# Analýza úspěšnosti klasifikace vozidel

Bc. Darina Bajusová

Diplomová práce
2021

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav počítačových a komunikačních systémů

Akademický rok: 2020/2021

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Darina Bajusová** |
| Osobní číslo: | **A18559** |
| Studijní program: | **N3902 Inženýrská informatika** |
| Studijní obor: | **Počítačové a komunikační systémy** |
| Forma studia: | **Prezenční** |
| Téma práce: | **Analýza úspěšnosti klasifikace vozidel** |
| Téma práce anglicky: | **Vehicle Classification Success Analysis** |

## Zásady pro vypracování

1. Proveďte analýzu požadavků potřebných pro klasifikaci vozidel.
2. Zpracujte rozbor technologií a způsobů klasifikace vozidel pomocí senzorů integrovaných ve vozovce.
3. Navrhnete databázi pro ukládání testovacích dát.
4. Navrhněte způsob parametrizace klasifikačních nástrojů.
5. Navrhněte způsob vyhodnocení jejich úspěšnosti.
6. Zrealizujte navržený nástroj a získané výsledky vyhodnoťte.

Forma zpracování diplomové práce: **Tištěná/elektronická**

Seznam doporučené literatury:


1. BLAKE, Gregory. SQL Server 2017 – A Practical Guide for Beginners. CreateSpace Independent Publishing Platform, 2017. ISBN 978-1975875060.
2. BOEHM, Anne a Joel MURACH. Murach's C Sharp 2015. 6th ed. Fresno, CA, Mike Murach & Associates, 2016. ISBN 978-1890774943.
3. CONOLLY, Thomas, Carolyn E. BEGG a Richard HOLOWCZAK. Mistrovství – databáze: profesionální průvodce tvorbou efektivních databází. Brno, Computer Press, 2009. ISBN 9788025123287.
4. CORONEL, Carlos a Steven MORRIS. Database systems: design, implementation, and management. 13th ed. United States, Cengage Learning, 2019. ISBN 978-1337627900.
5. MOREIRA, Joao, Andre Carlos Ponce de Leon Ferreira CARVALHO, and Tomas HORVATH. A general introduction to data analytics. Hoboken, NJ, John Wiley, 2018. ISBN 978-1119296249.
6. NAGEL, Christian. Professional C Sharp 7 and .NET Core 2.0. Indianapolis, Wrox, John Wiley & Sons, 2018, xviii, 1368 s. ISBN 9781119449270.
7. SMITH, Jon P. Entity Framework core in action. Shelter Island, New York, Manning Publications Co., 2018. ISBN 978-1617294563.


Vedoucí diplomové práce: **doc. Ing. Zdenka Prokopová, CSc.**
Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce: **15. ledna 2021**
Termín odevzdání diplomové práce: **17. května 2021**

**doc. Mgr. Milan Adámek, Ph.D.** v.r.
děkan

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
DĚKAN
L.S.

**Ing. Miroslav Matýsek, Ph.D.** v.r.
ředitel ústavu

Ve Zlíně dne 15. ledna 2021

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 25.05.2021

Darina Bajusová, v. r.
podpis diplomanta

## ABSTRAKT

Táto diplomová práca sa zaoberá tvorbou nástroja, ktorý klasifikuje vozidlá na základe topológie podvozku. Takýto spôsob klasifikácie pomerne presne kopíruje používanú legislatívu v rôznych krajinách, avšak problémy nastávajú predovšetkým pri rozlíšení ľahkých vozidiel ako sú osobné vozidlá, dodávky a malé nákladné vozidlá. V rámci práce je prevedená analýza požiadaviek potrebných pre klasifikáciu vozidiel, rozbor použitých technológii a spôsoby klasifikácie vozidiel pomocou senzorov integrovaných vo vozovke. Súčasťou práce je návrh databáze pre ukladanie testovacích dát, návrh parametrizácie a vyhodnotenia úspešnosti klasifikačného nástroja. V poslednej časti sú prezentované výsledky zrealizovaného klasifikačného nástroja.



Kľúčové slová: klasifikácia vozidiel, parametrizácia, relačná databáza, MS SQL server, .NET Core, programovací jazyk C#, Entity Framework Core, LINQ, skriptovací jazyk Lua



## ABSTRACT

This diploma thesis deals with the creation of a tool that classifies vehicles based on chassis topology. This way of classification quite accurately copies the legislation used in different countries. However, problems occur especially in distinguishing light vehicles such as passenger cars, vans and light trucks. The work analyzes the requirements for vehicle classification, analysis of the technologies used and methods of vehicle classification using sensors integrated in the roadway. Part of the work is the design of a database for storing test data, design of parameterization and evaluation of the success of the classification tool. In the last part, the results of the implemented classification tool are presented.



Keywords: vehicle classification, parameterization, relational database, MS SQL server, .NET Core, programming language C#, Entity Framework Core, LINQ, scripting language Lua

Týmto by som sa veľmi rada poďakovala vedúcej práce doc. Ing. Zdenke Prokopovej, CSc., za odborný dohľad, rady a pripomienky, bez ktorých by sa táto práca nezaobišla. Moje poďakovanie patrí aj konzultantovi práce Ing. Janovi Görigovi a jeho kolegovi z firmy Cross Zlín, a. s. za ich cenné rady a čas, ktorý mi venovali pri spracovávaní tejto diplomovej práce a predovšetkým za poskytnutie príležitosti vypracovať danú tému. Veľká vďaka patrí aj rodine a priateľovi za trpezlivosť a podporu v štúdiu.

Prehlasujem, že odovzdaná verzia bakalárskej práce a verzia elektronická nahraná do IS/STAG sú totožné.

# OBSAH

# ÚVOD

V dnešnej dobe je doprava jednou z kľúčových tém spoločnosti. Jej neustály vývoj a s tým súvisiaci neustále sa zvyšujúci počet vozidiel so sebou prináša aj radu problémov. Tie vznikajú predovšetkým z nedodržiavania dopravných predpisov. Preto je dôležité, aby doprava bola adekvátne kontrolovaná. Kontrola dopravy si vyžaduje meranie rôznych parametrov z cestnej premávky. Získané parametre môžu pomôcť zlepšiť situáciu v rozvoji cestnej infraštruktúry. Medzi takéto parametre patrí napr. rýchlosť a hmotnosť vozidla, počet pohybujúcich sa vozidiel, dĺžka dopravnej zápchy, ale aj časová vzdialenosť medzi vozidlami. Zároveň jedným z veľmi dôležitých parametrov je trieda vozidla (kategória). Poznanie triedy vozidla má v určitých oblastiach zásadný vplyv pri rozhodovaní v rámci kontroly a riadenia dopravy. Z toho dôvodu je predmetom tejto práce klasifikácia vozidiel (kategorizácia).

Hlavnou úlohou práce je navrhnutie a vytvorenie nástroja (aplikácia) určeného pre klasifikáciu vozidiel. Klasifikácia vozidiel je realizovaná len na základe jedného parametra. Týmto parametrom je vzdialenosť medzi jednotlivými nápravami vozidla, t. j. rázvory. Takáto klasifikácia jednoducho splňuje požiadavky prevádzkovateľa na rozlíšenie rôznych druhov vozidiel.

Práca je rozdelená do dvoch častí. V teoretickej časti je predstavená problematika týkajúca sa klasifikácie vozidiel a s tým súvisiace spôsoby klasifikácie pomocou senzorov integrovaných vo vozovke. Ďalej sú rozobraté technológie, ktoré boli použité pri realizácii klasifikačného nástroja (základy databázových systémov, .NET Core a skriptovacie jazyky). Praktická časť zahŕňa analýzu požiadaviek potrebných pre klasifikáciu vozidiel, návrh relačnej databáze pre ukladanie testovacích dát, návrh parametrizácie a návrh vyhodnotenia úspešnosti klasifikácie. Následne je popísaná realizácia klasifikačného nástroja (konzolová aplikácia .NET Core) a sú prezentované dosiahnuté výsledky z testovania.

# I. TEORETICKÁ ČÁST

# 1 KLASIFIKÁCIA VOZIDIEL

Klasifikácia vozidiel môže byť založená na rôznych pravidlách. V súčasnosti existuje niekoľko klasifikačných štandardov, ktoré sa odlišujú predovšetkým v počte kategórii, resp. tried vozidiel. Tie sa zakladajú prevažne na funkčných vlastnostiach vozidla a jeho celkovej hmotnosti. Medzi najznámejšie európske klasifikácie patrí napr. EN 8+1 alebo EUR13. Naopak v Amerike sa často uvádza klasifikácia FHWA. Klasifikačný štandard EN 8+1 je najmenej selektívny a zahŕňa iba osem tried. Patria sem motorky, osobné vozidlá, osobné vozidlá s prívesmi, dodávky, autobusy, nákladné vozidlá, nákladné vozidlá s prívesmi, nákladné vozidlá s návesmi a iné vozidlá. Klasifikácia EUR13 (Obr. 1) je podrobnejšia a každá trieda zahŕňa len vozidlá s podobnou celkovou hmotnosťou vozidla a rozmermi. Avšak, aj táto klasifikácia ignoruje kategorizáciu podľa typu podvozku. Klasifikácia FHWA oproti predošlým klasifikáciám kladie dôraz aj na konfiguráciu náprav. Zároveň ale obsahuje vozidlá, ktoré sa nevyskytujú na európskych cestách. [1, 2]

| Vehicle Class | Classification |
|---|---|
| 1 | Car, Light Van<br>Light Good Vehicles |
| 2 | Rigid 2-Axle Truck |
| 3 | Rigid 3-Axle Truck |
| 4 | Rigid 4-Axle Truck |
| 5 | Rigid 2-Axle Truck & Trailer |
| 6 | Rigid 3-Axle Truck & Trailer |
| 7 | 2-Axle Tractor & 1-Axle Trailer |
| 8 | 2-Axle Tractor & 2-Axle Trailer |
| 9 | 2-Axle Tractor & 3-Axle Trailer |
| 10 | 3-Axle Tractor & 1-Axle Trailer<br>3-Axle Tractor & 2-Axle Trailer |
| 11 | 3-Axle Tractor & 3-Axle Trailer |
| 12 | Bus or Coach |

Obrázok 1. Klasifikácia vozidiel - EUR13 [2]

Klasifikácia sa vykonáva v rámci sčítania dopravy. Automatický sčítač dopravy klasifikuje vozidlá do kategórii. Ako klasifikátor môže slúžiť aj systém automatického váženia vozidiel v pohybe (WIM). Tieto systémy majú veľký význam pri ochrane ciest pred preťaženými vozidlami, s čím súvisí aj automatické pokutovanie vozidiel. Z toho dôvodu sa pre správne priradenie hmotnostných limitov (sú odlišné pre rôzne kategórie) vyžaduje veľmi presná klasifikácia váženého vozidla.

Okrem iného klasifikácia je dôležitá aj pri sledovaní vyťaženosti vozovky v rámci tried. Zdokumentovanie cestnej prevádzky má vplyv na vytvorenie ideálneho návrhu riešenia pri výstavbe novej cestnej infraštruktúry, resp. pri vyhľadávaní vhodných obchádzkových trás. Svoje uplatnenie si našla aj pri riadení dopravy v rámci uzavretia vozovky pre niektoré triedy vozidiel alebo tiež pri vyhodnocovaní miery hluku a znečistenia.

Vo výsledku klasifikovanie dopravy so sebou prináša niekoľko výhod. Napomáha odstraňovať dopravné zápchy, predlžuje životnosť ciest a zamedzuje porušovaniu pravidiel – zvyšuje sa bezpečnosť dopravy.

## 1.1 Vysokorýchlostné váženie vozidiel za jazdy

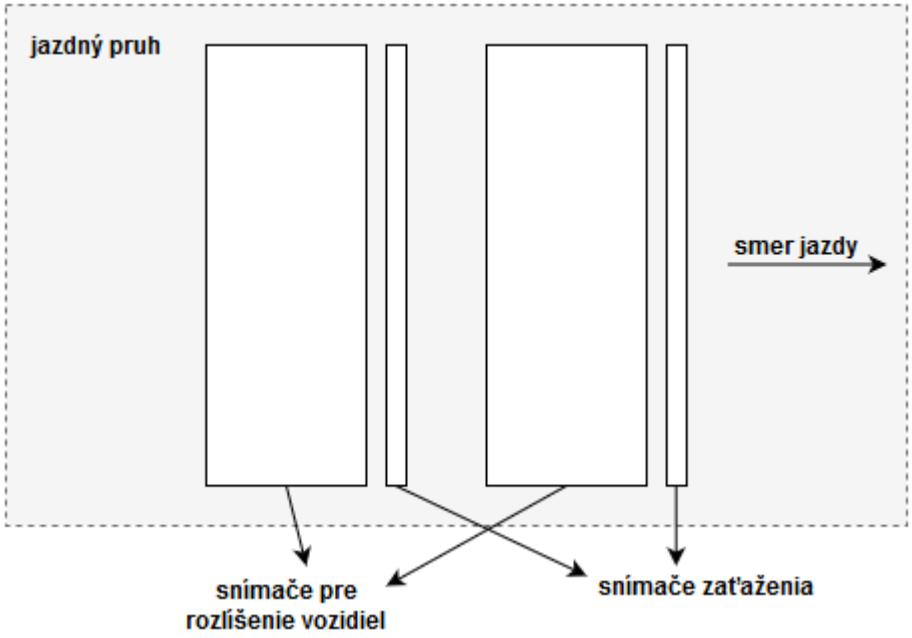
Systémy pre vysokorýchlostné váženie vozidiel za jazdy sú primárne určené na merania zaťaženia náprav a celkovej hmotnosti vozidla. Tieto systémy sú umiestnené v oblastiach s veľkým výskytom vozidiel ako sú napr. rýchlostné cesty, diaľnice a všade tam, kde je potrebné monitorovať dopravnú prevádzku. Na meranie sa využívajú rôzne typy senzorov, ktoré sú integrované vo vozovke. Súčasťou systému je aj kamerový systém slúžiaci k zaznamenávaniu situácií pri vážení a k rozpoznávaniu tabuliek s evidenčnými číslami vozidiel. Vážiace stanoviština sú dynamické a za predpokladu dodržiavania cestných predpisov nie sú vozidlá pri meraní v jazdnom pruhu nijak rýchlostne obmedzované.

Vývojom takýchto vážiacich systémov sa zaoberá aj miestna česká firma Cross Zlín, a.s., ktorá dnes patrí medzi popredných výrobcov nie len doma, ale aj vo svete. Konkrétne ide o vážiaci systém CrossWIM. Systém bol navrhnutý s dôrazom na presnosť, spoľahlivosť a jednoduchosť.

Systém WIM je ďalej rozobratý len z hľadiska jeho schopnosti klasifikácie. Jedná sa o presný klasifikátor, ktorý kladie veľký dôraz na presnosť samotnej klasifikácie.

### 1.1.1 Senzory integrované vo vozovke

Vážiaci systém má vo vozovke integrované dva druhy senzorov. Patria sem snímače pre rozlíšenie vozidiel a snímače zaťaženia. Obrázok (Obr. 2) zobrazuje základnú schému rozloženia týchto snímačov.




Obrázok 2. Snímače integrované vo vozovke [3]

Každý jazdný pruh pozostáva vždy minimálne z dvojice snímačov zaťaženia LINEAS KISTLER. Okrem základného dvojradového rozloženia týchto snímačov zobrazeného na obrázku (Obr. 2) existuje aj niekoľko ďalších konfigurácii. Ich výhodou je dosiahnutie presnejších výsledkov z váženia a získanie ďalších doplňujúcich informácii o vozidle. Ďalej je pred každým snímačom zaťaženia umiestnený snímač pre rozlíšenie vozidiel, t. j. indukčná slučka. [3]

### 1.1.2 Spôsoby klasifikácie

Z hľadiska klasifikácie vozidiel sú základom práve senzory uvedené na obrázku (Obr. 2). Klasifikácia môže prebiehať dvoma spôsobmi. Prvým spôsobom je automatická klasifikácia využitím indukčných slučiek, ktoré detekujú prítomnosť vozidla na vozovke. Slučky sú napojené na štvorkanálový slučkový detektor VEK S4C od spoločnosti FEIG, ktorý zachytáva signály z dvojice indukčných slučiek a po ich vyhodnotení poskytuje informácie o rýchlosti, dĺžke a kategórii vozidla. Dáta sú následne predané nadradenému systému. Pri takomto spôsobe klasifikácie sú vozidlá automaticky kategorizované len do základných tried EN 8+1. [4]

Takáto jednoduchá klasifikácia nemusí byť v istých prípadoch postačujúcou. Z toho dôvodu sa táto práca zaoberá rozšírenou klasifikáciou vozidiel, ktorá je založená na vzdialenostiach medzi jednotlivými nápravami vozidla, resp. jazdnej súpravy. Takéto riešenie predstavuje druhý spôsob klasifikácie – vytvorením vlastného algoritmu.

Tieto vzdialenosti medzi osami vozidla sa zisťujú z merania prechodu vozidla skrz snímače zaťaženia. Pri prechode vozidla sa na každom senzore generuje časová postupnosť pulzov, kde každá náprava predstavuje jeden pulz. Spracovaním týchto signálov sa zistia príslušné váhy a rázvory. Na výpočet rázvorov je najprv potrebné poznať priemernú rýchlosť vozidla, ktorá je daná vzťahom

$$v = \frac{\Delta s}{\Delta t}, \tag{1}$$

kde $\Delta s$ je dĺžka meracieho úseku, t. j. vzdialenosť počiatkov senzorov a $\Delta t$ je doba prejazdu vozidla cez merací úsek. Vzdialenosti medzi jednotlivými nápravami vozidla sú potom dané súčinom rýchlosti vozidla a doby prejazdu medzi dvoma konkrétnymi nápravami.

Rýchlosť je možné získať zo snímačov zaťaženia a indukčnými slučkami. Aby sa zabránilo chybe, porovnávajú sa tieto rýchlosti medzi sebou a ako kontrolná rýchlosť sa používa rýchlosť z dvojice indukčných slučiek.

# 2 ZÁKLADY DATABÁZOVÝCH SYSTÉMOV

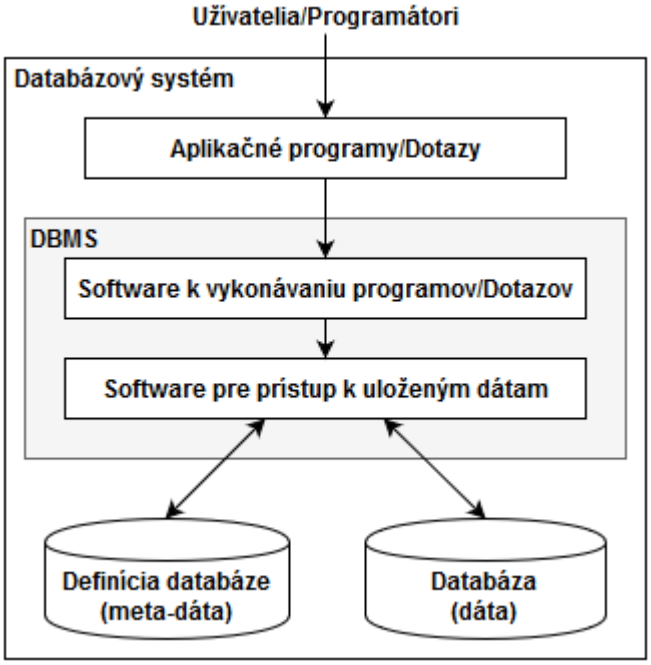
Databázové systémy sú v dnešnej dobe nevyhnutnou súčasťou nášho každodenného života. Väčšina z nás sa niekoľkokrát denne stretáva s činnosťami, ktoré zahrňujú určitú interakciu s databázou. Databázy zohrávajú rozhodujúcu úlohu takmer vo všetkých oblastiach, od zdravotníctva cez školstvo, sociálne média, elektronický obchod až po výskum. K vzniku databáz prispela najmä potreba spravovať enormné množstvo dát organizovaným a efektívnym spôsobom. [5, 6]

Pod pojmom dáta rozumieme údaje získané pozorovaním alebo meraním. Ide o nespracované fakty bez hlbšieho významu. Preto je cieľom dáta spracovať, a tým získajú požadovanú štruktúru a význam. Výsledkom sú informácie. [7]

## 2.1 Databázový systém

Databázový systém predstavuje celok, ktorý pozostáva zo štyroch základných častí:

- databáza,
- systém riadenia báze dát (DBMS),
- aplikačný program
- a užívatelia. [5]




Obrázok 3. Databázový systém [5]

### 2.1.1 Databáza

Databáza je organizovaný súbor navzájom súvisiacich dát. Každá databáza je navrhnutá a vytvorená tak, aby plnila špecifický účel a súčasne uľahčila ukladanie a manipuláciu s dátami. Databáza ukladá dva typy dát: dáta a meta-dáta. [6]

Meta-dáta definujú ako sú dáta koncového užívateľa integrované a spravované v databáze. Popisujú samotnú štruktúru databáze. Patrí sem ukladanie informácií ako sú napr. názvy, dátové typy a obmedzenia. [6]

### 2.1.2 Systém riadenia báze dát

Systém riadenia báze dát (DBMS) je softwarový systém, ktorý umožňuje užívateľom definovať, vytvárať a udržiavať databázu a zároveň riadi prístup k dátam uloženým v databáze. Slúži ako sprostredkovateľ medzi užívateľom a databázou skrz aplikačného programu, resp. dotazov. [7]

DBMS zlepšuje manipuláciu s databázou (vkladanie, mazanie, aktualizovanie a dotazovanie) a medzi jeho ďalšie dôležité funkcie patrí ochrana databázy, zdieľaný prístup a zaistenie integrity uložených dát. [5]

Pri realizácii klasifikačného nástroja bol ako DBMS použitý MS SQL Server.

#### *2.1.2.1 MS SQL Server*

MS SQL Server patrí medzi jeden z najznámejších DMBS. Bol vyvinutý pre správu relačných databáz spoločnosťou Microsoft. Je založený na jazyku SQL, čo je štandardizovaný programovací jazyk na interakciu s relačnými databázami. Microsoft zároveň implementoval vlastnú nadstavbu jazyka SQL, známu pod názvom T-SQL. Server je možné konfigurovať a spravovať zjednodušene prostredníctvom užívateľského grafického rozhrania SQL Server Management Studio. [8, 9]

V súčasnosti je aktuálnou verziou MS SQL Server 2019 a je podporovaný na platformách Windows, Linux a v kontajneroch Docker. K dispozícii sú podľa [8] štyri základné edície:

- Enterprise je plná verzia, ktorá obsahuje všetky funkcie SQL servera.
- Standard má na rozdiel od edície Enterprise menej dostupných funkcií a obmedzenia týkajúce sa počtu jadier procesora a veľkosti pamäte.
- Express je bezplatná verzia pre malé databázy. Maximálna veľkosť databáze je 10GB.

- Developer je ekvivalentom k Enterprise, ale slúži len na vývoj a testovanie databáz. Je zdarma.

### 2.1.3 Aplikačný program

Aplikačné programy sa najbežnejšie používajú na prístup a manipuláciu s dátami uloženými v databáze. S databázou komunikujú zasielaním požiadaviek DBMS. Systém riadenia báze dát žiadosti príjme, spracuje a náležite vykoná. Aplikačný program môže byť napísaný využitím vyšších programovacích jazykov. [6]

## 2.2 Relačná databáza

Relačná databáza je založená na relačnom dátovom modely, ktorý navrhol E. F. Codd v roku 1970. Dátový model definuje spôsob ukladania dát a väzby medzi nimi. [7]

Základom relačnej databáze je relácia. Relácia je tabuľka, ktorá pozostáva z pevného počtu stĺpcov a premenlivého počtu riadkov. Dáta sú tak logicky štruktúrované do skupiny tabuliek. Každý stĺpec reprezentuje jeden atribút a každý riadok jeden záznam. Záznam predstavuje kolekciu súvisiacich hodnôt jedného objektu. [7]

Pre relačnú databázu je dôležité, aby bol každý záznam v tabuľke jednoznačne identifikovateľný. K tomu slúžia kľúče relácie a podľa [7] rozlišujeme dva základné typy:

- Primárny kľúč je atribút, ktorý má za úlohu jednoznačne identifikovať záznamy v tabuľke. Ak je primárny kľúč zložený z dvoch alebo viacerých atribútov, hovoríme o zloženom primárnom kľúči. Každá tabuľka musí mať primárny kľúč.
- Cudzí kľúč je atribút, ktorý obsahuje hodnoty primárneho kľúča z inej tabuľky. Medzi tabuľkami tým vzniká väzba.

Väzby medzi tabuľkami:

- Väzba 1:1 vraví, že k jednému záznamu z jednej tabuľky existuje práve jeden záznam z druhej tabuľky. [6]
- Väzba 1:N vraví, že k jednému záznamu z jednej tabuľky môže byť priradených viac záznamov z druhej tabuľky. [6]
- Väzba M:N vraví, že k viacerým záznamom z jednej tabuľky môže byť priradených viac záznamov z druhej tabuľky. Tento vzťah sa realizuje cez pomocnú väzbovú tabuľku. [6]

### 2.2.1 Integritné obmedzenia

Integritné obmedzenia, resp. pravidlá zaisťujú úplnosť a správnosť dát v databáze. Dáta je možné pridávať do databáze, len ak splňujú vopred definované kritéria. Existuje niekoľko integritných obmedzení. [6, 7]

- Doménová integrita definuje množinu prípustných hodnôt, ktoré môže atribút nadobudnúť [7].
- Entitná integrita súvisí s primárnym kľúčom. Každý záznam v tabuľke musí obsahovať unikátny primárny kľúč, ktorý nesmie nadobudnúť prázdnu hodnotu. [7]
- Referenčná integrita zaisťuje správnosť vzťahov medzi dátami v súvisiacich tabuľkách. Hodnota cudzieho kľúča musí odpovedať hodnote primárneho kľúča v nadradenej tabuľke. [7]

### 2.2.2 Normalizácia

Normalizácia je proces, ktorého hlavným cieľom je eliminovať redundanciu dát v databáze. Preto je dôležité, aby tabuľky boli už pri návrhu štruktúrované vhodným spôsobom. K tomu slúžia pravidlá známe ako normálne formy. Za normalizovanú databázu sa považuje databáza, ktorá splňuje aspoň prvé tri normálne formy. [10]

- Prvá normálna forma (1NF) vyžaduje, aby každý atribút v tabuľke obsahoval len atomické hodnoty, t. j. hodnoty, ktoré sú ďalej nedeliteľné. [10]
- Druhá normálna forma (2NF) sa týka len tabuliek so zloženým primárnym kľúčom a požaduje, aby každý nekľúčový atribút bol plne závislý na celom primárnom kľúči a nie len od jeho časti. [10]
- Tretia normálna forma (3NF) je splnená, ak sú všetky nekľúčové atribúty navzájom nezávislé, t. j. všetky nekľúčové atribúty sú priamo závislé len na primárnom kľúči. [10]

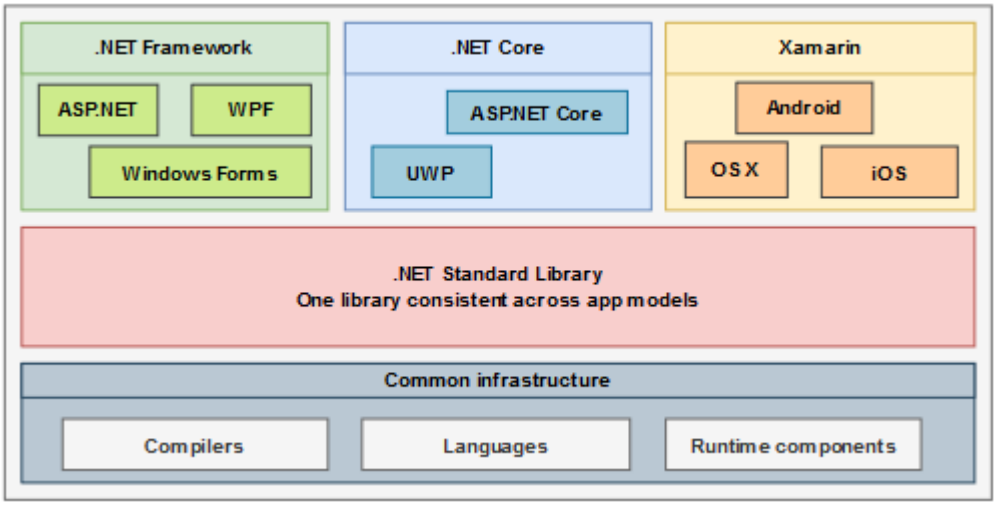
Zároveň musí platiť, že každá normálna forma musí splňovať podmienky normálnej formy, ktorá je o úroveň nižšie. [10]

# 3 .NET CORE

.NET Core je open-source, multiplatformný framework, ktorý beží na operačných systémoch Windows, Linux a MacOS. Je spravovaný spoločnosťou Microsoft a .NET komunitou. .NET Core je založený na .NET Frameworku (podpora len Windows) a umožňuje vytvárať rôzne aplikácie od mobilných, stolových cez webové až po IoT. Prvá verzia s obmedzenou funkčnosťou bola vydaná v roku 2016 a v súčasnosti je poslednou LTS verziou .NET Core 3.1. V novembri 2020 bol ako nástupca (.NET Core a .NET Frameworku) vydaný .NET 5.0. [11]

.NET Core plne podporuje vývoj v niekoľkých programovacích jazykoch ako je napr. C#, F#, VB a iné. Na rozdiel od .NET Frameworku je .NET Core výkonnejší, lepšie škálovateľný a môže byť nasadený aj v kontajneroch Docker. Jeho ďalšou vlastnosťou je modularita. Aplikácie tak môžu byť vytvárané výlučne len s knižnicami, ktoré sú potrebné. Knižnice sú poskytované v podobe balíkov NuGet. [11, 12, 13]

.NET Core je kompatibilný s .NET Frameworkom a Xamarinom. Už pri jeho budovaní bolo cieľom dosiahnuť, aby všetky implementácie .NETu mohli zdieľať rovnaké knižnice. Preto bola zavedená štandardná knižnica .NET Standard Library, vďaka ktorej je možné všetky knižnice napísané podľa tohto štandardu nasadiť na akúkoľvek platformu. .NET Standard Library je spolu so spoločnou infraštruktúrou súčasťou zjednoteného .NET ekosystému. [12]




Obrázok 4. Štandardizácia prostredníctvom .NET Standard Library [12]

Súčasťou .NET Core je rozhranie príkazového riadku (CLI). Obsahuje množstvo vstavaných príkazov, ktoré umožňujú vykonávať rôzne operácie od generovania projektov cez kompiláciu, spustenie až po správu migrácii a balíčkov NuGet. [12]

## 3.1 Programovací jazyk C#

C# je objektovo orientovaný jazyk, ktorý je najpoužívanejším programovacím jazykom v .NETe. Základ našiel v jazykoch C++ a Java. Bol navrhnutý tak, aby bol jednoduchý, moderný, ľahko použiteľný a výkonný. [14]

Zároveň je C# typovo bezpečný jazyk čo znamená, že jednotlivé inštancie typov musia dodržiavať presne definované pravidlá a žiadna operácia nesmie viesť k nedefinovanému správaniu. Kontrola prebieha už v dobe kompilácie (statická typová kontrola). Súčasťou C# je automatická správa pamäte. Garbage Collector vyhľadáva úseky pamäte, ktoré už viac nie sú programom využívané a zabezpečí ich uvoľnenie pre ďalšie použitie. [15, 16]

### 3.1.1 Verzie

C# je programovací jazyk, ktorý sa neustále vyvíja a jeho posledná verzia vydaná ako súčasť .NET Core 3.0 je C# 8. Nasledujúci obrázok zobrazuje prehľad verzií a ich nové, resp. vylepšené funkcie. [16]

| Verzia | Rok vydania | Nové funkcie |
|---|---|---|
| C# 1.0 | 2002 | triedy, štruktúry, rozhrania, udalosti, vlastnosti, výrazy, atribúty, delegáty |
| C# 2.0 | 2005 | generiká, statické triedy, parciálne triedy, anonymné metódy, typ nullable, |
| C# 3.0 | 2007 | lambda výrazy, LINQ, implicitný typ var, anonymné typy, metódy rozšírenia |
| C# 4.0 | 2010 | typ dynamic, pomenované argumenty, voliteľné parametre, |
| C# 5.0 | 2012 | asynchrónne funkcie, atribút caller info |
| C# 6.0 | 2015 | filtrovanie výnimiek, operátor nameof, operátor null-condition |
| C# 7.0 | 2017 | premenné out, tuples,lokálne funkcie, pattern matching, návratové hodnoty Ref |
| C# 8.0 | 2019 | deklarácia using, vylepšené vzory, modifikátor readonly u štruktúr, statické lokálne funkcie |
| C# 9.0 | 2020 | záznamy, vlastnosť init-only, vylepšené porovnávacie vzory |

Obrázok 5. Verzie jazyka C# [16]

Prvé zásadné zmeny nastali už na začiatku s príchodom generík. Generiká umožňujú navrhovať univerzálne triedy a metódy bez bližšej špecifikácie typov. Typy sú definované v okamihu vytvorenia inštancie. [16]

Hlavným vylepšením v C# 3.0 bolo predstavenie jazyka LINQ spolu s lambda výrazmi a anonymnými typmi. LINQ zjednocuje syntax pre písanie dotazov v C#. [16]

Lambda výrazy sú anonymné funkcie a súčasťou každého lambda výrazu je operátor lambda =>, ktorý pozostáva z dvoch častí. Ľavá strana pred operátorom lambda je vstupom a pravá strana definuje výraz. Zároveň nie je potrebné špecifikovať typ vstupu. [16]

Anonymný typ je referenčný typ, ktorý je definovaný pomocou premennej typu var. Môže obsahovať niekoľko vlastností (určené len na čítanie). Názvy a typy vlastností objektu sú generované kompilátorom automaticky. Anonymné typy boli vytvorené predovšetkým za účelom selekcie dát. [16]

S príchodom C# 4.0 boli predstavené voliteľné parametre. Voliteľné parametre sa uvádzajú vždy na konci zoznamu parametrov a môžu byť použité napr. v metódach a delegátoch. Sú definované predvolenou hodnotou, ktorá sa použije v prípade, ak nepovinný parameter nie je k dispozícií. Užitočné môžu byť napr. pri preťažení metód. [16]

C# 6.0 priniesol reťazcovú interpoláciu, ktorá dovoľuje vkladať premenné priamo do reťazca. Pred reťazec sa vkladá znak dolára a do zložených zátvoriek vnútri reťazca premenná (namiesto indexu). Takýto zápis plne nahrádza string.Format. Zmeny nastali aj pri filtrovaní výnimiek. Rozhodovanie len na základe typu zachytenej výnimky sa rozšírilo aj o vonkajší stav. [16]

Niektoré situácie si vyžadujú použiť štruktúru, ktorá obsahuje viac dátových prvkov. V C# 7.0 došlo k vylepšeniu dátovej štruktúry tuple. Jej najčastejšie využitie je pri návrate viacerých hodnôt z metódy (náhrada za parameter out). Prvky v tuple môžu byť zastúpené rôznymi dátovými typmi a definované vlastným názvom namiesto predvolených (Item1, Item2, ...). Vlastné názvy zjednodušujú prístup k jednotlivým prvkom. [16]

## 3.2 Entity Framework Core

Entity Framework Core (EF Core) je knižnica, ktorá poskytuje objektovo-orientovaný prístup k relačným databázam v prostredí .NETu (bola použitá pri vývoji klasifikačného nástroja). Ide o novú odľahčenú, rozšíriteľnú, open-source verziu Entity Frameworku, ktorá bola vydaná v roku 2017 ako súčasť .NET Core 2.0. Rovnako ako aj iné knižnice .NETu je aj EF Core multiplatformný. [17]

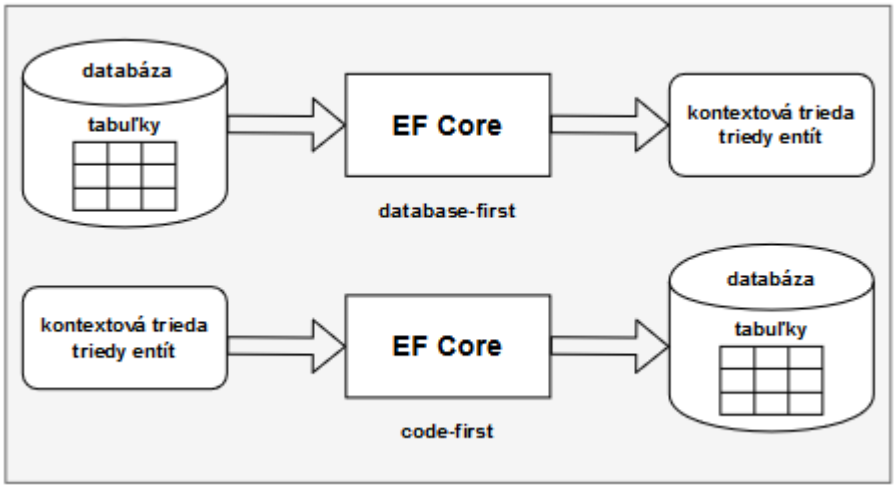
EF Core je založený na objektovo-relačnom mapovaní (ORM). ORM je typ programovacej techniky, prostredníctvom ktorej je možné pracovať s dátami objektovo-orientovaným spôsobom. Princíp spočíva v mapovaní. Tabuľka relačnej databáze reprezentuje triedu v .NETe, stĺpec, resp. atribút tabuľky odpovedá vlastnosti triedy (property) a riadok tabuľky predstavuje jeden element, resp. objekt .NET kolekcie. [17]

EF Core používa pri mapovaní objektov model. Model pozostáva z kontextovej triedy a tried entít. Trieda entity je klasická trieda .NETu, ktorá definuje jednu tabuľku v databáze. Kontextová trieda dedí z triedy DbContext a obsahuje informácie, ktoré EF Core potrebuje k mapovaniu, t. j. obsahuje sety entít, ktorých vlastnosti sa mapujú do stĺpcov tabuliek v databáze. Zároveň inštancia tejto kontextovej triedy reprezentuje reláciu s databázou, a tým je možné manipulovať s databázou. [17, 18]

### 3.2.1 Možnosti prístupu

Entity Framework Core poskytuje podľa [17] dva základné prístupy:

- Database-first je prístup, ktorý umožňuje pristupovať k už existujúcej databáze. Príkaz Scaffold-DbContext skontroluje databázovú schému a na jej základe EF Core vytvorí triedu kontextu a triedy entít.
- Code-first využíva opačný prístup. Ako prvé je potrebné navrhnúť triedy entít a triedu kontextu. Spustením príkazu pre migráciu vygeneruje EF Core sadu migračného kódu, z ktorého sa vytvorí databáza. Migrácie sú štandardným spôsobom pre vytváranie a aktualizovanie databáze z EF Core.




Obrázok 6. Prístupy k databáze [18]

### 3.2.2 Spôsoby konfigurácie

Na vytvorenie modelu alebo jeho modifikáciu je možné využiť tri prístupy:

- By Convention je prístup, pri ktorom EF Core automaticky nakonfiguruje základné funkcie – potreba zachovať vopred definované štandardy. Tento prístup je ľahko a rýchlo implementovateľný, ale u zložitejších návrhov databáze je častokrát nepostačujúci a je nutné ho doplniť využitím aspoň jedného z nasledujúcich prístupov. [17]
- Data Annotations používa na doplnenie návrhu databáze špecifické typy .NET atribútov, ktoré sa aplikujú na triedy alebo ich vlastnosti. Ide napr. o nastavenie maximálne dĺžky reťazca, zakázanie prázdnej hodnoty vlastnosti a iné. [17]
- Fluent API má najkomplexnejšiu sadu príkazov a obsahuje niekoľko funkcií, ktoré sú dostupné len týmto prístupom. Zmeny sa realizujú vo fáze modelovania v metóde OnModelCreating. [17]

By Convention predstavuje základný prístup s najnižšou prioritou, a preto môže byť jeho konfigurácia prepísaná, resp. doplnená prístupom Data Annotations alebo Fluent API. Naopak Fluent API má najvyššiu prioritu. [17]

## 3.3 LINQ

Language Integrated Query (LINQ) je dotazovací jazyk, ktorý je priamo integrovaný do syntaxe programovacích jazykov .NETu (C#, VB a iné). Ide o unifikovaný spôsob dotazovania sa v .NETe a podľa [19, 20] existuje niekoľko implementácii LINQ:

- LINQ to Objects slúži na dotazovanie sa nad .NET kolekciami uloženými v pamäti.
- LINQ to DataSet umožňuje využiť dotazy LINQ nad ADO.NET datasetmi.
- LINQ to SQL bol prvý pokus Microsoftu o využitie ORM. Podporuje vykonávanie dotazov len nad databázami využívajúcimi MS SQL Server.
- LINQ to Entities je obdoba LINQ to SQL, ale úlohu ORM plní EF, resp. jeho novšia verzia EF Core. Okrem toho už nie je viazaný len na MS SQL Server.
- LINQ to XML slúži pre písanie dotazov nad súbormi XML.

EF Core používa LINQ ako primárny spôsob dotazovania sa. V .NETe predstavuje takmer patričnú náhradu za jazyk SQL. Pri dotazovaní sa odkazuje na databázové objekty cez kontextovú triedu a triedy entít. Dotazovanie skrz LINQ je oveľa jednoduchšie, podporuje sa objektovo-orientovaný prístup a zároveň je aj zdrojový kód ľahko čitateľný. [17, 21]

### 3.3.1 Operátory

K zápisu LINQ dotazu sa využívajú operátory alebo im odpovedajúce kľúčové slová v C#, ak také slovo existuje. Napr. operátor Where odpovedá kľúčovému slovu where. Operátormi sa označujú metódy, ktoré umožňujú manipuláciu s dátami a jasné indikujú účel. [17]

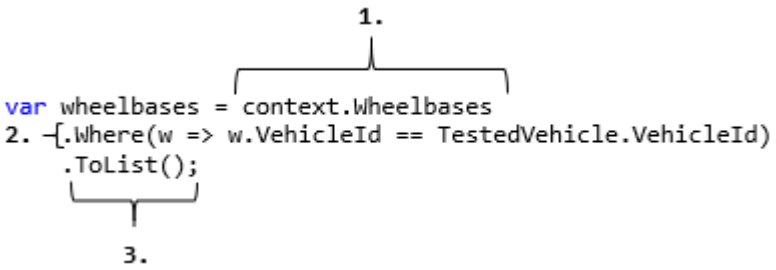
Tabuľka 1. Základné operátory LINQ [17]

| **Účel** | **Operátory** |
|---|---|
| Triedenie | OrderBy, OrderByDescending, Reverse |
| Filtrovanie | Where, Take, Skip |
| Výber prvku | First, FirstOrDefault |
| Projekcia | Select |
| Zoskupenie | GroupBy |
| Agregácia | Max, Min, Sum, Count, Average |
| Booleovské testy | Any, All, Contains |

### 3.3.2 Syntax

LINQ uvádza dve základné možnosti syntaxe.

Syntax method/lambda používa reťazové volanie metód, kde niektoré metódy si vyžadujú rozšírenie o lambda výrazy. Táto syntax je podobná syntaxi jazyka C# a bola použitá pri dotazovaní sa v rámci klasifikačného nástroja. Nasledujúci obrázok zobrazuje príklad takéhoto LINQ dotazu. [17]




Obrázok 7. Method/lambda syntax

Konkrétne sa jedná o dotazovania v databáze skrz EF Core. Dotaz pozostáva z troch hlavných častí. Prvá časť je kľúčovou. context je inštanciou kontextovej triedy (spája EF Core s databázou) a odvoláva na databázovú tabuľku (WheelBases). Druhá časť je tvorená sadou príkazov, ktoré definujú čo sa bude diať nad tabuľkou. Pozostáva prevažne z LINQ operátorov a prípadne aj iných doplňujúcich metód. Posledná časť je tvorená príkazom, ktorý spustí preklad LINQ dotazu do jazyka poskytovateľa databáze a jeho následne vykonávania v databáze. Ide o príkazy ako ToList, ToDictionary alebo operátory LINQ First, Any a iné. [17]

Query syntax je syntax založená na podobnosti s jazykom SQL. Primárne využíva kľúčové slová doplnené o potrebné operátory. Dotaz štandardne začína kľúčovým slovom from a končí slovom select. Pri kompilácii kódu sa takto napísaný dotaz musí najprv preložiť na volanie metód, čo vyvolávajú štandardné operátory uvedené v tabuľke (Tab. 1). Z toho dôvodu môžu byť volané aj priamo použitím prvej syntaxe Method/Lambda. [17, 22]

```
var wheelbases = (from wb in context.Wheelbases
                  where wb.VehicleId == TestedVehicle.VehicleId
                  select wb).ToList();
```

Obrázok 8. Query syntax

# 4 SKRIPTOVACIE JAZYKY

Skriptovací jazyk je jednoduchý programovací jazyk, ktorý sa používa na písanie spustiteľného zoznamu príkazov (skript). Skriptovacie jazyky sú určené na integráciu a komunikáciu s inými programovacími jazykmi. Na rozdiel od programovacích jazykov sú skriptovacie jazyky jazykmi interpretovanými. Interpreter kontroluje a vykonáva príkazy zdrojového kódu postupne za behu programu (dynamická typová kontrola). [23]

V prípade potreby úpravy algoritmu sú skripty jednoducho a rýchlo modifikovateľné. V porovnaní s kompilovaným programami je nevýhodou pomalší beh programu, a preto sa obvykle používajú na tvorbu komponent, ktorými sa rozšíri už existujúca aplikácia napísaná v akomkoľvek štandardnom programovacom jazyku. [23]

Klasifikačný nástroj priraďuje vozidlám podtriedy na základe pravidiel, ktoré sú definované v skripte napísanom v jazyku Lua.

## 4.1 Vložiteľné skriptovacie jazyky

Vložiteľný skriptovací jazyk (embedded scripting language) je jazyk, ktorého kód napísaný v takomto jazyku je vložený do hostiteľského programu aplikácie (pracuje zabudovane). Medzi vložiteľným skriptovacím jazykom a hostiteľským programom musí prebiehať obojsmerná komunikácia. Hostiteľský program tak môže volať funkcie na vykonanie časti kódu skriptu, manipulovať s premennými skriptovacieho jazyka, registrovať funkcie a iné. [24]

### 4.1.1 Skriptovací jazyk Lua

Lua je malý, rozšíriteľný, prenositeľný, výkonný programovací jazyk, ktorý bol špeciálne navrhnutý ako jazyk vložiteľný do hostiteľského programu. Podporuje procedurálne programovanie, funkčné programovanie, objektovo orientované programovanie a dátovo riadené programovanie. [24]

Lua je dynamický jazyk s automatickú správou pamäte. Je vykonávaná interpretáciou bytového kódu na virtuálnom stroji, ktorý je založený na registroch. Lua je implementovaná ako knižnica napísaná v programovacom jazyku C, má otvorený zdrojový kód a môže byť použitá za akýmkoľvek účelom. Svoje uplatnenie si našla v rôznych oblastiach, ale predovšetkým v hrách. Aktuálnou verziou je Lua 5.4. [24]

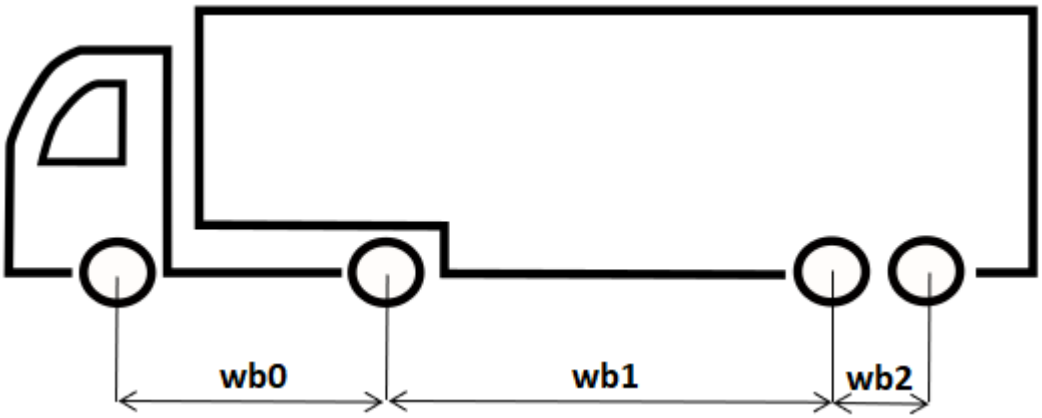
Prvotne bola Lua navrhnutá len na rozšírenie programov napísaných v C/C++. Avšak, vďaka jednoduchému a dobre zdokumentovanému API, ktoré umožňuje silnú integráciu s kódom napísanom aj v inom programovacom jazyku sú takéto programy Luou ľahko rozšíriteľné. Zároveň je možné Luu ľahké rozšíriť aj o knižnice napísané v iných jazykoch ako je C#, Java, Python a ďalšie. [24]

Lua je pomerne jednoduchým programovacím jazykom, ktorý je aj napriek nedostatku dátových štruktúr veľmi populárny. Náhradou za chýbajúce štruktúry je tabuľka, pomocou ktorej sa vytvárajú polia, listy atď. Tabuľka je spolu s odpovedajúcimi meta-tabuľkami základným prvkom jazyka Lua. [25]

# II. PRAKTICKÁ ČÁST

# 5 ANALÝZA POŽIADAVIEK

Hlavným cieľom práce je navrhnúť a vytvoriť nástroj, pomocou ktorého bude možné klasifikovať vozidlá. Klasifikácia kladie dôraz na topológiu podvozku, a preto bude prebiehať len na základe vzdialenosti medzi jednotlivými nápravami vozidla. Nasledujúci obrázok zobrazuje takto definované vozidlo a použité značenie bude v práci používané.




Obrázok 9. 4-nápravové vozidlo

## 5.1 Testovacie dáta

K návrhu, zrealizovaniu a otestovaniu klasifikačného nástroja sú nevyhnutnou súčasťou testovacie dáta. Dáta sú uložené v .csv súboroch a obsahujú informácie o vozidlách. Každý záznam (vozidlo) zahŕňa údaje, ako identifikačné číslo vozidla *id*, čas prejazdu vozidla *timestamp*, identifikačné číslo skupiny *ucid* a vzdialenosti medzi jednotlivými nápravami vozidla udávané v milimetroch *wb0, wb1, ..., wb14*. Jediným chýbajúcim údajom bolo identifikačné číslo podtriedy *subclassid*, a preto bolo potrebné každé vozidlo na základe jeho snímky z prejazdu najprv oklasifikovať a podtriedu mu doplniť. Z hľadiska klasifikácie má význam len identifikačné číslo vozidla, vzdialenosti a podtrieda.

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | id,timestamp,ucid,subclassid,wb0,wb1,wb2,wb3,wb4,wb5,wb6,wb7,wb8,wb9,wb10,wb11, | | | | | | | | |
| 2 | 7119213,01/14/2020 14:36:02,11,20,4092,5603,714,,,,,,,,,, | | | | | | | | |
| 3 | 7119016,01/14/2020 14:28:00,11,17,5509,7523,1297,,,,,,,,,, | | | | | | | | |
| 4 | 7118729,01/14/2020 14:17:14,11,20,3313,4206,846,,,,,,,,,, | | | | | | | | |
| 5 | 7118627,01/14/2020 14:13:34,11,20,3308,3866,491,,,,,,,,,, | | | | | | | | |
| 6 | 7118613,01/14/2020 14:12:41,11,21,2856,4079,601,,,,,,,,,, | | | | | | | | |

Obrázok 10. Testovacie dáta

Dáta boli poskytnuté firmou CROSS Zlín a.s. a z dôvodu GDPR snímky z prejazdu vozidiel nie sú k dispozícii a nebudú ani súčasťou práce.

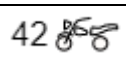
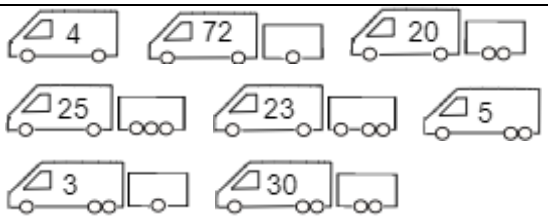
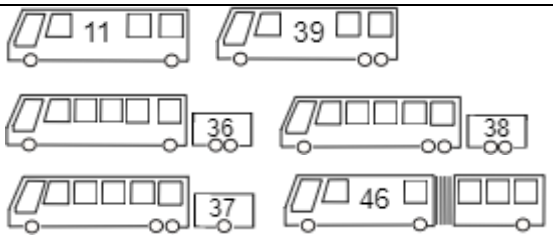
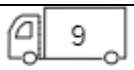
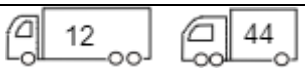
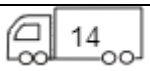
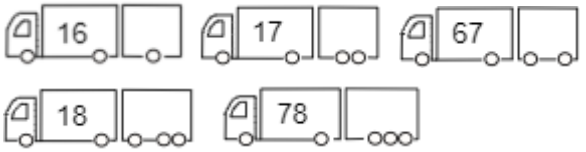
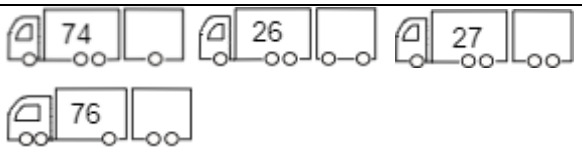
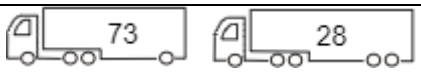
## 5.2 Klasifikačný štandard

Na začiatok je nutnú definovať štandard (Tab. 2), z ktorého sa bude pri klasifikácii vychádzať. Ako základ pre triedy bol použitý európsky štandard EUR13 (Obr. 1), ktorý definuje dvanásť základných tried a jednu triedu pre neklasifikované vozidlá. Pre účely tejto práce bol tento štandard mierne upravený. Úpravy spočívajú v tom, že v prípade prvej triedy došlo k separácii osobných vozidiel od ľahkých úžitkových vozidiel, t. j. dodávok do samostatných tried. Ďalej bola pridaná trieda pre motorky a iné vozidlá. Ako posledné boli zlúčené triedy ťahačov podľa počtu náprav ťahača a odstránená jedenásta trieda, ktorá sa týka 6-nápravových vozidiel. Klasifikácia sa zaoberá len vozidlami, resp. jazdnými súpravami o maximálnom počte päť náprav, avšak štandard môže byť kedykoľvek rozšírený aj o viac nápravové vozidlá.

Cieľom vyvíjaného nástroja nie je rozlišovať vozidlá podľa tried, ale predovšetkým ide o zameranie sa na podtriedy týchto tried (Tab. 2). Trieda trinásť (iné) je rezervovaná pre podtriedy, ktoré nie sú uvedené ako súčasť tohto štandardu. Pri definovaní podtried sa vychádzalo z testovacích dát a celkovo bolo nájdených 38 podtried. Vozidlá sú roztriedené do podtried na základe konfigurácie náprav a funkčných vlastnosti. Keďže sa vychádza zo štandardu EUR13, dá sa povedať, že vozidlá v spoločnej triede by si aspoň z časti mohli byť podobné aj hmotnosťou.

Vo výsledku je podstatou klasifikácie priradiť vozidlu správnu podtriedu tak, aby odpovedala podtriede, ktorú má vozidlo priradenú v súbore s testovacími dátami.

Tabuľka 2. Klasifikačný štandard

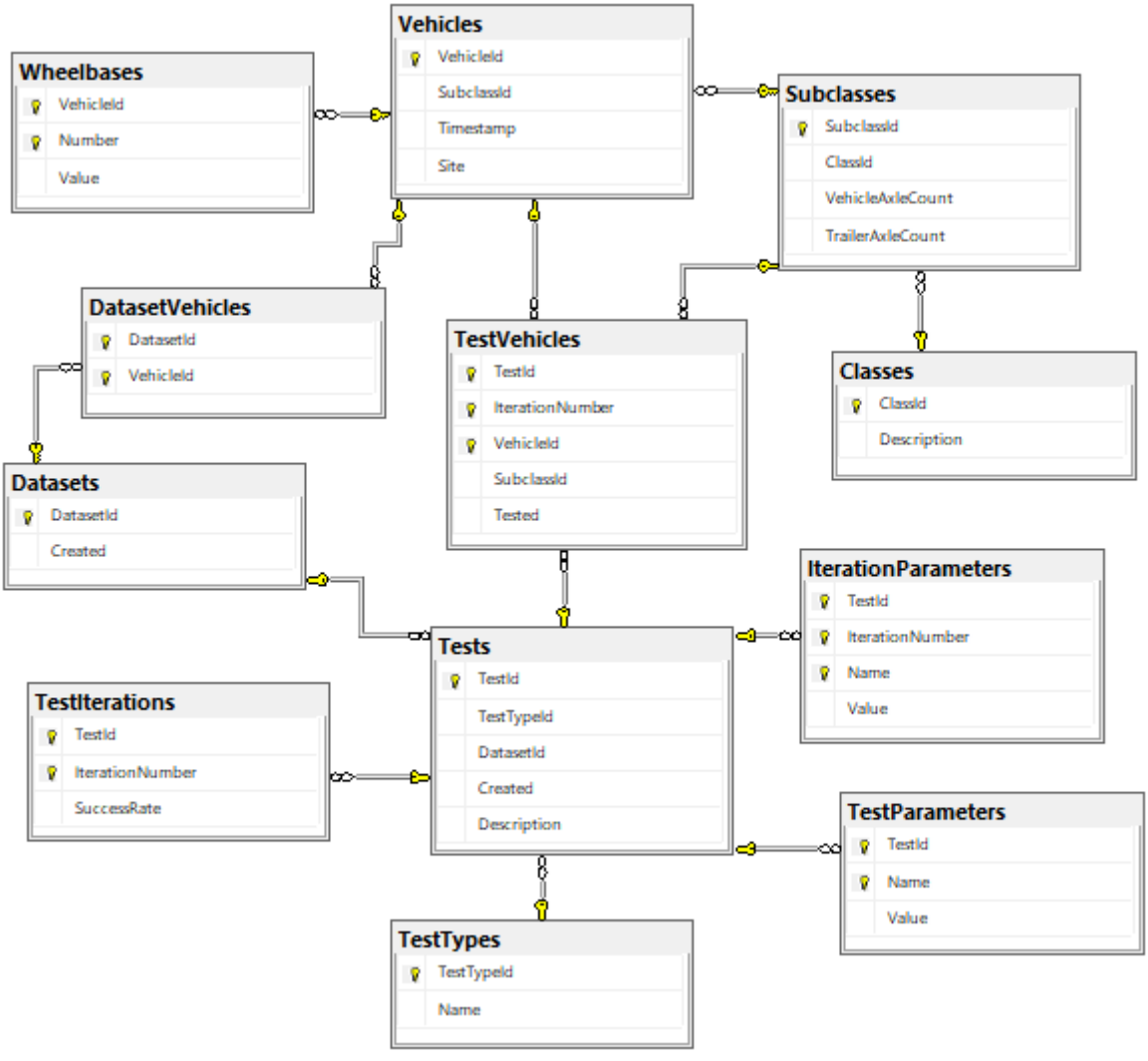
| **Trieda** | **Popis** | **Podtriedy** |
|---|---|---|
| 1 | neklasifikované | 1 neklasifikované |
| 2 | motorky |  |
| 3 | osobné vozidlá |  |
| 4 | dodávky |  |
| 5 | autobusy |  |
| 6 | 2-nápravové nákladné vozidlá |  |
| 7 | 3-nápravové nákladné vozidlá |  |
| 8 | 4-nápravové nákladné vozidlá |  |
| 9 | 2-nápravové nákladné vozidlá + príves |  |
| 10 | 3-nápravové nákladné vozidlá + príves |  |
| 11 | 2-nápravové ťahače + 1/2/3-nápravový náves |  |
| 12 | 3-nápravové ťahače + 1/2-nápravový náves |  |
| 13 | iné | |

# 6 NÁVRH RIEŠENIA

Návrh aplikácie klasifikačného nástroja je vypracovaný so zámerom vytvoriť aplikáciu, ktorá umožní pridávanie vozidiel v podobe datasetu a ich následne testovanie klasifikačným testom s možnosťou využitia doplnkových testov tzv. hraničné. Tie majú dopomôcť k získaniu lepších výsledkov z klasifikácie. V poslednej rade pôjde o zobrazenie výsledkov.

## 6.1 Databáza

Databázová štruktúra je navrhnutá tak, aby neslúžila len k ukladaniu testovacích dát, ale aby zároveň uchovávala aj dáta potrebné k parametrizácii a jej výsledky. Entitne-relačný (ER) diagram graficky znázorňuje konceptuálny ER model databáze. Zobrazuje entity a vzťahy medzi nimi a atribúty entít. Z ER modelu je možné vytvoriť relačnú schému databáze, ktoré už obsahuje tabuľky, ich štruktúry a vzťahy medzi nimi (Obr. 11).




Obrázok 11. Relačná schéma databáze

### 6.1.1 Tabuľky

Tabuľka **Classes** obsahuje primárny kľúč ClassId a jej popis.

Tabuľka **Subclasses** obsahuje primárny kľúč SubclassId a atribúty, ktoré nesú údaje o počte náprav vozidla a počte náprav prívesu danej podtriedy. Referencia na údaj o triede je zaistené cudzím kľúčom ClassId.

Obe tieto tabuľky majú len informačný charakter a budú naplnené dátami už pri vytváraní databáze. Prehľad týchto tried a podtried zobrazuje tabuľka (Tab. 2).

Tabuľky **Vehicles**, **WheelBases**, **Datasets** a **DatasetVehicles** slúžia k ukladaniu testovacích dát z .csv súborov.

Tabuľka **Dataset** obsahuje primárny kľúč DatasetId a atribút, ktorý nesie informáciu o čase vytvorenia. Každý dataset môže pozostávať z niekoľkých .csv súborov.

Tabuľka **Vehicles** obsahuje primárny kľúč VehicleId. Tento kľúč sa nebude generovať automaticky, ale je prevzatý z identifikačného čísla vozidla z testovacích dát (*vehicleid*). Ďalej obsahuje atribúty, ktorú nesú informácie o skutočnej podtriede vozidla, čase prejazdu vozidla a pôvodnom umiestnení vozidla. Rovnako sú aj tieto údaje prevzaté z testovacích dát (*subclassid, timestamp, ucid*). Atribút definujúci skutočnú triedu vozidla SubclassId je v tejto tabuľke cudzím kľúčom.

Tabuľka **Wheelbases** obsahuje zložený primárny kľúč (VehicleId, Number), kde VehicleId je súčasne aj cudzím kľúčom. Táto tabuľka ukladá vzdialenosti medzi jednotlivými nápravami vozidla. Atribút Number je číslo, ktoré definuje, o ktorú vzdialenosť medzi dvoma nápravami sa jedná. Napr. 0 je vzdialenosť medzi 1. a 2. nápravou vozidla a atribút Value je jeho hodnota prevzatá z testovacích dát (*wb0*).

Tabuľka **DatasetVehicles** je pomocnou väzbovou tabuľkou k realizáciu vzťahu M:N medzi tabuľkami Datasets a Vehicles. Obsahuje zložený primárny kľúč (DatasetId, VehicelId), ktoré sú zároveň aj cudzími kľúčmi. Táto tabuľka ukladá k datasetom im prislúchajúce vozidlá. Predpokladá sa, že jedno vozidlo môže byť súčasťou viacerých datasetov.

Tabuľky **TestTypes**, **Tests**, **TestParameters**, **TestIterations**, **IterationParameters** a **TestVehicles** slúžia k ukladaniu dát, ktoré sa týkajú testov.

Tabuľka **TestType** obsahuje primárny kľúč TestTypeId a názov, ktorý charakterizuje typ testu (hraničný test a klasifikačný test).

Tabuľka **Tests** obsahuje primárny kľúč TestId. Referencie na údaje o datasete a type testu sú zabezpečené cudzími kľúčmi DatasetId a TestTypeId. Nový test sa vytvára vždy nad datasetom, ktorý bol do databázy pridaný ako posledný. Ďalej táto tabuľka obsahuje údaje o čase vytvorenie testu a popis. Popis uvádza, na ktorých vozidlách bude test vykonávaný. Pri klasifikačnom teste pôjde vždy o všetky vozidlá, kdežto pri hraničnom teste sa testujú vždy len dve podtriedy.

Tabuľka **TestParameters** obsahuje zložený primárny kľuč (TestId, Name), kde TestId je zároveň cudzím kľúčom. Táto tabuľka uchováva parametre testu a atribút Name ich špecifikuje (script, overlapType, subclassA, subclassB, wheelbaseNumber). Klasifikačný test zaznamenáva názov skriptu a hraničný test typ prekrytia, poradie podtried a číslo rázvora, na ktorom sa testuje.

Tabuľka **TestIterations** obsahuje zložený primárny kľúč (TestId, IterationNumber), kde TestId je súčasne aj cudzí kľúčom. Pre každý test je počet iterácii ovplyvnený typom testu. Klasifikačný test sa vykonáva vždy len s jednou iteráciou (nultou), zatiaľ čo hraničný test ich môže mať niekoľko. Ďalej táto tabuľka ukladá ku každej iterácií dosiahnutú percentuálnu úspešnosť z testovania.

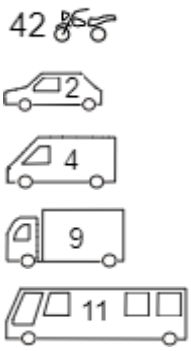
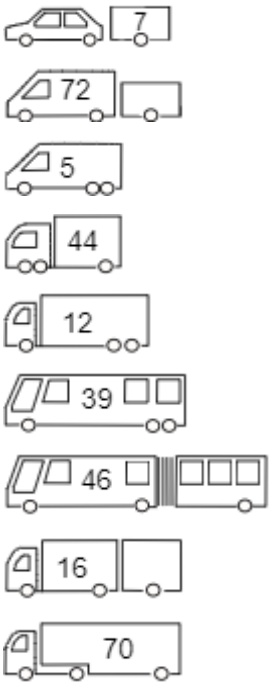
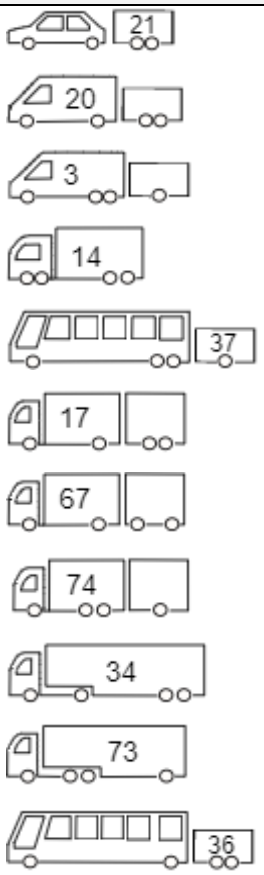
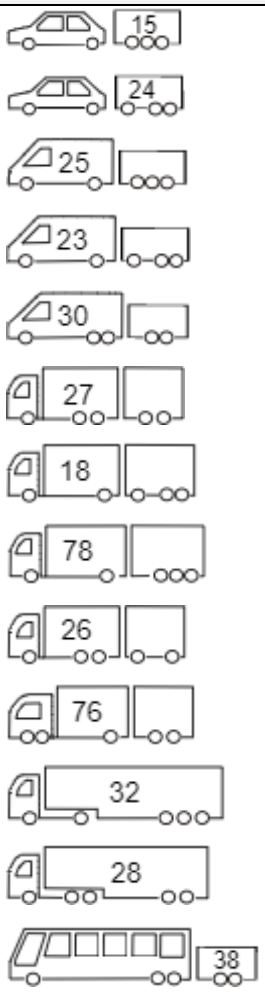
Tabuľka **IterationParameters** obsahuje zložený primárny kľúč (TestId, IterationNumber, Name), kde TestId plní funkciu cudzieho kľúča. Táto tabuľka ukladá všetky parametre (rázvory), ktoré budú v danom teste a v danej iterácii pri testovaní použité. Atribút Name nesie názov parametra a atribút Value určuje jeho hodnotu.

Tabuľka **TestVehicles** obsahuje zložený primárny kľúč (TestId, IterationNumber, VehicleId). Atribúty TestId, VehicleId a SubclassId sú v tejto tabuľke cudzími kľúčmi. Táto tabuľka ukladá vozidlá, na ktorých sa bude v danom teste a v danej iterácii testovať. Atribút Tested je typu boolean a zaznamenáva stav vozidla (testované/netestované). Podtriedu priradenú klasifikáciou ukladá atribút SubclassId.

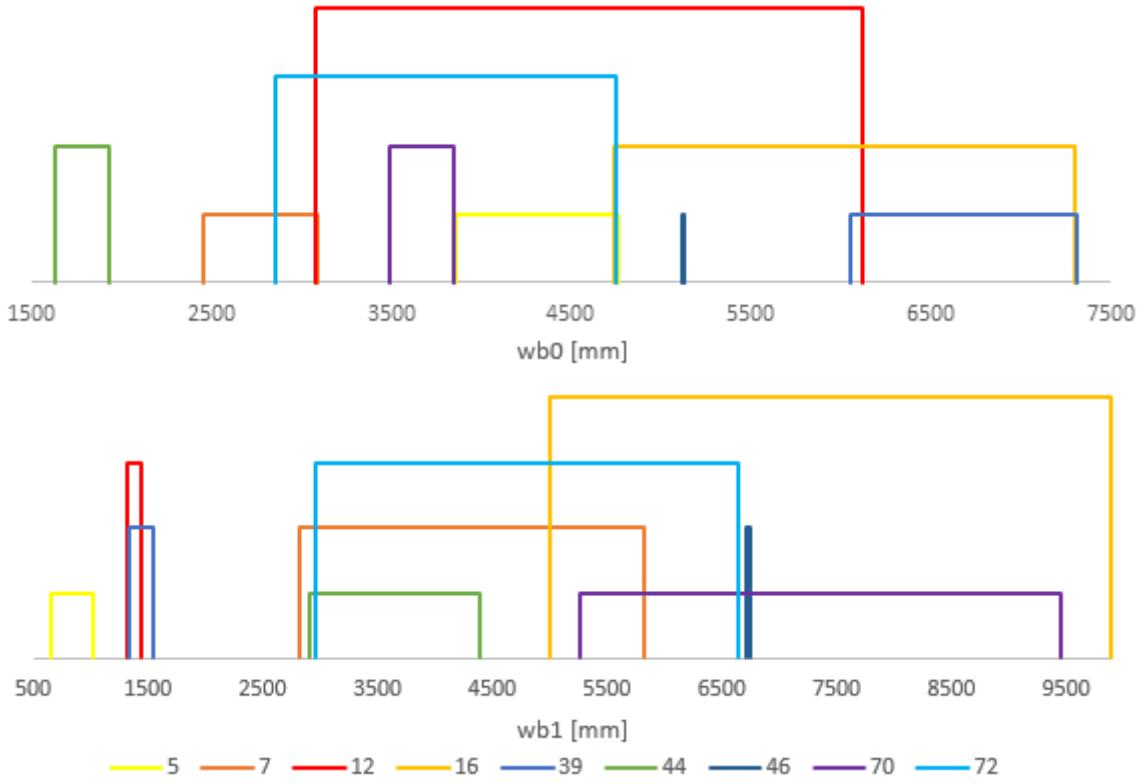
## 6.2 Parametrizácia

Pri klasifikácii vychádzajúcej len zo vzdialenosti medzi susednými nápravami vozidla je najvhodnejšie pracovať s podtriedami rozdelenými podľa počtu náprav.

Tabuľka 3. Podtriedy podľa počtu náprav

| 2-nápravové | 3-nápravové | 4-nápravové | 5-nápravové |
|---|---|---|---|
|  |  |  |  |

Vzhľadom k veľkému počtu týchto podtried a podobnosti v ich konfigurácii podvozku nie je identifikácia podtriedy vozidla vždy jednoznačná. Je to je spôsobené tým, že rázvory jednej podtriedy sa buď z časti alebo v niektorých prípadoch aj úplne prekrývajú s rázvormi inej podtriedy. Je dôležité, aby pred samotnou klasifikáciou najprv došlo k zanalyzovaniu stavu medzi jednotlivých podtriedami rozdelenými podľa tabuľky (Tab. 3) a následnému nájdeniu hraníc.

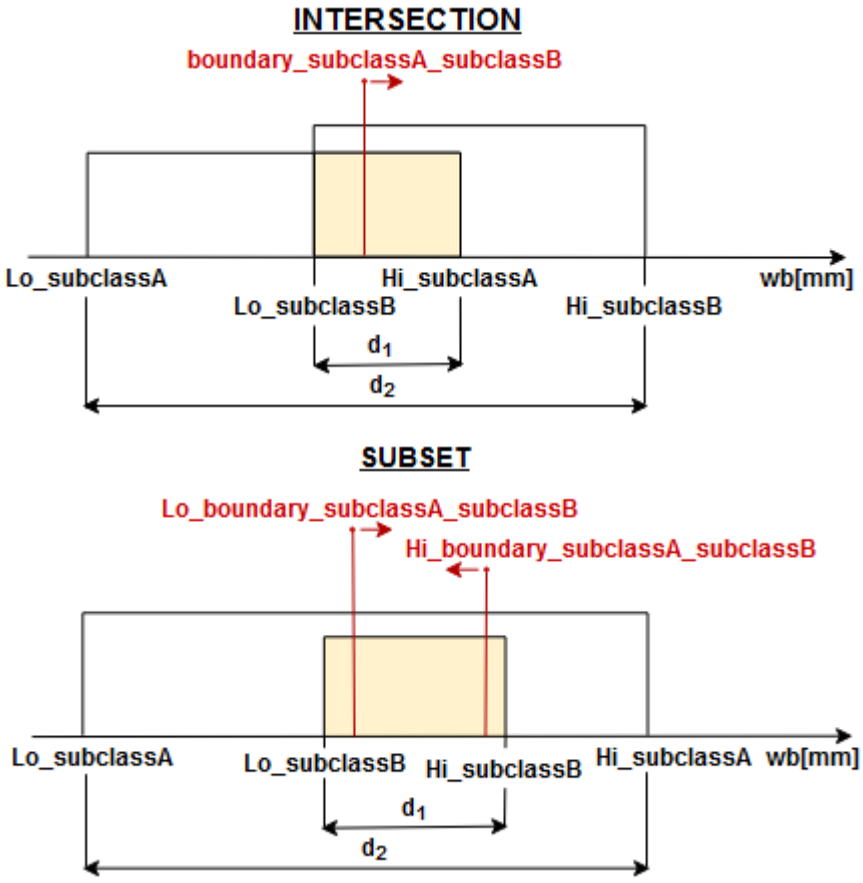



Obrázok 12. Rázvory 3-nápravových podtried – intervaly

Obrázok (Obr. 12) znázorňuje rozpätia rázvorov (wb0, wb1) pre podtriedy 3-nápravových vozidiel. Pre každú podtriedu je jej každý rázvor daný vždy dvojicou hodnôt (najnižšou a najvyššou), resp. rázvor je daný intervalom $wb \in [Lo, Hi]$. Každý interval je stanovený len z vozidiel, ktoré sa nachádzajú v práve testovanom datasete.

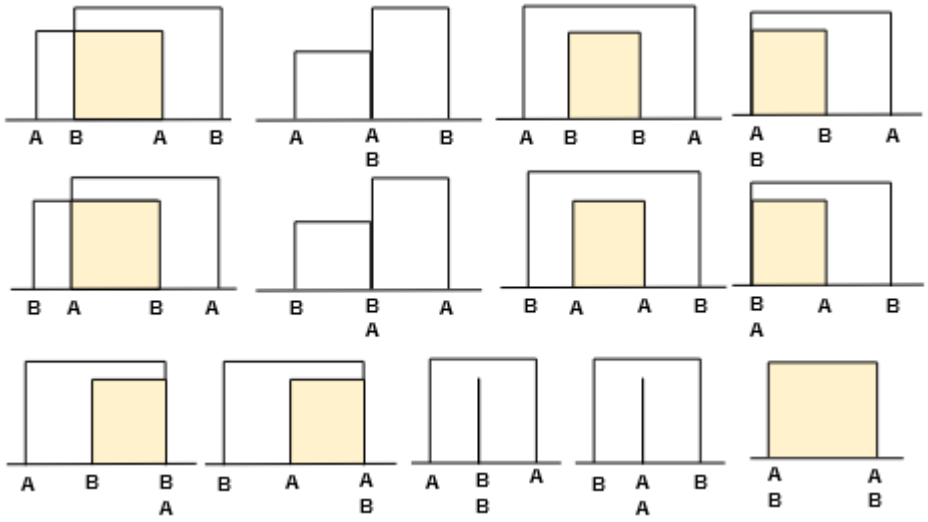
### 6.2.1 Hranice – spôsoby prekrytia

Na začiatok je potrebné nájsť podtriedy, ktoré si vzájomne odporujú. Z bližšieho preskúmania vozidiel jednotlivých podtried podľa tabuľky (Tab. 3) bolo zistené, že stret záujmov medzi podtriedami, na ktoré je potreba sa zamerať nastáva vždy len medzi dvoma podtriedami. Z toho dôvodu sa za dve podtriedy nejednoznačne identifikovateľné považujú tie podtriedy, u ktorých sa ich všetky rázvory vzájomne prekrývajú akýmkoľvek spôsobom. Z obrázka (Obr. 12) je možné vidieť, že k takému prekrytiu dochádza u 3-nápravových vozidiel medzi podtriedami 7-72, 12-39, 16-46, 16-72 a 70-72 a nastávajú medzi nimi dve základné spôsoby prekrytia (Obr. 13). Takéto dvojice budú ďalej označované ako hraničné typy a poradie ich čísel sa uvádza vzostupne.



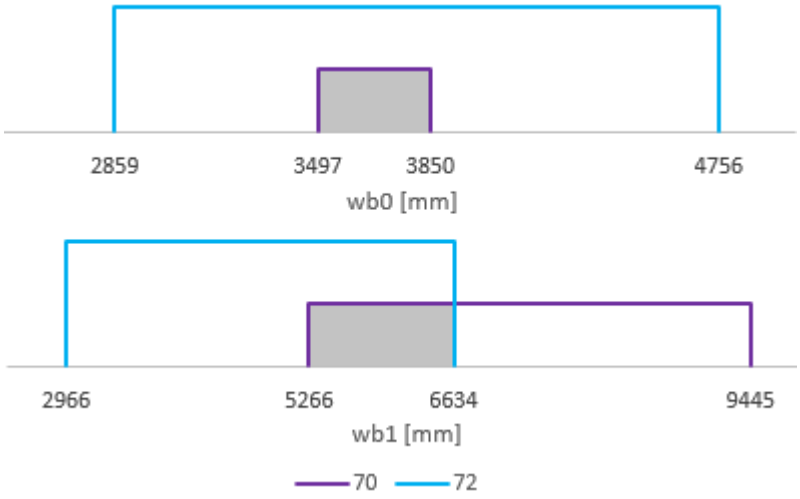

Obrázok 13. Spôsoby prekrytia

Horná časť obrázka (Obr. 13) zobrazuje najjednoduchší a najčastejšie sa vyskytujúci spôsob prekrytia. Medzi dvoma podtriedami nastáva prienik (intersection) a k presnému definovaniu podtried postačuje nájdenie jednej hranice. V spodnej časti je jedna podtrieda podmnožinou druhej podtriedy (subset) a takáto situácia si vyžaduje nájdenie dvoch hraníc. Pre účely tejto práce mal byť dôraz kladený predovšetkým na prieniky, ale pre úplnosť obsahuje riešenie aj pre vyhodnocovanie podmnožín. Okrem typu prekrytia je dôležité poznať aj poradie podtried, a preto aplikácia rozoznáva všetky situácie zobrazené na nasledujúcom obrázku (Obr. 14).




Obrázok 14. Spôsoby prekrytia – možné situácie

Obrázok (Obr. 15) zobrazuje prekrytia rázvorov medzi podtriedami 70 a 72 z obrázka (Obr. 12). Tieto podtriedy zahŕňajú oba spôsoby prekrytia a k tomu, aby boli jednoznačne identifikovateľné je postačujúce nájsť hranicu na jednom rázvore (wb0 alebo wb1).




Obrázok 15. Prekrytie rázvorov 70-72

Počet testovaných hraníc, resp. počet iterácii testu pre konkrétny hraničný typ vychádza z dĺžky prekrytia intervalu $d_1$ a typu prekrytia (Obr. 13). Rozostupy týchto hraníc alebo to ako podrobne sú tieto oblasti postupne testované závisí od dĺžky podintervalov. Dĺžka podintervalu $d_3$ je stanovená ich počtom a minimálny počet podintervalov je jeden (definuje pôvodnú oblasť prekrytia). Nasledujúce rovnice využívajú celočíselne delenie.

Ak $d_1 < 1000$ mm, potom pre počet podintervalov $n$, kde $n \in \mathbb{N}$ platí

$$n = \frac{d_1}{50} + 1, \tag{2}$$

ak $d_1 \geq 1000$ mm, potom

$$n = \frac{d_1}{100} \tag{3}$$

a ich dĺžka $d_3$, kde $d_3 \in \mathbb{N}$ je daná

$$d_3 = \frac{d_1}{n}. \tag{4}$$

#### *6.2.1.1 Prekrytie typu prienik*

Tento typ prekrytia presne definuje počet iterácií testu $i$ a platí

$$i = n + 1, \tag{5}$$

kde $i \in \mathbb{N}$ a $n$ je počet podintervalov podľa vzťahu (2), resp. (3).

Nech dĺžka prekrytia intervalu $d_1$ je daná intervalom $d_1 \in [Lo, Hi]$ a pre každú iteráciu existuje práve jedna testovaná hranica, potom sú tieto hranice definované postupnosťou čísel $\{a_m\} = \{a_m\}_{m=0}^{i-1} = \{a_0, a_1, \dots, a_{i-1}\}$ a pre každé $m \in [0, i-1]$ platí

$$a_m = \begin{cases} Lo & \text{ak } m = 0 \\ Hi & \text{ak } m = i - 1 \\ a_{m-1} + d_3 & iné, \end{cases} \tag{6}$$

kde $d_3$ je dĺžka podintervalu daná vzťahom (4).

#### *6.2.1.2 Prekrytie typu podmnožina*

Nech dĺžka prekrytia intervalu $d_1 \in [Lo, Hi]$, potom je počet iterácii testu daný počtom prvkov množiny definovanej nasledovne

$$\left\{[Lo, Hi], \{[Lo, a_m]\}_{m=1}^{n-1}, \{[a_m, Hi]\}_{m=1}^{n-1}, \left\{\left[a_j, a_k\right]\right\}_{j,k=1}^{n-1}\right\}, \tag{7}$$

kde $m, j, k \in \mathbb{N}$ a $n$ je počet podintervalov podľa vzťahu (2), resp. (3).

Každá iterácia testu pozostáva z jedného prvku množiny, ktorý je definovaný dvojicou testovaných hraníc (interval). Pri prekrytí typu prienik sa testuje pôvodný interval prekrytia a pôvodný interval postupne zmenšovaný sprava, zľava a z oboch strán. Pre členy postupnosti s jednou pevnou hranicou $\{[Lo, a_m]\}_{m=1}^{n-1}$ a $\{[a_m, Hi]\}_{m=1}^{n-1}$ platia nasledujúce vzorce

$$[Lo, a_m] = [Lo, Hi - m \cdot d_3] \tag{8}$$

$$[a_m, Hi] = [Lo + m \cdot d_3, Hi] \tag{9}$$

a pre členy postupnosti s premenlivými hranicami $\{[a_j, a_k]\}_{j,k=1}^{n-1}$ platí

$$[a_j, a_k] = [Lo + j \cdot d_3, Hi - k \cdot d_3] \qquad \text{ak } a_k > a_j, \tag{10}$$

kde dĺžka podintervalu $d_3$ je daná vzťahom (4).

### 6.2.2 Hraničné typy

Nájdená hranica môže mať zásadný vplyv na správnosť zatriedenia vozidla, a preto je dôležité zvoliť správny rázvor. Pri výbere vhodného rázvora môže rolu zohrávať niekoľko faktorov. Tento návrh klasifikačného nástroja kladie dôraz na percentuálne vyjadrenie dĺžky prekrytia, ktoré je dané vzťahom

$$overlap\ [\%] = \frac{d_1}{d_2} \cdot 100, \tag{11}$$

kde $d_1$ je dĺžka oblasti prekrytia a $d_2$ je celková vzájomná dĺžka oboch podtried (Obr. 13). Zároveň bol ako štandardný rázvor uprednostnený rázvor s typom prekrytia prienik a voľba rázvora bola ovplyvnené aj počtom vozidiel, ktoré sa nachádzajú v oblasti prekrytia. Výber rázvora nemusí byť vždy jednoznačný, a preto klasifikačný nástroj umožňuje užívateľovi definovať aj iný rázvor. Nasledujúca tabuľka zobrazuje prehľad hraničných typov a ich štandardný (predvolený) rázvor a ďalšie rázvory, na ktorých sa môže testovať taktiež. Súčasťou každého rázvora je poznámka o type prekrytia, ktorý v sebe už klasifikácia zahŕňa (prienik = I, podmnožina = S). V prípade voľby rázvora, resp. hraničného typu mimo uvedených hranica nájdená bude, ale súčasťou klasifikácie už nie je (nutné doplniť).

Tabuľka 4. Hraničné typy a ich rázvory

| Hraničný typ | Štandardný rázvor | Ďalšie rázvory |
|---|---|---|
| 2-4, 7-72, 20-21, 15-25, 23-24 | wb0 (I) | - |
| 4-9 | wb0 (I) | - |
| 9-11 | wb0 (I/S) | - |
| 12-39 | wb0 (I) | - |
| 16-46 | wb0 (S) | wb1 (S) |
| 16-72 | wb0 (I) | - |
| 70-72 | wb1 (I) | wb0 (S) |
| 17-20 | wb1 (I) | - |
| 17-36 | wb2 (I) | wb0 (I) |
| 21-34 | wb2 (I) | - |
| 27-28 | wb0 (I) | - |
| 27-38 | wb3 (I) | wb0 (I) |
| 32-78 | wb0 (I) | wb1 (I) |

### 6.2.3 Klasifikácia

Klasifikácia je založená na princípe vytvorenia rozhodovacej štruktúry. Ide o deterministický algoritmus, ktorého cieľom je identifikovať objekty (vozidlá) do podtried podľa tabuľky (Tab. 2). K rozhodovaniu dochádza sekvenciou príkazov, kde sa postupne vyhodnocuje splnenie, resp. nesplnenie podmienok. Pre tento algoritmus musí platiť, že po skončení vyhodnocovania je vždy vrátená akákoľvek podtrieda. Ak nevyhovuje žiadna vetva z bloku príkazov je štandardne vrátená podtrieda jeden (neklasifikované vozidlo).

Klasifikačné testy sú tvorené len jednou iteráciou a môžu využívať parametre z hraničných testov.

## 6.3 Stanovenie úspešnosti

Celková percentuálna úspešnosť testu, resp. jeho iterácií je všeobecne daná vzťahom

$$success\ [\%] = \frac{n_{true}}{n} \cdot 100, \tag{12}$$

kde $n_{true}$ je počet správne klasifikovaných vozidiel a $n$ je počet všetkých klasifikovaných vozidiel. Pre jednotlivé podtriedy, potom platí

$$success_i\ [\%] = \frac{n_{true,i}}{n_{true,i} + n_{false,i}} \cdot 100 \tag{13}$$

a pre triedy, resp. skupinu podtried platí

$$success\ [\%] = \frac{\sum_{i=1}^{N} n_{true,i}}{\sum_{i=1}^{N} (n_{true,i} + n_{false,i})} \cdot 100, \tag{14}$$

kde $n_{true,i}$ je počet správne klasifikovaných vozidiel triedy $i$, $n_{false,i}$ je počet nesprávne klasifikovaných vozidiel triedy $i$ a $N$ je počet vyhodnocovaných podtried.

# 7 KLASIFIKAČNÝ NÁSTROJ

Klasifikačný nástroj je realizovaný ako konzolová aplikácia .NET Core s využitím programovacieho jazyka C#. Samotná klasifikácia využíva skript vložiteľný do hostiteľského programu a ako skriptovací jazyk bol použitý jazyk Lua. K ukladaniu dát je vytvorená relačná databáza a systémom pre riadenie báze dát je MS SQL Server. Objektovo-relačné mapovanie zabezpečuje Entity Framework Core, ktorý umožňuje ľahko pracovať s databázou pomocou .NET objektov. Dotazovanie prebieha využitím dotazovacieho jazyka LINQ (syntaxou lambda) a k vývoju klasifikačného nástroja bolo použité vývojové prostredie MS Visual Studio.

## 7.1 Databáza

Na vytvorenie databáze bol zvolený prístup code-first. K zostaveniu modelu je potrebné najprv nadefinovať triedy entít a triedu kontextu. Každá trieda entity definuje jednu tabuľku v relačnej databáze. Nasledujúci obrázok zobrazuje triedu entity Vehicle, ktorú EF Core mapuje do databázovej tabuľky Vehicles.

```
public class Vehicle
{
    // Primárny kľúč.
    public int VehicleId { get; set; }
    // Cudzí kľúč.
    public short SubclassId { get; set; }
    public DateTime Timestamp { get; set; }
    public short Site { get; set; }

    // Referenčná navigačná vlastnosť.
    public Subclass Subclass { get; set; }
    // Navigačná vlastnosť typu kolekcia.
    public ICollection<Wheelbase> Wheelbases { get; set; }
    public ICollection<TestVehicle> TestVehicles { get; set; }
    public ICollection<DatasetVehicle> DatasetVehicles { get; set; }
}
```

Obrázok 16. Trieda entity Vehicle

Pri konfigurácii týchto tried boli dodržané štandardy stanovené základným prístupom By-Convention. Predovšetkým ide o pravidlá týkajúce sa tried entít (modifikátor prístupu public), používanie primitívnych dátových typov, zachovanie konvencie pre názvy vlastnosti napr. k identifikácii primárneho kľúča (VehicleId) a ďalšie. Tento prístup zahŕňa len základnú konfiguráciu a k doplňujúcemu dodefinovaniu tried použitý prístup Fluent API.

Okrem vlastnosti, ktoré reprezentujú stĺpce v tabuľke sú súčasťou tried entít aj navigačné vlastnosti. Navigačná vlastnosť typu kolekcia obsahuje odkaz na niekoľko súvisiacich entít, kdežto referenčná navigačná vlastnosť obsahuje odkaz na jednu súvisiacu entitu, a tým udáva, že sa jedná o cudzí kľúč v tabuľke. Tieto navigačné vlastnosti definujú vzťahy medzi triedami entít. Konkrétne pre obrázok (Obr. 16) je trieda Vehicle vo vzťahu 1:N k triedam Wheelbase, TestVehicle a DatasetVehicle. Pri určovaní vzťahov bolo použité plné definovanie vzťahov, kedy navigačné vlastnosti sú na oboch koncoch vzťahu a súčasťou je aj definícia cudzieho kľúča (SubclassId). Tento spôsob definuje kaskádne mazanie dát a zároveň definuje cudzí kľúč ako not null, čo ale môže byť upravené operátorom ?.

### 7.1.1 Trieda kontextu

Po vytvorení tried entít je potrebné definovať triedu kontextu. Táto trieda dedí z triedy DbContext a jej inštancia predstavuje reláciu s databázou.

```
public class VehicleClassificationDbContext : DbContext
{
    private const string _connectionString = @"Server=localhost;
                                               Database=VehicleClassificationDb;
                                               Trusted_Connection=True;";
    protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(_connectionString);
    }
    public DbSet<Class> Classes { get; set; }
    public DbSet<Dataset> Datasets { get; set; }
    public DbSet<DatasetVehicle> DatasetVehicles { get; set; }
```

Obrázok 17. Kontextová trieda aplikácie

Inštancia DbContextOptionBuilder metódy OnConfiguring() svojím rozšírením špecifikuje poskytovateľa databáze (UseSqlServer) a prostredníctvom reťazca pripojenia mu predáva potrebné informácie k naviazaniu spojenia s databázou. Reťazec pripojenia definuje databázový server, názov databáze a spôsob autorizácie. Rozširujúce metódy sú rovnako ako EF Core dostupné v podobe balíkov NuGet. Pre prístup k MS SQL serveru ide o NuGet Microsoft.EntityFrameworkCore.SqlServer.

V kontextovej triede sú pre každú entitu vytvorené vlastnosti typu DbSet<TEntity>. Tie hovoria EF Core, že existuje v databáze tabuľka Classes, ktorá má atribúty, ktoré sú k dispozícii v triede Class, t. j. mapuje triedu Class k tabuľke Classes. Tieto sety entít umožňujú vykonávať základné operácie CRUD.

Súčasťou kontextovej triedy je metóda OnModelCreating(), ktorá poskytuje informácie o dodatočnej konfigurácii modelu prístupom Fluent API skrz metódy definované v triede ModelBuilder. Patrí sem napr. zakázanie generovania primárneho kľúča s automatickou inkrementáciou (identity(1,1)), nastavenie maximálnej dĺžky reťazca a definovanie zloženého primárneho kľúča.

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    // Presunutie konfigurácií Fluent API.
    modelBuilder.ApplyConfiguration(new Configurations.ClassConfig());
    modelBuilder.ApplyConfiguration(new Configurations.TestTypeConfig());
    modelBuilder.ApplyConfiguration(new Configurations.SubclassConfig());

    modelBuilder.Entity<Vehicle>(entity =>
    {
        // Zakázanie automatického generovania primárneho kľúča.
        entity.Property(t => t.VehicleId).ValueGeneratedNever();
    });

    modelBuilder.Entity<IterationParameter>(entity =>
    {
        // Zložený primárny kľúč.
        entity.HasKey(ip => new { ip.TestId, ip.IterationNumber, ip.Name });
        // Nastavenie maximálnej dĺžky reťazca.
        entity.Property(ip => ip.Name).HasMaxLength(30);
    });
```

Obrázok 18. Metóda OnModelCreating() v triede kontextu

Okrem vyššie uvedených metód bola k naplneniu tabuliek počiatočnými dátami využitá aj metóda HasData(). Z dôvodu rozsiahlosti bol konfiguračný kód Fluent API pre triedy entít (Class, Subclass, TestType) presunutý do vlastných konfiguračných tried, ktoré implementujú rozhranie IEntityTypeConfiguration<TEntity>.

```
internal class ClassConfig : IEntityTypeConfiguration<Class>
{
    public void Configure(EntityTypeBuilder<Class> entity)
    {
        entity.HasData(
            new Class { ClassId = 1, Description = "unclassified" },
            new Class { ClassId = 2, Description = "motorcycle" },
            new Class { ClassId = 3, Description = "passenger car" },
            new Class { ClassId = 4, Description = "delivery vehicle" },
```

Obrázok 19. Konfiguračná trieda entity Class

### 7.1.2 Migrácie

Po navrhnutí tried entít a kontextovej triedy je databáza vytvorená použitím migrácií. Migrácie sú súčasťou NuGet balíka Microsoft.EntityFrameworkCore.Tools. Príkaz add-migration zostaví z kontextovej triedy a tried entít model databáze. Následne sa na základe modelu vygeneruje súbor so sadou príkazov, ktoré tento model popisujú. Príkazom update-database sa z vygenerovaných príkazov vytvorí databáza. Štruktúru databáze môže byť aj naďalej upravovaná a k aplikovaniu zmien je potrebné príkazy zopakovať. Obrázok (Obr. 20) zobrazuje časť vygenerovaného kódu pre entitu Wheelbase.

```
migrationBuilder.CreateTable(
    name: "Wheelbases",
    columns: table => new
    {
        VehicleId = table.Column<int>(type: "int", nullable: false),
        Number = table.Column<short>(type: "smallint", nullable: false),
        Value = table.Column<int>(type: "int", nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Wheelbases", x => new { x.VehicleId, x.Number });
        table.ForeignKey(
            name: "FK_Wheelbases_Vehicles_VehicleId",
            column: x => x.VehicleId,
            principalTable: "Vehicles",
            principalColumn: "VehicleId",
            onDelete: ReferentialAction.Cascade);
    });
```

Obrázok 20. Migrácia – vytvorenie tabuľky Wheelbases

## 7.2 Dataset

Dataset obsahuje zoznam vozidiel, na ktorých sa vykonávajú testy za účelom ich klasifikácie. Potrebné dáta k týmto vozidlám sú uložené v .csv súboroch a boli podrobne rozobrané v časti 5.1 Testovacie dáta. Každý dataset môže pozostávať z niekoľkých súborov a pre jednoduchú manipuláciu sú tieto súbory umiestnené v aktuálnom pracovnom adresári aplikácie (priečinok dataset).

### 7.2.1 Pridávanie datasetu

Možnosť pridania datasetu je podmienená existenciou aspoň jedného súboru s testovacími dátami. Po splnení tejto podmienky má nový dataset automaticky vygenerované identifikačné číslo a pridelený časový údaj o jeho vytvorení. Následne sú dáta zo súborov postupne načítavané a ukladané. Súčasťou datasetu sú len tie vozidlá, ktoré majú v súbore jednoznačne priradenú podtriedu (*subclassid*). Vozidlá s podtriedou (1) nebolo možné podľa záznamov z prejazdu rozpoznať, resp. obsahujú neplatné dáta a sú vynechané.

```
try
{
    using var context = new Models.VehicleClassificationDbContext();

    // Naplnenie tabuľky DatasetVehicles.
    context.DatasetVehicles.AddRange(DatasetVehicles(context));
    // Naplnenie tabuľky Vehicles.
    context.Vehicles.AddRange(Vehicles(context));
    // Naplnenie tabuľky WheelBases.
    context.Wheelbases.AddRange(WheelBases());

    context.SaveChanges();
```

Obrázok 21. Ukladanie dát zo súboru do databáze

Ukladanie dát zo súboru prebieha v niekoľkých krokoch. Ako prvé je do tabuľky DatasetVehicles k novému datasetu ukladaný zoznam vozidiel v podobe identifikačných čísel (kontroluje sa jedinečnosť). Obrázok (Obr. 22) zobrazuje novovzniknutú tabuľku vytvorenú spojením záznamov z tabuliek DatasetVehicles a Vehicles spoločným kľúčom VehicleId.

| DatasetId | VehicleId | SubclassId | Timestamp | Site |
|---|---|---|---|---|
| 1 | 3588727 | 46 | 2019-08-20 15:23:52... | 36 |
| 1 | 3634386 | 16 | 2019-08-22 12:20:18... | 36 |
| 1 | 3777124 | 16 | 2019-08-28 09:12:59... | 36 |
| 1 | 4265381 | 21 | 2019-09-16 10:11:18... | 35 |
| 1 | 4292769 | 21 | 2019-09-17 10:45:07... | 35 |
| 1 | 4320589 | 46 | 2019-09-18 10:36:58... | 36 |
| 1 | 4381963 | 21 | 2019-09-20 14:55:31... | 35 |

Obrázok 22. Spojené tabuľky DatasetVehicles a Vehicles

Vozidlá môžu byť súčasťou viacerých datasetov, a preto tabuľky Vehicles a WheelBases ukladajú len dáta k tým vozidlám, ktoré sa ešte v databáze nenachádzajú. V prípade už existujúceho vozidla sa kontroluje podtrieda. Ak je podtrieda uložená v databáze odlišná od podtriedy uloženej v súbore, potom bude záznam modifikovaný. Je potrebné brať v úvahu, že takáto zmena môže mať vplyv pri zobrazení prehľadu z histórie klasifikačných testov uvedených na obrázkoch (Obr. 44 a Obr. 46) v časti 7.3.5.1 Prehľad. Vyhodnotenie úspešnosti testov podľa vzťahu (12) nebude ovplyvnené.

| VehicleId | Number | Value |
|---|---|---|
| 7048004 | 0 | 2795 |
| 7048004 | 1 | 5158 |
| 7048448 | 0 | 5126 |
| 7048448 | 1 | 6707 |
| 7049072 | 0 | 1398 |
| 7049450 | 0 | 4098 |
| 7049450 | 1 | 7303 |
| 7049450 | 2 | 755 |
| 7049450 | 3 | 762 |

Obrázok 23. Tabuľka Wheelbases

V prípade pokusu o pridanie vozidla s podtriedou mimo uvedených v tabuľke (Tab. 2) je podtrieda pridaná do tabuľky Subclasses a je jej automaticky pridelená trieda 13 (iné vozidlá), počet náprav vozidla je určený počtom rázvorov a počet prívesov je nastavený na nula. Takto pridaná podtrieda nie je súčasťou navrhnutého algoritmu rozhodovacej štruktúry klasifikácie (nutné doplniť).

Keďže je dôležité aby zmeny v databáze prebehli ako atomická operácia je ukladanie dát zo súboru do databáze spracované transakciou. V prípade ukladania dát zo súboru (Obr. 21) je pre takúto operáciu postačujúca metóda SaveChanges(), ktorá všetky zmeny kontextu prevedie ako jednu transakciu. Ak je transakcia úspešné dokončená, potom sú všetky operácie v databáze vykonané. V opačnom prípade sa operácie vrátia do predchádzajúceho stavu a testovacie dáta z daného súboru nebudú do databáze vložené.

Súbory sú spracovávané postupne a každý pridaný, resp. nepridaný súbor s prípadnou chybou je zaznamenaný ako výstup na konzole. Po prejdení všetkých súborov je výstupom taktiež výpis o úspešnom, resp. neúspešnom pridaní datasetu. Za úspešne pridaný dataset je považovaný len ten, ktorý obsahuje aspoň jedno vozidlo.

```
The file ucid35.csv added to dataset.
The file ucid36.csv added to dataset.
The file ucid5.csv added to dataset.
The file ucid6.csv added to dataset.
The file ucid9.csv added to dataset.

Dataset added successfully.
```

Obrázok 24. Pridávanie datasetu

Po úspešnom pridaní datasetu sú odstránené nedokončené testy z predchádzajúceho datasetu a zároveň aj datasety, ktoré nemajú žiadny klasifikačný test. Na záver je zobrazený prehľad datasetu.

### 7.2.2 Prehľad

Prehľad zobrazuje informácie týkajúce sa testovacích dát datasetu. Nasledujúci obrázok znázorňuje časť tabuľky zoskupujúcej vozidlá podľa podtried. Stĺpce zaznamenávajú identifikačné čísla podtried, počty vozidiel a rázvory, ktoré sú špecifické pre danú podtriedu. Vzdialenosti medzi jednotlivými nápravami definuje interval $wb \in [Lo, Hi]$, ktorý bol rozobratý ako súčasť popisu obrázka (Obr. 12).

| subclassId | count | wb0 | wb1 | wb2 | wb3 |
|---|---|---|---|---|---|
| 2 | 94 | [1785,3228] | | | |
| 4 | 99 | [3014,4748] | | | |
| 9 | 84 | [3103,6934] | | | |
| 11 | 36 | [5794,7051] | | | |
| 42 | 29 | [930,1702] | | | |
| 5 | 8 | [3862,4760] | [596,810] | | |
| 7 | 62 | [2457,3096] | [2816,5979] | | |
| 12 | 87 | [3084,6125] | [1324,1452] | | |
| 16 | 79 | [4744,7304] | [4996,9887] | | |
| 39 | 70 | [6060,7316] | [1333,1545] | | |
| 44 | 2 | [1636,1932] | [2902,4383] | | |
| 46 | 5 | [5119,5126] | [6707,6745] | | |
| 70 | 36 | [3497,3850] | [5266,9445] | | |
| 72 | 41 | [2859,4756] | [2966,6634] | | |
| 3 | 3 | [4307,4315] | [1013,1024] | [5519,6267] | |
| 14 | 65 | [1683,2229] | [2310,4706] | [1340,1404] | |
| 17 | 71 | [4506,6302] | [6080,8612] | [724,1849] | |
| 20 | 64 | [2935,4348] | [3231,6481] | [491,927] | |
| 21 | 44 | [2157,3196] | [3355,6423] | [559,1192] | |
| 34 | 52 | [2654,4111] | [3953,9149] | [1117,1938] | |

Obrázok 25. Dataset – parametre

Rozhodujúcu úlohu pri testovaní majú hraničné typy, a preto je dôležité poznať podtriedy, ktoré nie sú jednoznačne identifikovateľné (prekrytie nastáva na každom rázvore). Zo skupín podtried rozdelených podľa počtu náprav (Tab. 3) sú vytvorené dvojice na princípe kombinácii bez opakovania a ich rázvory sú medzi sebou postupne porovnávané. Rozpätia rázvorov sú známe z predošlého obrázka. Nasledujúci obrázok zobrazuje nájdené hraničné typy a rozbor rázvorov.

| type | wb0 | wb1 | wb2 | wb3 |
|---|---|---|---|---|
| 2_4 | I:[3014,3228]:7,25%<br>:[(12+9)/193] | | | |
| 2_9 | I:[3103,3228]:2,45%<br>:[(5+1)/178] | | | |
| 4_9 | I:[3103,4748]:41,98%<br>:[(93+18)/183] | | | |
| 9_11 | I:[5794,6934]:28,89%<br>:[(27+32)/120] | | | |
| 7_72 | I:[2859,3096]:10,35%<br>:[(20+12)/103] | I:[2966,5979]:78,92%<br>:[(59+39)/103] | | |
| 12_39 | I:[6060,6125]:1,56%<br>:[(3+5)/157] | I:[1333,1452]:54,05%<br>:[(85+44)/157] | | |
| 16_46 | S:[5119,5126]:0,31%<br>:[(0+5)/84] | S:[6707,6745]:0,8%<br>:[(0+5)/84] | | |
| 16_72 | I:[4744,4756]:0,29%<br>:[(1+1)/120] | I:[4996,6634]:23,68%<br>:[(2+8)/120] | | |
| 70_72 | S:[3497,3850]:18,65%<br>:[(36+6)/77] | I:[5266,6634]:21,13%<br>:[(10+4)/77] | | |
| 20_21 | I:[2935,3196]:11,95%<br>:[(20+6)/108] | S:[3355,6423]:94,4%<br>:[(62+44)/108] | I:[559,927]:52,56%<br>:[(62+26)/108] | |
| 21_34 | I:[2654,3196]:27,77%<br>:[(34+1)/96] | I:[3953,6423]:42,64%<br>:[(40+7)/96] | I:[1117,1192]:5,51%<br>:[(2+1)/96] | |
| 27_28 | I:[3173,3631]:13,42%<br>:[(2+32)/121] | I:[1298,1413]:33,24%<br>:[(86+33)/121] | I:[4772,6473]:39,74%<br>:[(41+3)/121] | S:[1304,1812]:52,64%<br>:[(57+34)/121] |
| 27_38 | I:[6099,6397]:7,5%<br>:[(15+4)/102] | S:[1309,1358]:14,75%<br>:[(47+15)/102] | S:[5500,6324]:29,05%<br>:[(22+15)/102] | I:[863,1000]:12,02%<br>:[(2+5)/102] |
| 32_78 | I:[4524,5756]:51,25%<br>:[(1+9)/84] | I:[6082,6933]:16,2%<br>:[(1+10)/84] | S:[968,1412]:85,91%<br>:[(66+11)/84] | S:[965,1432]:94,93%<br>:[(66+15)/84] |

Obrázok 26. Dataset – hraničné typy

Každý rázvor zaznamenáva v prvom riadku typ prekrytia (prienik = I, podmnožina = S), interval prekrytia a percentuálne vyjadrenie dĺžky prekrytia $overlap$ podľa vzťahu (11). Druhý riadok zobrazuje počet vozidiel (počet vozidiel podtriedy A + počet vozidiel podtriedy B), ktoré sa nachádzajú v oblasti prekrytia z celkového počtu vozidiel pre daný hraničný typ. Zároveň z celkového počtu vozidiel je zrejmé, koľko vozidiel bude testovaných v každej iterácií hraničného testu pre daný rázvor.

Okrem vyššie rozobratých vlastností sa zisťuje aj poradie podtried. Poznanie poradia podtried je dôležité pre testovaní hraníc. Nasledujúci obrázok zobrazuje ukážku časti kódu, ktorá porovnáva rozpätia rázvorov dvoch podtried (napr. na wb0) a zisťuje tieto všetky potrebné informácie, ktoré sa týkajú prekrytia medzi dvoma podtriedami. Príkaz else if v ukážke rieši dve situácie, t. j. dve možnosti poradia podtried, aké môžu nastať pri prekrytí typu prienik. Následne sú definované ďalšie dve vetvy pre prekrytie typu podmnožina a v prípade nesplnenia žiadnej z predchádzajúcich podmienok je vrátené prekrytie typu žiadne (none). Podmienky sú napísané tak, aby zahŕňali všetky situácie uvedené na obrázku (Obr. 14).

```
if ((LoSubclassB <= HiSubclassA) && (HiSubclassA < HiSubclassB) && (LoSubclassA < LoSubclassB)
    && (LoSubclassA != HiSubclassA) && (LoSubclassB != HiSubclassB))
{
    // Typ prekrytia.
    overlap.Type = OverlapTypeOption.intersection.ToString();
    // Poradia podtried.
    overlap.SubclassA = subclassA;
    overlap.SubclassB = subclassB;
    // Počiatok intervalu prekrytia.
    overlap.From = LoSubclassB;
    // Koniec intervalu prekytia.
    overlap.To = HiSubclassA;
    // Percentuáne vyjadrenie dĺžky prekrytia podľa vzťahu (2).
    overlap.Percentage =
        Math.Round(((double)(overlap.To-overlap.From + 1)/(HiSubclassB-LoSubclassA + 1)) * 100, 2);
    // Počet vozidiel v oblasti prekrytia z celkového počtu vozidiel hraničného typu.
    overlap.VehiclesCount =
        OverlapVehiclesCount(overlap.From, overlap.To, subclassA, subclassB, wheelbaseNumber);
}
else if ((LoSubclassA <= HiSubclassB) && (HiSubclassB < HiSubclassA) && (LoSubclassB < LoSubclassA)
    && (LoSubclassB != HiSubclassB) && (LoSubclassA != HiSubclassA))
{
    overlap.Type = OverlapTypeOption.intersection.ToString();
    overlap.SubclassA = subclassB;
    overlap.SubclassB = subclassA;
    overlap.From = LoSubclassA;
    overlap.To = HiSubclassB;
```

Obrázok 27. Prehľad – prekrytie typu prienik

## 7.3 Testy

Na výber existujú dva typy testov. Nájdenie hranice medzi dvoma podtriedami, ktoré nie sú jednoznačné rozlíšiteľné je úlohou hraničných testov. Druhým typom testu je test klasifikačný a jeho cieľom je roztriediť vozidlá z datasetu do podtried uvedených v tabuľke (Tab. 2). Aby bola klasifikácia čo najviac presná, sú do rozhodovacej štruktúry (skript) zakomponované parametre z hraničných testov. Tieto dodatočné parametre nemusia byť nutne použité. To, ktoré hraničné typy budú použité definuje nižšie uvedený textový súbor (Obr. 28).

Vytváranie testov a postupné testovanie ich vozidiel prebieha vždy len nad datasetom, ktorý bol do databázy pridaný ako posledný. Pred vytvorením akéhokoľvek nového testu dochádza najprv ku kontrole stavu predošlých testov. V prípade nájdenia nedokončeného testu nastáva opätovný pokus o úspešné otestovanie vozidiel a vyhodnotenie úspešnosti klasifikácie. Vznik takejto situácie sa predpokladá len pri klasifikácií, a to napr. pri úprave algoritmu rozhodovacej štruktúry navrhnutého skriptu alebo pri pridávaní vlastných klasifikačných skriptov. Tieto testy nemusia byť nutne dokončené, a preto aplikácia ponúka možnosť odstránenia nedokončených testov.

### 7.3.1 Hraničné typy

Ak sú všetky testy úspešne prevedené, potom sú ako prvé načítané hraničné typy podľa tabuľky (Tab. 4) a až následne vytváraný test. Hraničné typy môžu byť pred každým pridávaním nového testu modifikované, a preto sú definované v textovom súbore (boundaryTypes.txt) vo formáte JSON.

```
[{"Name":"2_4", "WheelbaseNumber":0},{"Name":"4_9", "WheelbaseNumber":0},
 {"Name":"9_11", "WheelbaseNumber":0},{"Name":"7_44", "WheelbaseNumber":0},
 {"Name":"7_72", "WheelbaseNumber":0},{"Name":"12_39", "WheelbaseNumber":0},
 {"Name":"16_72", "WheelbaseNumber":0},{"Name":"70_72", "WheelbaseNumber":1},
 {"Name":"20_21", "WheelbaseNumber":0},{"Name":"21_34", "WheelbaseNumber":2},
 {"Name":"15_25", "WheelbaseNumber":0},{"Name":"23_24", "WheelbaseNumber":0},
 {"Name":"27_28", "WheelbaseNumber":0}]
```

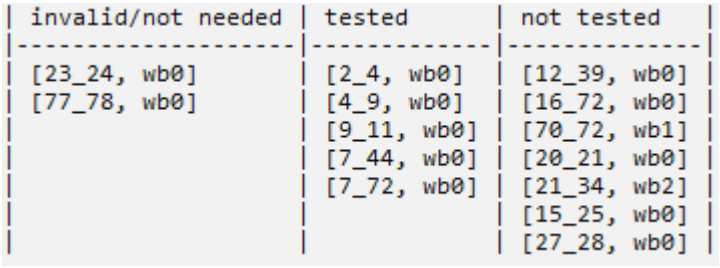
Obrázok 28. Hraničné typy – boundaryTypes.txt

Ak súbor nie je umiestnený v aktuálnom pracovnom adresári aplikácie alebo ak je pri deserializácií JSON reťazca na kolekciu .NET objektov vyvolaná výnimka, potom sú pri vytváraní nového testu použité hraničné typy uvedené v tabuľke (Tab. 4), ktorých súčasťou je štandardný rázvor. Ak súbor obsahuje prázdne pole bez objektov, potom nový klasifikačný test nebude používať žiadne hraničné typy, a teda aj možnosť zvoliť pridávanie hraničných testov nemá význam. Dosiahnutý stav týkajúci sa načítania súboru je zaznamenaný ako výstup na konzole.

### 7.3.2 Kontrola testov

Pred začatím vytvárania nového testu sú ako ďalšie kontrolované predchádzajúce testy aktuálneho datasetu. Dôvodom je zabránenie zbytočnému vytváraniu totožných testov, a to predovšetkým u hraničných testov. Nespočetné množstvo iterácii takého testu môže mať v závislosti od objemu dát datasetu veľký nárok na časovú zložitosť počas testovania vozidiel. Aby sa zamedzilo duplicite hraničných testov môže mať každý dataset vytvorený vždy len jeden test pre danú dvojicu. Dvojica pozostáva z hraničného typu a rázvoru. Okrem toho sa zoznam takýchto dvojíc (získané z textového súboru alebo predvolené) vždy porovnáva s hraničnými typmi, ktoré boli nájdené pri rozbore datasetu (Obr. 26). Testy sú tak vytvárané len pre tie hraničné typy, ktoré si to skutočne vyžadujú.

Ako výstup je na konzole zobrazený výpis roztriedených dvojíc do troch skupín. Prvú skupinu tvoria dvojice, ktoré testovanie nepotrebujú, resp. boli zadané nesprávne zápisy (napr. neexistujúce podtriedy alebo nesprávne poradie podtried). Druhú skupinu tvoria dvojice, ktoré už hraničný test majú. Posledná skupina je pre tie dvojice, ktoré ešte vytvorený test na hľadanie hraníc nemajú a ich zoznam je predaný k vytváraniu hraničných testov.



| invalid/not needed | tested | not tested |
|---|---|---|
| [23_24, wb0] | [2_4, wb0] | [12_39, wb0] |
| [77_78, wb0] | [4_9, wb0] | [16_72, wb0] |
| | [9_11, wb0] | [70_72, wb1] |
| | [7_44, wb0] | [20_21, wb0] |
| | [7_72, wb0] | [21_34, wb2] |
| | | [15_25, wb0] |
| | | [27_28, wb0] |

Obrázok 29. Hraničné typy – kontrola

V prípade voľby pridania klasifikačného testu prebehne rovnaká kontrola ako na obrázku (Obr. 29). Ak existuje aspoň jeden hraničný typ, ktorý ešte nemá vytvorený test na hľadanie hraníc, potom klasifikačný test, resp. testy nebudú pridávané a očakáva sa doplnenie hraničných testov alebo v prípade použitia vstupného súboru jeho upravenie. V opačnom prípade nasleduje kontrola klasifikačných testov, ktorá sa týka len skriptu Classification.lua (7.3.4.2 Klasifikácia). Takto navrhnutý skript pracuje s parametrami rázvorov ich hodnoty vychádzajú vždy len z dát získaných z aktuálneho datasetu, a teda nie je možné dosiahnuť odlišných výsledkov pri opätovnom použitý rovnakých hraničných typov a ich rázvorov. Z toho dôvodu sa kontrolujú použité dvojice a je zrejmé, že v úvahu sú brané len tie hraničné typy, ktoré aj reálne môžu mať hraničné testy. Ak vstupný súbor obsahuje jeden hraničný typ viackrát, potom sa na vytváranie klasifikačného testu použije prvý zadaný (platí pre všetky skripty). Použitie viac hraníc pre jeden hraničný typ (iné rázvory) by vo výsledku prinieslo zhoršenie úspešnosti klasifikácie. Skripty obsahujúce rozhodovací algoritmus klasifikácie sú umiestnené v priečinku Lua v adresári riešenia projektu. K vytváraniu nových klasifikačných testov je predaný zoznam vhodných skriptov.

### 7.3.3 Vytváranie testu

Testy sú vytvárané postupne pre všetky doposiaľ neotestované hraničné typy a ich rázvory, resp. skripty. Každý ďalší test je vytvorený až po úspešnom otestovaní a vyhodnotení predchádzajúceho testu. Pre klasifikačné testy je ako výstup na konzole zobrazený výpis z rozhodnutia o prípadnom nepridávaní nového testu a následne sú postupne pridávané nové testy pre každý vyhovujúci skript. V prípade hraničných testov je už z obrázka (Obr. 29) vidieť, ktoré testy budú pridávané, preto výpis zaznamenáva len, že prebieha vkladanie testov.

```
Classification test [Classification.lua] with required boundaries already exists.
Adding classification test [Classification2.lua]...
```

Obrázok 30. Konzola – pridávanie klasifikačných testov

Pridávanie nového testu si rovnako ako aj pridávanie datasetu vyžaduje úplne naplnenie tabuliek počiatočnými dátami. V tomto prípade je metóda SaveChanges() nepostačujúcou. Nový test musí byť do databázy pridaný ako prvý – pred naplnením tabuliek (Obr. 31) je nutné poznať jeho vygenerované identifikačné číslo. Ak by chyba nastala pri naplnení tabuliek, test by zostal v databáze vytvorený bez potrebných dát. Z toho dôvodu je transakcia riadená skrz API DbContext.Database metódou BeginTransaction(), ktorá vytvorí novú transakciu a trvalé zmeny v databáze nastanú až po potvrdení transakcie metódou Commit().

Takýto prístup umožňuje volať metódu SaveChanges() viackrát a v prípade akéhokoľvek zlyhania test nebude vytvorený a tabuľky Tests, TestParameters, TestIterations, IterationParameters, TestVehicles zostanú bez zmeny.

```
try
{
    using var context = new Models.VehicleClassificationDbContext();
    using var transaction = context.Database.BeginTransaction();

    // Vytvorenie a pridanie nového testu do databáze.
    Test = new Models.Test()
    {
        TestTypeId = (short)Models.TestTypeOption.classification,
        DatasetId = Dataset.Actual.DatasetId,
        Created = DateTime.Now,
        Description = "all vehicles"
    };
    context.Tests.Add(Test);
    context.SaveChanges();

    // Naplnenie tabuliek.
    context.TestParameters.Add(TestParameter(script));
    context.TestIterations.Add(TestIteration());
    context.IterationParameters.AddRange(IterationParameters());
    context.TestVehicles.AddRange(TestVehicles());

    context.SaveChanges();
    transaction.Commit();
```

Obrázok 31. Ukladanie nového klasifikačného testu do databáze

### *7.3.3.1 Tabuľky*

Nový test má v tabuľke Tests automaticky vygenerované identifikačné číslo testu, pridelenú referenciu pre typ testu (klasifikačný test = 1, hraničný test = 2), pridelenú referenciu aktuálneho datasetu, časový údaj o vytvorení a popis testu. Hraničný test má v popise uložený hraničný typ (podtriedy medzi ktorými dochádza k hľadaniu hraníc) a klasifikačný test ukladá reťazec all vehicles (testuje sa na všetkých vozidlách).

| TestId | TestTypeId | DatasetId | Created | Description |
|---|---|---|---|---|
| 2 | 1 | 4 | 2021-04-13 23:36:26 | all vehicles |
| 3 | 2 | 4 | 2021-04-15 23:43:00 | 2_4 |
| 4 | 2 | 4 | 2021-04-15 23:43:30 | 4_9 |
| 5 | 2 | 4 | 2021-04-15 23:44:08 | 9_11 |
| 6 | 2 | 4 | 2021-04-15 23:44:17 | 7_44 |
| 7 | 2 | 4 | 2021-04-15 23:44:22 | 7_72 |
| 11 | 2 | 4 | 2021-04-19 01:14:41 | 70_72 |
| 14 | 1 | 4 | 2021-04-22 01:14:56 | all vehicles |

Obrázok 32. Tabuľka Tests

Následne sú do tabuľky TestParameters uložené parametre testu. Klasifikačný test ukladá len jeden parameter, a tým je názov skriptu. Hraničný test ukladá číslo rázvora, na ktorom sa hranica hľadá, typ prekrytia a poradie podtried pri prekrytí. Všetky tieto parametre sú potrebné k testovaniu (sú totožné pre každú iteráciu testu) a boli zistené už pri zobrazovaní prehľadu datasetu (Obr. 26).

| TestId | Name | Value |
|---|---|---|
| 2 | script | Classification.lua |
| 3 | overlapType | intersection |
| 3 | subclassA | 2 |
| 3 | subclassB | 4 |
| 3 | wheelbaseNumber | 0 |
| 11 | overlapType | subset |
| 11 | subclassA | 70 |
| 11 | subclassB | 72 |
| 11 | wheelbaseNumber | 0 |
| 14 | script | Classification.lua |

Obrázok 33. Tabuľka TestParameters

Obrázok (Obr. 34) zobrazuje spojenie záznamov z tabuliek TestIterations a IterationsParameters spoločným zloženým kľúčom (TestId, IterationNumber). Záznamy z dočasnej novovzniknutej tabuľky boli očíslované a pre názornú ukážku boli odfiltrované niektoré riadky z vybraných testov. Testy boli zvolené tak, aby boli súčasťou aj predošlých tabuliek.

| RowNumber | TestId | IterationNumber | SuccessRate | Name | Value |
|---|---|---|---|---|---|
| 1 | 3 | 0 | 93,78 | wb0_Boundary_2_4 | 3014 |
| 5 | 3 | 4 | 95,85 | wb0_Boundary_2_4 | 3182 |
| 6 | 3 | 5 | 95,34 | wb0_Boundary_2_4 | 3229 |
| 7 | 11 | 0 | 92,21 | wb0_Hi_Boundary_70_72 | 3850 |
| 8 | 11 | 0 | 92,21 | wb0_Lo_Boundary_70_72 | 3497 |
| 9 | 11 | 1 | 83,12 | wb0_Hi_Boundary_70_72 | 3806 |
| 10 | 11 | 1 | 83,12 | wb0_Lo_Boundary_70_72 | 3497 |
| 41 | 11 | 17 | 63,64 | wb0_Hi_Boundary_70_72 | 3850 |
| 42 | 11 | 17 | 63,64 | wb0_Lo_Boundary_70_72 | 3805 |
| 45 | 14 | 0 | NULL | wb0_Boundary_2_4 | 3182 |
| 49 | 14 | 0 | NULL | wb0_Boundary_4_9 | 4429 |
| 52 | 14 | 0 | NULL | wb0_Hi_11 | 7051 |
| 90 | 14 | 0 | NULL | wb0_Lo_11 | 5794 |
| 192 | 14 | 0 | NULL | wb1_Hi_20 | 6481 |

Obrázok 34. Spojené tabuľky TestIterations a IterationsParameters

Počet iterácii hraničného testu v tabuľke TestIterations je pri prekrytí typu prienik daný vzťahom (5) a parametre iterácii uložené v tabuľke IterationParameters vychádzajú zo vzťahu (6). Test s identifikačným číslom TestId = 3 je typickou ukážkou takého testu. Celkovo má šesť iterácií a z toho iterácia s číslom 4 dosiahla najlepší výsledok. Iterácie sú indexované od nuly. Prekrytie typu podmnožina má pre každú iteráciu dvojicu parametrov a sú definované množinou (7). Počet iterácií je odvodený z počtu takýchto dvojíc. Príkladom takého testu je test s identifikačnými číslom TestId = 11. Klasifikačný test má v tabuľke TestIterations vždy len jednu iteráciu a ako jej parametre sú v tabuľke TestParameters uložené charakteristické rázvory podtried a nájdené hranice z už vykonaných hraničných testov aktuálneho datasetu. Pri výbere hraníc je pre každý platný požadovaný hraničný typ nájdená iterácia s najlepšou úspešnosťou (successRate). Ukážkou klasifikačného testu je test s identifikačným číslom TestId = 14. Riadok 45 zobrazuje prevzatý parameter wb0_Boundary_2_4 s hodnotou rázvoru 3182 z najlepšej iterácie hraničného testu TestId = 3. Parametre z posledných troch záznamov odpovedajú charakteristickým rázvorom podtried podľa obrázka (Obr. 25).

Tabuľka TestVehicles ukladá pre každú iteráciu testu zoznam vozidiel, na ktorých bude testovanie hraníc, resp. klasifikácia prebiehať. Vozidlám je priradený stav netestovaného vozidla (false). Klasifikačný test testuje na všetkých vozidlách aktuálneho datasetu, kdežto hraničný test ukladá ku každej iterácii len vozidlá, ktoré odpovedajú podtriedam konkrétneho hraničného typu.

| TestId | IterationNumber | VehicleId | SubclassId | Tested |
|---|---|---|---|---|
| 3 | 5 | 7119323 | 2 | 1 |
| 3 | 5 | 7119325 | 2 | 1 |
| 3 | 5 | 7119506 | 2 | 1 |
| 14 | 0 | 3588727 | NULL | 0 |
| 14 | 0 | 3634386 | NULL | 0 |

Obrázok 35. Tabuľka TestVehicles

Naplnením tabuliek je test úspešne vytvorený. Ďalšou fázou je testovanie vozidiel s následným vyhodnotením.

Vytváranie hraničných testov zahŕňa výnimku, ktorá sa týka hraničných testov, ktoré sú tvorené len jednou iteráciou. Takéto testovanie hranice nemá význam a z toho dôvodu ani tabuľka TestVehicles neobsahuje žiadne vozidlá k testovaniu. Zároveň je úspešnosť tejto iterácie vždy považovaná za najlepšiu a test je tým úspešne dokončený.

### 7.3.4 Testovanie

Testovanie vozidiel prebieha v cykle a je ukončené v prípade, že atribút Tested z tabuľky TestVehicles nenadobúda u žiadneho záznamu hodnotu false (Obr. 35).

Prvé nájdené neotestované vozidlo z daného testu a z danej iterácie je spolu s jemu odpovedajúcim parametrami predané rozhodovaciemu algoritmu za účelom priradenia podtriedy. Medzi základné údaje k testovaniu patria rázvory vozidla z tabuľky WheelBases, parametre iterácie z tabuľky TestIterations a parametre testu z tabuľky TestParameters. V závislosti od typu testu o podtriede rozhoduje časť 7.3.4.1 Hranice alebo časť 7.3.4.2 Klasifikácia. Priradením podtriedy testovanému vozidlu je záznam v tabuľke TestVehicles modifikovaný. Dochádza k uloženiu podtriedy (SubclassId) a nastaveniu atribútu Tested na hodnotu true (Obr. 35).

#### *7.3.4.1 Hranice*

Ak neotestované vozidlo patrí hraničnému testu, potom o jeho podtriede rozhoduje nasledujúci algoritmus.

```
// Prekrytie typu prienik.
if (testParameters["overlapType"].Equals(OverlapTypeOption.intersection.ToString()))
{
    int? value = parameters[$"wb{wheelbaseNumber}_Boundary_{boundaryType}"];
    if (wheelbases[wheelbaseNumber] < value)
    {
        return testParameters["subclassA"];
    }
    else
    {
        return testParameters["subclassB"];
    }
}
// Prekrytie typu podmnožina.
else
{
    int? valueFrom = parameters[$"wb{wheelbaseNumber}_Lo_Boundary_{boundaryType}"];
    int? valueTo = parameters[$"wb{wheelbaseNumber}_Hi_Boundary_{boundaryType}"];
    if ((wheelbases[wheelbaseNumber] >= valueFrom) && (wheelbases[wheelbaseNumber] <= valueTo))
    {
        return testParameters["subclassA"];
    }
    else
    {
        return testParameters["subclassB"];
    }
}
```

Obrázok 36. Hranice – rozhodujúci algoritmus

Ako prvé sa rozhoduje o type prekrytia. Ak práve testované vozidlo pochádza z testu, kde medzi dvoma podtriedami nastalo prekrytie typu prienik, potom je hodnota parametra z konkrétnej iterácie (value) porovnávaná s hodnotou rázvora testovaného vozidla. O tom, medzi ktorými dvoma nápravami vozidla sa hranica hľadá rozhoduje číslo rázvora (wheelbaseNumber) z tabuľky TestParameters. Ak je rázvor vozidla menší ako hodnota potenciálnej hranice (value), potom je vozidlu priradená podtrieda A, inak B. Podtriedy A a B zastupujú identifikačné čísla podtried, ktoré sú uvedené v hraničnom type (boundaryType) a ich poradie bolo známe už pri napĺňaní tabuľky TestParameters. V prípade, že vozidlo pochádza z testu, v ktorom nastalo prekrytie typu podmnožina, potom priradenie podtriedy vozidlu závisí na tom, či hodnota daného rázvora testovaného vozidla spadá, resp. nespadá do intervalu [valueFrom, valueTo]. Interval je daný dvojicou parametrov z iterácie, z ktorej sa vozidlo testuje.

#### *7.3.4.2 Klasifikácia*

Rozhodovací algoritmus nie je napísaný v jazyku C#, ale je súčasťou Lua skriptu. Odčlenenie hlavného algoritmu klasifikácie do skriptu umožňuje jednoduché pridávanie vlastných klasifikácií a predovšetkým ich bezproblémovú modifikáciu počas behu programu. Medzi hostiteľským programom .NET aplikácie a vloženým Lua skriptom musí prebiehať obojsmerná komunikácia. K takému účelu existuje v .NETe knižnica NLua, ktorá umožňuje ľahko integrovať jazyk Lua do .NETu a je voľne k dispozícii ako NuGet balík. NLua beží na vrchole KeraLua (používa najnovšiu verziu Lua 5.4.) a vychádza z pôvodného projektu LuaInterface a ďalej je vyvíjaná nezávislé. API umožňuje aplikácii vytvoriť inštanciu, a tým ovládať Lua interpretera. Jeho trieda potom poskytuje rôzne metódy na vykonávanie Lua kódu, na čítanie a zápis globálnych premenných, na registráciu CLR metód a iné.

```
try
{
    var (wheelbases, parameters, testParameters) = TestedVehicleData();
    int axleCount = wheelbases.Count + 1;

    // Lua interpreter.
    using Lua state = new Lua();
    state.DoFile(Constants.PathLua + testParameters["script"]);
    var scriptFunction = state["FindSubclass"] as LuaFunction;
    var subclass = scriptFunction.Call(axleCount, wheelbases, parameters).First();

    DbUpdateTestedVehicle(Convert.ToInt16(subclass));
```

Obrázok 37. Lua interpreter

Vytvorením novej inštancii triedy Lua sa spustí nový Lua interpreter. Metóda DoFile() tejto triedy vykonáva zdrojový kód súboru Lua, ktorého názov je uložený ako parameter testu v tabuľke TestParameters. Následne je do premennej scriptFunction načítaná funkcia Lua skriptu ako inštancia triedy LuaFunction. Táto trieda definuje rozšírenú metódu Call, ktorá volá odpovedajúcu funkciu a predáva jej vstupné parametre. Po vykonaní funkcie, Lua vždy vracia návratové hodnoty ako pole. Z neho je prevzatá priradená podtrieda a záznam testovaného vozidla je modifikovaný.

Obrázok (Obr. 38) zobrazuje ukážku časti kódu rozhodovacieho algoritmu funkcie FindSubclass(), ktorá je súčasťou Lua skriptu.

```
function FindSubclass(axleCount, wheelbases, parameters)

    -- Dvojnápravové vozidlá.
    if (axleCount == 2) then

        -- Podtrieda (42).
        if parameters:ContainsKey("wb0_Lo_42") then
            if (wheelbases[0] >= parameters["wb0_Lo_42"])
                and (wheelbases[0] <= parameters["wb0_Hi_42"]) then
                return 42
            end
        end

        -- Podtrieda (2).
        if parameters:ContainsKey("wb0_Lo_2") then
            if parameters:ContainsKey("wb0_Boundary_2_4") then
                if (wheelbases[0] >= parameters["wb0_Lo_2"])
                    and (wheelbases[0] < parameters["wb0_Boundary_2_4"]) then
                    return 2
                end
            else
                if (wheelbases[0] >= parameters["wb0_Lo_2"])
                    and (wheelbases[0] <= parameters["wb0_Hi_2"]) then
                    return 2
                end
            end
        end

        -- Podtrieda (4).
        if parameters:ContainsKey("wb0_Lo_4") then...
        end
```

Obrázok 38. Klasifikácia – rozhodujúci algoritmus (Classification.lua)

Na začiatok je dôležité, aby vozidlá boli od seba čo najviac odlíšiteľné, a preto prvým rozhodujúcim atribútom je počet náprav vozidla a až ďalším sú rázvory. V prípade rázvora sa kontroluje, či testované vozidlo spolu so svojimi rázvormi patrí do rozpätia rázvorov, ktoré sú typické pre danú podtriedu (Obr. 25). Tieto hodnoty rázvorov vychádzajú vždy z aktuálneho datasetu a môžu byť upravené nájdenou hranicou, resp. hranicami. Ak dataset nemá zástupcu (vozidlo) pre každú podtriedu, potom sú tieto podtriedy z klasifikácie vynechané.

Z toho dôvodu rozšírená metóda ContainsKey() kontroluje prítomnosť aspoň jedného typického rázvora danej podtriedy. Použitím notácie : je možné jednoducho volať metódy .NET objektu v skripte. Zároveň platí, že ani algoritmus nemusí byť vždy ovplyvnený nájdenou hranicou, resp. hranicami, a preto je vždy vytvorená aj alternatívna cesta.

Tento skript bol navrhnutý tak, aby dokázal bez dodatočných úprav bezchybne riešiť viacero situácii, aké by mohli v datasete nastať. Z toho dôvodu toleruje chýbajúce podtriedy, ale aj voľbu hraničných typov podľa tabuľky (Tab. 4), kde u niektorých je možné si vybrať až z dvoch rázvorov u nich by mohlo dôjsť k získaniu zaujímavých výsledkov.

V závislosti od účelu klasifikácie nie vždy musí byť tolerancia chýbajúcich podtried vhodná, a preto je vytvorený skript (Obr. 39), ktorý vyžaduje prítomnosť všetkých podtried uvedených v klasifikačnom štandarde (Tab. 2.). Tento skript kopíruje logiku skriptu (Classification.lua) a kontroluje aj prítomnosť hraníc. Avšak v prípade, že niektorá podtrieda nemá žiadneho zástupcu je vyvolaná výnimka s výpisom chýbajúceho parametra. Rozsahy rázvorov, ktoré sú špecifické pre chýbajúcu podtriedu je tak nutné dodefinovať.

```
function FindSubclass(axleCount, wheelbases, parameters)

    -- Chýbajúce parametre.
    parameters["wb0_Lo_42"] = 930
    parameters["wb0_Hi_42"] = 1702

    -- Dvojnápravové vozidlá.
    if (axleCount == 2) then

        -- Podtrieda (42).
        if (wheelbases[0] >= parameters["wb0_Lo_42"])
            and (wheelbases[0] <= parameters["wb0_Hi_42"]) then
            return 42
        end

        -- Podtrieda (2).
        if parameters:ContainsKey("wb0_Boundary_2_4") then...
        else...
        end
```

Obrázok 39. Klasifikácia – rozhodujúci algoritmus (Classification2.lua)

### 7.3.5 Vyhodnotenie

Vyhodnotenie úspešnosti testu nastáva po úspešnom otestovaní všetkých vozidiel a pre každú iteráciu testu platí vzťah (12). Počet správne klasifikovaných vozidiel je získaný porovnávaním skutočnej podtriedy vozidla (tabuľka Vehicles) s podtriedou priradenou z klasifikácie (tabuľka TestVehicles) a počet všetkých testovaných vozidiel je daný počtom záznamov z tabuľky TestVehicles pre konkrétny test a danú iteráciu. Dosiahnuté percentuálne vyjadrenie zaznamenáva atribút SuccessRate v tabuľke TestIterations (Obr. 34).

```
// Počet správne klasifikovaných vozidiel danej iterácie testu.
int correctCount = context.TestVehicles
    .Include(i => i.Vehicle)
    .Where(w => w.TestId == testIterations[i].TestId
                && w.IterationNumber == testIterations[i].IterationNumber
                && w.SubclassId == w.Vehicle.SubclassId)
    .Count();

// Počet vsetkých testovaných vozidiel danej iterácie testu.
int vehiclesCount = context.TestVehicles
    .Where(w => w.TestId == testIterations[i].TestId
                && w.IterationNumber == testIterations[i].IterationNumber)
    .Count();

// Úspešnosť iterácie.
double succesRate = Math.Round((double)correctCount / vehiclesCount * 100, 2);
```

Obrázok 40. Vyhodnotenie úspešnosti iterácie podľa vzťahu (12)

Na záver konzola podľa typu testu zaznamenáva odpovedajúci výstup pre každý úspešne dokončený test.

```
Adding boundary tests...

Boundary test [2_4, wb0] completed.
Boundary test [4_9, wb0] completed.
Boundary test [9_11, wb0] completed.
Boundary test [7_72, wb0] completed.
Boundary test [12_39, wb0] completed.
```

Obrázok 41. Úspešné dokončené hraničné testy

#### *7.3.5.1 Prehľad*

Prehľad vychádza z aktuálne dostupných dát uložených v databáze a zaoberá sa dodatočným rozborom dosiahnutých výsledkov z testovania klasifikačných testov. Štandardné vyhodnotenie testov podľa vzťahu (12) je doplnené o ďalšie informácie.

Ako prvé je zobrazený výpis úspešné dokončených klasifikačných testov (Obr. 42). Tabuľka zobrazuje identifikačné čísla testov, časový údaj o vytvorení testu, referenciu na dataset, počet vozidiel datasetu, celkovú úspešnosť testu a názov skriptu, podľa ktorého boli vozidlá klasifikované. Prehľad klasifikačných testov je zobrazený od najnovších a vždy o maximálnom počte 20.

| testId | created | datasetId | success | count | script |
|---|---|---|---|---|---|
| 28 | 21. 4. 2021 13:42:34 | 4 | 94,79% | 1421 | Classification.lua |
| 23 | 20. 4. 2021 11:14:56 | 4 | 95,07% | 1421 | Classification.lua |
| 22 | 13. 4. 2021 23:36:26 | 4 | 95,07% | 1421 | Classification.lua |
| 14 | 11. 4. 2021 22:14:57 | 1 | 93,38% | 1421 | Classification.lua |

Select a testId:

Obrázok 42. Prehľad klasifikačných testov

Zvolením identifikačného čísla testu sú zobrazené dodatočné informácie o teste. V prvom rade sa jedná o informácie týkajúce sa datasetu. Tie boli podrobne rozobraté v časti 7.2.2 Prehľad a konkrétne ide o výpis pre typické parametre podtried (Obr. 25) a nájdené hraničné typy datasetu (Obr. 26).

Ďalšie výpisy sa týkajú samotného testu. Nasledujúci obrázok zobrazuje prehľad hraničných typov, ktoré boli použité klasifikačným testom. Udáva sa číslo rázvora, na ktorom bola hranica hľadaná a nájdená hodnota, ktorá odpovedá najlepšej iterácii z hraničného testu.

| type | wb number | value |
|---|---|---|
| 2_4 | 0 | 3182 |
| 4_9 | 0 | 4429 |
| 7_72 | 0 | 3000 |
| 9_11 | 0 | 6000 |
| 12_39 | 0 | 6060 |
| 16_72 | 0 | 4757 |
| 20_21 | 0 | 2935 |
| 27_28 | 0 | 3632 |
| 32_78 | 0 | 4524 |
| 70_72 | 1 | 5791 |
| 21_34 | 2 | 1193 |
| 27_38 | 3 | 1001 |
| 16_46 | 0 | [5119, 5126] |

Obrázok 43. Prehľad – hranice

Následne je súčasťou prehľadu výpis nesprávne klasifikovaných vozidiel. Vozidlá sú zoskupené do dvojíc podtried. Jedná sa o skutočnú podtriedu vozidla (actual) z tabuľky Vehicles a podtriedu, ktorá bola vozidlu priradená klasifikáciou (predicted). Druhý stĺpec udáva počet koľkokrát takéto nesprávne priradenie nastalo.

| actual -> predicted subclassId | count |
|---|---|
| 2 -> 4 | 1 |
| 4 -> 2 | 7 |
| 4 -> 9 | 7 |
| 7 -> 72 | 4 |
| 9 -> 2 | 1 |
| 9 -> 4 | 8 |
| 9 -> 11 | 22 |
| 11 -> 9 | 2 |
| 12 -> 39 | 3 |
| 21 -> 1 | 4 |
| 21 -> 20 | 3 |
| 27 -> 28 | 1 |
| 32 -> 78 | 1 |
| 34 -> 21 | 1 |
| 70 -> 72 | 1 |
| 72 -> 7 | 4 |

Obrázok 44. Prehľad – nesprávne priradenia

Na písanie dotazov za účelom získania rôznych potrebných dát z databáze je použitý jazyk LINQ (syntax lambda). Načítanie súvisiacich dát entít zaisťuje metóda Eager loading. Dotaz pre jeden typ entity, tak súčasne načítava aj súvisiace entity. Rozširujúca metóda Include() definuje, ktoré súvisiace entity majú byť zahrnuté do výsledkov dotazu. Okrem iného bola pri vývoji klasifikačného nástroja použitá rada ďalších rozširujúcich metód uvedených v tabuľke (Tab. 1) a iné. Nasledujúca ukážka využíva metódy filtrácie, projekcie, zoskupenia a usporiadania dát. Získané údaje odpovedajú výpisu z obrázka (Obr. 44).

```
var incorrectlyAssigned = context.TestVehicles
    .Include(i => i.Vehicle)
    .Where(w => w.Vehicle.SubclassId != w.SubclassId && w.TestId == testId)
    .Select(s => new
    {
        Actual = s.Vehicle.SubclassId,
        Predicted = s.SubclassId
    })
    .GroupBy(g => new { g.Actual, g.Predicted })
    .Select(s => new
    {
        s.Key.Actual,
        s.Key.Predicted,
        Count = s.Count()
    })
    .OrderBy(o => o.Actual)
    .ToList();
```

Obrázok 45. Dotaz – nesprávne priradenia

V poslednej rade prehľad zahŕňa úspešnosť jednotlivých podtried a tried. Nasledujúci obrázok zobrazuje výpis z dosiahnutých výsledkov. Ľavá strana tabuľky je venovaná podtriedam a pravá strana zoskupuje podtriedy do tried podľa tabuľky (Tab. 2). Vyhodnotenie úspešnosti prebehlo pre podtriedy podľa vzťahu (13) a pre triedy podľa vzťahu (14). Ďalej je vždy uvádzaný aj počet správne klasifikovaných vozidiel z celkového počtu vozidiel. Ukážka zobrazuje len časť podtried (2-nápravové a 3-nápravové vozidlá).

| subclassId | success | | classId | success |
|---|---|---|---|---|
| 2 | 98,94% (93/94) | | 2 | 100% (29/29) |
| 4 | 85,86% (85/99) | | 3 | 94,74% (216/228) |
| 9 | 63,1% (53/84) | | 4 | 94,06% (285/303) |
| 11 | 94,44% (34/36) | | 5 | 98,44% (126/128) |
| 42 | 100% (29/29) | | 6 | 63,1% (53/84) |
| 5 | 100% (8/8) | | 7 | 96,63% (86/89) |
| 7 | 93,55% (58/62) | | 8 | 100% (65/65) |
| 12 | 96,55% (84/87) | | 9 | 100% (180/180) |
| 16 | 100% (79/79) | | 10 | 99,13% (114/115) |
| 39 | 100% (70/70) | | 11 | 98,05% (151/154) |
| 44 | 100% (2/2) | | 12 | 100% (46/46) |
| 46 | 100% (5/5) | | | |
| 70 | 97,22% (35/36) | | | |
| 72 | 90,24% (37/41) | | | |

Obrázok 46. Prehľad – podtriedy a triedy (úspešnosť)

## 7.4 Menu

Aplikácia klasifikačného nástroja má vytvorené hlavné menu, ktoré pozostáva z piatich základných častí. Patrí sem možnosť vkladať nové datasety (7.2 Dataset), pridávať hraničné a klasifikačné testy (7.3 Testy), zobraziť prehľad aktuálneho datasetu (7.2.2 Prehľad) a zobraziť históriu klasifikačných testov a ich rozbor (7.3.5.1 Prehľad). Zároveň prehľad aktuálneho datasetu zahŕňa aj výpis najlepších nájdených hraníc z už doposiaľ vykonaných hraničných testov (Obr. 43), ktoré môžu byť v akomkoľvek zložení (bez opakovania hraničného typu) použité klasifikačným testom. Z dôvodu zachovania bezchybového stavu aplikácie je súčasťou mazanie nedokončených testov a na záver možnosť ukončiť aplikáciu.

```
VEHICLE CLASSIFICATION
1. Add a new dataset...
2. Boundary test...
3. Classification test...
4. Actual dataset - an overview...
5. Classification tests - an overview...
6. Remove incomplete tests...
7. Exit app...

Select an option:
```

Obrázok 47. Menu aplikácie

# 8 ZHODNOTENIE KLASIFIKAČNÉHO NÁSTROJA

Klasifikačný nástroj bol celkovo otestovaný na dvoch testovacích sadách a pre každý dataset prebehol klasifikačný test dvakrát. Prvé testovanie prebehlo bez využitia nájdených hraníc z hraničných testov a druhé testovanie prebehlo s hranicami. Ako rozhodovací algoritmus klasifikácie bol použitý skript Classification.lua. Nasledujúca tabuľka zobrazuje dosiahnutú úspešnosť týchto testov vyhodnotenú podľa vzťahu (12), celkový počet vozidiel datasetu a počet správne klasifikovaných vozidiel.

Tabuľka 5. Datasety – prehľad

| | **Dataset 1** | | **Dataset 2** | |
|---|---|---|---|---|
| | **Test 1** | **Test 2** | **Test 1** | **Test 2** |
| **Hranice** | nie | áno | nie | áno |
| **Vozidlá** | 1421 | | 1450 | |
| **Správne klasifikované vozidlá** | 1327 | 1351 | 1343 | 1380 |
| **Úspešnosť** | 93,38% | 95,07% | 92,62% | 95,17% |

Z porovnania vykonaných testov je zrejmé, že testy s hranicami dosahujú lepšie výsledky z klasifikácie.

## 8.1 Hranice

Hraničné testy boli vykonané pre všetky hraničné typy uvedené v tabuľke (Tab. 5) a bol u nich použitý štandardný rázvor. V závislosti od dát datasetu boli vytvorené len požadované testy vychádzajúce z obrázka hraničných typov pre daný dataset (Obr. 26).

Tabuľka 6. Datasety – použité hranice

| | | **Dataset 1 – Test 2** | **Dataset 2 – Test 2** |
|---|---|---|---|
| **Hraničný typ** | **Rázvor** | **Hranica** | |
| 2-4 | 0 | 3182 | 3025 |
| 4-9 | 0 | 4429 | 4786 |
| 9-11 | 0 | 6000 | [5581, 6226] |
| 7-72 | 0 | 3000 | 3221 |
| 12-39 | 0 | 6060 | 6141 |
| 16-46 | 0 | [5119, 5126] | - |
| 16-72 | 0 | 4757 | - |
| 70-72 | 1 | 5791 | 6711 |
| 17-20 | 1 | - | 6716 |
| 17-36 | 2 | - | 997 |
| 20-21 | 0 | 2935 | 2978 |
| 21-34 | 2 | 1193 | - |
| 15-25 | 0 | - | 3251 |
| 23-24 | 0 | - | - |
| 27-28 | 0 | 3632 | 3335 |
| 27-38 | 3 | 1001 | 1092 |
| 32-78 | 0 | 4524 | - |

### 8.1.1 Dodatočné testy

Okrem klasifikačných testov, ktoré hľadajú hranice na štandardných rázvoroch boli vytvorené dodatočné klasifikačné testy, ktoré hľadajú hranicu na ďalšom rázvore podľa tabuľky (Tab. 4). Pri testovaní dochádza k zmene vždy len jedného hraničného typu (16-46, 70-72, 17-36, 27-38, 32-78) a zvyšné hranice sú zachované podľa tabuľky (Tab. 6).

Prvý dodatočný klasifikačný test bol vytvorený pre hraničný typ 16-46 (nahradenie rázvora 0 za 1). Novonájdená hranica je pre dataset (Dataset 1) daná intervalom [6707, 6745] a celková úspešnosť testu zostala v porovnaní s pôvodným testom (Test 2) nezmenená, t. j. 95,07%.

Druhý dodatočný klasifikačný test sa týka zmeny u hraničného typu 70-72 (zámena rázvora 1 za 0). V prípade datasetu (Dataset 1) došlo v porovnaní s testom (Test 2) k zhoršeniu úspešnosti z 95,07% na 95% a nájdená hranica už nie je daná jedným bodom ale intervalom [3589, 3850]. Avšak, keďže sa jedná o rozlíšenie dodávok a nákladných vozidiel, mohli by rázvory v kombinácii s výškou vozidla dosahovať podstatne lepšie výsledky. Naopak u druhého datasetu (Dataset 2) došlo hranicou [3585, 3975] k zlepšeniu z 95,17% na 95,31%.

Tretí dodatočný klasifikačný test bol vytvorený pre hraničný typ 17-36 (nahradenie rázvora 2 za 0). Úspšnosť testu u druhého datasetu klesla z 95,17% na 94,83%. Hodnota testovanej hranice bola 6127.

Štvrtý dodatočný klasifikačný test bol vytvorený pre hraničný typ 27-38 (zámena rázvora 3 za 0). U oboch datasetoch došlo k poklesu celkovej úspešnosti. Test prvého datasetu s novonájdenou hranicou 6398 dosiahol 94,79% z 95,07% a test druhého datasetu s hranicou 6379 dosiahol 94,83% z 95,17%.

Piaty dodatočný klasifikačný test sa týka zmeny u hraničného typu 37-78 (zámena rázvora 0 za 1). V porovnaní s pôvodným testom (Dataset 1) nemala nájdená hranica 6082 žiadny vplyv na úspešnosť testu (95,07%).

## 8.2 Nesprávne priradenia

Nasledujúce tabuľky zobrazujú počet nesprávne priradených vozidiel pre každý dataset a jeho testy. Uvádza sa skutočná podtrieda vozidla a podtrieda priradená z testovania. Takýto výpis môže pomôcť pri vytváraní vlastnej rozhodovacej štruktúry klasifikácie.

Tabuľka 7. Datasety – nesprávne priradenia (2/4-nápravové vozidlá)

| | | **Podtrieda** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Skutočná** | **2** | **4** | | **9** | | | **11** | **20** | **21** | | | **34** |
| | | **Priradená** | **4** | **2** | **9** | **2** | **4** | **11** | **9** | **21** | **1** | **20** | **34** | **21** |
| **Dataset 1** | **Test 1** | | - | 9 | - | 1 | 17 | 27 | - | 19 | - | - | 1 | - |
| | **Test 2** | | 1 | 7 | 7 | 1 | 8 | 22 | 2 | - | 1 | 20 | - | 1 |
| **Dataset 2** | **Test 1** | | - | 18 | - | - | 21 | 34 | - | 6 | - | - | - | - |
| | **Test 2** | | 3 | 7 | 2 | - | 9 | 9 | 8 | - | - | 6 | - | - |

Z porovnania testov bez hraníc a s hranicami je zrejmé, že v prípade nejednoznačnosti rozlíšenia podtried uprednostňujú testy bez hraníc aspoň jednu podtriedu pred druhou. Čo je dôsledkom vplyvu poradia podtried v rozhodovacej štruktúre. Preto napr. pri rozlíšení osobných vozidiel (2) a dodávok (4) došlo k bezchybnému zatriedenie osobných vozidiel. Naopak u malého nákladného vozidla (9) je rázvor niektorých vozidiel tak malý, že jedno vozidlo bolo zatriedené ako osobné vozidlo a sedemnástim vozidlám bola priradená podtrieda dodávok (Dataset 1). V prípade testov s hranicami použitá hranica niekedy znemožní klasifikovať vozidlo a je mu priradená podtrieda jeden (neklasifikované vozidlo). Po ďalšom preskúmaní podtried boli do skriptov pridané doplňujúce podmienky na zamedzenie takýchto situácii. Na druhej strane nájdená hranica adekvátne rozdelí podtriedy, a tým určí, kedy ešte vozidlo bude patriť jednej, a kedy druhej podtriede.

Typickým príkladom pre priradenie podtriedy (1) je hraničný typ 16-72. Podtrieda (16) dosahuje maximálnu hodnotu rázvora wb1 = 9887, kdežto u podtriedy (72) dosahuje táto hodnota len 6634. Ak nájdená hranica medzi podtriedami 16 a 72 (wb0) je 4757 a testované vozidlo (z podtriedy (16)) má rázvory (wb0 = 4750, wb1 = 8500), potom by takéto vozidlo podľa hranice malo patriť podtriede (72). Avšak, keďže jeho wb1 je väčší ako 6634 zostane vozidlo neklasifikované. Preto, ak sa rázvor (wb1) testovaného vozidla nachádza v intervale $wb1 \in [6635, 9887]$, hranica sa zanedbá a vozidlu bude priradená podtrieda (16).

Tabuľka 8. Datasety – nesprávne priradenia (3/5-nápravové vozidlá)

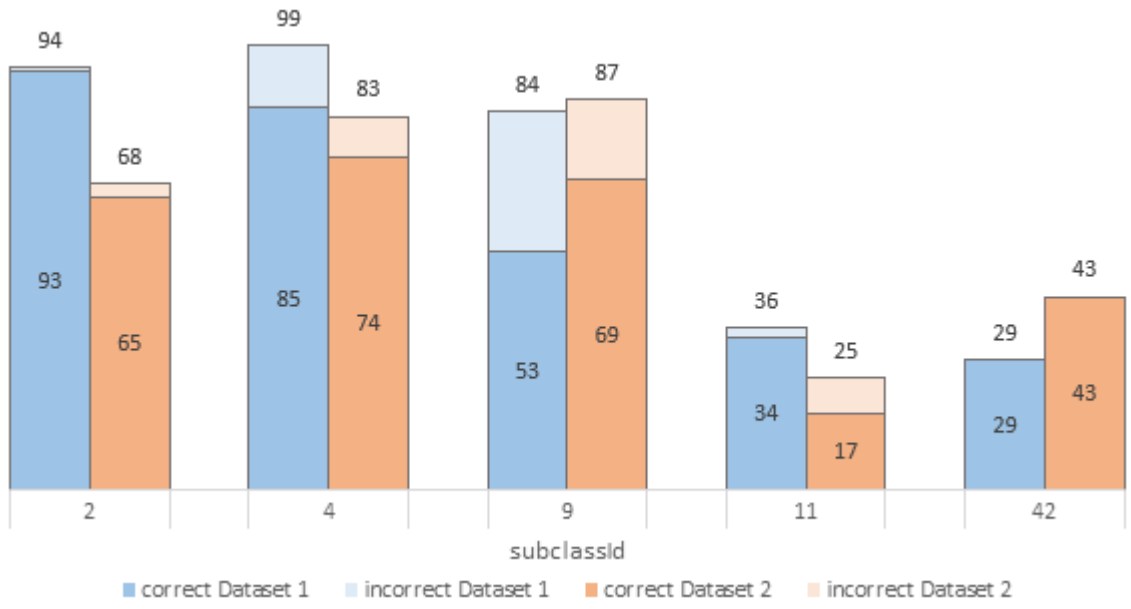
| | | **Podtrieda** | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **Skutočná** | **7** | **12** | **39** | **70** | **72** | | | **15** | **25** | **27** | **32** | **78** |
| | | **Priradená** | **72** | **39** | **12** | **72** | **7** | **16** | **70** | **1** | **15** | **28** | **78** | **32** |
| **Dataset 1** | **Test 1** | | - | - | 4 | - | 12 | 1 | - | - | - | 1 | - | 2 |
| | **Test 2** | | 4 | 3 | - | 1 | 4 | - | - | - | - | 1 | 1 | - |
| **Dataset 2** | **Test 1** | | - | - | 6 | - | 3 | - | 5 | - | 2 | 12 | - | - |
| | **Test 2** | | - | - | 6 | 4 | 3 | - | - | 1 | - | 12 | - | - |

## 8.3 Úspešnosť

Keďže hlavným cieľom klasifikácie je predovšetkým klasifikovať vozidlá s využitím nájdených hraníc (dosahujú podstatne lepšie výsledky) je táto časť zameraná len na tieto testy z tabuľky (Tab. 5) a dosiahnuté výsledky z oboch datasetov sú medzi sebou vzájomne porovnávané. Vzhľadom k nerovnomernému zastúpeniu vozidiel u podtried a tried datasetov sú výsledky zobrazené v grafoch a je uvádzaný počet všetkých vozidiel, počet správnych priradení a počet nesprávnych priradení pre danú podtriedu, resp. triedu daného datasetu.

### 8.3.1 Podtriedy

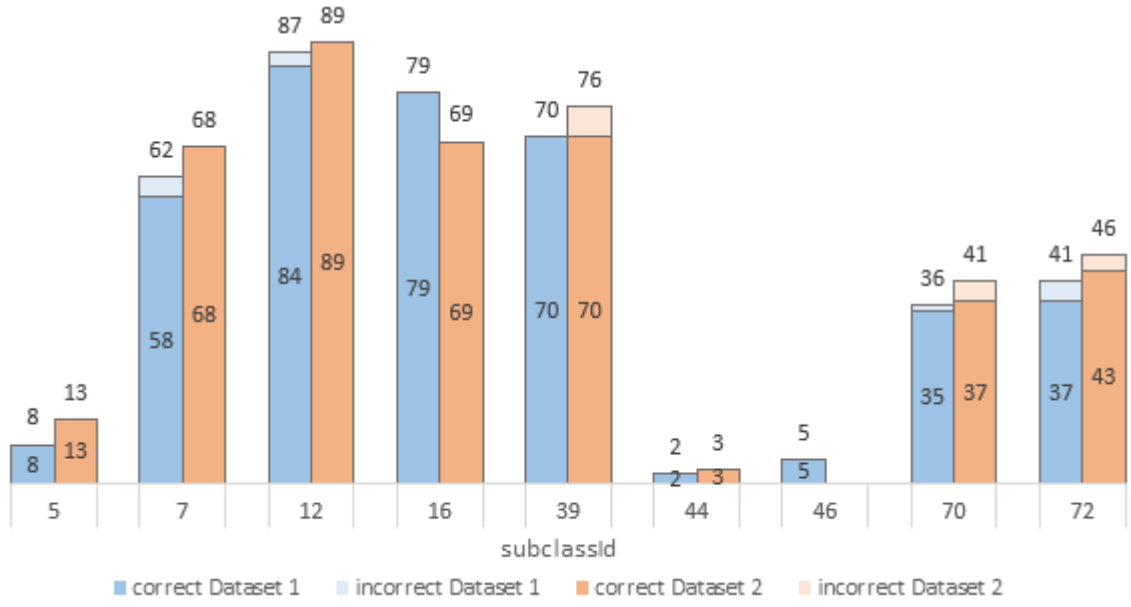
Obrázky (Obr. 48, Obr. 49, Obr. 50 a Obr. 51) zoskupujú vozidlá datasetov podľa identifikačných čísel podtried a tie sú na jednotlivých obrázkoch zobrazené podľa počtu náprav vozidla.




Obrázok 48. Porovnanie podtried – 2-nápravové vozidlá

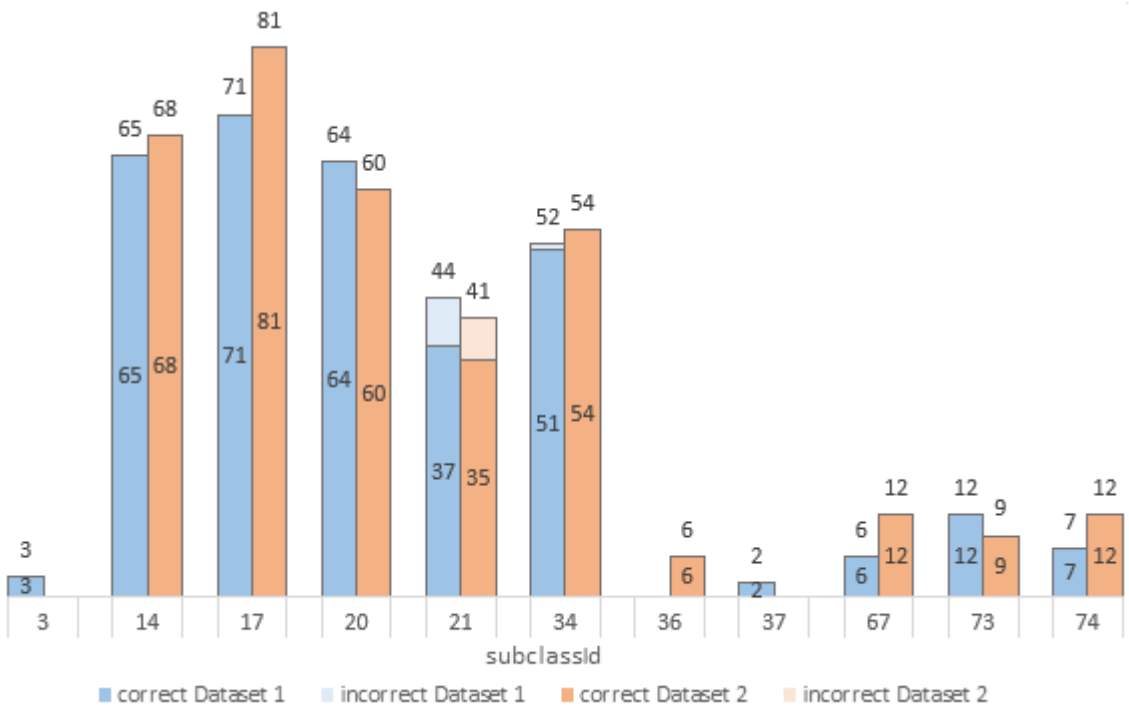
Z porovnania počtu správne priradených vozidiel a všetkých testovaných vozidiel u 2-nápravových podtried je zrejmé, že podtrieda pre malé nákladné vozidlá (9) z prvého datasetu dosiahla najmenšiu úspešnosť 63,1% (druhý dataset 79,31%). Tento výsledok je spôsobený vplyvom použitých hraníc (Tab. 6), ktoré sa ukázali ako najlepšie riešenie na rozdelenie nákladných vozidiel od dodávok (4) a autobusov (11). Spočiatku sa aj nákladné vozidla javili ako dobre odlíšiteľné od autobusov, avšak z druhého datasetu bolo zistené, že rázvory nákladných vozidiel presahujú cez typické rázvory autobusov (podmnožina). Autobusy tak dosiahli v prípade prvého datasetu úspešnosť až 94,44% a u druhého datasetu len 68%. Tieto výsledky môžu byť taktiež ovplyvnené nízkym zastúpením autobusov. Dodávky z prvého datasetu dosiahli úspešnosť 85,86% a z druhého 89,16%. Na druhej strane hranica medzi osobnými vozidlami (2) a dodávkami pomerne presne stanovila hranicu a úspešnosť osobných vozidiel je takmer bezchybná (98,94% a 95,59%).

Podtriedy 2-nápravových vozidiel s výnimkou podtriedy pre motorky (42) nie sú jednoznačne rozlíšiteľne a navyše ich nevýhodou je, že k testovaniu existuje vždy len jeden rázvor.



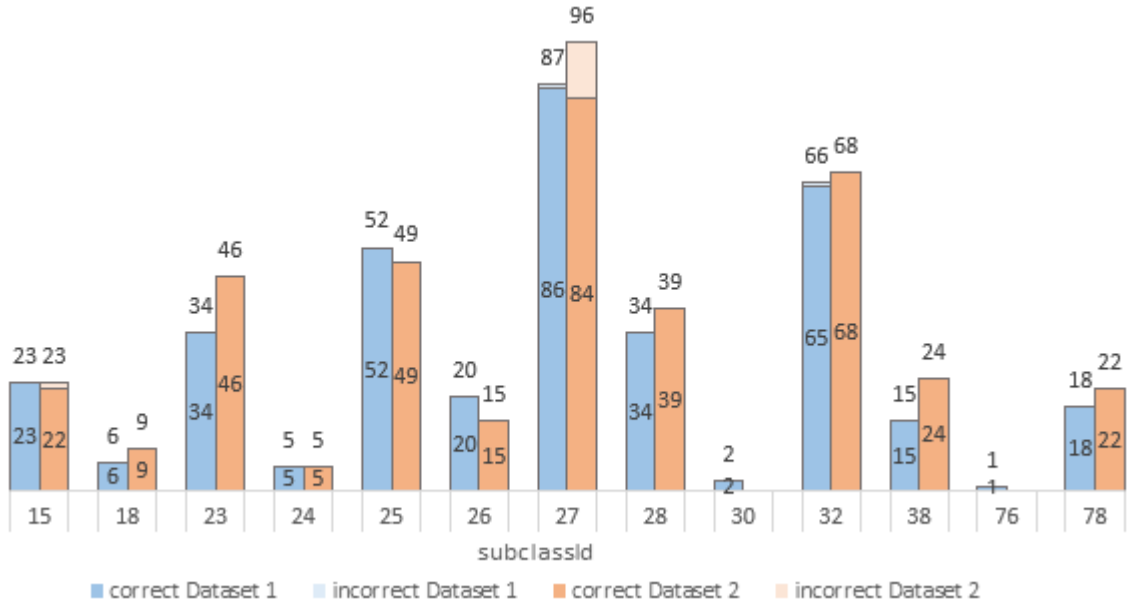

Obrázok 49. Porovnanie podtried – 3-nápravové vozidlá

Medzi podtriedami 3-nápravových vozidiel vznikajú pri prekrytí rázvorov najzložitejšie situácie – obsahujú najviac nerozlíšiteľných podtried. Zároveň platí, že obvykle je každá podtrieda súčasťou len jedného hraničného typu, kdežto v tomto prípade sú niektoré podtriedy súčasťou až dvoch, resp. troch hraničných typov (Tab. 6). Vo výsledku nájdené hranice v niektorých prípadoch pomohli znížiť počet nesprávne zatriedených vozidiel. Napr. pre prvý dataset nastala zmena u dodávok s prívesmi (72). Podľa tabuľky (Tab. 8) klesol tento počet z dvanásť na štyri a úspešnosť dosiahla 90,24%. Naopak u osobných vozidiel s prívesmi (7) stúpol počet nesprávne zatriedených vozidiel o štyri a úspešnosť klesla na 93,55%. Z čoho vyplýva, že hranica (7-72) medzi osobnými vozidlami a dodávkami priniesla zlepšenie o štyri správne priradenia. Na druhej strane nie každá hranica prináša zlepšenie. Hranica 12-39 zachovala počet nesprávne zatriedených vozidiel a úspešnosť podtriedy (39) druhého datasetu zostala nezmenená 92,11%. Pomerne dobré výsledky dosiahla aj podtrieda 2-nápravového ťahača s návesom (70) (97,22% a 90,24%) a podtrieda (12) z prvého datasetu (96,55%). Podtrieda (72) druhého datasetu dosiahla totožný výsledok ako u prvého datasetu.




Obrázok 50. Porovnanie podtried – 4-nápravové vozidlá

Neustálym problémom takmer každej skupiny podtried je nejednoznačná rozlíšiteľnosť osobných vozidiel a dodávok. Výnimkou nie sú ani podtriedy štvornáravových vozidiel. Úspešnosť osobných vozidiel s prívesom (21) dosiahla v prvom datasete 84,09% a v druhom 85,37%. Hranica 20-21 bola stanovená výrazne v prospech dodávok. Takmer 100% úspešnosť dosiahla aj podtrieda 2-nápravového ťahača s 2-nápravovým návesom (34).

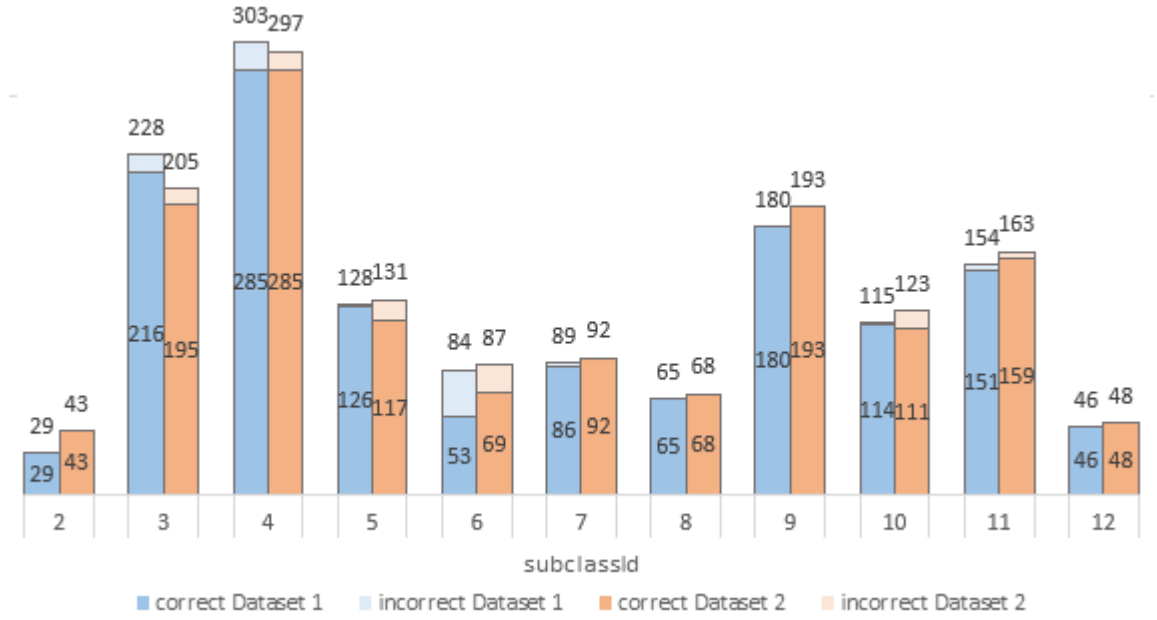



Obrázok 51. Porovnanie podtried – 5-nápravové vozidlá

5-nápravové vozidlá sa spolu so 4-nápravovými vozidlami ukázali ako najlepšie rozlíšiteľné. Rôzne rozloženie náprav a ich počet je dôsledkom, že medzi dvoma podtriedami dochádza k vzájomnému prekrytiu na všetkých rázvoroch len výnimočne. U oboch datasetoch sa ako najviac problematická podtrieda ukázala podtrieda 3-nápravového ťahača s prívesom (27) s úspešnosťou u prvého datasetu 98,85% a u druhého len 87,5%. Takmer zanedbateľné výsledky dosiahli podtriedy (15) a (32).

### 8.3.2 Triedy

Nasledujúci obrázok zoskupuje vozidlá do základných tried uvedených v tabuľke (Tab. 2) s výnimkou podtriedy (1), ktorá je rezervovaná pre neklasifikované vozidlá.




Obrázok 52. Porovnanie tried

Porovnaním výsledkov z klasifikácie je vidieť, že u oboch datasetoch dosiahla najhoršie výsledky trieda pre 2-nápravové nákladné vozidlá (6) s úspešnosťou len 63,1% a 79,31%. Podstatne lepšie výsledky dosiahla trieda osobných vozidiel (3) (94,74% a 95,12%), trieda dodávok (4) (94,06% a 95,96%), trieda autobusov (5) (98,44% a 89,31%), trieda 3-nápravových nákladných vozidiel s prívesmi (10) (99,13% a 90,24%), trieda 2-nápravových ťahačov s návesmi (11) (98,05% a 97,55%) a u prvého datesetu trieda 3-nápravových nákladných vozdiel (7) (96,63%). Z celkového počtu jedenásť tried dosiali len štyri triedy 100% úspešnosť (pre oba datasety).

# ZÁVER

Táto práca sa zaoberá návrhom a realizáciou aplikácie, ktorá klasifikuje vozidlá len na základe vzdialeností medzi jednotlivými nápravami vozidla. Za týmto účelom sa teoretická časť z počiatku zaoberá problematikou klasifikácie vozidiel. Následne sú rozobraté technológii, ktoré boli použité pri vývoji klasifikačného nástroja. V praktickej časti bol ako prvý predstavený navrhnutý klasifikačný štandard, z ktorého klasifikácia vychádza. Tento štandard môže byť bez vplyvu na funkčnosť aplikácie kedykoľvek rozšíriteľný aj o ďalšie typy vozidiel, a to bez ohľadu na počet náprav. Ďalšie časti práce boli venované návrhu, realizácii a zhodnoteniu klasifikačného nástroja.

V rámci aplikácie boli najprv navrhnuté a vytvorené databázové tabuľky k ukladaniu testovacích dát. Ďalej sa aplikácia zaoberá samotnou parametrizáciou. Vzhľadom k podobnosti topológie podvozku u viacerých typov vozidiel nie je vždy jednoduché klasifikovať vozidlá len na základe jedného parametra. Z toho dôvodu bol klasifikačný nástroj navrhnutý tak, aby dokázal riešiť aj takéto situácie, a preto je úlohou doplnkových hraničných testov najprv nájsť najlepšie riešenie na rozdelenie podtried tých vozidiel, ktoré nie sú jednoznačne rozlíšiteľne, a tým dopomôcť k dosiahnutiu čo najlepších výsledkov z klasifikácie. Z dosiahnutých výsledkov práce je zrejmé, že takto nájdené hranice síce pomôžu zvýšiť úspešnosť podtried, ale dosiahnutie 100% úspešnosti pri klasifikácii založenej len na základe rázvorov je u niektorých podtried takmer nemožné. Jedná sa predovšetkým o triedy osobných vozidiel, dodávok, autobusov, malých 2/3-nápravových nákladných vozidiel, ale aj niektorých zástupcov nákladných vozidiel s prívesmi a návesmi.

Zároveň v budúcnosti by takáto klasifikácia mohla byť rozšírená aj o ďalšie charakteristické parametre vozidiel, a to preskúmaním predovšetkým problematických typov vozidiel. Vo výsledku by doplňujúce parametre spolu s rázvormi mohli dosahovať podstatne lepšie výsledky. Medzi takéto parametre by mohla patriť výška, šírka a hmotnosť vozidla, ale aj previsy karosérie, vlastnosti pneumatík a iné.

# ZOZNAM POUŽITEJ LITERATÚRY

[1] BURNOS, Piotr. *Metrology and Measurement Systems* [online]. Poland: Polish Academy of Science, 2010 [cit. 2021-04-10]. Dostupné z: http://www.metrology.pg.gda.pl/full/2010/M&MS_2010_323.pdf

[2] Probability Based Evaluation of Vehicular Bridge Load using Weigh-in-Motion Data. *ResearchGate | Find and share research* [online]. 2016 [cit. 2021-04-10]. Dostupné z: https://www.researchgate.net/publication/303717079_Probability_Based_Evaluation_of_Vehicular_Bridge_Load_using_Weigh-in-Motion_Data

[3] Typover / Databáze certifikátů typů měřidel. *Váhy pro kontrolní vysokorychlostní vážení silničních vozidel* [online]. Brno: ČMI, 30.6.2011 [cit. 2021-04-10]. Dostupné z: http://typover.cmi.cz/typover_pdf/C/4847.pdf

[4] 4 channel loop detector for acquisition of vehicle speed and vehicle class - VEK S4C - Sensors - Products - FEIG ELECTRONIC. *FEIG ELECTRONIC* [online]. FEIG ELECTRONIC [cit. 2021-04-10]. Dostupné z: https://www.feig.de/en/products/sensors/product/vek-s4c/

[5] RAMEZ, Elmasri a NAVATHE Shamkant. *Fundamentals of Database Systems*. 7th ed. India: Pearson, [2015]. ISBN 978-0133970777.

[6] CORONEL, Carlos a Steven MORRIS. *Database systems: design, implementation, and management*. 13th ed. United States: Cengage Learning, [2019]. ISBN 978-1337627900.

[7] CONOLLY, Thomas, Carolyn E. BEGG a Richard HOLOWCZAK. *Mistrovství - databáze: profesionální průvodce tvorbou efektivních databází*. Brno: Computer Press, 2009. ISBN 978-8025123287.

[8] BLAKE, Gregory. *SQL Server 2017 - A Practical Guide for Beginners.* CreateSpace Independent Publishing Platform, 2017. ISBN 978-1975875060.

[9] What is Microsoft SQL Server? A definition from WhatIs.com. *SQL Server: Covering today's SQL Server topics* [online]. [cit. 2021-04-10]. Dostupné z: https://searchsqlserver.techtarget.com/definition/SQL-Server

[10] KOCH, Miloš. *Datové a funkční modelování.* Vyd. 1. Brno : Akademické nakladatelství CERM, 2004. ISBN 80-214-2724-8.

[11] What Is .NET Core. *C# Corner - Community of Software and Data Developers* [online]. [cit. 2021-04-10]. Dostupné z: https://www.c-sharpcorner.com/article/what-is-dot-net-core/

[12] METZGARD, D. a HANSELMAN, S. *.NET Core in action*. Shelter Island, NY: Manning Publications Co., [2018]. ISBN 9781617294273.

[13] .NET Core Overview. *TutorialsTeacher - Learn Web Technologies* [online]. [cit. 2021-04-10]. Dostupné z: https://www.tutorialsteacher.com/core/dotnet-core

[14] BOEHM, Anne a Joel MURACH. *Murach's C# 2015*. 6th ed. Fresno, CA: Mike Murach & Associates, 2016. ISBN 978-1890774943.

[15] ALBAHARI, Joseph a Ben ALBAHARI. *C# 7.0 in a nutshell*. 7th edition. Sebastopol: O´Reilly, [2018]. ISBN 9781491987650.

[16] The history of C# - C# Guide | Microsoft Docs. [online]. Microsoft, 2021 [cit. 2021-04-10]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/whats-new/csharp-version-history

[17] SMITH, Jon P. *Entity Framework core in action*. Shelter Island, New York: Manning Publications Co., [2018]. ISBN 978-1617294563.

[18] Entity Framework Core Tutorials. *Entity Framework Tutorial* [online]. [cit. 2021-04-10]. Dostupné z: https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx

[19] NAGEL, Christian. *Professional C# 7 and .NET Core 2.0*. Indianapolis: Wrox, John Wiley & Sons, [2018], xviii, 1368 s. ISBN 978-1119449270.

[20] LINQ In C#. *C# Corner - Community of Software and Data Developers* [online]. 2020 [cit. 2021-04-10]. Dostupné z: https://www.c-sharpcorner.com/UploadFile/72d20e/concept-of-linq-with-C-Sharp/

[21] Querying Data - EF Core | Microsoft Docs. [online]. Microsoft, 2021 [cit. 2021-04-10]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/querying/

[22] Query Syntax and Method Syntax in LINQ (C#) | Microsoft Docs. [online]. Microsoft, 2021 [cit. 2021-04-10]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/query-syntax-and-method-syntax-in-linq

[23] Programming Languages vs Scripting Languages | Which One is Better?. *Best Online Training & Video Courses | eduCBA* [online]. [cit. 2021-04-10]. Dostupné z: https://www.educba.com/programming-languages-vs-scripting-languages/

[24] Lua: about, 2021. *Lua.org* [online]. 2021 [cit. 2021-04-10]. Dostupné z: https://www.lua.org/about.html

[25] IERUSALIMSCHY, R., FIGUEIREDO, D. L. H. a CELES, W. *Lua 5.4 Reference Manual*. Lua.org. [online]. PUC-Rio, 2020-2021 [cit. 2021-04-10]. Dostupné z: https://www.lua.org/manual/5.4/

[26] MOREIRA, João, André Carlos Ponce de Leon Ferreira CARVALHO a Tomáš HORVÁTH. *A general introduction to data analytics*. Hoboken, NJ: John Wiley, 2018. ISBN 978-1119296249.

# ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

| | |
|---|---|
| ADO | ActiveX Data Objects. |
| API | Application programming interface. |
| atď. | A tak ďalej. |
| CLI | Commnad Line Interface. |
| CLR | Common language runtime. |
| CRUD | Create, read, update, delete. |
| DBMS | Data Base Management System. |
| EF | Entity Framework. |
| IoT | Internet of Things. |
| JSON | JavaScript Object Notation. |
| LINQ | Language Integrated Query. |
| LTS | Long-term support. |
| MS | Microsoft. |
| $\mathbb{N}$ | Množina prirodzených čísel. |
| napr. | Napríklad. |
| resp. | Respektíve. |
| SQL | Structured Query Language. |
| T-SQL | Transact-SQL. |
| t. j. | To jest. |
| VB | Visual Basic. |
| wb | Wheelbase. |
| WIM | Weigh in motion. |
| XML | Extensible Markup Language. |
| $\in$ | Je prvkom. |

# ZOZNAM OBRÁZKOV

# ZOZNAM TABULIEK

# ZOZNAM PRÍLOH

# PRÍLOHA P I: OBSAH CD

Obsah priloženého CD:

- text práce
- testovacie dáta
- zdrojové kódy aplikácie