Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Agregace otevřených dat relační databází

Lukáš Jirůšek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Lukáš Jirůšek

Informatika

Název práce

Agregace otevřených dat relační databází

Název anglicky

Open data aggregation using relational database

Cíle práce

Bakalářská práce je tematicky zaměřená na problematiku zvýšení využitelnosti otevřených dat v manažerské praxi prostřednictvím agregačních operací relačně databázové technologie.
Náplní a účelem této práce je:
• vymezit problematiku otevřených dat a s tím souvisejících prostředků a technik relačně databázové technologie umožňujících zvýšit jejich vypovídací schopnost,
• popsat jednotlivé agregační funkce v rámci této technologie,
• navrhnout odpovídající relační databázi a následně ji naplnit relevantními daty z portálu otevřených dat,
• navrhnout a ověřit konkrétní agregační operace v SQL a zhodnotit jejich využitelnost v existující manažerské praxi,
• zobecnit přínosy navrhovaných záležitostí pro další možná použití.

Metodika

Použitá metodika zadané bakalářské práce bude založena na studiu a analýze dostupných informačních zdrojů a případných existujících řešení v dané oblasti. Stěžejními metodami této práce budou metody a techniky relačně databázové technologie a SQL. Navrhované řešení bude zohledňovat identifikované požadavky a očekávání spojená s řešenou záležitostí. Na podkladě syntézy teoretických poznatků a dosažených výsledků budou formulovány závěry této bakalářské práce a následně zobecněny pro další možná použití.

Harmonogram práce:

Vymezení teoretických principů řešené problematiky, literární rešerše – do 5.9.2021: předmět 1. zápočtu z BP,

Zmapování současné situace řešené problematiky a navržení odpovídajícího řešení – do 10. 1. 2022,

Ověření navrženého řešení – do 15.2.2022: předmět 2. zápočtu z BP

Zobecnění navrhovaných záležitostí – do 10.3.2022: předmět 3. zápočtu z BP.

Doporučený rozsah práce

45-55 stran

Klíčová slova

open data, agregace dat, SQL, datová normalizace a integrita

Doporučené zdroje informací

HARRINGTON, Jan L. Relational database design and implementation. Morgan Kaufmann, 2016. ISBN 9780128043998

MELTON, Jim; SIMON, Alan R. SQL: 1999: understanding relational language components. Elsevier, 2001. ISBN 978-1558604568

MELTON, Jim; SIMON, Alan R. Understanding the new SQL: a complete guide. Morgan Kaufmann, 1993. ISBN 9781558602458

OWENS, Mike. The definitive guide to SQLite. Apress, 2006. ISBN 9781590596739

VALENTA, M., POKORNÝ, J. Databázové systémy. Praha: České vysoké učení technické v Praze, 2020. ISBN 978-80-01-06696-6

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

doc. Dr. Ing. Václav Vostrovský

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 11. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 14. 02. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Agregace otevřených dat relační databází" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne _____

Poděkování

Rád(a) bych touto cestou poděkoval doc. Ing. Václavovi Vostrovskému Ph.D. za odborné vedení této práce a za cenné návrhy a rady.

Agregace otevřených dat relační databází

Abstrakt

Tato bakalářská práce se zabývá agregací dat z Portálu otevřených dat České republiky využitím relačních databází, jazyka SQL a programovacího jazyka Python.

V první části této práce je čtenář seznámen s otevřenými daty, jaká data to jsou, jak jsou hodnocena a jaké mají výhody a nevýhody. Rovněž je seznámen s relačními databázemi a je mu popsána problematika normalizace dat v těchto databázích, nakonec je seznámen s jazykem SQL a především s agregačními příkazy, který tento jazyk nabízí.

Cílem praktické části je poté využít těchto poznatků k navržení a vytvoření vhodné relační databáze, která bude dále naplněna vhodně upravenými daty z Portálu otevřených dat, konkrétně bude tato databáze shromažďovat data o počtu cizinců v České republice mezi lety 2004 a 2020. Po naplnění databáze těmito daty bude vytvořeno několik agregačních dotazů v jazyce SQL, které tyto data kvantifikují podle různých parametrů, např. podle občanství cizinců anebo podle místa jejich bydliště. Poslední cílem praktické části bude vytvoření programu v jazyce Python, který tuto databázi a vytvořené SQL dotazy využije pro snadné vytvoření vlastních dotazů na tuto databázi, aby i technicky méně znalý člověk mohl tato data jednoduše využít.

Klíčová slova

Otevřená data, portál otevřených dat, cizinci v ČR, relační databáze, datové modelování, jazyk SQL, agregace dat, SQLite, Python

Open data aggregation using relational database

Abstract

This bachelor thesis focuses on aggregating data from Open data portal of the Czech Republic, using relational databases, SQL queries and programming language Python.

In first part of the theses, the reader is familiarized with open data, what kind of data it is, how are they ranked and what are their advantages and disadvantages. Reader is also familiarized with relational databases, and with problematics of database normalization. Then he is familiarized with SQL language, especially with aggregation functions that the language offers.

The main point of practical part is to use this knowledge to create suitable relational database, which will be filled with edited data from Open data portal of the Czech Republic, in particular with the data about the number of foreigners in the Czech Republic between years 2004 and 2020. Once the database is filled with this data, the next step will be to create several SQL aggregation queries, which will filter and accumulate data based on several parameters, for example by country of origin or by destination they live in. The last point of the practical part will be to create Python program that will use the database and the SQL queries to allow user of the program to easily create their own SQL queried based on their chosen parameters, even if the user does not have advanced technical capabilities.

Keywords

Open data, open data portal, foreigners in Czech Republic, relational database, data modelling, SQL language, data aggregation, SQLite, Python

Obsah

1. Úvod

V současné době jsou data klíčovým prvkem téměř každého systému, úřady České republiky ale i jiných státu shromažďují data o svých občanech i počtu cizinců, o výsledcích voleb, výsledcích ekonomiky a o mnoha jiných údajích. Soukromé firmy rovněž sbírají mnoho dat důležitých k jejich provozu, data mohou být také ale vytvářena během různých vědeckých pokusů, pozorování a měření.

Objem nashromážděných dat se s digitalizací těchto systémů mnohonásobně zvýšil, především díky snadnému ukládání dat, snadné manipulaci s daty a díky potencionálnímu užitku, která tato data mohou pro daný systém přinést.

Tato nashromážděná data ale nemusí být užitečná jen pro daný systém, který data nasbíral, ale i pro jiné organizace, firmy, ale i jednotlivce. Z tohoto důvodu vznikají různé iniciativy, aby tyto systémy, především pak vládní organizace a úřady, publikovaly nashromážděná data veřejnosti pro další využití.

Z tohoto důvodu vznikl v roce 2018 Portál otevřených dat, kde mohou ministerstva, obce, či jiné státní orgány publikovat nasbíraná data v podobě otevřených dat.

Otevřená data jsou data volně dostupná ke stažení, důraz je kladen na jejich kvalitu a snadnou strojovou čitelnost, které je docíleno využitím správného formátu souboru ve kterém jsou data sdílena, velmi důležitá je nejen úplnost a správnost dat, ale také možnost tato data dále zdarma šířit a nebo je využít komerčně. Otevřená data by také měla být opatřena dokumentací vysvětlující organizaci a strukturu dat. Otevřená data mohou být například telefonní seznamy, evidence psů, katalogy knihoven, data o vývoji naděje dožití, o počtu obyvatel a jejich pohybu, o hrubé mzdě atd.

Otevřená data pak mohou být využita podniky a společnostmi, ale i širokou veřejností, mezi zajímavé a užitečné projekty využívající otevřená data patří například projekt Mapy bez bariér mapující přístupnost objektů v ČR pro handicapované anebo projekt Hlídač státu monitorující smlouvy z registru smluv a státní zakázky s cílem zvýšení transparentnosti státní správy.

Otevřená data jsou ale často velmi detailní, což znesnadňuje, občas i znemožňuje tyto data v neupravené formě prakticky využít. Pokud bychom jako příklad měli data o počtu rodinných příslušníků v rodinných domech, můžeme v datech najít kolik lidí žije v jednom daném domě, což je informace, která sama o sobě žádný informační přínos nemá, informace se ale stává přínosnou v kontextu s jinými daty, ve zmíněném příkladu by se tedy dalo zjistit jaký je průměrný počet lidí v rodinném domě pro daný okres anebo kraj, tyto výsledky by se daly porovnávat a dalo by se pomocí nich dojít k jistým ekonomickým či sociálním závěrům.

Proto je velmi důležité či dokonce nezbytné tato data agregovat, agregací dat lze zjistit různé výsledky o daném souboru dat, lze díky nim snadněji tvořit o datech různé závěry a tyto výsledky lze pak dále využít. Agregace dat tedy zvyšuje informativní hodnotu daných dat, čehož může být využito manažery či jinými řídícími pracovníky pro vytvoření správného rozhodnutí.

Možnost agregace otevřených dat bude poté demonstrována v praktické části této práce, ve které bude probíhat agregace vybraného typu dat z portálu otevřených dat ČR. Toho bude dosaženo navrhnutím vhodné relační databáze pro tato data, která budou nejdříve upravena, do databáze pak budou posílány vhodně vytvořené SQL dotazy, které budou mít za cíl data v databázi agregovat do užitečnějších výsledků.

Konktrétně budou v praktické části agregována data o počtu cizinců v ČR, která byla vybrána pro svoji aktuálnost nejenom v České republice, ale v celé Evropě, kde se neustále mění počet cizinců, ať už se jedná jen o studenty, uprchlíky anebo ekonomické migranty. V roce 2015 vyvrcholila evropská migrační krize, v tomto roce zažádalo více než milion uprchlíků o azyl ve státech EU, v roce 2022 pak způsobila ruská invaze na Ukrajinu uprchlickou vlnu z Ukrajiny do států EU.

Samotná databáze a agregace dat bude probíhat využitím databázového systému SQLite a programovacího jazyka Python, agregované výsledky o počtu cizinců v ČR pak mohou sloužit například pracovníkům českých i zahraničních ministerstev, ale i starostům českých měst ke zjištění počtu cizinců v jejich městě, výsledky této práce ale mohou být využity pro statistické účely, podobně jako jiná data poskytovaná Českým statistickým úřadem.

2. Cíl práce a metodika

2.1. Cíl práce

Hlavním cílem této bakalářské práce je vytvoření relační databáze ke zvolené sadě otevřených dat, konktrétně k datům o počtu cizinců v České republice a vytvoření SQL dotazů, které budou data v této databázi agregovat do jednodušších výsledků.

Dílčí cíle této práce jsou:

- Vymezit teoretická východiska související s problematikou otevřených dat, relačních databází a jazyka SQL.
- Vytvořit script pro automatickou úpravu stažených dat z portálu otevřených dat do vhodného formátu.
- Vytvořit program pro interaktivní vytvoření dotazů pro tuto databázi.
- Zmapování současné situace dané problematiky a ověření funkčnosti navrženého řešení

2.2. Metodika

Tato práce se opírá o teoretická východiska a čerpá jak z odborné literatury týkající se dané problematiky, tak ze znalostí autora práce získaných během studia, tyto znalosti budou poté využity při tvorbě vlastní práce.

Vlastní prací pak bude vytvoření relační databáze, pro kterou bude využit databázový systém SQLite. Pro tuto databázi bude vytvořen takový relační model, který splňuje běžné normální formy, a to tak aby tento model odpovídal struktuře dat o počtu dat cizinců v ČR z Portálu otevřených dat České republiky. Tato data pak budou stažena a za pomocí programovacího jazyka Python vhodně upravena a převedena do SQL insertion scriptu pro snadné vložení dat do vytvořené databáze. Poté budou vytvořeny SQL dotazy, které budou data agregovat do kvantitativních hodnot, posledním krokem bude využití Pythonu pro vytvoření programu pro interaktivní dotazování této databáze.

Ve vlastní práci bude využita metoda modelování a implementace, nejdříve bude celá databáze navržena v podobě relačního modelu, poté bude tato databáze i implementována pomocí databázového systému SQLite.

Dále budou představeny již existující řešení zabývající se agregací dat o počtu cizinců, ty pak budou použity pro ověření správnosti navrženého řešení.

3. Teoretická východiska

3.1. Otevřená data

Otevřená data (anglicky „open data“) jsou taková data, která mohou být kýmkoli volně používána a redistribuována. Musí k nim být snadný přístup, nejlépe jsou k dispozici ke stažení zdarma přes internet bez jakýchkoli překážek jako je např. potřeba registrace anebo limit stahování. Měla by také být snadno zpracovatelná počítačem, a to pomocí využití běžně dostupných formátů, mezi které patří například formát CSV, XML anebo JSON. [1]

3.1.1. Škála otevřenosti dat

V roce 2010 definoval Tim Berners-Lee, jenž je také známý jako zakladatel internetu, 5hvězdičkovou škálu pro hodnocení otevřenosti dat, a to od nejuzavřenějšího po nejotevřenější. Podmínky pro tuto škálu jsou kumulativní (pro získání pěti hvězd musí data splňovat všechny podmínky pro 4hvězdičková data atd.). Zde jsou vyjmenovány jednotlivé podmínky na této škále: [2]

- 1 hvězda – data jsou dostupná v jakémkoli formátu
- 2 hvězdy – data jsou čitelná počítačem (nepatří sem třeba fotografie s daty)
- 3 hvězdy – data jsou dostupná v neproprietárním formátu (nepatří sem třeba soubory vytvořené v Excelu)
- 4 hvězdy – data využívají neproprietární specifikace od konsorcia W3C
- 5 hvězdy – data jsou systematicky propojená s jinými daty

Tim Berners-Lee pak v roce 2012 založil spolu s Nigelem Shadboltem neziskovou organizaci Open Data Institute (česky institut pro otevřená data) sídlící ve Velké Británii, kde je dodnes zaměstnán jako předseda. [3] Tato organizace spolupracuje jak s vládami, tak se soukromými společnostmi za účelem zlepšení kvality a zajištění volnosti otevřených dat.

3.1.2. Publikování otevřených dat

Otevřená data jsou většinou publikována složkami veřejné správy, mohou ale být publikována i soukromými společnostmi anebo jednotlivci. Tyto datové sady jsou pak shromažďovány v takzvaných portálech otevřených dat.

V České republice se tento portál nazývá „Portál otevřených dat“ a je k dispozici na webové stránce https://data.gov.cz/. Tento portál funguje od roku 2018 [4] a je provozován Ministerstvem vnitra. V současné době disponuje téměř 150 tisíci datovými sadami od více než 200 různých poskytovatelů [5], mezi něž se řadí mnoho ministerstev, úřadů, měst anebo také Český statistický úřad.

Mezi další portály s otevřenými daty se řadí například https://data.europa.eu/en shromažďující data pro celou Evropu obsahující téměř 1,4 milionu datových setů [6] anebo https://www.data.gov/ obsahující přes 300 tisíc datových sad ze Spojených států amerických. K hledání datových setů je také možné použít „Dataset search“ od společnosti Google na adrese https://datasetsearch.research.google.com/, která dohromady obsahuje přes 30 milionů datových sad z více než 4600 domén. [7]

3.1.3. Problémy otevřených dat

Otevřená data mohou být znehodnocena z několika různými důvody, mezi ně patří například omezený přístup anebo příliš restriktivní licenční podmínky omezující skutečné využití těchto dat. Problémem je také vzdání se odpovědnosti za korektnost dat, což může znamenat nesprávnost dat.

Dalším problémem jsou pak chyby při špatném využívání jednotlivých formátů, pro formát CSV to může být například použití špatného oddělovače, špatný zápis číslic anebo přebývající mezery. [8]

3.2. Relační databáze

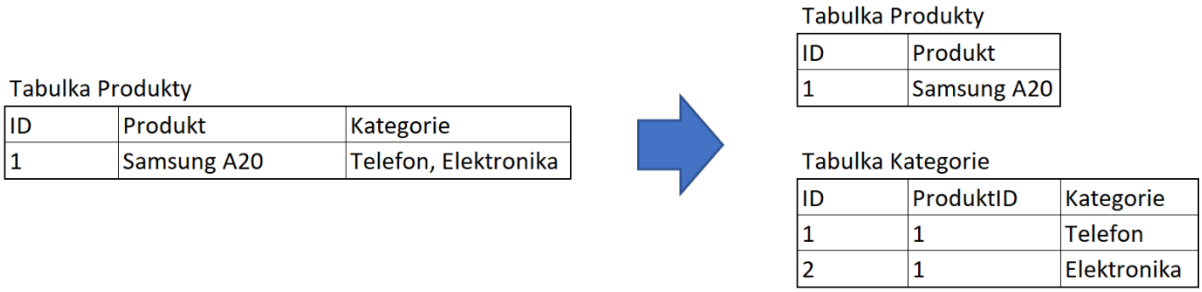
Relační databáze je založena na relačním modelu, což je způsob ukládání dat využívající relace – tabulky. Tabulka je struktura záznamů, každý záznam tvoří jeden řádek v této tabulce, sloupce tabulky pak tvoří atributy každého záznamu. Tyto sloupce mají určitý datový typ, mezi atributy těchto tabulek také často patří klíče odkazující na záznamy v jiných tabulkách (foreign key) a unikátní klíče identifikující daný záznam v tabulce (primary key). Kolekce těchto tabulek a vazby mezi nimi realizované klíči pak tvoří relační model. [9]

3.2.1. Normalizace databáze

Normalizace databáze označuje proces pozměňovaní relačního modelu tak, aby bylo plně využito jeho výhod a snížil se počet duplicitních dat v databázi. To vede ke zvýšení efektivity ukládání, hledání a zpracování dat. Existuje několik normálních forem, za normalizovanou databázi se považuje taková databáze, která splňuje alespoň první tři normální formy. [10]

1. Normální forma

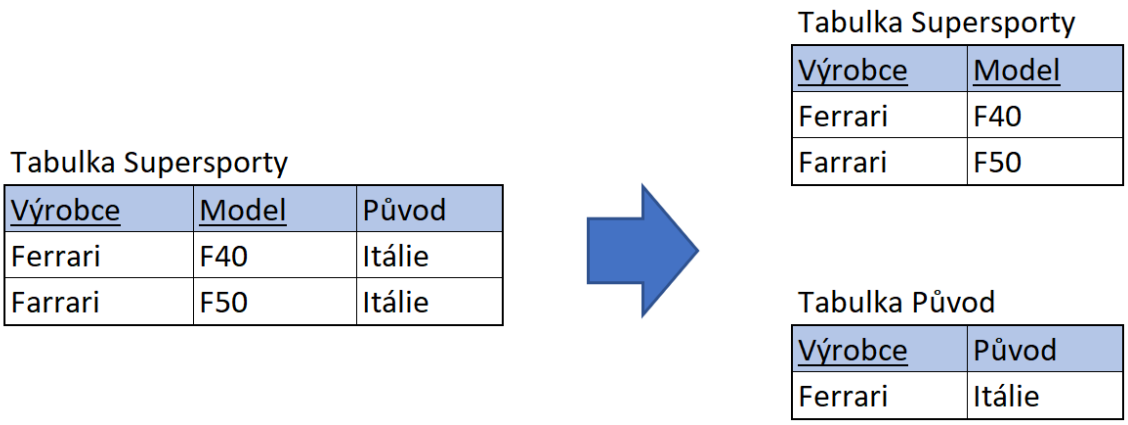
1. Normální forma zaručuje že všechny atributy databáze jsou atomické (nedělitelné) [11], v následujícím příkladu rozdělíme v tabulce Produkty atributu Kategorie, která může obsahovat více hodnot na dvě samostatné tabulky, Produkty a Kategorie. Tabulka kategorie pak obsahuje cizí klíč ProduktID, který tyto kategorie váže zpět k danému produktu.



Obrázek 1 - Normalizace tabulky pro splnění 1. NF (vlastní tvorba)

2. Normální forma

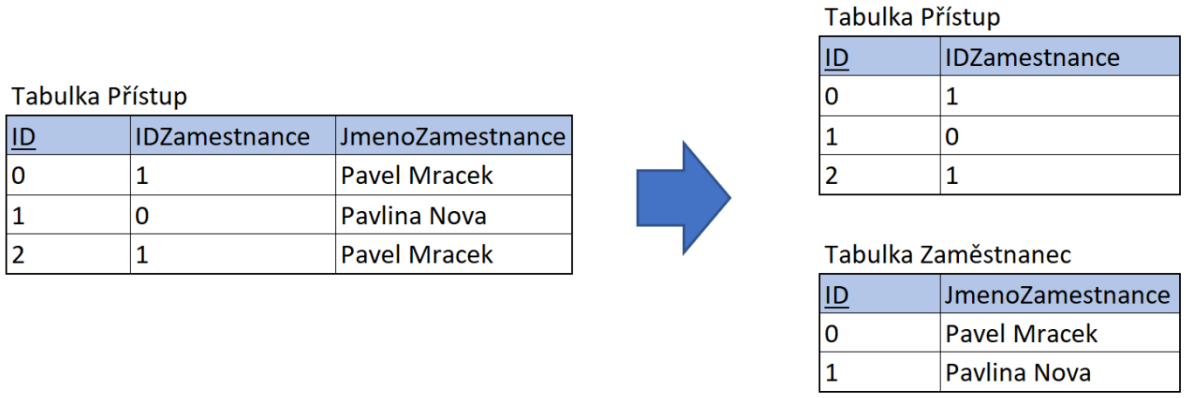
2. Normální forma říká že všechny atributy tabulky jsou závislé na primárním klíči. Pokud je primární klíč tabulky složen z více atributů a ke zjištění nějaké atributy stačí jen jeden klíč, jedná se o porušení této formy. [23] V následující tabulce je supersport identifikován kombinací Výrobce a Modelu, atributa Původ nezávisí na modelu, ale jen na výrobci. Tímto je porušena druhá normální forma. Normalizaci lze provést oddělením země původu do samostatné tabulky, jinou možností by bylo dát atributu Model do samostatné tabulky.



Obrázek 2 - Normalizace tabulky pro splnění 2. NF (vlastní tvorba)

3. Normální forma

3. Normální forma zaručuje že tabulka neobsahuje žádné tranzitivní závislosti. [12] Na příkladu vidíme že v tabulce Přístup je ukládána atributa JmenoZamestnance, i přesto že tento údaj lze vyčíst z ID zaměstnance. Rozdělením tabulky na dvě tabulky – Přístup a Zaměstnanec je tento problém eliminován.



Obrázek 3 - Normalizace tabulky pro splnění 3. NF (vlastní tvorba)

3.3. Jazyk SQL

Jazyk SQL (Structured Query Language) je jazyk určený pro správu a organizaci dat v relačních databázích. Byl vyvinut v 70. letech 20. století americkou společností IBM, největší podíl na vývoji toho jazyka měli informatici Donald Chamberlin a Raymond Boyce. Na konci tohoto desetiletí se ale k vývoji přidala i firma Relational Software, Inc. (dnes Oracle), která své databáze zpřístupnila i jiným společnostem a mimo to i americké vládě. [13] S rostoucí popularitou relačních databází vznikla snaha standardizovat jejich jazyk, o tuto standardizaci se pokoušel Americký národní standardizační institut (ANSI), namísto nového standardu byl za standard přijat právě jazyk SQL, jenž byl v následujících letech osmkrát aktualizován, naposledy v roce 2016. Tyto revize přidaly nové funkcionality, např. možnost definování vlastních datových typů, nebo podporu pro formáty XML a JSON. [14]

Jazyk SQL se řadí mezi dotazovací jazyky, které jsou oproti programovacím jazykům většinou snazší a bližší mluvené řeči. Zařazení jazyka mezi dotazovací jazyky je ale trochu zavádějící, jazyk SQL umožnuje také tabulky definovat a měnit, ale také jí naplnit daty.

3.3.1. Datové typy jazyka SQL

Datové typy slouží k uchovávání dat v určitém, předem definovaném formátu. Například atribut věk bude uložen v nějakém celočíselném typu (integer), cena bude uložená v čísle s pohyblivou řádovou čárkou (float) a text, kterým může být například jméno bude uložen v textovém řetězci (string). [15]

Tyto datové typy jsou ale v různých databázích realizovány různě a jinak se jmenují (v MySQL je celočíselný typ pojmenován INTEGER, v databázi ORACLE se pak jedná o typ NUMBER), dokonce je běžné že některé databáze nějaký datový typ nemají (databáze SQLite nemá dedikovaný typ pro datum, datum je pak ukládáno jako text ve formátu „YYYY-MM-DD HH:MM:SS.SSS“).

Kromě typů definovaných v samotném softwaru databáze je možné definovat svoje vlastní typy uživatelem. [16]

3.3.2. Dotazy jazyka SQL

Příkazy, pomocí kterých pracujeme s databází se v jazyce SQL nazývají dotazy, ty se podle svých funkcionalit dělí na následující kategorie: [13]

DDL příkazy (data definition language)

Sem patří příkazy vytvářející struktury databáze (tabulky, pohledy atd.), mezi ně patří příkazy **CREATE**, **DROP** a **ALTER**. [24] Jako příklad poslouží následující dotaz, který vytvoří tabulku pro zákazníky, ta bude obsahovat identifikační číslo, jméno, příjmení a věk zákazníka.

```
CREATE TABLE Zakaznici (
    ID number,
    Jmeno varchar(255),
    Prijmeni varchar(255)
);
```

Tato tabulka pak může být příkazem **ALTER** pozměněna tak aby tabulka se zákazníky obsahovala i atributu věk.

```
ALTER TABLE Zakaznici ADD Vek number;
```

DML příkazy (data manipulation language)

Tyto příkazy slouží pro získání, úpravu a mazání dat z databází, patří sem například příkazy **SELECT**, **INSERT**, **UPDATE** a **DELETE**. [25] Série následujících dotazů přidá do výše vytvořené tabulky 3 zákazníky s využitím příkazu **INSERT**:

```
INSERT INTO Zakaznici VALUES (0, 'Pavel', 'Novak', 24);
INSERT INTO Zakaznici VALUES (1, 'Hana', 'Vesela', 13);
INSERT INTO Zakaznici VALUES (2, 'Karel', 'Cech', 26);
```

Příkazem **UPDATE** pak můžeme změnit věk uživatele s využitím jeho ID:

```
UPDATE Zakaznici
SET Vek = 25
WHERE ID = 0;
```

Příkaz **SELECT** můžeme použít pro vypsání všech uživatelů, nebo jen těch, kteří splňují nějaký predikát. Tohoto vyfiltrování lze dosáhnout pomocí zapsání podmínky za klauzuli **WHERE**, v následujícím případě jsou vyhledání všichni dospělí zákazníci:

```
SELECT Jmeno, Prijmeni, Vek
FROM Zakaznici
WHERE Vek >= 18;
```

A nakonec lze pomocí příkazu **DELETE** uživatele z tabulky smazat, pro ukázku z této tabulky budou odstraněni nezletilí zákazníci:

```
DELETE FROM Zakaznici
WHERE Vek < 18;
```

DCL příkazy (data control language)

Tyto příkazy slouží k nastavování přístupových práv a k řízení transakcí. Databázová transakce symbolizuje změnu databáze z jednoho stavu do druhého, obvykle po nějakých změnách v databázi způsobené výše vyjmenovanými příkazy. Mezi příkazy databázové transakce patří příkazy **BEGIN**, jenž zahájí transakci, **COMMIT**, který slouží k potvrzení transakce, což vede k uložení změn v databázi a příkaz **ROLLBACK**, který transakci zruší, kvůli čemu zůstane databáze ve stavu před započetím transakce. [13]

Databázová transakce musí splňovat 4 podmínky, ty jsou často označovány akronymem ACID – atomicity, consistency, isolation, durability, v češtině se jedná o podmínky atomicita, konzistence, izolovanost a trvalost. Atomicita zajišťuje že se daná transakce buď provede celá anebo vůbec, aby v databázi nedošlo k nedefinovanému mezistavu. Při transakci s více příkazy způsobí chyba v jednom příkazu rollback (vrácení) celé transakce. Konzistence poté zajišťuje dodržení všech integritních omezení v databázi, například to že věk uživatele nemůže být záporné číslo. Izolovanost ručí za to, že příkazy v jedné transakci neovlivní transakci jinou, trvalost je pak vlastností garantující uložení všech změn po dokončené transakci, i v případě selhání systému. Toho je dosaženo uložením změn v databázi po každé transakci na disk. [17]

Příkazy sloužící k nastavování práv jsou příkazy **GRANT**, jenž přidělí uživatelům nějaké právo a příkaz **REVOKE**, který tato práva naopak od uživatelů odebírá.

3.3.3. Agregační funkce

Agregační funkce jsou statistické funkce, které vracejí jediný, většinou numerický výsledek vypočítaný z několika řádků v tabulce anebo z tabulky celé. [18]

Pro snadnější popsání těchto funkcí bude využita následující tabulka se zbožím, na které budou prováděny různé agregační funkce.

ID	Výrobce	Cena	Typ
0	Samsung	4500	mobil
1	Honor	5600	mobil
2	LG	2300	mobil
3	Apple	15560	mobil
4	LG	10360	TV
5	Sony	12500	TV
6	Samsung	15000	TV
7	Toshiba	3000	TV

Tabulka 1 - Vzorová data pro testování agragačních dotazů (vlastní tvorba)

Tato tabulka a její data je v SQL vytvořena následující sekvencí SQL příkazů:

```
CREATE TABLE Zbozi(
   ID        number,
   Vyrobce   varchar(7),
   Cena      number,
   Typ       varchar(5)
);

INSERT INTO Zbozi VALUES (0,'Samsung',4500,'mobil');
INSERT INTO Zbozi VALUES (1,'Honor',5600,'mobil');
INSERT INTO Zbozi VALUES (2,'LG',2300,'mobil');
INSERT INTO Zbozi VALUES (3,'Apple',15560,'mobil');
INSERT INTO Zbozi VALUES (4,'LG',10360,'TV');
INSERT INTO Zbozi VALUES (5,'Sony',12500,'TV');
INSERT INTO Zbozi VALUES (6,'Samsung',15000,'TV');
INSERT INTO Zbozi VALUES (7,'Toshiba',3000,'TV');
```

Agregační funkce **COUNT()** vrátí celkový počet řádků splňující nějakou podmínku, pokud není specifikovaná, vrátí počet řádků v celé tabulce. [19] Jelikož zde nezáleží, v jakém sloupci řádky počítáme, můžeme jako argument této funkce použít *, což je symbol označující všechny sloupce tabulky. Pokud chceme spočítat celkový počet unikátních hodnot v daném sloupci, využijeme klíčové slovo **DISTINCT** spolu s daným sloupcem jako argument. První příkaz v následujícím kódu spočítá celkový počet zboží v tabulce, druhý pak počet zboží s cenou nad 10000, poslední příkaz počet typů prodávaných zařízení (mobil a TV):

```
SELECT COUNT(*)
FROM Zbozi;
-- vrati vysledek 8

SELECT COUNT(*)
FROM Zbozi
WHERE Cena > 10000;
-- vrati vysledek 4

SELECT COUNT(DISTINCT Typ)
FROM Zbozi;
-- vrati vysledek 2
```

Funkce **MIN()**, **MAX()** jsou velmi jednoduché a vracejí nejmenší / největší hodnotu v daném sloupci, pokud daný sloupec obsahuje text, bude vrácen text s nejnižší / nejvyšší hodnotou podle lexikografického uspořádání. [19] V následujícím kódu je spočítána cena nejdražšího mobilu a výrobce s nejnižší hodnotou při uspořádání podle abecedy:

```
SELECT MAX(Cena)
FROM Zbozi
WHERE Typ = 'mobil';
-- vrati 15560

SELECT MIN(Vyrobce)
FROM Zbozi;
-- vrati retezec Apple
```

Funkce **AVG()** vrátí průměrnou hodnotu v daném sloupci, funkce **SUM()** pak zase sumu daných hodnot, [19] v této ukázce je spočítána průměrná cena televize a celková cena produktů od výrobce LG.

```
SELECT AVG(Cena)
FROM Zbozi
WHERE Typ = 'TV';
-- vrati 10215

SELECT AVG(Cena)
FROM Zbozi
WHERE Vyrobce = 'LG';
-- vrati 12660
```

Výše zmíněné agregační funkce prozatím vracely pouze jeden výsledek pro vybrané řádky, je ale možné je využít i na skupiny dat z dané tabulky, například je možné najít průměrnou cenu každého typu zboží v tabulce. Toto seskupení je v SQL umožněno klauzulí **GROUP BY** (seskup podle). [20] Tato klauzule seskupí stejné hodnoty v daném sloupci a je pak možné data agregovat na těchto skupinách. Následující příkazy spočítají průměrnou cenu všech typů zboží a nejnižší cenu produktu od jakéhokoli výrobce:

```
SELECT Typ, AVG(Cena) AS "Prumerna cena"
FROM Zbozi
GROUP BY Typ;

SELECT Vyrobce, MIN(Cena)
FROM Zbozi
GROUP BY Vyrobce;
```

Výstup prvního příkazu:

TYP	Průměrná cena
Mobil	6990
TV	10215

Tabulka 2 - Výstup dotazu s GROUP BY (vlastní tvorba)

Výstup druhého příkazu:

VYROBCE	MIN(CENA)
LG	2300
Sony	12500
Honor	5600
Samsung	4500
Toshiba	3000
Apple	15560

Tabulka 3 - Výstup 2. dotazu s GROUP BY (vlastní tvorba)

Dotaz obsahující klauzuli **GROUP BY** lze ještě obohatit o klauzuli **HAVING**, která podobně jako klauzule **WHERE** umožňuje vyfiltrovat data, na rozdíl od klauzule **WHERE** ale používáme v predikátu agregační funkci, což nám **WHERE** neumožňuje. [20] V tomto příkladu je zobrazena průměrná cena zboží všech výrobců pouze pokud tento průměr přesáhne hodnotu 10000.

```
SELECT Vyrobce, AVG(Cena)
FROM Zbozi
GROUP BY Vyrobce
HAVING AVG(Cena) > 10000;
```

VYROBCE	AVG(CENA)
Sony	12500
Apple	15560

Tabulka 4 - Výstup dotazu s HAVING (vlastní tvorba)

Výstup těchto dotazů není seřazen, seřazení ale můžeme vynutit klauzulí **ORDER BY**, který výslednou tabulku seřadí podle zvoleného sloupce od nejmenší po největší hodnotu. [21] Opačné řazení jde dosáhnout přidáním slova **DESC** (zkratka slova descending – klesající). Řadit je možné i podle více atribut, porovnávání druhé a další atributy se provede v případě stejné hodnoty v předchozím srovnávacím atributu. Následující příkaz seřadí produkty od nejdražšího po nejlacinější:

```
SELECT * FROM Zbozi ORDER BY Cena DESC;
```

Výše zmíněné agregační funkce jsou k dispozici téměř ve všech databázových systémech, každý databázový systém ale nabízí další různé agregační funkce, především pro získání různých statistických charakteristik.

Kromě již zmíněné statistické funkce `AVG()` je také často dostupná funkce `STD()` nebo `STDDEV()` pro výpočet směrodatné odchylky, dále bývá k dispozici funkce `VARIANCE()`, která počítá rozptyl a nebo funkce `CORR()`, sloužící k výpočtu korelačního koeficientu pro dvě datové sady. [22] V závislosti na výběru databázového systému může ale být k dispozici i více agregačních funkcí, vždy je vhodné hledat informace o dostupnosti agregačních funkcí přímo v dokumentaci daného databázového systému.

Další agregační funkce pak umožňují data z databáze převést do jiného datového typu, čehož může být využito pro vytvoření čitelného výstupu pro uživatele, pro export dat z databáze anebo pro další zpracování dat danou databází.

Databázový systém PostgreSQL například nabízí funkci `string_agg()` agregující daná data ze sloupce do jednoho textového řetězce, funkce `json_agg()` agreguje výsledná data do jednoho řetězce odpovídající souboru JSON, který je pak snadno čitelný jinými programovacími jazyky a v neposlední řadě je k dispozici funkce `array_agg()`, která převede data do datové struktury pole. [22]

PostgreSQL dále nabízí funkci pro logickou operaci konjunkce a disjunkce pomocí funkcí `bool_and()` a `bool_or()` a také implementuje bitový součin a bitový součet funkcemi `bit_and()` a `bit_or()`. [22]

4. Vlastní práce

Cílem této práce bude demonstrovat přínos agregace otevřených dat vytvořením relační databáze pro vybranou datovou sadu z portálu otevřených dat ČR, naplnění této databáze daty z tohoto portálu a vytvoření užitečných agregačních dotazů, které budou data z této databáze shromažďovat do jednodušších a užitečnějších tabulek.

Data, která budou v této práci zpracovávána se nazývají „Cizinci podle státního občanství, věku a pohlaví“ a jsou k dispozici na portálu otevřených dat, v době psaní této práce (rok 2022) jsou dostupná data od roku 2004 do roku 2020. Tato data se dají zobrazit na portálu dat pomocí vyhledávání klíčového slova „cizinec“, nebo lze použít tento hypertextový odkaz:

https://data.gov.cz/datové-sady?velikost-stránky=20&dotaz=cizinec.

Tato data byla vybrána pro svoji kvalitu, která na škále otevřenosti dat dosahuje čtyř hvězd, tak pro přiměřené množství obsažených dat. Databáze s těmito daty tedy nebude příliš jednoduchá, což by znemožňovalo vytváření komplexních SQL dotazů, ale ani natolik složitá, aby překračovala rozsah této práce. I přes svou kvalitu mají tato data své menší problémy, které budou adresovány dále v práci.

Zpracování dat stažených z tohoto portálu bude provedeno v programovacím jazyku Python, který je v současnosti nejpoužívanější programovací jazyk, především pro svou jednoduchost a efektivitu programování. Kromě samotného zpracování dat bude Python sloužit i jako nástroj pro komunikaci s databází, což zahrnuje především vytvoření databáze, naplnění databáze daty a provádění různých SQL příkazů. Výsledek těchto příkazů pak bude pomocí Pythonu interpretován uživateli.

Nejdůležitější součástí této práce je ale samotná relační databáze, zde bude využit databázový systém SQLite, především díky skutečnosti, že Python v sobě obsahuje knihovnu sqlite3, která SQLite využívá a není tedy třeba instalovat žádný databázový server či jiný program.

Důraz bude rovněž kladen na automatizaci celého tohoto procesu, a to, aby data a databáze byly generovány s co nejmenším počtem zásahu uživatele díky čemuž bude triviální upravit databázi v budoucnosti kdy budou dostupná data pro rok 2021, 2022, atd.

4.1 Zmapování situace řešené problematiky

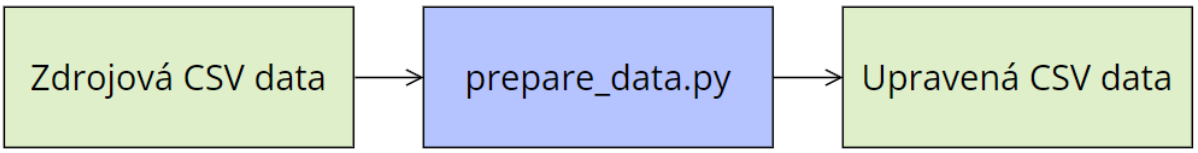
Data o počtu cizinců jsou veřejnosti k dispozici na stránkách Českého statistického úřadu na webové adrese https://www.czso.cz/csu/cizinci/cizinci-pocet-cizincu, kde jsou k dispozici i poměrně podrobné tabulky, a to buď ve formátu pro Excel anebo jako .pdf.

Dalším možností, jak získat data o cizincích je navštívit webové stránky Ministerstva vnitra, konktrétně adresu https://www.mvcr.cz/clanek/informativni-pocty-obyvatel-v-obcich.aspx, zde jsou data k dispozici ve formátu pro Excel.

Ačkoli obě stránky nabízí velké množství dat o počtu cizinců v ČR, ani jedna z nich ale neumožnuje uživateli interaktivně filtrovat data která ho zajímají a celkově neobsahují takové množství dat, jako bude umožňovat vlastní řešení.

4.2 Úprava zdrojových dat

V této části bude vytvořen Python script s názvem „prepare_data.py“ upravující zdrojová CSV data podle následujícího diagramu:



Obrázek 4 - Diagram úpravy dat (vlastní tvorba)

Zdrojová data jsou po stažení a po vybalení z archivu dostupná ve formátu .csv s názvem „CIZ01_{rok}.csv“, kde {rok} označuje rok, jehož informace daný soubor obsahuje. Data v CSV formátu mají formát tabulky, kde jednotlivé řádky souboru odpovídají řádku v tabulce a sloupce jednotlivého řádku se dají získat rozdělením řádku v CSV souboru na každé čárce (odtud také vznikl název CSV souboru – Comma separated values, česky: hodnoty oddělené čárkou).

Ve stažených datech s cizinci je celkem 19 sloupců, mnoho z nich ale obsahuje duplicitní data, např. v každém řádku je uložen kód občanství cizince a textová reprezentace tohoto občanství, např. občanství Ukrajina je v CSV souboru uložen pomocí kódu 804 s textovou reprezentací „Ukrajina“. Při správném návrhu databáze a při splnění všech normálních forem je možné tato duplikátní data eliminovat a nahradit v jednom záznamu výše zmíněný pár kód – text pouze kódem občanství.

Po sloučení duplikátních atribut do jednoho objektu a po přeskočení nepotřebných sloupců (těch které obsahují hodnoty stejné pro celou databázi), obsahuje každý záznam tyto atributy: občanství záznamu, pohlaví záznamu, věková kategorie záznamu (každá kategorie s rozmezím 5 let), území záznamu (okres a kraj), rok, pro který daný záznam platí (přestože by se tato vlastnost dala zjistit i pomocí názvu souboru) a počet cizinců které předchozí vlastnosti splňují.

Zdrojová CSV data ale mají několik menších problémů, které znesnadňují práci s daným souborem.

Prvních z nich je použití čárky v samotném řetězci s hodnotou, což může způsobit problémy při rozdělování dat na sloupce. Tento problém se objevil ve sloupci s věkovým rozmezím, který vypadal např. následovně "<25, 30)". Čárka oddělující hodnoty 25 a 30 může být při použití špatného algoritmu reprezentována jako oddělovač sloupců, což může způsobit špatné zpracování dat. Tohoto nedostatku si ale autoři této datové sady všimli, a proto je v datech od roku 2019 použit k oddělení čísel ve věkovém rozmezí středník.

Dalším problémem je způsob zápisu prázdné hodnoty, v datech do roku 2018 je prázdná hodnota reprezentována řetězcem "" (v souboru by pak bylo napsáno "hodnota1","","hodnota3"), v datech od roku 2019 je ale nulová hodnota reprezentována úplnou absencí jakékoli řetězce (jako ekvivalentní příklad uvedu text "hodnota1",,"hodnota3", povšimněte si dvou hned po sobě jdoucích čárek oddělující sloupec s nulovou hodnotou).

Posledním problémem je důvod proč vůbec existují nulové hodnoty, ve zdrojových souborech jsou totiž data již do jisté míry agregována, např. je zde záznam obsahující informace o celkovém počtu cizinců z Ukrajiny, bez ohledu na to, v jakém kraji jsou, poté mají sloupce zastupující kraj a okres nulovou hodnotu.

Tyto problémy jsou vyřešil pomocí menšího Python skriptu, který zpracuje všechny zdrojové CSV soubory, a to především pomocí vymazání záznamů, které obsahují nulovou hodnotu, jelikož ve výsledné databázi budou pouze úplná data a případná agregace bude provedena pomocí SQL příkazů. Tento skript byl pojmenován „prepare_data.py“ a jeho spuštění vygeneruje upravený CSV soubor pro každý zdrojový CSV soubor. Tyto upravené soubory poté budou použity k vytvoření dat pro databázi.

Úprava těchto dat může být shrnuta následujícím výňatkem ze souboru „prepare_data.py“, ve kterém se vyfiltrují řádky, které obsahují nulovou hodnotu.

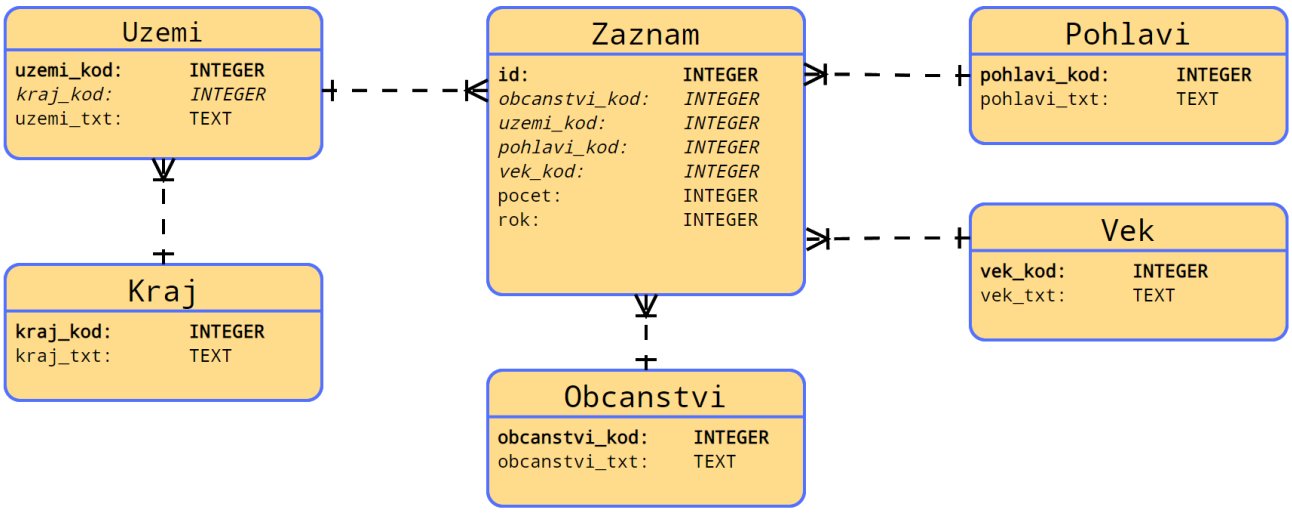
```
#prepare_data.py (část kodu)

#otevreme soubor pro cteni
with open(constants.SOURCE_FILENAME.format(year=year), "r", encoding="utf-8") as input:
    lines = input.readlines()

filtered_lines = []
for line in lines:
    line = line.strip().split(",")
    for column in line:
        if column == "" or column == '""':
            #pokud radek obsahuje nulovou hodnotu, nezapisujeme ho
            break
    else:
        #pokud maji vsechny sloupce data, radek ulozime
        filtered_lines.append(",".join(line))
```

4.3 Vytvoření relačního modelu

Dalším krokem je vytvoření relačního modelu, který bude databáze využívat. Pro každou z atribut pohlaví, věk, kraj, okres a občanství je vytvořena vlastní tabulka obsahující kód dané atributy, který je rovněž použit jako unikátní primární klíč a textovou reprezentaci dané atributy. Tabulka s okresem pak má navíc cizí klíč propojující daný okres s krajem.

Nejdůležitější tabulkou je pak tabulka obsahující samotný záznam, obsahující informace o roce, počtu cizinců a obsahující cizí klíče na výše zmíněné tabulky (kromě atributy kraj, ten je propojen se záznamem skrze tabulku okres).

Tyto tabulky a jejich atributy pak vytváří následující relační model:



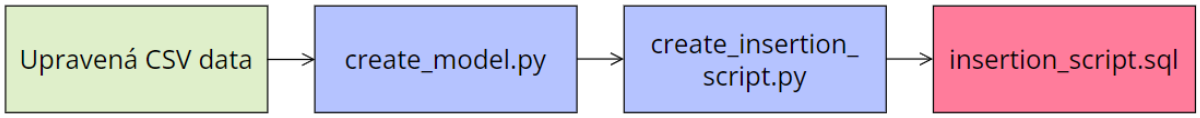
Obrázek 5 - Relační model navržené databáze (vlastní tvorba)

Z tohoto modelu pak bylo snadné vytvořit SQL příkazy pro vytvoření všech tabulek, tento SQL skript byl uložen do souboru „creation_script.sql“, zde je výňatek z tohoto souboru:

```sql
-- creation_script.sql (cast kodu)

CREATE TABLE uzemi(
    uzemi_kod INTEGER PRIMARY KEY,
    uzemi_txt TEXT NOT NULL,
    kraj_kod INTEGER NOT NULL
);

CREATE TABLE zaznam(
    id INTEGER PRIMARY KEY,
    hodnota INTEGER NOT NULL,
    rok INTEGER NOT NULL,
    pohlavi_kod INTEGER NOT NULL,
    obcanstvi_kod INTEGER NOT NULL,
    vek_kod INTEGER NOT NULL,
    uzemi_kod INTEGER NOT NULL
);
```

4.4 Vytvoření SQL dat

Po vytvoření relačního modelu je dalším krokem vytvoření skriptu, který vygeneruje příkazy pro vložení dat do databáze, k čemuž byl opět použit Python a již upravené CSV soubory byly převedeny do SQL skriptu využívající příkaz INSERT, skripty vytvořené v tomto kroku budou implementovat následující diagram:



Obrázek 6 - Diagram vytváření SQL dat (vlastní tvorba)

4.4.1 Vytvoření datového modelu v Pythonu

Přestože jsou data ve vygenerovaných CSV souborech upravená, stále ještě obsahují redundantní data a načtení dat v tomto formátu do databáze by nebylo možné a porušovalo by normální formy. Proto byly jako první v Pythonu navrženy datové typy odpovídající tabulkám v databázi, ty pak byly použity k načtení všech dat z upravených CSV souborů do datového modelu v Pythonu a z něj budou později data převedena do SQL skriptu.

Poté byl vytvořen skript obsahující datové typy odpovídající těm v „creation scriptu“, tento soubor je uložen v souboru s názvem „database_types.py“, část kódu z tohoto souboru je k prohlídnutí zde:

```
#database_types.py (cast kodu)

@dataclass
class Vek:
    vek_kod: int            #kod veku, napr: 400000600005000, pouzito jako klic
    vek_txt: str            #vekove rozmezi jako text, napr. <0; 5)

    def __init__(self, vek_kod, vek_txt):
        self.vek_kod = vek_kod
        self.vek_txt = vek_txt

@dataclass
class Kraj:
    kraj_kod: int           #kod kraje: napr: 3069, pouzito jako klic
    kraj_txt: str           #nazev kraje: "Ústecký kraj"

    def __init__(self, kraj_kod, kraj_txt):
        self.kraj_kod = kraj_kod
        self.kraj_txt = kraj_txt
```

Tyto datové typy poté byly využity v Python programu s názvem „create_model.py“, který přečte všechny upravené CSV soubory, z každého záznamu vyjme sloupce s relevantními daty a vytvoří instance těchto typů se skutečnými daty.

Zároveň jsou tyto instance uloženy v datových kontejnerech podle jednotlivých typů, například je zde kontejner shromažďující všechna občanství, všechny kraje atd., ukázka souboru „create_model.py“ je k dispozici níže, v této části jsou do proměnných ukládány jednotlivé sloupce ze záznamu, poté je proveden test, zda bylo dané pohlaví již v modelu uloženo a pokud ne, je do modelu uloženo.

```
#create_model.py (cast kodu)

vek_kod = int(line[8])
kraj_kod = int(line[13])
uzemi_kod = int(line[11])

hodnota = int(line[1])
rok = int(line[9])

if pohlavi_kod not in self.seznam_pohlavi:
    pohlavi_txt = line[16]
    self.seznam_pohlavi[pohlavi_kod] = Pohlavi(pohlavi_kod, pohlavi_txt)
```

4.4.2 Vytvoření insertion skriptu

Posledním krokem k vytvoření SQL insertion skriptu je převedení dat z datového modelu do SQL příkazu INSERT. Zde stačí jednoduše vzít existující datový model z minulého kroku a naformátovat atributy jednotlivých typů, aby odpovídaly syntaxi INSERT příkazu.

K tomuto kroku slouží Python skript „create_insertion_script.py“, jehož ukázka je k prohlídnutí zde:

```
# create_insertion_script.py (cast kodu)

for pohlavi in model.seznam_pohlavi.values():
    print(f'''INSERT INTO pohlavi VALUES
            ({pohlavi.pohlavi_kod}, "{pohlavi.pohlavi_txt}");
            ''', file=output)
print(file=output)

for obcanstvi in model.seznam_obcanstvi.values():
    print(f'''INSERT INTO obcanstvi VALUES
            ({obcanstvi.stobcan_kod}, "{obcanstvi.stobcan_txt}");
            ''', file=output)
print(file=output)
```

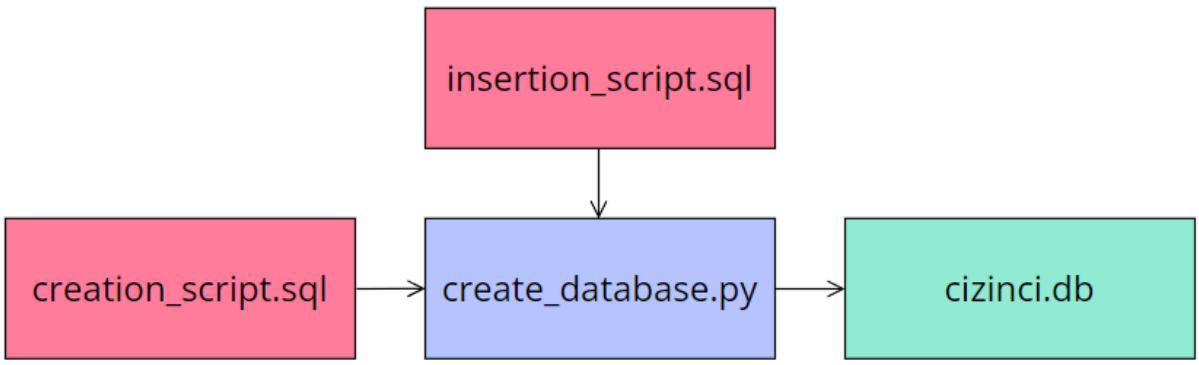
Tento kód nahradí hodnoty uvnitř {} (s využitím Python f-stringu) skutečnou hodnotou z modelu a uloží všechny **INSERT** příkazy do souboru s názvem „insertion_script.sql“. Toto je provedeno pro všechny tabulky v databázi, výsledkem je

pak téměř 700 tisíc řádku dlouhý SQL soubor který se dá použít pro naplnění databáze daty, část tohoto souboru je k dispozici zde:

```
-- insertion_script.sql (cast kodu)

INSERT INTO pohlavi VALUES (1, "muž");
INSERT INTO pohlavi VALUES (2, "žena");

INSERT INTO obcanstvi VALUES (4, "Afghánistán");
INSERT INTO obcanstvi VALUES (12, "Alžírsko");
INSERT INTO obcanstvi VALUES (31, "Ázerbájdžán");
```

4.5 Vytvoření databáze

Jelikož jsou již připraveny SQL skripty k vytvoření tabulek a k naplnění těchto tabulek daty, je nyní už možno databázi skutečně vytvořit. K vytvoření databáze byl napsán jednoduchý Python skript, který s použitím knihovny sqlite3 vytvoří databázový soubor, připojí se k němu, a spustí nejdříve creation_script.sql k vytvoření tabulek a poté insertion_script.sql, který ji naplní daty. Tyto změny jsou poté uloženy a vzniká soubor „cizinci.db“, ke kterému se později můžeme opět připojit a provádět na něm různé dotazy, vytvoření databáze je znázorněno v následujícím diagramu:



Obrázek 7 - Diagram vytvoření databáze (vlastní tvorba)

4.6 Vytvoření agregačních SQL dotazů

Nyní je již k dispozici funkční databáze s daty, ta může být využita k získávání agregovaných dat pomocí SQL dotazů. Dotazy byly voleny tak, aby jejich výsledky přinášely nějaké demografické poznatky o cizincích podle zvolených filtrů anebo tak aby byly užitečné v programu, který bude představen v další části.

První dotaz je velmi jednoduchý a slouží k nalezení textové reprezentace daného klíče, např. k nalezení občanství ke kódu občanství 804. Dotaz byl cíleně napsán s použitím proměnné, a to proto aby bylo možné tento dotaz modifikovat pro jakýkoli kód občanství. Podobný dotaz byl napsán pro všechny tabulky, které mají textovou reprezentaci.

```
SELECT stobcan_txt
FROM obcanstvi
WHERE stobcan_kod = {stobcan_kod};
```

Proměnná „stobcan_kod“ vymezená složenými závorkami je nahrazena kódem, jehož reprezentaci má najít. Tato substituce je provedena v Pythonu, do databáze je pak poslán příkaz např:

```
SELECT stobcan_txt
FROM obcanstvi
WHERE stobcan_kod = 804;
```

Druhý vytvořený dotaz slouží k získání věkové pyramidy pro zvolenou skupinu cizinců, pokud je cílem zjistit věkovou pyramidu pro celou ČR nebo nějaký okres, stačí jednotlivé věkové kategorie seskupit přímo na tabulce záznamy, pokud je za oblast zvolen nějaký kraj, je nutné nejdřív spojit tabulku záznamů s tabulkou s okresy.

Tento dotaz opět využívá proměnné, aby šel použít s libovolnými restrikcemi:

```
SELECT vek_kod, SUM(hodnota)
FROM zaznam
JOIN uzemi ON zaznam.uzemi_kod = uzemi.uzemi_kod
WHERE kraj_kod = {oblast}
AND rok = {rok}
AND obcanstvi_kod = {obcanstvi}
GROUP BY vek_kod
ORDER BY vek_kod;
```

V případě že nějaká atributa nemá být omezena (např. je cílem zjistit věkové složení pro všechny cizince bez ohledu na občanství), Python nahradí tyto proměnné názvem sloupce, což způsobí vrácení všech záznamů. Za příklad může sloužit dotaz, ve kterém je cílem zjistit věkovou pyramidu cizinců v roce 2015 v Ústeckém kraji (s kódem kraje 3069), bez ohledu na občanství cizince.

```
SELECT vek_kod, SUM(hodnota)
FROM zaznam
JOIN uzemi ON zaznam.uzemi_kod = uzemi.uzemi_kod
WHERE kraj_kod = 3069
AND rok = 2015
AND obcanstvi_kod = obcanstvi_kod
GROUP BY vek_kod
ORDER BY vek_kod;
```

Tento dotaz vygeneruje následující výsledek:

```
<0, 5)          | 1208
<5, 10)         | 1113
<10, 15)        | 934
<15, 20)        | 953
<20, 25)        | 1386
<25, 30)        | 3204
<30, 35)        | 3808
atd.
```

Tento výsledek pak může být využit k výpočtu průměrného věku vynásobením střední hodnoty z věkových rozmezí s počtem cizinců v daném věku, sečtením výsledků a vydělením sumy celkovým počtem cizinců.

Obdobný dotaz byl také vytvořen pro zjištění pohlaví cizinců, průměrný věk byl ale nahrazen procentuálním zastoupením jednotlivých pohlaví.

Databáze celkem obsahuje přes 180 občanství, a protože zobrazení takto velké tabulky by mohlo být nepřehledné, bude počet občanství omezen pouze na N nejpočetnějších občanství, hodnota N byla nastavena na 10, je ale možné ji změnit v souboru „constant.py“, níže zobrazený dotaz slouží k získání N nejpočetnějších občanství, tato občanství jsou poté využita pro vygenerování tabulek zobrazující vývoj počtu cizinců v jednotlivých letech.

```
SELECT obcanstvi_kod, SUM(hodnota) AS suma
FROM zaznam
WHERE rok = {rok}
GROUP BY obcanstvi_kod
ORDER BY SUM(hodnota) DESC LIMIT {constants.ROW_LIMIT};
```

Jelikož je vhodné uživatele informovat o tom, že je výpis dat omezen, bude mu zobrazena informace o tom, kolik občanství bylo opomenuto. Toto se dá zjistit odečtením limitu N od celkového počtu občanství. Pro získání celkového počtu občanství v databázi byl proto napsán následující dotaz:

```
SELECT COUNT(stobcan_kod) from obcanstvi;
```

4.7 Vytvoření programu pro interakci s daty

Funkční databáze a SQL dotazy jsou perfektním nástrojem pro uživatele, který s těmito technologiemi zacházet umí, většina populace ale s databázemi pracovat neumí, a proto byl vytvořen program, který se na tuto databázi napojí, umožní uživateli v terminálu interaktivně volit určité parametry – období, občanství a území a program poté využije výše napsané SQL dotazy a vygeneruje souhrnné informace dle zvolených parametrů.

Bez ohledu na to, jaké parametry jsou zvolené, uživatel vždy obdrží informaci o celkovém počtu cizinců a o změně oproti minulému roku (s výjimkou prvního roku 2004), informaci o průměrném věku a věkovou pyramidu a také rozbor pohlaví.

Podle toho, jaké parametry byly zvoleny, je také zobrazena tabulka zobrazující buď změnu v čase anebo stav v daném roce pro všechny kraje. Řádky a sloupce této tabulky závisí na parametru který nebyl zvolen, pokud bude například zvolen pouze okres, tabulka bude mít sloupce s počtem cizinců v jednotlivých letech, zatímco řádky obsahují jednotlivá občanství.

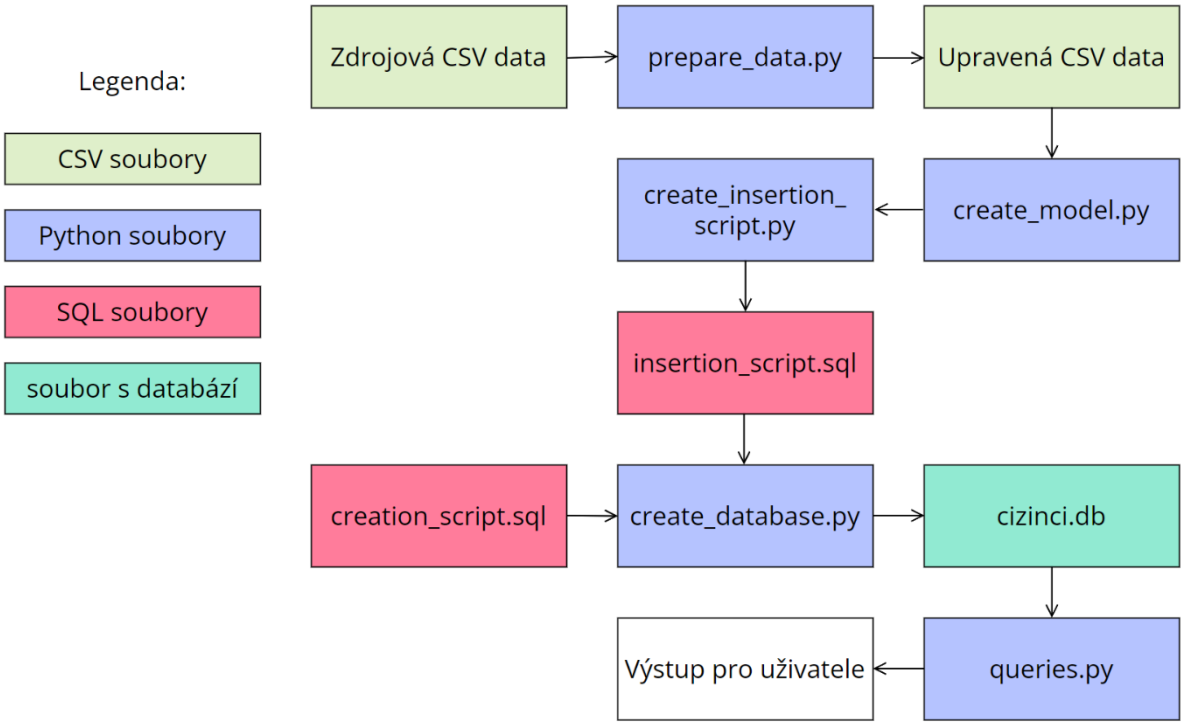
Tento program byl pojmenován „queries.py“ a pro ukázku funkcionality tohoto programu je uveden výstup programu po zvolení Ústeckého kraje, bez omezení na národnost a rok. Vzhledem k velikosti tabulky je výstup vložen jako obrázek, a to s odříznutím let 2011–2020, uživateli programu je ale tabulka zobrazena celá.

```
Výsledek dotazu vyhovující následujícím parametrům:

 - Zvolená oblast:                Ústecký kraj
 - Zvolené občanství cizince:     Jakékoli
 - Zvolený rok:                   2004 - 2020

Celkový počet cizinců: 39038    +177 změna oproti roku 2019

Věkové složení:

        Věkové rozmezí   | Počet
        -------------------------------
        <0, 5)           | 1105
        <5, 10)          | 1606
        <10, 15)         | 1397
        <15, 20)         | 1144
        <20, 25)         | 1426
        <25, 30)         | 2772
        <30, 35)         | 4348
        <35, 40)         | 4709
        <40, 45)         | 4567
        <45, 50)         | 3971
        <50, 55)         | 3491
        <55, 60)         | 2935
        <60, 65)         | 2122
        <65, 70)         | 1560
        <70, 75)         | 915
        <75, 80)         | 450
        <80, 85)         | 265
        <85, N)          | 255

        Průměrný věk: 40.9

Složení dle pohlaví:
        Pohlaví  | Počet    | Procento
        -----------------------------------
        muž      | 24686    |  63.24 %
        žena     | 14352    |  36.76 %

Vývoj počtu cizinců v letech 2004 - 2020 (omezeno na 5 občanství s nejvíce cizinci v roce 2020):
------------------------------ |-------- |-------- |-------- |-------- |-------- |-------- |-------- |
                               |    2004 |    2005 |    2006 |    2007 |    2008 |    2009 |    2010 |
------------------------------ |-------- |-------- |-------- |-------- |-------- |-------- |-------- |
SOUČET                         |   19731 |   22130 |   26336 |   33053 |   35451 |   31995 |   30578 |
------------------------------ |-------- |-------- |-------- |-------- |-------- |-------- |-------- |
Ukrajina                       |    5545 |    6484 |    8161 |    8931 |    8894 |    7847 |    6585 |
Slovensko                      |    2273 |    2511 |    2744 |    3063 |    3350 |    3283 |    3340 |
Vietnam                        |    5676 |    6182 |    6879 |    8416 |    9344 |    8997 |    8542 |
Rusko                          |    1050 |    1123 |    1296 |    1636 |    1871 |    2044 |    1921 |
Německo                        |     490 |     896 |    1977 |    4805 |    5362 |    3101 |    3338 |
 + 183 dalších občanství

Pokračujte zmáčknutím enteru.
```

Obrázek 8 - Výstup programu "queries.py" (vlastní tvorba)

4.8 Shrnutí funkcionality vlastní práce

Ve vlastní práci bylo vytvořeno několik Python skriptů, které dohromady vytvoří databázi a umožní uživateli získávat agregovaná data o cizincích dle zvolených parametrů. Vzhledem k většímu počtu souborů byl také vytvořen následující digram popisující závislosti souborech na datech a popisující v jakém pořadí jsou data zpracována.



Obrázek 9 - Diagram úpravy dat a vytvoření databáze (vlastní tvorba)

Všechny Python skripty, zdrojové i upravené CSV soubory i soubor s databází jsou veřejně dostupné k prohlídnutí a ke stažení online v online GitHub repositáři na adrese www.github.com/lukas-jirusek/cizinci/, díky čemu je možné, aby si kdokoli potřebná data stáhl a je mu tím umožněno tvořit si vlastní dotazy.

4.9 Ověření správnosti navrženého řešení

Ověření správnosti řešení lze provést několika způsoby, prvním z nich je otevření CSV souborů z portálu otevřených dat s počtem cizinců a manuálním či automatizovaným vybráním relevantních řádků a sečtením hodnot ve sloupci s počtem cizinců.

Za příklad je uveden Bash příkaz, který zobrazí řádky s počtem cizinců z Ukrajiny ve městě Benešov ve věkové kategorii [40–45) let. Tento příkaz několikrát využívá příkaz grep, který filtruje CSV soubor tak, aby byly vypsány pouze relevantní řádky, ukázku takového příkazu a jeho výstup je vidět zde:

```
$ cat 2020.csv | grep "Benešov" | grep "Ukrajina" | grep "<40; 45)"
"889585871","70","6121","102","1","3228","804","7700","410040610045000",…
"889596463","79","6121","102","2","3228","804","7700","410040610045000",…
```

Lze vidět že tento příkaz vybral dvě řádky s relevatními daty, dalším krokem je sečtení hodnot v druhém sloupci v těchto řádcích, čímž zjistíme že v Benešově žilo v roce 2020 149 cizinců z Ukrajiny ve věku 40 – 45 let, tento výsledek ale můžeme získat snadněji využitím interaktivního Python programu queries.py.

V tomto programu si interaktivně vybereme oblast Benešov, národnost Ukrajina a rok 2020 a spustíme příkaz s následujícím výsledkem:

```
Výsledek dotazu vyhovující následujícím parametrům:

 - Zvolená oblast:                Benešov (Středočeský kraj)
 - Zvolené občanství cizince:     Ukrajina
 - Zvolený rok:                   2020

Celkový počet cizinců: 1110     +130 změna oproti roku 2019

Věkové složení:

        Věkové rozmezí   | Počet
        -----------------------------
        <0, 5)           | 26
        <5, 10)          | 49
        <10, 15)         | 37
        <15, 20)         | 37
        <20, 25)         | 56
        <25, 30)         | 103
        <30, 35)         | 104
        <35, 40)         | 137
        <40, 45)         | 149
        <45, 50)         | 147
        …
```

Z tohoto výstupu je poté vyjmuta relevantní hodnota z věkové pyramidy, ve věkovém rozmezí 40 – 45 let je 149 cizinců, což je stejná hodnota jako ta zjištěná příkazem grep, ale hodnotu lze získat mnohem snadněji, rychleji a také jsou k dispozici data z více let, průměrný věk a jiné agregované údaje.

K ověření správnosti dat lze také použít jiná, již existující řešení, která rovněž řeší tento problém, konktétně je možno využít data z webu Českého statistického úřadu, která byla představena na začátku vlastní práce.

Přestože data na této stránce nejsou tak podrobná jak ve vytvořené databázi, lze je i tak využít k získání celkového počtu cizinců jednotlivých národností v celé ČR, pro ověření několika hodnot z mojí databáze využiji soubor “R04 Cizinci v ČR podle státního občanství v letech 1994 - 2020 (k 31. 12.)” ze stránek ČSÚ, který je dostupný zde: https://www.czso.cz/csu/cizinci/4-ciz_pocet_cizincu.

Z tohoto souboru lze zjistit že celkový počet cizinců v ČR v roce 2020 bez ohledu na jejich národnost je 632 570, z toho je 20 861 němců a 41 692 rusů. Pomocí těchto hodnot lze ověřit správnost navrženého řešení vybráním daných atribut v programu queries.py s následujícími výsledky:

```
Výsledek dotazu vyhovující následujícím parametrům:

 - Zvolená oblast:                Celá ČR
 - Zvolené občanství cizince:     Jakékoli
 - Zvolený rok:                   2020

Celkový počet cizinců: 632570   +39204 změna oproti roku 2019

Výsledek dotazu vyhovující následujícím parametrům:

 - Zvolená oblast:                Celá ČR
 - Zvolené občanství cizince:     Německo
 - Zvolený rok:                   2020

Celkový počet cizinců: 20861    -617 změna oproti roku 2019

Výsledek dotazu vyhovující následujícím parametrům:

 - Zvolená oblast:                Celá ČR
 - Zvolené občanství cizince:     Rusko
 - Zvolený rok:                   2020

Celkový počet cizinců: 41692    +3682 změna oproti roku
```

Opět lze vidět že se agregované hodnoty v získané z vytvořeného programu shodují s hodnotami z webových stránek ČSÚ, a lze tedy tento výsledek označit za správný.

Bylo provedeno několik testů s náhodně zvolenými parametry k ověření správnosti navrženého řešení a to jak s využitím zdrojových CSV souborů, tak i s využitím dat dostupných z webu ČSÚ. Tyto hodnoty poté byly porovnány a výsledkem spočítaným vlastním řešením a bylo zjištěno že se ve všech testech shodují a tím lze potvrdit správnost řešení.

5. Výsledky a diskuse

5.2. Zobecnění řešení

Řešení bylo navrženo tak, aby dané skripty a programy fungovaly i při přidání nových dat (v době psaní práce existují data pouze do roku 2020), v případě vydání nových dat Českým statistickým úřadem tak stačí tato data stáhnout a umístit je do správného adresáře ve složce projektu, pozměnit hodnotu maximálního roku a spustit veškeré skripty znovu pro vygenerování nové databáze obsahující tato data.

Toto řešení ale nezobecňuje navrženou databázi tak, aby v ní byla možná uložit jakákoli jiná otevřená data, navržené řešení slouží jen a pouze k agregaci dat o počtu cizinců, agregování jiných dat by vyžadovalo navržení jiné databáze a tabulky odpovídající právě těm datům, která by měla být agregována.

5.3. Diskuse

Navržené řešení by mohlo být dále vylepšeno, současné navržené řešení dává uživatelovi možnost si výslednou databázi stáhnout pro vlastní užití, které ale vyžaduje znalost relačních databází a jazyka SQL, což znemožnuje využití databáze těm, které tyto znalosti nemají.

Tento problém byl adresován vytvořením Python programu, který s touto databází komunikuje a vytváří pro ní dotazy, díky čemuž se velice snižují požadavky na technické znalosti uživatele, přesto je nutné aby měl uživatel dostatek znalostí k nainstalování a stažení potřebných programů a souborů.

Možným zlepšením by bylo vytvoření webové stránky, kde by agregovaná data byla přehledně zobrazena online, uživatelovi by pak stačilo tyto stránky navštívit a mohl by požadovaná data nalézt jednodušše v prohlížeči.

Dalším potencionálním problémem by byla změna formátu poskytovaných dat, jak se již v roce 2019 stalo, nicméně v současné době jsou data ve velmi vysoké kvalitě a neexistuje důvod, proč by taková změna měla nastat, pokud by se tak ale stalo, musel by být změněn program upravující zdrojová CSV data tak, aby byl tento nový formát dat také schopen upravit.

Posledním možným zlepšením by bylo zoptimalizování SQL dotazů pro zvýšení rychlosti generování výsledných tabulek, v současném řešení jsou data několikrát přeposílána mezi Pythonem a databází, kvůli čemuž není naplno využito plného potenciálu relačních databází.

6. Závěr

Hlavním cílem této práce bylo vytvoření relační databáze, ta byla naplněna daty z portálu otevřených dat, k této databázi byly poté vytvořeny vhodné SQL dotazy agredující data o počtu cizinců do vhodných výsledku.

Tím byl demonstrován přínos agregace dat pro zvýšení informační hodnoty často příliš podrobných otevřených dat, výsledky agregované vytvořenými SQL dotazy této práce pak mohou sloužit pro statistické účely, ale i pro zaměstnance místních úřadů a nebo českých i zahraničních úřadů, kteří by díky datům která tato práce agreguje mohou vytvářet informovanější rozhodnutí. Přínos této práce je ještě navýšen díky současné situaci v Evropě, především kvůli invazi Ruska na Ukrajinu, která způsobila a i v budoucnosti způsobí (práce byla psána na začátku ruské invaze) velkou migrační vlnu, práce se tedy zabývá velmi aktuálním problémem.

V práci byly použity metody modelování, nejdříve byl vytvořen vhodný relační model databáze a to tak, aby databáze co nejlépe ukládala data o cizincích a to se splněním tří normálních forem, poté byla použita metoda implementace, během čehož byla databáze skutečně vytvořena s využitím databázového systému SQLite.

Mimoto byl také použit programovací jazyk Python a to jak pro počáteční úpravu zdrojových dat, tak pro vytvoření SQL skriptů pro vytvoření tabulek a pro vložení dat do databáze, Python byl rovněž použit pro samotné vytvoření databáze a pro komunikaci s touto databází. Dále byl Python využit k vytvoření programu, díky kterému může agregovaná data získat každý s minimální znalostí jazyka Python.

Aby byl přínos této práce co největší, všechny Python a SQL scripty, upravené CSV soubory a i samotný soubor s databází vytvořené v této práci byly publikovány ve veřejném GitHub repositáři, díky čemuž může výsledky této práce využít každý, což také podporuje myšlenku otevřených dat tím, že tato data jsou k dispozici poskytována dále. Tento repositář je k dispozici na adrese https://github.com/lukas-jirusek/cizinci.

Důraz byl rovněž kladen na zobecnění navrženého řešení a to tak aby navržené řešení fungovalo i s novými daty, který by měly být v budoucnu k dispozici a je tedy možné vždy databázi snadno aktualizovat. Dále byla ověřena správnost navrženého řešení a proběhla diskuze na možné problémy s navrženým řešením a jak by bylo možné tyto problémy vyřešit a jak by bylo možné řešení vylepšit.

Otevřená data mohou být skvělým nástrojem pro získávání informací a ke zvýšení informovanosti o daném subjektu, často jsou ale příliš detailní a jejich pravý potenciál je získán až když jsou data agregována do souhrných hodnot a nebo do statistických výsledků. Cílem práce proto bylo demonstrovat jak mohou být tato data agregována a jaký užitek mohou přinést.

7. Seznam použitých zdrojů

[1] What is Open Data? Opendatahandbook.org [online]. 2018 [cit. 2022-03-14]. Dostupné z: http://opendatahandbook.org/guide/en/what-is-open-data/

[2] 5 Star Linked Data. W3.org [online]. 2013 [cit. 2022-03-14]. Dostupné z: https://www.w3.org/2011/gld/wiki/5_Star_Linked_Data

[3] About the ODI. Theodi.org [online]. [cit. 2022-03-14]. Dostupné z: https://theodi.org/about-the-odi/

[4] Novinky v otevřených datech. Data.gov.cz [online]. [cit. 2022-03-14]. Dostupné z: https://data.gov.cz/novinky/

[5] Datové sady. Data.gov.cz [online]. [cit. 2022-03-14]. Dostupné z: https://data.gov.cz/datové-sady

[6] The official portal for European data. Data.europa.eu [online]. [cit. 2022-03-14]. Dostupné z: https://data.europa.eu/en

[7] NOY, Natasha. An Analysis of Online Datasets Using Dataset Search. Ai.googleblog.com [online]. [cit. 2022-03-14]. Dostupné z: https://ai.googleblog.com/2020/08/an-analysis-of-online-datasets-using.html

[8] KLÍMEK, Jakub. Příklady špatné praxe v oblasti otevřených dat. Opendata.gov.cz [online]. 2021 [cit. 2022-03-14]. Dostupné z: https://opendata.gov.cz/%C5%A1patn%C3%A1-praxe:start

[9] Co je relační databáze. Oracle.com [online]. [cit. 2022-03-14]. Dostupné z: https://www.oracle.com/cz/database/what-is-a-relational-database/

[10] KULHAN, Jakub. Normalizace relačních databází. Programujte.com [online]. 2008 [cit. 2022-03-14]. Dostupné z: http://programujte.com/clanek/2008071900-normalizace-relacnich-databazi/

[11] Základy normalizace databáze. Docs.microsoft.com [online]. [cit. 2022-03-14]. Dostupné z: https://docs.microsoft.com/cs-cz/office/troubleshoot/access/database-normalization-description

[12] ŠKODA, Jan. Normalizace databáze. Vovcr.cz [online]. [cit. 2022-03-14]. Dostupné z: https://www.vovcr.cz/odz/tech/393/page20.html

[13] HORDĚJČUK, Vojtěch. Jazyk SQL. Voho.eu [online]. 2021 [cit. 2022-03-14]. Dostupné z: http://voho.eu/wiki/sql/

[14] WINAND, Markus. What's New in SQL:2016. Modern-sql.com [online]. [cit. 2022-03-14]. Dostupné z: https://modern-sql.com/blog/2017-06/whats-new-in-sql-2016

[15] SQL Data Types for MySQL, SQL Server, and MS Access. W3schools.com [online]. [cit. 2022-03-14]. Dostupné z: https://www.w3schools.com/sql/sql_datatypes.asp

[16] PETROVIC, Bojan. Implementing SQL data types. Sqlshack.com [online]. [cit. 2022-03-14]. Dostupné z: https://www.sqlshack.com/implementing-sql-data-types/

[17] What does ACID mean in Database Systems? Database.guide [online]. [cit. 2022-03-14]. Dostupné z: https://database.guide/what-is-acid-in-databases/

[18] SQL Aggregate Functions. Mode.com [online]. 2021 [cit. 2022-03-14]. Dostupné z: https://mode.com/sql-tutorial/sql-aggregate-functions/

[19] Built-in Aggregate Functions. Sqlite.org [online]. [cit. 2022-03-14]. Dostupné z: https://www.sqlite.org/lang_aggfunc.html

[20] PETERSON, Richard. SQL GROUP BY and HAVING Clause. Guru99.com [online]. [cit. 2022-03-14]. Dostupné z: https://www.guru99.com/group-by.html

[21] SQL ORDER BY Keyword. *W3schools.com* [online]. [cit. 2022-03-14]. Dostupné z: https://www.w3schools.com/sql/sql_orderby.asp

[22] Aggregate Functions. *Postgresql.org* [online]. [cit. 2022-03-14]. Dostupné z: https://www.postgresql.org/docs/9.5/functions-aggregate.html

[23] HARRINGTON, Jan L. Relational database design and implementation. Morgan Kaufmann, 2016. ISBN 9780128043998

[24] MELTON, Jim; SIMON, Alan R. Understanding the new SQL: a complete guide. Morgan Kaufmann, 1993. ISBN 9781558602458

[25] VALENTA, M., POKORNÝ, J. Databázové systémy. Praha: České vysoké učení technické v Praze, 2020. ISBN 978-80-01-06696-6

8. Seznam obrázků

9. Seznam tabulek