FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Validátor anonymizačních modelů
Student:	Kateřina Kindlová
Vedoucí:	Ing. Jiří Mlejnek
Studijní program:	Informatika
Obor / specializace:	Webové a softwarové inženýrství, zaměření Softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Seznamte se s existující implementací nástroje Winch, který slouží pro anonymizaci osobních údajů v relačních databázích. Navrhněte rozšíření tohoto nástroje umožňující provést validaci vytvořeného anonymizačního modelu ještě před spuštěním vlastní anonymizace.
Analyzujte informace, které jsou v anonymizačním modelu dostupné, a na jejich základě navrhněte pravidla, která je možné kontrolovat. Pro validaci využijte jak informace o struktuře jednotlivých tabulek, tak i vlastní data uložená v databázi. Navržená pravidla detailně popište, včetně dopadů, které může jejich porušení způsobit.
Navrhněte a popište architekturu tohoto rozšíření a jeho začlenění do existujícího nástroje. Zaměřte se na snadnou rozšiřitelnost pro více databázových systémů. Implementujte kontrolu navržených pravidel pro zvolené databázové systémy. Rozsah implementovaných pravidel a podporovaných databázových systémů konzultujte s vedoucím práce. Výsledné řešení důkladně otestujte.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 12. října 2021 v Praze.

Bakalářská práce

VALIDÁTOR ANONYMIZAČNÍHO MODELU

Kateřina Kindlová

Fakulta informačních technologií
Katedra softwarového inženýrství
Vedoucí: Ing. Jiří Mlejnek
11. května 2022

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů kódu

Nejprve bych chtěla poděkovat vedoucímu své bakalářské práce, Ing. Jiřímu Mlejnkovi, za vstřícnost, ochotu a pomoc během tvorby bakalářské práce. Dále bych ráda poděkovala své rodině a přátelům za podporu a trpělivost.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 11. května 2022 .....

Abstrakt

Tato bakalářská práce se zabývá návrhem a implementací validátoru anonymizačního modelu do nástroje GEM Winch. Rozšíření je napsané do konzolové aplikace Winch Actor v programovacím jazyce Groovy.

Analytická část se zabývá popisem databázového modelu a integritních omezení a představuje nástroj GEM Winch. Na základě získaných informací jsou v části návrhu řešení určena pravidla, která se budou v procesu validovat. Každé takové pravidlo implementuje příslušná třída validátoru. Pokud dojde k porušení pravidla a mohlo by dojít k neúspěchu procesu anonymizace, je vypsáno varování o chybném nastavení. Jednoduše lze zobrazit všechna možná varovaní, která mohou byt vypsána.

Výsledkem práce je praktické rozšíření nástroje GEM Winch, které ulehčuje práci uživatelům nástroje, a zároveň nová implementace a úprava funkcí do aplikace Winch Actor, která usnadní budoucí rozšiřování nástroje.

Klíčová slova validátor, nastavení anonymizačního modelu, anonymizace dat, GEM Winch, rozšíření nástroje Winch, integritní omezení, Groovy

Abstract

The bachelor thesis focuses on creating and implementing anonymization model validator into GEM Winch. This expansion is programmed into Winch Actor application in the programming language Groovy.

The analytical part contains information about a database model, data constraints and introduces the tool GEM Winch. Based on gained information, new rules of validation are drafted in the next chapter. Each rule is implemented by a validator class. If the rule could be violated and the anonymization process might fail, a warning is written. All possible warnings can be easily displayed.

The result of this thesis is a practical extension of GEM Winch, which makes it work better for users, and new implementation and changes to the functions in Winch Actor, which makes future expanding of the project easier.

Keywords validator, anonymization model settings, anonymization, GEM Winch, Winch extension, constraints, Groovy

Seznam zkratek

SW	Software
SQL	Structured Query Language
MSSQL	Microsoft SQL Server
PK	Primární klíč
FK	Cizí klíč
IO	Integritní omezení
GDPR	General Data Protection Regulation
AF	Anonymizační funkce
TD	TruncateDecorator
IBAN	International Bank Account Number
IČO/IC	Identifikační číslo osoby
DIČ	Daňové identifikační číslo
PSČ	Poštovní směrovací číslo
RČ	Rodné číslo

Úvod

Tato bakalářská práce se zabývá tvorbou rozšíření nástroje GEM Winch. Winch je univerzální systém, sloužící k anonymizaci a vytváření řezů dat. Potřeba anonymizovat data vznikla kvůli nutnosti chránit osobní údaje uživatelů, což bylo určeno Obecným nařízením o ochraně osobních údajů, zkráceně ONOOÚ (anglicky GDPR, General Data Protection Regulation), které vešlo v platnost dne 25. května 2018. GDPR je dosud nejvíce uceleným souborem pravidel na ochranu osobních dat. Jednou z povinností zpracovatele je nakládat s osobními daty občana podle toho, jak ten uzná za vhodné, a občan má tedy právo správce požádat o vymazání a zapomenutí svých osobních údajů.

S pomocí nástroje GEM Winch, který je v SW Enterprise Architect nainstalován jako Add-In, lze snadno určit, která data budou předmětem anonymizace, jaký bude jejich objem a jak se budou anonymizovat.

Cílem této bakalářské práce je rozšířit nástroj Winch o možnost validace anonymizačního modelu ještě před puštěním anonymizace dat. Validátor porovná uživatelem nastavený anonymizační model se schématem databáze v Enterprise Architect a připojenou databází. Pokud by nastavení a použití anonymizační třídy mohlo porušit pravidla databázového modelu nebo konkrétní data v databázi neodpovídají vybraným anonymizačním funkcím, validátor v Enterprise Architect vytvoří soupis chyb a upozornění, na které uživatel může nebo nemusí reagovat a svůj anonymizační model podle nich upravit.

Rozšíření implementované v bakalářské práci přinese uživatelům nástroje GEM Winch usnadnění a urychlení jejich práce. Uživatel nebude muset hledat, proč anonymizace neproběhla a omezí možnost špatného využití anonymizačních nástrojů.

Téma bakalářské práce jsem si zvolila, jelikož ochrana osobních údajů je v dnešní digitální době velmi důležitá. Všichni jsme každodenně dotazování na udělení souhlasu se zpracováním osobních údajů, bylo ale zajímavé se na tento problém podívat ze strany zpracovatele. Jak využít získaná data v souladu s GDPR atp. Zároveň mi přišla zajímavá možnost, zapojit se do již existujícího projektu a pomoci ho dále rozvinout.

V analytické části práce (kapitola č. 1) se nachází seznámení s SQL databází a integritními omezeními. Je představen nástroj Winch, jeho funkce a reprezentace anonymizačního modelu. Následuje kapitola č. 2 s návrhem řešení. Na základě analýzy jsou navržena pravidla, která bude validační proces kontrolovat. V další části práce (kapitola č. 3) se nachází popis implementace navržených pravidel a dalších změn, které byly v nástroji Winch provedeny. Poté je v kapitole č. 4 doplněn popis testů, které byly vytvořeny a provedeny. Na závěr jsou v kapitole č. 5 doplněny náměty k budoucímu rozvoji rozšíření.

Cíle

Cílem bakalářské práce je vytvořit rozšíření do již existujícího nástroje GEM Winch umožňujícího provést validaci anonymizačního modelu.

Cílem rešeršní části práce je analyzovat existující nástroj Winch a informace, které jsou v anonymizačním modelu dostupné. Součástí analýzy jsou základní pojmy z oblasti databázového modelu, jazyka SQL, omezení dat a podporovaných datových typů v databázových systémech. Na základě těchto informací budou navržena a detailně popsána pravidla, která je možné při validaci kontrolovat. Vytyčená pravidla validují anonymizační model na základě informací o struktuře jednotlivých tabulek i vlastních dat uložených v databázi.

Cílem praktické části práce je navrhnout a popsat architekturu rozšíření nástroje Winch a začlenit ho do nástroje Winch. Zpracování umožňuje snadnou budoucí rozšiřitelnost pro více databázových systémů. Cílem je implementovat kontrolu navržených pravidel z teoretické části pro zvolené databázové systémy. Součástí praktické části je důkladné testování implementace.

Přínos této bakalářské práce pro uživatele nástroje Winch je usnadnění a urychlení práce. Umožní validovat uživatelem nastavený anonymizační model ještě před spuštěním samotné anonymizace. Upozorní na chyby v nastavení anonymizačního modelu, a předejde tak použití nevhodných anonymizačních tříd a funkcí.

Kapitola 1
Analýza

1.1 Anonymizace a osobní údaje

V této sekci se nachází stručné vysvětlení pojmu anonymizace. Poté se analýza zabývá osobními a citlivými osobními údaji a popisuje několik konkrétních příkladů údajů.

1.1.1 Anonymizace

Proces anonymizace dat spočívá v nezvratné úpravě osobních nebo citlivých údajů v databázi nebo dokumentu. Jelikož je proces nezvratný, neexistuje způsob, jak vrátit data do původní podoby. Takto upravená data nesmí být ani nepřímo spojitelná s jejich vlastníkem a nepomáhají v jeho identifikaci. Jelikož se anonymizovaná data netýkají identifikované či identifikovatelné fyzické osoby, neplatí na ně přísná pravidla o ochraně osobních údajů a lze s nimi volněji nakládat. Výhodou anonymizace je fakt, že anonymizovaná data stále vypadají jako reálná, ale nenesou informace o původních údajích. [1]

1.1.2 Popis osobních údajů

Osobní údaje jsou jakékoli informace o identifikovaném nebo identifikovatelném subjektu údajů.

Do obecných osobních údajů se řadí například jméno, pohlaví, věk a datum narození, osobní stav, IP adresa, fotografický záznam a tzv. organizační údaje (pro podnikající fyzické osoby), kterými jsou například e-mailová adresa, telefonní číslo a identifikační údaje vydané státem. [2]

Mezi osobní údaje patří i tzv. citlivé údaje, což je podle nařízení GDPR speciální kategorie, kam se řadí například údaje o rasovém či etnickém původu, politických názorech, náboženském nebo filozofickém vyznání, členství v odborech, zdravotním stavu, sexuální orientaci, trestních deliktech či pravomocném odsouzení osob atd. [3]

K implementaci validace je potřeba zjistit formát některých osobních údajů, které nástroj GEM Winch aktuálně již umí anonymizovat. Cílem je určení maximální délky (počtu znaků), které mohou údaje nabývat. Následuje analýza těchto osobních údajů:

1. Údaje související s účtem

   Číslo účtu „v národním formátu se skládá z maximálně **16 číselných znaků**. První část, dříve nazývaná předčíslí, je nepovinná. Když je použita, má délku minimálně 2 znaků, maximálně 6 a je od druhé části, která je také dlouhá minimálně 2 a maximálně 6 číselných znaků, oddělena pomlčkou. (…) Kód platebního styku obsahuje 4 číselné znaky, které označují platební instituci, která účet klienta vede.“ [4]

Kód banky je **čtyřmístné číslo**, např. 0800 – Česká spořitelna a.s.

IBAN (International Bank Account Number) je mezinárodní formát čísla účtu. Formát čísla účtu podle standardu může obsahovat číslice a velká písmena se strukturou:

- **2 znaky** pro kód země („CZ“ pro Českou republiku),
- **2 znaky** jako kontrolní číslice,které umožňují kontrolu čísla k ochraně proti chybně zadanému číslu účtu (např. z důvodu překlepu)
- kód banky a číslo účtu v rámci banky (pro Českou republiku **20 znaků**). [5]

Číslo platební karty slouží k jedinečné identifikaci a je základním ochranným prvkem každé platební karty. Číslo karty se skládá ze **16 číslic**, která se řídí pravidly.

- první čtyři čísla určují typ karty a stát původu,
- další čísla pak obsahují informace o instituci, která kartu vydala, a další údaje. [6]

2. Adresa

Poštovní směrovací číslo (PSČ) je číselným označením územního obvodu adresní pošty, které je pro obvod jedinečné. Podle definice jde o **pěticiferné číslo** v rozsahu 00001-99999 a nesmí obsahovat mezery ani žádné jiné alfanumerické znaky. [7]

Číslo popisné (ČP) je v České republice územně prostorový číselný identifikátor budovy, který je v dané části obce jedinečné.

3. Datum

Datum může být zapsáno v různých formátech. Například v nástoroji GEM Winch jsou podporovány následující textové formáty: *„MM/DD/YYYY“*, *„MM/DD/YY“*, *„DD/MM/YYYY“*, *„DD/MM/YY“*, *„YYYY/MM/DD“*, *„YY/MM/DD“*, *„YYYYMMDD“*, *„MON DD YYYY“*, *„Month DD YYYY“*, *„DD Mon YYYY“*, atd.

Nejdelší možný řetězec je tedy *„September DD YYYY“*, který má **17 znaků**. Nebo lze datum zapsat v datovém typu datum (DATE/DATETIME), délka je od **19 do 23** znaků.

4. IČO

„Identifikační číslo osoby (zkráceně IČO) je v České republice unikátní **osmimístné** identifikační číslo právnické osoby, podnikající fyzické osoby nebo organizační složky státu.“ [8]

5. DIČ

Daňové identifikační číslo (DIČ) představuje jednoznačnou identifikaci daňového subjektu (plátce nebo poplatníka). Formát českého DIČ je obvykle řetězec „CZ“ a následně IČO subjektu. Jde tedy o **8-10 číslic**, např. CZ12345678, CZ1234567890. [9]

6. Rodné číslo

„Rodné číslo je **desetimístné** číslo, které je dělitelné jedenácti beze zbytku. První dvojčíslí vyjadřuje poslední dvě číslice roku narození, druhé dvojčíslí vyjadřuje měsíc narození, u žen zvýšené o 50, třetí dvojčíslí vyjadřuje den narození. Čtyřmístná koncovka je rozlišujícím znakem fyzických osob narozených v tomtéž kalendářním dnu o rozsahu přidělovaných koncovek 0000 až 9999.“ [10]

7. Telefonní číslo

Číslo může opět být zapsané v různých formátech (např. s nebo bez předvolby země). Formáty telefonního čísla podporované v nástroji GEM Winch jsou následující: 723527643, 723 527 643, +420723527643, 00420 723 527 643. Nejdelší možný řetězec má tedy celkem 18 znaků, ale nejčastěji používaný formát má pouze **9 znaků**.

1.2 SQL databáze

Validátor bude porovnávat anonymizační model s databází a databázovým modelem, je proto potřeba vysvětlit některé související pojmy. V této části bude popsán databázový model, integritní omezení a jak mohou být definována.

1.2.1 Databázový model

Relační model je nejčastěji používaný databázový model k ukládání dat v databázi. Relačními databázovými systémy jsou např. **Microsoft SQL Server**, **Oracle Database**, **IBM DB2** nebo objektově-relační databázový systém **PostgreSQL**. Nástroj GEM Winch je aktuálně implementovaný pro právě tyto čtyři databázové systémy.

1.2.1.1 Relační databáze

„Databáze založená na relačním modelu, v němž jsou data logicky uspořádána do relací, tj. výsledků kartézského součinu nad doménami neboli množinami údajů. Fyzickou reprezentací relace je **dvojrozměrná tabulka**, jejíž řádky obsahují kolekce údajů o sledovaném objektu (tzv. entity, n-tice, záznamy, věty) a ve sloupcích jsou vždy obsaženy údaje **stejného typu** (tzv. atributy, pole, položky). Tabulky a sloupce jsou identifikovány svými **jmény**, řádky jsou identifikovány **primárním klíčem**. Veškeré vztahy mezi daty jsou vyjádřeny rovněž pomocí **relací**; vztahy mezi souvisejícími záznamy se definují prostřednictvím stejných údajů ve dvojici položek (cizí klíč). Množinový princip relačních databází umožňuje realizovat při manipulaci s daty neprocedurální operace založené na relační algebře a relačním kalkulu – projekce (výběr sloupců), selekce (výběr řádků), spojení (spojení řádků z různých tabulek na základě společné položky).“ [11]

1.2.1.2 Schéma relační databáze

„Schéma relační databáze je dvojice ⟨R, IO⟩, kde R je množina schémat relací a IO je množina **integritních omezení**. (Přípustná) relační databáze pro dané schéma je množina relací se schématem daným v R, které vyhovují všem integritním omezením z množiny IO.“ [12]

Z této definice vyplývá, že při anonymizaci a úpravě dat databáze je potřeba dbát na strukturu tabulek a sloupců a dodržovat definovaná integritní omezení schématu. Validátor má za úkol zkontrolovat nastavení anonymizace a zhodnotit, zda může dojít k jejich porušení.

1.2.1.3 Hodnota NULL

Speciální hodnota NULL se v relační databázi používá, pokud je hodnota ve sloupci neznámá nebo chybí. Nejedná se ani o prázdný řetězec nebo nulovou hodnotu. Specifikace *ANSI SQL-92* uvádí, že hodnota NULL musí být stejná pro všechny datové typy, aby se všechny hodnoty NULL zpracovávaly konzistentně. [13]

1.2.2 Vlastnosti sloupců a integritní omezení

V následující části analýzy budou uvedeny vlastnosti sloupců a integritních omezení (IO) sloupců a tabulek, které bude při validaci mezi databázovým a anonymizačním modelem potřeba kontrolovat. Pokud by anonymizací mělo dojít k porušení nastavených omezení, je pravděpodobně nastaven anonymizační model nesprávně.

1.2.2.1 Indexy v tabulkách

Index (někdy označován jako klíč) je informace ke zrychlení vyhledávacích a dotazovacích procesů v databázi. Například „pokud je potřeba řádky v tabulce zobrazit v pořadí abecedně podle některého textového sloupce, databázový stroj musí napřed vybrat texty začínající písmenem ‘a’, potom texty začínající písmenem ‘b’ atd. U rozsáhlých tabulek může být toto seřazení časově náročné a bylo by efektivnější, kdyby bylo v tabulce zaznamenáno pořadí, v jakém mají být jednotlivé hodnoty v poli seřazeny. Tato informace o pořadí hodnot se označuje jako index. Je možné si je představit jako další skrytý sloupec, ve kterém je pořadí zaznamenáno.“ [14]

V relačních databázích se používají dva základní typy indexů:

1. **Regulární** – hodnoty v indexovaném sloupci se mohou opakovat,
2. **Unikátní** – hodnoty v indexovaném sloupci se nemohou opakovat.

1.2.2.2 Primární klíč

Každá tabulka by měla obsahovat typ indexu, který se označuje jako primární klíč, definovaný na jednom nebo i více sloupcích tabulky. Tyto sloupce jednoznačným způsobem charakterizují jednotlivé záznamy v tabulce. Jedná se v podstatě o **unikátní index**, který však může být v tabulce **pouze jeden**, zatímco obecně unikátních indexů může být definováno i více.

Pro vytvoření primárního klíče je možné použít sloupec, který splňuje podmínky:

1. hodnoty ve sloupci musí být **jedinečné**,
2. sloupec **nesmí obsahovat hodnotu NULL**. [14]

1.2.2.3 Povinné zadávání údajů (*NOT NULL*)

„Zadávání hodnoty NULL je možné u některých sloupců zakázat. Do takto upraveného sloupce je vždy **nutné zadat konkrétní údaj** (u textových polí i prázdný řetězec), jinak není možné záznam vložit.“ [15]

1.2.2.4 Automatické číslování (autoincrement)

Celočíselných sloupce v databázovém systému MSSQL lze upravit tak, aby se v něm hodnoty vytvářely automaticky. Do sloupce s automatickým číslováním **nelze zapisovat**, při vkládání nového řádku se použije nejvyšší použitá hodnota plus jedna. Tento sloupec je výhodné zároveň nastavit jako primární klíč („id“). [15]

V jiných databázových systémech (např. Oracle) lze k vytvoření stejného efektu využít např. ke generování automatických hodnot objekt *SEQUENCE* a jejich přiřazení pomocí uloženého programu *TRIGGER*. [16]

1.2.2.5 Referenční integrita a cizí klíč

V relačních databázích referenční integrita definuje a pomáhá udržovat logické vztahy mezi dvěma tabulkami. Jedna tabulka je vždy nadřízená a druhá podřízená. Cizí klíč je sloupec nebo skupina sloupců, jejichž hodnota v řádku podřízené tabulky je buď prázdná, a nebo musí obsahovat hodnotu primárního klíče z nadřízené tabulky. Jinak řečeno, cizí klíč v podřízené tabulce musí odkazovat na existující klíč v nadřízené tabulce nebo musí obsahovat hodnotu NULL.

Při použití SQL aktualizačních příkazů (*INSERT, DELETE, UPDATE*) se kontroluje platnost referenční integrity a v případě pokusu o její narušení dojde k zahlášení chyby. Při přidání či změně záznamu v podřízené tabulce se kontroluje, zda stejná hodnota klíče existuje v nadřízené tabulce – při porušení se operace neprovede. Při mazání nebo úpravě záznamů v nadřízené tabulce se kontroluje, zda v podřízené tabulce není záznam se stejnou hodnotou klíče – porušení

pravidla může vyvolat chybu nebo předem definovanou úpravu dat podřízené tabulky (nahrazení hodnoty NULL hodnotou, kaskádové odstranění řádků, atd.). [17] [18]

O cizím klíči lze tedy říct, že:

1. může obsahovat **NULL hodnoty** (např. knihu z knihovny nemá zapůjčenou žádný zákazník),

2. nemusí být unikátní. Spojení mezi záznamy v obou tabulkách musí být jednoznačné, proto sloupec z nadřízené tabulky musí být primárním klíčem. V podřazené tabulce ale hodnota cizího klíče **nemusí být jedinečná** a může se vyskytovat vícekrát (např. jeden zákazník si může z knihovny vypůjčit více knih najednou),

3. společné údaje v obou tabulkách musí být **stejného typu**, tedy číslo – číslo nebo text – text. Sloupce se společnými hodnotami by také měli mít **stejnou délku** (což není nutností, ale doporučením).

1.2.3 Příkaz *CREATE TABLE* a definice IO

Syntaxi příkazu *CREATE TABLE* lze zapsat následovně: 1.1

Výpis kódu 1.1 SQL příkaz *CREATE TABLE*

```
CREATE TABLE jmeno_tab(seznam_prvku_tab)
        seznam_prvku_tab ::= prvek_tab [, prvek_tab]...
        prvek_tab        ::= definice_sloupce [definice_IO_tab]
        definice_sloupce ::= jmeno_sloupce datovy_typ [IO_sloupce]
}
```

Příklady integritních omezení sloupce ve schématu databáze:

1. *NOT NULL* – hodnota sloupce nesmí nabývat NULL hodnoty,

2. *DEFAULT* – slouží k určení implicitní hodnoty sloupce. Vyhrazeným slovem *DEFAULT* lze předepsat, co má být do sloupce dosazeno v případě, že není k dispozici jeho hodnota,

3. *UNIQUE* – všechny hodnoty ve sloupci musí být unikátní. Pomocí *UNIQUE* tedy lze definovat další jednoznačný klíč tabulky,

4. *PRIMARY KEY* – sloupec je primárním klíčem tabulky. *PRIMARY KEY* definuje, že daný atribut je primárním klíčem tabulky, tj. jeho hodnoty jsou v tabulce vždy unikátní a nenabývají hodnoty NULL. Složený primární klíč (z více sloupců) je třeba definovat jako IO tabulky.

5. *FOREIGN KEY* – sloupec je cizím klíčem, tj. definuje referenční integritu vzhledem k jiné tabulce,

6. *CHECK* – IO je zadané logickým výrazem. Za vyhrazeným slovem *CHECK* mohou stát jednoduché podmínky omezující hodnoty daného sloupce, ale také podmínky odkazující na jinou relaci.

Všechna IO sloupce se mohou definovat i na **úrovni tabulky** („definice_IO_tab“). Pro referenční integritu je to nutné tehdy, jsou-li klíče složeny z **více sloupců**. Obecně však IO tabulky umožňují vyjádřit mnohem složitější IO. [19]

1.3 Představení nástroje Winch

„Aplikace Winch umožňuje generování anonymizovaných dat z podporovaných relačních databází, anonymizaci souborů s prostým textem i binárním obsahem a zjišťování možných výskytů osobních údajů v relačních databázích.“[20]

Samotný nástroj se skládá z:

1. konzolové aplikace **Winch Actor** vykonávající zmiňované a další procesy,
2. **rozšíření (Add-In) do aplikace Enterprise Architect**, skrze který se spouští Winch Actor s různými příkazy a parametry dle vybraných elementů v Enterprise Architect.

Nástroj obsahuje různé režimy, se kterými lze aplikaci spustit, jako například:

„Initialize“ – instaluje podpůrné databázové objekty do databáze.

„Discovery“ – prozkoumá strukturu a data připojené databáze a pokusí se v nich vyhledat osobní a citlivé údaje (dále viz 1.3.8).

„Deploy“ – připraví prázdné struktury pro uložení anonymizovaných dat

„Execute“ – spustí proces anonymizace

Implementace nového režimu validace je vytvořena v konzolové aplikaci Winch Actor, která je vyvíjena v programovacím jazyce **Groovy** a sestavována nástrojem Gradle. Add-In nástroje Winch pro SW Enterprise Architect je napsán v jazyce C#. [20]

1.3.1 Struktura anonymizačního modelu

Třídy anonymizačního modelu z obrázku 1.1 a jejich asociace v aplikaci Winch Actor kopírují strukturu anonymizačního modelu v nástroji Enterprise Architect.

Třída *WinchApplication* odpovídá stereotypu WApplication v EA, v němž představuje nejvyšší úroveň anonymizačního modelu. Sdružuje pod sebou schémata anonymizačního modelu a elementy označené stereotypem *WinchContext*, jež slouží ke konfiguraci běhu aplikace Winch Actor.

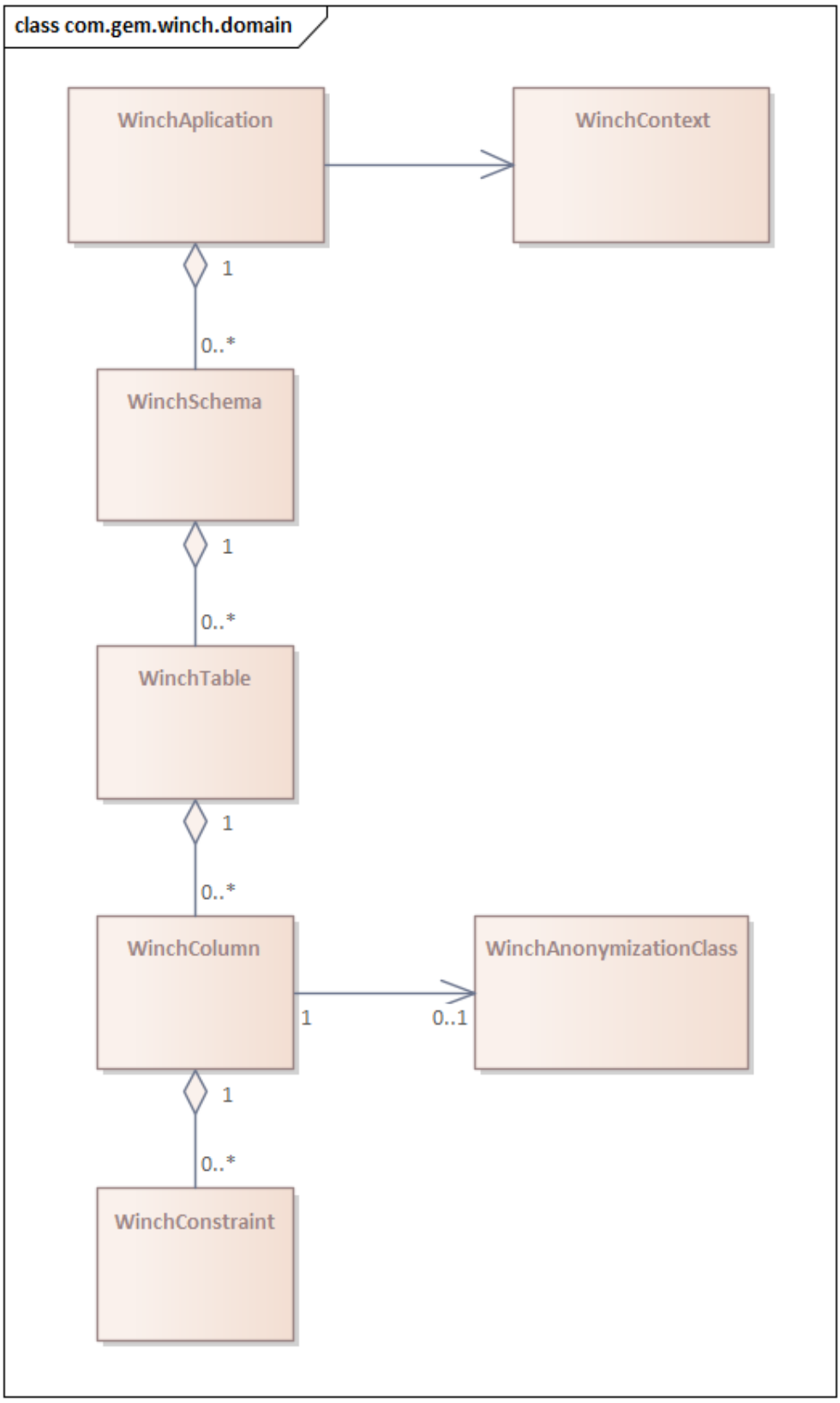
Třída *WinchAnonymizationClass* reflektuje element se stereotypem WAnonymizationClass.

Třída *WinchSchema* představuje EA balíček označený stereotypem WSchema. Obě entity v logickém významu odpovídají schématům v databázích. Každá instance *WinchSchema* si drží referenci na instanci třídy *WinchApplication*, do které patří.

Třída *WinchTable* odpovídá EA elementu se stereotypem WTable. Tato třída i EA element reprezentují **databázovou tabulku**. Instance *WinchTable* uchovává odkaz na instanci *WinchSchema*.

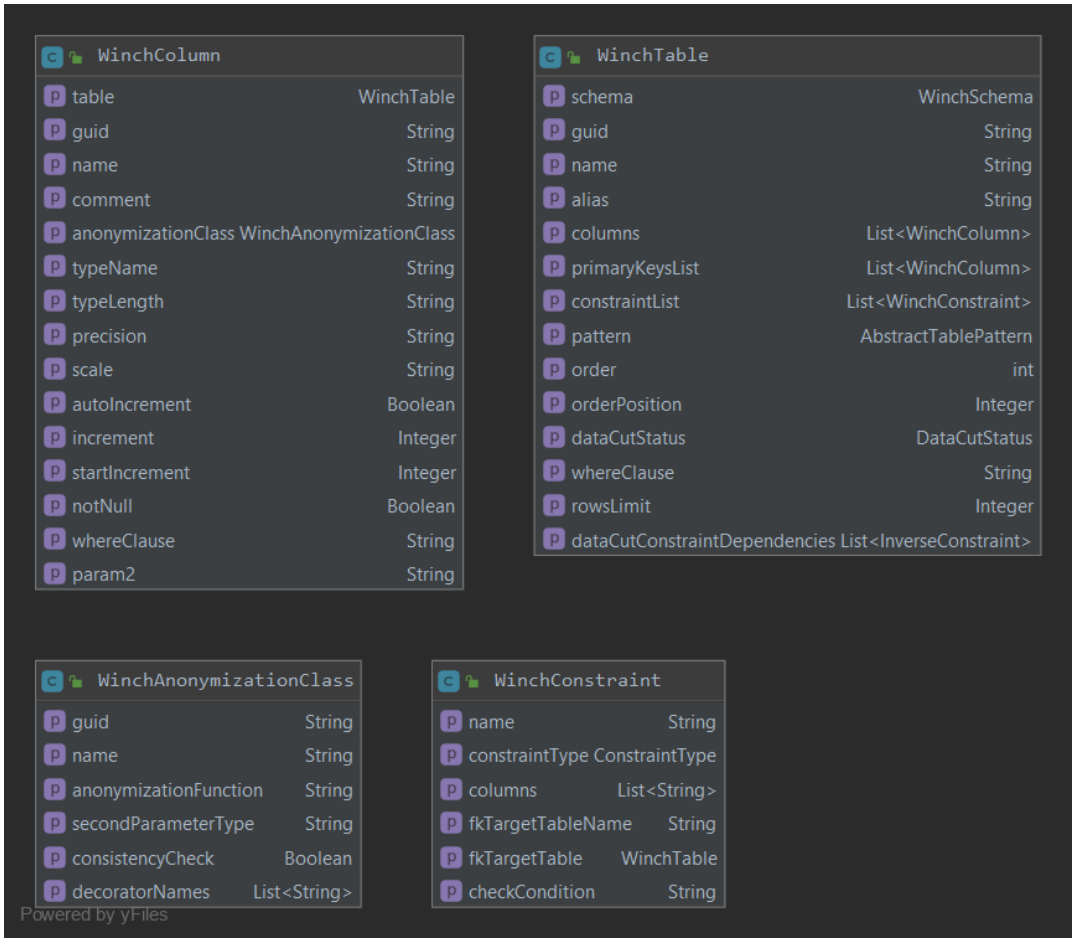
WinchColumn je reflexí EA atributu nesoucí atribut WColumn. Logicky představují **sloupce databázových tabulek**. Instance třídy *WinchColumn* jsou schraňovány v instancích třídy *WinchTable*. Každá instance *WinchColumn* si uchovává referenci na instanci třídy *WinchTable*, jež je sdružuje. Pokud má být sloupec anonymizován, třída *WinchColumn* bude obsahovat třídu *WinchAnonymizationClass*.

Třída *WinchConstraint* představuje EA element označený stereotypem WConstraint. Třída i stereotyp odpovídají **integritním omezením** v relačních databázích. Pokud *WinchConstraint* vyjadřuje IO cizí klíč (typ integritního omezení udává výčtový typ *ConstraintType*), uchovává si daný objekt **referenci na cílovou tabulku** (instanci třídy *WinchTable*). [20]



Obrázek 1.1 Vztahy tříd anonymizačního modelu

Na obrázku 1.2 jsou vidět vlastnosti tříd anonymizačního modelu, které jsou důležité pro proces validace a odkazuje se na ně při implementaci řešení.



Obrázek 1.2 Třídy anonymizačního modelu

1.3.2 Slovníky

Anonymizační funkce mohou k anonymizaci využívat slovníkové tabulky. Slovníky obsahují kompletní **výčet možných hodnot** pro danou anonymizační třídu, případně výběr nejčastěji se vyskytujících hodnot. Používají se jak pro anonymizaci údajů v anonymizačních funkcích, tak pro vyhledávání osobních údajů (viz proces discovery 1.3.8).

1.3.3 Anonymizační funkce

„Winch Connector pro provedení anonymizace jednotlivých osobních údajů používá databázové funkce. Pro každý osobní údaj má Winch implementovanou anonymizační funkci (AF), jejíž SQL kód pro konkrétní databázi je uložen ve zdrojovém souboru v příslušném modulu Connectoru. Anonymizační funkce využívají k anonymizaci různé techniky, jako jsou výpočty s ASCII hodnotami jednotlivých znaků původní hodnoty, modulární aritmetika, maskovaní hodnot, nahrazovaní původních hodnot novými hodnotami z předdefinovaných slovníků atp.“ [21]

Anonymizační funkce je funkce, která na vstupu obdrží původní hodnotu údaje a potenciálně další upřesňující parametry, a na svém výstupu poskytuje anonymizovanou hodnotu, která od-

povídá specifickým potřebám (např. rodné číslo zanonymizuje, a přitom zachová správný formát RČ).

Pro každou anonymizační funkci rozlišujeme několik vlastností, které určují její chování.

Konzistentnost Pokud je AF konzistentní, znamená to, že je **deterministická** tj. pokud na svůj vstup opakovaně dostane stejnou hodnotu, vrátí na výstupu také **shodnou hodnotu**. Z toho vyplývá použití těchto funkcí v případě, kdy je potřeba zaručit, že bude dodržena referenční integrita mezi tabulkami i v případě, kdy je mezi nimi vazba realizována pomocí anonymizovaných hodnot (např. cizí klíč).

Unikátnost Pokud je AF unikátní, pak to znamená, že pro **dvě různé vstupní hodnoty** vždy vrací **různé hodnoty na výstupu**. Bude se hodit v případě provádění anonymizace nad sloupci, které mají unikátní index.

Nenulovost Pokud má funkce tuto vlastnost, pak to znamená, že nikdy **nevrátí na výstupu NULL hodnotu**, pokud neobdržela hodnotu NULL na vstupu.

AF má zadefinované vlastnosti pomocí anotace třídy *AnonFunc* („java.lang.annotation“), viz ukázka kódu 1.2. Anotace obsahuje popis AF, tři logické datové typy, zda funkce splňuje tři výše uvedené vlastnosti nebo ne, a poté seznam parametrů na vstupu AF. Prvním parametrem je ve většině případech původní hodnota osobního údaje.

Výpis kódu 1.2 Anotace anonymizační funkce

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@interface AnonFunc {
    String description() default "N/A"
    boolean consistent()
    boolean unique()
    boolean nullable()
    AnonFuncParameter[] parameters() default []
}
```

Například AF *FunctionFirstName*, která anonymizuje křestní jména, je konzistentní, není unikátní a výsledná hodnota nemůže být NULL. Prvním parametrem je původní hodnota křestního jména a může mít zadán jeden další parametr, který určuje, zda má být použité mužské nebo ženské jméno. Anotace AF potom vypadá následovně: 1.3

Výpis kódu 1.3 Implementace anotací AF *FunctionFirstName*

```
@AnonFunc(description = uv{Anonymizuje krestni jmeno},
          consistent = true, unique = false, nullable = false,
          parameters = [
  @AnonFuncParameter(name="puvodni", description="vstupni jmeno"),
  @AnonFuncParameter(name="puvodni", description="urcuje pohlavi")
])
class FunctionFirstName extends DictionaryFunctionName{...}
```

Dále AF může potenciálně obsahovat informaci o tom, které slovníky se mohou při anonymizaci použít. Například funkce *FunctionFirstName* pracuje se slovníky *FirstnameFDictionary*, který obsahuje ženská křestní jména a *FirstnameMDictionary*, který obsahuje mužská křestní jména. Slovníky lze získat metodou *getRequiredDictionaries()*, viz ukázka 1.4.

Výpis kódu 1.4 Metoda získávající potřebné slovníky pro AF *FunctionFirstName*

```
@Override
List<Class<? extends AbstractDictionary>>getRequiredDictionaries(){
    Arrays.asList(FirstnameFDictionary.class,
                  FirstnameMDictionary.class)
}
```

Aktuální implementace obsahuje 53 anonymizačních funkcí sloužících k anonymizaci různých osobních údajů. V tabulce 1.1 se nachází popis jednotlivých funkcí, seznam všech AF se nachází v příloze B.

Tabulka 1.1 Příklady anonymizačních funkcí

Třída	Popis	Konzistentní	Unikátnost	Nenulovost
FunctionCity	Anonymizuje město	Konzistentní	Neunikátní	Vrací null
FunctionCityDistrict	Anonymizuje město na jiné město v okrese, který je předán jako druhý parametr	Konzistentní	Neunikátní	Vrací null
FunctionFirstName	Anonymizuje křestní jméno	Konzistentní	Neunikátní	Nevrací null
FunctionMail	Anonymizuje email	Konzistentní	Neunikátní	Vrací null
FunctionMailUniq	Anonymizuje email	Konzistentní	Unikátní	Nevrací null

1.3.4 Dekorátory

Třídy dekorátorů umožňují upravit výslednou hodnotu anonymizační funkce. Dekorátory je potřeba do modulu zařadit pouze pokud nestačí základní implementace AF a je nutné ji upravit.

UpperDecorator výstup AF změní na velká písmena. Dekorátor nemá definován parametr. Příklad použití: „UpperDecorator()“.

LowerDecorator výstup AF změní na malá písmena. Dekorátor nemá definován parametr. Příklad použití: „LowerDecorator()“.

TruncateDecorator zkrátí délku výstupu AF od začátku po zvolenou délku. Dekorátor má jeden potenciální číselný parametr. Pokud není zadán parametr, zkrátí se délka výstupu na délku příslušné originální hodnoty. Příklady použití: „TruncateDecorator(10)“, „TruncateDecorator()“.

DefaultValueDecorator vrací hodnotu definovanou v parametru, pokud je hodnota na vstupu shodná s touto definovanou hodnotou. Příklad použití: „DefaultValueDecorator('N/A')“.

NotNullDecorator vrátí hodnotu definovanou v parametru, pokud výstup AF je NULL hodnota. Pokud není zadán parametr, bude vrácena originální hodnota údaje. Příklady použití: „NotNullDecorator('N/A')“, „NotNullDecorator()“.

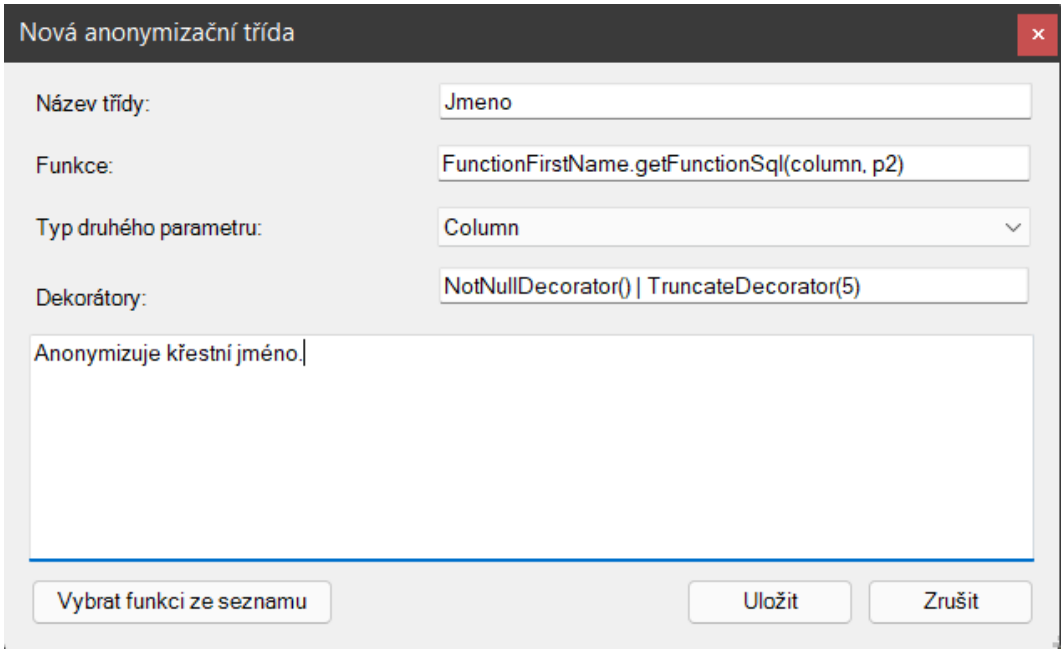
ReplaceDecorator vymění řetězec v prvním parametru za řetězec v druhém parametru ve výstupu anonymizační funkce. Použití: „ReplaceDecorator('from', 'to')“.

Dekorátory *LowerDecorator, UpperDecorator* nemohou ovlivnit ani porušit výslednou délku nebo typ výstupu, pouze upravují znaky řetězce. *DefaultValueDecorator* může pouze změnit výstup anonymizační funkce na vstupní hodnotu údaje, která ve sloupci již byla uložena předtím čili již dodržuje strukturu sloupce.

Dále pro validaci bude tedy uvažováno pouze s dekorátory *NotNullDecorator* a *TruncateDecorator*.

1.3.5 Anonymizační třída

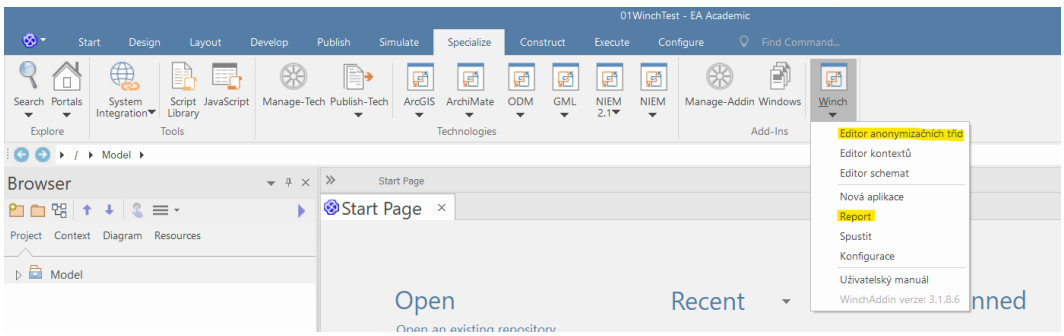
Anonymizační třída je reprezentována třídou *WinchAnonymizationClass*, která má guid a název, jméno vybrané anonymizační funkce (př. „FunctionFirstName.getFunctionSql(column, p2)“), typ druhého parametru AF (př. „Column“) a seznam jmen nastavených dekorátorů s jejich parametrem (př. „NotNullDecorator()“, „TruncateDecorator(5)“). V EA ji lze nastavit jak ukazuje obrázek 1.3.



Obrázek 1.3 Nastavení anonymizační třídy v EA

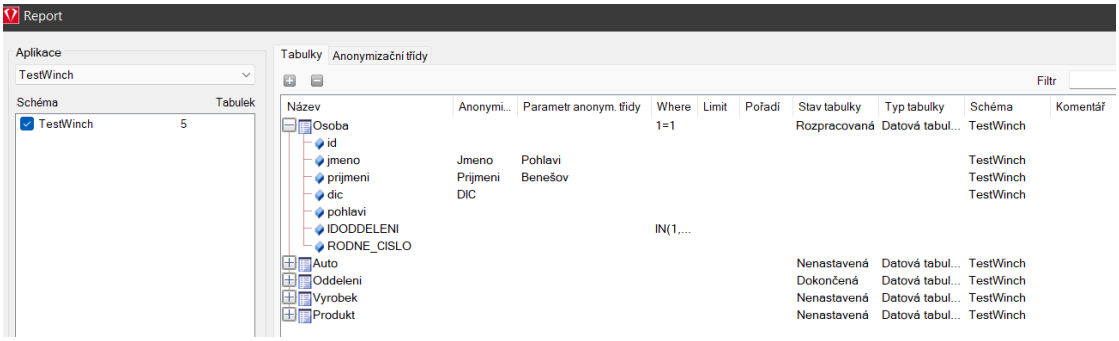
1.3.6 Enterprise Architect a nastavení anonymizace

Poté, co si uživatel nainstaluje GEM Winch Add-In a provede všechna potřebná nastavení, může začít nastavovat konfiguraci pro anonymizaci tabulek v rámci aplikace. Přehled dostupných tabulek a anonymizačních tříd lze zobrazit v menu „Specializace – Winch – Report“, viz obrázek 1.4.

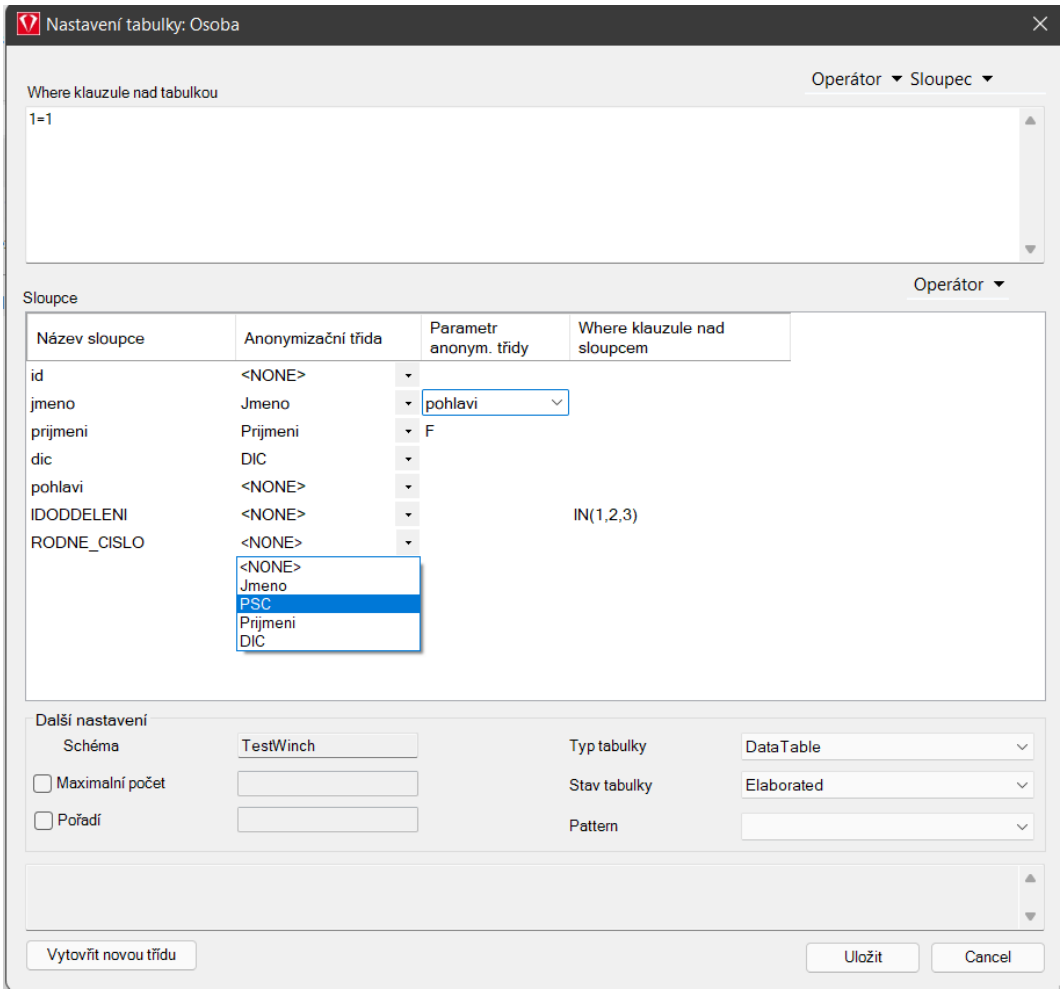


Obrázek 1.4 Nástroj Winch v EA menu

U tabulky lze zobrazit její sloupce a přehled nastavených anonymizačních tříd pro sloupce, druhý parametr anonymizační třídy a WHERE klauzule pro tabulku i sloupec. Viz obrázek 1.5.



Obrázek 1.5 Přehled tabulek a jejich anonymizačních tříd v EA



Obrázek 1.6 Nastavení tabulky v EA

Editační okno pro nastavení tabulky 1.6, na které se lze dostat dvojklikem na jméno tabulky, se skládá ze tří částí:

WHERE klauzule nad tabulkou V první části lze přímo nastavit SQL klauzuli WHERE do textového pole, která slouží k řezům (výběru) dat nad zvolenou tabulkou. Klauzule může

být i složitějšího charakteru. Toto okno tedy slouží k omezení výběru záznamů, které budou výstupem provedené anonymizace.

Sloupce V druhé části okna lze pro každý sloupec nastavit anonymizační třída ze seznamu vytvořených anonymizačních tříd.

- Sloupec „Název sloupce“ je převzat z načtené struktury dané tabulky.
- Sloupec „Anonymizační třída“ slouží k nastavení anonymizační třídy z výběru všech vytvořených.
- Sloupec „Parametr anonymizační třídy“ slouží jako doplňující parametr vybrané anonymizační třídy resp. AF, pokud AF pracuje s parametrem a v anonymizační třídě je nastaven typ druhého parametru jako „Column“ nebo „Variable“ viz níže. Funkce, které tento parametr používají, většinou očekávají název sloupce, na základě kterého se rozlišuje způsob anonymizace, nebo hodnotu, podle které se anonymizační funkce řídí.
- Do sloupce „WHERE klauzule nad sloupcem“ můžete vložit klauzuli WHERE, která slouží jako doplnění nebo alternativa k „WHERE klauzule nad tabulkou“. Umožňuje snadněji zadat řez podle hodnoty v nějakém sloupci. Mezi jednotlivými podmínkami ve „WHERE klauzule nad sloupcem“ a podmínkou zadanou v „WHERE klauzule nad tabulkou“ se používá logický operátor AND.

Další nastavení V třetí části lze nastavit:

- maximální počet přenášených řádků,
- pořadí zpracovávání tabulek v databázi, pokud aplikace obsahuje více tabulek a anonymizace má proběhnout ve striktně zvoleném pořadí,
- typ tabulky – slouží k informačnímu rozlišení různých typů tabulek,
- stav tabulky – slouží k informačnímu rozlišení stavu zpracování parametrů pro danou tabulku. Stav tabulky si uživatel nastavuje sám pro vlastní orientaci,
- vzor – umožňuje ovlivnit vlastní proces anonymizace:
  - zda vytvořit novou tabulku pro anonymizovaná data (vzor „Create as new“),
  - změnit zdrojová data (vzor „Update Source Data“),
  - smazat zdrojová data (vzor „Delete SourceData“).

V menu „Specializace – Winch – Editor anonymizačních tříd“ lze upravovat a vytvářet nové anonymizační třídy. Viz již výše uvedený obrázek: 1.3.

V případě přidávání nové třídy ji uživatel pojmenuje a určí AF ze seznamu anonymizačních funkcí. Typ druhého parametru je možné vybrat z těchto možností:

Žádný („None“) – při přiřazení anonymizační třídy ke sloupečku tabulky nebude možné zadat žádnou hodnotu druhého parametru

Sloupec („Column“) – při přiřazení anonymizační třídy ke sloupečku tabulky bude možné jako druhý parametr vybrat ze seznamu jeden ze sloupečků této tabulky. Například výběr sloupečku obsahující pohlaví, dle kterého se má generovat jméno odpovídající danému pohlaví.

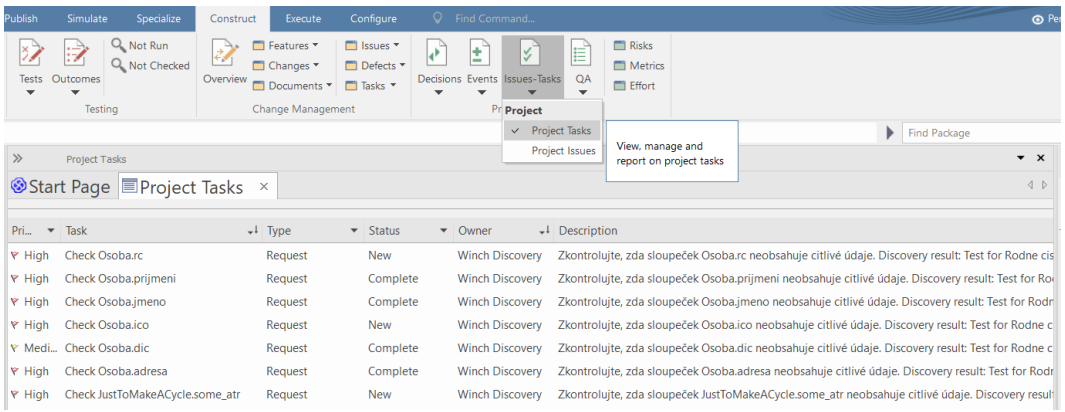
Konstanta („Constant“) – při přiřazení anonymizační třídy ke sloupečku tabulky nebude možné zadat žádnou hodnotu druhého parametru. Hodnota druhého parametru může být zadána jako konstanta přímo v definici anonymizační funkce. Například požadovaný formát data pro datumovou funkci. V tomto případě je konstanta shodná pro použití anonymizační třídy ve všech tabulkách.

Proměnná („Variable“) – při přiřazení anonymizační třídy ke sloupečku tabulky bude možné zadat jako druhý parametr celý SQL výraz. Například v případě, že u AF *FunctionFirstName* není pohlaví uvedeno, použij ženské jméno.

1.3.7 Zapisování úkolů – *Task*

Pokud by při procesu validace mělo dojít k porušení kontrolovaného pravidla, bude potřeba uživatele na tento problém upozornit. V rámci validace budou vznikat různá varování u různých tabulek a sloupců. Uživatel by měl být schopen jednoduše zjistit, co je v anonymizačním modelu za chybu, kde problém vznikl, jakou má závažnost a mohlo by mu být i poskytnuto možné řešení problému.

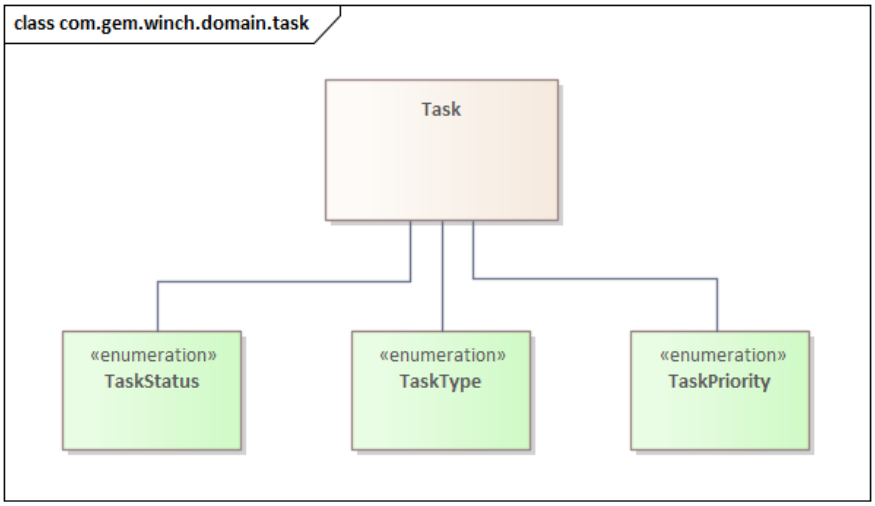
V SW Enterprise Architekt lze využít funkcionalitu v menu „Construct – Project Management – Issues/Tasks – Project Tasks“ jak je zobrazeno na obrázku 1.7.



Obrázek 1.7 Project Tasks v EA menu

V aktuální implementaci nástroje Winch zapisování úkolů do „Project Tasks“ využívá proces discovery.

Úkol reprezentuje třída *Task*, viz obrázek 1.8.



Obrázek 1.8 Třída *Task* v nástroji Winch

K zapisovaní úkolů vytvořených v discovery procesu slouží abstraktní třída *TaskReaderWriter*. Atribut „filepath“ určuje cestu k modelu s úkoly. V případě úložiště anonymizačního modelu v repositáři nástroje Enterprise Architect se může jednat o cestu k souboru s EA modelem, poté

se úkoly v potomkovi *EATaskReaderWriter* zapíší do instance EA. Třída *JsonTaskReaderWriter* zapisuje úkoly do nově vytvořeného JSON souboru.

Pokud při procesu vznikne nový úkol, je uložen a uchován v třídě *TaskSingleton* (návrhový vzor jedináček). Pomocí metody *getTasks()* jde k zapsaným úkolům odkudkoli přistupovat a metodou *addTask()* nový úkol do *TaskSingleton* přidat.

V rámci inicializace exekučního režimu (discovery) lze pomocí třídy *ModelLoadingUtil* získat správného potomka třídy *TaskReaderWriter*. Po dokončení hlavní funkce režimu (objevování citlivých údajů) je třeba nově vzniklé úkoly získat ze třídy *TaskSingleton* a pomocí získané třídy *TaskReaderWriter* zapsat do anonymizačního modelu. Zapsat úkoly do anonymizačního modelu jde pomocí metody *writeTasks()*. Naopak, úkoly již zapsané v anonymizačním modelu lze získat metodou *readTasks()*. [20]

1.3.8 GEM Winch Discovery

Proces pomáhá uživateli s vyhledáváním osobních a citlivých údajů v databázi. Proces projde tabulky a jejích sloupce a pokud u některého sloupce spočítá, že by s dostatečnou pravděpodobností mohl osobní údaje obsahovat, pak doporučí uživateli jeho kontrolu. Ke kontrole sloupců využívá abstraktní třídu *Discoverer*, ze které dědí třídy, které umí vyhledávat konkrétní druh osobního údaje. Výsledek prohledávání zadává jako jednotlivé úkoly (*Task*). V tabulce 1.2 je uvedeno pár příkladů potomků třídy *Discoverer*.

Tabulka 1.2 Příklad tříd dědících z třídy *Discoverer*

Třída	Popis	Typické názvy sloupců	Klíčová slova v komentáři	Přípustné datové typy	Min délka sloupce
CityDiscoverer	Objevuje jména měst ve sloupci.	MESTO, CITY, PLACE, TOWN, OBEC, ADRESA, ADDRESS		TEXT	12
FirstNameDiscoverer	Objevuje křestní jména ve sloupci.	JMENO, FIRST_NAME, FIRSTNAME, FORENAME		TEXT	7
IBANDiscoverer	Oběvuje IBAN ve sloupci.	UCET, CISLO_UCTU, BANKOVNI_UCET, IBAN, BANK, BANK_ACCOUNT	účet, bankovní, číslo účtu, iban, bank, bank account	TEXT	15

Kapitola 2
Návrh řešení

‘

V této sekci budou na základě analýzy navržena pravidla, která bude proces validace kontrolovat. Pravidla se budou dělit na tři typy podle úrovně kontroly:

- pravidla pro formát a vlastnosti anonymizačních tříd,
- pravidla, kde databázové schéma neodpovídá anonymizačnímu modelu, a
- pravidla, kde nesouhlasí data v databázi s nastavením anonymizačního modelu.

2.1 Pravidla pro zadávání anonymizační třídy

Pokud má sloupec být anonymizován, má nastavenou anonymizační třídu (viz implementace 2.1). Anonymizační třída *WinchAnonymizationClass* nese informace potřebné k validaci modelu. Bude potřeba získat vybranou anonymizační funkci, typ druhého parametru a zadané dekorátory s jejich parametrem. Pokud je typ druhého parametru AF zadaný jako „Constant“, bude druhý parametr zadán v řetězci parametru *anonymizationFunction* společně s názvem AF. Pokud je druhý parametr jiného typu, bude se nacházet ve vlastnostech třídy *WinchColumn*.

Výpis kódu 2.1 Implementace anonymizační třídy

```
class WinchAnonymizationClass {
    String guid
    String name
    String anonymizationFunction
    String secondParameterType
    Boolean consistencyCheck
    List<String> decoratorNames
}
```

Název anonymizační funkce si uživatel vybírá ze seznamu funkcí, ale řetězec může upravovat (např. omylem smazat písmeno nebo zadat konstantní parametr). Dekorátory uživatel zadává ručně. Je proto potřeba zkontrolovat, zda zadané řetězce splňují očekávaný formát a zda dávají smysl.

- Anonymizační funkce: Formát řetězce je složen ze 3 částí. První částí je název funkce, ze které se bude vytvářet třída anonymizační funkce, která se bude při procesu validace používat ke

zjištění anotací, maximální délky atd. Poté následuje tečka a předdefinovaný řetězec příkazu „getFunctionSql“. Nakonec jsou v kulatých závorkách zadané parametry funkce. Pokud se jedná o řetězec, který uživatel ručně zadává jako druhý parametr, měl by být označen v jednoduchých uvozovkách. Naopak pokud je prvním parametrem vstupní údaj pro anonymizaci, je většinou zapsán slovem „column“ bez jednoduchých uvozovek.

[NázevFunkce].getFunctionSql(column, ‘p2’)

- Dekorátor: Formát řetězce se skládá z názvu dekorátoru (př. *NotNullDecorator*) a v kulatých závorkách zadaného parametru dekorátoru (př. ‘N/A’).

[NázevDekorátoru](parameter)

Pokud řetězce neobsahují očekávané informace a znaky nebo se podle jména dekorátoru nebo AF nepodařilo najít příslušnou třídu, vypíše se varování pro uživatele.

2.2 Pravidla, kde databázové schéma neodpovídá anonymizačnímu modelu

V databázovém modelu existují tabulky, které jsou tvořeny sloupci. Integritní omezení lze definovat pro celou tabulku nebo pro konkrétní sloupec. Proto budou definována kontrolní pravidla jak pro celé tabulky, tak pro sloupce zvlášť.

2.2.1 Pravidla na úrovni tabulky

Nad tabulkou mohou být definovány IO pro jeden nebo více sloupců tabulky. Pravidla budou vytvořena pro omezení *UNIQUE, PK a FK*.

2.2.1.1 *UNIQUE*

U tohoto omezení se bude sledovat, zda je zadaná AF unikátní nebo ne. Pro dvě různé hodnoty v původních datech vrátí unikátní AF dvě různé hodnoty v nových datech, takže pokud původní dvojice údajů byla unikátní, bude unikátní i nová dvojice.

- Pokud je v unikátním indexu pouze jeden sloupec, a je na něj při anonymizaci aplikována AF, která je unikátní, bude i výsledek anonymizace unikátní. Pokud je AF neunikátní, nemusí být *UNIQUE* omezení dodrženo a je třeba uživatele upozornit.
- Pokud je sloupců dva a více, může nastat více možností (viz tabulka 2.1):
  1. Pokud je všem sloupcům přiřazená anonymizační funkce, která je unikátní, bude i výsledná kombinace unikátní. Jelikož existující data v databázi již pravidlo *UNIQUE* musejí splňovat, je každá kombinace původních údajů již unikátní.
  2. Pokud je použita AF, která je unikátní, a ostatní sloupce AF nemají přiřazenou, bude i výsledná kombinace stále unikátní, jelikož neanonymizovaná data budou s novou kombinací nadále unikátní.
  3. Pokud je alespoň na jeden sloupec použita alespoň jedna neunikátní AF, nebude možné zaručit unikátnost výsledné kombinace. I pokud je na jeden sloupec použita unikátní AF a na druhý neunikátní, data ve sloupci, který je anonymizován unikátně, nemusí mít na vstupu unikátní data, jelikož v tomto sloupci mohli být dva a více údajů stejných a unikátnost mohla být zajištěna kombinací s ostatními sloupci. Tyto stejné údaje v jednom

sloupci muže i unikátní AF anonymizovat na stejné údaje a poté by anonymizací v jiném sloupci neunikátní funkcí mohla vzniknout neunikátní kombinace.

Tabulka 2.1 Kombinace použití anonymizačních funkcí a *UNIQUE* IO

1. sloupec	2.,3.,.. sloupec	Vyhodnocení	Komentář
Neanonymizován	Neanonymizovány,..	OK	Ani jeden sloupec není anonymizován
Unikátní AF	Unikátní AF, …	OK	Každý sloupec má přiřazenou unikátní AF
Unikátní AF	Neanonymizovány,…	OK	Sloupce mají unikátní AF nebo jsou neanonymizovány
Neunikátní AF	Neanonymizovány,…	Chyba	Nezaručí unikátnost
Neunikátní AF	Neunikátní AF,…	Chyba	Nezaručí unikátnost
Unikátní AF	Neunikátní AF,…	Chyba	Unikátní AF nezaručí unikátnost, pokud je použita i neunikátní AF

Závěrem tedy je, že jakmile je použita alespoň jedna neunikátní funkce, nepůjde zaručit unikátnost a uživatel by měl být varován.

- Pokud je u anonymizační třídy nastaven dekorátor *TruncateDecorator*, který může zkrátit délku anonymizovaného údaje, mohlo by se stát, že jeho oříznutý výsledek bude stejný, jako jiný zkrácený údaj ve sloupci. Například pokud budou dva údaje anonymizovány AF *FunctionCity* na dva různé údaje ze slovníku „Brandýs nad Labem-Stará Boleslav“ a „Brandýs nad Orlicí“, ale bude použit *TruncateDecorator(7)*, bude výsledkem dvakrát „Brandýs“ a dojde k porušení *UNIQUE* omezení.

- Údaj ve sloupci s *UNIQUE* omezením může nabývat hodnoty NULL. Pokud je v anonymizační třídě nastaven *NotNullDecorator* s parametrem, bude každá NULL hodnota ve sloupci anonymizována na tento parametr, například ‘N/A’, čímž dojde k porušení unikátnosti. Pokud je *NotNullDecorator* použit bez parametru a (unikátní) AF vrátí NULL hodnotu, bude použita originální hodnota údaje, která se může shodovat s ostatními anonymizovanými údaji. V obou případech bude uživatel varován, jelikož omezení může být porušeno.

- Problém s anonymizací *UNIQUE* omezení by vždy mohl nastat, pokud bude použit vzor „Update zdrojových dat“ (viz 1.3.6). I unikátní AF by se pak vždy mohla trefit do hodnoty, která se ve sloupci již vyskytuje a pouze nebyla anonymizována (kvůli použití WHERE klauzule). Tento problém platí obecně pro jakékoli *UNIQUE* omezení nad libovolným počtem sloupců (i jedním). Toto nemá jednoduché řešení, proto bude uživatel vždy upozorněn. Při užití vzoru *Create as New* se anonymizovaná data přenášejí do nově vytvořené prázdné tabulky, kdy tedy tento problém s částí neanonymizovaných dat nastat nemůže.

2.2.1.2 Primární klíč

Nad tabulkou je definován *PRIMARY KEY*. Primární klíč je definován pro jeden nebo více sloupců.

- Integritní omezení *PRIMARY KEY* musí být v tabulce definováno pouze jednou. Pokud by jich v jedné tabulce schématu bylo nastaveno více, bude uživatel varován.

- Primární klíč musí být unikátní identifikátor tabulky. Toto pravidlo bylo již výše rozepsáno u unikátního indexu (2.2.1.1).

- Sloupce v PK musí splňovat, že neobsahují NULL hodnoty. Pro každý sloupec v PK, na kterém je nastavena anonymizační třída, se zkontroluje, zda je použita funkce, která nemůže vracet NULL hodnoty. Pokud AF může vrátit NULL hodnotu, bude uživatel varován. Zde bohužel

nelze problém řešit přidáním *NotNullDecorator*, jelikož tím by se mohla porušit unikátnost údajů (opět viz 2.2.1.1, platí i pro dekorátor TD). Možným řešením je upravit anonymizační třídu a vybrat nenulovou AF.

2.2.1.3 Cizí klíč

V tabulce je definován *FOREIGN KEY*. Cizí klíč může být definován na jednom nebo více sloupcích a může jich být v tabulce nula, jedna i více. Cizí klíč může obsahovat NULL hodnoty a hodnoty nemusí být unikátní (např. zákazník může mít vypůjčeno jednu i více knih z knihovny najednou – neunikátní – a kniha nemusí být vypůjčena žádnému zákazníkovi – NULL). Aby byl vztah mezi sloupci cizího klíče v podřazené tabulce a sloupci v nadřazené tabulce zachován, musí společné sloupce v obou tabulkách být stejného datového typu, stejné délky a spojení mezi záznamy v tabulkách musí být jednoznačné, neboli sloupce z nadřazené tabulky musí být primárním klíčem.

- Sloupec z podřazené tabulky musí mít nastaven stejný datový typ a jeho délku jako příslušný sloupec z nadřazené tabulky. Pokud se typ liší, bude uživatel varován, že je chyba v nastavení databázového schématu.
- Pokud je u sloupce v cizím klíči nastavena anonymizační třída a bude u něj tedy probíhat anonymizace, musí být stejná anonymizační třída nastavená i u příslušného sloupce z nadřazené tabulky, viz tabulka 2.2. Anonymizační funkce v této anonymizační třídě musí být konzistentní, tedy pokud na vstupu dostane stejnou hodnotu, vždy vrátí na výstupu také shodnou hodnotu. Toto je potřeba, aby u obou sloupců v nadřazené a podřazené tabulce byla původní hodnota převedena na stejnou hodnotu a tím byla zachována referenční integrita dat tabulek. Pokud funkce není konzistentní, bude uživatel upozorněn.
- AF použitá na sloupec cizího klíče nemusí být unikátní a může vracet NULL hodnoty, protože to u cizího klíče nevadí. Jelikož ale u sloupce musí být použita stejná anonymizační třída jako u sloupce v nadřazené tabulce, který je v omezení primárního klíče, budou kontrolována i tyto pravidla (to se ale provádí v nadřízené tabulce s primárním klíčem)

Tabulka 2.2 Kombinace AF u sloupců podřazené a nadřazené tabulky

FK sloupec	PK sloupec	Vyhodnoceni
Neanonymizován	Neanonymizován	OK
Anonymizační třída 1	Anonymizační třída 1	OK, kontrola konzistence
Anonymizační třída 1	Anonymizační třída 2	Chyba
Anonymizační třída 1	Neanonymizován	Chyba
Neanonymizován	Anonymizační třída 1	Chyba

2.2.2 Pravidla na úrovní sloupce

Na sloupci může být definováno integritní omezení *CHECK* a *NOT NULL*. IO *CHECK* slouží k omezení hodnost určitého sloupce tabulky. Zajišťuje, že hodnota atributu leží vždy v korektním oboru hodnot, který lze kontrolovat pomocí *CHECK* klauzule. Interpretace predikátu omezení by byla poměrně složitá, bude ale zmíněna v budoucím rozvoji.

V této práci bude zpracováno omezení *NOT NULL* a vlastnost sloupce *autoincrement*.

2.2.2.1 *NOT NULL*

Sloupec má definované *NOT NULL* omezení. Sloupec musí splňovat, že nebude obsahovat NULL hodnoty. Zkontroluje se, zda je použita použita anonymizační třída s AF, která nemůže vracet NULL hodnoty, anebo jestli je přidán *NotNullDecorator*. Pokud funkce může vrátit NULL

hodnotu a není v anonymizační třídě nastaven *NotNullDecorator* (s nebo bez parametru), bude uživatel upozorněn. Možným řešením je upravit anonymizační třídu, tedy buďto přidat *NotNull-Decorator* nebo vybrat jinou nenulovou AF.

2.2.2.2 *Autoincrement*

Ve sloupci s příznakem *autoincrement* (tedy s automatickým číslováním) se hodnoty vytvářejí automaticky a normálně do nich nelze zapisovat. Takový sloupec nemá vůbec smysl anonymizovat. Pokud má tento sloupec nastavenou anonymizační třídu (jakoukoli), bude uživatel varován.

2.3 Pravidla, kdy nesouhlasí data v databázi

V této části budou navržena pravidla pro situaci, kdy data v databázi nesouhlasí s nastaveným anonymizačním modelem. Opět budou pravidla rozdělena pro tabulku a sloupce.

2.3.1 Datová pravidla na úrovní tabulky

U tabulky proběhne kontrola, zda je v pořádku připojení k databázi a přístup k datům.

2.3.1.1 Data v tabulce

Bude zkontrolováno, zda je zajištěn přístup k databázi a že databáze obsahuje data k anonymizaci. Pokud je nastavená anonymizace pro tabulku, kde by nedošlo k úpravě žádných dat, bude uživatel varován.

2.3.1.2 WHERE klauzule nad tabulkou

Nad tabulkou může být definována WHERE klauzule (datový řez). Klauzule bude použita na připojenou databázi a zjistí se počet údajů, které podmínce odpovídají. Pokud by se kvůli WHERE klauzuli neanonymizovala žádná data, bude uživatel upozorněn. Anonymizovat prázdnou kolekci dat je zbytečné a klauzule pravděpodobně není nastavena správně.

2.3.2 Datová pravidla na úrovni sloupce

2.3.2.1 Odpovídající data

Pomocí Winch funkcionality discovery by mělo být zkontrolováno, že i konkrétní data v tabulce odpovídají očekávaným údajům pro použitou anonymizační funkci. Proces discovery zkoumá jak strukturu sloupce (např. typické názvy sloupců, klíčová slova, minimální délka, …), tak prochází i konkrétní údaje a kontroluje jejich formát (např. ve sloupci kde by měly být telefonní čísla by se měly vyskytovat číslovky, ve sloupci s křestními jmény by se neměly vyskytovat čísla, email by měl obsahovat znaky jako je zavináč, atd.) a používá dostupné slovníky a porovnává, zda se hodnoty sloupce ve slovníku nacházejí (např. ve sloupci, kde by měli být křestní jména by se hodnoty sloupce měly nacházet ve slovníku s nejčastěji se vyskytujícími jmény). Anonymizační funkce by měla mít vazbu na *Discoverer*, který dokáže objevit odpovídající data pro AF, a měla by se použít pouze kontrola na úrovni dat a jejich formátu. Pro validaci není vhodné kontrolovat použití AF na základě struktury sloupce (jeho typického názvu, atd.). Např. pokud discovery proces zkoumá, zda se ve sloupci mohou vyskytovat křestní jména, bere v potaz i to, zda má sloupec název „JMÉNO“ nebo „FIRST_NAME“. Při validaci dat pro AF *FunctionFirstName*, kdy uživatel použil funkci na sloupec, který nemá tento typická název, to ale neznamená, že je anonymizace nastavená špatně.

2.3.2.2 Datový typ

Pokud sloupec nemá nastavený datový typ, který je typický pro anonymizační funkci, bude uživatel upozorněn. Například pro AF *FunctionFirstName*, která anonymizuje křestní jména, by měl být datový typ sloupce textový (př. „VARCHAR“, dle konkrétní databáze), pro *FunctionTelNumber* může být typ textový nebo numerický. Uživatel bude pobídnut ke kontrole nastavení a přiřazení anonymizační třídy sloupci.

2.3.2.3 Délka datového typu

Pokud má sloupec nastavenou anonymizační funkci, která může vrátit výstupní údaj delší, než je délka datového typu, měl by uživatel být také varován. Například, pokud funkce *FunctionFirstName* používá k anonymizaci slovník s křestními jmény, pak je maximální délka nejdelší slovo ze slovníku. Pokud má nejdelší křestní jméno ve slovníku např. 10 znaků, měla by být délka textového datového typu také alespoň 10 znaků. Měl by být brán v potaz i druhý parametr, pokud je pro AF zadefinován.

Možným řešením tohoto problému by pro uživatele bylo použít *TruncateDecorator*, který zkrátí délku výstupu AF na požadovanou. Kombinace nastavení délek a řešení jsou uvedeny v tabulce 2.3.

Tabulka 2.3 Řešení maximální délky výstupního údaje a délky datového typy

Nejdelší návratová délka AF	*TruncateDecorator*	Délka sloupce	Vyhodnocení	Komentář
Libovolná	Použit bez parametru	Libovolná	OK	TD bez parametru vrací řetězec o délce původního slova, které se do sloupce vždy vejde
Dlouhá (např. 20)	Zkracující (např. 5)	Delší nebo stejná (např. 5)	OK	TD dostatečně zkrátí výstup
Dlouhá (např. 20)	Zkracující (např. 10)	Menší (např. 5)	Chyba	TD nestačí, upravit již použitý TD
Krátká (např. 5)	Delší (např. 20)	Delší nebo stejná (např.10)	Upozornění	Mít nastavený parametr TD, který je větší než možná návratová délka AF nemá smysl.
Krátká (např. 10)	Delší (např. 20)	Menší (např. 5)	Chyba	TD nestačí, upravit již použitý TD
Délka (např. 10)	Nepoužit	Delší nebo stejná (např. 10)	OK	
Délka (např. 10)	Nepoužit	Menší (např. 5)	Chyba	Použít TD, nebo upravit datový typ

2.3.2.4 WHERE klauzule nad sloupcem

Nad sloupcem může být také definována WHERE klauzule. Pokud je definována klauzule nad sloupcem i nad tabulkou, budou podmínky spojeny pomocí predikátu AND. Výsledná klauzule bude použita na data v databázi, a zjistí se počet řádků splňujících tuto podmínku. Pokud v databázi data existují a aplikace k nim má přístup, měl by i výsledek vrátit několik řádků. Pokud by se zjistilo, že se použitím výsledné klauzule nebudou anonymizovat žádná data (žádné údaje nesplňují podmínku), měl by být uživatel upozorněn.

Kapitola 3
Implementace

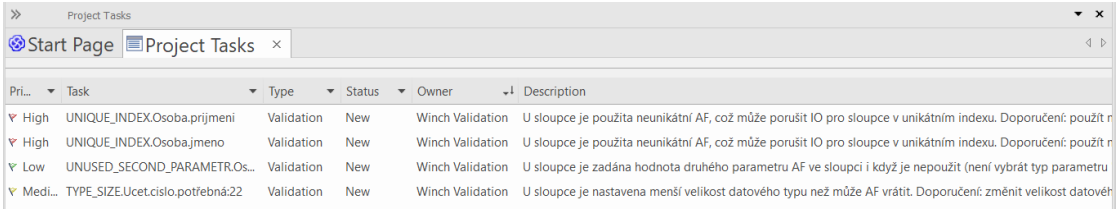
V této sekci následuje popis veškeré implementace, která proběhla kvůli rozšíření aplikace GEM Winch o funkcionalitu validace anonymizačního modelu. Rozšíření bylo implementováno do konzolové aplikace Winch Actor v programovacím jazyce Groovy. Vznikly dva nové spustitelné režimy aplikace:

- Režim *WExecutionModeValidate*, který lze spustit argumentem „-v“, spustí proces validace anonymizačního modelu.

  Pro každou tabulku spustí validační krok (*ValidateTableStep*). V kroku se získá příslušná třída dědící z *AbstractValidationTableManager* (např. MssqlValidationTableManager). Manager poté řídí validaci tabulky. Na tabulku postupně spustí vybranou kolekci validátorů (třídy dědící ze třídy *Validator*).

  Pokud validátor vyhodnotí, že bylo porušeno pravidlo, které kontroluje, zapíše varování (*ValidationWarning*) jako nový úkol (*Task*) do instance třídy *TaskSingleton*.

  Po zpracování všech tabulek se následně v *WExecutionModeValidate* zkontroluje list úkolů *Task* v třídě *TaskSingleton* a uloží se do modelu. Výsledkem režimu validace je tedy seznam úkolů zapsaný v anonymizačním modelu, viz obrázek 3.1.



Obrázek 3.1 Úkoly procesu validace zapsané v EA

- Režim *WExecutionModeValidationWarnings*, který se spustí s argumentem „-warn“, poskytne seznam (ve formátu WIKI nebo JSON) všech možných varování (*ValidationWarning*), které může režim validace vypsat. Třída, která může vypsat varování musí implementovat rozhraní *ValidationWarningWritable* a držet u sebe přístup k seznamu varování. Varování obsahuje informaci o tom, která třída ho může vypsat, název varovací hlášky, její popis, návrh možného řešení a prioritu, s jakou varování řešit.

  Výsledkem spuštění režimu je tedy přehled všech možných varování v požadovaném formátu, ukázka výstupu je vidět v tabulce 3.1 a všechna varování jsou v příloze A.

Tabulka 3.1 Ukázka možných varování

Třída	Jméno	Zpráva	Priorita	Možné řešení
DecoratorGetter	DECORATOR_PARAMETR	Parametr zadaný pro dekorátor nelze u tohoto dekorátoru použít.	LOW	zkontolovat parametr dekorátotu, např. TruncateDecoratoru musíte zadat číslo
NotNullValidator	NOTNULL	U sloupce může být porušeno omezení NOTNULL kvůli použití AF, která může vracet NULL hodnoty.	HIGH	přidat NotNullDecorator nebo vybrat nenulovou AF
UniqueValidator	UNIQUE_INDEX	U sloupce je použita neunikátní AF, což může porušit IO pro sloupce v unikátním indexu.	HIGH	použít na sloupce v unikátním indexu pouze unikátní AF
FKValidator	FK_CONSTISTENT	U sloupce může být porušena referenční integrita mezi FK a PK tabulek, kvůli použití nekonzistentní AF.	MEDIUM	použít k anonymizaci PK i FK konzistentní AF
WhereClauseTableValidator	WHERE_CLAUSE	U tabulky je nastavena where klauzule, která způsobí, že se neanonymizují žádná data.	HIGH	upravit where clause

Kromě režimů pro validaci bylo potřeba rozšířit již existující třídy aplikace jako například anonymizační funkce, slovníky, třídy discovery procesu, atd. viz sekce 3.5.

Většina nově vzniklých tříd se nachází ve složce „com.gem.winch.validation“. Třídy budou podrobně vysvětleny v následujících sekcích.

3.1 Pomocné třídy validačního rozhraní

Pomocné třídy se nacházejí ve složce „validation.util“. Tyto třídy se využívají v obou nově implementovaných režimech. Jedná se o třídu *ValidationLevel* určující úroveň validace, *ValidationWarning* reprezentující varování k označení porušení navrženého pravidla a *ValidatedColumn*, který nese důležité informace o sloupci tabulky.

Dále se zde nachází složka s nově vytvořenými primitivními potomky třídy *Discoverer*, kteří vznikli jako doplnění discovery procesu pro AF, které nemají již implementovaný průzkumník. Jsou to průzkumníci *DateDiscoverer*, *NumberDiscoverer* a *TextDiscoverer*.

3.1.1 *ValidationLevel*

Výčtový typ *ValidationLevel* určuje úroveň validace, která má být provedena. Aktuálně existují tři úrovně validace:

SCHEMA – validace na úrovni pouze databázového schématu. Při validaci se budou používat pouze validátory, které kontrolují pouze databázové schéma bez přístupu do databáze.

DATABASE – validace na úrovni databáze. Při validaci se budou používat pouze validátory kontrolující data připojené databáze.

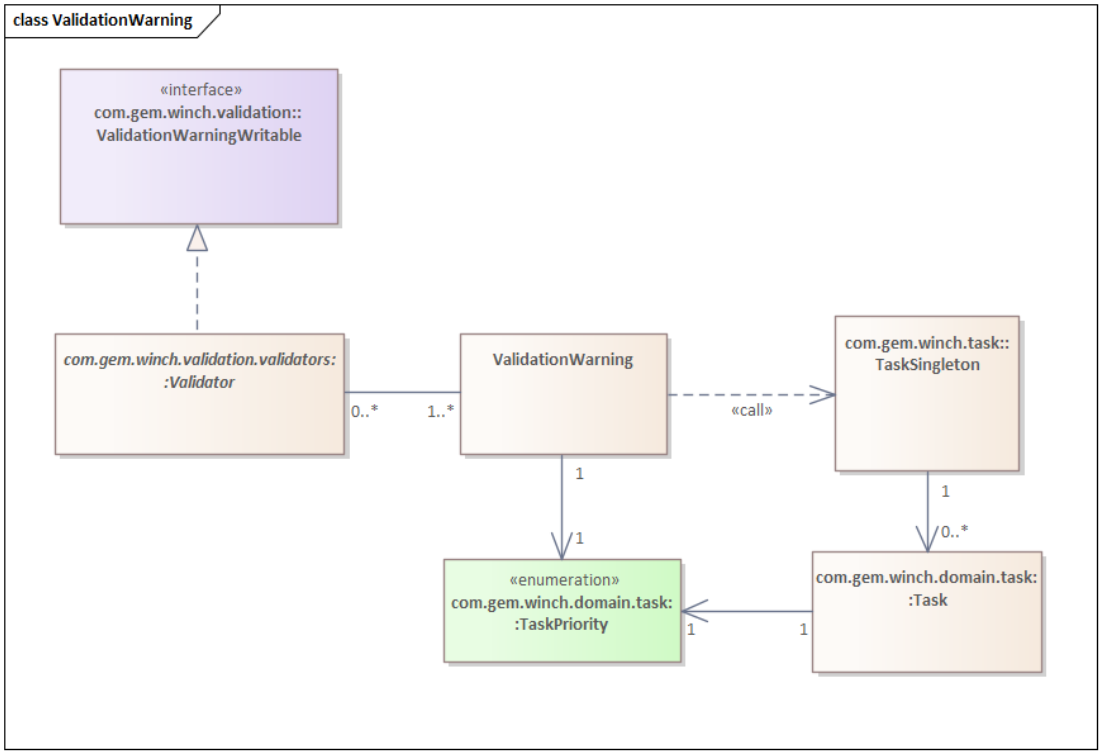
ALL – úplná validace. Pro validaci budou použity všechny implementované validátory.

Úroveň validace je v aplikaci dostupná jako jedna z vlastností *WinchContext*. Vlastnost se nazývá *winch.validationLevel*. Zadání v kontextu:

- *winch.validationLevel* = *schema_only* – odpovídá typu SCHEMA
- *winch.validationLevel* = *database_only*– odpovídá typu DATABASE
- *winch.validationLevel* = „“ – nevyplněná vlastnost nebo prázdný řetězec odpovídá typu ALL

Zároveň každý validátor dědící ze třídy *Validator* by měl mít vyplněný parametr, který určuje, jakou úroveň validace validátor provádí. Výběr validátorů na základě tohoto parametru probíhá ve třídě *ValidationSettingsGetter* (viz 3.2.2.1).

3.1.2 *ValidationWarning*



Obrázek 3.2 Rozhraní třídy *ValidationWarning*

Třída *ValidationWarning* (zobrazená na obrázku 3.2) slouží k vytvoření varování, které může vzniknout v průběhu validace. Varování obsahuje informaci o tom, která třída ho může vypsat, název varovací hlášky, její popis, návrh možného řešení a prioritu, s jakou varování řešit.

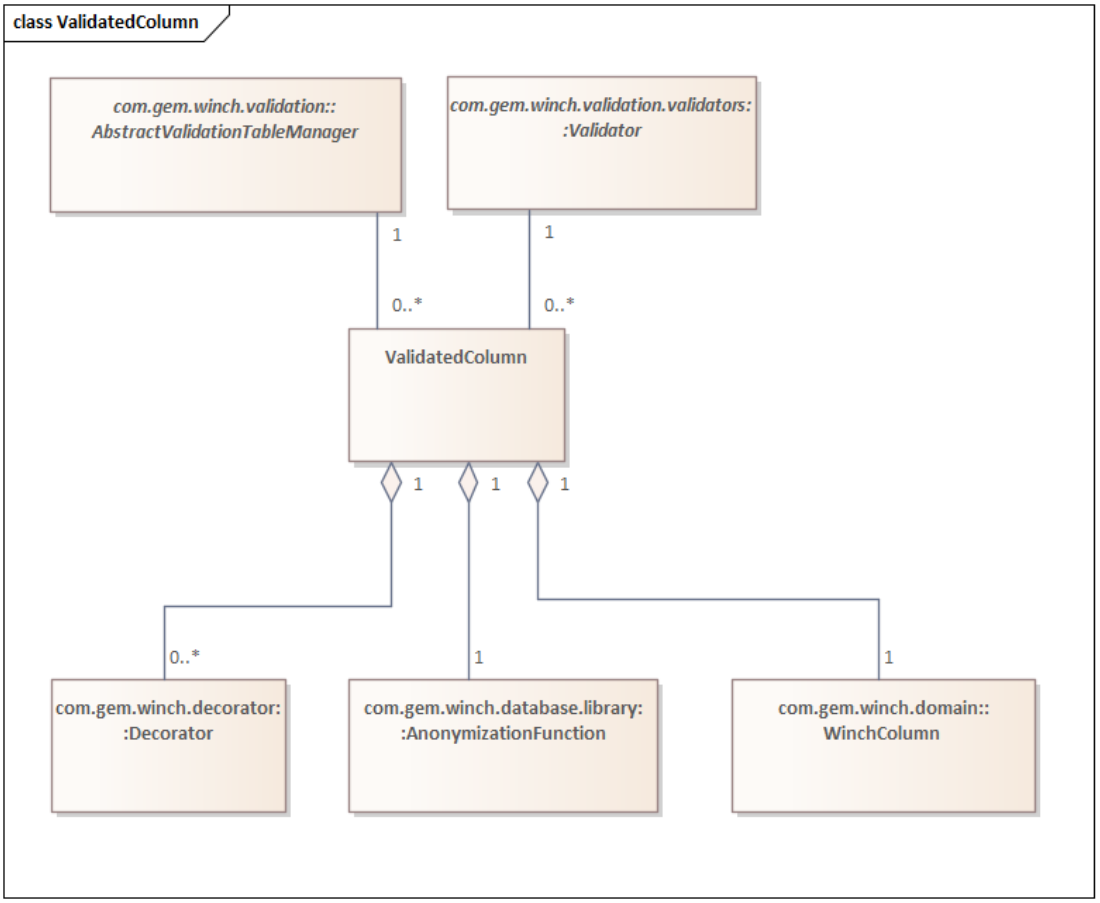
Implementuje metodu *writeWarningAsTask(String place)* (ukázka kódu 3.1), která řeší zapsání varování jako *Task* do *TaskSingleton*. Jméno úkolu je složeno z názvu varování a přidaným označením místa (parametr place), kde varování vzniklo (tabulka, sloupec, anonymizační třída, …). Do poznámek úkolu (parametr notes) se uloží zpráva varování a doporučené řešení. Priorita úkolu je priorita varování (implementována stejně jako typ *TaskPriority*). Typ bude nastaven na nový typ pro validaci uložený v konstantě *VALIDATOR_TASK_TYPE*. Jelikož *Task* zapisuje proces validace, jako vlastník se napíše konstanta *VALIDATOR_TASK_OWNER* (řetězec „WinchValidation“), kterou poté ale může uživatel nastavit na libovolného vlastníka.

Výpis kódu 3.1 Metoda *writeWarningAsTask* k zapsání varování

```
void writeWarningAsTask(String place) {
    Task newTask = new Task(name: name + "." + place, notes:
        message + "\čnDoporuení: " + solution, priority: priority)
    newTask.type = Constants.VALIDATOR_TASK_TYPE
    newTask.owner = Constants.VALIDATOR_TASK_OWNER
    TaskSingleton.getInstance().addTask(newTask)
}
```

Jelikož ve stávající implementaci mohl *Task* do anonymizačního modelu zapisovat pouze proces discovery (viz 1.3.7), bylo potřeba rozšířit typ úkolu, který může vzniknout. Vznikl nový typ *VALIDATION* ve výčtovém typu *TaskType* – tedy *TaskType.VALIDATION*. Typ je označen v konstantě *VALIDATOR_TASK_TYPE*. V konstantě *DEFAULT_TASK_TYPE* zůstal původní typ, který zapisoval proces discovery a to *TaskType.REQUEST*. Také bylo potřeba učinit menší úpravy ve třídě *TaskReaderWriter*, aby docházelo ke správnému ukládání úkolu s obecným typem a vlastníkem.

3.1.3 *ValidatedColumn*



Obrázek 3.3 Rozhraní třídy *ValidatedColumn*

ValidatedColumn je třída, reprezentující sloupec, který se bude validovat, jelikož je mu nastavena anonymizační třída. Její využití je uvedeno na obrázku 3.3. Nese informace o sloupci:

- původní třídu *WinchColumn*, která v aplikaci reprezentuje sloupec tabulky,
- instanci použité AF v anonymizační třídě nastavené sloupci,
- seznam potenciálně použitých parametrů AF,
- mapu obsahující informaci o dekorátorech zadaných v anonymizační třídě. Klíč mapy je třída dekorátoru a hodnotou je potenciálně zadaný parametr dekorátoru.

Třída *WinchColumn*, která reprezentuje sloupec tabulky *WinchTable*, obsahuje informace o anonymizační třídě ve formě řetězců. Anonymizační funkce použitá v anonymizační třídě je uložena jako řetězec, v seznamu dekorátorů je dekorátor zapsán jako řetězec (společně s jeho parametrem).

K rychlejšímu přístupu a využití nově napsaných metod (které nejsou statické, viz 3.5.1) je ale u validátorů potřeba využít instanci AF. Při zjišťování, zda byl použit konkrétní dekorátor, je lepší ho vyhledávat pomocí jeho třídy, ne řetězce. Zda zadaný řetězec odpovídá požadované třídě by se mělo řešit na jiném místě. Z těchto důvodu je lepší ve validátorech používat pro validované sloupce místo třídy *WinchColumn* novou třídu *ValidatedColumn*, která má již lépe předpřipravené informace o sloupci potřebné k validaci.

3.1.3.1 Získání informací anonymizační funkce

Pokud je potřeba získat u validovaného sloupce anotaci použité AF, využije se následující kód: 3.2

Výpis kódu 3.2 Získání anotace anonymizační funkce

```
AnonFunc anonFuncAnnotation = validatedColumnMap.get(nameFK).
        anonymizationFunction.class.getAnnotation(AnonFunc)
```

Poté má již *anonFuncAnnotation* přístup k jednotlivým vlastnostem AF (viz 1.3.3).

- *anonFuncAnnotation.unique()* vrací pravda/nepravda, pokud je AF unikátní nebo není,
- *anonFuncAnnotation.consistent()* vrací pravda/nepravda, pokud je AF konzistentní nebo není,
- *anonFuncAnnotation.nullable()* vrací pravda/nepravda, pokud AF může na výstupu vrátit hodnotu NULL nebo nemůže.

3.2 Režim *WExecutionModeValidate*

Režim dědí ze tříd *WExecutionMode*, které inicializují model (*WinchContext*, *WinchSchema*, jejich tabulky *WinchTable*, atd.), a spustí na model vybraný režim, v tomto případě *WExecutionModeValidate*. Pro každou tabulku ve všech zadaných schématech spustí validační krok (*ValidateTableStep*), který provádí validaci jednotlivých tabulek.

3.2.1 *ValidateTableStep*

V rámci provádění *executeInternal()* (viz 3.3) kroku *ValidateTableStep* se vytvoří *AbstractValidationTableManager* a zavolá se jeho metoda *validateTable()*, která nastaví potřebné zdroje a vybere příslušné validátory, které mají být použity, a spustí na tabulku zadanou v konstruktoru manageru vybrané validátory.

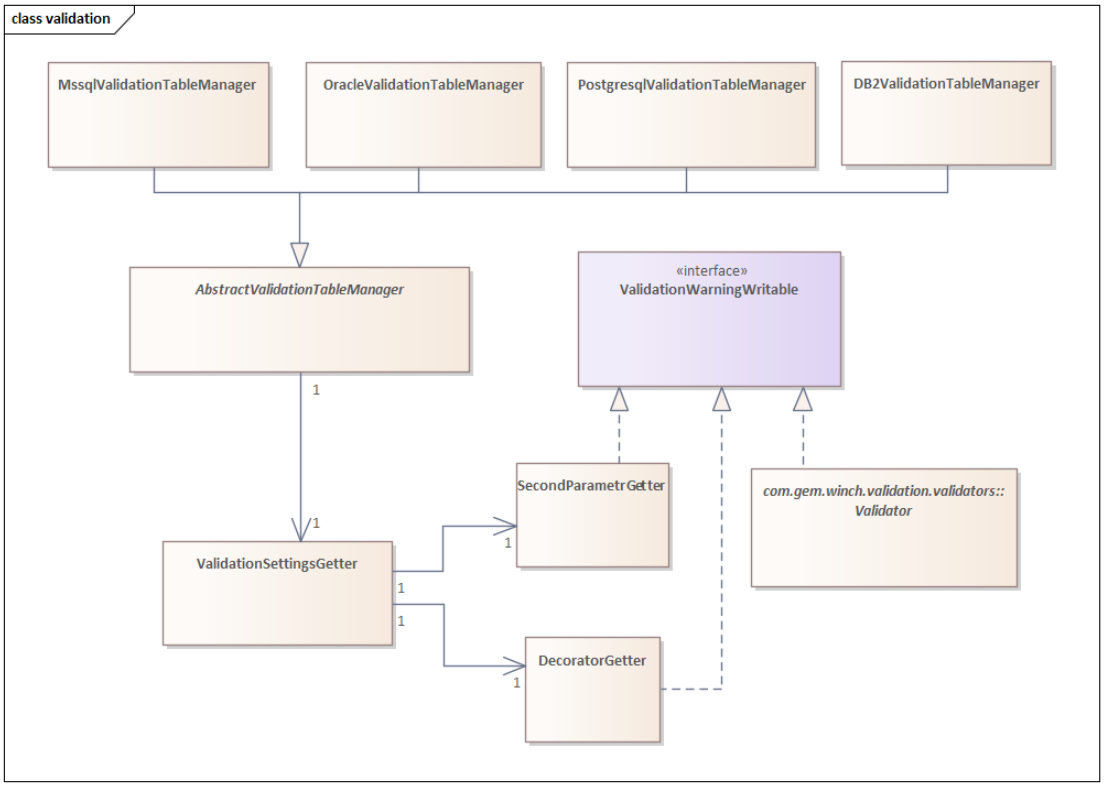
Výpis kódu 3.3 Metoda *executeInternal()* třídy *ValidateTableStep*

```
@Override
public int executeInternal() {
    AbstractValidationTableManager validationTableManager =
        databaseHelper.getValidationTableManager(table)
    validationTableManager.validateTable()
    1
}
```

3.2.2 *AbstractValidationTableManager*

Manager slouží k řízení validace jedné tabulky. Rozhraní je uvedeno na obrázku 3.4.

V konstruktoru je managerovi předána tabulka *WinchTable*, pro kterou bude validace probíhat, a seznam datových typů sloupce, které podporuje konkrétní databáze (př. MSSQL má VARCHAR, INT, NUMERIC atd. viz 3.5.5). V konkrétní implementaci třídy *AbstractValidationTableManager* (např. *MssqlValidationTableManager* nebo *OracleValidationTableManager*) by se měla v konstruktoru ještě nastavit vlastnost *DiscoveryExecutor* (executor), která bude používána ve validátorech. Např. *executor = new MssqlDiscoverExecutor()*.



Obrázek 3.4 Rozhraní třídy *AbstractValidationTableManager*

Metodou managera *getValidationLevel()* lze z kontextu dostat parametr typu *ValidationLevel* (3.1.1), který určuje, jaké validátory se budou při validaci tabulky používat. K inicializaci validátorů manager používá třídu *ValidationSettingsGetter*.

Metodou *getCountryCode()* lze z kontextu dostat parametr kód země (řetězec výčtového typu *Country*), který určuje, jaké slovníky by se při anonymizaci používaly. Aktuálně existují slovníky měst, ulic, atd. ve variantách pro Českou republiku a Slovensko (např. *winch.countrycode=sk*). Parametr může ovlivnit délku maximálního možného výstupu vráceného AF, která slovník používá. Parametr země se používá pro inicializaci validátorů, které s touto informací mohou pracovat.

Veřejnou metodou *validateTable()* se spustí validace tabulky. Pomocí instance třídy *ValidationSettingsGetter* se nastaví mapa validovaných sloupců a validátory.

Validovaný sloupec je každý, který má nastavenou anonymizační třídu. Metoda třídy *ValidationSettingsGetter* *getAnonymizationSources()* vrací mapu, která má jako klíč název sloupce pro snadné vyhledání, a jako hodnotu příslušný inicializovaný *ValidatedColumn*. Tato mapa se

používá pro inicializaci všech validátorů. Validátory vrací metoda třídy *ValidationSettingsGetter getValidators()*. Poté se pro každý vytvořený validátor zavolá metoda *validate(WinchTable table)*.

Jakmile proběhlo volání všech validátorů, tabulka by měla být plně zkontrolována.

3.2.2.1 *ValidationSettingsGetter*

Třída slouží k nastavení a současné kontrole potřebných zdrojů pro třídu *AbstractValidationTableManager*.

Metodou *getValidators()* získá seznam validátorů, které se budou při validaci používat na základě parametru *ValidationLevel*, který je metodě předán. V metodě jsou získány třídy všech validátorů ze složky *VALIDATOR_LIBRARY* (cesta je uložena v pomocné třídě *ScriptHelper*) a z neabstraktních tříd, které dědí ze třídy *Validator*, je vytvořena instance s příslušnými parametry konstruktoru. Validátor má vždy alespoň jeden parametr – mapu validovaných sloupců. Poté může mít jako parametr *DiscoveryExecutor*, kód země a seznam podporovaných datových typů.

Do seznamu, který metoda vrátí, budou přidány pouze takové validátory, které validují model podle příslušného parametru *validationLevel* – *validatorInstance.getValidationLevel() == validationLevel* anebo pokud je zadána úroveň validace *ValidationLevel.ALL*, budou vráceny všechny získané validátory.

Druhá veřejná metoda *getAnonymizationSources()* získá mapu validovaných sloupců. Jak již bylo řečeno, klíčem bude název sloupce a hodnotou odpovídající *ValidatedColumn*. Získané sloupce mají přiřazenou anonymizační třídu – budou anonymizovány – a proto budou i validovány. Tato metoda využívá dvě třídy:

1. ***DecoratorGetter*** k získání mapy přidaných validátorů. Klíčem je třída dekorátoru a hodnotou je jeho parametr (pokud dekorátor nemá definovaný parametr, bude nastavena hodnota „“ neboli prázdný řetězec). Třída *DecoratorGetter* parsuje řetězec, ve kterém je uložen název dekorátoru a jeho parametr ve formátu *[NázevDekorátoru](parameter)*. Z názvu dekorátoru získá jeho třídu. Potenciální parametr zkontroluje pomocí nově doplněné funkce třídy *Decorator validateParametr(parametr)* popsané níže v 3.5.2, která určí, zda jde parametr v konkrétním dekorátoru využit nebo ne. Pokud validace parametru vrátí neúspěch , bude vypsáno varování a hodnota mapy se nastaví na prázdný řetězec. Pokud vrátí úspěch, nastaví se hodnota mapy na zadaný parametr.

2. ***SecondParametrGetter*** k získání druhého parametru anonymizační funkce. V metodě nazvané *getAnonymizationSources()* se jako první získá třída a instance AF z jejího řetězce. Ve třídě *SecondParametrGetter* se pro sloupec a instanci AF získá seznam potenciálních druhých parametrů. Typ druhého parametru může být „None“, „Variable“, „Column“, „Constant“ (viz 1.3.6). Podle typu se budou získávat a zároveň kontrolovat parametry:

   „None“ – pokud je nastaven typ „None“, bude vrácena hodnota NULL. Může nastat výjimka, a to u AF *FunctionConst*, která na vstupu vůbec nedostává vstupní hodnotu sloupce, ale měla by dostat rovnou řetězec, který bude vracet na výstup. U ní je pak nastaven typ druhého parametru také na „None“. Řetězec bude pro validaci získáván jako druhý parametr, aby se s ním dalo dále pracovat.

   „Variable“ – druhý parametr typu „Variable“ je SQL výraz, jehož parsování a validace by byla poměrně složitá . Zde se tedy vždy bude vracet místo seznamu druhých parametrů hodnota NULL a nebude se s SQL výrazem dále pracovat. Pokud je nastaven typ Variable, ale hodnota sloupce „Parametr anonym. třídy“ není vyplněná, bude vypsáno příslušné varování, jelikož uživatel buď zapomněl hodnotu vyplnit, nebo je typ parametru nastaven zbytečně.

„Constant“ – při typu „Constant“ se seznam druhých parametrů zadává do řetězce AF. Tento řetězec bude parsován a získá se seznam parametrů. Pokud není zadán, bude vypsáno varování. Pokud list obsahuje alespoň jeden druhý parametr, zkusí se s tímto seznamem zavolat metoda AF *getMaxReturnLengthWithParameters(countryCode, parameters)*. Pokud metoda nevyvolá výjimku *IllegalArgumentException*, znamená to, že parametry jde při anonymizaci u funkce v pořádku využít. Jsou tedy zadány správně a bude vrácen seznam těchto druhých parametrů. Pokud vznikla výjimka *IllegalArgumentException*, seznam je funkci nevhodně zadán, bude na to uživatel upozorněn a vrátí se hodnota NULL (bude se pokračovat v procesu validace bez druhých parametrů).

„Column“ – pokud je zadán typ „Column“, bude jméno pomocného sloupce definované v sloupci „Parametr anonym. třídy“. Pokud se jméno sloupce nepodaří vyhledat, bude zapsáno varování a vrátí se hodnota NULL. Pokud se má validovat anonymizační model na úrovni schématu, nebudou se kontrolovat údaje sloupce a nebude vrácen seznam parametrů, vrátí se NULL.

Jinak, pokud se může validovat i databáze, budou načtena data sloupce přidanou metodou *DiscoveryExecutoru getDistinctDataSet(table, secondColumn)* (dále v sekci 3.5.4). Metoda vrátí set unikátních údajů zadaných ve sloupci (maximálně 50 údajů). Pokud metoda *getDistinctDataSet()* vrátí NULL, nepodařilo se data sloupce načíst a bude vypsáno varování. Jinak se pro každý údaj (který není databázová NULL hodnota) spustí metoda *getMaxReturnLengthWithParameters(countryCode, údaj)*. Pokud nevznikla výjimka *IllegalArgumentException*, přidá se údaj sloupce zadaného jako druhý parametr do seznamu druhých parametrů. Pokud ale výjimka vznikla, do seznamu se tento údaj nepřidá a vypíše se varování. Vrátí se seznam použitelných údajů sloupce v AF.

Pokud pro AF u jakéhokoli typu druhého parametru není definována metoda *getMaxReturnLengthWithParameters()* (metoda vrátí výjimku *NotImplementedException*), bude vrácena místo seznamu parametrů hodnota NULL, ale uživatel o tomto varován nebude.

Pokud je sloupec „Parametr anonym. třídy“ vyplněn textem a přitom není zadán typ druhého parametrů, který by tuto hodnotu využíval, bude vypsáno varování.

Obě tyto třídy – *DecoratorGetter* a *SecondParametrGetter* – implementují rozhraní *ValidationWarningWritable* (viz obrázek 3.4), jelikož mohou vypisovat *ValidationWarning* během jejich průběhu, což již bylo zmíněno u jejich popisu.

3.2.3 Zapsání úkolů *Task*

V rámci inicializační fáze *WExecutionModeValidate* se získá správného potomka abstraktního předka *TaskReaderWriter*, stejně jako se to dělá v discovery procesu.

Poté, co proběhlo zpracování všech tabulek, se v metodě třídy *WExecutionModeValidate manageNewTasks()* zpracuje zapsání úkolů do anonymizačního modelu. Získají se dva seznamy úkolů *Task*:

- původní úkoly, které jsou již v anonymizačním modelu zapsány, pomocí *TaskReaderWriter readTasks()*,
- úkoly přidané do *TaskSingleton* v průběhu validace tabulek pomocí *getInstance().getTasks()*.

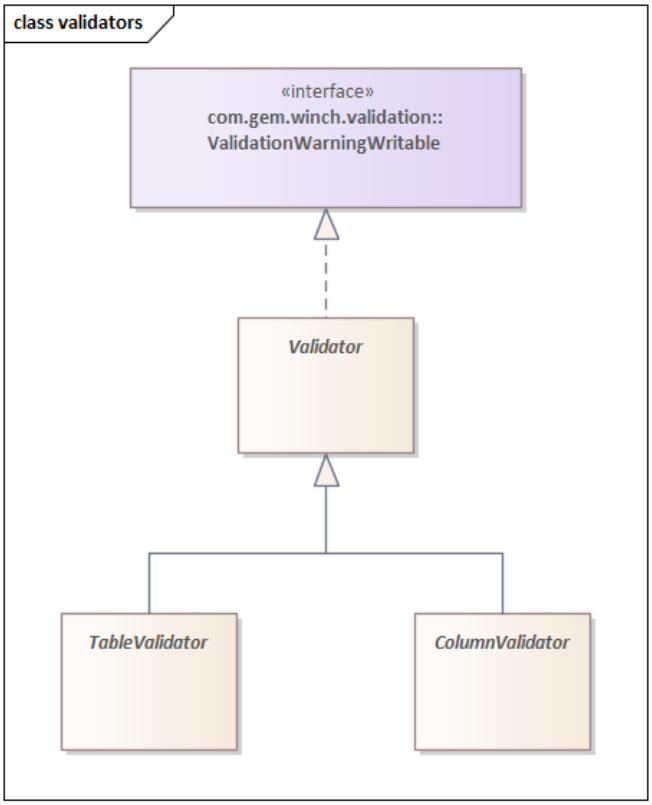
Pro každý původní *Task*, který má nastaven typ validátoru (*TaskType.VALIDATION*) se zjistí, zda stejný *Task* vznikl při procesu validace znovu. Pokud úkol při validaci už nevznikl, bude jeho stav nastaven na „splněný“, jelikož se předpokládá, že uživatel problém opravil, a proto již při validaci nevznikl (*status = Constants.COMPLETE_STATUS*). Tento původní *Task* s upraveným stavem bude znovu zapsán do *TaskSingleton*.

Po projití původních úkolů se znovu načtou všechny úkoly z *TaskSingleton* a ty se zapíší do anonymizačního modelu pomocí *TaskReaderWriter writeTasks()*. Pokud dojde k pokusu zapsat existující *Task* v modelu s upraveným stavem, bude v modelu přepsán (a tedy se mu změní stav na „splněný“).

Zapsáním vzniklých úkolů do modelu proces validace končí.

3.3 Validátory

Pro implementaci pravidel, navržených v sekci 2.2, jsou vytvořeny třídy validátorů. Pro každé pravidlo je vytvořena vlastní třída, která toto pravidlo kontroluje. Validátory jsou rozdělené do podskupin podle úrovně a druhu validace, kterou provádějí (viz validátory na obrázku 3.5).



Obrázek 3.5 Rozhraní tříd validátorů
Z tříd *TableValidator* a *ColumnValidator* dále dědí další validátory, viz následující diagramy.

Abstraktní třída *Validator* je nejobecnější třída nadřazená všem implementovaným validátorům. *Validator* má své jméno a úroveň validace, kterou provádí (zda validuje pouze schéma nebo i data databáze, viz 3.1.1). Dále má vždy vlastnost *validatedColumnMap*, která obsahuje všechny validované sloupce tabulky (*ValidatedColumn*), pro kterou byl validátor vytvořen.

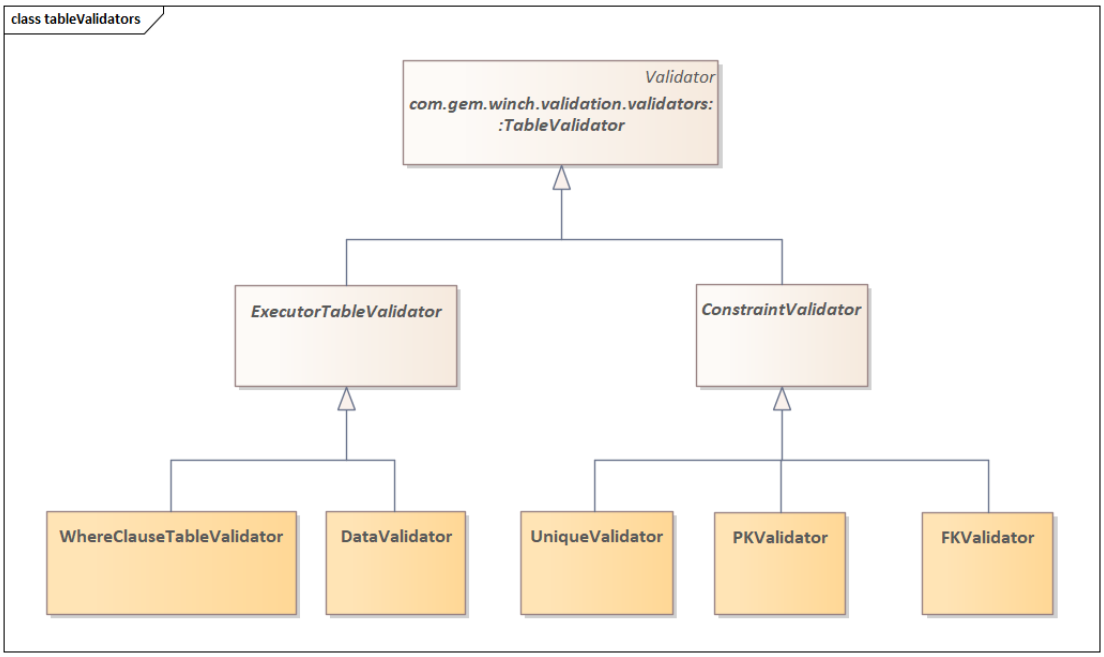
Abstraktní metodou, kterou pro každý validátor volá *AbstractValidationTableManager* je metoda *validate(WinchTable table)*, a každý validátor ji musí implementovat.

Validátor implementuje rozhraní *ValidationWarningWritable*, které by měly implementovat všechny třídy, které mohou vypisovat *ValidationWarning* během validace (viz obrázek 3.5).

Základní rozdělení validátorů určuje, zda validují pravidla nad celou tabulkou nebo kontrolují pravidla nad konkrétním sloupcem, který má nastavenou anonymizační třídu.

3.3.1 *TableValidator*

TableValidator validuje pravidla definovaná nad tabulkou. Dále lze validátory ještě dělit podle úrovně validace viz obrázek 3.6.



Obrázek 3.6 Potomci abstraktní třídy *TableValidator*

ExecutorTableValidator je abstraktní třída (viz obrázek 3.7), která ke kontrole potřebuje *DiscoveryExecutor* a validuje anonymizační model na úrovni dat v databázi (*ValidationLevel.DATABASE*). Potomek exekutoru implementovaný pro použitý databázový systém je předán validátoru v konstruktoru (např. *MssqlDiscoverExecutor*, *OracleDiscoverExecutor*).

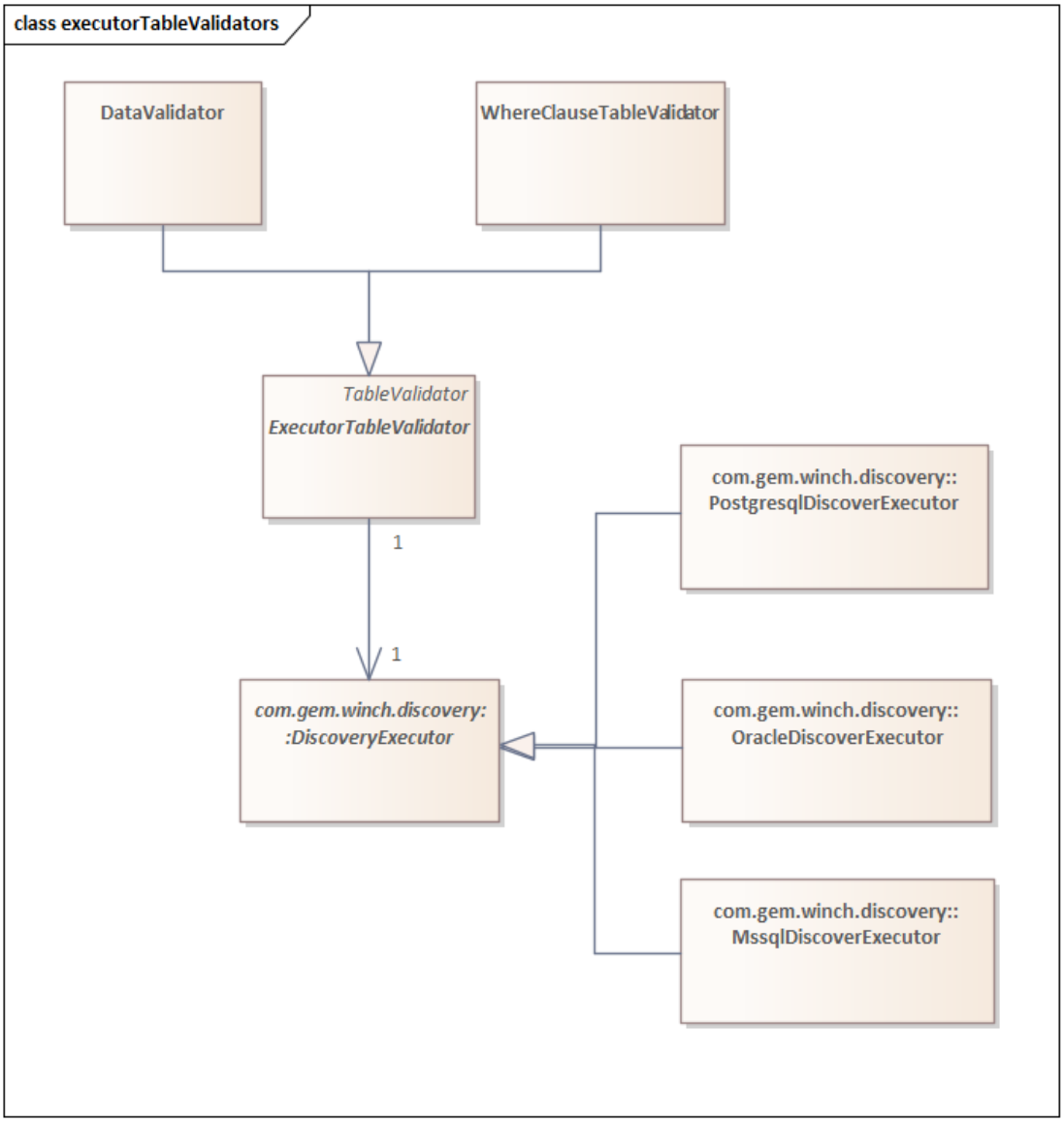
Z tohoto validátoru dědí:

1. *DataValidator*

   Tento validátor použije metodu exekutoru *getRowCount(WinchTable)*, která vrací počet údajů v tabulce. Validátor by měl být použit pouze tehdy, pokud se mají validovat data na úrovni databáze a je k nim tedy přístup (*ValidationLevel.DATABASE*). Pokud byl validátor přidán, předpokládá se, že uživatel chce validovat i databázi. Pokud metoda vrátila počet údajů 0, bude vypsáno varování, že v tabulce nejsou žádná data k anonymizaci. V případě, že se nepodařilo k databázi připojit, vrací metoda *getRowCount()* speciální hodnotu -1 a bude také vypsáno varování, že se nepovedlo data načíst.

2. WhereClauseTableValidator

   Obdobně jako *DataValidator* kontroluje počet řádku/údajů v tabulce, ale tentokrát využije i nově přidanou metodu exekutoru *getRowCountWhere(table, whereClause)*, jejíž implementace je popsaná dále v 3.5.4. Metoda prací počet záznamů vyhovující zadané WHERE klauzuli. Pokud nad tabulkou žádná WHERE klauzule zadaná není, validátor nic nevaliduje. Jinak validátor metodu zavolá s parametry: *table* – tabulka, kterou validátor validuje, a *whereClause* – klauzule zadaná nad tabulkou. Pokud původně v databázi nějaká data byla (metoda *getRowCount* vrátila číslo vetší než 0), ale kvůli použití WHERE klauzule by se žádná neanonymizovala, vypíše validátor upozornění.

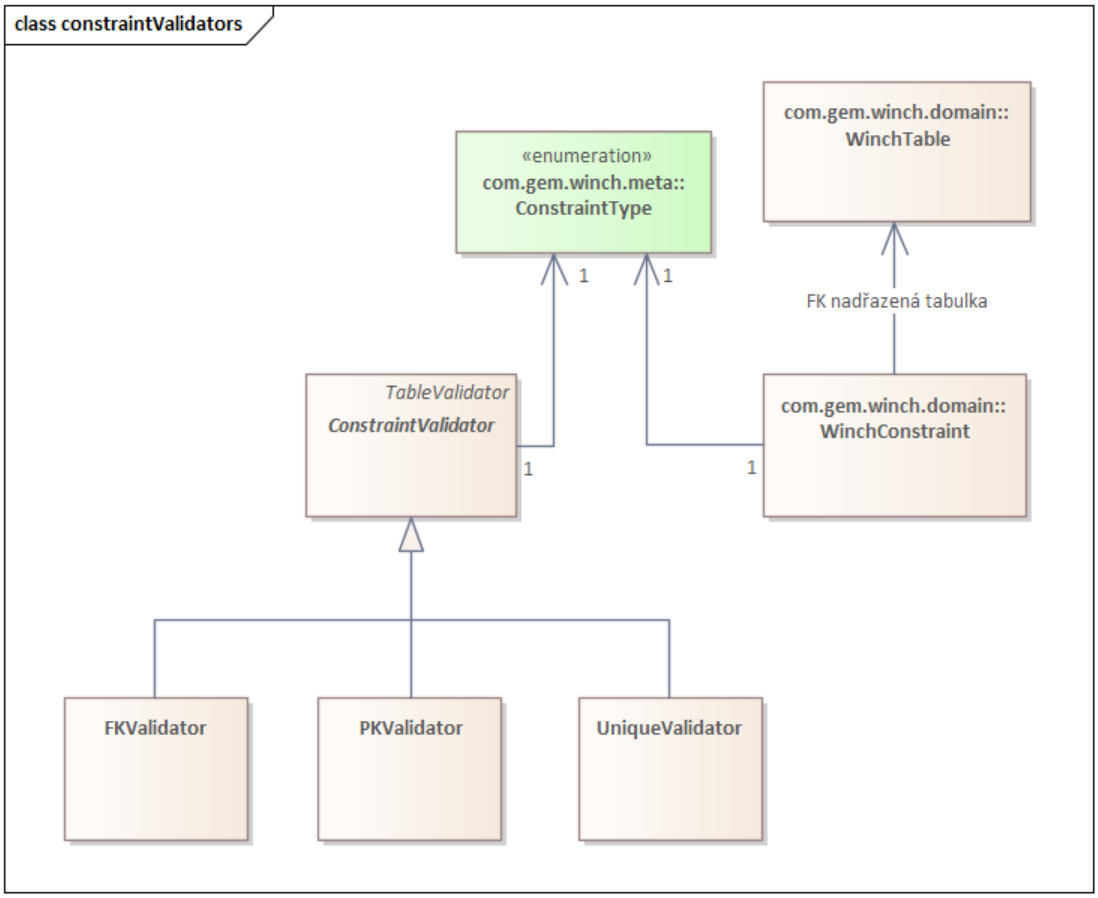


Obrázek 3.7 Rozhraní abstraktní třídy *ExecutorTableValidator*

ConstraintValidator je abstraktní validátor, který validuje vždy jeden konkrétní typ integritního omezení (*ConstraintType*). Pracuje tedy na úrovni schématu databáze (*ValidationLevel.SCHEMA*). Rozhraní validátoru je zobrazeno na obrázku: 3.8.

ConstraintType je výčtový typ, který obsahuje všechny IO řešené v aplikaci Winch. Jde o typy: *PK, unique, index, FK, check, trigger, unknown.* Implementovány jsou validátory kontrolující IO podle navržených pravidel 2.2.1 – PK, FK a unique.

Třída implementuje metodu *validate(WinchTable table)*. Pro každý *WinchConstraint* definovaný nad tabulkou *WinchTable.*, který se shoduje s typem IO v režii validátoru, zavolá metodu *validateConstraint()*.



Obrázek 3.8 Rozhraní abstraktní třídy *ConstraintValidator*

1. FKValidator

   Validátor kontroluje integritní omezení *FOREIGN KEY*, tedy *ConstraintType.FK*.

   Třída *WinchConstraint* obsahuje atribut *fkTargetTable*, který v případě, že se jedná o FK obsahuje nadřazenou tabulku k cizímu klíči (druhá *WinchTable*). Třída *WinchConstraint* také obsahuje metodu *getTargetTableColumns()*, která najde jména odpovídajících sloupců z nadřazené tabulky (sloupců, které jsou v IO typu *ConstraintType.PK*).

   Pro každou dvojici sloupce podřízené tabulky a příslušného sloupce z nadřazené tabulky se provede kontrola. Sloupce nemusejí mít přiřazené anonymizační třídy (nemusí být anonymizovány), proto je nelze najít v mapě *validatedColumnMap* (jako se to dělá u většiny ostatních validátorů). Oba listy sloupců se procházejí pomocí iterátoru.

U sloupců se kontroluje, zda mají definovaný stejný datový typ, tedy že se u obou sloupců rovnají všechny parametry: název typu (*WinchColumn.typeName*), délku textového datového typu (*WinchColumn.typeLength*), počet míst u číselného datového typu (*WinchColumn.precision*) a počet desetinných míst u číselného typu (*WinchColumn.scale*). Pokud se nerovnají, bude zapsáno varování.

Dále se validuje, že je u obou sloupců nastavená stejná anonymizační třída, jak je popsáno v návrhu (stejný *WinchAnonymizationClass.guid*). Pokud se nerovnají, bude zapsáno varování.

Nakonec se kontroluje, že (pokud má nastavenou anonymizační třídu) nastavená AF u sloupce z podřazené tabulky je konzistentní (*anonFuncAnnotation.consistent()*, vysvětlono v 3.1.3.1). Pokud není AF konzistentní, bude vypsáno varování.

2. *PKValidator*

   Validátor kontroluje integritní omezení *PRIMARY KEY*, tedy *ConstraintType.PK*.

   Pro každý sloupec v IO se zkontroluje, zda AF může na výstupu vrátit hodnotu NULL (*anonFuncAnnotation.nullable()* je pravda). Pokud může vrátit NULL hodnotu, bude hned vypsáno varování, že může dojít k porušení vlastnosti PK. Pokud je u validovaného sloupce navíc definována třída *NotNullDecorator*, bude vypsáno další varování.

   Poté se kontroluje, zda bude dodržena unikátnost údajů PK. Pokud je použita AF, která není unikátní anebo je definován dekorátor *TruncateDecorator*, bude vypsáno varování.

3. *UniqueValidator*

   Ve validátoru se kontroluje omezení *UNIQUE*, tedy *ConstraintType.unique*.

   Z anotace použité AF se opět zjistí, zda je funkce unikátní nebo není. Pokud není unikátní, zapíše se varování jako *Task*. Také se zkontroluje, zda je u validovaného sloupce nastaven dekorátor, který může porušit unikátnost údajů, tedy buďto třída *NotNullDecorator* nebo *TruncateDecorator*. Pokud je použit, vypíše varování.

   Z návrhu pravidel vyplynulo, že pokud je u tabulky nastaven vzor měnící zdrojová data („Update Source Data“), nepůjde zaručit dodržení unikátnosti údajů, viz 1.3.6. Pokud je u validované tabulky nastaveno unikátní omezení a je nastaven tento vzor, UniqueValidator vždy vypíše varování. (V aplikaci existuje abstraktní třída *AbstractTablePattern*, která tyto vzory reprezentuje, implementují ji např. *CreateAsNewTablePattern, UpdateTablePattern, DeleteTablePattern* atd.)

3.3.2 *ColumnValidator*

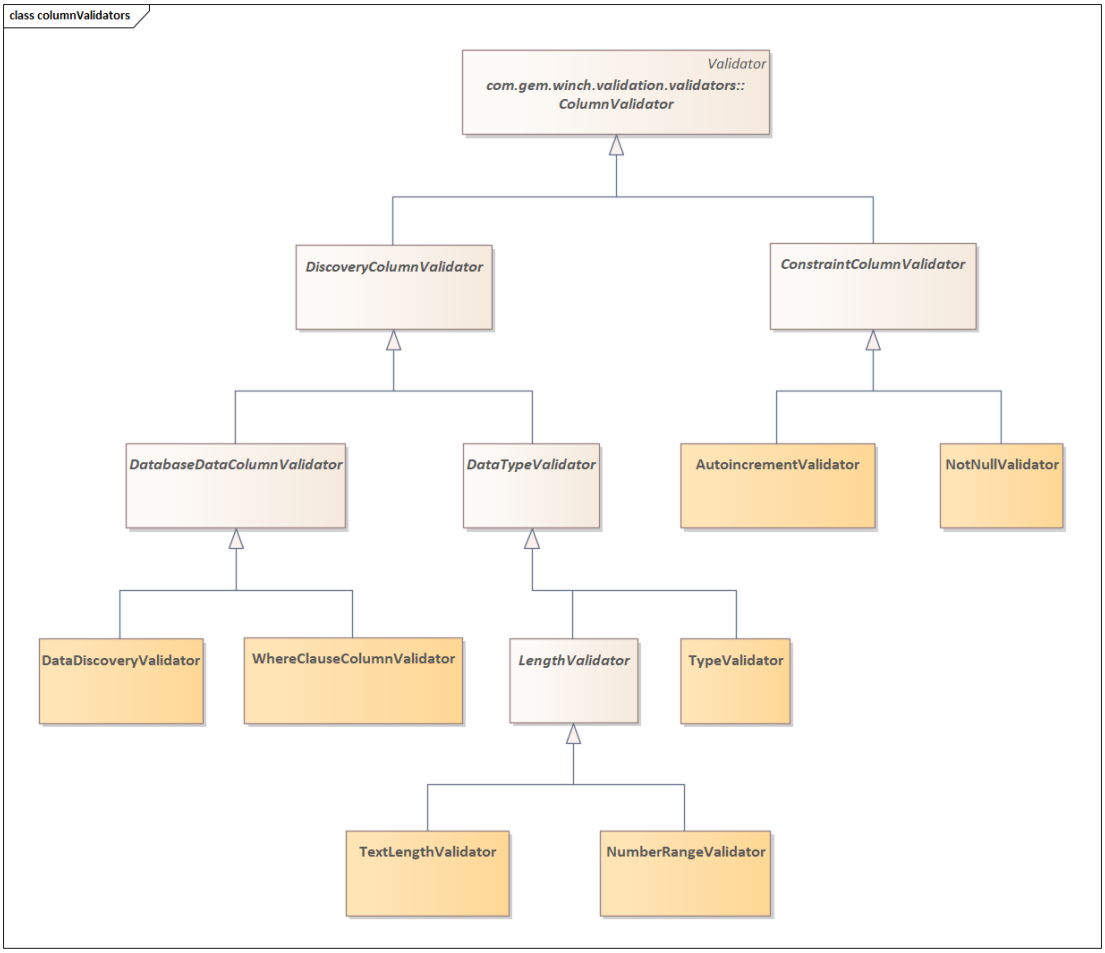
ColumnValidator validuje pravidla na úrovni sloupce. Jeho implementace a potomci jsou zachyceny na obrázku 3.9. Přepisuje metodu *validate(WinchTable table)*, kde na každý sloupec tabulky, který se hodí k validaci, spustí metodu *validateColumn(WinchTable table, WinchColumn column)*.

Metoda sloužící k získání vhodných sloupců ke kontrole *getSuitableColumns()* vrací seznam jmen všech sloupců, které jsou použité v mapě *validatedColumnMap* (keySet mapy). Jelikož jsou sloupce v mapě, znamená to, že mají nastavenou anonymizační třídu – budou anonymizovány – a je tedy potřeba zkontrolovat jejich nastavení.

ConstraintColumnValidator – validátor kontroluje nastavená databázová omezení a vlastnosti sloupce.

1. *AutoincrementValidator* v metodě *validateColumn* kontroluje, zda je u sloupce nastaven příznak autoincrementu (*WinchColumn.autoincrement*). Jelikož se metoda volá pouze pro sloupce, které mají nastavenou anonymizační třídu, pokud se zde bude validovat sloupec s příznakem autoincrementu, bude vypsáno varování.

2. *NotNullValidator* kontroluje, zda nedojde k porušení potenciálně nastavené vlastnosti, že sloupec nemůže obsahovat NULL hodnoty. Pokud má sloupec nastaven příznak *NOT NULL* (*WinchColumn.getNotNull()*) nebo je sloupec použit primárním klíči (*WinchColumn.getIsPrimaryKey()*), bude se kontrolovat anonymizační třída. Pokud je nastavena AF, která může vrátit NULL hodnoty (*anonFuncAnnotation.nullable()* je pravda) a zároveň u funkce není nastaven *NotNullDecorator*, bude vypsáno příslušné varování.



Obrázek 3.9 Potomci abstraktní třídy *ColumnValidator*

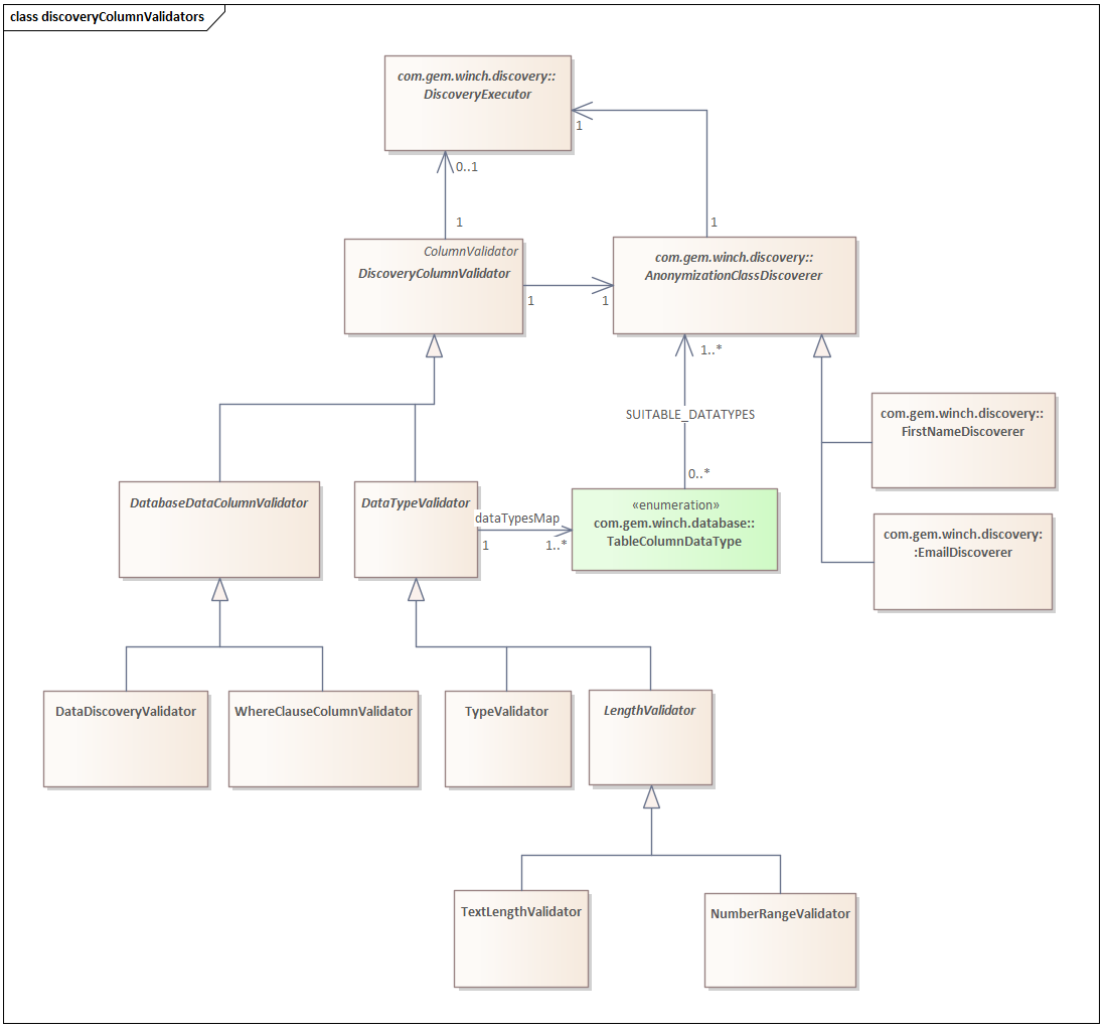
DiscoveryColumnValidator – validátory, které ke kontrole využívají průzkumník *AnonymizationClassDiscoverer*. Průzkumník potřebuje ke své inicializaci *DiscoveryExecutor* a obsahuje informace o konkrétním typu osobních údajů (např. *FirstNameDiscoverer* objevuje, zda jsou ve sloupci křestní jména). Rozhraní validátroru je rozbrazeno na obrázku 3.10.

Průzkumník se pro validovaný sloupec přiřadí v metodě *setDiscovererByAF()*. Argumentem této metody je anonymizační funkce, která byla u sloupce použitá. AF nově implementuje metodu *getAnonFunctionDiscoverer()* viz.3.5.1.2, která vrátí průzkumník, který se hodí k objevování stejných nebo velmi podobných osobních údajů (např. funkce *FunctionFirstName* bude vracet *FirstNameDiscoverer*).

Z vlastností průzkumníka se při validaci může využít vlastnost *SUITABLE_DATATYPES*, která určuje jaké datové typy se hodí pro ukládání tohoto osobního údaje (např. pro křestní

jména se hodí textový datový typ). Druh datového typu je reprezentován výčtovým typem TableColumnDataType, který aktuálně obsahuje 4 druhy typů – *NUMBER, TEXT, DATE, BINARY*. K získání vhodného druhu není potřeba využít exekutor, tato kontrola tedy probíhá na úrovni schématu databáze.

DiscoveryExecutor je potřeba pro průzkumníkovu metodu *validateColumnData()*. Metoda vypočítá pravděpodobnost, že daný sloupec obsahuje příslušný typ osobních údajů. Pro tento výpočet je potřeba pracovat s daty ve sloupci, úroveň validace je tedy databázová.



Obrázek 3.10 Rozhraní abstraktní třídy *DiscoveryColumnValidator*
Z třídy *AnonymizationClassDiscoverer* dědí řada průzkumníků, na diagramu jsou zachyceny pouze 2 příklady

DataTypeValidator – abstraktní validátor slouží k validaci datového typu sloupce. Jelikož validuje pouze datový typ sloupce, nepotřebuje k tomu zasahovat do konkrétních dat sloupce a pracuje na úrovni schématu.

K jeho inicializaci je navíc potřeba mapa datových typů podporovaných v databázi. Klíčem mapy je druh datového typu TableColumnDataType a hodnotou je seznam konkrétních jmen datových typů odpovídajícího druhu. Například textovým datovým typům v databázi MSSQL odpovídá „VARCHAR“, „NVARCHAR“, „CHAR“, atd. Při implementaci došlo k rozšíření těchto typů, viz 3.5.5.

1. *LengthValidator*

   Validátor kontroluje správně nastavenou délku datového typu. Jeho cílem je porovnat maximální možnou délku údaje na výstupu anonymizace s velikostí datového typu.

   Délka datového typu sloupce záleží na druhu datového typu. Pokud se jedná o textový datový typ (*TableColumnDataType.TEXT*), bude se délka získávat z vlastnosti *WinchColumn.typeLength*. Konkrétní implementaci pro textový datový typ řeší třída ***TextLengthValidator*** dědící ze třídy *LengthValidator*.

   Pokud se jedná o číselný datový typ (*TableColumnDataType.NUMBER*), implementace validátoru je ve třídě ***NumberRangeValidator*** a délka datové typu se získá z *WinchColumn.precision* a *WinchColumn.scale*. Pokud je délka datového typu podezřele nastavená (např. délka 0 nebo je desetinná část scale delší, než počet míst precision), bude zapsáno varování z potomka validátoru.

   Maximální možná délka údaje, která se bude muset do sloupce vejít se získá z kombinace maximální délky slova na výstupu AF a potenciálního dekorátoru *TruncateDecorator* a jeho parametru. Zkontroluje se, zda je tato délka menší než nebo stejná jako délka datového typu. Pokud je delší, slovo by se do sloupce nemuselo vejít a bude vypsáno varování.

   Délka AF se získá pomocí nové metody *getMaxReturnLengthWithParameters()*. Metodě se pošle kód země, který je v kontextu nastaven, jelikož může ovlivnit slovník, který se bude v AF používat. Druhým argumentem je seznam případného druhého parametru nastaveného u funkce. Tento seznam je nastaven v mapě validovaných sloupců. Pokud typem druhého parametru je sloupec („Column“), bude se funkce *getMaxReturnLengthWithParameters()* volat pro každý údaj seznamu druhých parametrů zvlášť. Pokud není typ parametru sloupec, bude zavolána metoda s celým seznamem najednou. Jelikož byl parametr již kontrolován ve třídě *SecondParametrGetter*, nemělo by zde správně dojít k vyvolání výjimky *IllegalArgumentException* v metodě, a bude vrácena maximální délka funkce.

   Pokud je zadán dekorátor *TruncateDecorator*, bude délka výstupu následně zkracována. Délka se získává v metodě *getTrucateDecoratorLength()*. Pokud TD není nastaven, vrátí NULL. Pokud je TD zadán bez parametru, vrátí délku 0, jelikož to znamená, že se údaj bude zkracovat na délku původního údaje, který ve sloupci již byl. Protože původní hodnota se vždy do validního nastavení sloupce vejde, nebude vypsáno varování. Jinak metoda vrátí číslo zadané v parametru dekorátoru.

   Pokud je u dekorátoru nastavena větší hodnota parametru, než je maximální možná délka slova AF, bude vypsáno varování, že je dekorátor nemá na délku slova žádný vliv a je použit zbytečně.

2. *TypeValidator*

   Kontroluje datový typ sloupce s doporučeným druhem datového typu pro osobní údaje, které jsou ve sloupci očekávány na základě výběru AF. Doporučený druh se získá pomocí průzkumníka *getSuitableDatatypes()*. Typ sloupce je uložen ve vlastnosti *WinchColumn.typeName*.

   Ve validátoru se zjistí, zda datový typ sloupce patří do doporučeného druhu pro AF. Např. sloupec má nastavený typ „VARCHAR“ a je doporučen textový datový typ. V seznamu datových typů pro *TableColumnDataType.TEXT* se najde typ „VARCHAR“ a je to tedy v pořádku. Pokud se v seznamu použitý datový typ nenajde, bude vypsáno varování.

   Navíc se ve validátoru zjišťuje, zda je u sloupce použit *NotNullDecorator*. Pokud má dekorátor nastavený parametr na nějakou konstantní hodnotu, např. „N/A“, může nastat při anonymizaci možnost, že bude potřeba do sloupce uložit tuto hodnotu. Pokud je u sloupce nastaven textový datový typ, lze do něj uložit libovolný řetězec (složený z písmen, znaků i číslic). Pokud je ale nastaven číselný datový typ, např. „NUMERIC“, musí jít parametr dekorátoru uložit jako číslo. Validátor se pokusí parametr převést na

číslo funkcí *Integer.parseInt* a pokud bude vyhozena výjimka *NumberFormatException*, znamená to, že parametr nelze převést na číslo a tím pádem uložit do sloupce, a bude vypsáno varování.

DatabaseDataColumnValidator – validátory budou využívat průzkumníka a exekutor k průzkumu dat databáze a jejich úroveň validace tedy bude databázová.

1. *WhereClauseColumnValidator*
   Pokud je u sloupce nastavena klauzule WHERE (*WinchColumn*.whereClause), bude se validovat, zda i po aplikaci klauzule budou k dispozici nenulový seznam údajů. WHERE klauzule u sloupce se nejprve spojí s potenciální klauzulí tabulky do jednoho SQL výrazu pomocí operace „AND“. Poté se zavolá metoda exekutoru *getRowCountWhere()* (3.5.4). Metoda prací počet záznamů vyhovující zadané WHERE klauzuli. Pokud původně v databázi nějaká data byla (metoda getRowCount vrátila číslo vetší než 0), ale kvůli použití složené klauzule by se žádná neanonymizovala, vypíše validátor upozornění.
2. *DataDiscoveryValidator*
   Validátor použije metodu průzkumníka *validateColumnData()* a pokud vypočtená pravděpodobnost je menší než 70 procent, bude vypsáno varování, že se ve sloupci nevyskytuje předpokládaný typ osobních údajů.

3.4 Režim *WExecutionModeValidationWarnings*

Režim slouží k vypsání všech varování *ValidationWarning*, které mohou být při procesu validace režimem *WExecutionModeValidate* vytvořeny. Slouží tedy k získání souhrnného přehledu všech existujících tříd, které které implementují rozhraní *ValidationWarningWritable* a chyb, které mohou při validaci vypsat.

Tento režim je implementován stejně jako ostatní existující režimy aplikace, které slouží k získání metadat o aplikaci (např. *WExecutionModeDecorators* slouží k výpisu všech dekorátorů).

Hlavní implementace pro tento režim proběhla v nové třídě *ValidationWarningsProvider*, kde jde metodou *getWikiLines()* získat výpis všech dostupných *ValidationWarning* ve formátu *MetadataFormat.WIKI* jako řetězec. Tabulky vytvořené z tohoto výpisu jsou ukázány v příloze A.

Metodou *getMetadata()* lze získat výpis ve formátu *MetadataFormat.JSON*. Tato metoda používá třídy *ValidationWarningLibraryDTO* a *ValidationWarningDTO*. Začátek tohoto výpisu je ukázán níže: 3.4

Výpis kódu 3.4 Příklad výpisu ve formátu *MetadataFormat.JSON*

```
{
    "warnings": [
        {
            "validatorName": "AutoincrementValidator",
            "message": "V tabulce je nastavena anonymizacni trida
               pro sloupec s priznakem autoincrement.",
            "className": "AUTOINCREMENT",
            "solution": "sloupec neanonymizovat",
            "priority": "Medium",
            "name": "AUTOINCREMENT"
        },
    //atd. ],
    "description": "Library of validation warnings",
    "name": "Winch"
}
```

3.5 Rozšíření existujících tříd

K implementaci rozšíření validátoru anonymizačního modelu bylo potřeba rozšířit již existující třídy a funkcionalitu aplikace.

3.5.1 Rozšíření anonymizačních funkcí

Do abstraktní třídy *AnonymizationFunction*, ze které dědí existující anonymizační funkce, byly přidány dvě abstraktní metody: *getMaxReturnLengthWithParameters()* a *getAnonFunctionDiscoverer()*.

3.5.1.1 Maximální délka

Výpis kódu 3.5 Metoda anonymizační funkce *getMaxReturnLengthWithParameters()*

```
abstract public int getMaxReturnLengthWithParameters(
                        String countryCode,
                        List<String> parametrs = null)
            throws IllegalArgumentException, NotImplementedException
```

Do anonymizačních funkcí byla doplněna metoda *getMaxReturnLengthWithParameters()* 3.5, která vrátí číslo, určující délku nejdelšího možného slova, které může AF vrátit na výstupu anonymizace. Požadavek na toto rozšíření AF vznikl kvůli kontrole dostatečné délky datové typu. K tomu je potřeba znát nejdelší možnou délku návratové hodnoty zvolené AF.

- U AF, které využívají slovník se bude jednat o délku nejdelšího slova ze slovníku. To může být omezeno druhým parametrem, například u AF *FunctionCityDistrict* se jako druhý parametr očekává název okresu, ve kterém se anonymizované město nachází. Pokud je funkci poslán druhý parametr např. okres „Benešov“, budou se ze slovníku vybírat pouze města z okresu Benešov a maximální délka se bude počítat pouze z těch slov. To umožnily úpravy, které byli provedeny do třídy *AbstractDictionary* 3.5.3. Pokud parametr zadán není, vrátí délku celkově nejdelšího slova ve slovníku.
- U AF, které anonymizují údaje s pevně daným formátem byly nastaveny konstanty, které metoda vrací. Konstanty byly určeny na základě analýzy 1.1.2. Například AF *FunctionIco* anonymizuje IC 3.6, což je unikátní osmimístné identifikační číslo.

Výpis kódu 3.6 Implementace metody *getMaxReturnLengthWithParameters() pro* FunctionIco

```
class FunctionIco extends AnonymizationFunction {
    private static final String DB_FUNCTION_NAME = "fc_ico"
    private static final int IC_LENGTH = 8
    FunctionIco() {
        super(DB_FUNCTION_NAME)
    }
    @Override
    int getMaxReturnLengthWithParameters(String countryCode,
       List<String> parametrs = null) {
        return IC_LENGTH
    }
//...
}
```

- AF může vracet hodnotu nezávislou na vstupní hodnotě a typu údajů ve sloupci. V tom případě určuje délku seznam druhých parametrů. Např. AF *FunctionConst* vrací řetězec zadaný jako parametr – maximální délka, kterou tedy může vrátit, je právě délka tohoto řetězce. AF *FunctionIntegerNumberRandom* vrací celé číslo z intervalu zadaného dvěma druhými parametry (min_number, max_number) – maximální délka výstupní hodnoty je tedy počet číslic nejdelšího parametru.

Přehled všech konkrétních hodnot, které mohou anonymizační funkce vracet je sepsán v příloze u seznamu funkcí B.

První parametr metody určuje zemi, která je zadaná v kontextu. Země ovlivňuje, jaký slovník se má využít (u AF které k anonymizaci používají slovník). Např. pokud je *winch.countrycode=sk*, budou se při anonymizaci měst AF *FunctionCity* používat slovenská města. Jaký se bude používat slovník ovlivňuje délku nejdelšího možného slova, které taková funkce může vrátit, jelikož v českém a slovenském slovníku budou jiné údaje.

Druhý parametr určuje seznam druhých parametrů, který je u AF zadán. Metoda *getMaxReturnLengthWithParameters()* se seznamem pracuje. Pokud je seznam druhých parametrů ve funkci nepoužitelný, bude vyvolána výjimka *IllegalArgumentException*. To nastane:

- pokud seznam obsahuje jiný počet parametrů, než který by funkce měla dostat. Např. AF *FunctionFirstName* má definovaný právě jeden druhý parametr, který určuje pohlaví,
- pokud hodnotu zadaných parametrů ve funkci nelze využít. Např. AF *FunctionFirstName* očekává, že v druhém parametru bude zadané pohlaví, podle kterého bude použit slovník buďto ženských nebo mužských jmen. Platné hodnoty druhého parametru jsou tedy „m“, „M“, „F“, atd. Pokud by v druhém parametru této AF byl např. řetězec „Praha“, je pro funkci takový parametr nepoužitelný.

3.5.1.2 Metoda určující *AnonymizationClassDiscoverer*

AF byla přidána metoda *getAnonFunctionDiscoverer()*, která vrací průzkumník dědící z třídy *AnonymizationClassDiscoverer*. AF slouží vždy k anonymizaci určitého typu osobních údajů. Anonymizační funkce by měla vracet takový průzkumník, který slouží k objevování stejného typu osobních údajů ve sloupci, jako by AF měla anonymizovat. Například *FunctionCity* anonymizuje města pomocí předdefinovaného slovníku *CityDictionary* a průzkumník *CityDiscoverer* zjišťuje, zda data sloupce obsahují názvy měst.

Seznam všech přiřazení mezi anonymizační funkcí a průzkumníkem je zaznamenán v příloze seznamu AF.

3.5.2 Rozšíření dekorátorů

Do rozhraní *Decorator*, které implementují všechny existující dekorátory, byla přidána metoda *validateParametr(String parametr)*. Metoda vrací pravdu nebo nepravdu podle toho, zda jde parametr u dekorátoru použít.

Dekorátory *UpperDecorator* a *LowerDecorator* nepřímají žádný parameter, proto bude metoda *validateParametr()* vždy vracet nepravdu.

NotNullDecorator může mít zadán parametr typu řetězec (text). V textu může být cokoli, co si uživatel přeje do tabulky zadat. Metoda tedy bude vždy vracet true. To samé platí pro *ReplaceDecorator* a *DefaultValueDecorator*.

Parametrem dekorátoru *TruncateDecorator* musí být číslo, které udává délku, na kterou se budou zkracovat výstupy anonymizační funkce. Na parametr se v metodě zavolá funkce *Integer.parseInt(parametr)*. Pokud funkce vyvolá *NumberFormatException*, znamená to, že parametr nelze převést na číslo a bude vrácena nepravda, jelikož parametr není pro dekorátor použitelný. Pokud výjimka nevznikne, vrátí metoda pravdu.

3.5.3 Rozšíření prohledávání slovníků

Byla rozšířena abstraktní třída *AbstractDictionary*, ze které dědí všechny slovníky implementované v aplikaci (např. *CityDictionary*, *FirstnameFDictionary* atd.). Při procesu validace vznikla potřeba, aby byla možnost procházet předdefinované slovníky a získávat různé informace.

Ve slovníku je vždy v prvním sloupci (*columns[0]*) identifikátor záznamu a ve druhém sloupci (*columns[1]*) je hodnota záznamu. Poté může slovník obsahovat další doplňující sloupce, které např. obsahují identifikátor jiného údaje z jiného slovníku.

Ukázka slovníku měst („list_city_dictionary.csv“) je zobrazena v tabulce 3.2 – v prvním sloupci je id města, v druhém je název města a ve třetím je id okresu, ve kterém město leží.

Tabulka 3.2 Ukázka záznamů ze slovníku „list_city_dictionary.csv“

columns[0]	columns[1]	columns[2]
1	Abertamy	23
2	Adamov	3
3	Andělská Hora	6
4	Aš	8
5	Bakov nad Jizerou	33
6	Bavorov	59
7	Bečov nad Teplou	23

První novou metodou byla *maximalWordLength()*, která vrátí délku nejdelší hodnoty ze slovníku. Používá se v ní metoda *getValueAtIndex()*, která vrací hodnotu záznamu na zadaném indexu (řádku). Hodnota se bere z druhého sloupce (*columns[1]*), kde je vždy zapsána hlavní hodnota záznamu.

Dále byla tato metoda přetížena o parametr *otherId* (*maximalWordLength(int otherId)*), kde se vrátí délka nejdelšího slova, které obsahuje ve sloupci *columns[2]* požadované id. Jinými slovy, číslo v tomto sloupci se shoduje s *otherId*. Využívá se například, pokud chci ze slovníku měst získat nejdelší název města ze všech měst, které se nacházejí v okrese s určitým id.

Třetí variantou této funkce je *maximalWordLength(List<Integer> otherIds)*, které vrací délku nejdelšího slova, které ve sloupci *columns[2]* má zadané jedno z požadovaných id v seznamu otherIds. Tedy např. nejdelší město ze všech měst, které se nacházejí ve dvou/třech/… okresech.

Metoda *maximalWordLengthFromColumn(int dictionaryColumn)* vrací délku nejdelší hodnoty ze slovníku v zadaném sloupci. Například ve slovníku zemí 3.3 je zapsáno více formátů země v jednom řádku, může být tedy potřeba získat délku slova ze sloupce jiného než *columns[1]*.

Tabulka 3.3 Ukázka záznamů ze slovníku „list_country_dictionary.csv“

columns[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
42	Černá Hora	Černá Hora	499	ME	MNE	Černohorec	Montenegro	Montenegro
43	Česká republika	Česko	203	CZ	CZE	Čech	Czech Republic	Czechia
44	Čínská lidová republika	Čína	156	CN	CHN	Číňan	Peoples Republic of China	China
45	Dánské království	Dánsko	208	DK	DNK	Dán	Kingdom of Denmark	Denmark

Jelikož byly doplněny metody, které pracují s id slova, je potřeba mít možnost zjistit id konkrétního záznamu ve slovníku. K tomu vznikla metoda *getId(String name)*, která vrátí id uložené v prvním sloupci *columns[0]* pro zadaný název slova (name).

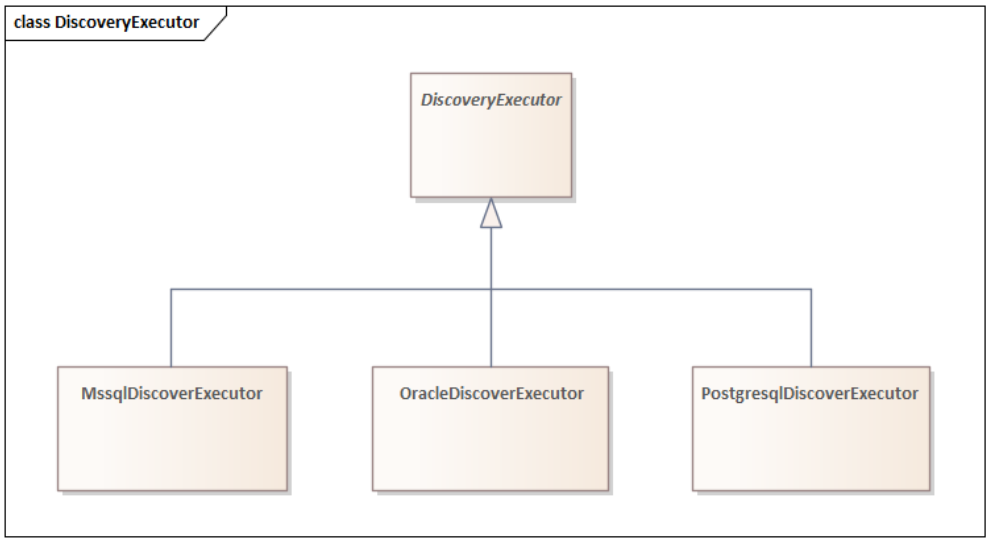
Metoda *getIds(int otherId)* vrátí id (*columns[0]*) všech záznamů, které splňují, že mají ve sloupci *columns[2]* požadované id zadané v parametru *otherId*.

Příkladem použití nových metod je například zjišťování nejdelší možné výstupní hodnoty AF *FunctionCityRegion*. AF anonymizuje město z kraje na jiné město ze stejného kraje. Druhý parameter funkce je tento kraj. Funkce tedy používá slovník měst, okresů a krajů. Ve slovníku krajů je potřeba najít identifikátor kraje, který je zadán jako druhý parameter. To se získá metodou *getId(parametr)*. Ve slovníku okresů je potřeba najít id všech okresů, které se nacházejí

v tomto kraji, tedy *getIds(id_regionu)*. Nakonec se na slovník měst spustí metoda *maximalWordLength(seznam_id_okresů)*, která vrátí hledanou délku.

3.5.4 Rozšíření třídy *DiscoveryExecutor*

Abstraktní třídě *DiscoveryExecutor* byly přidány dvě nové abstraktní metody *getRowCountWhere()* a *getDistinctDataSet()*, které byly implementovány do všech potomků této třídy pro konkrétní databázové systémy (viz obrázek 3.11).



Obrázek 3.11 Třída *DiscoveryExecutor*

Metoda *getRowCountWhere(WinchTable table, String whereClause)* vrátí počet záznamů v databázi, které vyhovují WHERE podmínce. Implementace SQL příkazu pro MSSQL, Oracle a Postgres s WHERE klauzulí vypadá následovně:

MSSQL :
```
SELECT count(*) as pocet
FROM [${table.originSchemaName}].[${table.name}]
WHERE ${getSampleSQL(table)} AND ${whereClause}
```

Oracle :
```
SELECT count(*) as pocet
FROM {table.originSchemaName}.${table.name} ${getSampleSQL(table)}
WHERE ${whereClause}
```

Postgres :
```
SELECT count(*) as pocet
FROM {table.originSchemaName}.${table.name} s ${getSampleSQL(table)}
WHERE ${whereClause}
```

Metoda *getDistinctDataSet(WinchTable table, WinchColumn column)* vrací unikátní set záznamů ze sloupce. SQL příkaz musí vracet unikátní záznamy. V případě, že je ve sloupci více unikátních údajů, vrátí prvních 50. Omezení na 50 údajů je přidáno, jelikož při práci s tabulkou, která může mít miliony záznamů, nelze riskovat načítání všech do paměti aplikace. Jelikož se

aktuálně metoda *getDistinctDataSet()* používá ke kontrole dat sloupce zadaného jako druhý parametr, který by měl většinou obsahovat sloupec, který bude mít málo unikátních údajů (např. krajů je 14, pohlaví jsou dvě), je v pořádku pracovat pouze s omezeným vzorkem dat.

Implementace SQL příkazu pro databázové systémy:

MSSQL :
```
SELECT DISTINCT TOP 50 ${column.name}
FROM [${table.originSchemaName}].[${table.name}]
```

Oracle :
```
SELECT DISTINCT ${column.name}
FROM [${table.originSchemaName}].[${table.name}]
LIMIT 50
```

Postgres :
```
SELECT DISTINCT ${column.name}
FROM [${table.originSchemaName}].[${table.name}]
LIMIT 50
```

3.5.5 Úprava třídy *DatabaseHelper* a datových typů

Byla upravena abstraktní třída *WAbstractDatabaseHelper* a její potomci. Jelikož proces validace i discovery proces ke své funkcionalitě potřebují mapu podporovaných datových typů, byla přidána metoda *getDatabaseSupportedTypes()*, která vrací definované typy konkrétního databázového systému a poskytuje tuto mapu do obou manažerů. Zároveň byly do mapy doplněné další datové typy. Nejprve byl přidán nový druh datového typu, který se bude používat k reprezentaci datumu – *TableColumnDataType.DATE*. Původní a nově přidané datové typy, se kterými lze v aplikaci pracovat, reprezentují pro databázové systémy následující tabulky – pro MSSQL 3.4, Oracle 3.5.

Tabulka 3.4 Datové typy databázového systému MSSQL

TableColumnDataType	Původní typy	Přidané typy
TEXT	„VARCHAR“, „NVARCHAR“, „CHAR“, „NCHAR“, „TEXT“, „NTEXT“	
NUMBER	„INT“, „NUMERIC“	„BIGINT“, „SMALLINT“, „TINYINT“, „BIT“, „DECIMAL“
BINARY	„BINARY“, „VARBINARY“, „IMAGE“	
DATE		„DATE“, „DATETIME“

Tabulka 3.5 Datové typy databázového systému Oracle

TableColumnDataType	Původní typy	Přidané typy
TEXT	„VARCHAR2“, „VARCHAR“, „CHAR“„NCHAR“, „NVARCHAR2“	
NUMBER	„NUMBER“	
BINARY	„RAW“, „LONG RAW“ , „BLOB“, „BFILE“, „CLOB“, „NCLOB“	
DATE		„DATE“

Kapitola 4
Testování

V této kapitole budou popsány testy, které byly implementovány pro kontrolu nových funkcí aplikace.

4.1 Unit testy procesu validace

Testy nových tříd pro rozšíření validace vznikly nezávisle na databázovém systému v adresáři „disl-winch-connector.src.test.groovy.com.gem.winch.validation“. Celkem vzniklo 95 Unit testů pro 17 tříd.

Ve složce „validation.mockData“ byly vytvořeny pomocné Mock třídy pro Unit testy. Mockování je proces, kdy není volána konkrétní instance dané třídy, ale její Mock. Mock je náhražka za reálný objekt pro získávání různých informací o volání daného objektu. Např. v tomto případě byla potřeba vzniku Mock tříd jako je *MockWinchColumn* a *MockWinchTable*, pro vytvoření testovacího prostředí pro validátory.

Pokrytí kódu ve složce „com.gem.winch.validation“ testy reprezentuje následující tabulka: 4.1

Tabulka 4.1 Pokrytí kódu „com.gem.winch.validation“

Element	Class, %	Method, %	Line, %
package util	35% (15/42)	27% (18/66)	33% (48/144)
package validators	100% (106/106)	79% (288/363)	86% (1592/1838)
AbstractValidationTableManager	0% (0/1)	0% (0/4)	0% (0/19)
DecoratorGetter	100% (1/1)	83% (5/6)	72% (36/50)
MssqlValidationTableManager	0% (0/1)	0% (0/1)	0% (0/2)
SecondParametrGetter	100% (1/1)	75% (6/8)	53% (69/128)
ValidationSettingsGetter	100% (2/2)	57% (4/7)	52% (32/61)

4.1.1 Unit testy validátorů

Pro každou nově implementovanou třídu validátoru vznikla testovací třída, která obsahuje Unit testy pro daný validátor. Testy kontrolují požadovanou funkcionalitu validátoru. Test vytvoří požadovaný příklad z Mock tříd (tabulku se sloupci, sloupce s nastavenou anonymizační třídou s Mock anonymizační funkcí). Poté se kontroluje zda, validátor vrátí požadovaný logický výsledek (pravda/nepravda) a zda vypsal do instance *TaskSingleton* správný počet nových úkolů.

Testy jsou vytvořeny pro všechny implementované validátory a testují validaci v různých scénářích:

- validní model (tabulka a sloupce), kdy nedojde k porušení pravidel

- nevalidní model, kdy dojde k vypsání varování
- kombinace více validovaných sloupců a nevalidovaných sloupců
- použití Mock anonymizačních funkcí s různými anotacemi
- kombinace s různými definovanými dekorátory v anonymizační třídě, atd.

Pokrytí kódu ve složce „com.gem.winch.validation.validators“ testy reprezentuje následující tabulka: 4.2

Tabulka 4.2 Pokrytí kódu „com.gem.winch.validation.validators“

Element	Class, %	Method, %	Line, %
ColumnValidator	100% (1/1)	100% (3/3)	100% (11/11)
TableValidator	100% (1/1)	100% (1/1)	100% (1/1)
Validator	100% (1/1)	100% (2/2)	100% (4/4)
ConstraintValidator	100% (1/1)	100% (2/2)	100% (10/10)
ExecutorTableValidator	100% (1/1)	100% (1/1)	100% (3/3)
DataValidator	100% (1/1)	75% (3/4)	88% (8/9)
WhereClauseTableValidator	100% (1/1)	75% (3/4)	91% (11/12)
UniqueValidator	100% (1/1)	80% (4/5)	96% (28/29)
PKValidator	100% (1/1)	75% (3/4)	88% (30/34)
FKValidator	100% (1/1)	75% (3/4)	93% (44/47)
DiscoveryColumnValidator	100% (1/1)	66% (2/3)	54% (6/11)
ConstraintColumnValidator	100% (1/1)	100% (1/1)	100% (2/2)
DataTypeValidator	100% (1/1)	100% (1/1)	100% (3/3)
DatabaseDataColumnValidator	100% (1/1)	100% (1/1)	100% (2/2)
TypeValidator	100% (1/1)	75% (3/4)	85% (24/28)
TextLengthValidator	100% (1/1)	80% (4/5)	81% (31/38)
NumberRangeValidator	100% (1/1)	80% (4/5)	78% (22/28)
LengthValidator	100% (1/1)	80% (4/5)	81% (26/32)
WhereClauseColumnValidator	100% (1/1)	75% (3/4)	92% (13/14)
DataDiscoveryValidator	100% (1/1)	75% (3/4)	81% (9/11)

4.1.2 Ostatní Unit testy

Také vznikly testy pro třídy *DecoratorGetterTest*, *SecondParametrGetterTest*, *ValidationSettingsGetterTest*. Tyto Unit testy ověřují jak správné vypisování varování do instance *TaskSingleton* tak správnou funkcionalitu získávání potřebných informací (např. správný počet validátorů pro úroveň validace, správný parametr dekorátoru, očekávaný seznam druhých parametrů pro AF při zadaném formátu). Testy jsou vytvořeny pro nejdůležitější metody tříd.

4.2 Test režimu *WExecutionModeValidate*

V adresáři vznikl test *ValidateApplicationTest*, který spouští celý průběh aplikace s režimem *WExecutionModeValidate*. Zavolá se test aplikace s argumentem „-v“. Test probíhá pro databázový systém MSSQL, pro který jsem při vývoji režimu validace měla připravené databázové schéma v Enterprise Architect a připojenou odpovídající databázi přes Microsoft SQL Server Express. Test kontroluje, zda spuštění proběhne a že na konci běhu je v třídě *WinchContext* doopravdy nastaven režim validace (*Constants.ExecModeValidate*).

4.3 Testy režimu *WExecutionModeValidationWarnings*

Pro test režimu vznikla třída *ValidationWarningsProviderTest*, která kontroluje, zda režim vypisuje očekávaný výstup.

4.4 Rozšíření testů anonymizačních funkcí

Zde bude popsáno testování metody *getMaxReturnLengthWithParameters()* popsané v sekci 3.5.1.1. Jelikož implementace této metody nezáleží na konkrétním databázovém systému, byly testy doplněny do složky:

„disl-winch-connector.src.test.groovy.com.gem.winch.database.anonymization“.

Byla přidána metoda *runMaxLengthTest()* do třídy *AbstractFunctionTest*. Parametrem metody je mapa testovaných případů. Klíčem mapy je očekávaná délka, kterou vrátí funkce *getMaxReturnLengthWithParameters()* po jejím zavolání s parametry definovanými hodnotou mapy. V případě, že se očekává, že metoda AF vytvoří výjimku *IllegalArgumentException*, bude očekávaná hodnota délky nastavena na speciální hodnotu -2.

V metodě *runMaxLengthTest()* je potřeba získat instanci AF, která je testována. K tomu byla vytvořena nová abstraktní metoda *getAnonymizationFunctionForTest()*, která je implementována ve všech konkrétních implementacích testů anonymizačních tříd pro databázové systémy. Testy tedy probíhají pro všechny AF všech databázových systémů (např. pro abstraktní AF FunctionFirstName test probíhá v MssqlFunctionFirstName, OracleFunctionFirstName, PostgresqlFunctionFirstName, atd.)

Jednotlivé třídy, které dědí ze třídy *AbstractFunctionTest*, poté ve svých testech volají metodu *runMaxLengthTest()*. Jednalo se tedy o doplnění Unit testů anonymizačních funkcí. Nové testy kontrolují jakou délku metoda *getMaxReturnLengthWithParameters()* vrací s různě zadanými parametry:

- bez zadaného seznamu druhých parametrů,
- s jiným počtem druhých parametrů než metoda AF očekává,
- správné průchody s kontrolou, že metoda vrátí konkrétní délku, atd.

Test s několika vytvořenými testovacími případy vznikl pro každou abstraktní anonymizační funkci v aplikaci.

Kapitola 5

Budoucí rozvoj

V této kapitole jsou sepsány všechny možnosti pro rozšíření režimu validace, které byly objeveny v průběhu tvorby bakalářské práce. Nejsou součástí implementace této práce, ale mohou být v budoucnu doplněny.

Návrhy pro budoucí rozvoj validace:

1. vytvoření nových průzkumníků. Aktuálně existuje několik průzkumníků pro určité osobní údaje, většina je zaměřena na citlivé osobní údaje. Implementovaných anonymizačních funkcí je více a anonymizují větší škálu údajů. V metodě *getAnonFunctionDiscoverer()* se aktuálně u funkcí, které již neměly jasně přiřaditelný průzkumník, vrací pomocný průzkumník, který umí kontrolovat pouze to, zda mají být ve sloupci údaje obsahující číslice či nikoli. Každá AF by mohla mít přiřazený průzkumník, který umí identifikovat příslušný druh osobního údaje podle např. nějakého vzoru,
2. validace druhého parametru AF typu „Variable“. Parametrem bude SQL příkaz, který se dá zkusit spustit a případně dál vyhodnotit,
3. kontrola zda databázový model odpovídá schématu v Enterprise Architect,
4. validace dalších pravidel. Zatím nejsou implementovány validátory pro integritní omezení jako je např. *CHECK* a *DEFAULT*,
5. individuální úprava pro uživatele. Do budoucna je možnost implementovat mechanismus umožňující přidávat a upravovat kontrolovaná pravidla specifická pro konkrétní prostředí nebo zákazníka. Například změna priorit, povolení/zakázání jednotlivých pravidel, tvorba vlastních pravidel, atd.
6. validace by mohla být použita i pro kontrolu nově navrhovaných databází s nějakými firemními standardy bez ohledu na použití anonymizačních funkcí (např. pojmenování tabulek, primárních klíčů, cizích klíčů, dodržování minimálních rozsahů pro ukládání jednotlivých typů dat, atd.).

Kapitola 6
Závěr

Bakalářská práce čtenáře seznámila s databázovým modelem, anonymizací osobních údajů a nástrojem GEM Winch. V teoretické části byly vysvětleny a popsány potřebné pojmy spojené s touto problematikou.

Na základě získaných informací byla v části návrhu řešení vytvořena pravidla popisující validační proces a všechny podmínky, které by se v něm měly kontrolovat. Pravidla se dala rozdělit do tří kategorií – na pravidla, která kontrolují databázové schéma, která kontrolují konkrétní data v databázi, a nakonec byla ještě doplněna pravidla hlídající nastavení anonymizačních tříd.

V implementační části byla popsána architektura rozšíření nástroje GEM Winch, které bylo naprogramováno. Při realizaci navržených pravidel vznikly dva nové režimy – režim, který slouží k vypsání všech varování, která může uživatel v anonymizačním modelu vytvořit, a druhý režim, který spustí proces validace modelu. V bakalářské práci došlo k implementaci validátorů, která kontrolují navržená pravidla, a pokud by mohlo dojít k porušení pravidel, je uživatel upozorněn. Celkem vzniklo 14 neabstraktních tříd kontrolujících 34 různých varovacích hlášek.

Řešení je snadno rozšiřitelné pro budoucí rozvoj aplikace a bylo otestováno pomocí navržených Unit testů.

Implementace bakalářské práce již byla integrována do současné hlavní vývojové větve projektu a je tedy začleněna do nástroje GEM Winch. Přínosem práce je funkční rozšíření nástroje GEM Winch o proces validace anonymizačního modelu, což vylepší a usnadní práci uživatelům aplikace. Navíc rozšíření obecné funkcionality může v budoucnu zjednodušit následný rozvoj aplikace.

Příloha A

Seznam možných varování

Následující tabulky A.1, A.2 a A.3 obsahují všechna možná varování, která proces validace může vypsat. Tento formát a text tabulek vznikl z výstupu režimu *WExecutionModeValidationWarnings*.

Tabulka A.1 Seznam varování ze tříd *DecoratorGetter* a *SecondParametrGetter*

Třída	Jméno	Zpráva	Priorita	Možné řešení
DecoratorGetter	UNRECOGNISED_DECORATOR	U anonymizační třídy je nastaven dekorátor, který není definován/nelze načíst.	MEDIUM	upravit seznam dekorátorů anonymizační třídy
DecoratorGetter	MULTIPLE_DECORATORS	U anonymizační třídy je nastaveno více dekorátorů stejného druhu.	LOW	nechat pouze jeden stejný dekorátor ve třídě
DecoratorGetter	DECORATOR_SYNTAX_BRACKETS	Zadaný dekorátor má špatnou syntaxi, chybí závorky.	LOW	zkontrolovat dekorátor, správný zápis např. NotNullDecorator()
DecoratorGetter	DECORATOR_PARAMETR	Parametr zadaný pro dekorátor nelze u tohoto dekorátoru použít.	LOW	zkontolovat parametr dekorátotu, např. TruncateDecoratoru musíte zadat číslo
SecondParametrGetter	UNUSED_SECOND_PARAMETR	U sloupce je zadána hodnota druhého parametru AF ve sloupci i když je nepoužit (není vybrát typ parametru ‘Column ani ‘Variable).	LOW	zkontrolovat druhý parametr AF
SecondParametrGetter	MISSING_SECOND_PARAMETR	U sloupce není zadána hodnota druhého parametru pro AF.	MEDIUM	zkontrolovat druhý parametr AF
SecondParametrGetter	WRONG_SECOND_PARAMETR	U sloupce je zadána hodnota druhého parametru AF, která nelze v AF využít.	MEDIUM	zkontrolovat druhý parametr AF
SecondParametrGetter	COLUMN_DATA_SECOND_PARAMETR	Sloupec zadaný v druhém parametru pro AF neobsahuje žádná data nebo je nelze načíst.	MEDIUM	zkontrolovat druhý parametr AF / zkontrolovat připojení k databázi.
SecondParametrGetter	COLUMN_SECOND_PARAMETR_NOTFOUND	Sloupec zadaný v druhém parametru pro AF neexistuje.	MEDIUM	zkontrolovat druhý parametr AF / zkontrolovat připojení k databázi.

Tabulka A.2 Seznam varování z potomků validátoru *ColumnValidator*

Validátor	Jméno	Zpráva	Priorita	Možné řešení
NotNullValidator	NOTNULL	U sloupce může být porušeno omezení NOTNULL kvůli použití AF, která může vracet NULL hodnoty..	HIGH	přidat NotNullDecorator nebo vybrat nenulovou AF
AutoincrementValidator	AUTOINCREMENT	V tabulce je nastavena anonymizační třída pro sloupec a příkazem autoincrement.	MEDIUM	sloupec neanonymizovat
TypeValidator	NOTNULL_DECORATOR_PARAMETER_TYPE	U sloupce nelze použít parametr NotNullDecoratoru v anonymizační třídě pro číselný datový typ.	MEDIUM	upravit parametr NotNullDecoratoru
TypeValidator	EXPECTED_TYPE	U sloupce je nastavena menší velikost datového typu než může nastavená AF vrátit na výstupu.	MEDIUM	zkontrolovat zda je AF vhodně vybrána
LengthValidator	TRUNCATE_DECORATOR_ZERO	TruncateDecorator u anonymizační třídy zkracuje délku výstupní hodnoty AF na 0.	LOW	zvážit, zda je vhodné
LengthValidator	TYPE_SIZE	U sloupce je nastavena menší velikost datového typu než může AF vrátit.	MEDIUM	změnit velikost datového typu / použít TruncateDecorator
TextLengthValidator	TYPE_SIZE_WITH_TRUNCATE_DECORATOR	U sloupce je nastavena menší velikost datového typu než může AF vrátit i po použití TruncateDecoratoru.	HIGH	upravit parametr TruncateDecoratoru / změnit velikost datového typu
TextLengthValidator	UNUSED_TRUNCATE_DECORATOR	U sloupce vybraná AF vrací řetězce kratší než je nastavený parametr TruncateDecoratoru.	LOW	TruncateDecorator lze odstranit
TextLengthValidator	TEXT_TYPE_LENGTH_ZERO	U sloupce je nastaven textový datový typ s nulovou velikostí.	MEDIUM	zkontrolovat datový typ sloupce
NumberRangeValidator	NUMBER_TYPE_LENGTH	U sloupce je nastaven číselný datový typ se zvláštní velikostí range a scale.	MEDIUM	zkontrolovat datový typ sloupce
WhereClauseColumnValidator	WHERE_CLAUSE_COLUMN	U sloupce je nastavena where klauzule, která (společně s where tabulky) způsobí, že se neanonymizují žádná data.	HIGH	upravit where clause
DataDiscoveryValidator	EXPECTED_DATA	U sloupce je podle vstupních dat pravděpodobnost, že je použita nevhodná anonymizační funkce.	MEDIUM	zkontrolovat anonymizační třídu

Tabulka A.3 Seznam varování z potomků validátoru *TableValidator*

Validátor	Jméno	Zpráva	Priorita	Možné řešení
UniqueValidator	UNIQUE_INDEX	U sloupce je použita neunikátní AF, což může porušit IO pro sloupce v unikátním indexu.	HIGH	použít na sloupce v unikátním indexu pouze unikátní AF
UniqueValidator	UNIQUE_UPDATE	Nelze při anonymizaci zaručit UNIQUE IO kvůli použití patternu 'Update source table.	LOW	
UniqueValidator	DECORATOR_IN_UNIQUE	U sloupce je použitá anonymizační třída s dekorátorem, který může způsobit porušení unikátnosti pro sloupce v unikátním indexu.	MEDIUM	
FKValidator	FK_SAME_TYPE	V tabulkách není u PK a FK nastaven stejný datový typ sloupce.	MEDIUM	upravit schéma
FKValidator	FK_SAME_ANONYMIZATION_CLASS	V tabulkách není u PK a FK pro sloupec nastavená stejná anonymizační třída.	MEDIUM	použít stejnou třídu k zachování integrity
FKValidator	FK_CONSTISTENT	U sloupce může být porušena referenční integrita mezi FK a PK tabulek, kvůli použití nekonzistentní AF.	MEDIUM	použít k anonymizaci PK i FK konzistentní AF
PKValidator	PK_COUNT	V tabulce je nastaveno více primárních klíčů.	HIGH	upravit schéma, v tabulce může být pouze jeden primární klíč
PKValidator	PK_NOTNULL	U sloupce může být porušeno omezení PK kvůli použití AF, která může vracet NULL hodnoty.	HIGH	vybrat nenulovou AF
PKValidator	PK_NOTNULL_DECORATOR	U sloupce je použita anonymizační třída s NotNullDecorator, který sice zabrání možnosti NULL hodnot z AF, ale může kvůli němu dojít k porušení unikátnosti.	HIGH	u PK NotNullDecorator nepoužívat, vybrat AF, která nevrací NULL hodnoty
PKValidator	PK_INDEX	U sloupce je použita neunikátní AF, což může porušit unikátnost dat pro sloupce v primárním klíči.	HIGH	použít na sloupce PK pouze unikátní AF
PKValidator	PK_TRUNCATE_DECORATOR	U sloupce je použitá anonymizační třída s TruncateDecorator, který může způsobit porušení unikátnosti pro sloupce v PK.	MEDIUM	u PK TruncateDecorator nepoužívat
DataValidator	NO_DATA	V tabulce nejsou žádná data k anonymizaci nebo se je nepovedlo načíst.	HIGH	zkontrolovat databazové připojení / validovat pouze schéma
WhereClauseTableValidator	WHERE_CLAUSE	U tabulky je nastavena where klauzule, která způsobí, že se neanonymizují žádná data.	HIGH	upravit where clause

Příloha B

Seznam anonymizačních funkcí

Následující tabulka obsahuje seznam všech AF a jejich nových vlastností. V tabulce je sloupec „Maximální délka“ s hodnotou, kterou AF bude vracet v metodě *getMaxReturnLengthWithParameters()* (důvod je doplněn ve sloupci „Komentář“). Pokud je ve sloupci uvedena třída slovníku, je tím myšlena délka nejdelšího slova ze slovníku. Konstantní délky byly převzaty z analýzy.

Ve sloupci „Discoverer“ je uvedena třída průzkumníka, kterou AF vrací metodou *getAnonFunctionDiscoverer()* a v dalším sloupci „Vhodný druh“ jsou uvedeny druhy datových typů (v průzkumníkovi, *SUITABLE_DATATYPES*), které jsou vhodné pro osobní údaje, které AF anonymizuje.

Přehled všech vlastností anonymizačních funkcí lze zobrazit v tabulce „anonymizacniFunkce“ na přiloženém médiu.

Třída	Maximální délka	Komentář	*Discoverer*	Vhodný druh
FunctionAccntAll	22	předčíslí-číslo účtu/kód banky, např. 000000-1234567890/0600	*BBANDiscoverer*	textový, číselný
FunctionAccntAllUnique	22	předčíslí-číslo účtu/kód banky, např. 000000-1234567890/0600	*BBANDiscoverer*	textový, číselný
FunctionAccntNo	16	včetně předčíslí, bez kódu banky	*BBANDiscoverer*	textový, číselný
FunctionAccntNoInt	16		*BBANDiscoverer*	textový, číselný
FunctionAccntNoIntUnique	16		*BBANDiscoverer*	textový, číselný
FunctionAccntPfx	6	předčíslí	*NumberDiscoverer*	textový, číselný
FunctionAccntPfxUnique	6	předčíslí	*NumberDiscoverer*	textový, číselný
FunctionActualDate	19	formát: YYYY-MM-DD HH:MM:SS	*DateDiscoverer*	textový, datum
FunctionAdrCity	StreetDictionary + 4 + CityDictionary + 2	formát: Ulice ČP Obec, délka ČP je 4, 2 mezery	*CityDiscoverer*	textový
FunctionAdrStreet	StreetDictionary + 4 +1	formát: Ulice ČP, délka ČP je 4, 1 mezera	*StreetDiscoverer*	textový
FunctionAdrZip	10	formáty: format 10000, 100 00, PSČ 100 00	*PSCDiscoverer*	textový, číselný
FunctionAdrZipCity	10 + 1 + CityDictionary	formát: PSČ Obec	*PSCDiscoverer*	textový, číselný
FunctionBankCode	4		*NumberDiscoverer*	textový, číselný
FunctionBankCodeUnique	4		*NumberDiscoverer*	textový, číselný
FunctionBusinessName	BusinessNameDictionary			
FunctionCity	CityDictionary		*CityDiscoverer*	textový
FunctionCityDistrict	CityDictionary omezené na města z odpovídajícího okresu		*CityDiscoverer*	textový
FunctionCityRegion	CityDictionary omezené na města z odpovídajícího kraje		*CityDiscoverer*	textový
FunctionConst	constValue.length()	délku určuje konstantní parametr funkce		
FunctionCountry	délka formátu nebo z CountryDictionary	formát zadává druhý parametr: ‘Short, ‘N-3, ‘A-2, ‘A-3, ‘Default. Výstupy: Česko, 203, CZ, CZE, Česká republika	*CountryDiscoverer*	textový, číselný
FunctionDataBox	7			

FunctionDateChar	délka formátu	formát podle parametru, nejdelší možnost \uv{September DD YYYY}	*DateDiscoverer*	textový, datum
FunctionDateDate	délka formátu	formát: yyyy-MM-dd HH:mm:ss.S	*DateDiscoverer*	textový, datum
FunctionDateDateTrunc	délka formátu	formát: yyyy-MM-dd HH:mm:ss.S	*DateDiscoverer*	textový, datum
FunctionDateDateUnique	23		*DateDiscoverer*	textový, datum
FunctionDecimalNumberRandom	max_number.length() nebo min_number.length()	délku určují parametry intervalu	*NumberDiscoverer*	textový, číselný
FunctionDic	10	CZ+IČO	*NumberDiscoverer*	textový, číselný
FunctionDicUnique	10	CZ+IČO	*NumberDiscoverer*	textový, číselný
FunctionDistrict	DistrictDictionary		*TextDiscoverer*	textový
FunctionDistrictRegion	DistrictDictionary omezená na okresy z kraje	kraj může být zadán druhým parametrem	*TextDiscoverer*	textový
FunctionFirstName	FirstnameMDictionary nebo FirstnameFDictionary	můžský/ženský slovník určuje druhý parametr \uv{pohlavi}	*FirstNameDiscoverer*	textový
FunctionGenericUnique				
FunctionHousenum	4	předpokládaný rozsah ČP: 1-9999	*NumberDiscoverer*	textový, číselný
FunctionIBAN	34 obecný, 24 pro ČR/SK	rozlišuje IBAN podle kódu země	*IBANDiscoverer*	textový
FunctionIco	8		*IdentifikacniCisloDiscoverer*	textový, číselný
FunctionIcoUnique	8		*IdentifikacniCisloDiscoverer*	textový, číselný
FunctionIntegerNumberRandom	max_number.length() nebo min_number.length()	délku určují parametry intervalu	*NumberDiscoverer*	textový, číselný
FunctionMail	25	zvolena doporučovaná průměrná délka emailové adresy	*EmailDiscoverer*	textový
FunctionMailUniq	25	zvolena doporučovaná průměrná délka emailové adresy	*EmailDiscoverer*	textový
FunctionNameAll	FirstnameMDictionary + SurnameMDictionary + 1 nebo ženské	můžský/ženský slovník jmen a přijmení určuje druhý parametr \uv{pohlavi}	*TextDiscoverer*	textový
FunctionPaymentCardNumber	21		*NumberDiscoverer*	textový, číselný
FunctionRc	10		*RodneCisloDiscoverer*	textový, číselný
FunctionRcUnique	10		*RodneCisloDiscoverer*	textový, číselný
FunctionRcZaLomitkem	4		*NumberDiscoverer*	textový, číselný
FunctionRegion	RegionDictionary		*TextDiscoverer*	textový
FunctionStreet	StreetDictionary		*StreetDiscoverer*	textový

FunctionSurnameName	SurnameMDictionary nebo SurnameFDictionary	můžský/ženský slovník určuje druhý parametr \uv{pohlavi}	*SurnameDiscoverer*	textový
FunctionTelNumber	9		*PhoneDiscoverer*	textový, číselný
FunctionTelNumberUniq	9		*PhoneDiscoverer*	textový, číselný
FunctionTextRandom	in_length	délka řetězce je zadána přímo parametrem	*TextDiscoverer*	textový
FunctionTitulAfter	TitulAfterDictionary		*TextDiscoverer*	textový
FunctionTitulBefore	TitulBeforeDictionary		*TextDiscoverer*	textový

Bibliografie

1. SCHUH, Matěj. *Implementace datové vrstvy pro anonymizační nástroj: bakalářská práce.* Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.
2. ŠKORNIČKOVÁ, Eva. *Obecné nařízení o ochraně osobních údajů: Osobní údaje* [online]. 2017-09-11 [cit. 2022-04-21]. Dostupné z: `https://www.gdpr.cz/gdpr/heslo/osobni-udaje/`.
3. ŠKORNIČKOVÁ, Eva. *Obecné nařízení o ochraně osobních údajů: Citlivé osobní údaje* [online]. 2017-09-04 [cit. 2022-04-21]. Dostupné z: `https://www.gdpr.cz/gdpr/heslo/citlive-osobni-udaje/`.
4. *Číslo účtu* [online]. Česká bankovní asociace, 2021 [cit. 2022-04-21]. Dostupné z: `https://www.financnivzdelavani.cz/svet-financi/platebni-sluzby/cislo-uctu/`.
5. *IBAN - mezinárodní formát čísla účtu* [online]. Česká národní banka, 2021 [cit. 2022-04-21]. Dostupné z: `https://www.cnb.cz/cs/platebni-styk/iban/iban-mezinarodni-format-cisla-uctu/`.
6. *Co je číslo karty?* [Online]. MONETA Money Bank, a.s., 2022 [cit. 2022-04-30]. Dostupné z: `https://www.moneta.cz/slovnik-pojmu/detail/co-je-cislo-karty`.
7. *Poštovní směrovací číslo* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-30]. Dostupné z: `https://cs.wikipedia.org/wiki/Po%C5%A1tovn%C3%AD_sm%C4%9Brovac%C3%AD_%C4%8D%C3%ADslo`.
8. *Identifikační číslo osoby* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-30]. Dostupné z: `https://cs.wikipedia.org/wiki/Identifika%C4%8Dn%C3%AD_%C4%8D%C3%ADslo_osoby`.
9. *Daňové identifikační číslo* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2022-04-30]. Dostupné z: `https://cs.wikipedia.org/wiki/Da%C5%88ov%C3%A9_identifika%C4%8Dn%C3%AD_%C4%8D%C3%ADslo`.
10. *Rodné číslo*. Ministerstvo vnitra České republiky, Odbor správních činností, 2020. Dostupné také z: `https://www.mvcr.cz/clanek/rady-a-sluzby-dokumenty-rodne-cislo.aspx`.
11. KUČEROVÁ, Helena. *Relační databáze* [online]. Praha : Národní knihovna ČR: KTD: Česká terminologická databáze knihovnictví a informační vědy, 2003 [cit. 2022-04-21]. Dostupné z: `https://aleph.nkp.cz/F/?func=direct&doc_number=000000126&local_base=KTD`.
12. POKORNÝ, Jaroslav; VALENTA, Michal. *Databázové systémy: Schéma relace, schéma relační databáze*. 2. přepracované vydání. Praha: Česká technika - nakladatelství ČVUT, 2020. ISBN 978-80-01-06696-6.

13. *Zpracování hodnot null* [online]. Microsoft, 2022-04-20 [cit. 2022-04-21]. Dostupné z: `https://docs.microsoft.com/cs-cz/dotnet/framework/data/adonet/sql/handling-null-values`.

14. LAURENČÍK, Marek. *SQL - Podrobný průvodce uživatele: Indexy v tabulkách*. První vydání. Praha: Grada Publishing, 2018. ISBN 978-80-271-0774-2.

15. LAURENČÍK, Marek. *SQL - Podrobný průvodce uživatele: Další vlastnosti sloupců*. První vydání. Praha: Grada Publishing, 2018. ISBN 978-80-271-0774-2.

16. *Data Tutorials: How to Define an Auto Increment Primary Key in Oracle* [online]. Chartio, 2021 [cit. 2022-05-01]. Dostupné z: `https://chartio.com/resources/tutorials/how-to-define-an-auto-increment-primary-key-in-oracle/`.

17. *Referenční integrita a omezení cizího klíče* [online]. Distančně.cz, 2022 [cit. 2022-04-24]. Dostupné z: `https://www.distancne.cz/kurz/relacni-databaze-a-dotazovaci-jazyk-sql/lekce-8/`.

18. POKORNÝ, Jaroslav; VALENTA, Michal. *Databázové systémy: Referenční integrita*. 2. přepracované vydání. Praha: Česká technika - nakladatelství ČVUT, 2020. ISBN 978-80-01-06696-6.

19. POKORNÝ, Jaroslav; VALENTA, Michal. *Databázové systémy: Příkaz CREATE TABLE*. 2. přepracované vydání. Praha: Česká technika - nakladatelství ČVUT, 2020. ISBN 978-80-01-06696-6.

20. PERNER, Pavel. *Refaktoring a metodika testování nástroje pro anonymizaci dat: bakalářská práce*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

21. BRYCHTA, Ondřej. *Anonymizace osobních údajů pro databáze MySQL a Teradata: bakalářská práce*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

Obsah přiloženého média

- `readme.txt` ..... stručný popis obsahu média
- `src`
  - `implementace` ..... zdrojové kódy implementace
  - `BP_ValidatorAnonymizacnihoModelu` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_ValidatorAnonymizacnihoModelu.pdf` ..... text práce ve formátu PDF
  - `anonymizacniFunkce.xlsl` ..... připojená tabulka všech anonymizačních funkcí