments, často definované jako poměr vydělaných peněz a investovaných peněz.

IT poté navrhuje, implementuje a provozuje monitoring datové kvality v produkci. Reporting umožňuje agregovaný přehled chyb a detailní pohled na jednotlivé chybové záznamy nevyhovující definovaným DQ pravidlům sloužící jako poklad k opravě.

**Náprava** Jde o kontinuální proces oprav reportovaných chybných dat. Zahrnuje nejen opravu existujících dat, ale také odstranění příčiny. Opravu dat zajišťují data stewardi identifikací příčiny a místa vzniku chyb a přípravou plánu k opravě dat a zabránění jejich opakovanému vzniku.

Na trhu existuje celá řada nástrojů pro aplikační podporu implementace DQM procesu. Hotová řešení existují zejména pro profiling dat, definici a vyhodnocování DQ pravidel a reporting chybových záznamů. Samotné opravy dat je ovšem nutné typicky realizovat v samotných primárních systémech než v DQM nástroji. Z provozního hlediska je nutné brát v úvahu výkonnostní dopady DQ. Vyhodnocování složitějších (zejména konzistenčních) DQ pravidel může generovat nezanedbatelnou zátěž na sledované systémy. V případě primárních systémů toto nemusí být akceptovatelné a je nutné sledovaná data kopírovat jinam či vytvořit speciální prostředí pro DQ.

Tento proces je opakován neustále ve stejném pořadí. Zavedení DQM není jednorázový projekt, jde o **trvalý** proces. Čerpáno z [19].

# Část II

# Praktická část

Kapitola 2

# Architektura datového skladu ČVUT

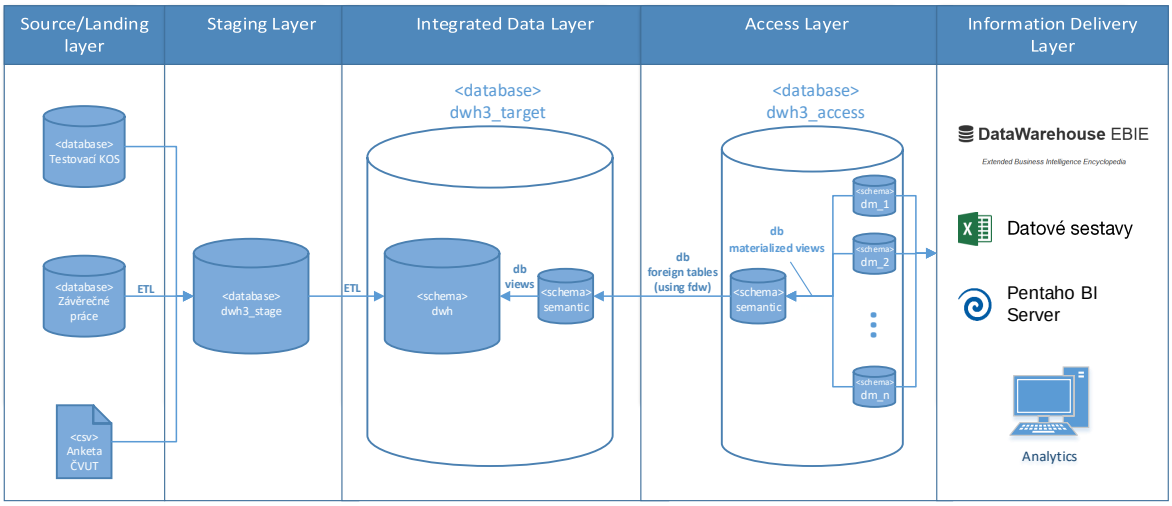
V rámci rozvojových projektů v letech 2015-2017 (**DÚ č. 20, DÚ č. 46**) na Fakultě informačních technologií vzniká datový sklad ČVUT. V následující části se budu věnovat jednotlivým krokům implementace.

Architektura datového skladu byla navržena tak, aby byla co nejbližší současným architekturám (viz sekce 1.2.4). Ve své práci jsem se věnoval především způsobům získávání dat, stage area a nahrávání dat do IDL. Zdrojovými systémy našeho řešení byly systémy KOS[13] (studijní informační systém/Komponenta Studium), Anketa ČVUT[14] a systém závěrečných prací[15] (ZP). K jednotlivým systémům a způsobům získávání dat se váže kapitola 2.1. Avšak architektura je navržená s ohledem na rozšíření. V době, kdy píši tuto práci, již analyzujeme a připravujeme integraci systémů EDUX[16], Progtest[17], Aplikace na evidenci výsledků vědy a výzkumu[18] (**V3S**) a Portál spolupráce s průmyslem[19] (**SSP**).

Jako stage area nám slouží PostgreSQL databáze, ve které jsme vytvořili schéma pro každý zdrojový systém. Dále jsme stage vrstvu rozdělili na 3 části, pomocí kterých zaznamenáváme změny zdrojových tabulek.

Vrstvu integrovaných dat navrhl ve své práci Jakub Krejčí [20]. Datová vrstva je navržena ve třetí normální formě a z této vrstvy vytváříme sémantické pohledy. Tyto pohledy pomocí „SQL/MED“[20] vystavíme přístupové

[13] https://kos.is.cvut.cz
[14] https://anketa.is.cvut.cz
[15] https://bpm.cvut.cz
[16] https://edux.fit.cvut.cz/
[17] https://progtest.fit.cvut.cz/
[18] https://v3s.cvut.cz
[19] https://ssp.fit.cvut.cz/
[20] SQL Management of External Data – specifikace sql pro správu dat v jiné databázi. V PostgreSQL implementována jako Foreign Data Wrapper.




Obrázek 2.1: ČVUT DWH 3.0

vrstvě, kde jsou reprezentovány jako foreign tables[21]. Ze sémantických pohledů následně tvoříme jednotlivá datová tržiště, která jsou následně přístupná uživatelům.

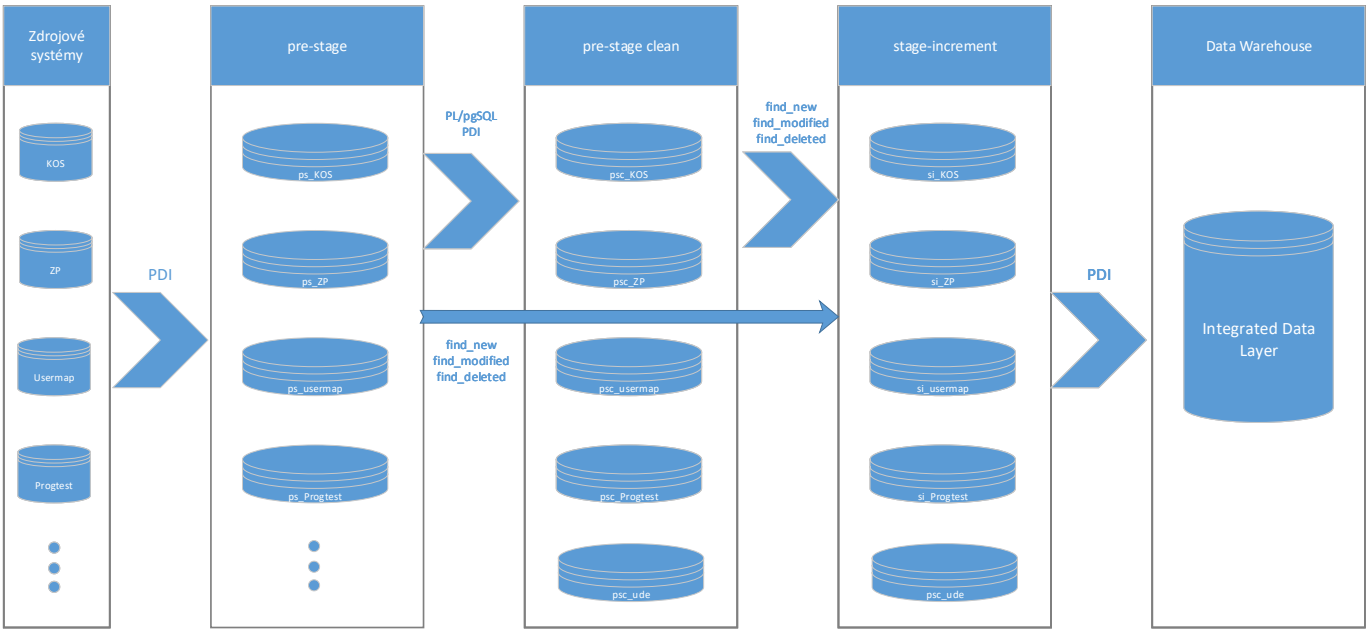
## 2.1 Staging area datového skladu ČVUT

Staging area jsme v 1. verzi datového skladu implementovali pomocí jedné PostgreSQL databáze, ve které jsme vytvořili schéma pro každý zdrojový systém.

V mé bakalářské práci [13] jsem navrhoval ETL procesy a historizaci dat plně pomocí nástroje Pentaho Data Integration (dále PDI), který se však ukázal pro naše použití nedostačující. Historizace pomocí pomalu se měnících dimenzí, jak je implementována v PDI, zpracovává záznam po záznamu všechny, které se vyskytují ve zdrojové databázi. Každý tento záznam tato komponenta dohledává v cílové databázi a podle toho, zda je záznam změněný nebo nový, dále probíhal databázový insert nebo update původního záznamu a insert nového. Tento způsob byl velice pomalý (modifikací a nových záznamů je minimum v porovnání s nezměněnými záznamy). Dále tato komponenta neumí detekovat smazané záznamy.

Jako vylepšení jsem navrhl oddělení 2 vrstev stage databáze jakožto pre–stage a stage–increment, které používáme na sledování změn ve zdrojových systémech. Dále jsme oproti původnímu řešení přidali hrubé, technické čištění

[21] Foreign tables využívají Foreign Data Wraper a tím zaručují komunikaci s externí databází bez toho, aby byl uživatel zatěžován implementací, jak se data získávají a kde jsou uložena.




Obrázek 2.2: Architektura Stageing Area

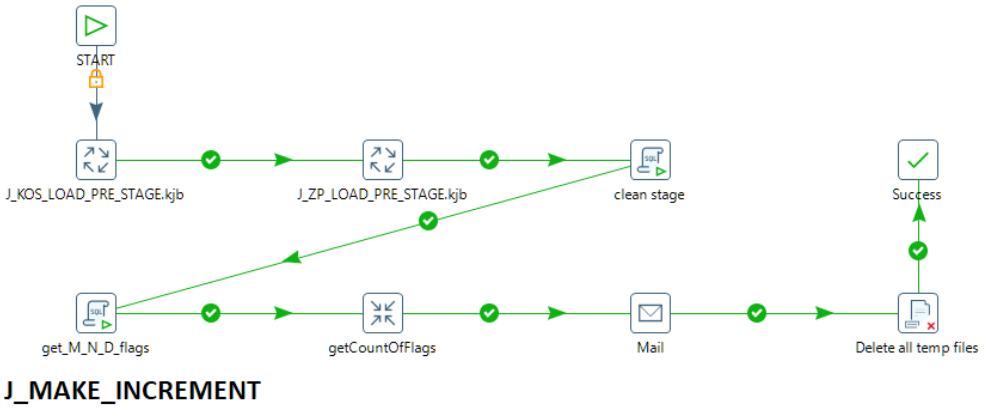
dat (viz sekce 2.1.2), automatizaci a sledování nahrávání dat do datového skladu.

Na obrázku 2.2 je vyobrazen popis nahrávání dat do **IDL**. Nejdříve se data nahrají pomocí **PDI** do pre–stage, která je implementována jako sada schémat, pro každý systém jedno schéma s prefixem **ps__**. Při tomto nahrávání se každému záznamu spočítá hash, kterou přidáme jako nový sloupec. Tomuto procesu se podrobně věnuje sekce 2.1.1.

Další sadou schémat, kterou je třeba vytvořit, je sada pro pre–stage clean. Této vrstvě se věnuje sekce 2.1.2. Jde o vytvoření tabulek po nahrání pre-stage, které neobsahují pro nahrání nebezpečné datové chyby, jako je příznak smazání, duplicity v klíčích a podobně. Do této vrstvy jsou data nahrávána po stažení do pre–stage. V současné implementaci používáme pouze PL/pgSQL skripty, které jsou jednoduché na správu a verzování. Na spouštění těchto skriptů používáme **PDI**, které je pro orchestraci úkonů nahrání datového skladu ideální.

Poslední sadou schémat je sada stage–increment. Tato vrstva obsahuje data, která jsou aktuální ke stavu před nahráním pre–stage. S těmito daty porovnáváme pre–stage a rozpoznáváme změny. K rozpoznání změn nám slouží MD5 hash[22] a metadatová tabulka obchodních klíčů tabulek. Do detailu se této vrstvě věnuji v sekci 2.1.1.

[22]MD5 hash je kolizní, avšak pro rozpoznání změny záznamu, který má většinu sloupců shodných (alespoň obchodní klíč), je vhodná. Koliznosti jsem si vědom a výsledek neovlivňuje.




Obrázek 2.3: ETL pro nahrání Stage databáze

### 2.1.1 Hledání změn ve zdrojových systémech

Použil jsem standardního konstruktu CDC – data jsou stažena do pre–stage, naleznou se změny oproti minulému stavu a tyto změněné záznamy se označí. Sledujeme 3 typy změn – záznam je nový, záznam se změnil a nebo byl záznam v původní databázi smazán.

Změnu rozeznáváme ze 2 parametrů – MD5 hashe záznamu a předem určeného obchodního klíče. Při nahrávání dat vytvoříme každému záznamu MD5 hash a tu přidáme jako textový sloupec. Poté, co se nahrají data do stage databáze, proběhne hledání změn, které je realizováno třemi databázovými procedurami. Následně je zjištěn počet upravených záznamů a tyto počty jsou odeslány administrátorovi na e-mail společně s logem transformace. Nejsnáze vysvětlí obrázek ETL 2.3.

Pro hledání těchto záznamů jsem využil dynamické SQL, kde pomocí procedury zpracuji postupně všechny tabulky z pre-stage, spustím nad nimi připravené příkazy a pokud najdu změnu, propaguji ji do stage-increment s patřičným značením změny. Kvůli tomu bylo nutné přidat všem tabulkám v stage-increment tyto pomocné sloupce:

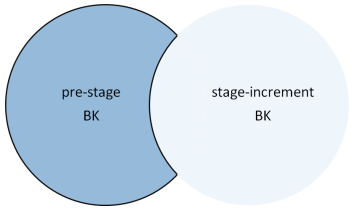
**md5** Textové pole obsahující MD5 hash záznamu bez technických sloupců.

**active** Příznakový sloupec obsahující informaci, zda je sloupec aktivní, tzn. že záznam nebyl smazán v původním systému.

**insertion–date** Datum a čas nahrání záznamu.

**last–update** Datum a čas poslední změny záznamu.

**state** Příznakový sloupec obsahující informaci, jak se záznam změnil – M (modified) pro změnu, N (new) pro označení nového záznamu a D (deleted) pro indikaci smazaného záznamu.




Obrázek 2.4: Diagram určení nového záznamu

Tabulky v pre–stage potřebují navíc pouze sloupec MD5. Dále jsme vytvořili tabulku obchodních klíčů[23] (business-keys) všech tabulek ve stage. Tato tabulka má jednoduchou strukturu: název tabulky, pole klíčů a název zdrojového systému. Dále uvádím části kódu, které vypíší ty záznamy, které každá procedura zpracovává. Na přiloženém CD jsou vloženy kódy celé, avšak pro tuto práci nejsou důležité detaily těchto procedur. V kódu používám prefixy „si_“ pro stage–increment a „ps_“ pro pre–stage.

#### 2.1.1.1 Hledání nových záznamů

Pro hledání nových záznamů jsem využil databázového joinu nad obchodním klíčem. Jde o smyčku nad všemi tabulkami, kde do stage–increment vložím ty záznamy, které vyhovují podmínce: nejsou ve stage–increment, ale jsou v pre–stage. U každé tabulky spustím následující kód:

```
1 INSERT INTO si_B
2 SELECT <> -- zkráceno, obsahuje automaticky generovaný seznam sloupců
3 FROM tableA ps_A
4 LEFT JOIN tableB si_B ON ps_A.BK = si_B.BK
5 WHERE si_B.key IS NULL;
```

Výpis 2.1: Dotaz pro vypsání nových záznamů

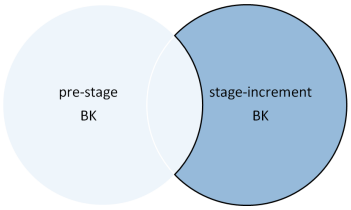
Určení nového záznamu je přehledné a s indexy nad obchodními klíči i rychlé. Na obrázku 2.4 je tmavě vyznačena výseč záznamů, které jsou označeny jako nové.

#### 2.1.1.2 Hledání smazaných záznamů

Hledání smazaných záznamů funguje na stejném principu jako hledání nových, ale přesně opačné. Znovu smyčkou projdu všechny tabulky a nad těmi spustím kód, který nejprve vytvoří dočasnou tabulku s obchodními klíči tabulek a následně zavolám UPDATE nad těmito záznamy, kterým nastavím stav na D – smazáno a active na 0:

```
1 -- vše, co je uzavřené v~<> obsahuje data získaná pomocí
```

[23] Nelze zaručit, že všechny tabulky obsahují primární klíč, tak jsme logický kandidátní klíč označili za obchodní.




Obrázek 2.5: Diagram určení smazaného záznamu

```
-- dynamického SQL tak, aby pro každou tabulku byl následující příkaz validní
CREATE TEMP TABLE <temp_table_name> AS
SELECT <business_keys>
FROM <tableA> si_A
WHERE active = '1'
EXCEPT
SELECT <business_keys>
FROM <tableB> ps_B ;

UPDATE <tableA> si_A
SET state = 'D',
    active = '0',
    last_update = <CURRENT_TIMESTAMP>,md5 = '0'
FROM <temp_table_name> T
WHERE si_A.BK = T.BK
```

Výpis 2.2: Dotaz pro vypsání smazaných záznamů

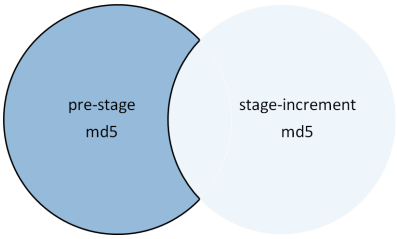
Pro ilustraci jsem přiložil obrázek 2.5.

#### 2.1.1.3 Hledání změněných záznamů

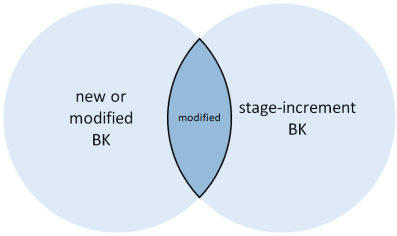
Hledání změněných záznamů jsem rozdělil na dvě části. V první části porovnám hashe záznamů a vyberu ty, které nejsou ve stage–increment. Tato množina obsahuje záznamy, které jsou nové a nebo změněné. Z této množiny vyberu změněné následovně – porovnám obchodní klíče této množiny a stage–increment a průnikem jsou nové záznamy. Nejlépe vysvětlí obrázek 2.6 nebo následující kód:

```
-- vše, co je uzavřené v~<> obsahuje data získaná pomocí
-- dynamického SQL tak, aby pro každou tabulku byl následující příkaz validní
CREATE TEMP TABLE <temp_table_name> AS
SELECT <>
FROM (
  SELECT ps_A.*
  FROM (
    SELECT *
    FROM tableA ps_A
      LEFT JOIN
        (SELECT md5,<business keys>
          FROM tableB si_B ON ps_A.md5 = si_B.md5

```




(a) Diagram nalezení nového nebo změněného záznamu.




(b) Diagram určení změněného záznamu.

Obrázek 2.6: Diagram určení změněného záznamu.

```
14          WHERE si_B.md5 IS NULL) NM
15        INNER JOIN
16
17          (SELECT *
18            FROM tableB) si_B ON ps_A.BK = si_B.BK
19
20  UPDATE <tableB> si_B
21  SET <si_B.column = I.column>
22    FROM <temp_table_name>
23  I~WHERE si_B.BK = I.BK
```

Výpis 2.3: Dotaz pro vypsání změněných záznamů

Tato procedura označí jako změněné i záznamy, které byly smazány a znovu vytvořeny se stejným obchodním klíčem. Tato funkcionalita je zapříčiněna tím, že ze stage–increment nejsou záznamy smazány, ale jsou označeny jako neaktivní. Tudíž pokud je záznam znovu vytvořen, nemůže být označen jako nový – obchodní klíč již ve stage–increment existuje, ale jako změněný.

### 2.1.2 Technické čištění dat

Z důvodu vlastních procedur pro hledání změn (viz 2.1.1) bylo nutné navrhnout vlastní hrubé technické čištění dat tak, aby bylo možné tyto procedury spouštět.

Vytvořili jsme proto další schéma ve stage databázi pro každý systém, a to s prefixem **psc** (např. psc_kos) s významem pre–stage clean. Každou z těchto tabulek generují procedury spuštěné po stažení pre–stage. Tyto procedury upravují následující chyby v datech:

**Příznak smazání** Pokud tabulka obsahuje tento příznak, je třeba tyto záznamy z tabulky vyloučit, jako by byly smazané.

**Zdrojová tabulka je historizovaná** Pokud tabulka obsahuje i historické záznamy, pak její obchodní klíč není unikátní a tudíž je třeba tuto historii odstranit a pracovat pouze s aktuálními záznamy, které historizujeme při nahrávání do IDL.

**ID může být null** Pokud v tabulce je identifikátor záznamu (kandidátní klíč), který může nabývat hodnoty null a jsme schopni tuto hodnotu dohledat, dohledáme ji na této vrstvě, než se začnou data integrovat s dalšími systémy. Jako příklad bych uvedl tabulku kontaktů v systému KOS, kde je uvedeno buď ID osoby anebo ID studenta. Jelikož i student je osoba, lze dohledat jeho osobní číslo a tím vyplnit null záznamy.

Tyto procedury se spouští v transformaci z obrázku 2.3 v části **clean stage**, která vytvoří „čisté“ tabulky a také smaže příznaky z minulého nahrání, aby mohly být označeny změny.

Dále jsme také vytvořili psc_ude, to znamená pre–stage clean uměle definovaných entit. Tyto entity jsme vytvořili pro snazší a rychlejší práci v ETL. Jelikož některá ETL potřebují složité databázové joiny více než jednou, rozhodli jsme se zařadit do architektury a procesů stage i tyto tabulky. Po stažení pre–stage se vytvoří tabulky pre–stage clean a následně i psc_ude. Poté se spustí procedury hledání změn i nad těmito tabulkami.

Jak jsem již psal, snažil jsem se v nové implementaci celého nahrávání datového skladu co nejméně používat PDI, a to kvůli snažším úpravám kódu, a také z výkonnostních důvodů. Tyto uměle definované entity připravují přímo tabulku IDL tak, aby mohla být začleněna do procesu hledaných záznamů pomocí přírůstků.

### 2.1.3 Rozdílné způsoby získávání dat

V době odevzdání této práce jsou do datového skladu zakomponovány systémy KOS, Anketa ČVUT a systém závěrečných prací. O způsobech získávání dat z jednotlivých systémů píši v následujících subsekcích.

#### 2.1.3.1 KOS

Databáze systému KOS je pod správou Výpočetního a informačního centra ČVUT. Z výkonnostních důvodů jsme nedostali přístup do produkční databáze a z finančních důvodů není možná replikace dat ani zaručení dodání exportů na naše úložiště.

Máme přístup do testovací databáze, která se obnovuje jednou týdně, avšak se stejnou strukturou a se stejnými daty jako produkční. Z důvodu přímého přístupu do databáze jsme nuceni stahovat celé tabulky a sami zjišťovat, jaké změny byly provedeny nad daty, viz sekce 2.1.1.

#### 2.1.3.2 Systém závěrečných prací

Systém závěrečných prací vznikl na FIT pro snazší zadávání a kontrolu procesu závěrečné práce. K tomuto systému jsme dostali plný přístup do produkční databáze, tudíž znovu využíváme stejného postupu jako u systému KOS.

#### 2.1.3.3 Anketa ČVUT

Systém Anketa slouží k hodnocení učitelů a předmětů na ČVUT studenty. Přístup k tomuto systému jsme dostali částečný. Po uzavření ankety jsou nám příslušné tabulky nahrány na úložiště, abychom je mohli stáhnout. Jelikož se anketa uzavírá vždy po skončení semestru a v nepravidelných intervalech, tak stahování dat provádíme ručně na žádost správce. Data exportujeme do csv a následně nahráváme do databáze.

Kapitola 3

# ETL modelování

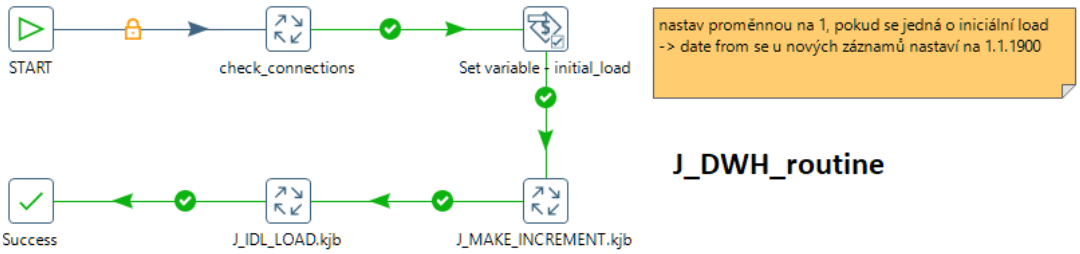
Pro tento projekt jsme zvolili sadu nástrojů Pentaho Data Integration Community Edition[24]. Jedním z důvodů je to, že je open source a že je k němu dostatečná dokumentace. Obsahuje všechny nástroje, které pro náš případ potřebujeme, avšak historizační procedury jsem z důvodu urychlení celého procesu nahrávání vytvořil vlastní v SQL. ETL vytvořené v PDI se dělí na dvě části – transformations (transformace) a jobs (úlohy). Trasformace slouží k přesunu a transformaci jednotlivých záznamů mezi dvěma umístěními (databáze - databáze, soubor - databáze, atd). Úlohy slouží k orchestraci a spouštění jednotlivých transformací a PDI modulů (například poslání e-mailu, práce se souborovým systémem atd.). Hlavní úlohou, která spouští nahrání dat do IDL, je *J_DWH_routine*, viz obrázek 3.1. V prvním kroku úlohy je nastavena proměnná, která značí, zda se jedná o iniciální nahrání[25] nebo jde o běžný přírůstek do datového skladu. Dále je spuštěna úloha, která zajistí stažení dat všech zdrojových systémů a následné zpracování přírůstku ke Stage. Jako poslední je úloha nahrání dat do IDL. V této kapitole popíši, jak jsem postupoval v návrhu a tvorbě ETL.

## 3.1 Přírůstek ke Stage

V sekci 2.1.1 jsem popsal jakým způsobem rozpoznávám změny. Tyto procedury spouštím z úlohy *J_MAKE_INCREMENT* (viz obrázek 2.3). Důvod, proč jsem nepoužil nástrojů PDI, je hlavně v rychlosti zpracování a také proto, že dynamické SQL je snazší a přehlednější na správu, i když náročnější na vývoj. Úloha zpracování přírůstku dat je velice přímočará a přesně kopíruje popis sekce 2.1.1. Nejdříve jsou stažena data do pre–stage pomocí úloh *J_KOS_LOAD_PRE_STAGE* a *J_ZP_LOAD_PRE_STAGE*. Následně je

[24] http://community.pentaho.com/

[25] Pro iniciální nahrání datového skladu platí jiná historizační pravidla. Příznak začátku platnosti záznamu je pro iniciální nahrání nastaven na nulté datum dle konvence stanovené architekturou skladu – v našem případě 1.1.1900.




Obrázek 3.1: Úloha nahrání dat do IDL

vytvořena vrstva pre–stage clean a poté proběhne hledání změn ve zdrojových systémech. Toto hledání změn je realizováno databázovými procedurami, které spouštím z PDI. Předposlední transformací je zjištění počtu změněných záznamů a následné odeslání tohoto počtu společně s logem úlohy administrátorům datového skladu.

## 3.2 Návrh vlastní historizační procedury

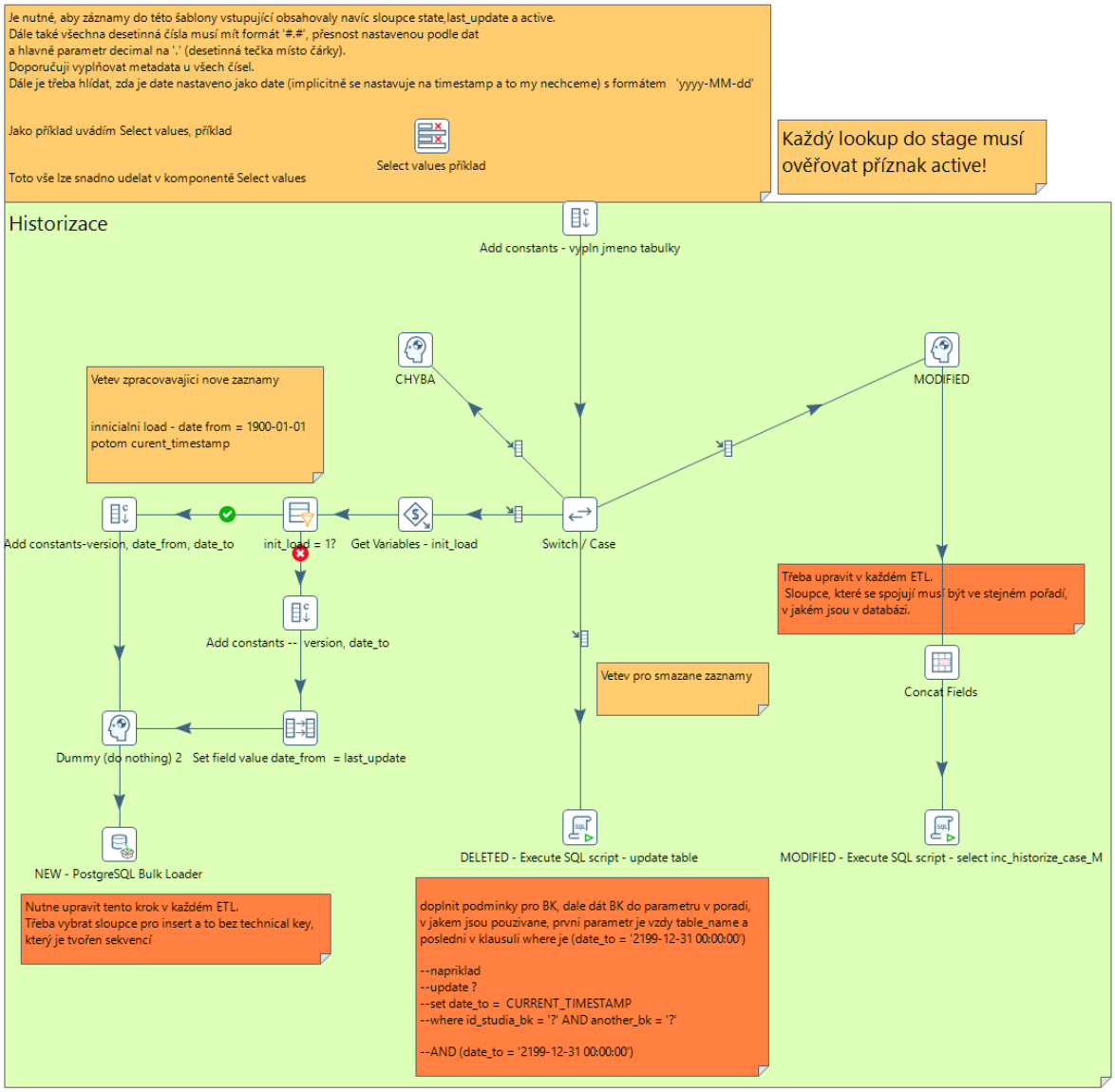
Dalším zrychlením našich ETL procesů bylo navržení vlastní historizační procedury, která zpracovává námi označené záznamy. V datovém skladu používáme SCD0, SCD1 a SCD2 na úrovni sloupců. To znamená, že každý sloupec má příznak, jestli se nemůže měnit (SCD0), je volně přepisovatelný (SCD1) anebo se udržuje celá historie jeho změn (SCD2). K tomu jsme vytvořili 2 metadatové tabulky, ve kterých udržujeme tyto informace. Uložené máme informace o sloupcích označených SCD0 nebo SCD1, pokud sloupec není ani v jedné z těchto tabulek, je udržována jeho plná historie. K tomuto účelu jsem vytvořil šablonu v PDI, která tyto záznamy zpracovává, viz obrázek 3.2. Tato šablona má na vstupu řádek/záznam takový, jaký má být uložen do IDL, obohacený o **active**, **last_update** a **state** příznaky. Následně se podle příznaku rozdělí na 3 větve popsané v následujících sekcích.

### 3.2.1 Nový a smazaný záznam

**Nový záznam** se zpracovává celý v PDI. Nejdříve se ověří, zda jde o iniciální nahrání. Pokud ano, je počáteční datum platnosti záznamu 1.1.1900. Pokud ne, tak je datum zpracování záznamu ve stage databázi. Následně se nahraje pomocí PDI komponenty bulk loader[26] do cílové tabulky v IDL.

**Smazaný záznam** se zpracovává následovně – zavolá se databázový update nad záznamem. V následujícím kódu je vidět, že pro každý zpracovávaný řá-

[26]Tato komponenta nahrává data po seřazených skupinách, aby začlenění každého záznamu do stromu indexů netrvalo příliš dlouho.




Obrázek 3.2: Šablona pro historizaci ETL

dek je spuštěn update ukončující platnost záznamu. První otazník nahradí proměnná **table_name** a druhý je nahrazen příslušným obchodním klíčem. Tento kód je z ETL pro nahrání studijního oboru, který má jako obchodní klíč **studijni_obor_bk** (bigint).

```
update dwh.?
set date_to =  CURRENT_TIMESTAMP
where studijni_obor_bk = '?'
AND (date_to = '2199-12-31 00:00:00');
```

Tento update je spouštěn PDI komponentou Execute SQL statements, která pro každý zpracovávaný řádek spustí příslušný SQL příkaz. Tuto komponentu lze nakonfigurovat tak, že nahradí otazníky určitými parametry ze vstupního řádku nebo libovolnou proměnnou. Dále také tato komponenta ošetří řetězec tak, aby nezpůsoboval problémy při spouštění SQL příkazu – vyruší význam znaků se speciálním významem, jako je pomlčka, ampersand a jiné.

### 3.2.2 Změněný záznam

Změněný záznam zpracováváme mimo PDI ve vlastní databázové proceduře. Tuto proceduru spouštíme z PDI pomocí komponenty Execute SQL statements, kterou jsem již popisoval v předchozím odstavci. Tato procedura má za vstup spojený řetězec sloupců, který vytvoříme v komponentě Concat fields.

V této komponentě je nutné pouze udržet pořadí sloupců takové, jaké je v databázi. PostgreSQL garantuje, že pořadí sloupců v databázi je neměnné. Výstupem je tedy řetězec sloupců oddělených znakem **\x1f** (Information Separator One).

Dalším vstupem procedury zpracovávající změněné záznamy je název tabulky, příznak **active** a čas poslední změny záznamu ve stage. Tato procedura zpracovává v závislosti na **active** flagu záznamy následujícím způsobem:

```

CREATE OR REPLACE FUNCTION public.inc_historize_case_m(
    _record text,
    _table_name text,
    _active integer)
  RETURNS void AS
BEGIN
  IF _active = 1
  THEN
    PERFORM inc_historize_case_M_1(_record, _table_name);
  ELSE
    PERFORM inc_historize_case_M_0(_record, _table_name);
  END IF;
END;
```

#### 3.2.2.1 Active flag = 0

Pokud je tento příznak nastaven na 0, znamená to, že v minulosti byl záznam smazán a byl znovu založen se stejným obchodním klíčem. Z tohoto důvodu je nutné pouze vložit tento záznam jako nový s **date_from** nastaveným na den poslední změny záznamu ve stage a s příznakem **version** inkrementovaným o 1 oproti poslednímu záznamu.

Tento proces zajišťuje procedura inc_historize_case_m_0(text, text,timestamp without time zone). Procedura využívá dynamického SQL pro zjištění názvu sloupců a dalších pomocných atributů. Nejdůležitější kus kódu je (zjednodušeně):

```
1 INSERT INTO || _target_schema_CONS || . || _table_name
2  || ( || _non_technical_columns_separated_by_comma || ,
3 version,
4 date_from,
5 date_to,
  last_update)
6 VALUES(|| _record_for_insert || , || _version || ,
7 || _current_timestamp
8 || ,2199-12-31 00:00:00, || _current_timestamp || );
```

Jednotlivé proměnné kódu:

**__target__schema__CONS** obsahuje schéma datového skladu,

**__table__name** je název tabulky do které se záznam vkládá,

**__non__technical__columns__separated__by__comma** je řetězec obsahující všechny netechnické sloupce oddělené čárkou v tom pořadí, v jakém jsou definovány v příslušné tabulce,

**__record__for__insert** je upravený vstup (záznam určený pro vložení do databáze) v takové podobě, aby byl využitelný v insert příkazu,

**__version** je verze vkládaného záznamu – získává se inkrementací nejvyšší současné hodnoty verze záznamu se stejným obchodním klíčem,

**__current__timestamp** je čas poslední úpravy záznamu ve Stage.

#### 3.2.2.2 Active flag = 1

Pokud je příznak **active** nastaven na 1, znamená to, že je třeba provést standardní historizační proceduru podle pravidel SCD (viz sekce 1.4.2).

Této funkcionality je docíleno pomocí procedury inc_historize_case_m_1(text, text, timestamp without time zone), která pomocí dynamického SQL nejdříve zjistí názvy sloupců a pomocné atributy. Následně porovná, jestli se změnil

nějaký z námi použitých sloupců, protože změna sloupce ve zdroji, který ale nepoužíváme v IDL, nevyvolá samozřejmě žádnou akci.

Pokud tento změněný sloupec patří do skupiny označené SCD0, je záznam vložen do metadatové tabulky a je o této chybě informován administrátor, který změnu zpracuje ručně. Pokud je změněný sloupec ze skupiny s SCD1, je zavolán update záznamu. Pokud je změněn sloupec, který nepatří ani do jedné ze skupin, je vytvořen nový záznam, jehož platnost začíná časem poslední změny záznamu ve Stage a na tento čas se nastaví konec platnosti původního záznamu.

Základní krok procedury je totožný s předchozím popisem a volání SQL příkazů je velice podobné předchozímu případu, tudíž zdrojový kód přikládám jen na přiložené CD.

## 3.3 Tvorba ETL pro integraci dat do datového skladu ČVUT

V rámci své práce jsem také navrhl šablonu (viz obrázek 3.2) a upravil všechna ETL, aby odpovídala novému standardu. V rámci dipolomové práce Jakuba Krejčího [20] vznikla jmenná konvence datového skladu, tudíž bylo potřeba všechna ETL pozměnit a přejmenovat sloupce. Z tohoto důvodu vznikly i menší úpravy datového modelu, které jsme objevili používáním.

Jelikož je má implementace na obsluhu složitější, než původní historizační procedura, pokusil jsem se ji co nejvíce vysvětlit v komentářích šablony, avšak stejně bylo nutné vytvořit metodiku. Tato metodika slouží v současné době při vytváření dalších ETL procesů. V budoucnu může fungovat jako dokumentace či jen popis procesu. V této sekci ukážu nejdůležitější body, které zatím nejsou popsané v této práci. Postup úpravy ETL je však samovysvětlující a osvětlí celý proces a postup změny již existujících ETL.

Jelikož jsem pozměnil celý způsob rozpoznávání změn z **PDI** komponenty **Dimension Lookup Update** na vlastní sadu databázových procedur, je nutné nejprve vysvětlit celý ETL proces nahrání dat do IDL. Tento proces jsem rozdělil na 2 části – nahrání dat do Stage databáze a nahrání dat do IDL (někdy také označujeme jako Target database).

**Nahrání dat do Stage** databáze se skládá z 6 kroků:

1. Smazání pre_stage.
2. Nahrání otisku dat zdrojových systémů do pre_stage s přidáním technického sloupce s md5 hash, která se počítá při stahování dat.
3. Vynulování **state** sloupce tabulek pre_stage.
4. Vyhledání nových záznamů v pre_stage a jejich nahrání do stage_increment se **state = N**.

5. Vyhledání smazaných záznamů v pre_stage a modifikace těchto záznamů ve stage_increment na **state = D** a **active = 0**.

6. Vyhledání modifikovaných záznamů v pre_stage a modifikace těchto záznamů ve stage_increment na **state = M**.

**Nahrání dat do IDL** probíhá standardními ETL procesy, kterým je jen upraven proces nahrání do databáze. Původně jsme používali komponentu **Dimension Lookup Update**, která zapracovávala záznamy do cílové databáze, ale v druhé verzi datového skladu je použit stage_increment, pomocí kterého je historizace mnohem snazší a rychlejší. Výhodou je, že ETL záznamy, které berou data jen z jednoho zdroje, nemusí zpracovávat záznamy, které mají **state = null**.

Je tedy potřeba upravit původní ETL následujícím způsobem:

- Místo **Dimension lookup/update** bude použita šablona historizace. Tato šablona obsahuje přepínač, který rozhoduje podle **state**, která větev historizace bude použita.

- Je nutné, aby záznamy do této šablony vstupující, obsahovaly navíc sloupce **state** a **active**. Dále také všechna desetinná čísla musí mít formát '#.#', přesnost nastavenou podle dat a hlavně parametr decimal na '.' (desetinná tečka místo čárky).

- Jelikož záznamy ze stage_incrementu nemažeme, je nutné přidat při použití komponenty **Database Lookup** do stage podmínku active = '1'.

- V šabloně je třeba upravit:
  - Add constants
    - Přiřadit konstantě „TABLE_NAME“ správnou hodnotu.
  - PostgreSQL Bulk Loader
    - Zde je třeba vybrat všechny sloupce, které náleží TARGET tabulce. Nejsnazší je použít **Get fields** a odebrat **TABLE_NAME**, **active** a **state**. Technický klíč je tvořen sekvencí celých čísel.
  - Execute SQL script - update target table
    - Tato větev zpracovává smazané záznamy, v našem případě se smazání řeší ukončením platnosti záznamu.
    - Tento skript je třeba upravit v každém ETL, a to podmínku where – v této podmínce je třeba vypsat všechny business klíče oddělené čárkou. Tyto BK musí korespondovat s parametry.
  - Concat Fields

    - Tato komponenta vytvoří řetězec, ve kterém jsou spojené všechny sloupce a oddělené znakem **\x1f** (Information Separator One).
    - V této komponentě je nutné pouze vybrat všechny netechnické sloupce (tlačítko **Get Fields** ulehčí práci) a zkontrolovat, jestli jsou sloupce ve stejném pořadí, jako v DDL tabulky.
  - Execute SQL script - select temp.inc_historize_case_M
    - Tato větev je pro změněné záznamy, které zpracovávají 2 procedury, a to podle toho, zda byly před změnou aktivní anebo ne. Neaktivní změněný záznam znamená, že byl záznam se stejným BK smazán a znovu vložen.
    - Funkce temp.inc_historize_case_m( _record text, _table_name text,_active integer) má 3 argumenty, a to:
      - záznam, který je potřeba vyhodnotit, typu string, oddělený **\x1f**
      - název tabulky
      - active flag
  - Tato šablona zpracovává SCD0, SCD1 a SCD2 a to tak, že bere metadata o typu historizace z tabulek:
    - metadata.scd0_tables_columns
    - metadata.scd1_tables_columns
    - tyto tabulky obsahují vždy název tabulky a pole atributů, které jsou buď neměnné (SCD0) a nebo volné k přepisování (SCD1)

Tento postup je třeba dodržet u každého nově vytvořeného ETL. Dále jsem musel upravit všechna stávající ETL. Změna tohoto procesu způsobila zrychlení v řádu hodin. Dřívější nahrávání datového skladu trvalo mezi 8 a 12 hodinami, nyní je sklad nahrán mezi 90 - 120 minutami.

## 3.4 Automatické spouštění nahrávání dat do datového skladu ČVUT

V této sekci využívám bakalářskou práci Radima Lengera [21], konkrétně upraveného ETL Daemona, kterého kolega Lenger implementoval v C++.

Pro spouštění úloh v PDI lze využít jejich vnitřní nástroj, který umí plánovat a také spravovat běh transformací. Toto řešení je však dobré pouze pro lokální využití. Funkcionalitu vzdáleného spouštění umožňuje součást PDI, která se jmenuje Carte. Carte je jednoduchý web server, který umožňuje spouštět a monitorovat jednotlivé procesy.

Instalace a správa však této součásti není tak přímočará, že bychom ji chtěli použít. V nové implementaci nahrávání dat do IDL jsem se snažil co nejméně používat služby PDI z důvodu špatně zdokumentovaných chybových hlášení a

složitého testování. Zvolil jsem proto jednodušší možnost, a to součást Kitchen pro spouštění úloh a transformací, které vytvořím v PDI. Tato kombinace se osvědčila a nepotřebuje složité nasazování. Kitchen je shellový skript, který spustí dodanou úlohu (ve formátu xml) a na standardní výstup posílá log této úlohy.

Pro automatické spouštění jsem napsal shellový skript, který má plánované spouštění. Nejdříve tento skript spustí úlohu, které ověří, že jsou všechna připojení k databázím přístupná a následně spustí samotnou úlohu nahrávání.

**Check connection** je shell-script, který spustí **PDI Kitchen** a jako vstup mu předá jednoduchou úlohu, která ověří, zda jsou všechna připojení dostupná. Pokud úloha dopadne dobře, zašle e-mail administrátorům, že je vše v pořádku a začíná nahrávání spuštěním skriptu **Run ETL Jobs**. V opačném případě zašle e-mail, že se nepodařilo připojit k některé z databází a skript končí chybou.

**Run ETL Jobs** je shell-scipt, který spustí ETL úlohu a vyhodnotí návratovou hodnotu. Podle této návratové hodnoty rozhodne, který z předpřipravených e-mailů zašle administrátorovi. Více v sekci 3.5.1.

## 3.5 Monitoring ETL procesů

Pro monitoring ETL procesů jsem zvolil zasílání strukturovaných e-mailů. Tyto e-maily vždy obsahují i log nahrávání. Dále je samozřejmě log archivován a shellový skript, který spouští nahrávání, loguje své chování do syslogu systému.

Pokud ETL dopadne dobře, je zaslán e-mail s datem a zprávou, že je vše v pořádku.

Pokud ETL skončí s chybovou hláškou, dekóduji tuto hlášku a zašlu instrukce administrátorům. Více v sekci 3.5.1.

Dále v ETL pro nahrání Stage increment ukládám informace o tom, kolik změněných, nových nebo smazaných záznamů měla každá tabulka. Tuto informaci pak zašlu administrátorovi ještě před tím, než se spustí nahrávání do IDL. Administrátor může zasáhnout, pokud bude doba zakomponování změn do datového skladu příliš dlouhá.

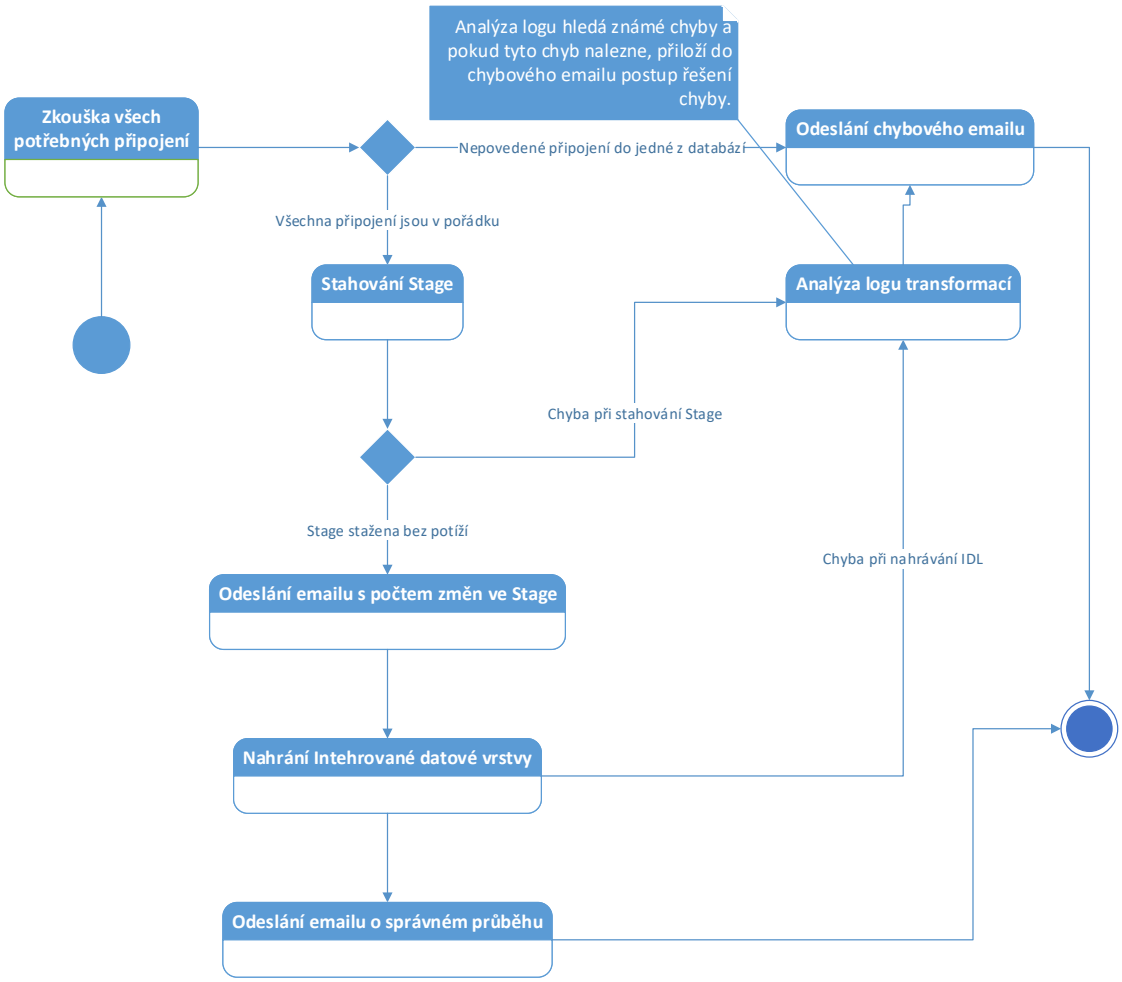
Průběh nahrávání datového skladu probíhá podle stavového diagramu, viz obrázek 3.3.

Scénář nahrávání datového skladu, pokud nenastane žádná z chyb, je následující:

Přijde první e-mail:

---

**from:** ETL agent <big.etl.agent@gmail.com>




Obrázek 3.3: Stavový diagram nahrávání datového skladu

**to:** <adminMails>

**subject:** ETL server - zdařilé připojení ke zdrojovým databázím

**message:** Po bře 27 09:20:22 CEST 2017 : ETL serveru se podařilo připojit ke všem potřebným datovým zdrojům. DWH rutina může začít.

---

Pokud proces trvá déle než je obvyklé (více než 90 minut) nebo je jen nutné zjistit, jak dlouho ještě přibližně poběží, je možné sledovat 2 logy:

1. **Postgres log**

Na serveru, kde je postgres databáze, lze sledovat log **postgresql-9.5-main.log**, do kterého zapisují zprávy jednotlivé SQL procedury pro hledání změněných / nových / smazaných záznamů. Nejdéle trvá hledání změněných záznamů, avšak zpracování největší tabulky trvá maximálně půl hodiny, tudíž lze snadno sledovat, kde se zrovna proces nachází.

Jako ukázku vložím několik řádků:

```
... etl_agent@dwh3_stage NOTICE: inc_find_modified_in_pre_stage>zpracoval jsem tzkousky_nahr a dotklo se to 0 radku
... etl_agent@dwh3_stage NOTICE: inc_find_modified_in_pre_stage>zpracovavam tstpruch
... etl_agent@dwh3_stage NOTICE: inc_find_modified_in_pre_stage>zpracoval jsem tstpruch a dotklo se to 0 radku
... etl_agent@dwh3_stage NOTICE: inc_find_modified_in_pre_stage>zpracovavam tpredmety
... etl_agent@dwh3_stage NOTICE: inc_find_modified_in_pre_stage>zpracoval jsem tpredmety a dotklo se to 76 radku
```

2. **PDI log**

   Transformace spuštěná v **PDI Kitchen** má vlastní log, který je ukládán do složky ke všem logům, které se týkají datového skladu. Log má jednoduché pojmenování ***datum_spuštění*-kitchenLog**.

Další e-mail informuje o stažení a označení všech upravených záznamů.

---

**from:** ETL agent <big.etl.agent@gmail.com>

**to:** <adminMails>

**subject:** ETL server - Nahrání stage

**message:** Job:

---

JobName : J_MAKE_INCREMENT

Directory : /

JobEntry : Mail

Message date: 2017/03/27 11:05:21.346

<rest of log>

**attachement:**

| sum | operation_type | process_date |
|---|---|---|
| M | 40067 | 27.3.2017 9:54 |
| D | 549 | 27.3.2017 9:54 |
| N | 11631 | 27.3.2017 9:54 |

---

Poslední e-mail, který administrátor dostane, informuje o správném ukončení ETL procesů. Pro ukázku:

---

**from:** ETL agent <big.etl.agent@gmail.com>

**to:** <adminMails>

**subject:** ETL server - korektní průběh ETL procesů

**message:** Po bře 27 09:20:22 CEST 2017 : ETL procesy proběhly v pořádku.
V příloze zasílám log celé transformace.

**attachement:** <2017-03-27-kitchenLog>

---

Zde popsaný proces je platný pouze tehdy, pokud nenastane žádná z chyb při nahrávání dat. V další sekci popíši chyby, ke kterým může dojít, a jak na ně reagovat. Není možné v této práci pokrýt všechny chyby, které mohou při běhu ETL nastat, avšak z logu transformace je možné chybu pochopit, reprodukovat a opravit.

### 3.5.1 Chyby při nahrávání ETL procesů a reakce na ně

V rámci mé práce jsem měl za úkol vymyslet a sepsat reakční scénáře na běžné chyby. Tyto reakční scénáře jsem pro snazší použitelnost přibalil přímo do chybového e-mailu, který je odesílán administrátorům.

Základní chyba, která může nastat, je nenalezení xml souboru s předpisem ETL. Tato skutečnost je ohlášena e-mailem a zapsána do systémového logu.

Pokud se ETL spustí, je možná chyba už jen v běhu PDI. Pentaho v dokumentaci k programu Kitchen definuje různé návratové kódy a jejich význam. V následující části popíši, co jednotlivé kódy znamenají a jak vypadá e-mail odeslaný správci, pokud takováto chyba nastane. Jelikož všechny transformace probíhají transakčně, stačí opravit chybu a spustit úlohu od tohoto místa dále a nebude porušena integrita dat.

**Návratová hodnota je 0** – pak je vše v pořádku a bude odeslán e-mail, který je popsán v sekci 3.5 a do syslogu je zapsáno, že vše proběhlo bez problémů.

**Návratová hodnota je 1** – znamená, dle dokumentace **PDI**, problém při běhu ETL transformací. Tento návratový kód je jeden z nejčastějších. Jelikož jen kód nevypoví vše o chybě, je nutné nahlédnout do logu transformace. Log transformace procházím skriptem, který když nalezne některou z obvyklých a

již řešených chyb, přidá do e-mailu postup, jak tuto chybu nahrání opravit. Tyto problémy jsou:

**Error connecting to database** V překladu „Nastal problém s připojením do databáze“. Tento problém může nastat již při pokusu o připojení nebo v průběhu stahování dat ze zdrojových systémů.

**Unable to find field PDI** očekává v některé z tabulek či souborů určitý sloupec, který tam není. Ve většině případů jde o to, že se změnila struktura zdrojové tabulky (sloupec byl přejmenován nebo smazán).

**Unexepected conversion error** Nějaký sloupec či pole v souboru obsahuje data v jiném formátu. Tento problém nastává, pokud se změní zdrojový systém a obsluha datového skladu o tom není informována (stejně jako předchozí bod).

Pokud by se v jednom běhu úlohy objevily všechny tyto chyby, e-mail by vypadal následovně:

---

**from:** ETL agent <big.etl.agent@gmail.com>

**to:** <adminMails>

**subject:** ETL server - chyba při běhu ETL transformací

**message:** Ne dub 9 15:40:24 CEST 2017 : Problém běhu ETL transformací : Errors occurred during processing

Nastal problém s připojením do databáze. Doporučuji projít tyto body:

-zkontroluj nastavení proměnných prostředí v souboru /var/big/PDI/data-integration/spoon.sh,

-zkontroluj nastavení VPN rozvoje - připojení do systému závěrečných prací,

-projdi log a zkontroluj dostupnost zasažených serverů.

Problém, který nastal, souvisí s špatným pojmenováním sloupce. Tato chyba se objevuje, pokud zdrojový systém přejmenuje sloupec a Pentaho Kettle se s tím neumí vypořádat. Doporučuji projít log, lokalizovat transformaci, která hlásí chybu a zkontrolovat vstup.

Problém, který nastal, souvisí s konverzí dat. Doporučuji lokalizovat příslušnou transformaci a krok. Tato chyba nejčastěji nastává, pokud se změní datový typ ve zdroji.

Jelikož jsou všechny transformace prováděny v transakci, není třeba obnovovat zálohy. Stačí pouze opravit ETL a spustit rutinu od tohoto bodu dále.

**attachement:** <2017-04-09-kitchenLog>

---

**Návratová hodnota je 2** – znamená neočekávanou chybu při běhu. Na identifikaci chyby bude potřeba prohledat manuálně (administrátorem) syslog a log úlohy.

**Návratová hodnota je 7** – chybí nebo je porušen některý z xml předpisů ETL úloh. Podle logu úlohy lze chybu lokalizovat.

**Návratová hodnota je 8** – chybí nebo je porušen některý z použitých pluginů PDI.

Monitoring postrádá grafické prostředí, avšak současná implementace je dostatečná pro zkušenou obsluhu a při dalším rozvoji tohoto projektu by určitě bylo zajímavé investovat prostředky do webového prostředí, ve kterém by bylo možné sledovat jak běh ETL, tak zátěž jednotlivých serverů. V současné době je nutné vše dělat ručně, avšak projekt je ve fázi prototypu, tudíž je takovéto řešení dostatečné.

Kapitola 4

# Testování navrženého řešení

Testování řešení jsem rozdělil na dvě části. V první části testuji rozpoznávání změn dat ve zdrojových systémech a druhá část se týká propagace a těchto změn do datového skladu.

Pro testování první části jsem vytvořil kopii používané tabulky a napsal jsem databázové funkce (Výpisy 4.1 a 4.2), kde první nejdříve provede sto změn od každého typu (přidá záznamy, smaže záznamy a změní záznamy) a druhá poté vyvolá hledání těchto změn. Pro hledání změn jsem upravil původní procedury tak, aby přijaly název tabulky, která se má prohledávat. Výstup těchto testovacích funkcí – výpis 4.3 ukazuje, že procedury pracují správně a naleznou změny.

Tímto jsem otestoval pouze nalezení změn. Testování historizační procedury jsem provedl spuštěním ETL nad kopií tabulky z IDL, které jsem vymazal statistiky. Po nahrání dat jsem zkontroloval databázové statistiky, jestli budou dosahovat žádaných výsledků. Na obrázku 4.1 je vidět:

**Vložených záznamů je 200** Bylo vloženo 100 záznamů (nových) a každý změněný záznam vyvolal vložení nové verze tohoto záznamu (dalších 100 záznamů), podle historizace typu SCD2 (viz sekce 1.4).

**Aktualizovaných záznamů je 200** Změna záznamu ve zdrojovém systému vyvolá ukončení platnosti původního záznamu a vložení nového záznamu – to je jedno sto záznamů. Druhé sto záznamů jsou smazané záznamy – těm se pouze ukončí platnost, což je reprezentováno aktualizací záznamu (ukončení platnosti), viz sekce 1.4.

**Smazáných záznamů je 0** V datovém skladu nemažeme, pouze ukončujeme platnost.

Testování zpracování těchto označených záznamů je tedy provedeno nahráním do IDL a kontrolou statistik tabulky. Mé řešení (část týkající se automatického nahrávání dat do datového skladu) spočívalo pouze v úpravě již fungujících a ověřených ETL procesů – tudíž bylo potřeba pouze zkontrolovat

správnost samotného nahrání, ne faktická správnost záznamu. Řešení bylo nasazeno do testovacího prostředí již v lednu tohoto roku a nebyly zaznamenány jakékoliv chyby spojené se špatnou integrací dat.

Detailní zaznamenávání procesu nahrávání dat začalo až v březnu roku 2017. Z tohoto důvodu uvádím statistiky pouze od tohoto měsíce. Grafy 4.2 ukazují, že nahrávání datového skladu ČVUT trvá mezi jednou až dvěmi hodinami. Specifikace ETL serveru jsou následující:

**CPU** Intel(R) Xeon(R) CPU E7- 2870 @ 2.40GHz

**RAM** 16 GB

Dřívější implementace nahrávání dat do datového skladu trvala přibližně 8 – 12 hodin kvůli nevhodnému použití PDI komponenty „Dimension Lokup/Update“. Při zachování vniřní struktury ETL procesu, pouze změnou rozpoznávání a propagaci změn, jsem dosáhl zrychlení nahrávání na použitelnou úroveň. Jak je také vidět z grafu 4.2, nahrávání probíhá již 3 měsíce jednou týdně (důvod vysvětlen v sekci 2.1.3.1) a nevykazuje žádné chyby. Data jsou dostupná v integrované datové vrstvě a přístupná dalšímu zpracování.

```
1 NOTICE:  inc_test_flags_create_changes> smazal jsem 100 radku
2 NOTICE:  inc_test_flags_create_changes> zmenil jsem 100 radku
3 NOTICE:  inc_test_flags_create_changes> pridal jsem 100 radku
4 NOTICE:  inc_test_flags_find_changes> nalezl jsem 100 smazanych radku
5 NOTICE:  inc_test_flags_find_changes> nalezl jsem 100 zmenenych radku
6 NOTICE:  inc_test_flags_find_changes> nalezl jsem 100 novych radku
```

Výpis 4.3: Výstup testování navržených CDC procedur

```
CREATE OR REPLACE FUNCTION PUBLIC.inc_test_flags_create_changes()
  RETURNS VOID
AS
$body$DECLARE
  _row_count INT;
  _iterator  INT;
BEGIN
  --test deleted
  EXECUTE 'delete from ps_temp.tekosoby
    where peridno_bk in
      (select peridno_bk from ps_temp.tekosoby
       limit 100);';
  GET DIAGNOSTICS _row_count = ROW_COUNT;
  RAISE NOTICE 'inc_test_flags_create_changes> smazal jsem % radku', _row_count;

  --test modified
  EXECUTE 'update ps_temp.tekosoby set md5 = md5(random()::text)
   , jmeno = 'lorem'
  where peridno_bk in
  (select peridno_bk from ps_temp.tekosoby limit 100);';
  GET DIAGNOSTICS _row_count = ROW_COUNT;
  RAISE NOTICE 'inc_test_flags_create_changes> zmenil jsem % radku', _row_count;

--test new
FOR i IN 1..100
  LOOP
    INSERT INTO ps_temp.tekosoby
      (
        SELECT
          (
            SELECT max(peridno_bk)
            FROM ps_temp.tekosoby) + 1,
<all other columns> -- jen pro úsporu místa; zde jsou vypsány všechny sloupce

          md5(random() :: TEXT)
        FROM ps_temp.tekosoby
        ORDER BY random()
        LIMIT 1
      );


 END LOOP;
    RAISE NOTICE 'inc_test_flags_create_changes> pridal jsem 100 radku';

  END $body$ LANGUAGE plpgsql VOLATILE COST 100;
```

Výpis 4.1: Testování navržených CDC procedur 1

```
1  CREATE OR REPLACE FUNCTION public.inc_test_flags_find_changes()
2    RETURNS void AS
3  $BODY$
4
5  DECLARE
6    _row_count                    INT;
7
8   execute 'select inc_find_deleted_in_pre_stage(''tekosoby'');'
9    INTO _row_count;
10  RAISE NOTICE
11  'inc_test_flags_find_changes> nalezl jsem % smazanych radku', _row_count;
12
13  execute 'select inc_find_modified_in_pre_stage(''tekosoby'');'
14   INTO _row_count;
15  RAISE NOTICE
16  'inc_test_flags_find_changes> nalezl jsem % zmenenych radku', _row_count;
17
18  execute 'select inc_find_new_in_pre_stage(''tekosoby'');'
19   INTO _row_count;
20  RAISE NOTICE
21  'inc_test_flags_find_changes> nalezl jsem % novych radku', _row_count;
22
23 END
24
25 $BODY$
26   LANGUAGE plpgsql VOLATILE
27   COST 100;
```

Výpis 4.2: Testování navržených CDC procedur 2

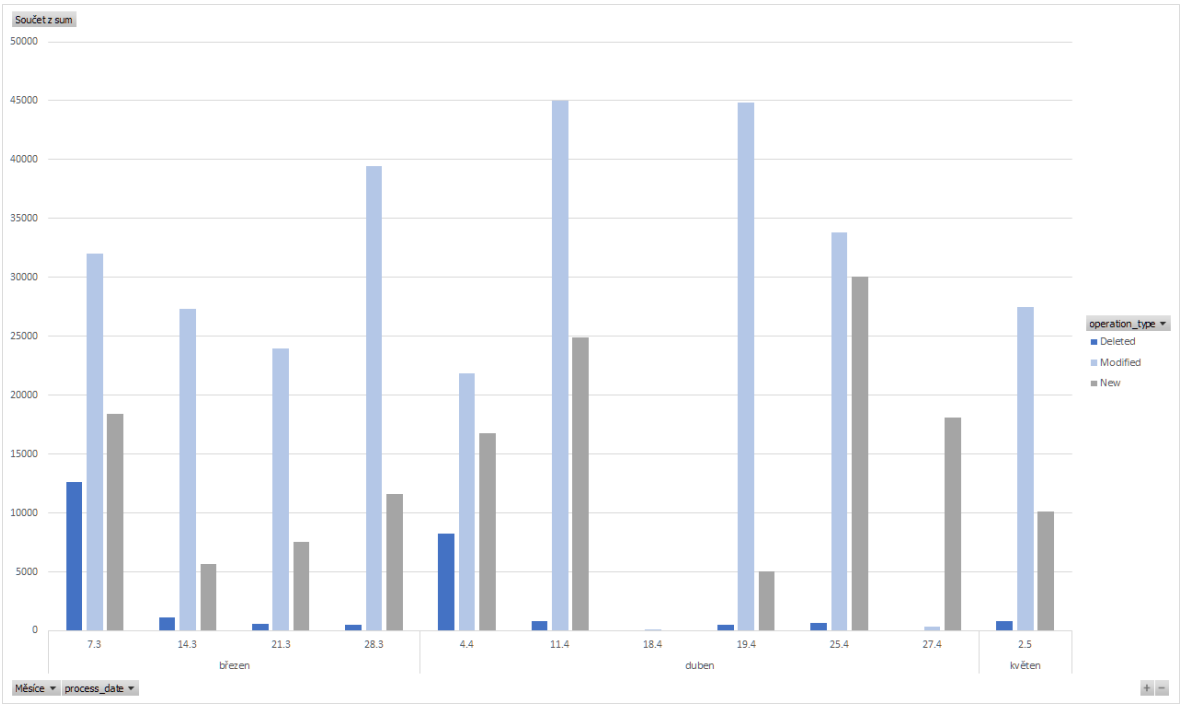
# Výstupní sestava se statistikami tabulky - t_osob_osoba

**Vytvořeno:** 5.5.2017 13:47:40
**Server:**
**Databáze:** dwh3_target
**Schéma:** temp

## Statistiky tabulky

| Statistický údaj | Hodnota |
|---|---|
| Sekvenčně prohledáno | 7 |
| Sekvenčně přečteno n-tic | 336100 |
| Prohledáno indexů | |
| Indexových n-tic načteno | |
| Vloženo n-tic | 200 |
| Aktualizováno n-tic | 200 |
| Smazáno n-tic | 0 |
| HOT aktualizací n-tic | 0 |
| Živých n-tic | 0 |
| Mrtvých n-tic | 1 |
| Bloků hromady z disku | 0 |
| Bloků hromady z mězipaměti | 5707 |
| Indexových bloků z disku | |
| Indexových bloků z mezipaměti | |
| Bloků TOAST z disku | |
| Bloků TOAST z mezipaměti | |
| Indexů TOAST z disku | |
| Indexů TOAST z mezipaměti | |
| Poslední úklid | |
| Poslední automatický úklid | |
| Poslední analýza | |
| Poslední automatická analýza | |
| Počitadlo úklidů | 0 |
| Počitadlo automatických úklidů | 0 |
| Počitadlo analýz | 0 |
| Počitadlo automatických analýz | 0 |
| Velikost tabulky | 23 MB |
| Velikost tabulky TOAST | žádná |
| Velikost indexů | 0 bytes |

Obrázek 4.1: Statistiky tabulky t_osob_osoba




(a) Počet změn dat ve zdrojových systémech v době mezi 7.3.2017 a 2.5.2017


(b) Doba nahrávání dat do datového skladu

Obrázek 4.2: Statistiky plnění datového skladu

# Závěr

Zadáním práce bylo navrhnout a implementovat proces nahrávání dat ze zdrojových systémů do integrované datové vrstvy (IDL) datového skladu ČVUT. Tento úkol se skládal z několika částí.

První částí bylo navržení architektury a procesů Stage area, díky kterým dochází k automatickému rozpoznávání změn dat ve zdrojových systémech a následné technické „čištění“ těchto dat.

Druhou částí bylo navržení historizační procedury, která dokáže zpracovat tato změněná a označená data. S druhou částí také souvisí úprava stávajících ETL procesů, které jsou zodpovědné za nahrání dat ze Stage area do IDL. Všechny ETL procesy jsem upravil a vytvořil jsem šablonu a postup pro tvorbu nových ETL, aby odpovídaly novému standardu.

Třetí částí bylo navržení automatického spouštění těchto ETL procesů a jejich monitoring. Ke spouštění jsem použil shell skript, který je spouštěn démonem Cron na ETL serveru. Pokud při běhu ETL procesů nastane nějaká chyba, je odeslán informační e-mail administrátorům datového skladu. Pokud je tato chyba známá a je znám způsob jejího odstranění, je postup opravy této chyby přiložen v chybovém e-mailu. Pokud žádná chyba nenastane, je odeslán e-mail obsahující log transformace.

Jako ukázku funkcionality mé práce jsem přiložil statistiky nahrávání dat a statistiky změn dat ve zdrojových systémech. Jelikož jsem upravil již fungující a otestované ETL procesy, není nutné kontrolovat jejich funkčnost.

Zadání a cíle této práce se mi podařilo splnit. Největším přínosem této práce je plně automatizované nahrávání datového skladu ČVUT a zrychlení celého procesu nahrávání dat do IDL. Praktická část této práce je již nasazena v produkčním prostředí datového skladu ČVUT a poskytuje tak data pro jejich další analytické zpracování.

# Literatura


[1] Full, incremental or differential: How to choose the correct backup type. 2008. Dostupné z: `http://searchdatabackup.techtarget.com/feature/Full-incremental-or-differential-How-to-choose-the-correct-backup-type`

[2] Novotný, O.; Pour, J.; Slánský, D.: *Business inteligence*. Praha: Grada, první vydání, 2005, ISBN 8024710943.

[3] Matouk, K.; Owoc, M. L.; aj.: A survey of data warehouse architectures—Preliminary results. In *Computer Science and Information Systems (FedCSIS), 2012 Federated Conference on*, IEEE, 2012, s. 1121–1126.

[4] Inmon, W. H.: *Building the data warehouse*. New York: Wiley Computer Pub., druhé vydání, 1996, ISBN 0471141615.

[5] Kimball, R.: *The data warehouse toolkit: practical techniques for building dimensional data warehouses*. Wiley, 1997.

[6] Arnošt, D.: Kontext BI v rámci komerční firmy, Framework architektury BI a BI architektura. přednášky MI-EDW.16, ČVUT FIT, 2017.

[7] BIG DATA MAGIC: CREATING GOLD FROM NOTHING. 2013. Dostupné z: `http://www.attunity.com/blog/big-data-magic-creating-gold-nothing`

[8] Kimball, R.; Ross, M.; Becker, B.; aj.: *The Kimball Group reader*. Second edition vydání, ISBN 1119216591.

[9] Mundy, J.: Design Tip #158 Making Sense of the Semantic Layer. 2013, [cit. 2017-03-25]. Dostupné z: `http://www.kimballgroup.com/2013/08/design-tip-158-making-sense-of-the-semantic-layer/`

[10] Laberge, R.: *Datové sklady*. Brno: Computer Press, první vydání, 2012, ISBN 9788025137291.

[11] Inmon, W. H.: *Building the data warehouse*. Indianapolis, Ind.: Wiley, čtvrté vydání, c2005, ISBN 9780764599446.

[12] Kimball, R.: *The data warehouse lifecycle toolkit*. New York: Wiley, c1998, ISBN 0471255475.

[13] Kotlář, R.: *Historizace dat pro potřeby datového skladu fakulty*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2015.

[14] Data Integration Architectures for Operational Data Warehousing. Technická zpráva, Oracle, Redwood Shores, CA 94065, 2012.

[15] KIMBALL, R.; ROSS, M.: *The data warehouse toolkit : the complete guide to dimensional modeling*. Robert Ipsen, 2002, ISBN 0-471-20024-7.

[16] Data Warehouse Schema Architecture. 2008, [cit. 2017-03-25]. Dostupné z: `http://datawarehouse4u.info/Data-Warehouse-Schema-Architecture.html`

[17] Yost, K.; Saylor, M.; Wilding, P.; aj.: System and method for management of an automatic OLAP report broadcast system. Říjen 25 2016, uS Patent 9,477,740. Dostupné z: `https://www.google.com/patents/US9477740`

[18] Patrný, V.: Master data management. *IT Systems*, ročník 2011, č. 11, 2011: s. 50–52.

[19] Ulrych, J.: Data Quality Management. *IT Systems*, ročník 2016, č. 12, 2016: s. 28–31.

[20] Krejčí, J.: *Návrh datových vrstev pro datový sklad ČVUT*. Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2017.

[21] Lenger, R.: *ETL server pro potreby datového skladu fakulty*. Bakalářská práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2015.

Příloha A

# Seznam použitých zkratek

**BI** Business intelligence

**ETL** Extract transform load

**CDC** Changing Data Capture

**ČVUT** České Vysoké Učení Technické

**DDL** Data Definition Language

**DQM** Data Quality Management

**DWH** Data Warehouse

**ELT** Extract Load Transform

**ETL** Extract Transform Load

**ETLT** Extract Transform Load Transform

**FIT** Fakulta Informačních Technologí

**MOLAP** Multidimensional Online Analytical Processing

**OLAP** Online Analytical Processing

**ROLAP** Relational Online Analytical Processing

**HOLAP** Hybrid Online Analytical Processing

**IDL** Integrated Data Layer

**MDM** Master Data Management

**ODS** Operational Data Source

**PDI** Pentaho Data Integration

**ROI** Return of Investment

**SCD** Slowly Changing Dimension

**SLA** Service-level Agreement

**SQL** Structured Query Language

**URL** Uniform Resource Locator

**VPN** Virtual Private Network

Příloha B

# Obsah přiloženého CD

- `src`
  - `ETL_agent` ..... zdrojové kódy automatizace nahrávání
  - `historizace` ..... zdrojové kódy historizačních procedur
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF