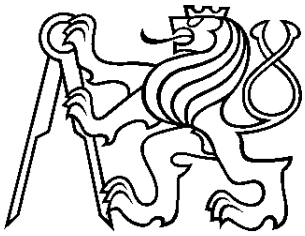


ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA DOPRAVNÍ

Bc. Lukáš Kacar

Analýza databáze jízdních dokladů (whitelistu) v Pražské integrované dopravě

Diplomová práce

2024

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta dopravní
děkan
Konviktská 20, 110 00 Praha 1

K620 ..... Ústav dopravní telematiky

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení studenta (včetně titulů):

Bc. Lukáš Kacar

Studijní program (obor/specializace) studenta:

navazující magisterský – IS – Inteligentní dopravní systémy

Název tématu (česky): **Analýza databáze jízdních dokladů (whitelistu) v Pražské integrované dopravě**

Název tématu (anglicky): Analysis of the ticketing database (whitelist) in Prague Integrated Transport

Zásady pro vypracování

Při zpracování diplomové práce se řiďte následujícími pokyny:

- Analyzujte rozsah a účel informací zasílaných ve whitelistu
- Navrhněte vhodné SW nástroje pro práci s whitelistem (velký objem dat řádu GB)
- Navrhněte a realizujte způsob validace a kontroly konzistentnosti dat
- Proveďte kontrolu reálně zasílaných dat, formulujte závěry

Rozsah grafických prací: dle požadavků vedoucího práce

Rozsah průvodní zprávy: minimálně 55 stran textu (včetně obrázků, grafů a tabulek, které jsou součástí průvodní zprávy)

Seznam odborné literatury: Mastrodomenico, R. The Python Book. Wiley, 2022.
Smluvní přepravní podmínky PID platné k 1.9.2023.
Odbavovací a informační zařízení ve vozidlech PID - Standardy odbavení. 7/2022.

Vedoucí diplomové práce: **Ing. Milan Sliacky, Ph.D.**
Ing. Lukáš Hrdina

Datum zadání diplomové práce: **10. července 2023**
(datum prvního zadání této práce, které musí být nejpozději 10 měsíců před datem prvního předpokládaného odevzdání této práce vyplývajícího ze standardní doby studia)

Datum odevzdání diplomové práce: **15. května 2024**
a) datum prvního předpokládaného odevzdání práce vyplývající ze standardní doby studia a z doporučeného časového plánu studia
b) v případě odkladu odevzdání práce následující datum odevzdání práce vyplývající z doporučeného časového plánu studia

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE Fakulta dopravní L. S.

Ing. Zuzana Bělinová, Ph.D.
vedoucí
Ústavu dopravní telematiky

prof. Ing. Ondřej Přibyl, Ph.D.
děkan fakulty

Potvrzuji převzetí zadání diplomové práce.

Bc. Lukáš Kacar
jméno a podpis studenta

V Praze dne 10. července 2023

Poděkování

Děkuji panu Ing. Milanovi Sliackému, Ph.D. z Katedry dopravní telematiky Fakulty dopravní ČVUT za vedení práce, rady a připomínky. Pak dále děkuji panu Ing. Lukášovi Hrdinovi z Oddělení technického rozvoje a projektů ROPID p. o. též za vedení práce, rady, připomínky a poskytnuté informace k whitelistu.

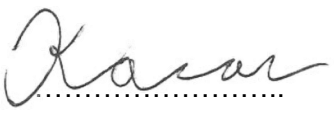
Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia ČVUT v Praze Fakultě dopravní.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a uvedl jsem veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací a Rámcovými pravidly používání umělé inteligence na ČVUT pro studijní a pedagogické účely v Bc. a NM studiu.

Nemám závažný důvod proti užívání tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 15. 5. 2024



……..

podpis

Abstrakt

Tato diplomová práce se zabývá analýzou a zpracováním dat ve whitelistu Pražské integrované dopravy, který obsahuje informace o jízdních dokladech. Cílem práce bylo převést rozsáhlou databázi whitelistu do zpracovatelné formy a provést její důkladnou analýzu. K tomuto účelu byl použit programovací jazyk Python a relační databázový systém SQL. První část práce se věnovala technickému postupu převodu whitelistu do databázové podoby a základní analýze dat. Následně byla provedena kontrola a validace dat z hlediska splnění tarifních a smluvních podmínek PID. Důraz byl kladen zejména na konzistenci dat a platnost atributů. Nakonec byly analyzovány statistiky dat ve whitelistu, které poskytly důležité informace o struktuře a charakteristikách jízdních dokladů v PID. Byly identifikovány potenciální nedostatky a možnosti pro budoucí optimalizaci správy dat.

Klíčová slova

Veřejná doprava, jízdní doklady, odbavovací informační systémy, datová analýza, databáze, Python, SQL, JSON

Abstract

This master's thesis deals with the analysis and processing of data in the whitelist of the Prague Integrated Transport System, which contains information about tickets. The goal of the thesis was to convert the extensive whitelist database into a processable format and perform its thorough analysis. For this purpose, the Python programming language and the SQL relational database system were used. The first part of the thesis focused on the technical procedure of converting the whitelist into a database format and basic data analysis. Subsequently, data was checked and validated in terms of meeting the tariff and contractual conditions of PID. Particular emphasis was placed on data consistency and attribute validity. Finally, whitelist data statistics were analysed, which provided important information about the structure and characteristics of tickets in PID. Potential shortcomings and opportunities for future optimization of data management were identified.

Key words

Public transport, tickets, automated fare collection systems, data analysis, database, Python, SQL, JSON

Obsah

Seznam použitých zkratek 5
Úvod 6
1. Whitelist 7
1.1. Struktura whitelistu 8
1.2. Problémy v aktuální podobě whitelistu 12
2. SW řešení zpracování whitelistu 12
2.1. Řešení převodu WL do databáze pomocí Pythonu 12
2.1.1. Příklad kódu pro sekci Contracts added 14
2.1.2. Příklady kódů ostatních sekcí 19
3. Kontrola a zpracování dat z whitelistu 25
3.1. Whitelist databáze 25
3.2. Kontrola dat 28
3.2.1. Hledání duplicit 28
3.2.2. Kontrola dat pro identifikátory a jejich návaznosti 29
3.2.3. Kontrola dat pro kupóny a slevy 31
3.3. Analýza dat ve Whitelistu 34
3.3.1. Validace dat platností kupónů 34
3.3.2. Validace hodnot CP 38
3.3.3. Validace dat platností identifikátorů 43
3.4. Statistiky dat ve Whitelistu 44
3.4.1. Informační analýza klientů 44
3.4.1. Informační analýza identifikátorů 48
3.4.2. Informační analýza kupónů 55
3.4.3. Informační analýza slev 59
3.4.4. Analýza velikosti whitelistu 61
Závěr 63
Použité zdroje 65

Seznam použitých zkratek

CP	Customer profile
CSV	Comma-separated values
ER	Entity-relationship
GUI	Graphical user interface
HW	Hardware
IDSK	Integrovaná doprava Středočeského kraje
ISIC	International Student Identity Card
JSON	Java script object notation
MOS	Multikanálový odbavovací systém
PID	Pražská integrovaná doprava
PVP	Prague visitor pass
ROPID	Regionální organizátor pražské integrované dopravy
SPP	Smluvní přepravní podmínky
SQL	Structured query language
TLV	Type length value
TMS	Terminal management system
TP	Tarif profile
UČZ	Univerzální číslo zákazníka (WLMOSPssngrAcct)
WL	Whitelist

Úvod

Praha má zhruba 1,4 milionu obyvatel a Středočeský kraj má též kolem 1,4 milionu obyvatel. Velká část těchto občanů se v tomto území pohybuje veřejnou dopravou. Jde převážně o metropolitní oblast, kde v rámci veřejné dopravy cestují obyvatelé do a z Prahy anebo po Praze obecně. Ve většině případů těchto cest se dá předpokládat, že jde o pravidelné cesty za dojížděním. S tím souvisí skladba jízdních dokladů. Dá se očekávat, že tito obyvatelé využívají hlavně předplatné časové kupóny pro cesty veřejnou dopravou. Organizátorem veřejné dopravy ve Středočeském kraji je Integrovaná doprava Středočeského kraje a v hlavním městě Praha Regionální organizátor pražské integrované dopravy. Dohromady tvoří jeden systém veřejné dopravy pod názvem Pražská integrovaná doprava. Tento systém zajišťuje jednotný tarif po Praze a dnes už i v celém Středočeském kraji. To znamená, že všechny elektronické předplatné časové kupóny se musí nějakým způsobem datově zpracovávat. Vzhledem k počtu obyvatel a jeho postupného nárůstu v této oblasti je zjevné, že záznamy jízdních dokladů narůstají do již markantních velikostí. Dá se odhadovat, že velikost těchto záznamů bude i nadále růst.

Databáze jízdních dokladů PID, která je nazývána whitelist, již dnes narůstá skoro ke 2 GB dat. Takový nárůst dat je zatěžující. Velikost databáze je náročná, jelikož takovou databázi musí mít v sobě každé kontrolní zařízení PID stažené offline. Navíc stažení dat probíhá ve většině případů bezdrátově pomocí mobilní sítě. Stejně tak zpracování takových dat už klade vyšší nároky na HW.

Cílem této práce je tedy převést databázi do zpracovatelné formy a analyzovat veškerá data. Kromě zefektivnění dat whitelistu je také cílem analytické a statistické zpracování. Data z whitelistu se mohou využít i pro dlouhodobé statistiky využívání veřejné dopravy v dané oblasti. V datech se dají zjistit zajímavé informace o skladbě nosičů jízdních dokladů či typech jízdních dokladů. Součástí práce je kontrola formátu dat a validace dat, zda splňují i tarifní předpoklady systému PID.

Práce je strukturovaná do následujících částí. Nejprve je popsán samotný whitelist, kde je řešeno, jak vypadá, jaká data obsahuje a jaké jsou jeho problémy. Další část řeší převedení whitelistu do zpracovatelné formy. Je vytvořen technický postup pro vytvoření analyzovatelné databáze. Poslední a největší částí je samotná analýza a zpracování dat. Zde jsou data z whitelistu kontrolována, dále jsou data analyzována a srovnána s danými tarifními předpisy PID a nakonec je řešena statistika údajů získaných z whitelistu. Pro toto zpracování byl vybrán konkrétní whitelist vygenerovaný k datu 13. 10. 2022. Důvodem je právně-adekvátní časový odstup pro popisování živých dat z veřejné dopravy v PID. Zároveň jsou veškerá osobní data z whitelistu šifrována a nejsou proto autorovi této práce dostupná.

V kontextu rychle se rozvíjejících městských aglomerací je efektivní a udržitelná veřejná doprava klíčovým prvkem pro zajištění plynulosti a udržitelnosti životního prostředí. Tato diplomová práce by měla přinést určitý přínos v oblasti správy a analýzy dat ve veřejné dopravě. Analytické a statistické zpracování dat whitelistu poskytne důležité informace pro optimalizaci, což může přispět ke zlepšení kvality těchto dopravních dat. Zároveň může práce představovat cenný nástroj pro dlouhodobou strategii udržitelné mobility v metropolitní oblasti, který může sloužit jako vzor pro další města a regiony čelící podobným výzvám veřejné dopravy. Věřím, že výsledky této práce přispějí k lepšímu chápání dopravního systému v Praze a Středočeském kraji, a tím i k celkovému zlepšení kvality života obyvatel této oblasti.

1. Whitelist

Whitelistem se v odbavovacím systému pro Prahu a Středočeský kraj označuje seznam jízdních dokladů vázaných k identifikátoru. Aktuálně platná verze whitelistu je uložena v repozitáři MOS. Takový whitelist je pak stahován jednotlivými revizorskými čtečkami či odbavovacími zařízeními. Stahování je iniciované koncovým zařízením v definované periodě nebo může být vynucené uživatelem koncového zařízení mimo definovanou periodu. Tato komunikace musí probíhat přes šifrovaný protokol a data v koncovém zařízení musí být chráněna autentizací uživatele. [1]

Formát datového whitelistu, který si stahují a čtou koncová zařízení, je TLV. Pro čtení whitelistu mimo odbavovací systém se využívá formát JSON. Aktuální provozní velikost whitelistu pro PID, který obsahuje údaje pro všechny platné identifikátory, se momentálně pohybuje kolem 700 MB. Během životního cyklu systému může velikost absolutního whitelistu narůstat, a odbavovací zařízení musí být schopno zpracovat a pracovat s absolutním whitelistem o velikosti až 2 GB. Aktualizace whitelistu probíhají ve formě přírůstkových dat. Koncová zařízení pravidelně kontrolují nové přírůstky na úložišti MOS, stahují je a automaticky je implementují. Kvalifikovaný odhad průměrného přírůstku za 15 minut se pohybuje v rozmezí 1 kB - 1 500 kB, přičemž průměrná hodnota pro 15minutový whitelist je kolem 40 kB. Základní frekvence aktualizace whitelistu je každých 15 minut. Rozdíly po změně whitelistu nejsou odstraněny po implementaci nových dat do WL, ale jsou zahrnuty do denního celkového inkrementu. Tento denní inkrement je uložen v úložišti MOS a je využit, pokud koncové zařízení vyžaduje aktualizaci whitelistu na dobu přesahující 24 hodin. Tyto konsolidované inkrementy jsou dostupné v hodinové a denní podobě. Přírůstky whitelistu jsou k dispozici až 12 dní zpětně. Pokud má odbavovací zařízení lokální whitelist starší než 12 dní, je nezbytné stáhnout absolutní whitelist. [2]

Whitelisty se rozdělují podle toho, jestli jsou zaměřeny přímo na specifické zařízení, které provádí online komunikaci s MOS, nebo na TMS konkrétního dopravce. Whitelisty pro zařízení nebo TMS obsahují různé složení přenášených dat.

Maska názvu whitelistu je následující: „1_3_2.tlvder“

1 - {id intervalu}

3 - {sekvence}

2 - {předchozí sekvence}

ID intervalu odpovídá následující legendě:

0 – absolutní WL

1 – 15 minut

2 – 60 minut

3 – denní

1.1. Struktura whitelistu

Whitelist určený pro konkrétní zařízení může mít odlišný formát v závislosti na typu zařízení. Delta whitelisty jsou vytvářeny MOS v pravidelných intervalech. Vždy se generuje delta whitelist od posledního vytvoření delta whitelistu. Dále existují hodinové delta whitelisty, které obsahují rozdíly od posledního hodinového vytvoření, a denní delta whitelisty, které obsahují rozdíly od posledního vytvoření denního whitelistu. Absolutní whitelist se vytváří jednou denně. Na serveru jsou k dispozici přírůstky, které poskytují data zpětně o 12 dní. Pokud zařízení nebylo aktualizováno déle než 12 dní, je vždy nutné stáhnout absolutní whitelist. Whitelisty jsou číslovány vzestupně a zahrnují jak pravidelné delta seznamy, tak denní, týdenní a absolutní verze.

Whitelist je složen ze tří sekcí, kterými jsou Uživatelská data, Identifikátory a Kontrakty. Uživatelská data jsou šifrována pomocí univerzálního číslo klienta (WLMOSPssngrAcct). Nosičem se rozumí identifikátor, kterým dnes v PID může být čipová karta Lítačka, In Karta Českých drah, bankovní platební karta a aplikace PID Lítačka v mobilním zařízení. Kontrakty ve whitelistu jsou dvojího typu. Prvním typem jsou průkazy neboli nároky na slevu, což znamená, že jsou do whitelistu například zaneseny slevy typu žákovský průkaz či studentská sleva ISIC. Druhým typem jsou elektronické časové předplatné jízdní doklady (kupóny). Tyto tři sekce obsahují každá tři podsekce Added, Changed a Delete. Kromě těchto datových sekcí obsahuje každý whitelist hlavičku s informacemi o whitelistu a patičku s podpisem. Každá datová sekce whitelistu má své atributy s danými datovými typy. Podrobnější popis struktury je vyobrazen v následujících tabulkách. [2]

Sekce whitelistu		**Atribut**	**Datový typ**	**Whitelist**		**Popis**
				Absolutní	**Delta**	
Clients						**Uživatelská data**
	Added					*Data přibývající oproti předchozímu whitelistu (data v absolutním WL)*
		WLMOSPssngrAcct	Int	X	X	Jedinečný identifikátor účtu klienta UČZ
		PersonalData	Byte	X	X	Šifrovaná část – osobní data
	Changed	*Stejné položky jako u záznamu Added*				*Měněné záznamy (absolutní WL tuto sekci neobsahuje)*
	Deleted					*Data, která se mají smazat (absolutní WL tuto sekci neobsahuje)*
		WLMOSPssngrAcct	Int	-	X	Jedinečný identifikátor účtu klienta UČZ

Tabulka 1 - Struktura uživatelských dat; zdroj: [2] upraveno autorem

Sekce whitelistu		Atribut	Datový typ	Whitelist		Popis
				Absolutní	Delta	
Identifiers						**Data o nosičích**
	Added					*Data přibývající oproti předchozímu whitelistu (data v absolutním WL)*
		VLMOSIdentId	Int	X	X	Jedinečný identifikátor identifikátoru
		WLToken1	Text	X	X	Token nosiče pro zařízení
		WLToken1Ver	Int	X	X	Číslo algoritmu tokenu
		WLCardType	Int	X	X	Typ nosiče: 0 - Lítačka 1 - Bankovní karta 2 - In Karta 3 - Mobilní telefon
		WLCardStatus	Int	X	X	Status identifikátoru: 1 - Aktivní 2 - Expirovaný 3 - Zablokovaný správcem 4 - Zablokovaný cestujícím 5 - Blacklistovaný
		WLCardExpdate	Date	X	X	Datum expirace identifikátoru
		WLMOSPssngrAcct	Int	X	X	Vazba na osobní data klienta (WLMOSPssngrAcct = 0, když nejsou uživatelská data k dispozici)
		WLToken2	Text	X	X	Token 2 nosiče pro zařízení
		WLToken2Ver	Int	X	X	Číslo algoritmu tokenu 2
	Changed	*Stejné položky jako u záznamu Added*				*Měněné záznamy (absolutní WL tuto sekci neobsahuje)*
	Deleted					*Data, která se mají smazat (absolutní WL tuto sekci neobsahuje)*
		VLMOSIdentId	Int	X	X	Jedinečný identifikátor identifikátoru

Tabulka 2 - Struktura identifikátorů; zdroj: [2] upraveno autorem

Sekce whitelistu		Atribut	Datový typ	Whitelist		Popis
				Absolutní	Delta	
Contracts						**Kontrakty**
	Added					*Data přibývající oproti předchozímu whitelistu (data v absolutním WL)*
		ConId	Int	X	X	Jedinečný identifikátor kontraktu
		WLIdentId	Int	X	X	Vazba na identifikátor
		WLValidFrom	Date	X	X	Platnost Od (datum a čas)
		WLValidTo	Date	X	X	Platnost Do (datum a čas)
		Data	Struktura	X	X	Data kontraktu dle typu
	Changed	*Stejné položky jako u záznamu Added*				*Měněné záznamy (absolutní WL tuto sekci neobsahuje)*
	Deleted					*Data, která se mají smazat (absolutní WL tuto sekci neobsahuje)*
		ConId	Int	-	X	Jedinečný identifikátor kontraktu

Tabulka 3 - Struktura kontraktů; zdroj: [2] upraveno autorem

Sekce whitelistu	Atribut	Datový typ	Whitelist		Popis
			Absolutní	Delta	
Header					**Hlavička souboru**
	WLVer	Int	X	X	Verze whitelistu
	WLFormatVer	Int	X	X	Formát WL je dvojciferné číslo XY X: 1 - Absolutní whitelist 2 - Delta Whitelist Y: 1 - Barevné fotografie 2 - Černobíle fotografie
	Seq	Int	X	X	Pořadové číslo whitelistu
	SeqPrev	Int	-	X	Číslo whitelistu, ke kterému se vztahuje delta
	Typ	Int	X	X	Typ Whitelistu podle zařízení pro budoucí rozvoj (0 = defaultní)
	WLScopeTimeTo	Date	X	X	Datum a čas generování
	WLTokenVer	Int	X	X	Číslo algoritmu tokenizace
	WLTest	Char	X	X	Zda se jedná o testovací data
	SigNo	Int	X	X	Číslo verze podpisu
Footer					**Patička souboru**
	Sig	Text	X	X	Podpis

Tabulka 4 - Struktura hlavičky a patičky; zdroj: [2] upraveno autorem

Typ kontraktu	Atribut	Datový typ	Popis	Poznámka
1 Průkaz/Sleva	CP	Int	Customer profile dle číselníku	
	NetworkID	Int	Číslo sítě	
	WLIDType	Int	Způsob ověření 1 - Bez dalšího ověření 2 - Nutné ověřit	Není povinné
	WLIDLogicalNum	Text	Viditelné číslo průkazu podle WLIDType	Není povinné
2 Časový kupón	CP	Int	Customer profile dle číselníku	
	TP	Int	Tarif profile dle číselníku	
	WLZones	Text	Názvy zón, pro které doklad platí	C = celosíťová jízdenka
	WLSupZones	Text	Názvy nadzón, pro které doklad platí	Není povinné
	NetworkID	Int	Číslo sítě, pro který doklad platí dle celostátního číselníku	
	WLIDType	Int	Způsob ověření 1 - Bez dalšího ověření 2 - Nutné ověřit	Není povinné
	WLIDLogicalNum	Text	Viditelné číslo průkazu podle WLIDType	Není povinné

Tabulka 5 - Struktura typů kontraktů; zdroj: [2] upraveno autorem

Definované číselníky ROPID pro atributy CP a TP jsou následující:

CP	Název zákaznického profilu
1	Občanské
2	Dítě 6-15
6	Zvýhodněné
9	Zaměstnanec
26	Hmotná nouze II
27	Invalida III. stupně
30	Zvýhodněná 1/2
31	Dospělý 15+ PVP
32	Dítě 6-15 PVP
33	Junior/student 15-26 PVP
35	Junior 15-18
36	Student 18-26
37	Senior 60-65
40	Sociálně potřební
43	Hmotná nouze
49	Rodinný příslušník
50	Senior 65+
51	Osoba 70+ (dříve)
53	Zvláštní 1/4
58	Doplatek vlak Praha
61	Odboj/PTP/VTPN
63	Přenosné

TP	Název tarifního profilu (časová platnost)
20	Měsíční
21	Čtvrtletní
23	Roční kalendářní
28	Bezplatná
32	Roční
37	48 hodin PVP
38	72 hodin PVP
39	120 hodin PVP

Tabulka 6 - Číselníky ROPID atributů CP a TP; zdroj: autor na základě informací z ROPID

1.2. Problémy v aktuální podobě whitelistu

Jedním z hlavních problémů aktuální podoby whitelistu je velikost absolutního whitelistu, která dosahuje skoro 2 GB a do budoucna bude určitě růst. Problém s neustále rostoucí velikostí whitelistu dále způsobuje sekundární problémy. Mezi ně patří dlouhá nahrávací doba whitelistu do jednotlivých zařízení a k tomu nepřidává ani fakt, že se whitelist nahrává pomocí mobilní telekomunikační sítě, která může být v určitých oblastech nestabilní. Pokud se například validační zařízení dlouhou neaktualizuje, pak musí být nahrán nový absolutní whitelist plné velikosti, který se poté může nahrávat na zařízení v řádech několika minut.

Dalším problémem pro druhotné zpracování dat v podobě statistik a dalších informací pro zlepšení služeb cestujících je samotný formát whitelistu. Jde o strukturovaný formát TLV s příponou tlvder. Pro zobrazení a čtení dat je nutné daný whitelist převést přes MOS na formát JSON. JSON je sice formát vysoce podporovaný a čitelný obecně v běžných zařízeních, ale jedná se formát, který organizuje data do polí čí objektů ve složitých řetězcích. Formát JSON tedy není ideálním čtecím formátem pro naše data. Daný formát navíc dosahující velikosti jednotek gigabajtů stěží otevře uživatel na běžném zařízení. S takto velkým souborem už se velmi těžko pracuje, natož například zpracovává průběžná statistika o cestujících.

2. SW řešení zpracování whitelistu

Z předchozí kapitoly vyplývá, že je potřeba najít vhodný způsob, jak převést data z whitelistu do formy, která bude lépe zpracovatelná na samotnou analýzu a kontrolu dat. Ze struktury whitelistu, jak je definovaná, a z dat, které se ve whitelistu nachází, plyne, že ideálním řešením bude strukturovat data do databáze.

Dalším úkolem je tedy najít vhodný způsob pro převod dat z formátu JSON do databázového formátu. Jelikož absolutní whitelist ve formátu JSON dosahuje velikosti skoro 2 GB, je potřeba nalézt takový způsob, který v rozumném času zvládne převést data do databáze. Při rešerši k problému převodu JSON souboru do databáze byl zvolen jako nejideálnější přístup v podobě využití programovacího jazyka Python. [3]

2.1. Řešení převodu WL do databáze pomocí Pythonu

Pro převedení whitelistu do databáze byl vytvořen skript v programovacím jazyce Python, který byl napsán přesně pro tuto konkrétní potřebu. Skript řeší převod a pracuje se strukturou, jak je popsána ve specifikacích whitelistu. Pro finální výstup převodu se vytváří SQLite databáze s příponou db.

Začátek Python skriptu obsahuje používané knihovny v průběhu kódu. [4]

```
import json
import sqlite3
import os
import tkinter as tk
from tkinter import filedialog
import cProfile
```

Obrázek 1 - Ukázka kódu 1; zdroj: autor

Součástí kódu je pět knihoven. JSON knihovna je importována pro potřeby zpracování JSON souborů. SQLITE3 knihovna je importována pro definování struktury vytvořené databáze ve formátu SQL. Knihovna TKINTER slouží pro vyskakovací okno k výběru konkrétního JSON souboru k převedení do databáze. Poslední importovanou knihovnou je CPROFILE pro vložení profileru, který sleduje průběh kódu. Díky profileru je možné analyzovat, které procedury skriptu jsou paměťově a časově nejnáročnější, a tím pádem je možné efektivně upravovat kód, aby byl co nejefektivnější. [5]

Začátek kódu definuje způsob výběru JSON souboru, který se má převést. Pro pohodlné uživatelské prostředí byla vybrána forma vyskakovacího okna TKINTER. Uživatelovým výběrem souboru je zaznamenána cesta k danému JSON souboru.

```
def get_json_file_path():
    root = tk.Tk()
    root.withdraw() # Skryje hlavní okno tkinter

    # Získání názvu souboru od uživatele pomocí vyskakovacího okna
    json_file_path = filedialog.askopenfilename(title="Vyberte JSON soubor", filetypes=[("JSON files", "*.json")])

    return json_file_path
```

Obrázek 2 - Ukázka kódu 2; zdroj: autor

Dále kód pracuje s názvem vybraného JSON souboru, který je využit jako následný název vytvářeného souboru databáze. Z názvu je odebrána přípona JSON. Kód automaticky vytvoří soubor DB se stejným názvem zvoleného JSON souboru, do kterého jsou pak ukládány v databázové formě data z whitelistu v JSON souboru. [6]

```
# Získání názvu JSON souboru od uživatele
json_file_path = get_json_file_path()

# Odstranění přípony .json z názvu souboru
base_name = os.path.splitext(json_file_path)[0]

# Cesta k SQL souboru
sqlite_db_path = f'{base_name}.db'
```

Obrázek 3 - Ukázka kódu 3; zdroj: autor

Poté se načte konkrétní JSON soubor.

```
# Přečtení JSON souboru
with open(json_file_path, 'r') as file:
    data = json.load(file)
```

Obrázek 4 - Ukázka kódu 4; zdroj: autor

2.1.1. Příklad kódu pro sekci Contracts added

Následuje samotná velká část procedury převodu dat z JSON struktury do databáze. Definuje se zde funkce create_and_populate_db. Atributy této funkce jsou pouze proměnná, ve které je načtený JSON soubor, a cesta k vytvořenému databázovému souboru DB. Součástí této funkce je nadefinování podfunkcí převodu jednotlivých částí whitelistu. V rámci funkce jsou nadefinovány i jednotlivé tabulky v SQL formě.

```
def create_and_populate_db(data, sqlite_db_path):
  # Kód pro vytváření a naplňování databáze
   contracts_new_data = data.get('Contracts.New', []
  # Definování SQL
   drop_table_command = "DROP TABLE IF EXISTS contracts_new;"
   create_table_command = """
   CREATE TABLE contracts_new (
      Id INTEGER PRIMARY KEY AUTOINCREMENT,
      ConId INTEGER,
      WLIdentId INTEGER,
      WLValidFrom DATE,
      WLValidTo DATE,
      DiscountCard_CP INTEGER,
      DiscountCard_NetworkID INTEGER,
      DiscountCard_WLIDType INTEGER,
      DiscountCard_WLIDLogicalNum TEXT,
      TimeCoupon_CP INTEGER,
      TimeCoupon_TP INTEGER,
      TimeCoupon_WLZones TEXT,
      TimeCoupon_WLSupZones TEXT,
      TimeCoupon_NetworkID INTEGER,
      TimeCoupon_WLIDType INTEGER,
      TimeCoupon_WLIDLogicalNum TEXT
   );
   """
```

Obrázek 5 - Ukázka kódu 5; zdroj: autor

Tento kód (Obrázek 5) má za úkol připravit prostředí pro ukládání dat do databáze s názvem 'contracts_new'. Nejprve se snaží získat data s klíčem 'Contracts.New' ze slovníku data, do kterého byl uložen daný soubor JSON s whitelistem. Pokud taková data existují, jsou uložena do proměnné contracts_new_data. V opačném případě je vytvořen prázdný seznam. [3]

Následně je definován SQL příkaz pro odstranění existující tabulky 'contracts_new', pokud existuje. Příkaz DROP TABLE IF EXISTS zajišťuje, že tento krok nepovede k chybě v případě, že tabulka neexistuje.

Další část kódu obsahuje víceřádkový řetězec, který definuje SQL příkaz pro vytvoření nové tabulky 'contracts_new'. Tabulka má několik sloupců, zahrnující například Id, ConId, WLIdentId, WLValidFrom, WLValidTo a další. Tyto sloupce jsou zvoleny dle dané struktury whitelistu. Stejně tak jsou zvoleny i datové formáty pro tyto sloupce. Výjimkou je sloupec Id, který je definován jako primární klíč s automatickým inkrementováním (AUTOINCREMENT), což znamená, že každý nový záznam bude mít jedinečné číslo ID, které se automaticky zvyšuje. Tento postup byl zvolen schválně, pro zjišťování případných duplicitních dat. Kdyby byl zvolen primárním klíčem některý z původních atributů, jako například ConId, tak by při případné duplicitě dat nastala systémová chyba.

Celkově lze tedy říci, že tento kód připravuje strukturu tabulky 'contracts_new' do výsledné databáze a zajišťuje, že tabulka je připravena pro ukládání specifických typů dat do příslušných sloupců.

```
# Vytvoření SQL databáze na disku
    conn = sqlite3.connect(sqlite_db_path)
    cursor = conn.cursor()


    # Smazání staré tabulky, pokud existuje
    cursor.execute(drop_table_command)
    # Vytvoření nové tabulky dle parametrů
    cursor.execute(create_table_command)
```

Obrázek 6 - Ukázka kódu 6; zdroj: autor

Tato část kódu (Obrázek 6) nejprve vytváří připojení k databázi pomocí funkce sqlite3.connect(), kde sqlite_db_path je cesta k souboru, kde má být uložena databáze. Poté se vytváří kurzor, který umožňuje provádět operace s databází.

Dále následují dva kroky. Prvním je smazání staré tabulky, pokud existuje. Tento krok zajišťuje, že pokud v databázi již existuje tabulka se stejným názvem jako ta, která má být vytvořena, bude smazána. To se provádí pomocí funkce cursor.execute() s příkazem drop_table_command, který obsahuje SQL příkaz pro odstranění tabulky. Druhým je vytvoření nové tabulky dle parametrů. Následně se vytváří nová tabulka v databázi podle specifikací definovaných v create_table_command. Tento příkaz opět provádí funkce cursor.execute(), tentokrát s příkazem pro vytvoření tabulky.

```
# Definování funkce na vložení dat do tabulky
def insert_contracts_new(contracts_new):
    # Kontrola 'None' hodnot pro 'TimeCoupon' and 'DiscountCard'
    time_coupon_new = contracts_new['TimeCoupon'] if 'TimeCoupon' in contracts_new and contracts_new['TimeCoupon'] is not None else {}
    discount_card_new = contracts_new['DiscountCard'] if 'DiscountCard' in contracts_new and contracts_new['DiscountCard'] is not None else {}

    # Příprava dat pro vložení
    data_tuple = (
        contracts_new.get('ConId'),
        contracts_new.get('WLIdentId'),
        contracts_new.get('WLValidFrom'),
        contracts_new.get('WLValidTo'),
        discount_card_new.get('Cp'),
        discount_card_new.get('NetworkID'),
        discount_card_new.get('WLIDType'),
        discount_card_new.get('WLIDLogicalNum'),
        time_coupon_new.get('Cp'),
        time_coupon_new.get('Tp'),
        time_coupon_new.get('WLZones'),
        time_coupon_new.get('WLSupZones'),
        time_coupon_new.get('NetworkID'),
        time_coupon_new.get('WLIDType'),
        time_coupon_new.get('WLIDLogicalNum')
    )
```

Obrázek 7 - Ukázka kódu 7; zdroj: autor

Tato část kódu (Obrázek 7) definuje funkci insert_contracts_new, která slouží k vložení dat do tabulky. Nejprve se provádí kontrola, zda jsou ve vstupním slovníku contracts_new klíče 'TimeCoupon' a 'DiscountCard' přítomny a zda nejsou jejich hodnoty None. Pokud jsou tyto podmínky splněny, přiřadí se hodnoty klíčům 'TimeCoupon' a 'DiscountCard' do nových proměnných time_coupon_new a discount_card_new. V opačném případě se pro tyto proměnné nastaví prázdný slovník {}.

Poté následuje příprava dat pro vložení do tabulky. Hodnoty jsou získávány z contracts_new a případně z vnořených slovníků time_coupon_new a discount_card_new pomocí metody

get(). Tyto hodnoty jsou uloženy do data_tuple, který je n-ticí obsahující hodnoty, které budou vloženy do tabulky.

Celkově tato část kódu připravuje data pro vložení do tabulky na základě vstupního slovníku contracts_new a vnořených slovníků time_coupon_new a discount_card_new, které obsahují informace o kupónech a slevových kartách.

```
# Vložení dat - nadefinování
        cursor.execute("""
        INSERT INTO contracts_new (
            ConId,
            WLIdentId,
            WLValidFrom,
            WLValidTo,
            DiscountCard_CP,
            DiscountCard_NetworkID,
            DiscountCard_WLIDType,
            DiscountCard_WLIDLogicalNum,
            TimeCoupon_CP,
            TimeCoupon_TP,
            TimeCoupon_WLZones,
            TimeCoupon_WLSupZones,
            TimeCoupon_NetworkID,
            TimeCoupon_WLIDLogicalNum,
            TimeCoupon_WLIDType
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);
        """, data_tuple)
```

Obrázek 8 - Ukázka kódu 8; zdroj: autor

Tato část kódu (Obrázek 8) provádí vložení dat do tabulky v databázi pomocí SQL příkazu INSERT INTO. SQL příkaz je vykonáván pomocí metody execute() objektu cursor, který představuje kurzor pro práci s databází.

V tomto příkazu jsou uvedeny názvy sloupců, do kterých se budou data vkládat (např. ConId, WLIdentId, atd.), a to v rámci tabulky contracts_new. Následuje klíčové slovo VALUES, které definuje hodnoty, které se mají vložit. Místo samotných hodnot jsou zde použity otazníky (?), což je způsob, jakým se v SQL příkazu označují parametry. Tato syntaxe je často používána pro zabránění SQL injection útoků. [5]

Parametry pro vložení jsou předány jako druhý argument metody execute() ve formě n-tice data_tuple. Každá hodnota v n-tici bude vložena na místo odpovídajícího otazníku v SQL příkazu.

Celkově tato část kódu provádí vložení dat uložených v n-tici data_tuple do tabulky contracts_new v databázi pomocí SQL příkazu INSERT INTO.

```
# Vložení do tabulky
    for contracts_new in contracts_new_data:
        insert_contracts_new(contracts_new)


    # Commit
    conn.commit()
```

Obrázek 9 - Ukázka kódu 9; zdroj: autor

Nejprve je kód (Obrázek 9) iterován přes seznam contracts_new_data, který obsahuje data, jež mají být vložena do tabulky. Pro každý prvek tohoto seznamu je volána funkce insert_contracts_new(contracts_new), která je zodpovědná za vložení jednoho záznamu do tabulky.

Funkce insert_contracts_new() byla již dříve definována a připravuje data pro vložení do tabulky na základě vstupních parametrů.

Poté následuje příkaz conn.commit(). Objekt conn představuje spojení (connection) k databázi, a metoda commit() zajišťuje, že všechny provedené změny v databázi (v tomto případě vložení dat) budou trvale zapsány. Tím se zajišťuje konzistence dat a zároveň se zamezí ztrátě změn v případě chyb nebo pádu programu.

Celkově tato část kódu iteruje přes seznam dat a postupně je vkládá do tabulky pomocí funkce insert_contracts_new(). Po dokončení vkládání dat je proveden commit, který zajišťuje trvalé uložení provedených změn v databázi.

2.1.2. Příklady kódů ostatních sekcí

Jak vyplývá z popsané struktury v první kapitole, tak whitelist má tři sekce Contracts, Identifiers a Clients, respektive kupóny, identifikátory a zákazníci. Každá tato sekce je ve whitelistu ještě rozdělována na added, changed a delete, respektive přidané, změněné a smazané. V předchozí části je popsán kód pro převod Contracts added, který je strukturně nejkomplikovanější. Oproti ostatním sekcím je zde vnořená podstruktura, kde jsou jednak slevové karty a jednak kupóny. Tento fakt se tedy také musel zapracovat do kódu. U ostatních sekcí nebylo potřeba toto řešit.

Důležité je zmínit, že každá podsekce changed je vždy totožná s podsekcí added. Takže zde se kódy shodují. Odlišný kód je u podsekcí delete, kde je vždy uvedeno jen dané ID zápisu

ke smazání (pro Contracts ConId, pro Intentifiers VLMOSIdentId a pro Clients WLMOSPssngrAcct).

U ostatních sekcí je kód vždy intuitivně obdobný. Jedinými změnami, kromě zmíněných, jsou jednotlivé atributy struktury sekcí whitelistu. Drobná změna oproti definované struktuře whitelistu je u sekce Clients, kde druhý ze dvou atributů, samotná osobní data, je šifrovaný. Místo zašifrované hodnoty je do databáze převáděna jen hodnota označující délku zašifrované hodnoty a je tím vytvářen nový atribut PersonalDataLength. Kód je následující (Obrázek 10). Pro ukázku je vložena část kódu pro část Clients changed, aby bylo vidět, že postup funguje stejně dobře i pro část 'changed', která je formátovaná vždy stejně jako 'added'.

Ukázková část kódu (Obrázek 11) pro vybranou 'delete' podsekci se zabývá vytvořením a naplněním databáze pomocí identifikátorů a příslušných údajů.

Nejprve jsou získána data pro odstranění identifikátorů. Tato data jsou získána z objektu data pod klíčem 'Identifiers.Delete'. Pokud nejsou žádná data k dispozici, nastaví se výchozí hodnota jako prázdný seznam [].

Dalším krokem je definování SQL příkazů pro vytvoření a naplnění tabulky. Proměnná drop_table_command obsahuje SQL příkaz pro odstranění tabulky identifiers_delete, pokud již existuje. Proměnná create_table_command obsahuje SQL příkaz pro vytvoření nové tabulky identifiers_delete. Tato tabulka obsahuje nově vytvořený sloupec Id s primárním klíčem a sloupec VLMOSIdentId typu INTEGER. Nové Id je zde opět zavedeno čistě pro účely případné kontroly, aby bylo možno odhalit nežádoucí duplicity ve whitelistu. [7]

Po definování SQL příkazů se vytvoří připojení k SQLite databázi pomocí funkce sqlite3.connect(). Poté se vytvoří kurzor, který umožňuje provádět operace s databází. Stará tabulka se smaže, pokud existuje, pomocí příkazu cursor.execute(drop_table_command). Poté se vytvoří nová tabulka podle definice v create_table_command pomocí cursor.execute(create_table_command).

Následuje iterace přes seznam identifiers_delete_data, kde každý prvek je vložen do tabulky identifiers_delete. To se provádí pomocí SQL příkazu INSERT INTO, kde se vkládá hodnota VLMOSIdentId pro každý prvek.

Nakonec jsou změny potvrzeny pomocí metody conn.commit(), což trvale zaznamená změny v databázi.

```
# Kód pro vytváření a naplňování databáze
    clients_change_data = data.get('Clients.Change', [])
   # Definování SQL
    drop_table_command = "DROP TABLE IF EXISTS clients_change;"
    create_table_command = """
    CREATE TABLE clients_change (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        WLMOSPssngrAcct INTEGER,
        PersonalDataLength INTEGER ); """
     # Vytvoření SQL databáze na disku
    conn = sqlite3.connect(sqlite_db_path)
    cursor = conn.cursor()
    # Smazání staré tabulky, pokud existuje
    cursor.execute(drop_table_command)
    # Vytvoření nové tabulky dle parametrů
    cursor.execute(create_table_command)
    # Definování funkce na vložení dat do tabulky
    def insert_clients_change(clients_change):
        # Získání délky řetězce PersonalData
        personal_data_length = len(json.dumps(clients_new.get('PersonalData')))
        # Příprava dat pro vložení
        data_tuple = (
        clients_new.get('WLMOSPssngrAcct'),
        personal_data_length)
        # Vložení dat - nadefinování
        cursor.execute("""
        INSERT INTO clients_change (
            WLMOSPssngrAcct,
            PersonalDataLength
        ) VALUES (?, ?);
        """, data_tuple)
    # Vložení do tabulky
    for clients_change in clients_change_data:
        insert_clients_change(clients_change)
    # Commit
    conn.commit()
```

Obrázek 10 - Ukázka kódu 10; zdroj: autor

```
# Kód pro vytváření a naplňování databáze
identifiers_delete_data = data.get('Identifiers.Delete', [])
# Definování SQL
drop_table_command = "DROP TABLE IF EXISTS identifiers_delete;"
create_table_command = """
CREATE TABLE identifiers_delete (
   Id INTEGER PRIMARY KEY AUTOINCREMENT,
   VLMOSIdentId INTEGER
   );
   """
# Vytvoření SQL databáze na disku
conn = sqlite3.connect(sqlite_db_path)
cursor = conn.cursor()
# Smazání staré tabulky, pokud existuje
cursor.execute(drop_table_command)
# Vytvoření nové tabulky dle parametrů
cursor.execute(create_table_command)
# Vložení do tabulky
for identifiers_delete in identifiers_delete_data:
   cursor.execute('INSERT INTO identifiers_delete (VLMOSIdentId) VALUES (?)', (identifiers_delete,))
# Commit the changes to the database
conn.commit()
```

Obrázek 11 - Ukázka kódu 11; zdroj: autor

Závěrem je důležité zmínit, že kromě hlavních sekcí k převodu do databáze kód také řeší převedení hlavičky whitelistu a zápatí whitelistu. Vytváří se poslední tabulka, do které jsou vkládána data z hlavičky a zápatí. Tím se celý kód definování funkce převodu JSON do databáze ve skriptu uzavírá a nakonec se celá tato funkce zavolá. [8]

```
# Kód pro vytváření a naplňování databáze
    header_data = data.get('Header', {})
    footer_data = data.get('Footer', {})

   # Definování SQL
    drop_table_command = "DROP TABLE IF EXISTS header;"
    create_table_command = """
    CREATE TABLE header (
        Id INTEGER PRIMARY KEY AUTOINCREMENT,
        WLVer               INTEGER,
        WLFormatVer         INTEGER,
        Seq                 INTEGER,
        SeqPrev             INTEGER,
        WLScopeTimeTo       DATE,
        WLTokenVer          INTEGER,
        SigNo               INTEGER,
        Sig                 TEXT
    );
    """
     # Vytvoření SQL databáze na disku
    conn = sqlite3.connect(sqlite_db_path)
    cursor = conn.cursor()

    # Smazání staré tabulky, pokud existuje
    cursor.execute(drop_table_command)
    # Vytvoření nové tabulky dle parametrů
    cursor.execute(create_table_command)
```

Obrázek 12 - Ukázka kódu 12; zdroj: autor

```
# Definování funkce na vložení dat do tabulky
    def insert_data(header, footer):
        # Příprava dat pro vložení
        data_tuple = (
            header.get('WLVer'),
            header.get('WLFormatVer'),
            header.get('Seq'),
            header.get('SeqPrev'),
            header.get('WLScopeTimeTo'),
            header.get('WLTokenVer'),
            header.get('WLTest'),
            header.get('SigNo'),
            footer.get('Sig'))
        # Vložení dat - nadefinování
        cursor.execute("""
        INSERT INTO header (
        WLVer,
        WLFormatVer,
        Seq,
        SeqPrev,
        WLScopeTimeTo,
        WLTokenVer,
        WLTest,
        SigNo,
        Sig
        ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?);
        """, data_tuple)
    # Vložení do tabulky
    insert_data(header_data, footer_data)
    # Commit
    conn.commit()
    conn.close()
# Zavolání funkce
create_and_populate_db(data, sqlite_db_path)
```

Obrázek 13 - Ukázka kódu 13; zdroj: autor

Závěrečné části kódu (Obrázek 12 a Obrázek 13) se zabývají vytvářením a naplňováním databáze s ohledem na hlavičková data a data ze zápatí.

Nejprve jsou získána data pro hlavičku a zápatí z objektu 'data'. Tato data se získávají pod klíči 'Header' a 'Footer'. Pokud dané klíče nejsou ve slovníku data přítomny, nastaví se výchozí hodnoty prázdných slovníků {}.

Následně jsou definovány SQL příkazy pro vytvoření tabulky 'header'. Proměnná drop_table_command obsahuje SQL příkaz pro odstranění tabulky 'header', pokud již existuje. Proměnná create_table_command obsahuje SQL příkaz pro vytvoření nové tabulky 'header' s příslušnými sloupci.

Po definování SQL příkazů se vytvoří připojení k SQLite databázi a kurzor pro práci s databází. Vytvoří se nová tabulka podle definice v create_table_command.

Definuje se funkce insert_data(header, footer), která připraví data pro vložení do tabulky a provede vložení těchto dat pomocí SQL příkazu INSERT INTO. Data pro vložení jsou předána jako parametry funkce insert_data.

Nakonec jsou změny potvrzeny pomocí metody commit(), což trvale zaznamená změny v databázi. Po provedení všech operací je spojení s databází uzavřeno pomocí metody close(). Tím je uvolněna ze zdroje a je zajištěna správná práce s pamětí.

Na závěr je zavolána výsledná funkce celkového převodu všech dat z whitelistu z JSON formátu do nově vytvořené databáze.

3. Kontrola a zpracování dat z whitelistu

Po úspěšném zpracování whitelistu pomocí programovacího jazyka Python do podoby databázového souboru je potřeba zkontrolovat data, validovat veškeré hodnoty a zpracovat data do podoby dopravně-statistických informací. K účelu kontroly dat, byla napsána série SQL dotazů, která zjišťuje správnost a korektnost dat. K tomuto celému procesu byl zvolen program DB Browser for SQLite na práci s databází. [9]

3.1. Whitelist databáze

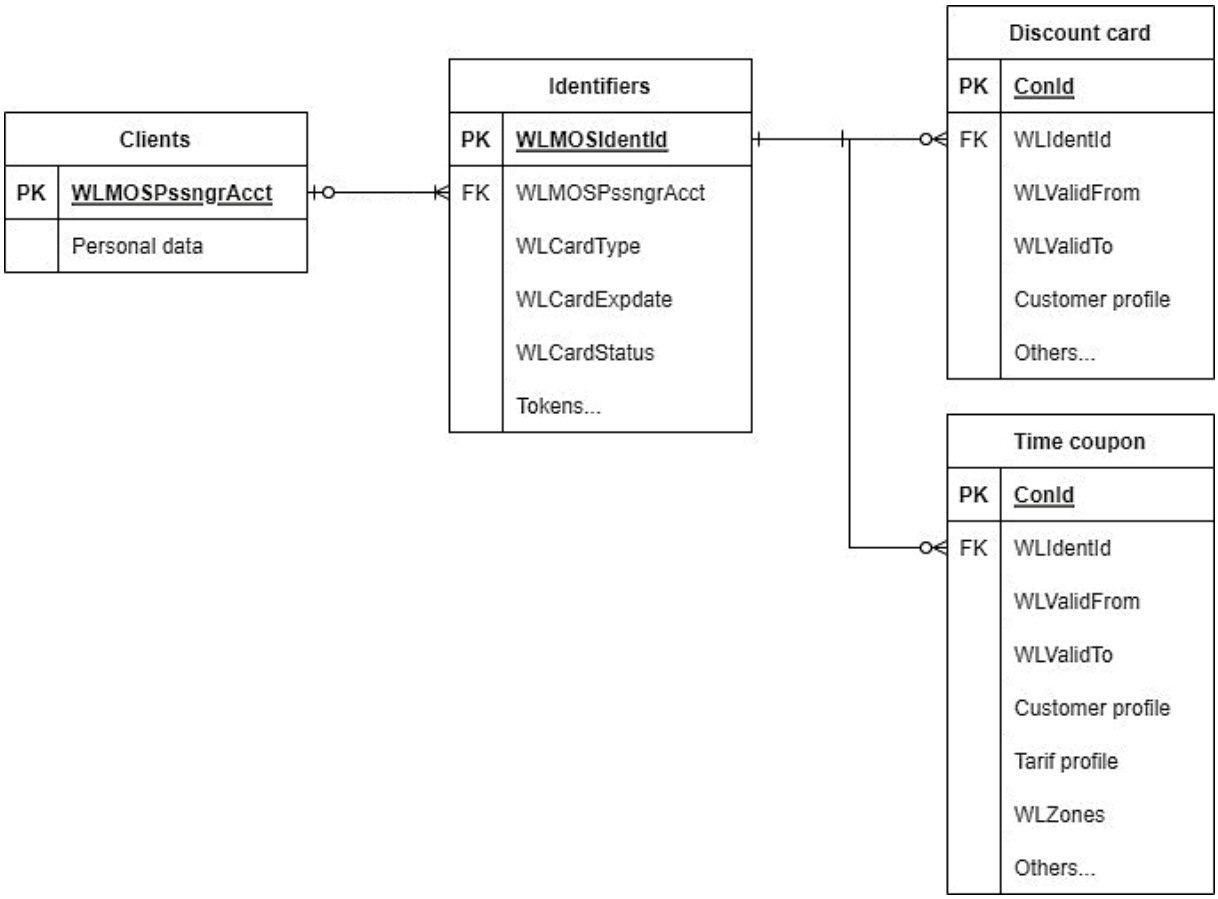
Dle struktury whitelistu jsou údaje o slevách a o časových kupónech v jedné strukturní sekci Contracts. Pro korektní databázi bylo tedy nutné tuto část rozdělit. K tomu byl sepsán následný SQL příkaz.

```
CREATE TABLE DiscountCard_new (
    ConId INTEGER PRIMARY KEY,
    WLIdentId INTEGER,
    WLValidFrom DATE,
    WLValidTo DATE,
    DiscountCard_CP INTEGER,
    DiscountCard_NetworkID INTEGER,
    DiscountCard_WLIDType INTEGER,
    DiscountCard_WLIDLogicalNum TEXT
);
INSERT INTO DiscountCard_new (ConId, WLIdentId, WLValidFrom, WLValidTo,
DiscountCard_CP, DiscountCard_NetworkID, DiscountCard_WLIDType,
DiscountCard_WLIDLogicalNum)
SELECT ConId, WLIdentId, WLValidFrom, WLValidTo, DiscountCard_CP,
DiscountCard_NetworkID, DiscountCard_WLIDType, DiscountCard_WLIDLogicalNum FROM
contracts_new WHERE DiscountCard_CP is not NULL

CREATE TABLE TimeCoupon_new (
    ConId INTEGER PRIMARY KEY,
    WLIdentId INTEGER,
    WLValidFrom DATE,
    WLValidTo DATE,
    TimeCoupon_CP INTEGER,
    TimeCoupon_TP INTEGER,
    TimeCoupon_WLZones TEXT,
    TimeCoupon_WLSupZones TEXT,
    TimeCoupon_NetworkID INTEGER,
    TimeCoupon_WLIDType INTEGER,
    TimeCoupon_WLIDLogicalNum TEXT
);
INSERT INTO TimeCoupon_new (ConId, WLIdentId, WLValidFrom, WLValidTo,
TimeCoupon_CP, TimeCoupon_TP, TimeCoupon_WLZones, TimeCoupon_WLSupZones,
TimeCoupon_NetworkID, TimeCoupon_WLIDType, TimeCoupon_WLIDLogicalNum)
SELECT ConId, WLIdentId, WLValidFrom, WLValidTo, TimeCoupon_CP, TimeCoupon_TP,
TimeCoupon_WLZones, TimeCoupon_WLSupZones,
TimeCoupon_NetworkID, TimeCoupon_WLIDType, TimeCoupon_WLIDLogicalNum FROM
contracts_new WHERE TimeCoupon_CP is not NULL
```

Obrázek 14 - SQL příkaz na rozdělení tabulky; zdroj: autor

Díky tomuto příkazu bylo dovršeno vytvoření databáze v korektní formě. Bylo zkontrolováno, že počet záznamů sečtených z obou nových tabulek se rovná počtu záznamů z původní jedné tabulky. Pro přehlednost databáze byl vytvořen ER diagram zobrazující schéma databáze.



Obrázek 15 - ER diagram whitelist databáze; zdroj: autor

V diagramu (Obrázek 15) jsou vidět jednotlivé entity. Těmi jsou Clients, Indentifiers, Discount card a Time coupon. Mezi atributy jsou zařazené ty, které mají pro práci informativní hodnotu, a ostatní jsou zobrazeny jako poslední s třemi tečkami. Je zde vidět, že hlavní entita je zde identifikátor, jehož ID je primárním klíčem. Některý identifikátor vyžaduje návaznost na osobní data. Například v případě nosiče ve formě bankovní karty nebo mobilního telefonu může být v databázi návaznost na osobní data. Vazba 1:N vychází z předpokladu, že osobní data se ve whitelistu zachovávají, i když se mění identifikátor. Dále jsou zde vidět slevy a časové kupóny, které jsou vázány na identifikátor. Opět platí vazba 1:N, protože uživatel může mít více časových kupónů a obdobně i slev. Zde nejsou vazby mandatorní, protože identifikátor nemusí mít návaznost na slevu, když má návaznost alespoň na kupón, a obráceně. Jelikož je ale mandatorní, aby nějakou takovou vazbu měl, tak je na vazbě vyobrazena krátká vertikála, která tuto mandatorní spojitost znázorňuje. Platí tedy fakt, že samotný identifikátor bez návaznosti na kupón či slevu by v absolutním whitelistu neměl být.

3.2. Kontrola dat

Kontrola dat probíhala na whitelistu vygenerovaném 13. 10. 2022 v 00:07. Jde o absolutní whitelist, který tedy má zaplněné jen sekce 'new'. Tudíž jen v daných sekcích probíhá kontrola dat. Vypsané SQL dotazy na vyhledání chyb jsou ale lehce upravitelné do podoby kontroly daných chyb i v delta whitelistech. Daný whitelist má přesně 254 870 záznamů s osobními daty, 994 773 záznamů identifikátorů a 1 492 933 záznamů kontraktů, kde časové kupóny tvoří 921 220 záznamů a slevy tvoří 571 713 záznamů.

3.2.1. Hledání duplicit

K hledání duplicit byl využit tento jednoduchý dotaz.

```
SELECT WLMOSIdentId
FROM identifiers_new
GROUP BY WLMOSIdentId
HAVING COUNT(WLMOSIdentId) > 1;
```

Obrázek 16 - Hledání duplicit; zdroj: autor

Tento dotaz se lehce upraví pro jakýkoli ze tří ID, které by mělo být vždy entitním primárním klíčem a tedy unikátní. Jde o WLMOSIdentId, WLMOSPssngrAcct a ConId. Jen u WLMOSIdentId byla nalezena duplicita.

WLMOSIdentId
1620683

Obrázek 17 - Výsledek SQL dotazu – nalezená duplicita; zdroj: autor

Pro daný identifikátor byl nalezen záznam, který byl opravdu duplicitní.

```
SELECT * FROM identifiers_new
WHERE WLMOSIdentId = 1620683;
```

Obrázek 18 - SQL dotaz na daný duplicitní záznam; zdroj: autor

Id	WLMOSIdentId	WLToken1	WLToken1Ver	WLCardType	WLCardStatus	WLCardExpdate	WLMOSPssngrAct	WLToken2	WLToken2Ver
203959	1620683	cV295Jj4yMdui9EuMKFYv7139+cDBRdRI7ebAUP5YWM=	1	0	1	2023-10-05 00:00:00	0	mInWXLG0uohL0w1hRsLk2HXJ7m3FRFzWIXJNbz3ldnw=	2
277569	1620683	tUbxdCmaIYdmEI6wxvtsf14+p90GeQW4oIHylw52kLk=	1	0	1	2023-10-05 00:00:00	0	viex4lDVs84x9PNHh8trLDEoZQ5Qhs7TY+DOwsuKlLM=	2

Tabulka 7 - Odpověď na SQL dotaz k vyhledání duplicitního záznamu; zdroj: autor na základě dat z WL

Z nalezeného duplicitního záznamu (Tabulka 7) jde vyčíst pár informací. Oba mají odlišné Id, což byl uměle přidaný primární klíč formou autoinkrementu při průběhu převádění whitelistu do databáze pomocí jazyka Python. Díky tomu byl v databázi zachován původní chybný duplicitní záznam. Dále zde je totožné ID identifikátoru, které by správně mělo být primárním klíčem v databázi. Poté se v záznamu vyskytují dvojice tokenu, kde každý je jiný. Nakonec zde máme souhlasné informace, že jde o nosič v podobě karty Lítačky (WLCardType = 0), nosič je aktivní (WLCardStatus = 1), není zde propojení k osobním datům (WLMOSPssngrAcct = 0 – což je správně, když jde o Lítačku) a nakonec datum expirace (WLCardExpdate) nosiče je 5. 10. 2023.

3.2.2. Kontrola dat pro identifikátory a jejich návaznosti

První z nutných ověření je, zda data odpovídají předepsané struktuře. To znamená, že bez ohledu na správnost dat se zjišťuje, že data odpovídají předepsanému oboru hodnot. Pro tyto účely byl vytvořen tento SQL dotaz.

```
SELECT COUNT(*) as kontrola_hodnot
FROM identifiers_new
WHERE WLCardStatus=1 and
LENGTH(WLToken1) = 44 and
LENGTH(WLToken2) = 44 and
WLToken1Ver=1 and
WLToken2Ver=2;
```

Obrázek 19 - Kontrola hodnot identifiers; zdroj: autor

V kódu (Obrázek 19) je vidět, hodnoty by měli splňovat následující kritéria. Oba tokeny mají 44 znaků, všechny identifikátory v záznamech jsou aktivní (WLCardStatus = 1), hodnota WLToken1Ver se rovná 1 a hodnota WLToken2Ver se rovná 2. Poslední dva jmenované jsou čísla algoritmů.

Provedením daného dotazu bylo zjištěno, že 1 694 z celkových 994 773 záznamů této struktuře neodpovídá. Po dohledání chybných hodnot bylo zjištěno, že jde vždy o stejnou

chybu. Problém je v hodnotách WLToken1Ver a WLToken2Ver, kdy v chybných záznamech se obě hodnoty vždy rovnají buď číslu 2, nebo 1. Tyto chybné hodnoty lze najít následujícím dotazem.

```
SELECT *
FROM identifiers_new
WHERE WLToken1Ver<>1 or
WLToken2Ver<>2;
```

Obrázek 20 - Vyhledání chybných tokenů; zdroj: autor

Entita Clients má pouze dva atributy WLMOSPssngrAcct a PersonalData. PersonalData jsou v databázi nahrazena atributem PersonalDataLength, jelikož přímá osobní data, která by se měla nacházet v daném sloupci, jsou šifrovaná. Proto pro získání alespoň nějakých údajů je zde atribut, který zjistí počet hodnot v původním atributu PersonalData. I kvůli rychlejšímu převodu z původního JSON souboru do databáze byla tato data nahrazena hodnotami o délce textového řetězce.

Byl vytvořen SQL dotaz, který zjistí, že všechny nenulové zápisy WLMOSPssngrAcct v entitě Identifiers mají svůj ekvivalentní zápis v entitě Clients. Důležité je myslet na fakt, že hodnota WLMOSPssngrAcct u Identifiers je neunikátní.

```
SELECT COUNT(DISTINCT
i.WLMOSPssngrAcct)
FROM identifiers_new i, clients_new c
WHERE
c.WLMOSPssngrAcct=i.WLMOSPssngr
Acct;
```

Obrázek 21 - Kontrola propojení WLMOSPssngrAcct; zdroj: autor

V kódu (Obrázek 21) je vidět, že se hledá počet unikátních zápisů WLMOSPssngrAcct pomocí funkce distinct, kde musí platit fakt, že hodnoty se v obou tabulkách rovnají. Daný počet vyšel 254 870, což je i počet zápisů v tabulce osobních dat. Při vynechání funkce distinct se dojde k počtu 300 209, což je počet všech záznamů identifikátorů s nenulovým WLMOSPssngrAcct. Je tedy příkladně vidět, že se na jednoho klienta (jedny osobní data) vztahuje více identifikátorů.

Obdobným způsobem lze zkontrolovat návaznost tabulky identifikátorů s entitou Contracts. Zde je nutno zmínit, že dle specifikací se atribut s hodnotou identifikačního čísla pro identifikátor v tabulce Contracts jmenuje WLIdentId.

```
SELECT COUNT(i.WLMOSIdentId)
FROM identifiers_new i, contracts_new c
WHERE i.WLMOSIdentId=c.WLIdentId;
```

Obrázek 22 - Kontrola propojení WLIdentId; zdroj: autor

Po použití tohoto kódu (Obrázek 22) byla výsledná hodnota na počtu 1 492 934, což je odpovídající hodnota k počtu záznamů všech kontraktů. Poznámkou je, že je zde započítán záznam navíc, jelikož se dvakrát počítá záznam spojený s chybným duplicitním záznamem u identifikátorů zaznamenaný v minulé kapitole. Když je zde lehce pozměněn kód, že je přidána funkce distinct, tak je počet 994 772, což je počet všech záznamů identifikátoru bez dané duplicity. Z těchto počtů se potvrzuje, že žádný záznam kontraktu není bez svého identifikátoru, a naopak žádný identifikátor, který by neměl návaznost na kupón či slevu, není uveden v daném whitelistu.

3.2.3. Kontrola dat pro kupóny a slevy

Nejprve je opět potřeba správným dotazem zjistit, zda data v databázi v těchto entitách vůbec splňují strukturní podmínky bez ohledu na informační správnost dat. První SQL dotazy řeší ty okrajové atributy, které nemají informační hodnotu, ale jsou důležité pro ověření strukturální integrity dat.

```
SELECT COUNT(*)
FROM DiscountCard_new
WHERE DiscountCard_NetworkID=203111
and DiscountCard_WLIDLogicalNum is null
and DiscountCard_WLIDType is null;
```

Obrázek 23 - Kontrola dat slevy; zdroj: autor

Zde je v kódu (Obrázek 23) vidět, že se ověřuje počet dat, které splňují podmínky, že NetworkID je 203111 a atributy WLIDLogigalNum a WLIDType jsou nulové. NetworkID je číslo sítě a pro PID je vždy 203111. Tímto dotazem bylo zjištěno, že všechny záznamy dané tabulky v kontrolovaném whitelistu splňují tyto podmínky. Obdobně pak vypadá i SQL dotaz pro časové kupóny.

```
SELECT COUNT(*)
FROM TimeCoupon_new
WHERE TimeCoupon_NetworkID=203111
and TimeCoupon_WLIDType is null and
TimeCoupon_WLSupZones is null and
TimeCoupon_WLIDLogicalNum is null;
```

Obrázek 24 - Kontrola dat kupóny; zdroj: autor

Zde (Obrázek 24) je kód velice podobný. Opět je zde tázáno na číslo sítě 203111 a jsou zde oba další atributy. Rozdíl je zde v tom, že přibývá kontrola u atributu WLSupZones s dotazem, zda je taktéž nulový. Jde o atribut vyjadřující takzvané nadzóny. Jelikož se tento údaj v PID nepoužívá, tak se ověřuje, zda je nulový. Tímto dotazem bylo opět stoprocentně ověřeno, že všechny záznamy z tabulky kupónů pro daný kontrolovaný whitelist tyto podmínky splňují.

Další z kontrol je ověření, že hodnoty atributů CP (customer profile) a TP (tarif profile) nabývají pouze hodnot, které jsou předepsány číselníky dle struktury whitelistu.

```
SELECT COUNT(*)
FROM contracts_new
WHERE DiscountCard_CP IN
(1,2,6,9,26,27,30,31,32,33,35,36,37,40,43
,49,50,51,53,58,61,63) OR
TimeCoupon_CP IN
(1,2,6,9,26,27,30,31,32,33,35,36,37,40,43
,49,50,51,53,58,61,63);
```

Obrázek 25 - Kontrola hodnot CP; zdroj: autor

V dotazu (Obrázek 25) je vždy vypsaná daná množina hodnot dle číselníku a je dotázáno, zda se hodnoty daných atributů vyskytují v této množině. Dotaz byl proveden nad celou původní entitou Contracts kvůli efektivnosti. Pokud se kontrolují hodnoty nacházející se jak ve slevách, tak v kupónech, je z časové náročnosti dotazů efektivnější se ptát nad celou původní entitu Contracts. Stejným způsobem je kontrolován atribut TP u kupónů.

```
SELECT COUNT(*)
FROM TimeCoupon_new
WHERE TimeCoupon_TP IN
(20,21,23,28,32,37,38,39);
```

Obrázek 26 - Kontrola hodnot TP; zdroj: autor

Poslední ze strukturálních kontrol v hodnotách atributů je atribut TimeCoupon_WLZones. Jde tedy o kombinace zón, kde daný časový kupón platí. V PID platí 12 vnějších zón označených

číselně. Dále jsou zde 4 vnitřní pražská pásma označené P, 0 a B, kde P je dvojí pásmo. Korektní forma kupónu tedy musí mít alespoň jedno z těchto pásem nebo správnou kombinaci, kdy kombinovaná pásma na sebe musí navazovat. Další kritérium časových kupónů pro Prahu je, že lze zakoupit pouze celé pražské čtyřpásmí. Pro region jsou možnosti následující. Lze zakoupit libovolnou kombinaci navazujících pásem od počtu jednoho do šesti. Předplatné jízdné v regionu se zásahem do pražských pásem lze koupit pouze do pásem 0 a B, kde jsou obě počítána jako jedno a ve whitelistu je značen pouze číslicí 0. Dále lze koupit předplatné jízdné od pásma 0 po 6, od pásma 0 po 12 (aktuálně po 13 ode dne vyhlášení) a od pásma 1 po 12. Nicméně při zkoumání hodnot v analyzované databázi whitelistu z roku 2022 jsou i kombinace od pásma 0 po pásmo 9 či od pásma 1 po pásmo 7. Takže je brán předpoklad, že tehdy šlo kombinovat pásma libovolně a jedinou podmínkou je návaznost pásem. Tudíž podmínka tohoto atributu je buď hodnoty pro Prahu P;0;B či občas v databázi psáno jen P;0 (historicky) anebo hodnoty pro region od 0 po 12. Zásadní problém v kontrole správnosti těchto kombinací je, že hodnoty jsou psány pro region z neznámého důvodu abecedně. To značně komplikuje případné SQL dotazy. [10]

```
SELECT COUNT(*)
FROM TimeCoupon_new
WHERE TimeCoupon_WLZones NOT LIKE
'%[^P0B123456789;]%'
-- Hodnota obsahuje znaky, které jsou v seznamu
AND TimeCoupon_WLZones NOT LIKE '%;;%'
-- Žádné dva středníky za sebou
AND TimeCoupon_WLZones NOT LIKE ';%'
AND TimeCoupon_WLZones NOT LIKE '%;'
-- Žádný středník na začátku nebo na konci hodnoty
AND (TimeCoupon_WLZones LIKE '%P;0;B%' OR
TimeCoupon_WLZones LIKE '%P;0%')
-- Pouze pražská pásma ve správně zapsaných
kombinacích
```

Obrázek 27 - Dotaz na korektní formu pásem pro Prahu; zdroj: autor

V dotazu (Obrázek 27) je vidět, jak se vyhledají záznamy s korektní formou zápisu kombinace pásem. Formu popsanou v dotazu splňuje 810 091 záznamů. Přidáním negace k podmínce „pouze pražská pásma“ se dá zjistit počet záznamů kupónů do regionu v této korektní formě. Poté je počet záznamů 111 129. Součet dává číslo 921 220, což je i plný počet všech záznamů kupónů v databázi. Bohužel tento dotaz neřeší, zda jsou pásma v záznamech správně navazující. Vzhledem k problému abecedního zápisu je pak dotaz značně komplikovaný. Dotaz, který by byl schopný tuto podmínku ověřit, již bohužel nesplňuje syntax jazyka SQLite. Nicméně vypadat by mohl následovně (Obrázek 28).

Tímto dotazem končí kontrola, zda hodnoty v databázi splňují korektní strukturální definice atributů a lze přejít k věcným kontrolám samotných informací, které z dat lze vyčíst. Z předchozích dotazů lze vyčíst, že 87,94 % všech předplatných kupónů v kontrolované databáze tohoto absolutního whitelistu z roku 2022 jsou kupóny pro Prahu. A pouze zbylých 12,06 % jsou předplatné kupóny pro region. [11]

Poslední poznámkou je, že dle definované struktury whitelistu by měla celosíťová jízdenka od pásma P po pásmo 12 být značena písmenem „C“, nicméně tato hodnota v kontrolovaném whitelistu nalezena nebyla.

```
... -- Začátek stejný
AND NOT(TimeCoupon_WLZones LIKE '%P;0;B%' OR TimeCoupon_WLZones LIKE '%P;0%')
-- Pouze mimopražská pásma ve správně zapsaných kombinacích
AND(
  -- Kontrola, zda jsou všechny hodnoty ve správném pořadí a v rozmezí 0 až 12
  -- První poddotaz: kontrola první hodnoty
  SUBSTRING_INDEX(TimeCoupon_WLZones, ';', 1) IN ('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12')
  AND (
    -- Další poddotazy: kontrola, zda následující hodnoty jdou za sebou
    SELECT COUNT(*)
    FROM (
      -- Rozdělení hodnot pomocí středníku a zjištění jejich pořadí
      SELECT CAST(value AS UNSIGNED) AS num,
        @prev := @current AS prev,
        @current := CAST(value AS UNSIGNED) AS current
      FROM (
        SELECT SUBSTRING_INDEX(SUBSTRING_INDEX(TimeCoupon_WLZones, ';', numbers.n), ';', -1) AS value
        FROM
        (SELECT 1 n UNION ALL SELECT 2 UNION ALL SELECT 3 UNION ALL SELECT 4 UNION ALL SELECT 5 UNION ALL SELECT 6 UNION
ALL SELECT 7 UNION ALL SELECT 8 UNION ALL SELECT 9 UNION ALL SELECT 10) numbers
        CROSS JOIN (SELECT @prev := NULL, @current := NULL) vars
        WHERE n <= 1 + (LENGTH(TimeCoupon_WLZones) - LENGTH(REPLACE(TimeCoupon_WLZones, ';', '')))
        ORDER BY n
      ) AS subquery
    ) AS sub_count
    WHERE current = prev + 1
  ) = COUNT(*) -- Kontrola, zda jsou všechny čísla spojená za sebou);
```

Obrázek 28 - Dotaz na úplnou korektní formu zápisu kombinací pásem; zdroj: autor

3.3. Analýza dat ve Whitelistu

Problematika analýzy dat představuje klíčový aspekt, který vyžaduje pečlivou pozornost a kontrolu. Validita dat zahrnuje důkladnou analýzu, zda informace obsažené v datech odpovídají stanoveným Smluvním přepravním podmínkám PID a Tarifu PID. Tento proces zahrnuje ověření řady parametrů jako časové platnosti kupónů, správné kategorie a další relevantní informace.

3.3.1. Validace dat platností kupónů

Důležitým atributem u kupónů pro validaci časových platností je hodnota TP (tarif profile). Dle číselníku (Tabulka 6) jsou platnosti měsíční, čtvrtletní, roční, roční kalendářní, bezplatná a nakonec 3 kategorie speciálně pro PVP (Prague visitor pass). Měsíční předplatné jízdné by mělo být platné 30 dní a čtvrtletní 90 dní. Ke kontrole validních kupónů je tedy potřeba vytvořit dotaz, který bude pracovat s rozdílem hodnot WLValidTo a WLValidFrom.

```
SELECT DISTINCT
   (CAST(strftime('%s', WLValidTo) AS INTEGER) -
CAST(strftime('%s', WLValidFrom) AS INTEGER)) / (60 * 60 * 24)
AS DayDifference
FROM
   TimeCoupon_new
WHERE
   TimeCoupon_TP = 20;
```

Obrázek 29 - Dotaz na časové platnosti; zdroj: autor

Daný dotaz (Obrázek 29) vytvoří tabulku, která zobrazuje vyskytující se rozdíly mezi hodnotami WLValidTo a WLValidFrom. Obdobným způsobem se dají validovat i ostatní časové platnosti. Tento konkrétní dotaz je na měsíční předplatné jízdné, tedy hodnota TimeCoupon_TP je 20. Rozdíl by tedy měl být pouze 30 dní. Z výsledků ale vyšly i další hodnoty. Například rozdíl 30 dnů pro měsíční předplatné kupóny byl nalezen u 82 749 záznamů nebo rozdíl 29 dnů byl nalezen u 115 930 záznamů. Tím by se nabízela otázka, zda měsíční jízdné není definováno délkou daného měsíce, ale dle tarifu by mělo jít vždy o 30 dní. Důležité je podotknout, že rozdíl se počítá z dat, a tedy časově se musí přidat dalších 24 hodin, jelikož například denní rozdíl mezi 23. 8. 2022 00:00:00 a 22. 9. 2022 23:59:59 je vyhodnocen jako 30 dní. To znamená, že rozdíl 29 dní je správný. Tím pádem se také ve výsledku dotazu objevuje i rozdíl 0, což znamená platnost od 0:00 do 23:59 téhož dne. Výsledek tohoto dotazu vyšel tak, že každá hodnota od 0 do 30 se v denních rozdílech objevila. Ostatní denní rozdíly už byly jen v menší početné míře. Pro vyhledání konkrétních záznamů s daným časovým rozdílem slouží následující dotaz.

```
SELECT *
FROM TimeCoupon_new
WHERE TimeCoupon_TP = 20 AND (CAST(strftime('%s',
WLValidTo) AS INTEGER) - CAST(strftime('%s', WLValidFrom) AS
INTEGER)) / (60 * 60 * 24) = 30;
```

Obrázek 30 - Vyhledání záznamů časových platností dle hodnoty rozdílu; zdroj: autor

Z dotazu (Obrázek 30) vyšlo, že 30denní platnost pro měsíční předplatné kupóny splňuje 115 930 záznamů z celkových 202 594, které jsou označeny jako měsíční kupóny (TimeCoupon_TP = 20). To je zhruba 57,22 %, což je velice nízká hodnota. Přidáme-li však i záznamy, kde rozdíl tvořil 31 dní, tak se počet záznamů navýší o 82 749. Dohromady to dělá 198 679 záznamů, což tvoří zhruba 98,07 %. To už je lepší výsledek, ale i přesto jde stále o systémovou chybu. Vychází tak, že nějakých 1,93 % záznamů měsíčních předplatných kupónů neodpovídá své časové platnosti.

U čtvrtletních předplatných kupónů obdobná analýza vyhodnotila následující hodnoty. Počet záznamů, kde čtvrtletní kupón (TimeCoupon_TP = 21) měl časový rozdíl 91 nebo 90 dní je dohromady 125 421 z celkových čtvrtletních kupónů 129 710. Jde tedy o 96,69% úspěšnost

korektních kupónů. Pro zajímavost se mezi chybnými hodnotami objevil i kupón, který měl zápornou časovou platnost.

Roční kalendářní kupóny byly všechny v pořádku. Naopak u ročních (365denních) kupónů šlo o úspěšnost 95,91 %. U časových kupónů pro kategorii PVP byly všechny časové rozdíly v pořádku dle tarifu. Poslední z TP kategorií jsou bezplatné časové kupóny. Takové jsou ve whitelistu pouze pro cestující, kteří musejí k tomu dokládat nárok. Jedná se o děti od 6 do 15 let a seniory 65 let a více. Zároveň tyto podmínky platí pouze pro pražská pásma. Ostatní kategorie (ať už v Praze či v regionu) nejsou vázány na předplatné kupóny elektronické, které by se tím nacházely též ve whitelistu. U nynějšího dotazu je tedy potřeba kontrolovat, zda se časová platnost rovná s časovou platností dané slevové kategorie pro daného cestujícího. Stejně tak musí splňovat daná pásma.

```
SELECT COUNT(*)
FROM TimeCoupon_new t, DiscountCard_new d
WHERE TimeCoupon_TP=28
AND (TimeCoupon_WLZones LIKE '%P;0;B%' OR TimeCoupon_WLZones LIKE '%P;0%')
AND (TimeCoupon_CP=2 or TimeCoupon_CP=50 or TimeCoupon_CP=51)
AND d.WLIdentId=t.WLIdentId AND d.DiscountCard_CP=t.TimeCoupon_CP
AND t.WLValidTo<=d.WLValidTo;
```

Obrázek 31 - Validace bezplatných kupónů; zdroj: autor

Výsledek dotazu (Obrázek 31) je takový, že dané podmínky splňuje 62 931 záznamů z celkových 88 383 bezplatných kupónů ve whitelistu. Nicméně po důkladnější analýze databáze bylo zjištěno, že všechny bezplatné kupóny v databázi mají pásmovou platnost pouze v Praze, jedná se pouze o kategorie *Děti 6-15* či *Senior 65+* a mají propojení s identifikátorem, na kterém je nějaká slevová kategorie. Zároveň převážná většina časových platností těchto kupónů je do data časové platnosti slevové kategorie daného cestujícího. Tudíž zbývá nějakých zhruba přes 20 tisíc záznamů, kde je rozpor v customer profile u kupónu a u slevy. Dalším dotazem byly tyto záznamy dohledány.

```
SELECT t.WLIdentId, t.WLValidTo as kupon_do, t.TimeCoupon_CP, t.TimeCoupon_TP,
t.TimeCoupon_WLZones, d.WLValidTo as sleva_do, d.DiscountCard_CP
FROM TimeCoupon_new t, DiscountCard_new d
WHERE TimeCoupon_TP=28
AND (TimeCoupon_WLZones LIKE '%P;0;B%' OR TimeCoupon_WLZones LIKE '%P;0%')
AND (TimeCoupon_CP=2 or TimeCoupon_CP=50 or TimeCoupon_CP=51)
AND d.WLIdentId=t.WLIdentId AND NOT(d.DiscountCard_CP=t.TimeCoupon_CP)
AND t.WLValidTo<=d.WLValidTo
ORDER BY d.DiscountCard_CP;
```

Obrázek 32 - Vyhledání záznamů bezplatných kupónů; zdroj: autor

Z tohoto dotazu (Obrázek 32) bylo nalezeno dalších 25 337 záznamů. Bylo zjištěno, že 25 023 z nich byly záznamy, kde customer profile kupónu byl na hodnotě 51 (*Senior 70+*) a customer profil na hodnotě 50 (*Senior 65+*). Kategorie 51 je ve whitelistu pouze historicky. To tedy znamená, že daní cestující mají aktualizovanou slevovou kategorii, ale jejich předplatný kupón

je stále stejný. Zároveň bylo zjištěno, že všechny hodnoty s kategorií DiscountCard_CP = 51 jsou tytéž a jediné hodnoty, kde se pásma pro Prahu zapisují ve formátu „P;0;B“, i když ve všech ostatních záznamech je už pouze aktualizovaný zápis „P;0“.

Dále byla zjištěna skutečnost, že dalších 279 záznamů jsou již propadlé kupóny v době generování kontrolovaného whitelistu a hodnota slevového customer profile se změnila z dítěte 6-15 na junior 15-18. Tento fakt byl analyzován pomocí dotazu, který pracuje s datem uloženým v hlavičce whitelistu.

```
SELECT COUNT(*)
FROM TimeCoupon_new t, DiscountCard_new d, header h
WHERE TimeCoupon_TP=28
AND (TimeCoupon_WLZones LIKE '%P;0;B%' OR TimeCoupon_WLZones LIKE '%P;0%')
AND TimeCoupon_CP=2
AND d.WLIdentId=t.WLIdentId
AND NOT(d.DiscountCard_CP=t.TimeCoupon_CP)
AND t.WLValidTo<=h.WLScopeTimeTo
AND d.DiscountCard_CP=35;
```

Obrázek 33 - Vyhledání počtu změněných slevových kategorií; zdroj: autor

Dotaz (Obrázek 33) zde vyhledává počet záznamů, které splňují podmínku, že kromě bezplatného kupónu v Praze jde také o cestujícího se slevovou kategorii *Junior 15-18* a jeho kupón pro kategorii *Dítě 6-15* má časovou platnost do data generování daného whitelistu (tj. v nedávné době cestující změnil kategorii z *Dítě 6-15* na *Junior 15-18*). Že daný kupón je stále vidět v daném whitelistu, je ale správně vzhledem k faktu, že se informace ve whitelistu udržují ještě dva týdny.

Celkově tedy dané podmínky bezplatných kupónů v daném whitelistu splňuje (62 931 + 25 023 + 279) 88 233 z celkových 88 383 záznamů bezplatných kupónů. To je úspěšnost 99,83 %. Ze zbylých chybných záznamů je zde v tabulce ukázka.

WLIdentId	TimeCoupon_WLValidTo	TimeCoupon_CP	TimeCoupon_TP	TimeCoupon_WLZones	DiscountCard_WLValidTo	DiscountCard_CP
3199023	2024-09-10 23:59:59	2	28	P;0	2122-09-01 23:59:59	32
1925481	2023-01-13 23:59:59	51	28	P;0;B	2026-05-29 23:59:59	37
2513836	2023-04-06 23:59:59	2	28	P;0	2023-12-31 23:59:59	53
3193398	2023-11-19 23:59:59	2	28	P;0	2023-11-30 23:59:59	53
3193415	2023-11-19 23:59:59	2	28	P;0	2023-11-30 23:59:59	53
1538375	2023-05-28 23:59:59	51	28	P;0;B	2023-07-12 23:59:59	61
2131655	2023-05-22 23:59:59	51	28	P;0;B	2023-07-20 23:59:59	61
3067320	2022-11-30 23:59:59	50	28	P;0	2023-06-13 23:59:59	61

Tabulka 8 - Chybné záznamy bezplatných kupónů; zdroj: autor na základě dat z WL

Chybné údaje v tabulce (Tabulka 8) ukazují určité speciality. Vlastně ani nemusí jít o nevalidní kupóny, ale jen špatně zaznamenané údaje do systému. Například první záznam z tabulky je evidentně *Dítě 6-15* (TimeCoupon_CP = 2), ale jeho slevová kategorie je s hodnotou 32 (*Dítě 6-15 PVP*). Obdobný problém má další záznam, kde dotyčný nemá aktualizovanou slevovou kategorii, ale má zapsanou historickou 37 (*Senior 65-70*).

3.3.2. Validace hodnot CP

V předchozí kapitole bylo již pracováno s hodnotou CP neboli customer profile. Jde o hodnotu, která určuje příslušnost cestujícího k tarifní kategorii. Kategorie jsou dány dle číselníku (Tabulka 6). Dle obyčejného filtrování dat pomocí programu DB browser for SQLite byla zjištěna následující fakta. Hodnoty 6 – *Zvýhodněné* a 30 – *Zvýhodněná ½* vůbec nebyly v datech zastoupeny. Dále je zde několik kategorií, které jsou tarifní, ale nejsou slevové. Tím pádem nemají zastoupení v entitě DiscountCard. Tyhle hodnoty jsou zastoupeny pouze mezi samotnými kupóny. Příkladem je právě základní obyčejná kategorie 1 – *Občanské*. Obdobně je tomu u kategorií 9 – *Zaměstnanec*, 49 – *Rodinný příslušník*, 63 – *Přenosné*, 58 – *Doplatek vlak Praha* a 51 – *Osoba 70+*. Zařazení těchto kategorií mimo slevy dává smysl kromě posledně zmiňované. Kategorie 70+ je zrušená a jak bylo zmíněno v předchozích kapitolách, tak tato kategorie spadá už do aktuální 65+. To znamená, že kupóny jsou zde pod hodnotou 51 – *Osoba 70+*, ale daní cestující mají již aktuální slevovou kategorii 50 – *Osoba 65+*.

Je vhodné také zmínit, že v hodnotách CP jsou také tři kategorie PVP (*Dítě 6-15, Junior 15-26* a *Dospělý 15+*). Tyto kategorie by se neměly objevovat mezi slevami, nicméně pár chybných záznamů bylo nalezeno. Pro každou kategorii minimálně jeden. Konkrétně pro *Junior 15-26 PVP* jich bylo 13, pro *Dítě 6-15 PVP* jich bylo 6 a pro *Dospělý 15+ PVP* byl jeden. Minimálně u posledně zmiňovaného tento záznam už vůbec nedává smysl, aby byl propsaný mezi slevy.

Je tedy potřeba ověřit, zda souhlasí hodnoty CP na kupóny a na slevě pro cestující, kteří mají prokazovat danou slevu. Dále jsou zde kategorie, které by měly mít omezenou časovou platnost. Například platnost slevové hodnoty u kategorie CP pro *Junior 15-18* logicky nemůže být delší než 3 roky. Stejně tak by to mělo platit i u kupónu.

```
SELECT COUNT(*)
FROM DiscountCard_new d, TimeCoupon_new t
WHERE d.WLIdentId=t.WLIdentId and d.DiscountCard_CP=t.TimeCoupon_CP;
```

Obrázek 34 - Počet kupónů s validní slevou - shoda v CP; zdroj: autor

Počet všech kupónů, které mají CP takové, že se k němu zároveň prokazuje sleva, je 361 458. Dále počet těchto kupónů, které mají návaznost přes identifikátor na slevu se shodnou kategorii CP je 356 917. K tomuto zjištění vedl tento kód (Obrázek 34). To znamená, že 98,74 % kupónů, které mají mít návaznost na danou slevu, tuto skutečnost naplňují. Druhou otázkou zde je, zda tyto kupóny mají zároveň časovou platnost do časové platnosti dané slevy. Po obdobném dotazu bylo zjištěno, že záznamů splňující tuto podmínku je 356 641.

Cislo_identifikatoru	Platnost_slevy_do	DiscountCard_CP	TimeCoupon_CP	Platnost_kuponu_do	TimeCoupon_TP	TimeCoupon_WL Zones
2878668	2025-02-22 23:59:59	2	2	2025-02-25 23:59:59	28	P;0
987544	2022-10-31 23:59:59	36	36	2023-04-06 23:59:59	32	P;0
980276	2023-04-07 23:59:59	37	37	2023-04-12 23:59:59	32	P;0
986069	2022-12-10 23:59:59	37	37	2022-12-18 23:59:59	21	P;0
2694117	2023-05-01 23:59:59	35	35	2023-05-10 23:59:59	32	1
3028429	2023-01-08 23:59:59	35	35	2023-02-21 23:59:59	32	1;2
3028429	2023-01-08 23:59:59	35	35	2023-02-21 23:59:59	32	P;0
3306025	2022-11-10 23:59:59	35	35	2022-11-30 23:59:59	21	1;2;3
3306025	2022-11-10 23:59:59	35	35	2022-11-30 23:59:59	21	P;0
1013300	2022-10-31 23:59:59	36	36	2023-09-15 23:59:59	32	P;0
3276761	2022-10-31 23:59:59	36	36	2022-12-26 23:59:59	21	1;2;3;4
3153083	2022-10-26 23:59:59	37	37	2022-10-31 23:59:59	32	P;0
2686365	2023-04-20 23:59:59	40	40	2023-07-08 23:59:59	32	P;0
1763034	2026-10-28 23:59:59	50	50	2027-09-28 23:59:59	28	P;0

Tabulka 9 - Ukázka hodnot kupónů a slev s kolizí časových platností; zdroj: autor na základě dat z WL

Zde (Tabulka 9) jsou vybrané hodnoty z tabulky získané náležitým dotazem, který vyhledal všechny záznamy kupónů s jejich náležitými slevami, které se shodují. Další podmínkou dotazu ale bylo, že platnost kupónu musí být pozdější než platnost náležité slevy k němu. Mezi záznamy jsou vidět například hodnoty kategorií *Student 18-26*, *Junior 15-18* a *Senior 60-65*. Obvykle mají čtvrtletní či roční kupón. Převažují kupóny pro Prahu. Pro zajímavost jsou zde vidět dva záznamy bezplatných kupónů (hodnota TP = 28), jejichž časová platnost by měla být omezena platností příslušné slevy, což neodpovídá.

Z předchozí analýzy vyšlo najevo, že 4 541 záznamů kupónů se slevou (z 361 458 všech takových) nemá ekvivalentní zápis slevy u stejného identifikátoru. Po následných dotazech (Obrázek 35 a Obrázek 36) vyšlo najevo, že z těchto záznamů 3 023 nemělo žádný zápis slevy a 1 518 záznamů mělo zápis se špatnou slevovou kategorií. K tomu navíc u 6 záznamů byl špatný slevový zápis hned dvakrát.

```
SELECT COUNT(*)
FROM TimeCoupon_new t
WHERE t.TimeCoupon_CP IN (2,26,27,35,36,37,40,43,50,53,61) AND
t.WLIdentId NOT IN(
SELECT d.WLIdentId
FROM DiscountCard_new d
);
```

Obrázek 35 - Dotaz na kupóny se slevou bez ekvivalentní slevy v záznamech; zdroj: autor

```
SELECT t.WLIdentId, t.ConId, t.WLValidFrom, t.WLValidTo, t.TimeCoupon_CP,
t.TimeCoupon_TP, t.TimeCoupon_WLZones, d.WLValidFrom, d.WLValidTo,
d.DiscountCard_CP
FROM DiscountCard_new d, TimeCoupon_new t
WHERE d.WLIdentId=t.WLIdentId AND d.DiscountCard_CP<>t.TimeCoupon_CP AND
t.TimeCoupon_CP IN (2,26,27,35,36,37,40,43,50,53,61) AND t.ConId NOT IN(
SELECT t.ConId
FROM DiscountCard_new d, TimeCoupon_new t
WHERE d.WLIdentId=t.WLIdentId AND d.DiscountCard_CP=t.TimeCoupon_CP
```

Obrázek 36 - Dotaz s výpisem kupónů se slevou se záznamem neekvivalentní slevy; zdroj: autor

WLIdentId	ConId	Kupon_od	Kupon_do	TimeCoupon_CP	TimeCoupon_TP	TimeCoupon_WLZones	Sleva_od	Sleva_do	DiscountCard_CP
2277888	6597888	07.11.2019	06.10.2022	2	28	P;0	22.09.2022	31.12.2098	50
2277888	6597888	07.11.2019	06.10.2022	2	28	P;0	07.10.2022	06.10.2025	35
2277892	6597895	07.11.2019	06.11.2025	2	28	P;0	22.09.2022	31.12.2098	50
2277892	6597895	07.11.2019	06.11.2025	2	28	P;0	07.10.2022	06.10.2025	35
1534723	12278731	28.08.2022	27.09.2022	37	20	P;0	31.08.2022	01.01.2199	50
2532228	12279135	05.09.2022	04.10.2022	35	20	1;2;3;4;5	12.10.2022	30.09.2023	36
3176928	12282367	01.09.2022	22.09.2022	35	20	P;0	02.10.2022	02.10.2122	33
3176928	12282367	01.09.2022	22.09.2022	35	20	P;0	02.10.2022	02.10.2122	36
3176928	12282370	01.09.2022	22.09.2022	35	20	1;2;3;4	02.10.2022	02.10.2122	33
3176928	12282370	01.09.2022	22.09.2022	35	20	1;2;3;4	02.10.2022	02.10.2122	36
2767235	12286038	22.08.2022	02.10.2022	2	32	4	03.10.2022	02.10.2025	35
1577513	12413661	02.09.2022	01.10.2022	37	20	P;0	10.09.2022	01.01.2199	50
3154402	12413999	02.09.2022	01.10.2022	35	20	P;0	10.10.2022	31.12.2022	26
3154402	12413999	02.09.2022	01.10.2022	35	20	P;0	10.10.2022	10.10.2030	36
3183954	12414006	03.09.2022	02.10.2022	35	20	1	11.10.2022	31.10.2023	36
3111305	12414593	02.09.2022	09.10.2022	35	21	P;0	10.10.2022	31.12.2022	26
2277892	12715425	05.10.2022	06.10.2022	2	32	0;1	22.09.2022	31.12.2098	50
2277892	12715425	05.10.2022	06.10.2022	2	32	0;1	07.10.2022	06.10.2025	35
2342717	12716988	05.10.2022	05.10.2022	35	21	P;0	06.10.2022	31.10.2023	36

Tabulka 10 - Ukázka výpisu kupónů, ke kterým nesedí sleva; zdroj: autor na základě dat z WL

V tabulce (Tabulka 10) jsou zobrazeny i kupóny, jejichž identifikátor neměl vazbu na ekvivalentní slevu, ale měl vazbu na více než jednu nesouhlasnou slevu. Kupříkladu je vidět kupón vydaný ke slevě *Dítě 6-15*, ale identifikátor k němu měl zapsanou slevu *Junior 15-18* a *Senior 65+*. Kombinace posledních dvou zmiňovaných je úplně nesmyslná. Nebo je zde kupón vydaný pro slevu *Junior 15-18* a na identifikátoru byly nalezeny slevy *Student 18-26* a *Student 18-26 PVP*. Tyto slevy mají platnost až po datu konce platnosti kupónu, nicméně ekvivalentní sleva chyběla, a ještě navíc záznam PVP je chybný. Oba záznamy jsou chybné i svými daty konců platností. Takových chybných záznamů je v tabulce více. Ze záznamů je vidět, že jde o kupóny s různou časovou platností a různou pásmovou platností.

S tímto tématem souvisí další část, kterou jsou nekorektní kombinace slev. Na identifikátoru může být vazba s více jak jednou slevou. Důležité je, zda to nejsou nesmyslné kombinace slev jako v předchozích případech. A pokud jsou slevy korektní, tak zda na sebe správně navazují. Kupříkladu den po konci platnosti slevy *Dítě 6-15* může následovat další záznam slevy kategorie *Junior 15-18*.

```
SELECT WLIdentId,
   MAX(CASE WHEN rn = 1 THEN DiscountCard_CP END) AS DiscountCard_CP_1,
          MAX(CASE WHEN rs = 1 THEN WLValidFrom END) AS WLValidFrom_1,
   MAX(CASE WHEN rp = 1 THEN WLValidTo END) AS WLValidTo_1,
          MAX(CASE WHEN rn = 2 THEN DiscountCard_CP END) AS DiscountCard_CP_2,
   MAX(CASE WHEN rs = 2 THEN WLValidFrom END) AS WLValidFrom_2,
   MAX(CASE WHEN rp = 2 THEN WLValidTo END) AS WLValidTo_2
FROM (
   SELECT
          ConId,
      WLIdentId,
      DiscountCard_CP,
                    WLValidFrom,
                    WLValidTo,
      ROW_NUMBER() OVER(PARTITION BY WLIdentId ORDER BY DiscountCard_CP) AS rn,
                    ROW_NUMBER() OVER(PARTITION BY WLIdentId ORDER BY WLValidFrom) AS rs,
                    ROW_NUMBER() OVER(PARTITION BY WLIdentId ORDER BY WLValidTo) AS rp
   FROM
      DiscountCard_new
) subquery
GROUP BY WLIdentId
HAVING COUNT(ConId)=2;
```

Obrázek 37 - Vypsání kombinací slev na identifikátoru; zdroj: autor

Pomocí dotazu (Obrázek 37) byl vytvořen výpis slev u identifikátorů, kde jsou právě dvě slevy. Obdobně se dají vyhledat i zbylé identifikátory, kde se slevy hromadí. Podrobně je tato problematika řešena v další kapitole ve statistické analýze, kde se objevují počty slev na identifikátorech. Identifikátory s počtem slev tři a více jsou pouze jednotky záznamů. Z vypsaných hodnot se dají lehce vyčíst data o korektních či nekorektních kombinacích slev či o správných či špatných překryvech časových platností daných slev. Kupříkladu se v záznamech potvrdil fakt, že kategorie *Senior 60-65* má platnost o měsíc prodlouženou a vzniká tak měsíční překryv s kategorií *Senior 65+*. Podobně také třeba platí, že kategorie *Odboj/PTP/VTPN* se objevuje jen u *Senior 60-65* a *Senior 65+*. V následující ukázce záznamů (Tabulka 11) je vidět, že se objevují i nesmyslné záznamy, kde například na jednom

identifikátoru je jak *Dítě 6-15* tak *Senior 65+*. Dále je i vidět, že některé slevy na sebe časově správně navazují a některé ne.

WLIdentId	DiscountCard_CP_1	WLValidFrom_1	WLValidTo_1	DiscountCard_CP_2	WLValidFrom_2	WLValidTo_2
981170	37	24.08.2018	25.10.2022	50	25.09.2022	30.06.2029
1158244	2	07.02.2019	13.10.2022	35	14.10.2022	13.10.2025
1460200	35	24.11.2019	23.11.2022	36	24.11.2021	24.11.2022
1013197	2	24.08.2018	29.10.2023	35	26.05.2022	29.10.2024
3165512	2	12.08.2022	09.01.2029	50	31.08.2022	31.12.2098
1494659	35	25.08.2022	06.05.2023	36	26.08.2022	31.10.2023
1558585	35	13.10.2020	12.10.2023	50	11.10.2022	31.12.2098
1386731	37	21.01.2022	06.08.2023	61	03.08.2022	20.01.2027
1434931	50	15.06.2022	14.06.2023	61	01.08.2022	15.11.2149
1956840	36	20.11.2020	23.04.2026	50	01.08.2022	31.12.2098

Tabulka 11 - Ukázka výpisu kombinací slev na identifikátoru; zdroj: autor na základě dat z WL

Poslední analýza pro hodnoty CP se týká časových platností slev, které musí být logicky časově omezené. Jde o kategorie *Dítě 6-15*, kde je hranice tedy maximálně 9 let. Obdobně by se měly zkontrolovat kategorie *Junior 15-18*, *Student 18-26* a *Senior 60-65*. Kde u posledně zmiňované se přidává k pěti letům navíc měsíc. Ostatní slevy jako *Sociálně potřební*, *Hmotná nouze* a *Zvláštní ¼* jsou časově omezeny na rok.

```
SELECT ConId, WLIdentId as Cislo_identifikatoru, WLValidFrom as Platnost_od,
WLValidTo as Platnost_do, DiscountCard_CP
FROM DiscountCard_new
WHERE DiscountCard_CP=2 and (CAST(strftime('%s', WLValidTo) AS INTEGER)
- CAST(strftime('%s', WLValidFrom) AS INTEGER)) / (60 * 60 * 24 * 365) >9;
```

Obrázek 38 - Dotaz na časovou platnost slevy; zdroj: autor

Daný dotaz (Obrázek 38) vypíše hodnoty ID kontraktu (dané slevy), číslo identifikátoru, rozsah platnosti slevy a kategorii dané slevy. Podmínkami zde byly, že daná kategorie je *Dítě 6-15* a časový rozsah platnosti je větší než 9 let. Tím byl nalezen seznam 27 chybných záznamů, které udávají výrazně delší časovou platnost, než jaká by měla být. Následující tabulka ukazuje ukázku z vyhledaných záznamů (Tabulka 12).

ConId	Cislo_identifikatoru	Platnost_od	Platnost_do	DiscountCard_CP
121939215	1939215	2021-12-11 00:00:00	2200-01-01 23:59:59	2
122753432	2753432	2021-12-11 00:00:00	2200-01-01 23:59:59	2
122766109	2766109	2012-10-13 00:00:00	2027-10-12 23:59:59	2
122894824	2894824	2021-12-11 00:00:00	2200-01-01 23:59:59	2
123017004	3017004	2022-04-26 00:00:00	2200-01-01 23:59:59	2
123105931	3105931	2022-06-22 00:00:00	2200-01-01 23:59:59	2
123168986	3168986	2022-08-15 00:00:00	2200-01-01 23:59:59	2
123173314	3173314	2021-12-11 00:00:00	2200-01-01 23:59:59	2
123187876	3187876	2022-08-29 00:00:00	2200-01-01 23:59:59	2
123203838	3203838	2022-09-03 00:00:00	2200-01-01 23:59:59	2

Tabulka 12 - Chybné záznamy slev s delší platností; zdroj: autor na základě dat z WL

Na první pohled je vidět, že jde o nekorektní záznamy, jelikož platnosti jsou ve většině záznamů prodloužené až do roku 2200. Hodně podobná situace nastala i u kategorie *Junior 15-18*, kde podobným dotazem bylo nalezeno 77 záznamů a kde slevy byly opět ve většině případů prodlouženy do roku 2200. U kategorie *Student 18-26* bylo takto chybných záznamů 214. Nakonec stejný problém byl i u chybných záznamů pro kategorii *Senior 60-65*. Zde se navíc i objevovaly chybné hodnoty takové, že ve whitelistu byly hodnoty Platnosti_od nastaveny na roky minulého století. Dá se tedy odhadovat, že počáteční platnost byla asi chybně do systému zapsána jako datum narození daného cestujícího. U kategorie *Hmotná nouze* byl nalezen pouze jeden záznam s platností výrazně delší než jeden rok. U kategorie *Zvláštní ¼* bylo nalezeno 107 záznamů s platností delší než jeden rok a u kategorie *Sociálně potřební* bylo nalezeno 89 záznamů s platností delší než jeden rok.

3.3.3. Validace dat platností identifikátorů

Další kapitolou jsou korektní časové platnosti kupónů v návaznosti na identifikátory. I identifikátory mají své datum expirace. Z toho vyplývá, že kupóny a slevy by neměly mít přesahující časovou platnost, než je datum expirace daného návazného identifikátoru. Pro zajímavost je zprvu dobré zjistit, kolik identifikátorů má datum expirace nižší, než je datum generování daného whitelistu.

```
SELECT i.WLCardExpdate
FROM identifiers_new i, header h
WHERE i.WLCardExpdate<=h.WLScopeTimeTo;
```

Obrázek 39 - Dotaz na expiraci identifikátorů; zdroj: autor

Tímto dotazem (Obrázek 39) bylo zjištěno, že 64 364 identifikátorů z 994 773 celkových má datum expirace nižší, než je datum generování daného whitelistu. Tvoří tak zhruba 6,50 %.

```
SELECT COUNT(*)
FROM TimeCoupon_new t, identifiers_new i
WHERE t.WLIdentId=i.WLMOSIdentId and
t.WLValidTo>i.WLCardExpdate;
```

Obrázek 40 - Dotaz na platnost kupónů v závislosti na identifikátoru; zdroj: autor

Z dotazu (Obrázek 40) vyšlo najevo, že 92 024 časových kupónů z celkových 921 220 má datum konce platnosti pozdější, než je datum expirace vázaného identifikátoru. Tento počet tvoří zhruba 9,99 %. U slev je číslo podstatně větší. Obdobným dotazem vyšel počet 302 875 ze všech celkových záznamů slev 571 713. Jde tedy o 52,98 %, což už je dost markantní výsledek. Vzhledem k hodnotám je dobré poznamenat, že záznamy kupónů a slev, jejichž platnost je pozdější než datum expirace identifikátoru, na který jsou vázány, nejsou chybné. Nicméně z logiky věci je takový stav neideální. Bez identifikátoru jsou dané kupóny a slevy neplatné, a tudíž cestující v průběhu platnosti svého dokladu musí identifikátor měnit či prodloužit jeho platnost. Dobré je dodat, že dnes už cestující mohou během chvíle přehodit svůj kupón pomocí mobilní aplikace Lítačka například z bankovní karty na identifikátor v podobě mobilní aplikace.

3.4. Statistiky dat ve Whitelistu

V této kapitole je popsána analýza dat, která vyhodnocuje statistické informace. Jde o informativní hodnotu dat, což mohou být četnosti výskytů, skladba kupónů či typy nosičů identifikátorů. Jsou zde popsány například počty různých druhů kupónů, slev a podobně. Je zde zaměřeno kupříkladu na data, která pozbývají platnosti v čase generování daného whitelistu. Daná data se nacházejí v databázi správně i po pozbytí platnosti, jelikož by se měla uchovávat zhruba 2 týdny ve whitelistu. V této kapitole nejde primárně o nalezení chyb, ale i při takové analýze bylo nalezeno mnoho chybných dat. Dále jde v této kapitole také o popsání vazeb 1:N mezi entitami databáze. Předpokládá se, že cestující neboli klient, který je popsán v databázi svými osobními daty, může na tato data navazovat více jak jedním identifikátorem. Tento vztah platí i pro vazby mezi identifikátorem a kupóny, respektive slevami. Důležité je zmínit, že všechny údaje popsané v této kapitole se vztahují k datu daného whitelistu, což je 13. 10. 2022.

3.4.1. Informační analýza klientů

V daném analyzovaném whitelistu se nachází 254 870 záznamů osobních dat. Konkrétní entita Clients má atributy WLMOSPssngrAcct, který udává jedinečné číslo účtu klienta, a PersonalData, kde jsou konkrétní údaje osobních dat, které jsou zašifrované. Druhý zmiňovaný atribut byl nahrazen hodnotou délky zašifrovaných dat pro zachování alespoň nějaké informativní hodnoty atributu. Jde tedy o atribut PersonalDataLength.

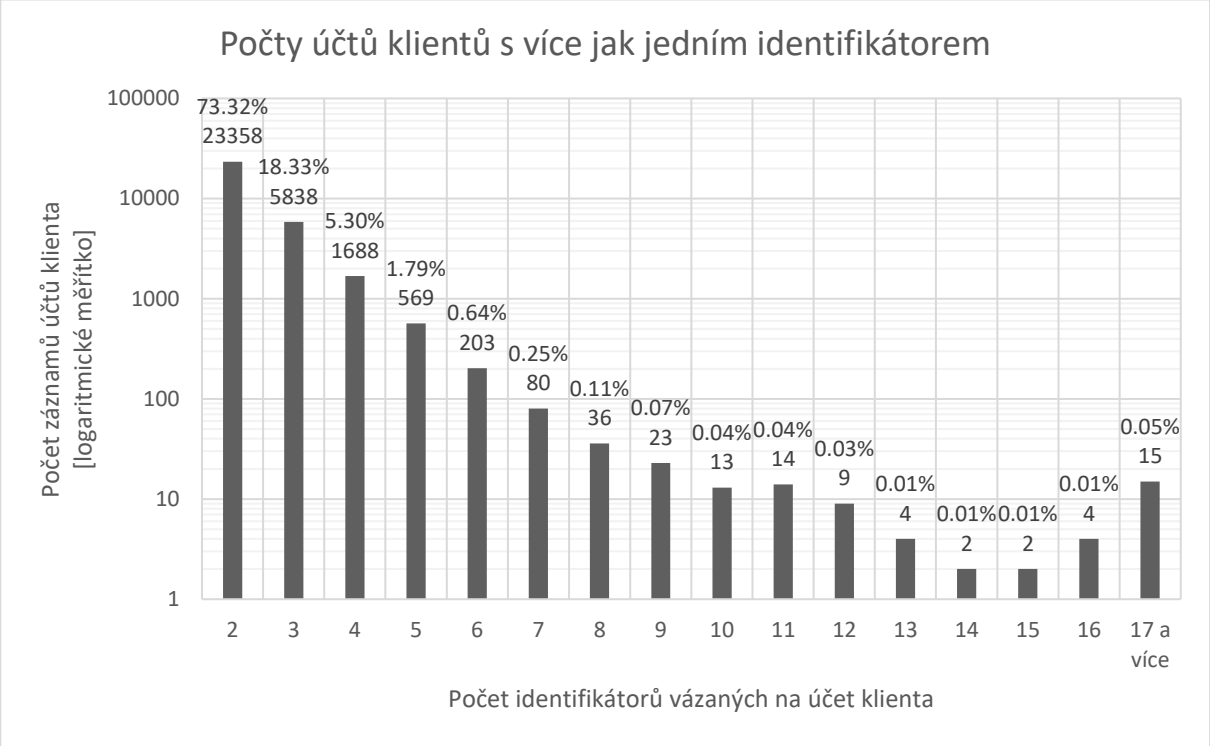
V této části je analýza zaměřena na záznamy účtu klientů, které mají návaznost na více jak jeden identifikátor. Obdobně je pak analýza zaměřena na účty s návazností na více jak jeden

kupón či s návazností na více jak jednu slevu. Pomocí jednoduchého dotazu (Obrázek 41) bylo nalezeno 31 858 záznamů účtů klientů s více jak jedním identifikátorem z celkových 254 870. To tvoří zhruba 12,50 % všech účtů klientů v daném whitelistu.

```
SELECT WLMOSPssngrAcct,
COUNT(WLMOSPssngrAcct) AS PocetOpakovani
FROM identifiers_new
GROUP BY WLMOSPssngrAcct
HAVING COUNT(WLMOSPssngrAcct) > 1;
```

Obrázek 41 - Zjištění počtu opakování zákaznických účtů u identifikátorů; zdroj: autor

Byly nalezeny jednotky záznamů, kdy k účtu klienta bylo vázáno klidně i 12 a více identifikátorů. Nalezené maximum však tvořila hodnota 86 identifikátorů k jednomu účtu. Podrobnou statistiku zobrazuje následující graf.



Obrázek 42 - Graf klienti s více identifikátory; zdroj: autor na základě dat z WL

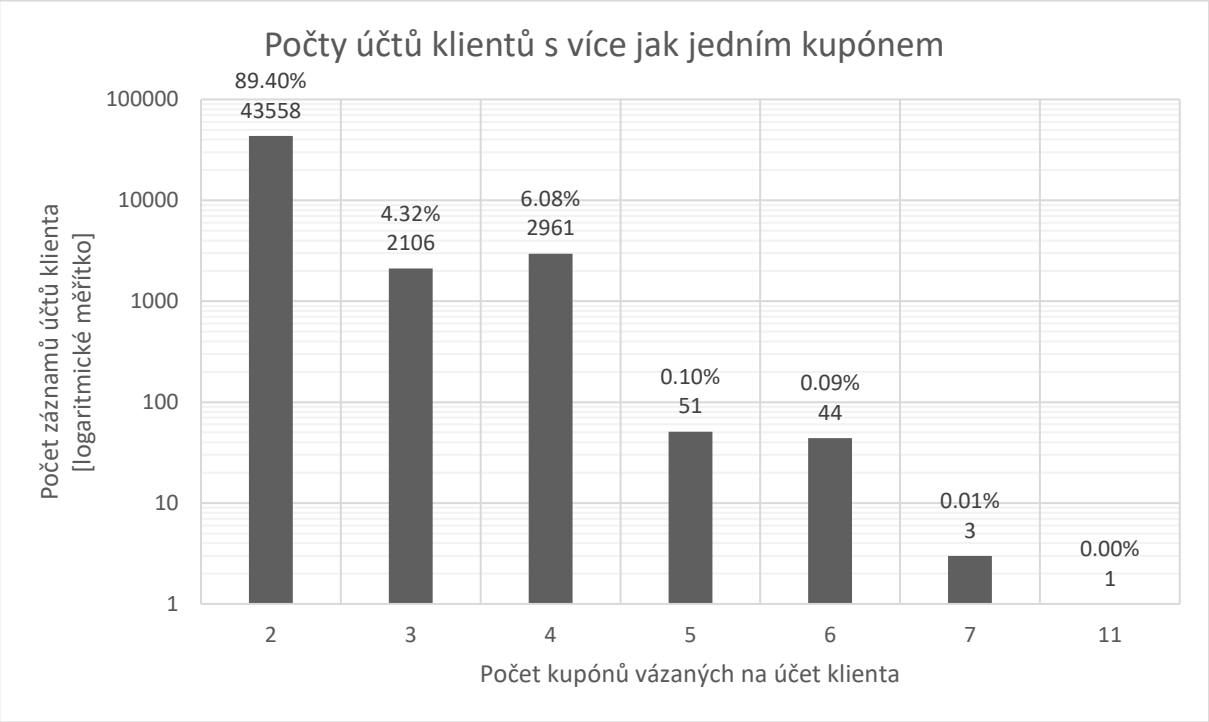
Pro přehlednější zobrazení je v grafu (Obrázek 42) použito logaritmické měřítko o základu 10. V grafu je vidět, že velká většina účtů s více identifikátory má návaznost pouze na dva. Jde konkrétně o 73,32 % ze všech záznamů účtů s více jak jedním identifikátorem a jde tak o 23 358 záznamů. Účtů se třemi identifikátory je pak 5 838, což tvoří 18,33 %. Následně už jsou hodnoty rapidně skokové. Účtů s 12 identifikátory a více jsou už pouze jednotky záznamů.

Obdobným způsobem lze vyhodnotit klientské účty s jejich počty kupónů a slev. Pomocí obměněného dotazu (Obrázek 43) bylo nalezeno 48 724 záznamů účtů, které obsahují více jak jeden kupón. Z celkového výčtu všech klientů tedy tvoří 19,12 %.

```
SELECT WLMOSPssngrAcct,
COUNT(WLMOSPssngrAcct) AS PocetOpakovani
FROM TimeCoupon_new t, identifiers_new i
WHERE t.WLIdentId=i.WLMOSIdentId
GROUP BY WLMOSPssngrAcct
HAVING COUNT(WLMOSPssngrAcct) > 1;
```

Obrázek 43 - Zjištění počtu opakování zákaznických účtů u kupónů; zdroj: autor

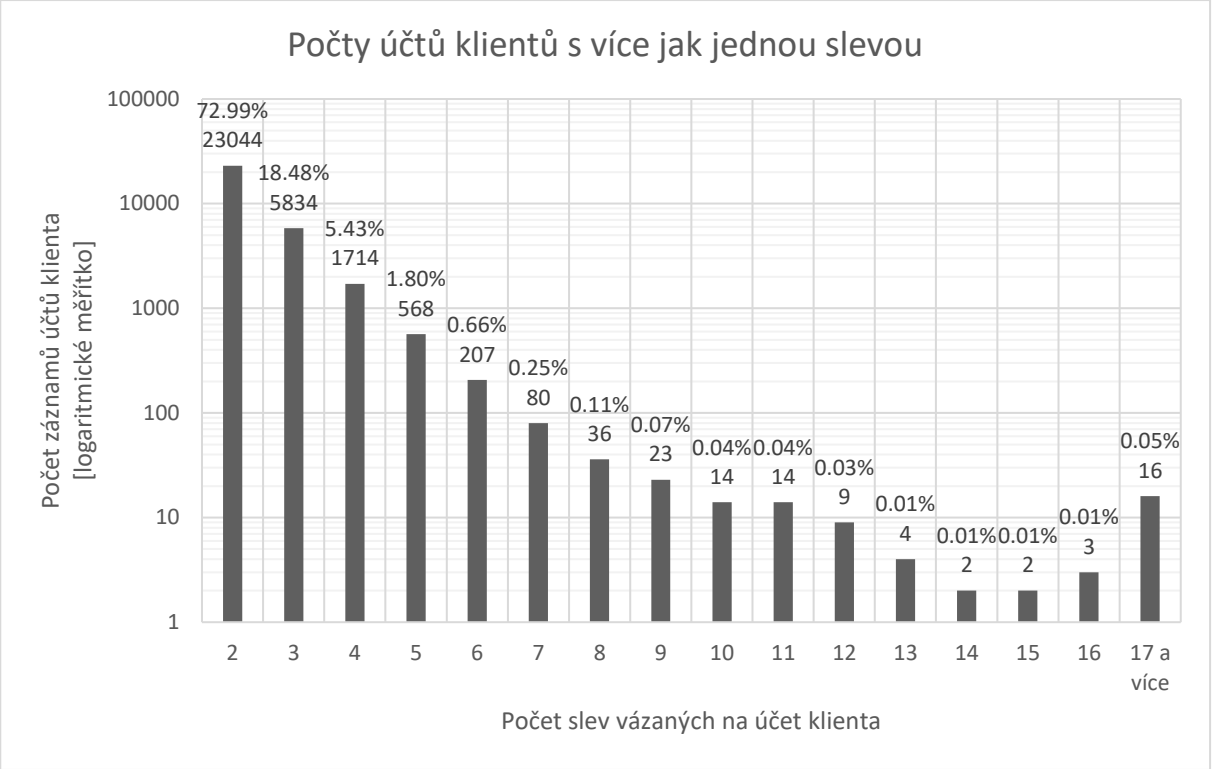
Bylo zjištěno, že opět převážná většina účtů s více kupóny má právě dva. Dále se počty vyskytují až do hodnoty sedm, kdy se řády změní v jednotky. Konečným extrémem je počet 11 kupónů vázaných k jednomu klientovi. Podrobnosti zobrazuje následující graf.



Obrázek 44 - Graf klienti s více kupóny; zdroj: autor na základě dat z WL

Stejným způsobem byly vyhodnoceny i slevy k účtům klientů. Bylo nalezeno 31 570 záznamů účtů, které obsahují více jak jednu slevu. Z celkového výčtu všech klientů tedy tvoří 12,39 %. Hodnoty se až nápadně podobají hodnotám počtů účtů klientů s více jak jedním identifikátorem. Samozřejmě převažují účty, kde jsou vedeny přesně dvě slevy. A obdobně se stoupajícím počtem slev klesá počet účtů. Zde už data naznačují, že se musí jednat i o chybná

data. Při výpisu některých z účtů bylo zjištěno, že na daný klientský účet byly kupříkladu vedeny nekombinovatelné slevy jako *Dítě 6-15* a *Senior 65+*.



Obrázek 45 - Graf klienti s více kupóny; zdroj: autor na základě dat z WL

Opět jsou v grafu (Obrázek 45) zobrazeny počty, včetně procentuálního zastoupení. Zajímavou chybou je zde naměřené maximum, kdy k jednomu účtu je vedeno 86 slev. Jde o záznamy související s maximem počtu identifikátorů na jednom účtu, kterých bylo také 86. Každý slevový záznam je veden na jiný identifikátor, ale zároveň na stejný klientský účet. Pomocí následujícího dotazu byl vyhledán výpis k danému účtu.

```
SELECT i.WLMOSPssngrAcct, i.WLMOSIdentId,
i.WLCardExpdate, i.WLCardType, c.ConId,
c.DiscountCard_CP, c.WLValidFrom, c.WLValidTo
FROM identifiers_new i, contracts_new c
where i.WLMOSIdentId=c.WLIdentId and
i.WLMOSPssngrAcct=4491940;
```

Obrázek 46 - Dotaz na výpis slev k danému klientskému účtu; zdroj: autor

Pomocí tohoto dotazu (Obrázek 46) byl vygenerován výpis všech 86 záznamů. Ukázka části z nich je v následující tabulce.

WLMOSPssngr Acct	WLMOSIden tId	WLCardExpd ate	WLCard Type	ConId	DiscountC ard_CP	WLValidFro m	WLValidTo
4491940	2931700	17.01.2032	3	1362931700	36	17.10.2021	08.09.2026
4491940	2932920	18.01.2032	3	1362932920	36	17.10.2021	08.09.2026
4491940	2962064	18.02.2032	3	1362962064	36	17.10.2021	08.09.2026
4491940	3044955	02.05.2032	3	1363044955	36	17.10.2021	08.09.2026
4491940	3053829	10.05.2032	3	1363053829	36	17.10.2021	08.09.2026
4491940	3069329	24.05.2032	3	1363069329	36	17.10.2021	08.09.2026
4491940	3097568	14.06.2032	3	1363097568	36	17.10.2021	08.09.2026
4491940	3099815	16.06.2032	3	1363099815	36	17.10.2021	08.09.2026
4491940	2928060	13.01.2032	3	1362928060	36	17.10.2021	08.09.2026
4491940	2900490	18.12.2031	3	1362900490	36	17.10.2021	08.09.2026

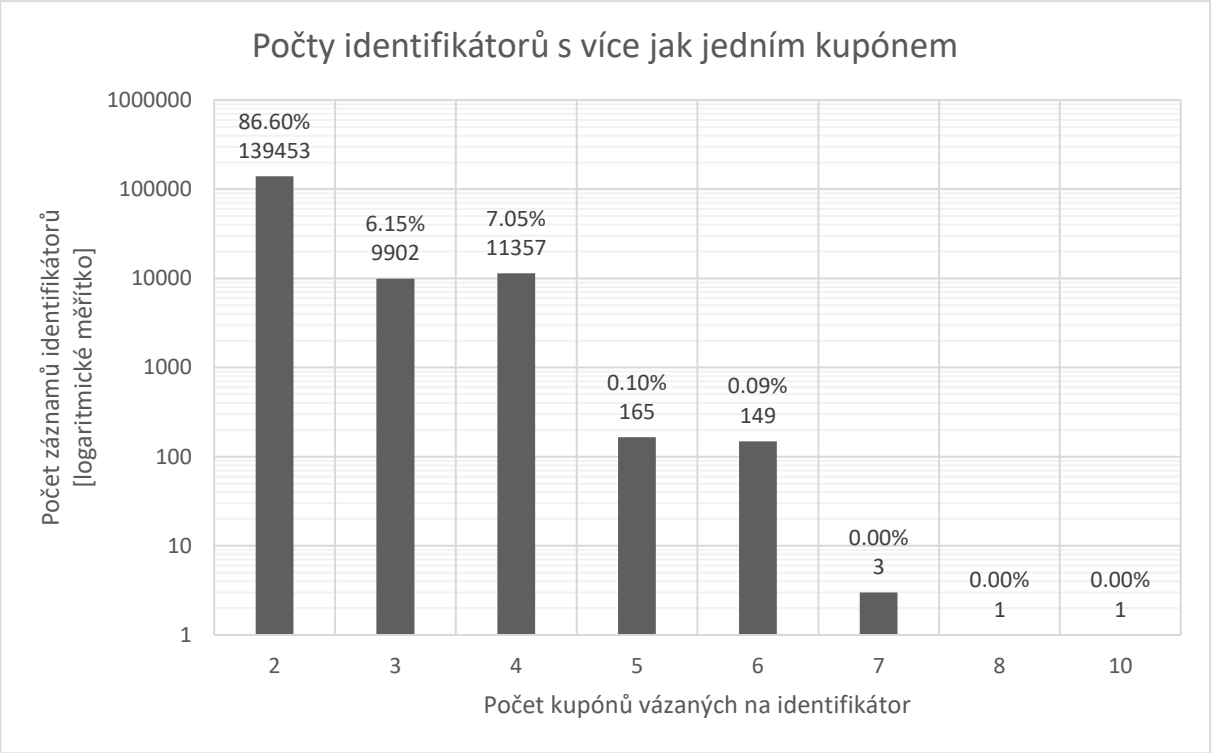
Tabulka 13 - Ukázka výpisu účtu s 86 záznamy slev; zdroj: autor na základě dat z WL

Ze záznamů (Tabulka 13) lze vidět, že všechny slevy jsou stejného typu (DiscountCardCP = 36) *Student 15-18*. WLMOSPssngrAcct je vždy dané číslo klientského účtu. WLMOSIdentId je pokaždé jiný, což znamená, že každý slevový záznam je veden na jiný identifikátor. Zároveň platí, že každý identifikátor má jiný datum expirace, ale ve všech případech jde o nosič typu mobilní telefon (WLCardType = 3). Posledním údajem v tabulce je, že všechny slevové záznamy mají stejnou časovou platnost. Ze všech těchto faktů lze odhadovat, že dané záznamy jsou vedeny chybně, jelikož není moc pravděpodobné, aby si cestující v krátkém časem měnil mobilní telefon, respektive nosič své slevy 86krát.

Podobným způsobem by šly analyzovat všechny účty, které mají vedeny na své číslo nezvyklé množství identifikátorů, kupónů či slev.

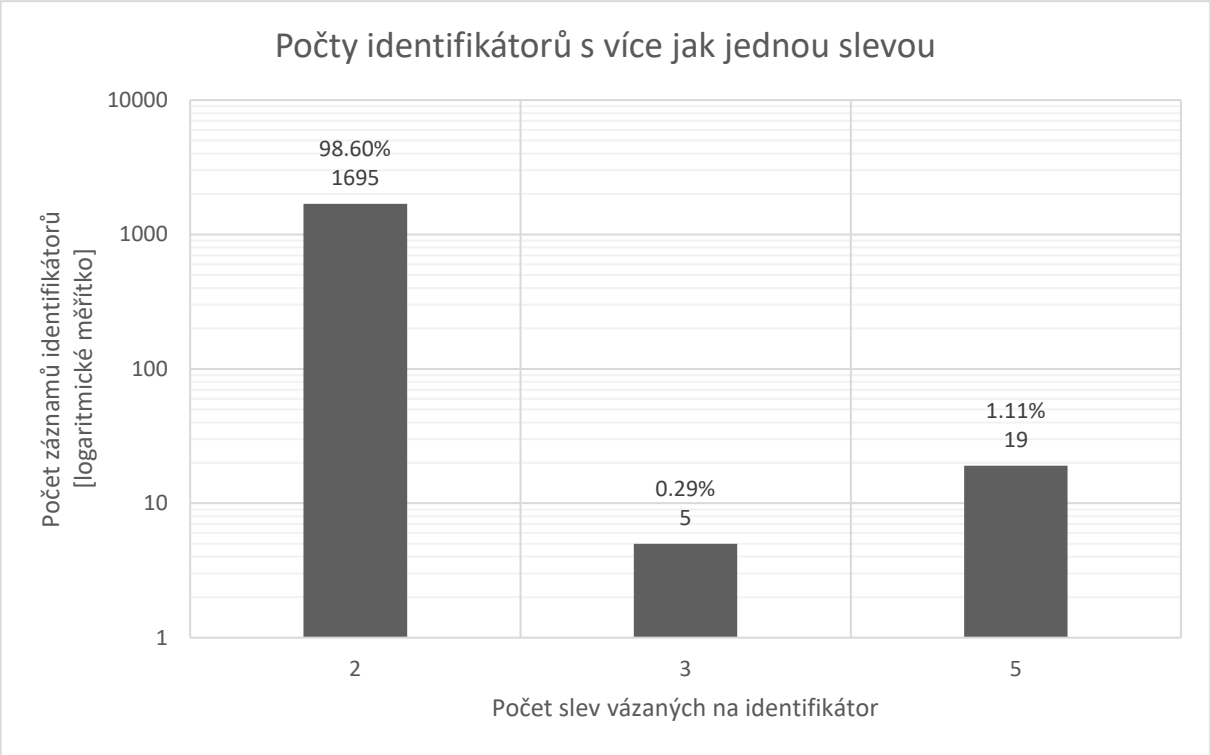
3.4.1. Informační analýza identifikátorů

Informační analýza u identifikátorů je výrazně bohatší. Je zde mnohem více popisujících informací. Už jen z definované struktury dané entity je vidět, že zajímavých atributů má mnohem více. Stejně jako v předchozí kapitole se zde dá zaměřit na počty kupónů a slev vázaných na identifikátory. Další zkoumané hodnoty mohou být informace o datech expirací kupónů a obdobně pak zkoumání této hodnoty vůči datu vygenerování daného whitelistu. Dále je zkoumáno, kolik identifikátorů má návaznost na osobní data klientů. Nakonec nejzajímavějším údajem jsou typy nosičů, které se vyskytují ve whitelistu. V tomto whitelistu se nachází celkem 994 773 záznamů identifikátorů. Data jsou ke dni 13. 10. 2022.



Obrázek 47 - Graf identifikátory s více kupóny; zdroj: autor na základě dat z WL

Na grafu (Obrázek 47) jsou vidět počty identifikátorů, které mají vazbu s více jak jedním kupónem. Celkem jich je 161 031 záznamů z celkových všech identifikátorů 994 773, což tvoří zhruba 16,19 %. Zároveň všech identifikátorů, které mají nějakou vazbu na alespoň jeden kupón, je 726 452, což je 73,03 % ze všech. Analogicky je tedy počet identifikátorů s více kupóny ze všech, které mají vazbu alespoň na jeden kupón, roven 22,17 %. Z nich jasně dominuje počet identifikátoru s vazbou na dva kupóny. Následují počty identifikátorů s vazbami se třemi a čtyřmi kupóny o hodnotách kolem deseti tisíc. Dále pak v řádu desítek hodnot se nacházejí počty identifikátorů s vazbou na pět a šest kuponů. Nakonec jde pouze už o jednotky případů, kdy identifikátory mají vazbu se sedmi, osmi a deseti kupóny.



Obrázek 48 - Graf identifikátory s více slevami; zdroj: autor na základě dat z WL

Posledním grafem s touto tématikou je graf (Obrázek 48) zobrazující počty identifikátorů, které mají vazbu s více jak jednou slevou. Celkem jich je 1 719 záznamů, což tvoří zhruba 0,17 % z celkového počtu všech identifikátorů 994 773. Zároveň všech identifikátorů, které mají nějakou vazbu na alespoň jednu slevu, je 569 932, což je 57,29 % ze všech. Analogicky je počet identifikátorů s více slevami roven 0,30 % ze všech identifikátorů, které mají vazbu alespoň s jednou slevou. Skoro veškeré záznamy z této množiny mají vazbu se dvěma slevami. Pouze pět identifikátorů má vazbu s třemi slevami a 19 identifikátorů s pěti slevami.

WLMOSIdentId	ConId	DiscountCard_CP	WLValidFrom	WLValidTo
3173314	123173314	2	2021-12-11 00:00:00	2200-01-01 23:59:59
3173314	1353173314	35	2021-12-11 00:00:00	2200-01-01 23:59:59
3173314	1363173314	36	2021-12-11 00:00:00	2200-01-01 23:59:59
3173314	1503173314	50	2021-12-11 00:00:00	2200-01-01 23:59:59
3173314	1373173314	37	2021-12-16 00:00:00	2200-01-01 23:59:59

Tabulka 14 - Výpis z dotazu na konkrétní identifikátor, který má vazbu na pět slev; zdroj: autor na základě dat z WL

Z tabulky (Tabulka 14) jsou vidět konkrétní záznamy slev s vazbou na tentýž identifikátor. Je zde vidět, že všechny slevy mají stejnou časovou platnost, která je zároveň nesmyslná, jelikož k pozbytí platnosti těchto slev dochází až v roce 2200. Zároveň je zde vidět, že k jednomu identifikátoru je vazba k nekombinovatelným slevám. Jsou tu slevy *Dítě 6-15*, *Junior 15-18*, *Student 18-26*, *Senior 60-65* a *Senior 65+*. To, že daný identifikátor má vazby s nekombinovatelnými slevami, nemusí být špatně. Identifikátor bez vazby na osobní data

může být i nepersonalizovaný, například pro přenosné časové kupóny. Tento konkrétní identifikátor však vazbu s osobními daty tvoří.

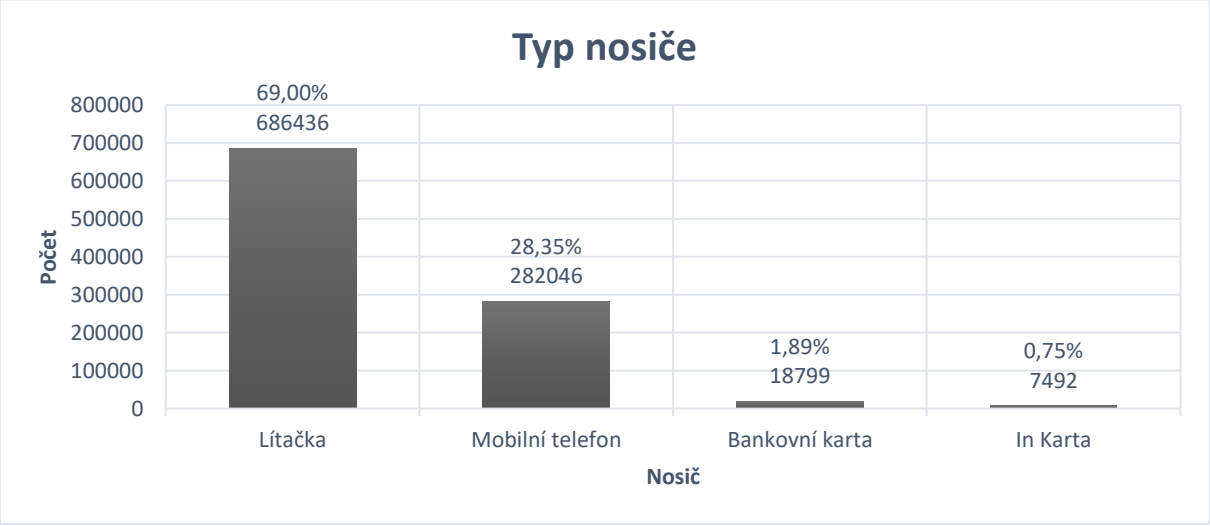
Je dobré představit, jaké druhy a typy nosičů se ve whitelistu vyskytují mezi identifikátory. Nejdříve je tu dle Smluvních přepravních podmínek PID řazení identifikátorů dle způsobu zpracování osobních dat. [12]

a) **Typ I – Osobní identifikátor s evidencí**
   Jde o identifikátor, který má v systému vazbu na osobní data. Tudíž například tento identifikátor nelze využít pro přenosné časové jízdné. Tento identifikátor je povinný, pokud cestující využívá nějakou slevovou kategorii.
b) **Typ II – Osobní identifikátor bez evidence**
   Tento typ identifikátoru se využívá bez poskytnutí osobních údajů do systému. Nelze ho využít pro spojení se slevovou kategorií. Lze ho použít pouze pro občanské jízdné.
c) **Anonymní nepersonalizovaný identifikátor**
   Tento identifikátor se využívá pro spojení s dlouhodobým přenosným jízdným. Neposkytují se žádné osobní údaje do systému a nelze ho využít pro slevové kategorie ani pro občanské jízdné.
d) **Semianonymní nepersonalizovaný identifikátor**
   Pro tento typ identifikátoru se poskytují například firemní údaje. Lze ho využít pouze pro spojení s dlouhodobým přenosným jízdným. Nelze tedy využít pro nepřenosné občanské jízdné či slevové kategorie.

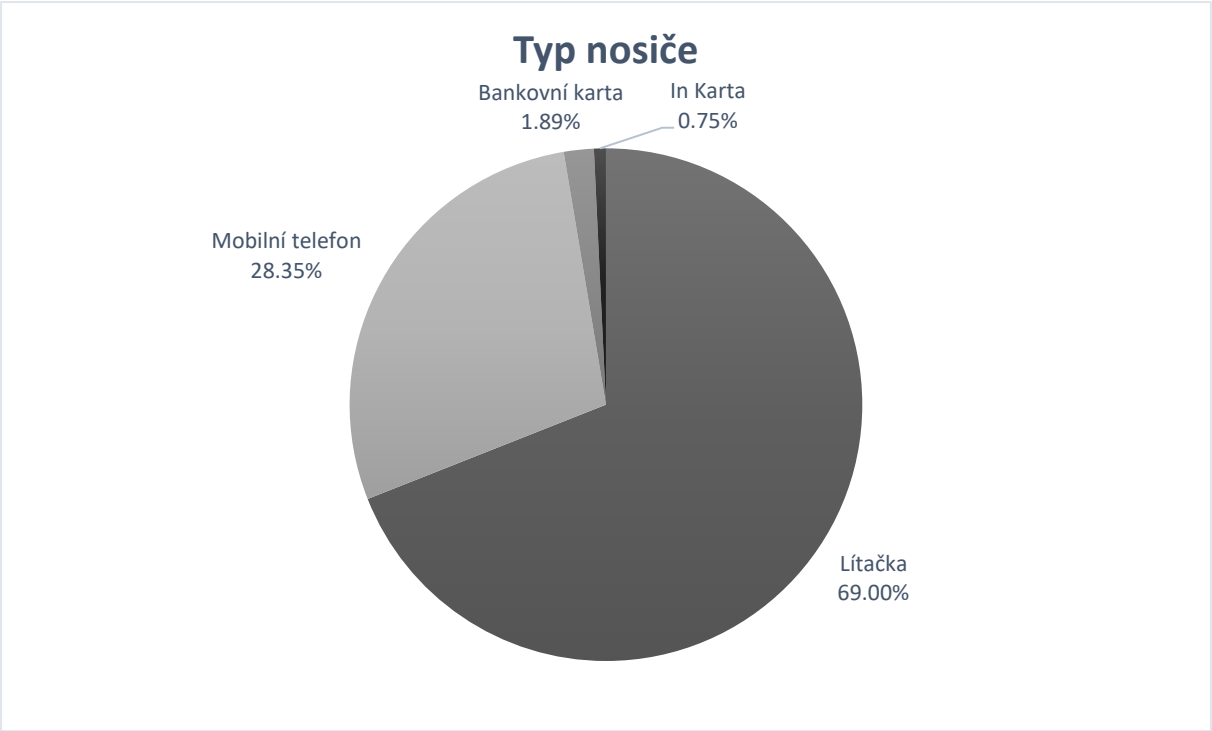
Další rozdělení je dle typu nosiče. Dané nosiče mohou využívat různé typy identifikátorů dle předchozího řazení. Dle SPP jsou definovány následovný čtyři typy nosičů. [12]

I. **Čipová karta Lítačka** – Může být typ **a), b), c)** i **d)**
II. **In Karta Českých drah** – Jde o typ **a)**
III. **Bankovní platební karta** – Může být typ **a), c)** a **d)**
IV. **Aplikace PID Lítačka v mobilním komunikačním zařízení** – Jde o typ **a)**

Ve zpracovávaném whitelistu je rozložení typů nosičů následující.



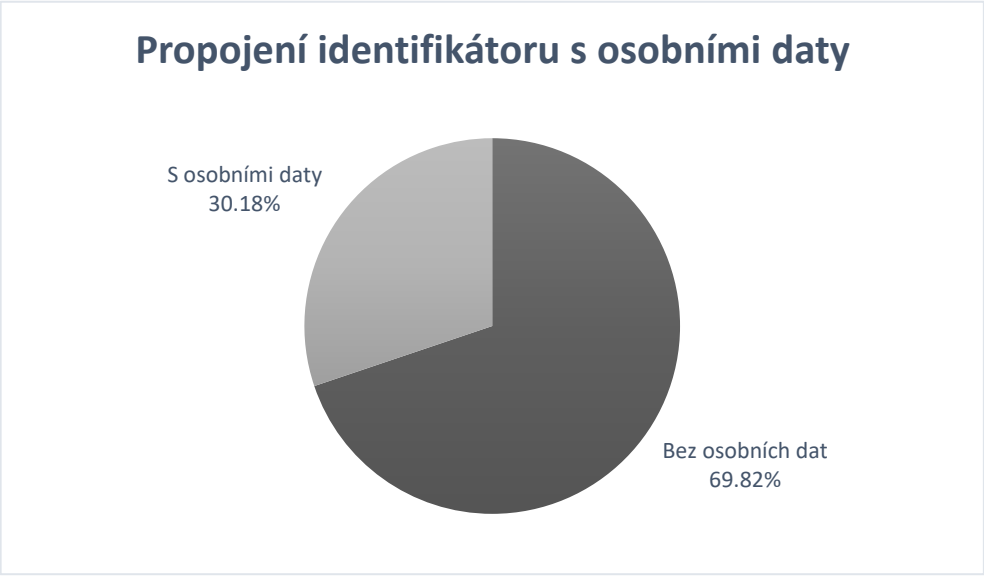
Obrázek 49 - Graf rozložení typů nosičů 1; zdroj: autor na základě dat z WL



Obrázek 50 - Graf rozložení typů nosičů 2; zdroj: autor na základě dat z WL

Z grafů (Obrázek 49 a Obrázek 50) plyne, že nejvíce je daném whitelistu zastoupena Lítačka mezi identifikátory. Z toho vyplývá, že v říjnu 2022 v elektronických kupónech cestující nejvíce využívali jako nosič kupónu Lítačku (více jak ze dvou třetin). Necelá třetina využívala jako nosič aplikaci Lítačka na mobilním telefonu. Jen hrstka cestujících využívala Bankovní kartu či In Kartu.

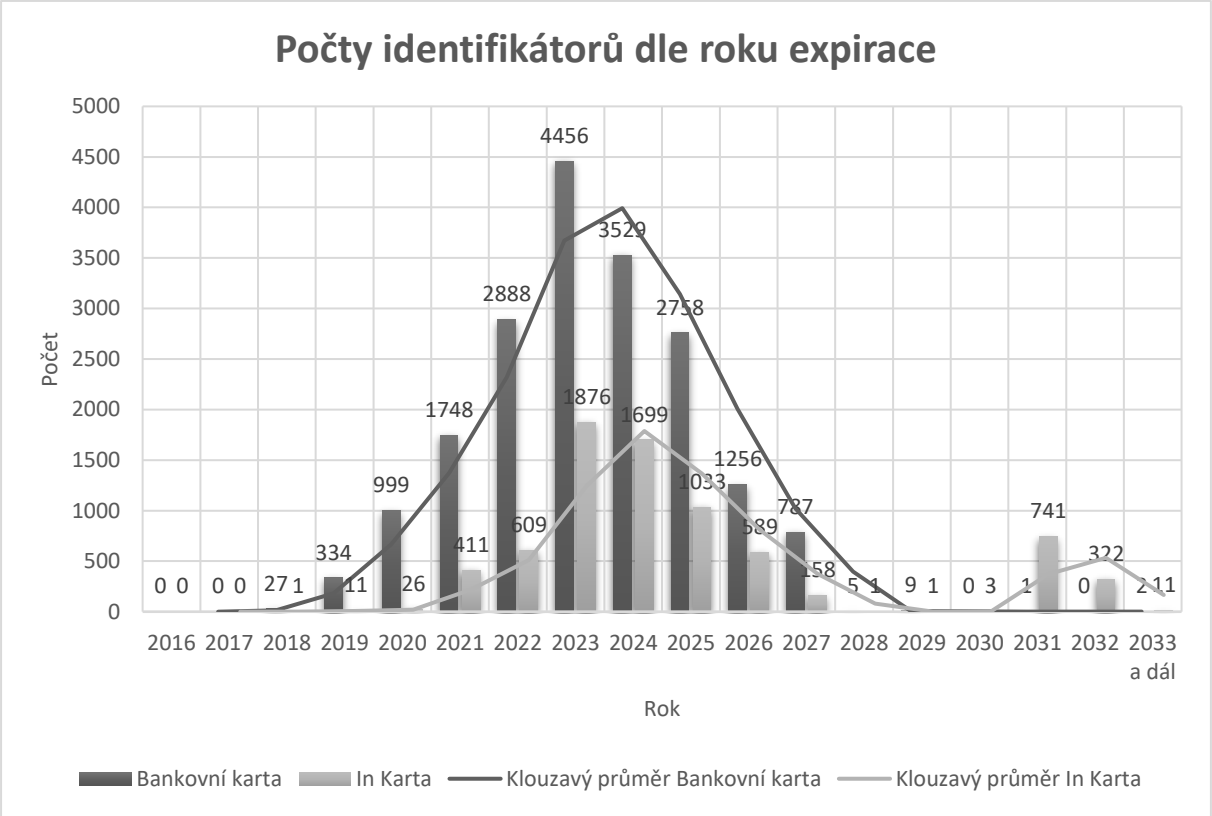
Ohledně vazby na osobní data je stav následující.



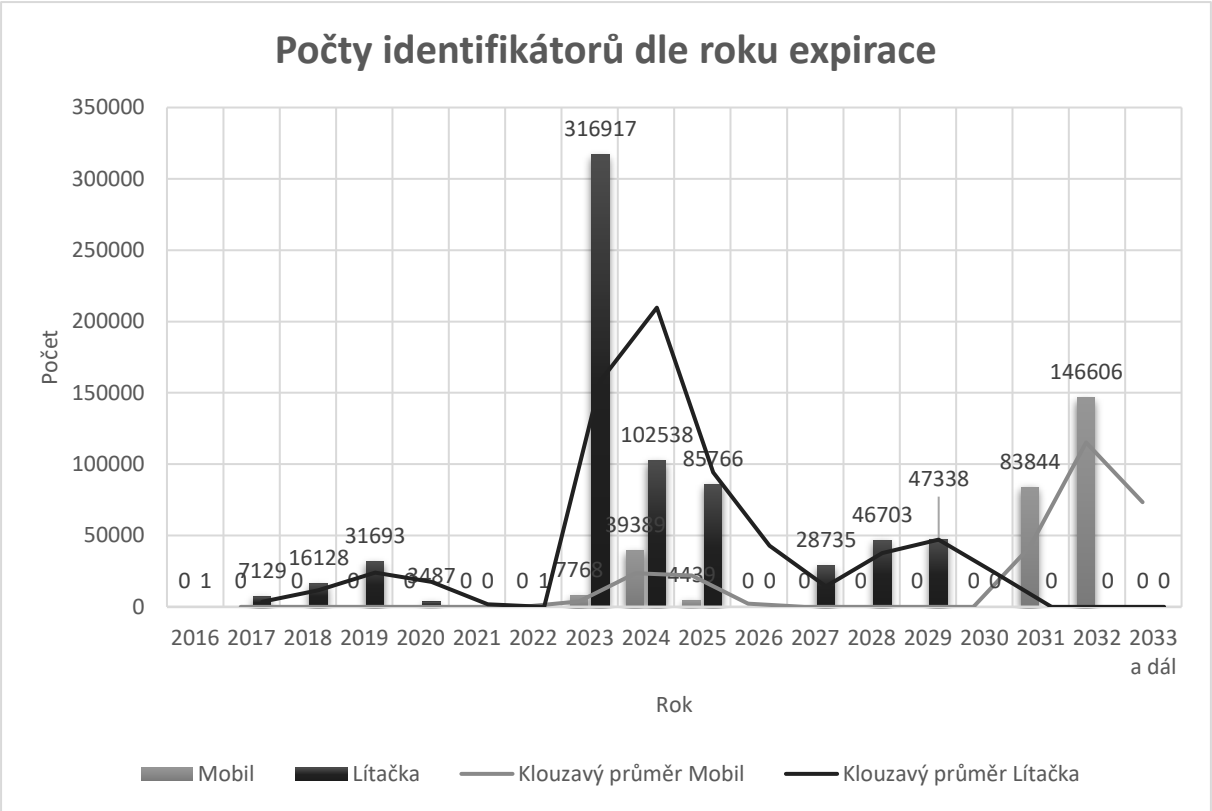
Obrázek 51 - Graf rozložení identifikátoru dle vazby na osobní data; zdroj: autor na základě dat z WL

Z grafu (Obrázek 51) vyplývá, že pouze necelá třetina identifikátoru má vazbu na osobní data. V daném whitelistu záznamy identifikátory s vazbou na osobní data mají pouze nosiče typu mobilní telefon (mobilní aplikace Lítačka) a bankovní karta. Zároveň existují malé desítky záznamů bankovních karet a mobilních telefonů bez vazby na osobní data. U bankovních karet to nemusí být špatně, ale mobilní aplikace by měla být vždy typ I, tedy mít vždy vazbu na osobní data. Stejná chyba je i u záznamů nosičů In karta, kdy by měly záznamy být opět vždy s vazbou na osobní data. Zvláštní je i skutečnost, že žádný záznam identifikátoru karty Lítačky nemá vazbu na osobní data.

Poslední zkoumanou hodnotou u identifikátorů jsou data expirací. Platnosti nosičů jsou následovné. Karta Lítačka má omezení platnosti natištěné na kartě, ale platnost karty se opakovaně prodlužovala. Mobilní aplikace má platnost 10 let dopředu. Bankovní karta platí po dobu platnosti karty. To může být v horizontu 3 až 5 let dle konkrétní banky. In Karta platí podle data platnosti karty, kdy České dráhy vydávají karty s platností na 10 let.

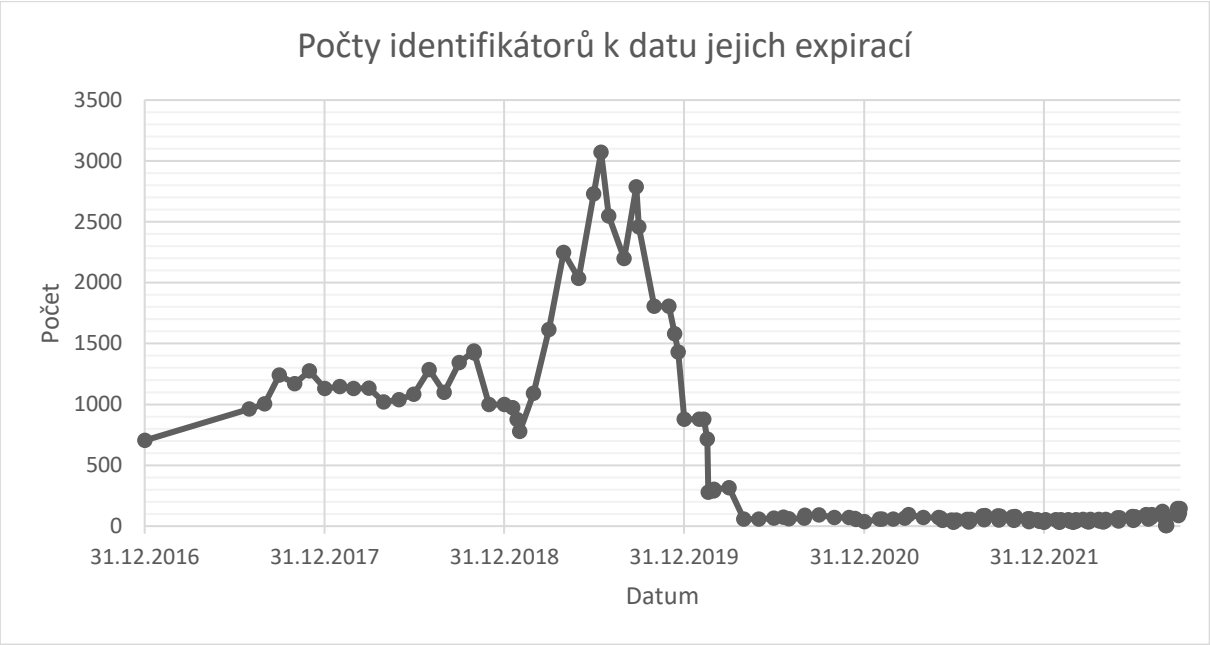


Obrázek 52 - Graf vybraných identifikátorů dle data expirace 1; zdroj: autor na základě dat z WL



Obrázek 53 - Graf vybraných identifikátorů dle data expirace 2; zdroj: autor na základě dat z WL

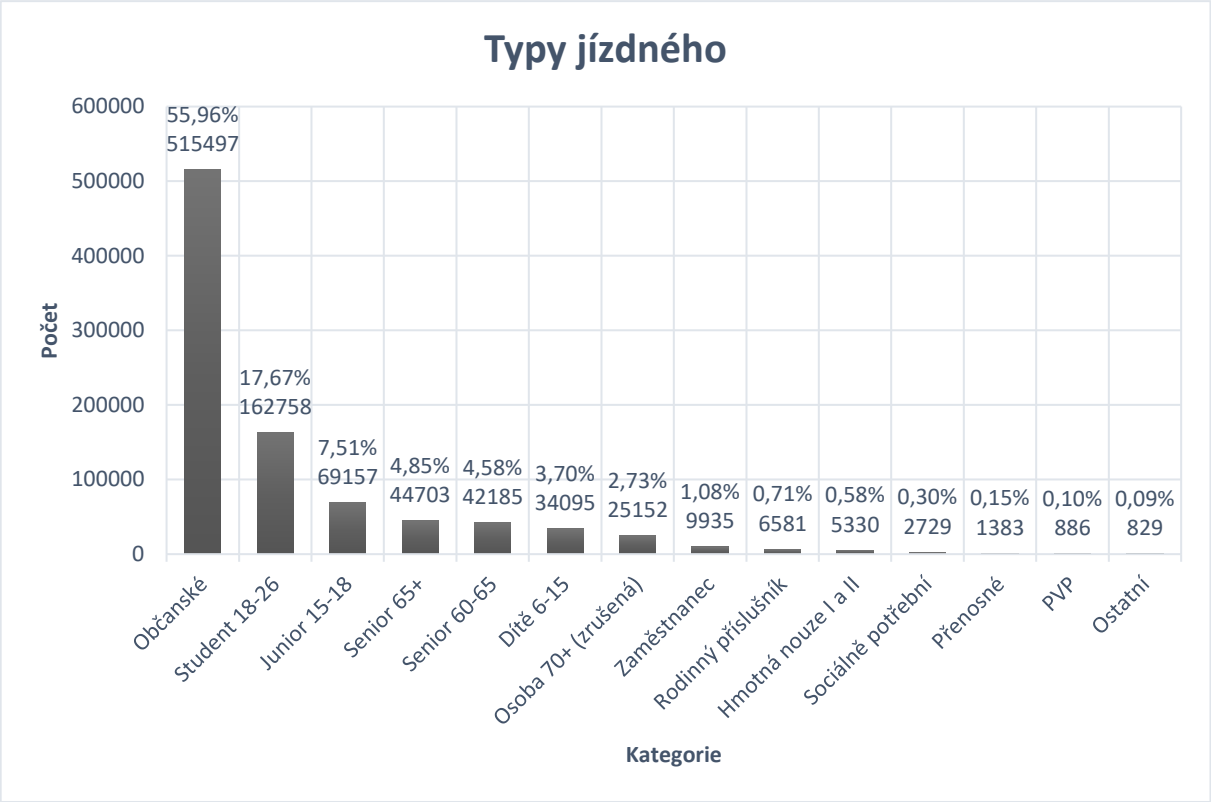
Z grafů (Obrázek 52 a Obrázek 53) je vidět nepoměr v datech expirace mezi nosiči typu mobilní aplikace a Lítačka a mezi nosiči In karta a bankovní karta. Početně řádově menší hodnoty In karet a bankovních karet ve whitelistu odpovídají normálnímu statistickému rozdělení. Je zde vidět vrchol v datu expirací kolem roků 2023 a 2024. Jinak je tomu u druhých dvou typů identifikátorů. U záznamů karet Lítačka dominuje počet identifikátorů s expirací v roce 2023. Jde o většinu záznamů karet typu Lítačka. U mobilní aplikace je vidět, že vrchol počtu záznamů je s datem v roce 2032. Což odpovídá deseti letům od data vygenerování daného whitelistu. Zvláštností je, že nemalé množství záznamů identifikátorů má v datu generování daného whitelistu již propadlou platnost. Následující graf zobrazuje jejich průměrný vývoj, jak expirují v průběhu času (Obrázek 54).



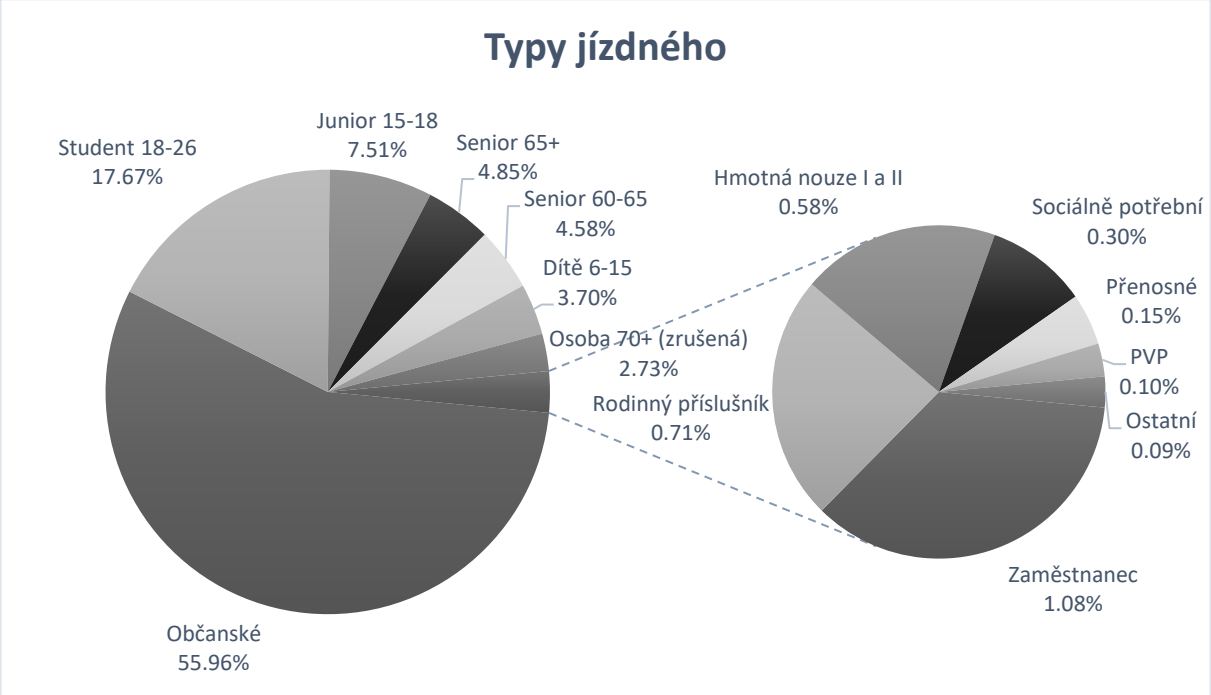
Obrázek 54 - Graf vývoje průměrných počtů expirací propadlých identifikátorů (k datu vydání whitelistu) v čase; zdroj: autor na základě dat z WL

3.4.2. Informační analýza kupónů

Analýza informací získaných z dat záznamů kupónů v databázi daného whitelistu je z hlediska odbavovacích systému nejzajímavější. Je zde mnoho popisných atributů. Kupóny se dají rozlišovat dle typu jízdného, což jsou kupříkladu kategorie jako student, dítě, senior či prosté občanské. Dále se dá sledovat časová platnost kupónů, kde jde o tarifní kategorie typu měsíční, čtvrtletní či roční jízdné. Další sledovanou hodnotou je územní platnost kupónů, kde se dá lehce rozlišit, zda jde o kupón regionální či pražský.

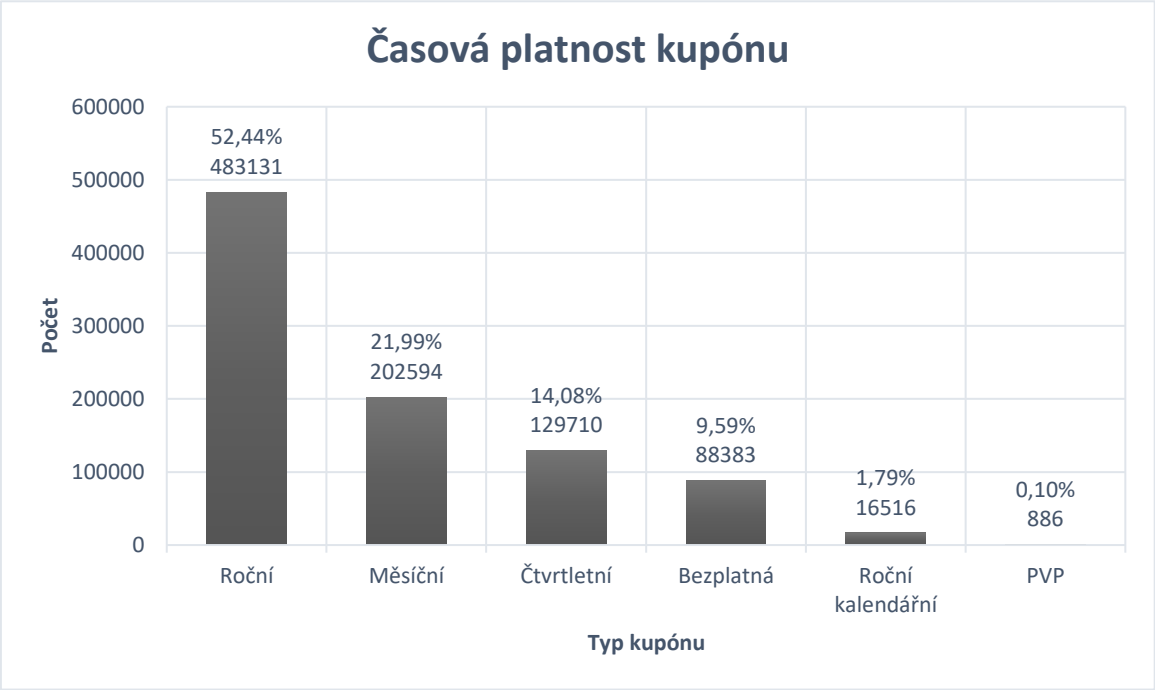


Obrázek 55 - Graf typy jízdného 1; zdroj: autor na základě dat z WL

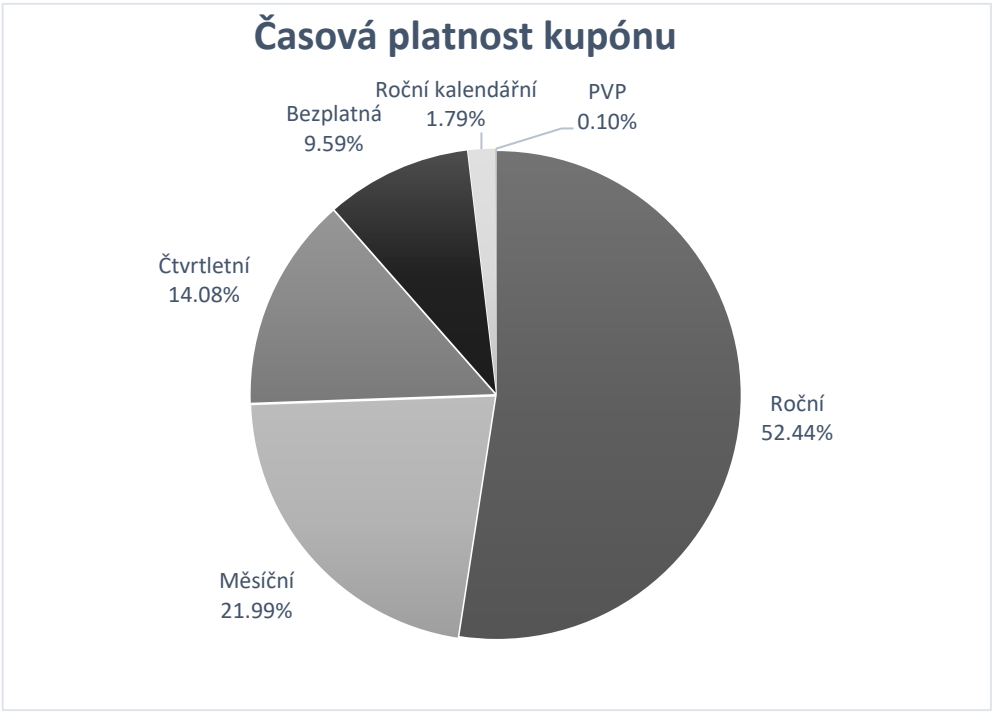


Obrázek 56 - Graf typy jízdného 2; zdroj: autor na základě dat z WL

Na grafech (Obrázek 55 a Obrázek 56) je jasně vidět, že mezi typy jízdného převládá kategorie *Občanské jízdné*. Jde o více jak polovinu záznamů kupónů v daném whitelistu. Dalšími výraznějšími kategoriemi jsou zde *Student 15-26*, *Junior 15-18*, *Senior 65+*, *Senior 60-65* a *Dítě 6-15*.

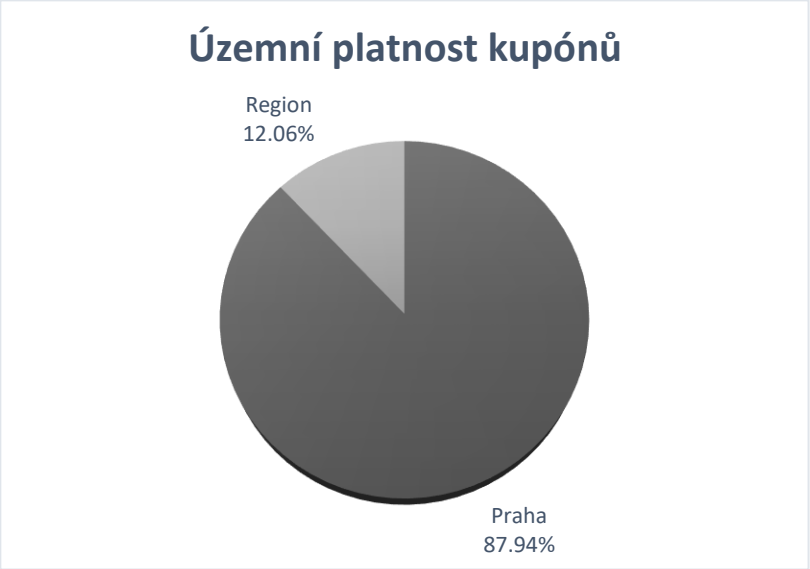


Obrázek 57 - Graf kupónů dle časové platnosti 1; zdroj: autor na základě dat z WL



Obrázek 58 - Graf kupónů dle časové platnosti 2; zdroj: autor na základě dat z WL

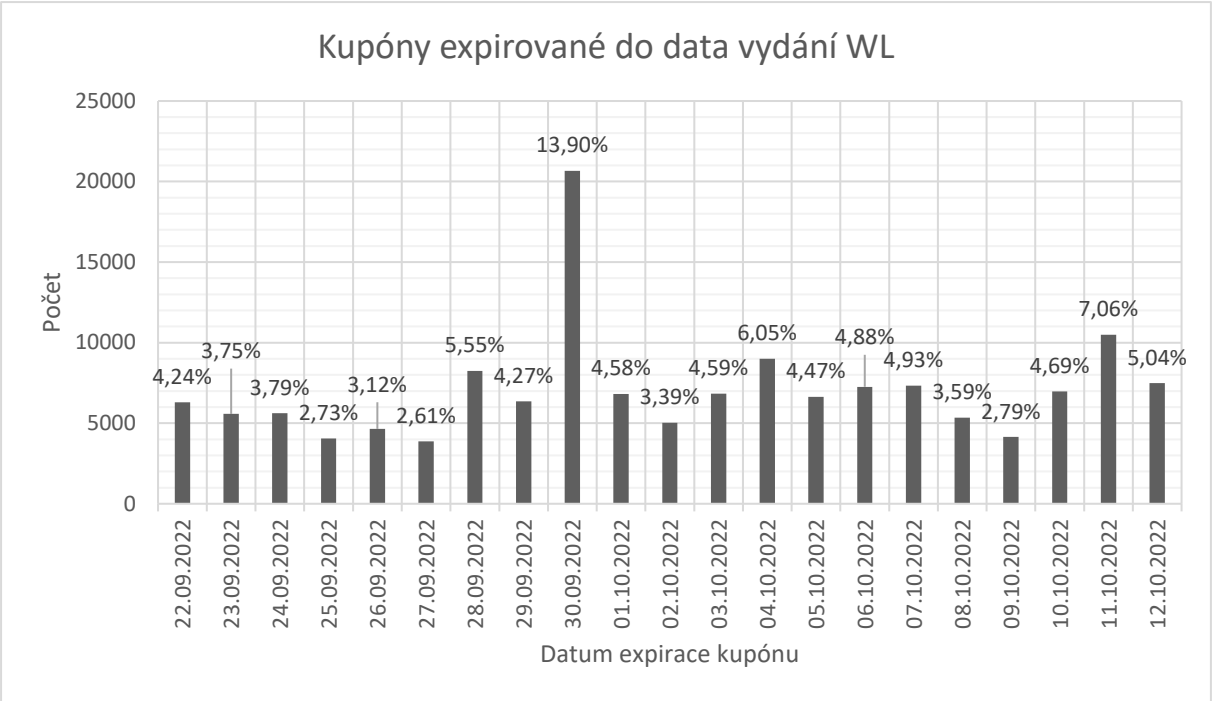
Na grafech (Obrázek 57 a Obrázek 58) je dobře znatelné, že většina tehdejších cestujících preferovala roční kupón. Jde o více jak polovinu záznamů kupónů v daném whitelistu. Více jak pětina kupónů je s platností na měsíc a čtvrtletní jízdné je zde zastoupeno na hodnotě přes čtrnáct procent. Necelá desetina záznamů obsahuje bezplatné jízdné.



Obrázek 59 - Graf územní platnosti kupónů; zdroj: autor na základě dat z WL

Dalším údajem v této kapitole je rozlišení kupónů dle územní platnosti. Z grafu (Obrázek 59) je s přehledem jasně viditelné, že ve většině záznamů daného whitelistu převládají časové jízdní kupóny pro Prahu. Jejich počet je 810 091. Regionální jízdné je zde zastoupeno pouze z dvanácti procent, což dělá absolutní počet ve whitelistu 111 129.

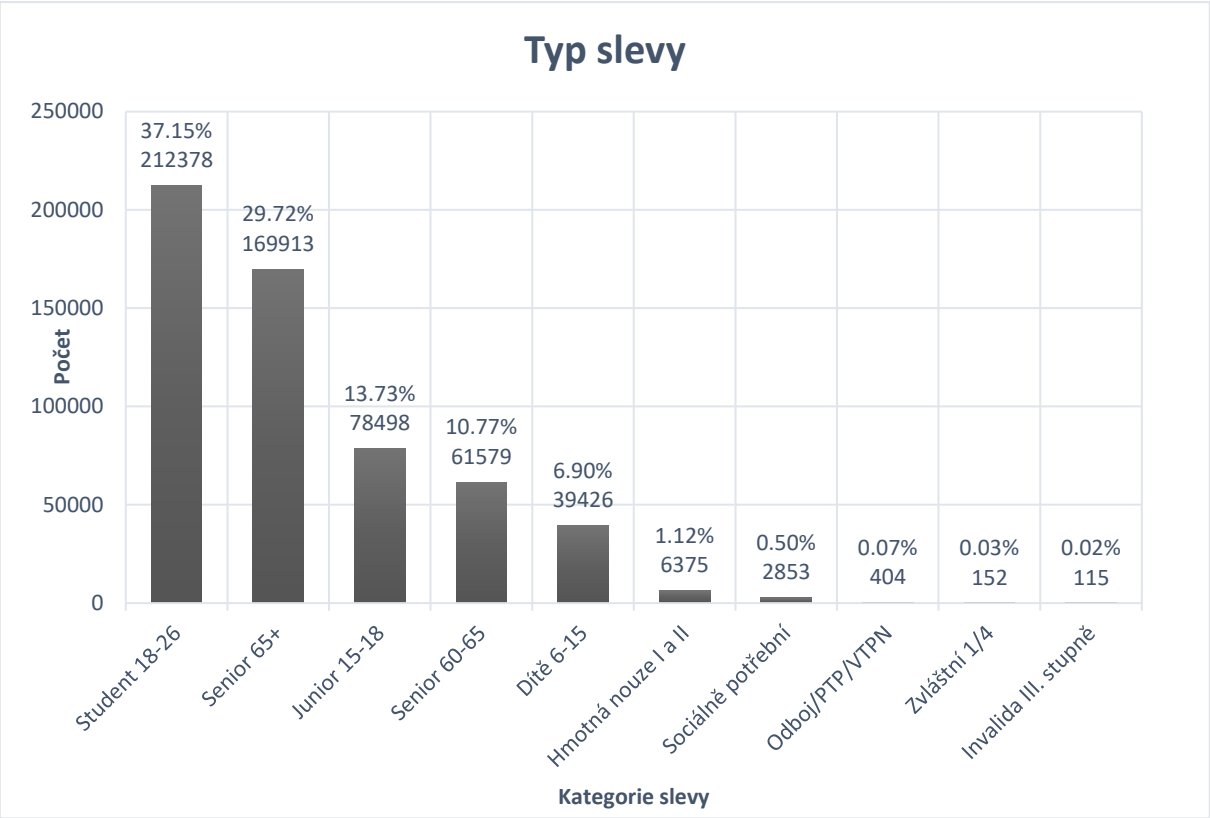
Posledními analyzovanými daty v kapitole jsou kupóny, které jsou k datu generování daného whitelistu již propadlé. Ze záznamů plyne, že v daném whitelistu jsou zachovány kupóny zhruba tři týdny dozadu. Početně jde o 148 721 kupónů a tvoří tak 16,14 % všech záznamů kupónů ve whitelistu. Jejich přehledný graf je uveden níže (Obrázek 60). Procentuální zastoupení v grafu je odvozeno od počtu všech propadlých kupónů ve whitelistu.



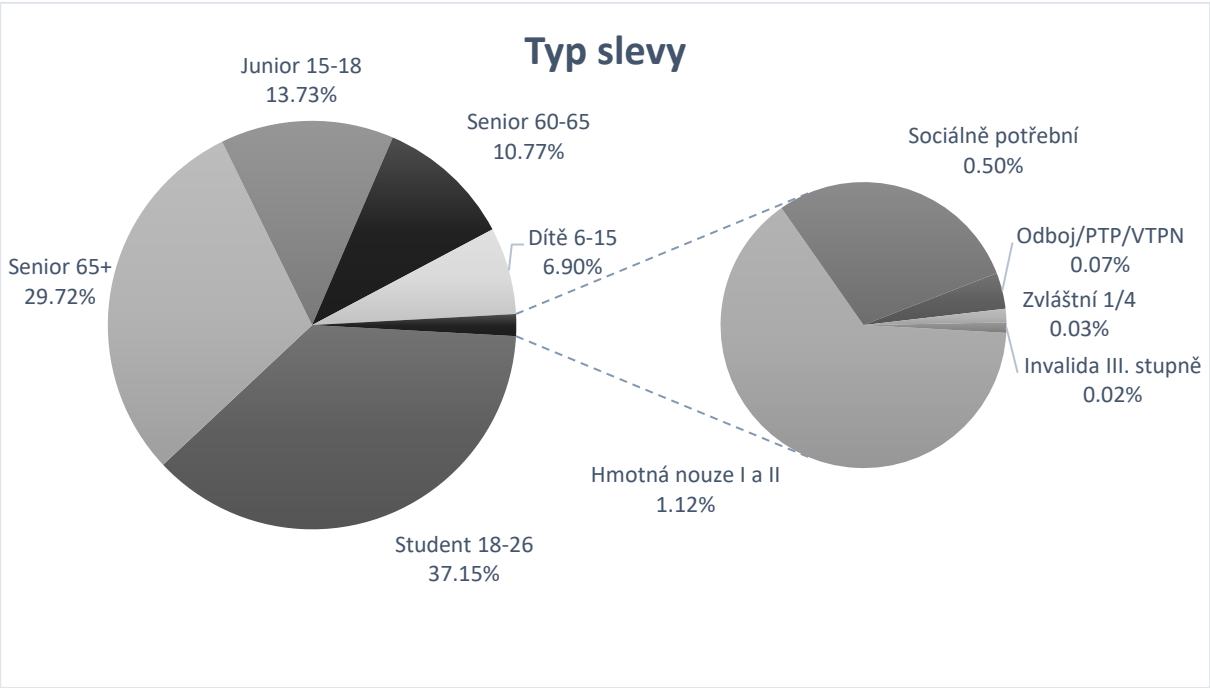
Obrázek 60 - Graf propadlých kupónů ve whitelistu; zdroj: autor na základě dat z WL

3.4.3. Informační analýza slev

Další analyzovanou částí jsou informace o slevách v daném whitelistu. Co se týče propadlých slev v čase generování whitelistu, tak jejich počet je na pouhých 178 záznamech, kterým končí platnost den před vygenerováním daného whitelistu, což bylo 12. 10. 2022. Daný whitelist je vygenerován k datu 13. 10. 2022 a je v něm celkově 571 713 záznamů slev. Jak plyne z definované struktury whitelistu, tak jediným zajímavým atributem nesoucím zajímavé popisné informace o slevách mimo časovou platnost je hodnota CP neboli Customer profile. Určuje typ slevy. Stejný atribut je přítomný i u kupónů, kde je obor hodnot CP vyšší. Mezi slevy logicky nepatří například kategorie občanského jízdného. Následující grafy ukazují rozvržení slev dle typu.



Obrázek 61 - Graf typy slev 1; zdroj: autor na základě dat z WL



Obrázek 62 - Graf typy slev 2; zdroj: autor na základě dat z WL

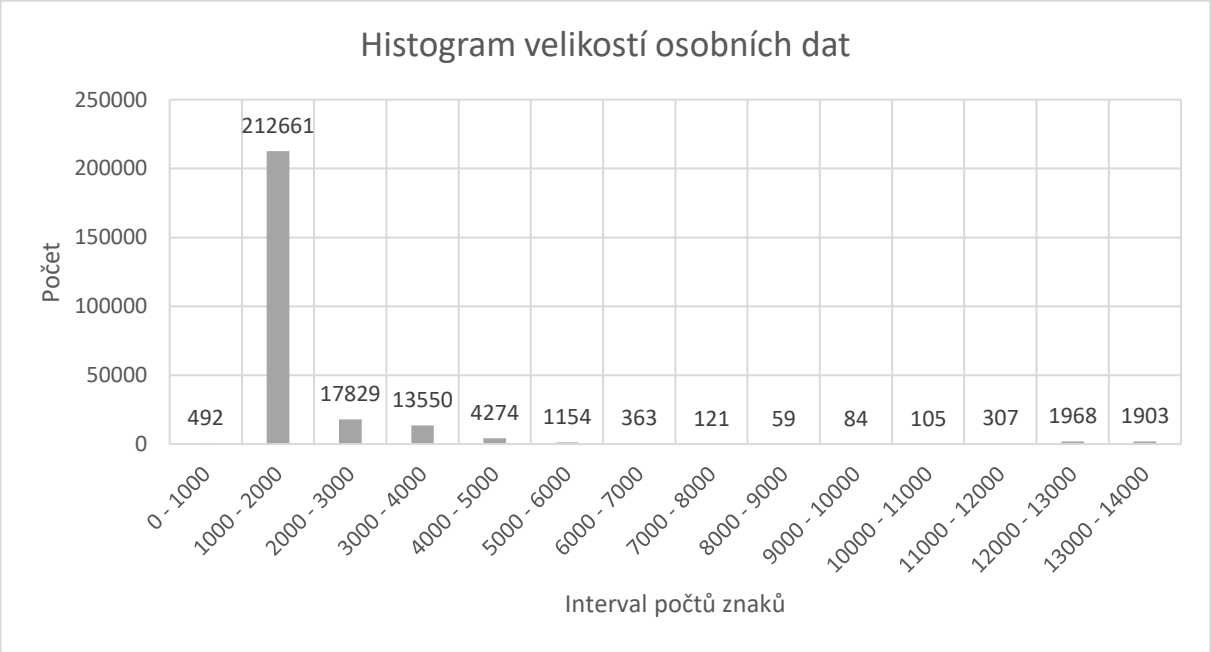
Z grafů (Obrázek 61 a Obrázek 62) je vidět, že mezi typy slev dominuje kategorie *Student 18-26* s 37,15 % a za ní kategorie *Senior 65+*, která má zastoupení 29,72 %. V menší míře jsou pak zastoupeny kategorie *Junior 15-18*, *Senior 60-65* a *Dítě 6-15*. Ostatní slevové kategorie jsou pak zastoupeny už jen v zanedbatelné míře kolem jednoho a méně než jednoho procenta. Jde o kategorie *Hmotná nouze I a II*, *Speciálně potřební*, *Odboj/PTP/VTPN*, *Zvláštní ¼* a *Invalida III. stupně*.

3.4.4. Analýza velikosti whitelistu

Poslední částí datové analýzy whitelistu je statistika velikosti dat ve whitelistu. Původní soubor absolutního whitelistu s datem 13. 10. 2022 má velikost necelých 1,5 GB ve formátu JSON. Po zpracování do podoby databáze pomocí Python skriptu má velikost cca 275 MB. Ze tří původních sekcí (v databázi entit) byla ohledně velikosti souboru nejobsáhlejší část sekce Clients. Přestože má pouze dva atributy, tak zašifrovaná osobní data byla na velikost nejnáročnější. Po vynechání osobních dat a jejich nahrazení informacemi o jejich délce se na první příčku nejnáročnějších dat přesunula sekce Contracts. Po rozdělení Contracts na Slevy a Kupóny jsou na špici datové náročnosti zhruba stejně část Kupóny a část Identifikátory. Kupříkladu, když se vygenerují všechny čtyři části whitelistu z databáze do CSV, tak velikosti jednotlivých souborů jsou následující.

- Identifikátory 141 MB
- Kupóny 84 MB
- Slevy 49 MB
- Klienti 5 MB

Je tedy výrazně znát vynechání osobních údajů klientů z whitelistu. Pro znázornění je k dispozici histogram zobrazující počty znaků osobních dat ve whitelistu.



Obrázek 63 - Histogram velikostí osobních dat; zdroj: autor na základě dat z WL

Z histogramu (Obrázek 63) je vidět, že naprostá většina záznamů osobních dat má počet znaků v rozsahu 1 000 – 2 000 znaků. Konkrétněji 78,07 % záznamů (celkem 198 985 z 254 870) má počet znaků v rozsahu 1 700 – 1 800 znaků. Pod 1 000 znaků je záznamů minimálně. Nezanedbatelné počty záznamů se ještě pohybují v rozsahu 2 000 – 5 000 znaků. Poté už počty záznamů výrazně klesají. Minimální záznam měl 42 znaků a maximální měl 13 722 znaků. Velikost osobních dat může být ovlivněna údaji jako jméno, příjmení, datum narození, kontaktní informace (může být například telefonní číslo nebo e-mail) a fotografie uživatele. Poslední zmiňovaný údaj je objemově nejnáročnější a zároveň může být datově velice různorodý. Kromě toho, že různé typy whitelistů ukládají buď černobílé či barevné fotografie, tak omezení na kladená fotografie uživatele jsou minimální. To znamená, že fotografie v systému můžou být různých kvalit, a tudíž i velikostí.

Závěr

Cílem práce bylo analyzovat a následně zkontrolovat veškerá data ve whitelistu. K tomu bylo nejprve nutné nalézt způsob, jak převést zdrojový JSON soubor do zpracovatelné podoby. Pro konkrétní postup řešení tohoto problému byl zvolen převod pomocí programovacího jazyka Python. Původní JSON soubor s whitelistem, který byl velký kolem 1,5 GB a byl prakticky nečitelný na průměrném uživatelském zařízení, se podařilo pomocí vytvořeného skriptu převést na běžný databázový soubor o velikost 275 MB. Pomocí softwarového nástroje DB Browser for SQLite byla možná již první analýza dat. Databáze byla vytvořena jako klasická relační databáze pomocí databázového jazyka SQL.

V následující části práce byla data zkontrolována a zpracována. Nejprve probíhala kontrola konzistence dat. Šlo primárně o kontrolu splnění předepsaných oborů hodnot pro jednotlivé atributy dle definované struktury whitelistu. Součástí kontroly bylo také hledání případných duplicit, kdy jedna duplicita byla opravdu nalezena. Následovala validace dat ve whitelistu, zda splňují tarifní a smluvní přepravní podmínky PID. Byly kontrolovány hlavně hodnoty atributů Tarif profile určující časovou platnost kupónů či slev a hodnoty atributů Customer profile určující kategorii cestujícího. Byly kontrolovány také platnosti identifikátorů, jejich vazby k osobním údajům, kombinace slev a kupónů či územní platnost časových kupónů.

V poslední části práce byla nad daty z whitelistu provedena statistická analýza, která přináší celkové a dopravně nejzajímavější informace o databázi jízdních dokladů v PID. Byly odhaleny nesmyslné hodnoty v datech, kdy k jednomu nosiči či jednomu zákazníkovi náleželo nejen mnoho časových kupónů, ale i nesmyslné množství slev. Obdobně jsou na tom i údaje o počtu identifikátorů k jednomu uživateli. Byla určena i konkrétní skladba nosičů identifikátorů, kdy přes dvě třetiny nosičů byla karta Lítačka a necelá třetina byla tvořena nosiči v podobě mobilní aplikace Lítačka. Tudíž nosiče v podobě In karty či bankovní karty byly z hlediska jejich počtu zanedbatelné.

Dále byla zpracována statistika skladby kupónů a slev. Přes polovinu záznamů kupónů tvořila kategorie Občanské jízdné. V menší míře pak studenti, senioři, junioři a děti. Bylo také zjištěno, že více jak polovina uživatelů dává přednost ročním kupónům před ostatními druhy. Čtvrtina záznamů pak obsahovala měsíční jízdné a v menší míře se objevovalo také čtvrtletní jízdné. Zajímavé je, že skoro devět z deseti kupónů ve zpracovávaném whitelistu je určeno pouze pro pražskou oblast integrovaného systému PID. Regionální jízdné zde tvoří dvanáct procent. Skladba slev kopíruje skladbu slevových kupónů, kdy nejvýznamnější slevovou kategorií jsou studenti ve věku 18 až 26 let. Hned za nimi jsou pak senioři ve věku na 65 let.

Poslední provedená statistická analýza byla zaměřena na velikost dat. Převedením JSON souboru do databázové formy bylo ušetřeno (pro další zpracování) přes jeden GB dat, a to zejména díky faktu, že osobní data nebyla do databáze zahrnuta z důvodu jejich šifrované podoby, která znemožňuje jejich hlubší analýzu. Po vynechaných osobních datech jsou velikostně nejnáročnější záznamy o identifikátorech a o kupónech.

Další možné navazující úpravy dané práce a doporučení k postupům či k podobě whitelistu vyplývající z diplomové práce vidím následující. Whitelist jako takový je přenášen v podobě TVL, ale pro práci s ním byl zpřístupněn soubor JSON, který z něj byl vygenerován pomocí zařízení MOS. Určitě by bylo možné vytvořit skript, který by dokázal převádět whitelist přímo z TVL do databáze. Tím by odpadl mezikrok v podobě JSON souboru.

Postupy využívání jazyka Python v této práci jsou základní, jelikož jsem s tímto jazykem pracoval prvně a naučil jsem se s ním pracovat pouze v rozsahu dostatečném pro potřeby zpracování této práce. Obdobně je tomu i s kódem, který převádí JSON whitelist do

databázové podoby. Věřím, že vytvořený skript by šel optimalizovat. V kódech jsou určité opakující se procedury převádění, které by šlo formulovat jako obecnou funkci a tu pak vyvolávat. Nicméně průběh kódu převedení whitelistu z JSON souboru do databáze probíhá minutu a půl, a tudíž se bod optimalizace kódu v rámci rozsahu práce stal bezpředmětný.

Jedním z poznatků této práce je, že číslo identifikátoru je dle poskytnutých zdrojů ke struktuře whitelistu nazývané pro každou entitu jinak. V jedné entitě se používá WLMOSIdentId a v další WLIdentID. Určitě je vhodné, aby se tento atribut všude nazýval stejně. V nejnovější podobě whitelistu už je tento název sjednocen.

Dalším bodem ke zlepšení aktuální podoby whitelistu je optimálnější zaznamenávání pásem v územní platnosti kupónu. Pásma jsou psána v abecedním pořadí, což nedává moc smysl (a výrazně to stěžuje datovou analýzu jazykem SQL). Dále jsou pásma v databázi oddělena středníky, což také není vhodný způsob pro další práci s daty.

Poslední doporučení nebo námět se týká samotné kontroly dat, analýzy dat a vyhodnocení statistik. Dokáži si představit sofistikovanější nástroje pro práci s whitelistem. Mohlo by vzniknout uživatelsky přívětivé prostředí (využívající GUI), které by datovou kontrolu whitelistu provedlo automatizovaně dle předepsaných pravidel. Stejně tak by se daly vyhodnocovat analýzy splnění podmínek PID nebo by se daly automaticky generovat statistické grafy.

Velice užitečný další krok v analýze whitelistu vidím v porovnávání vývoje dat ve whitelistu v čase. Dalo by se pak sledovat, jak se postupem času vyvíjí skladba jízdních dokladů, nebo jak moc narůstá počet uživatelů.

Díky provedené analýze a zpracování dat ve whitelistu jsem získal hlubší vhled do struktury a charakteristiky jízdních dokladů v Pražské integrované dopravě. Identifikoval jsem nejen technické nedostatky a potenciální problémy whitelistu, který je pilířem odbavovacího systému MOS, ale také možnosti pro budoucí optimalizaci a vylepšení správy dat. Využití moderních technologií a sofistikovaných nástrojů může výrazně zvýšit efektivitu a přesnost zpracování dat whitelistu. Případné navázání na práci v podobě implementace doporučených řešení pro kontrolu dat a generování statistik by mohlo výrazně usnadnit práci s daty a umožnit rychlejší reakci na změny v prostředí veřejné dopravy v rámci integrovaného systému PID.

Použité zdroje

[1] PRAŽSKÁ INTEGROVANÁ DOPRAVA. Odbavovací a informační zařízení ve vozidlech PID. Online. 2022. Dostupné z: https://pid.cz/wp-content/uploads/2022/07/Odbavovaci_a_informacni_zarizeni.pdf. [cit. 2024-04-04].

[2] OPERÁTOR ICT, A.S.; PRAŽSKÁ INTEGROVANÁ DOPRAVA. MOS – POŽADAVKY NA ODBAVOVACÍ ZAŘÍZENÍ: Struktura whitelist. Online. Verze 3.0. 2022. Dostupné z: https://zakazky.operatorict.cz/document_421/e58d7608d1bf45998f4dca1441a36bab-pr_4_technicka-specifikace_final-pdf. [cit. 2024-04-04].

[3] Python. Online. C2001-2024. Dostupné z: https://www.python.org/. [cit. 2024-04-26].

[4] SQL Tutorial. Online. C1999-2024. Dostupné z: https://www.w3schools.com/sql/. [cit. 2024-01-26].

[5] HARMS, Daryl D. a MCDONALD, Kenneth. Začínáme programovat v jazyce Python. Brno: Computer Press, 2003. ISBN 80-7226-799-X.

[6] PETKOVIĆ, Dušan. JSON Integration in Relational Database Systems. International Journal of Computer Applications. 2017, roč. 168, č. 5, s. 14-19. ISSN 0975 – 8887.

[7] PETKOVIĆ, Dušan. SQL/JSON standard: Properties and deficiencies. Datenbank-Spektrum. Online. 2017, roč. 17, č. 3, s. 277–287. Získáno z: doi:10.1007/s13222-017-0267-4

[8] SQL Server: JSON Data. Online. 2024. Dostupné z: https://learn.microsoft.com/en-us/sql/relational-databases/json/json-data-sql-server?view=sql-server-ver16. [cit. 2024-02-06].

[9] DB Browser for SQLite. Online. 2023. Dostupné z: https://sqlitebrowser.org/. [cit. 2024-04-26].

[10] PRAŽSKÁ INTEGROVANÁ DOPRAVA. Tarif PID. Online. Dostupné z: https://pid.cz/wp-content/uploads/ke-stazeni/tarif/tarif_PID_2023-04-01_komplet.pdf. [cit. 2024-04-09].

[11] MASTRODOMENICO, Rob. The Python Book. Hoboken, NJ; Chichester, West Sussex: Wiley, 2022. ISBN 978-1-119-57328-9.

[12] PRAŽSKÁ INTEGROVANÁ DOPRAVA. Smluvní přepravní podmínky PID. Online. Dostupné z: https://pid.cz/wp-content/uploads/ke-stazeni/tarif/SPP_PID_2021-08-01_komplet.pdf. [cit. 2024-04-09].