FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání bakalářské práce

Název:	Anonymizace osobních údajů pro databázi PostgreSQL
Student:	Tomáš Průšek
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 2023/2024

Pokyny pro vypracování

Seznamte se s existující implementací nástroje Winch, který slouží pro anonymizaci osobních údajů v relačních databázích. Popište aktuální stav implementace anonymizace pro PostgreSQL databázi. Identifikované chybějící části implementujte tak, aby výsledek anonymizace byl stejný jako v ostatních databázích (Oracle, MSSQL, DB2). Dále analyzujte požadavky na rozšíření implementované anonymizace o možnost anonymizovat údaje ve strukturovaných dokumentech (XML a JSON) uložených v databázi. Prostudujte možnosti, které databáze PostgreSQL pro práci s těmito dokumenty nabízí, a navrhněte vhodné řešení. Diskutujte různé přístupy, které je možné využít. Při volbě zohledněte rychlost provedení vlastní anonymizace u jednotlivých variant. Pro implementaci anonymizace v rámci dokumentu (XML, JSON) vycházejte z diplomové práce Jakuba Doležala [1]. Zvolené řešení implementujte a důkladně otestujte.

[1] Doležal, Jakub. Anonymizace osobních údajů ve strukturovaných dokumentech. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Elektronicky schválil/a Ing. Michal Valenta, Ph.D. dne 19. září 2022 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Anonymizace osobních údajů pro databázi PostgreSQL

Tomáš Průšek

Katedra softwarového inženýrství
Vedoucí práce: Ing. Jiří Mlejnek

10. května 2023

Poděkování

Rád bych poděkoval vedoucímu bakalářské práce Ing. Jiřímu Mlejnkovi za jeho rady, připomínky a ochotu během tvorby bakalářské práce. Dále bych rád poděkoval své rodině za podporu.

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 10. května 2023 . . . . . .

Odkaz na tuto práci

Průšek, Tomáš. *Anonymizace osobních údajů pro databázi PostgreSQL.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2023.

Abstrakt

Práce se zabývá analýzou současného stavu anonymizace v databázi PostgreSQL v nástroji Winch a implementováním chybějících částí anonymizace. Dále se věnuje návrhu a implementaci rozšíření anonymizace pro databázi PostgreSQL v nástroji Winch o možnost anonymizovat JSON a XML dokumenty.

Analytická část se zabývá identifikováním chybějících částí anonymizace v databázi PostgreSQL. Dále se věnuje analýze nástrojů, jež PostgreSQL databáze poskytuje pro práci s JSON a XML dokumenty.

Na základě zjištěných informací je anonymizace JSON dokumentů implementována pomocí nástrojů, které databáze PostgreSQL poskytuje, a k implementaci anonymizace XML dokumentů je využito rozšíření PL/Python.

Výsledkem práce je plně funkční anonymizace v databázi PostgreSQL včetně rozšíření umožňujícího anonymizovat strukturované dokumenty.

Klíčová slova Winch, anonymizace, osobní údaje, XML, JSON, PostgreSQL

Abstract

The bachelor thesis focuses on the analysis of the current state of anonymization in the PostgreSQL database in the tool Winch and the implementation of the missing parts of anonymization. In addition, it deals with the design and implementation of the extension for anonymizing XML and JSON documents in the PostgreSQL database in the Winch tool.

The analytical part focuses on identifying the missing parts of anonymization in the PostgreSQL database. Moreover, it analyzes the tools provided by the PostgreSQL database for working with JSON and XML documents.

Based on the information found, the anonymization of JSON documents was implemented using the tools provided by the PostgreSQL database, and the PL/Python extension was used to implement the anonymization of XML documents.

The result of the bachelor thesis is fully functional anonymization in the PostgreSQL database, including an extension allowing to anonymize structured documents.

Keywords Winch, anonymization, personal data, XML, JSON, PostgreSQL

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza **5**
2.1 Nástroj Winch 5
2.2 Využití anonymizace 5
2.3 Průběh anonymizace 6
2.3.1 Průběh anonymizace v databázi 6
2.3.1.1 Inicializace 6
2.3.1.2 Vyhledávání osobních údajů 6
2.3.1.3 Konfigurace anonymizace 7
2.3.1.4 Generace SQL kódu 7
2.3.1.5 Anonymizování 8
2.3.2 Průběh anonymizace na souborovém systému 8
2.3.2.1 Inicializace 8
2.3.2.2 Vyhledávání osobních údajů 8
2.3.2.3 Konfigurace 8
2.3.2.4 Anonymizace 9
2.4 Současný stav implementace anonymizace pro PostgreSQL databázi 9
2.4.1 Struktura balíčku disl-winch-postgresql 9
2.4.2 Generátory SQL kódu 9
2.4.3 Pomocná třída 12
2.4.4 Hledání citlivých údajů 12
2.4.5 Sestavení SQL dotazu 13
2.4.6 Validace 13
2.4.7 Vytvoření slovníků 13
2.4.8 Mapování logického schéma na fyzické 13

2.4.9 Anonymizační funkce 13
2.4.10 Testování 14
2.4.11 Revize existující dokumentace 15
2.5 Popis formátů používaných pro strukturovaná data 15
2.5.1 XML 15
2.5.1.1 XPath 16
2.5.2 JSON 16
2.5.2.1 JSONPath 17
2.6 Nástroje nutné k anonymizaci strukturovaného dokumentu v PostgreSQL databázi 17
2.7 Podpora pro XML v databázi PostgreSQL 18
2.7.1 Způsoby uložení XML dokumentů v databázi PostgreSQL 19
2.7.2 Funkce pro práci s XML dokumenty 19
2.7.3 Shrnutí 20
2.8 Řešení nedostatečné podpory práce s XML dokumenty v databázi PostgreSQL 20
2.8.1 Práce s XML dokumentem jako s textem 21
2.8.2 PL/Java 21
2.8.3 PL/Python 21
2.8.4 Zhodnocení 21
2.9 Podpora pro JSON v databázi PostgreSQL 22
2.9.1 Způsoby uložení JSON dokumentů v databázi PostgreSQL 22
2.9.2 Operátory pro práci s JSON dokumenty v PostgreSQL databázi 23
2.9.3 Funkce pro vytváření json a jsonb hodnot 24
2.9.4 Funkce pro manipulaci s json a jsonb hodnotami 24
2.9.5 Shrnutí 25

3 Návrh řešení **27**
3.1 Možná řešení 27
3.1.1 Konfigurace ve Winch pro souborový systém a anonymizace na souborovém systému 27
3.1.2 Konfigurace ve Winch pro souborový systém a anonymizace v databázi 28
3.1.3 Rozšíření Winch Add-in a anonymizace v databázi . . . 28
3.1.4 Zhodnocení 28
3.2 Pohled uživatele 29
3.3 Návrh implementace 31
3.3.1 Balíček disl-winch-filesystem 31
3.3.2 Balíček disl-winch-connector 31
3.3.3 Balíček disl-winch-postgresql 34
3.3.3.1 Groovy 34
3.3.3.2 SQL 34
3.3.4 Transformace konfigurace 35

3.3.4.1 Původní konfigurace . . . 36
3.3.4.2 Upravená konfigurace . . . 36
3.4 Omezení . . . 36
3.5 Rozšíření . . . 39

4 Realizace **41**
4.1 Anonymizační funkce . . . 41
4.2 Balíček pattern.codegenerator . . . 41
4.2.1 Procedury . . . 41
4.2.2 Testy . . . 42
4.3 Anonymizace strukturovaných dokumentů . . . 42
4.3.1 Balíček disl-winch-filesystem . . . 42
4.3.2 Balíček disl-winch-connector . . . 43
4.3.3 Balíček disl-winch-postgresql . . . 43
4.3.3.1 Groovy . . . 43
4.3.3.2 SQL . . . 45
4.3.4 Testování . . . 48
4.3.4.1 Anonymizační funkce pro strukturované dokumenty . . . 48
4.3.4.2 Zjištěné chyby již implementovaných testů . . . 48
4.3.5 Omezení . . . 48
4.3.5.1 Tečka v názvu atributu v JSON dokumentu . . . 49
4.3.5.2 Nastavení anonymizačních tříd . . . 49
4.4 Testování . . . 50

5 Testování výkonu implementace **51**
5.1 Způsob testování . . . 51
5.2 Výsledky . . . 52

Závěr **55**

Literatura **57**

A Seznam použitých zkratek **59**

B Obsah zip archivu **61**

Seznam obrázků

2.1 Struktura balíčku disl-winch-postgresql 10
2.2 Hierarchie generátorů kódu 11

3.1 Záložka Discovery v aplikaci Winch s novým tlačítkem 29
3.2 Vyskakovací okno na záložce Discovery v aplikaci Winch 30
3.3 Záložka Anonymization v aplikaci Winch s novým tlačítkem 30
3.4 Vyskakovací okno na záložce Anonymization v aplikaci Winch . . . 30
3.5 Hierarchie tříd načítajících konfiguraci 32
3.6 Hierarchie tříd anonymizačních funkcí 33
3.7 Hierarchie tříd testujících anonymizační funkce 33

4.1 Hierarchie implementovaných anonymizačních funkcí 44
4.2 Hierarchie tříd testujících anonymizační funkce 46

Seznam tabulek

2.1 Shrnutí stavu balíčků 14
2.2 Shrnutí podpory pro práci s XML dokumenty 20
2.3 Shrnutí podpory pro práci s JSON dokumenty 25

5.1 Test výkonu JSON, doba běhu v sekundách 53
5.2 Test výkonu XML, doba běhu v sekundách 53

Seznam výpisů kódu

2.1 XML dokument a XPath ke jménu autora 16
2.2 JSON dokument a JSONPath ke jménu autora 18
3.1 JSON dokument a původní konfigurace 37
3.2 Upravená konfigurace 38
3.3 Dva JSON dokumenty s rozdílnou strukturou a jejich spojení . 38
4.1 JSON dokument s tečkou 49
4.2 Ukázkový JSON dokument 49
4.3 Vygenerovaná konfigurace 50
5.1 Testovaný JSON dokument 51

Úvod

Aplikace Winch je nástroj pro anonymizaci osobních údajů v relačních databázích. Navíc se aktuálně rozšiřuje o možnost anonymizace osobních údajů na souborovém systému (na disku), např. v souborech formátu XLSX, JSON a XML. Hlavním důvodem, proč řešit anonymizaci osobních údajů, je splnění legislativních opatření na ochranu osobních údajů podle Zákona 110/2019 Sb. o zpracování osobních údajů. Anonymizace dat oproti šifrování je výhodná v tom, že anonymizovaná data jsou nerozeznatelná od reálných, ale neobsahují žádné citlivé informace. Jestliže je anonymizace provedena vhodným způsobem, tak není možné získat původní data.

Práce se zabývá analýzou aktuálního stavu implementace anonymizace pro databázi PostgreSQL a jejím dokončením. Dále se zabývá rozšířením anonymizace pro databázi PostgreSQL v aplikaci Winch o možnost anonymizovat strukturované dokumenty typu JSON a XML, přičemž tato část bude vycházet z diplomové práce z FIT ČVUT v Praze Jakuba Doležala na téma Anonymizace osobních údajů ve strukturovaných dokumentech. Toto téma bakalářské práce jsem si zvolil, jelikož ochrana osobních údajů je v současnosti velmi důležitá. Například na mnoha webových stránkách jsme žádáni o udělení souhlasu se zpracováním osobních údajů.

Kapitola Analýza se věnuje aktuálnímu stavu implementace anonymizace pro databázi PostgreSQL. Dále popisuje, jaké funkcionality jsou nutné k anonymizaci strukturovaného dokumentu. Také analyzuje nástroje databáze PostgreSQL pro práci s XML a JSON dokumenty a vyhodnocuje, jestli jsou dostatečné pro anonymizaci.

Následuje kapitola Návrh řešení, která se věnuje návrhu postupu, jakým způsobem se budou anonymizovat strukturované dokumenty v databázi PostgreSQL. Postup bude vycházet z části Analýzy, která se zabývá nástroji databáze PostgreSQL pro práci s XML a JSON dokumenty

Kapitola Realizace se věnuje samotné implementaci navrženého způsobu anonymizace strukturovaných dokumentů. Dále se zabývá odchylkami od na-

vrženého řešení, ke kterým došlo v průběhu implementace. Dále se věnuje implementaci chybějících částí anonymizace v databázi PostgreSQL.

Poslední kapitola Testování výkonu implementace se zabývá otestováním výkonu implementovaného řešení pro anonymizaci strukturovaných dokumentů v databázi PostgreSQL.

Kapitola 1

Cíl práce

Cílem práce je analýza aktuální implementace té části nástroje pro anonymizaci osobních údajů Winch, která má za úkol anonymizovat osobní údaje uložené v databázi PostgreSQL. Dílčím cílem je doplnit anonymizaci osobních údajů v databázi PostgreSQL o identifikované chybějící části tak, aby anonymizace v této databázi fungovala stejně jako v ostatních databázích (např. DB2, Oracle), pro něž je již implementace hotova.

Dále je cílem analyzovat požadavky na rozšíření implementované anonymizace v databázi PostgreSQL o možnost anonymizovat údaje ve strukturovaných dokumentech ve formátu XML a JSON uložených v databázi. Na základě této analýzy budou navrženy možné způsoby anonymizace. Poté bude zvolen způsob anonymizace strukturovaných dokumentů a vybere se, od jaké verze PostgreSQL bude anonymizace strukturovaných dokumentů podporována. Zvolený způsob bude implementován a otestován jednotkovými testy.

Kapitola 2

Analýza

Kapitola se zabývá analýzou současného stavu implementace anonymizace pro databázi PostgreSQL v nástroji Winch a podpory databáze PostgreSQL pro práci s JSON a XML dokumenty.

2.1 Nástroj Winch

Aplikace Winch umožňuje anonymizovat data v podporovaných relačních databázích a hledat v nich citlivé informace. Dále umožňuje anonymizovat soubory podporovaných formátů (např. XLSX, JSON, XML). [1]

Jednou z částí nástroje je konzolová aplikace Winch Actor vykonávající zmiňované procesy. Další částí je rozšíření (Add-In) do aplikace Enterprise Architect, přes které se spouští Winch Actor s různými příkazy a parametry dle nastavení v Enterprise Architect. Nejnovější částí je varianta Winch pro anonymizaci na souborovém systému. [1]

Části Winch Actor a Winch pro anonymizaci na souborovém systému jsou vyvíjeny v programovacím jazyce Groovy a sestavovány nástrojem Gradle. Winch Actor používá SQL pro práci s databázemi. Winch Add-In je napsán v jazyce C#. [1]

2.2 Využití anonymizace

Anonymizace osobních údajů může sloužit ke splnění legislativních opatření na ochranu osobních údajů podle Zákona 110/2019 Sb. o zpracování osobních údajů [2]. Mohou ji využívat organizace pracující s velkým množstvím osobních údajů, např. pojišťovny a banky.

První situací, kdy je anonymizace vhodná, je, když firmy předávají data nějakému externímu subjektu, např. si nechávají zpracovávat statistiku, např. kolik lidí má určitý produkt. Pak mohou data s anonymizovanými osobními údaji bez obav předat externímu subjektu, aby jim zpracoval statistiku, aniž

by hrozilo, že poruší zákon o ochraně osobních údajů, např. že dojde k úniku dat. [3]

Další situací je tvorba testovacích dat. Mohou se použít data reálná, ve kterých se anonymizují osobní údaje. [3]

2.3 Průběh anonymizace

Tato sekce popisuje průběh anonymizace a zabývá se jejími jednotlivými fázemi. Nejprve se zabývá průběhem anonymizace v databázi a poté na souborovém systému.

2.3.1 Průběh anonymizace v databázi

Část se zabývá tím, jak probíhají jednotlivé fáze anonymizace v databázi. Nejprve popisuje inicializaci databáze, poté vyhledávání osobních údajů v databázi, dále popisuje konfigurace anonymizace, generování SQL kódu a nakonec samotnou anonymizaci dat. Hlavní je, že při anonymizaci v databázi se pracuje s tabulkami a jejich sloupci.

2.3.1.1 Inicializace

Nejprve je nutné inicializovat podpůrné objekty v databázi, ve které bude probíhat anonymizace. K anonymizaci je nutné v databázi vytvořit anonymizační funkce, které poté anonymizují jednotlivé hodnoty, např. funkce anonymizující jméno. Některé anonymizační funkce potřebují slovníky, které jim slouží k tomu, že z nich vyberou novou hodnotu, např. funkce anonymizující jméno vypočítá z původního jména číslo, které určuje, kolikátou položkou ze slovníku jmen nahradí původní jméno.

Generování kódu anonymizačních funkcí, které anonymizují data, zajišťuje balíček *anonymization*. Vytvoření slovníků zajišťuje třída *PostgresqlDictionaryBuilder*, která umožňuje sestavit CREATE DROP a INSERT script slovníku.

2.3.1.2 Vyhledávání osobních údajů

Druhou fází anonymizace je proces discovery, který je volitelný a který prohledá databázi za účelem nalezení citlivých údajů. Většina funkčností sloužících k nalezení osobních údajů je implementována společně pro všechny databáze. Část hledání osobních údajů, která je specifická pro každou databázi, realizuje *PostgresqlDiscoverExecutor*, která poskytuje implementaci základních statistických funkcí pro tabulky v PostgreSQL, a *PostgresqlDiscoverTableManager*, která umožňuje kontrolovat, zda daná tabulka neobsahuje citlivé údaje.

Hledání osobních údajů funguje tak, že se vždy pracuje s jedním sloupcem v tabulce v databázi. Na tento sloupec se zkouší různé discovery třídy, které hodnoty v jednotlivých řádcích testují. Podle počtu řádků, které projdou testem, vypočítají pravděpodobnost, že daný sloupec obsahuje citlivé údaje. Hodnoty v řádcích mohou testovat, jestli odpovídají jejich regulárnímu výrazu. Např. v tabulce osoba je sloupec nazvaný email obsahující řetězec. Na tento sloupec se kromě jiných discovery tříd spustí i discovery třída pro email. Tato třída bude testovat, zda hodnoty ve sloupci odpovídají jejímu regulárnímu výrazu, tj. jestli hodnoty jsou ve tvaru alfanumerické znaky, po nich znak zavináč, po něm znaky anglické abecedy, pak tečka a po ní 2 až 4 znaky anglické abecedy. Hodnotu v jednotlivých řádcích mohou také testovat pomocí validačních funkcí, např. jestli je daná hodnota validní rodné číslo. Dále také mohou danou hodnotu zkusit vyhledat ve slovníku, např. jestli je daný řetězec jméno.

2.3.1.3 Konfigurace anonymizace

Třetí fází anonymizace je samotné nastavení anonymizačního procesu, tj. které sloupce a jakým způsobem se budou anonymizovat a nastavení anonymizačních tříd. Např. v databázi je tabulka osoba se sloupcem jméno, který má být anonymizován. Nejprve se vytvoří anonymizační třída, která se pojmenuje např. Jméno a nastaví se jí anonymizační funkce v databázi a druhý parametr, např. fc_firstname a F, což znamená, že hodnoty ve sloupci, k jehož anonymizaci se má využít anonymizační třída Jméno, budou předány anonymizační funkci fc_firstname spolu s druhým parametrem F a výsledek se použije jako nová hodnota. Druhý parametr F znamená, že výsledná jména budou ženská.

Kromě mapování logického schématu na fyzické, které zajišťuje *PostgresqlSchema*, řeší tuto fázi jiná část kódu společná pro všechny databáze.

2.3.1.4 Generace SQL kódu

Další fází je generace SQL kódu, který anonymizaci realizuje. Součástí této fáze je uložení vygenerovaného SQL kódu na souborový systém, nebo vložení do databáze. Zajišťuje případné vytvoření nových tabulek v cílovém schématu, nastavení integritních omezení tabulek a anonymizaci dat a jejich vložení do cílových tabulek. Tato část je pro každou databázi implementovaná zvlášť, jelikož databáze se liší dostupnými datovými typy a dostupnými funkcemi.

Nejprve se pro každou tabulku vytvoří SQL skript, který zajistí její vytvoření v cílovém schématu. Např. anonymizuje se tabulka osoba se sloupci jméno a příjmení, tak se vygeneruje skript, který vytvoří v cílovém schématu tabulku osoba se sloupci jméno a příjmení.

Dále se pro každou tabulku vytvoří skript, který vybere data z původní tabulky, anonymizuje je a vloží do cílové tabulky. Např. ve zdrojovém schématu je tabulka osoba se sloupci jméno a příjmení s tím, že se má anonymizovat

příjmení pomocí funkce fc_surname, tak se vygeneruje skript, který vybere všechny řádky z této tabulky, na všechna příjmení aplikuje funkci fc_surname a upravené řádky vloží do cílové tabulky.

Nakonec se pro každou tabulku vytvoří skript, který obnoví cizí klíče. Např. ve zdrojovém schématu je tabulka osoba se sloupci id, který je primárním klíčem (identifikátorem), jméno a příjmení. Také se ve zdrojovém schématu nachází tabulka auto, která má sloupec majitel, který obsahuje cizí klíč (id osoby). Vygenerovaný skript zajistí, že se obnoví integritní omezení na cizí klíč (Nebude např. možné smazat z tabulky osoba osobu, jejíž cizí klíč se nachází v nějakém záznamu v tabulce auto).

SQL kód pro vytvoření nových tabulek, jejich integritních omezení a vložení anonymizovaných dat generují třídy v balíčku *pattern.codegenerator*.

2.3.1.5 Anonymizování

Poslední fází je spuštění vygenerovaného SQL kódu v databázi. Celá tato část je implementována společně pro všechny databáze, kromě vytvoření SQL příkazu pro spouštění anonymizace, který poskytuje třída *PostgresqlDatabaseHelper*.

2.3.2 Průběh anonymizace na souborovém systému

Oproti anonymizaci v databázi se při anonymizaci na souborovém systému nepracuje se schématy, tabulkami a jejich sloupci, ale s jednotlivými soubory různých formátů.

2.3.2.1 Inicializace

Při založení nové anonymizace se nastaví zdrojová a pracovní složka. Volitelně se může nastavit složka, do které se zkopírují původní soubory. Na rozdíl od inicializace v databázi není nutné nic inicializovat.

2.3.2.2 Vyhledávání osobních údajů

Druhou fázi anonymizace je proces discovery, který je volitelný a prohledá rekurzivně soubory ve vybrané složce. Discovery je na souborovém systému omezené, podporuje jen omezenou množinu formátů souborů. Soubory, ve kterých najde citlivé údaje, doporučí k anonymizaci. Uživatel může doporučené nebo ručně označené soubory nastavit, že se budou anonymizovat.

2.3.2.3 Konfigurace

Další fází je nastavení anonymizace jednotlivých souborů. Možnosti nastavení se liší podle typu souboru. U tabulkových souborů (např. xlsx) lze nastavit, jak

se budou anonymizovat jednotlivé sloupce v jednotlivých tabulkách. U strukturovaných souborů (JSON a XML) lze nastavit, jak se budou anonymizovat atributy v jednotlivých uzlech. U ostatních souborů lze pouze nastavit, zda se má smazat nebo nahradit výchozí nebo vlastní šablonou stejného formátu. Pro každý soubor se konfigurace ukládá zvlášť jako JSON soubor do pracovní složky.

2.3.2.4 Anonymizace

Poslední fází je postupné anonymizování souborů podle vytvořené konfigurace.

2.4 Současný stav implementace anonymizace pro PostgreSQL databázi

Sekce popisuje současný stav implementace anonymizace pro databázi PostgreSQL. Probírá jednotlivé balíčky a zabývá se tím, které třídy v nich chybí nebo nejsou dokončené. V tabulce 2.1 je celkový přehled stavu jednotlivých balíčků.

2.4.1 Struktura balíčku disl-winch-postgresql

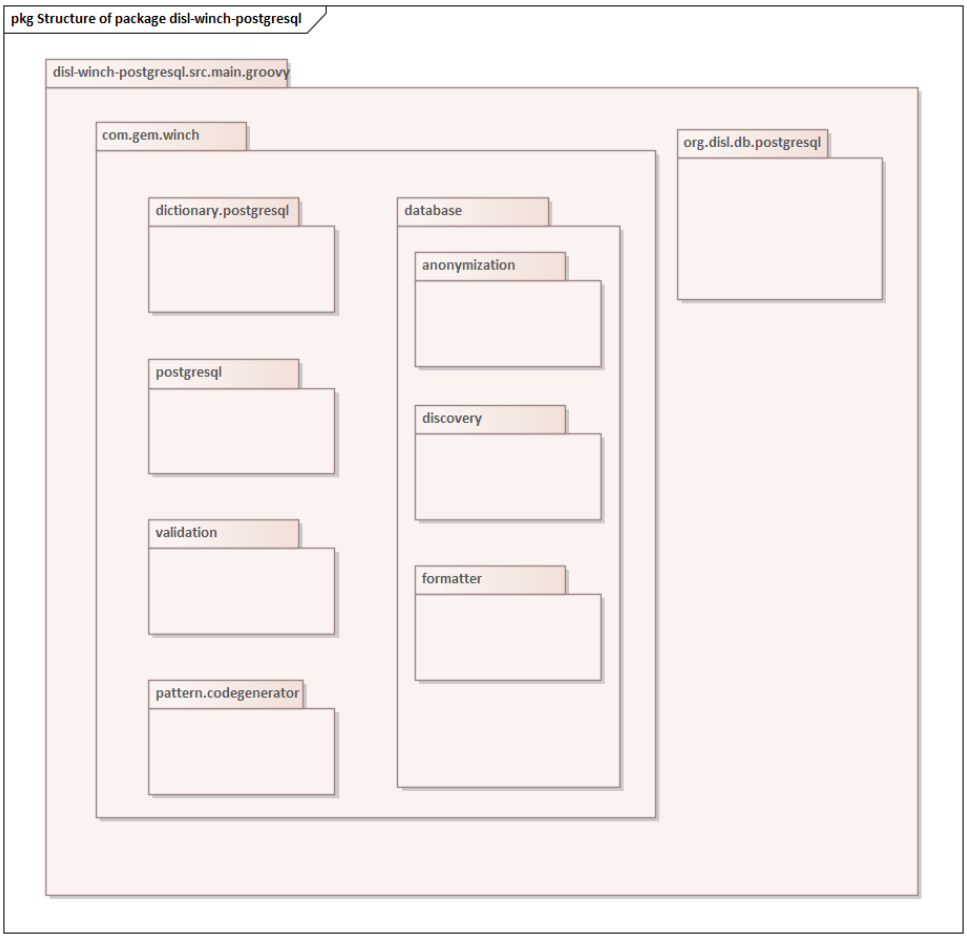
V tomto balíčku se nachází implementace specifická pro anonymizaci v databázi PostgreSQL. Má standardní strukturu Gradle projektu. Na obrázku 2.1 je zachycena jeho struktura.

2.4.2 Generátory SQL kódu

Balíček *pattern.codegenerator* je zodpovědný za generování SQL kódu. Nachází se v něm jednotlivé třídy generující SQL kód a třída *PostgresqlCodegeneratorFactory*, která poskytuje metody k získání jednotlivých generátorů SQL kódu. V některých metodách vyhazuje výjimku NotImplementedException, jelikož některé generátory, které mají tyto metody vracet, ještě nejsou implementovány.

Každý generátor obsahuje metodu *getCode*, která vrací SQL kód, který vytvoří procedury v databázi, které se poté spustí během anonymizace. Procedury mohou například nastavit tabulce cizí klíče, nebo vybrat data z tabulky, anonymizovat je a následně je vložit do cílové tabulky. Na obrázku 2.2 je zachycena hierarchie generátorů kódu.

Dále se zde nachází třída *PostgresqlDiscoverTableManager*, která umožňuje kontrolovat, zda daná tabulka neobsahuje osobní/citlivé údaje.



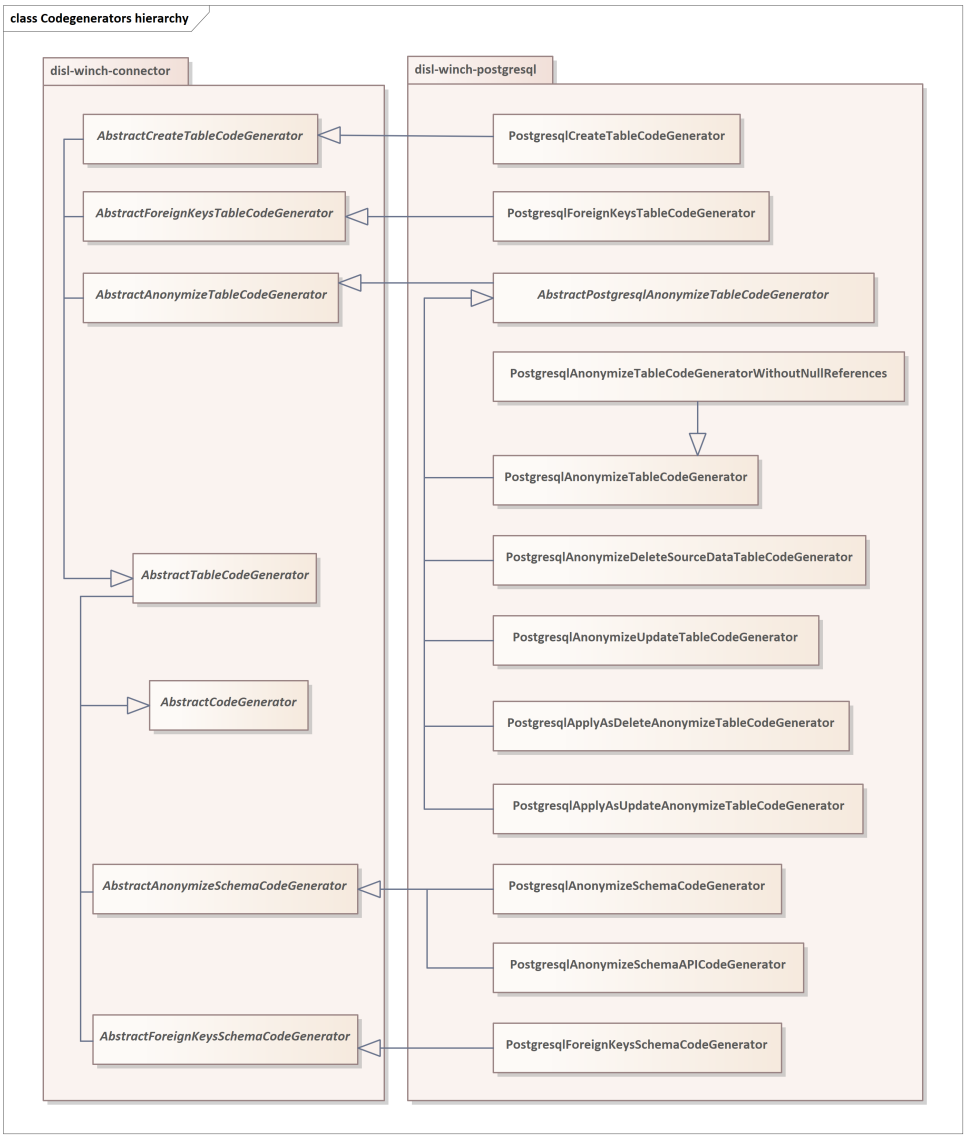
Obrázek 2.1: Struktura balíčku disl-winch-postgresql

Zde je popis hlavních tříd:

PostgresqlDiscoverTableManager umožňuje kontrolovat, zda daná tabulka neobsahuje osobní/citlivé údaje.

PostgresqlCodegeneratorFactory poskytuje metody, které vrací jednotlivé generátory SQL kódu.

AbstractPostgresqlAnonymizeTableCodeGenerator generuje SQL kód, který vytvoří v databázi funkci, která zkopíruje řádky ze zdrojové tabulky do cílové a zaznamená to v tabulce WORKFLOW_LOG. Negeneruje část, která se stará o výběr řádků a jejich vložení do cílové tabulky, jen volá metody *getOperationQuery* a *getSelectQuery*, které potomci této třídy přepisují, čímž vzniká jejich specifické chování. Metoda *getOperati-*



Obrázek 2.2: Hierarchie generátorů kódu

onQuery vrátí kód operace, která se provede, *getSelectQuery* vrátí kód, který vybere data z databáze.

PostgresqlAnonymizeSchemaCodeGenerator generuje kód pro vytvoření funkce, která spustí všechny funkce, které se vygenerovaly přes *PostgresqlAnonymizeTableCodeGenerator* a které anonymizují jednotlivé tabulky.

PostgresqlAnonymizeTableCodeGenerator dědí z *AbstractPostgresqlA-*

nonymizeTableCodeGenerator a přepisuje metodu *getSelectQuery*, aby vybrala řádky ze zdrojové tabulky podle podmínky a jejich počet nepřesáhl určený limit. Hodnoty ze sloupečků, které mají nastavenou anonymizaci, se před vložením do cílové tabulky anonymizují.

PostgresqlCreateTableCodeGenerator generuje kód pro vytvoření nové tabulky v cílovém schématu se všemi sloupečky a omezeními tabulky ve zdrojovém schématu.

PostgresqlForeignKeysTableCodeGenerator generuje SQL funkci, která zajistí obnovení všech cizích klíčů na dané tabulce.

Některé již implementované generátory kódu vrací SQL kód, který neprojde ani syntaktickou kontrolou databáze PostgreSQL. Například třída *PostgresqlCreateTableCodeGenerator* obsahovala v metodě *getCode* SQL kód, který byl pouze zkopírován z implementace pro MS SQL a bylo nutné ji přepsat. Celkově ze stavu tříd, které již byly implementované, bylo zřejmé, že se je nikdo ani nepokusil spustit v databázi. Dále chybí dokumentační komentáře kódu a u některých částí kódu je tedy velmi složité zjistit, co mají dělat. Celkově je tento balíček nedodělaný.

V ostatních databázích je málo testů na otestování tříd v balíčku *pattern.codegenerator*, pro databázi PostgreSQL nejsou v tomto balíčku žádné testy. Tyto testy jsou většinou napsány zbytečně složitě a SQL kód vygenerovaný generátory nespouštějí v databázi s tím, že by kontrolovaly výsledek, ale pouze vygenerovaný SQL kód porovnají s regulárním výrazem. Toto řešení není vhodné, jelikož SQL kód, který by dělal to, co se očekává, nemusí projít, ale SQL kód, který by naopak neprošel ani syntaktickou kontrolou, projít může. Původně měly být testy napsány podle metodiky definované prací jiného studenta. Avšak tato práce stále není hotova a neexistuje žádná jednotná metodika, jak se tyto testy mají psát. Bude tedy nutné testy napsat vlastním způsobem.

2.4.3 Pomocná třída

Balíček *database* obsahuje třídu *PostgresqlDatabaseHelper*, která má metody k získání instancí tříd a hodnot specifických pro databázi, např. podporované datové typy, oddělovač příkazů, atd. Využívá se při generování SQL kódu a spouštění vygenerovaných funkcí v databázi při samotné anonymizaci. Balíček je dle implementace v ostatních databázích hotový.

2.4.4 Hledání citlivých údajů

Balíček *discovery* obsahuje třídu *PostgresqlDiscoverExecutor*, která je využita během discovery procesu a poskytuje implementaci základních statistických

funkcí pro tabulky v PostgreSQL databázi. Například obsahuje funkce pro zjištění počtu řádků v tabulce, počtu řádků v tabulce vyhovující zadané podmínce nebo počtu řádků v tabulce, kde hodnota v zadaném sloupci projde zadanou validační funkcí (např. kolik řádků v tabulce osoba obsahuje ve sloupci rodné číslo validní rodné číslo). Dle implementace v ostatních databázích je tento balíček dodělaný.

2.4.5 Sestavení SQL dotazu

Balíček *formatter* obsahuje třídu *PostgresqlDatabaseFormatter*, která poskytuje rozhraní anonymizačním funkcím pro sestavení SQL dotazu. Má metody *getSelectQueryForFunction*, která slouží k vytvoření SELECT dotazu pro funkci, a *getFullyQualifiedFunctionName*, která vrátí plně kvalifikované jméno funkce. Tento balíček je již dodělaný.

2.4.6 Validace

V implementaci pro databázi PostgreSQL balíček *validation* zcela chybí. Dle implementace pro ostatní databáze by měl obsahovat třídu *PostgresqlValidationTableManager*, která slouží ke kontrole, že jsou správně nastavené anonymizační třídy pro jednotlivé sloupce (např. zkontroluje, že se nepoužije anonymizační funkce vracející neunikátní hodnoty na sloupec s integritním omezením UNIQUE).

2.4.7 Vytvoření slovníků

Balíček *dictionary.postgresql* obsahuje třídu *PostgresqlSqlDictionaryBuilder*, která umožňuje sestavit CREATE DROP a INSERT script slovníku pro spuštění v PostgreSQL databázi, což se využívá ve fázi inicializace.

2.4.8 Mapování logického schéma na fyzické

Balíček *org.disl.db.postgresql* obsahuje třídu *PostgresqlSchema*, která je potomkem *PhysicalSchema*. Tato třída rozšiřuje implementaci *PhysicalSchema*, případně ji přepisuje o vlastnosti specifické pro PostgreSQL. Mapuje logické schéma použité v DISL modelu na fyzické schéma a obsahuje informace pro připojení k databázi.

2.4.9 Anonymizační funkce

V balíčku *anonymization* se nacházejí anonymizační funkce. Téměř všechny anonymizační funkce jsou již implementovány, chybí pouze GenericUnique, RCZaLomitkem a TextRandom.

GenericUnique slouží k anonymizaci unikátních sloupců. Vstupním parametrem je původní řetězec, jehož každý znak poté funkce posune o určitý

počet pozic v ASCII. Posunutí je dáno výchozí maskou, nebo záznamem v tabulce anonym_conf. Malá písmena jsou nahrazena malými, velká velkými, čísla čísly a ostatní znaky jsou ponechány. Tato funkce vyžadovala, aby byla napsána SQL funkce fc_generic_unique, která poté tuto funkcionalitu realizuje v databázi.

RCZaLomitkem slouží k anonymizaci části rodného čísla za lomítkem. Má dva parametry: řetězec reprezentující rodné číslo a řetězec reprezentující, jestli bylo již ověřeno, že se jedná o validní rodné číslo. Druhý parametr je nepovinný, pokud není zadán, tak se provede před anonymizací ověření, že se jedná o validní rodné číslo. Tato funkce ke svému fungování vyžaduje SQL funkce fc_rodne_cislo_za_lomitkem a validaceRC. Funkce validaceRC provede kontrolu, že zadané rodné číslo je validní.

TextRandom vrátí náhodný řetězec. Má dva parametry: původní řetězec a délku výsledného řetězce. Druhý parametr je nepovinný, pokud není zadán, tak délka výsledného řetězce je stejná, jako délka původního. Pokud je vstupní řetězec NULL, řetězec délky nula nebo zadaná délka výsledného řetězce je nula, tak funkce vrátí NULL. Jinak generuje náhodně jednotlivé znaky výsledného řetězce. Tato funkce vyžadovala, aby byla napsána SQL funkce fc_text_rnd, která poté tuto funkcionalitu realizuje v databázi.

Tabulka 2.1: Shrnutí stavu balíčků

Balíček	Stav
pattern.codegenerator	nedodělaný, nutné přepsat
database	již hotové
discovery	již hotové
formatter	již hotové
validation	zcela chybí
dictionary.postgresql	již hotové
org.disl.db.postgresql	již hotové
anonymization	nedodělaný

2.4.10 Testování

Všechny již implementované anonymizační funkce jsou otestovány včetně hraničních hodnot a neplatných hodnot. Pro neimplementované anonymizační funkce bude nutné napsat testy s tím, že testovací data jsou již připravena. Dále bude nutné napsat testy pro všechny neimplementované i implementované generátory kódu.

2.4.11 Revize existující dokumentace

V dokumentaci k nástroji Winch chybí ucelený návod, jak zprovoznit testovací prostředí pro databázi PostgreSQL, aby se daly spouštět testy. Nikde není uvedeno, která verze PostgreSQL se má použít. Dále chybí, jaká schémata se mají v testovací databázi vytvořit, a jak se mají nastavit. Také chybí, jací uživatelé mají být v databázi definovaní a co mají mít za oprávnění. Navíc chybí, která rozšíření mají být v databázi vytvořena.

U mnoha tříd nikde v dokumentaci není popsáno, jaký je jejich účel a jak fungují. Ani v kódu často nejsou žádné dokumentační komentáře, které by jejich účel vysvětlovaly.

2.5 Popis formátů používaných pro strukturovaná data

Sekce krátce popisuje dva formáty pro strukturovaná data XML a JSON. Zabývá se základy jejich struktury a syntaxe. Dále se zabývá jazykem XPath, který je určený pro navigaci v XML dokumentech a jazykem JSONPath, který slouží k navigaci v JSON dokumentech. Zabývá se pouze těmito dvěma jazyky, jelikož ty byly použity ve výstupech práce Jakuba Doležala, které tato práce využívá.

2.5.1 XML

XML je textový formát pro výměnu a ukládání dat, který má stromovou strukturu. Skládá se z elementů. Každý element musí být ohraničený startovací a ukončovací značkou, toto se nevztahuje na prázdné elementy. Znaky mezi startovací a ukončovací značkou jsou obsahem elementu. Element může obsahovat text, další element nebo být prázdný. Značky jsou:

Startovací např. <tag>

Ukončovací např. </tag>

Značka prázdného elementu např. <tag/> [4], [5], [6]

Jakákoliv značka může obsahovat atribut. Atribut je dvojice klíč-hodnota, např. <img name=”picture”>. Hodnota atributu musí být vždy v uvozovkách, mohou být použity jednoduché i dvojité uvozovky. Atribut nemůže obsahovat více hodnot a stromovou strukturu (na rozdíl od elementů). Každy well-formed XML dokument musí mít jeden kořenový element, který je rodičem všech ostatních elementů. Dále existuje content fragment, což je XML dokument, který má více kořenových elementů. [4], [5], [6]

Výpis kódu 2.1: XML dokument a XPath ke jménu autora

```
<catalog>
   <book id="bk101" uid="5489adw101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating
      applications with XML.</description>
   </book>
</catalog>

//catalog/book/author
```

2.5.1.1 XPath

XPath je jazyk sloužící k vybrání uzlů nebo množiny uzlů v XML dokumentech. Jedná se o podmnožinu jazyka XQuery. Umožňuje specifikovat množinu uzlů pomocí relativní nebo absolutní cesty. Lze také přistupovat k atributům jednotlivých uzlů. V XPath se rozlišují malá a velká písmena. Umožňuje se pohybovat ve stromu jak vertikálně (předci i potomci), tak i horizontálně mezi sourozenci. Pokud se přistupuje v XPath k výsledku, který je množinou, tak se indexuje od čísla jedna. XPath dále obsahuje mnoho vestavěných funkcí, které například poskytují možnost filtrace podle textu uzlu. Viz příklad 2.1. Zde jsou základní výrazy:

název uzlu vybere všechny uzly s daným jménem.

/ vybere od kořenového uzlu.

// vybere uzly v dokumentu z aktuálního uzlu, který odpovídá výběru.

. vybere aktuální uzel.

.. vybere rodiče aktuálního uzlu.

@ vybere atribut. [4], [7], [8]

2.5.2 JSON

JSON je textový formát pro výměnu a ukládání dat, který má stromovou strukturu. Je postaven na objektech a polích. Každý JSON soubor má jeden kořenový objekt nebo pole. Objekt je neuspořádaná množina dvojic klíč - hodnota. Objekt je ohraničen složenými závorkami. Názvy hodnot jsou ohraničeny

dvojitými uvozovkami a hodnoty jsou od nich odděleny dvojtečkou. Jednotlivé dvojice klíč - hodnota jsou odděleny čárkou (za poslední dvojicí nemůže být čárka). [9], [10]

Hodnota může být řetězec, číslo, objekt, pole, boolovské hodnoty (true, false) a null. Řetězec musí být ohraničen dvojitými uvozovkami, číselné hodnoty, boolovské hodnoty a null nejsou ohraničeny uvozovkami. Při zápisu desetinných čísel se používá tečka. Pole je uspořádaná kolekce hodnot. Při jeho zápisu se používají hranaté závorky. V objektu nebo poli mohou být vnořena další pole nebo objekty. [9], [10]

2.5.2.1 JSONPath

JSONPath je ekvivalentem XML pro JSON dokumenty. Umožňuje specifikovat množinu uzlů pouze pomocí absolutní cesty. Avšak neumožňuje oproti XPath procházet strom směrem ke kořeni či horizontálně. Pokud se přistupuje v JSONPath k výsledku, který je množinou, tak se indexuje od čísla nula. Viz ukázka 2.2. Zde jsou základní výrazy:

název uzlu vybere všechny uzly s daným jménem.

$ vybere kořenový objekt/element.

@ vybere aktuální objekt/element.

. nebo [] slouží pro přístup k potomkovi.

.. vybere všechny potomky (i nepřímé) aktuálního uzlu, kteří odpovídají výběru.

***** je zástupný znak. [11]

2.6 Nástroje nutné k anonymizaci strukturovaného dokumentu v PostgreSQL databázi

Při anonymizování strukturovaného dokumentu dostaneme strukturovaný dokument a konfiguraci, která bude specifikovat, jak má být dokument anonymizován. K anonymizování strukturovaného dokumentu je nutné provést několik kroků.

Prvním krokem je ověření, že konfigurace obsahuje všechny potřebné údaje.

Druhým krokem je ověření, že vstupní dokument je validním XML nebo JSON dokumentem a jeho převod do správného formátu (jak bude uvedeno dále, strukturovaný dokument může být uložen v databázi vícero způsoby, a s některými z nich se jednodušeji a efektivněji pracuje).

Výpis kódu 2.2: JSON dokument a JSONPath ke jménu autora

```
{
   "catalog":{
      "book":[
         {
            "@id":"bk102",
            "author":"Ralls, Kim",
            "title":"Midnight Rain",
            "genre":"Fantasy",
            "price":"5.95",
            "publish_date":"2000-12-16",
            "description":"A former architect battles."
         }
      ]
   }
}

$.catalog.book[*].author
```

Dalším krokem je přečtení konfigurace, která definuje, které prvky a jakým způsobem se budou anonymizovat. Tato konfigurace je sama strukturovaným dokumentem (JSON).

Dalším krokem je provedení samotné anonymizace.

Posledním krokem je převod anonymizovaného dokumentu do původního formátu.

K převodu strukturovaného dokumentu do formátu, ve kterém bude anonymizován, a zpět do původního formátu jsou nutné funkce pro konverzi.

K přečtení konfigurace, ve které je definováno, jak bude anonymizace probíhat, je nutná funkce, která vrátí hodnotu prvku definovaného pomocí dotazu.

K anonymizaci prvků ve strukturovaném dokumentu bude nutná funkce, která vrátí hodnotu prvku definovaného pomocí dotazu. Dále bude nutná funkce, která bude schopna změnit původní hodnotu prvku definovaného pomocí dotazu na anonymizovanou hodnotu.

Je tedy nutné, aby nástroje databáze PostgreSQL pro práci se strukturovanými dokumenty podporovaly validaci, konverzi, hledání a modifikaci.

2.7 Podpora pro XML v databázi PostgreSQL

Funkcionality pro práci s XML dokumenty byly do databáze PostgreSQL přidávány postupně. Zde je krátký výčet verzí, kdy byla podpora zásadně rozšířena z hlediska potřeb anonymizace XML dokumentů:

8.2 je první verze PostgreSQL, která poskytuje podporu pro práci s XML dokumenty. Poskytuje pouze základní funkce pro práci s XML dokumenty, např. funkce pro validaci XML a indexování. Chybí datový typ optimalizovaný pro ukládání XML dokumentů.

8.3 je verzí PostgreSQL, ve které byla podpora pro práci s XML dokumenty rozšířena. Byl přidán datový typ xml a funkce pro vytváření XML dokumentů a jejich zpracování. [12], [13]

V pozdějších verzích se již nepřidávaly žádné funkcionality, které by byly zásadní z pohledu anonymizace. Následující sekce této části se budou tedy zabývat verzí 8.3.

2.7.1 Způsoby uložení XML dokumentů v databázi PostgreSQL

XML dokumenty mohou být v databázi PostgreSQL uloženy ve vícero datových typech:

text je jeden z datových typů, do kterých lze uložit XML dokument, ale pak není žádná kontrola ze strany PostgreSQL, že jsou validní. Dále nelze použít funkce, které PostgreSQL databáze nabízí pro práci s XML dokumenty.

xml má tu výhodu, že vstupní hodnoty jsou zkontrolovány, jestli jsou platnými XML dokumenty a databáze poskytuje funkce pro práci s nimi. Datový typ xml může uchovávat well-formed dokumenty definované XML standardem a content fragments. Datový typ xml nevaliduje vstupní hodnoty proti možné DTD deklaraci ani XSchema. Datový typ xml neposkytuje porovnávací operátory. Důsledkem toho je, že není možné přímo indexovat sloupce tohoto typu. [12]

2.7.2 Funkce pro práci s XML dokumenty

Pro převod anonymizovaného dokumentu do datového typu xml lze použít funkci XMLPARSE, pro převod z datového typu xml na řetězec lze použít XMLSERIALIZE. Funkci IS DOCUMENT lze použít k ověření, že anonymizovaný dokument je validní XML dokument. Funkci xpath(xpath, xml [, nsarray]) lze použít k získání konkrétních XML hodnot.

XMLPARSE (DOCUMENT | CONTENT value) konvertuje řetězec na hodnotu typu xml, DOCUMENT se použije, pokud je parametr well-formed, CONTENT, pokud se jedná o content fragment.

XMLSERIALIZE (DOCUMENT | CONTENT value AS type) převede xml hodnotu na hodnotu znakového řetězce (character, character varying a text).

xml IS DOCUMENT vrátí true, jestliže XML hodnota je validní XML dokument, false, pokud ne (včetně content fragmentu), nebo null, pokud argument je null.

xpath(xpath, xml [, nsarray]) vyhodnocuje výrazy XPath 1.0 v parametru xpath proti XML hodnotě v parametru xml. Vrací pole XML hodnot odpovídající množině uzlů, které vrátí XPath výraz. Třetím argumentem funkce je pole mapování namespace. Pole by mělo být dvoudimenzionální s délkou druhé osy rovné 2 (mělo by to být pole polí, z nichž každé je délky 2). [13]

2.7.3 Shrnutí

Podpora pro práci s XML dokumenty není ve verzi 8.3, ani v žádné další verzi dostatečná. Viz tabulka 2.2. Od této verze jsou sice již dostupné funkce pro převod anonymizovaného dokumentu do vhodného datového typu a zpět a funkce pro získání hodnot elementů, které se budou anonymizovat. Avšak chybí funkce pro přepsání původní hodnoty v XML dokumentu anonymizovanou hodnotou. Tato funkce není ani v nejnovější verzi PostgreSQL 15.

Tabulka 2.2: Shrnutí podpory pro práci s XML dokumenty

Operace	Stav
Validace	Podporováno
Konverze	Podporováno
Hledání	Podporováno
Modifikace	Nepodporováno

2.8 Řešení nedostatečné podpory práce s XML dokumenty v databázi PostgreSQL

Vzhledem k tomu, že funkcionality pro práci s XML dokumenty nejsou dostatečné, jak je uvedeno v předchozí části, je nutné najít náhradní řešení. Jednou z možností je pracovat s XML dokumentem jako textem, což je však velmi problematické, jak bude vysvětleno později. Další možností je doinstalování rozšíření, které umožní provést potřebné operace s XML dokumenty, do databáze PostgreSQL.

2.8.1 Práce s XML dokumentem jako s textem

Jednou z možností, jak nahradit hodnotu v XML dokumentu za jinou, je pracovat s daným dokumentem jako s textem a použít funkci pro nahrazení části řetězce za jiný. Nejprve by se XML dokument rozložil na jednotlivé uzly, v nich by se pomocí funkce replace nahradily původní hodnoty za anonymizované, a poté by se z upravených uzlů složil výsledný dokument. Toto by se realizovalo pomocí rekurzivní funkce. Dané řešení by však pravděpodobně bylo pomalé a náchylné k chybě, pokud by se v hodnotě uzlu vyskytovaly speciální znaky a určité řetězce.

2.8.2 PL/Java

Toto rozšíření umožňuje psát funkce v Javě. Navíc umožňuje importovat balíčky, takže by bylo možné importovat balíček pro práci s XML dokumenty nebo použít kód z implementace anonymizace XML dokumentů na souborovém systému. [14]

Avšak instalace rozšíření PL/Java je velmi složitá. Vyžaduje nainstalovaný kompilátor jazyků C a C++. Dále vyžaduje nainstalovanou specifickou verzi JDK a Maven. Dalším krokem je přes příkazovou řádku dané rozšíření nainstalovat, což může z mnoha důvodů selhat a řešení těchto problémů může být komplikované. Například je nutné při instalaci zjistit, jakým kompilátorem a s jakými přepínači byl zkompilován PostgreSQL. Pokud se k instalaci PostgreSQL použil .exe soubor na Windows, tak tato informace v konfiguraci zcela chybí a je velmi obtížné ji doplnit. Z toho vyplývá, že zprovoznění by pro běžného uživatele bylo velmi náročné. [14]

2.8.3 PL/Python

PL/Python je rozšíření, které umožňuje psát funkce v Pythonu. Stejně jako PL/Java umožňuje importovat balíčky, takže je možné importovat balíček pro práci s XML dokumenty. [15]

Instalace rozšíření je jednoduchá, stačí pouze nainstalovat konkrétní verzi Pythonu a v databázi spustit příkaz CREATE EXTENSION plpython3u. [15]

2.8.4 Zhodnocení

Z výše uvedených důvodů se jeví jako nejlepší použití rozšíření PL/Python, jelikož jeho instalace není příliš složitá a výsledné řešení nebude komplikované a náchylné k chybám. Rozšíření PL/Java je příliš složité na zprovoznění a vyžaduje velmi pokročilé znalosti. Práce s XML dokumentem jako s textem je náchylná k chybám a výsledné řešení bude komplikované.

2.9 Podpora pro JSON v databázi PostgreSQL

Podpora pro práci s JSON dokumenty byla do databáze PostgreSQL přidávána postupně. Zde je krátký výčet verzí, ve kterých se přidávaly zásadní funkcionality pro anonymizaci JSON dokumentů.

9.2 je první verze PostgreSQL, která poskytuje alespoň nějakou podporu pro práci s JSON dokumenty. V této verzi poskytuje pouze datový typ json a funkce pro vytváření json array_to_json - převede pole na json a row_to_json - převede řádek na json.

9.3 je verzí PostgreSQL, ve které byla podpora pro práci s JSON dokumenty rozšířena. Byly přidány JSON operátory a nové funkce pro vytváření a úpravu JSON dokumentů.

9.4 je další verzí PostgreSQL, ve které byla podpora pro práci s JSON dokumenty zásadně rozšířena. Byl přidán typ jsonb, nové JSON operátory a nové funkce pro vytváření a úpravu JSON dokumentů.

9.5 je další verzí PostgreSQL, ve které byla podpora pro práci s JSON dokumenty zásadně rozšířena. Byly přidány nové JSON operátory a nové funkce pro vytváření a úpravu JSON dokumentů. [16], [17]

V pozdějších verzích se již nepřidávaly žádné funkcionality, které by byly zásadní z pohledu anonymizace. Následující sekce této části se budou tedy zabývat verzí 9.5.

2.9.1 Způsoby uložení JSON dokumentů v databázi PostgreSQL

JSON dokumenty mohou být v databázi PostgreSQL uloženy ve vícero datových typech:

text je jeden z datových typů, do kterých lze uložit JSON dokument, ale pak není žádná kontrola ze strany PostgreSQL, že jsou validní. Dále nelze použít funkce a operátory, které PostgreSQL databáze nabízí pro práci s JSON dokumenty.

json ukládá přesnou kopii vstupního textu, který funkce pro práci s JSON dokumenty musí pokaždé zpracovat. Jelikož ukládá přesnou kopii vstupu, tak zachová sémanticky nepodstatné bílé znaky a pořadí klíčů v JSON objektech. Jestliže objekt obsahuje stejný klíč vícekrát, pak jsou všechny dvojice klíč/hodnota zachovány.

jsonb ukládá data v dekomponovaném binárním formátu. To trochu zpomaluje vkládání nových dokumentů, ale výrazně urychluje práci s nimi, jelikož se nemusí pokaždé zpracovat. Datový typ jsonb také podporuje

indexaci, která může dále urychlit práci s JSON dokumenty. Jsonb nezachovává bílé znaky, pořadí klíčů v objektech, ani nezachovává duplicitní klíče. Pokud jsou duplicitní klíče specifikovány, na vstupu je ponechána pouze poslední hodnota. Při konvertování textového JSONu na jsonb jsou primitivní hodnoty namapovány na nativní PostgreSQL typy. To způsobuje, že na hodnoty jsou kladena další omezení. Datový typ jsonb odmítne čísla, která jsou mimo rozsah numeric, NaN a infinity. Povoluje true a false pouze malými písmeny. Hodnota null se namapuje na (none), jelikož SQL NULL je jiný koncept. Nepovoluje Unicode zápis \uxxxx pro non-ASCII znaky(nad U+007F) a \u0000. Validní Unicode zápis znaku je konvertován na ekvivalent v ASCII nebo UTF8. [16]

Z těchto datových typů se pro proces anonymizace nejvíce hodí jsonb, jelikož mnoho operátorů pro práci s JSON dokumenty je dostupných pouze pro něj. Operace nad dokumentem uloženém v jsonb by také měly být rychlejší, což by mělo urychlit anonymizaci.

2.9.2 Operátory pro práci s JSON dokumenty v PostgreSQL databázi

V konfiguraci budou jednotlivé položky k anonymizaci uloženy jako JSON array. K získání jednotlivých položek bude užitečný operátor, -> nebo ->>. K ověření, že konfigurace obsahuje všechny potřebné položky, mohou být použity operátory @>, <@, ? a ?&. K získání hodnoty z JSON dokumentu, která se má anonymizovat, můžou být použity operátory #>nebo #>>. Zde je podrobnější popis jednotlivých operátorů pro práci s JSON dokumenty, které by mohly být užitečné při anonymizaci:

-> Když je pravý operand typu int, tak vrátí element JSON array s indexem specifikovaným pravým operandem. Když je pravý operand typu text, tak vrátí hodnotu daného klíče.

->> Jestliže je pravý operand typu int, tak vrátí element JSON array s indexem specifikovaným pravým operandem jako text. Když je pravý operand typu text, tak vrátí hodnotu daného klíče jako text.

#> vrátí JSON objekt na cestě specifikované pravým operandem.

#>> vrátí JSON objekt na cestě specifikované pravým operandem jako text.

@> je dostupný pouze pro jsonb, slouží k ověření, zdali JSON hodnota nalevo obsahuje JSON hodnotu napravo.

<@ je dostupný pouze pro jsonb, slouží k ověření, zdali JSON hodnota napravo obsahuje JSON hodnotu nalevo.

? je dostupný pouze pro jsonb, slouží k ověření, zdali JSON hodnota nalevo obsahuje klíč/element napravo.

?& je dostupný pouze pro jsonb, slouží k ověření, zdali JSON hodnota nalevo obsahuje všechny klíče/elementy napravo. [17]

2.9.3 Funkce pro vytváření json a jsonb hodnot

Pro konvertování JSON dokumentu z datového typu text nebo json do jsonb může být použita funkce to_jsonb. Zde je podrobnější popis funkcí pro vytváření json a jsonb hodnot:

to_jsonb(anyelement) vrátí hodnotu jako jsonb. Pole a složené objekty konvertuje rekurzivně na pole a objekty. Jestliže je dostupná konvertovací funkce, bude použita ke konverzi, jinak bude vytvořena skalární jsonb hodnota. Pro jakýkoliv skalární typ kromě number, boolean nebo null je použita textová reprezentace.

array_to_json(anyarray [, pretty_bool]) vrátí pole jako JSON pole. Vícedimenzionální pole bude převedeno na JSON pole polí. Pokud je parameter pretty_bool nastaven na true, tak mezi prvky 1. dimenze bude přidáno odřádkování. [17]

2.9.4 Funkce pro manipulaci s json a jsonb hodnotami

Při procházení JSON pole anonymizovatelných položek v konfiguraci bude potřebné vědět, kolik jich je, např. kvůli for-cyklu. K tomu lze použít funkci jsonb_array_length. K získání konkrétní json hodnoty je možné použít funkce jsonb_extract_path nebo jsonb_extract_path_text. Ke kontrole, že konfigurace obsahuje všechny klíče, lze použít funkci jsonb_object_keys. Přepsání původní hodnoty anonymizovanou hodnotou lze docílit pomocí funkce jsonb_set. Lze použít funkci jsonb_pretty ke zvýšení čitelnosti výsledku anonymizace. Zde je podrobnější popis funkcí pro manipulaci s json a jsonb hodnotami:

jsonb_array_length(jsonb) vrátí počet prvků ve vnějším JSON poli.

jsonb_extract_path(from_jsonb jsonb, VARIADIC path_elems text[]) vrátí json hodnotu definovanou v parametru path_elems (ekvivalentní s operátorem #>)

jsonb_extract_path_text(from_jsonb jsonb, VARIADIC path_elems text[]) vrátí jsonb hodnotu definovanou v parametru path_elems jako text (ekvivalent k operátoru #>>)

jsonb_object_keys(jsonb) vrátí množinu klíčů ve vnějším JSON objektu.

jsonb_set(target jsonb, path text[], new_value jsonb [, create_missing boolean]) vrátí target s částí definovanou v parametru path nahrazenou hodnotou parametru new_value, nebo přidanou, pokud parametr create_missing je true (výchozí je true) a položka definovaná parametrem path neexistuje, ale všechny položky definované parametrem path kromě poslední existují. Negativní celé číslo, které bude v parametru path, bude určovat pořadí prvku od konce JSON pole.

jsonb_pretty(from_json jsonb) konvertuje from_json na text a doplní odsazení. [17]

2.9.5 Shrnutí

Podpora pro práci s JSON dokumenty je od PostgreSQL 9.5 dostatečná pro potřeby anonymizace. Viz tabulka 2.3 Od této verze jsou již dostupné funkce pro převod anonymizovaného dokumentu do vhodného datového typu a zpět. Dále jsou již dostupné funkce pro získání hodnot elementů, které se budou anonymizovat, a funkce, která původní hodnoty nahradí za anonymizované.

Tabulka 2.3: Shrnutí podpory pro práci s JSON dokumenty

Operace	Stav
Validace	Podporováno
Konverze	Podporováno
Hledání	Podporováno
Modifikace	Podporováno

Kapitola 3

Návrh řešení

V analytické části byl představen aktuální stav implementace anonymizace v databázi PostgreSQL a nástroje pro práci s XML a JSON soubory, které PostgreSQL databáze poskytuje. Vzhledem k tomu, že podpora pro práci se strukturovanými dokumenty byla do databáze PostgreSQL přidávána postupně, tak řešení bude uděláno pro verzi 9.5 a novější, jelikož v této verzi jsou nabízené funkcionality již dostatečné. Dále vzhledem k tomu, že nastavení a provedení anonymizace XML a JSON souborů bylo již implementováno v rámci práce Jakuba Doležala, tak bude cílem, aby navrhnuté řešení tuto implementaci maximálně využilo.

3.1 Možná řešení

Tato sekce se zabývá možnými způsoby anonymizace strukturovaných dokumentů a výběrem nejvhodnějšího řešení. Konkrétně se zabývá tím, jak nastavit samotnou anonymizaci strukturovaných dokumentů v databázi a jakým způsobem podle konfigurace strukturované dokumenty anonymizovat.

3.1.1 Konfigurace ve Winch pro souborový systém a anonymizace na souborovém systému

První možností je, že by si uživatel vzal vzorový dokument ze sloupce obsahujícího strukturované dokumenty a stáhl by si ho na souborový systém. Poté by pro něj vytvořil konfiguraci anonymizace v aplikaci Winch pro souborový systém. Vytvořenou konfiguraci by zadal jako druhý parametr anonymizační funkce pro strukturované dokumenty při vytváření anonymizační třídy pro anonymizaci strukturovaných dokumentů v modulu Winch v aplikaci Enterprise Architect. Při spuštění anonymizace by strukturované dokumenty nejprve byly z databáze staženy na souborový systém, kde by se anonymizovaly. Následně by byly anonymizované dokumenty nahrány do databáze.

3.1.2 Konfigurace ve Winch pro souborový systém a anonymizace v databázi

Další možností je, že konfigurace anonymizace by probíhala stejně jako v řešení popsaném v kapitole 3.1.1. Avšak strukturované dokumenty by se anonymizovaly přímo v databázi jako ostatní sloupce.

3.1.3 Rozšíření Winch Add-in a anonymizace v databázi

Jednou z možností, jak rozšířit anonymizaci v PostgreSQL databázi o anonymizaci strukturovaných dokumentů, je rozšíření modulu Winch v aplikaci Enterprise Architect. Provedlo by se tak, že při nastavování anonymizace jednotlivých sloupců tabulek by měl uživatel u sloupců obsahujících XML a JSON dokumenty nakonfigurovat jejich anonymizaci podobně jako ve Winch pro souborový systém. Vytvořenou konfiguraci by si mohl zkopírovat a zadal by ji jako druhý parametr anonymizační funkce pro strukturované dokumenty při vytváření anonymizační třídy pro anonymizaci strukturovaných dokumentů. Po spuštění anonymizace by se strukturované dokumenty v daném sloupci anonymizovaly přímo v databázi jako ostatní sloupce.

3.1.4 Zhodnocení

Možnost, že anonymizace strukturovaného dokumentu by se konfigurovala ve Winch pro souborový systém a samotná anonymizace by probíhala na souborovém systému, by byla jednoduchá na implementaci, šlo by využít velkou část již existující implementace anonymizace strukturovaných dokumentů na souborovém systému, např. GUI pro konfiguraci a samotná anonymizace strukturovaného dokumentu. Toto řešení by pravděpodobně bylo pomalé, jelikož by si muselo nejprve stáhnout dokumenty z databáze, anonymizovat je a nahrát zpět do databáze. Dalším problémem je, že by nebylo uživatelsky přívětivé, jelikož by si musel nejprve stáhnout dokument z databáze, nastavit jeho anonymizaci v aplikaci Winch pro souborový systém a poté ji ještě zkopírovat do modulu Winch v aplikaci Enterprise Architect.

Možnost, že anonymizace strukturovaného dokumentu by se konfigurovala ve Winch pro souborový systém a samotná anonymizace by probíhala na souborovém systému, by byla jednoduchá na implementaci, jelikož by využívala již implementované GUI pro konfiguraci anonymizace strukturovaných dokumentů. Bylo by však nutné implementovat anonymizaci strukturovaných dokumentů přímo v databázi a stejně jako předchozí řešení by nebylo uživatelsky přívětivé.

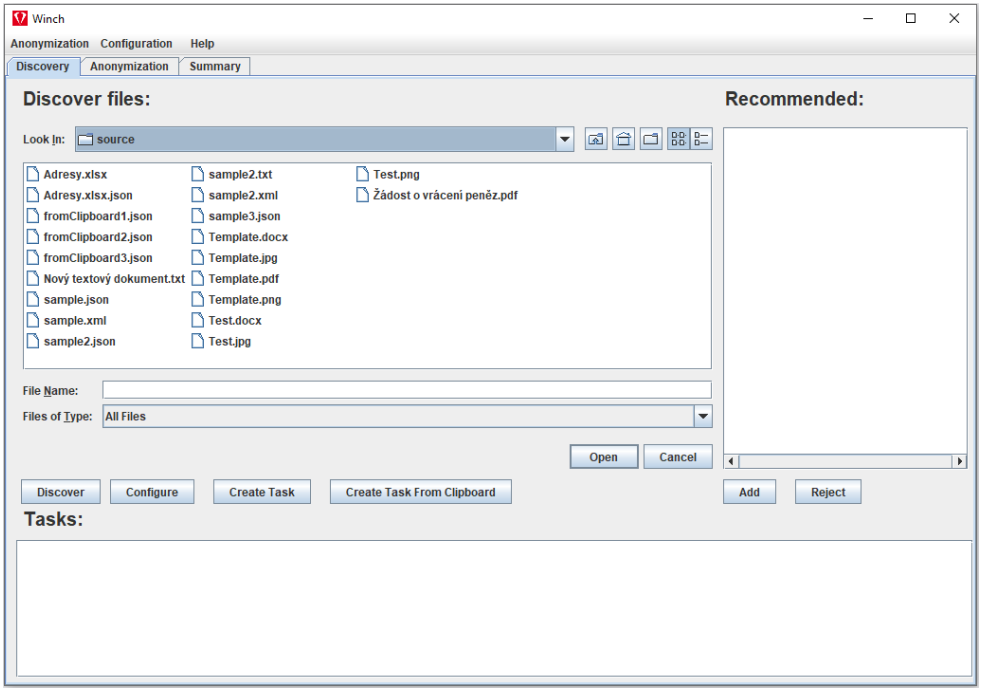
Rozšíření Winch Add-in by bylo složité na implementaci, jelikož je napsaný v programovacím jazyce C# a autor této práce nemá s tímto jazykem žádné zkušenosti. Navíc, jelikož je Winch Add-in napsaný v C# a Winch pro souborový systém v Groovy, tak by nebylo možné využít části implementace anonymizace strukturovaných dokumentů z Winch pro souborový systém.

Z výše uvedených důvodů byla zvolena možnost, kdy konfigurace anonymizace strukturovaného dokumentu bude probíhat ve Winch pro souborový systém a anonymizace v databázi.

3.2 Pohled uživatele

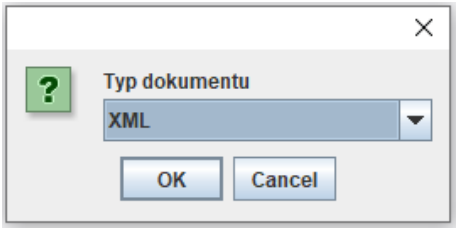
Konfigurace anonymizace bude probíhat tak, že uživatel si stáhne z databáze strukturovaný dokument, který bude chtít anonymizovat, a zkopíruje si ho do schránky.

Poté si otevře aplikaci Winch pro anonymizaci na souborovém systému. Vytvoří si novou anonymizaci. Poté si v záložce Discovery klikne na tlačítko Create Task From Clipboard (zobrazeno na obrázku 3.1), což mu zobrazí okno, ve kterém vybere formát nově vytvářené úlohy (JSON nebo XML) (zobrazeno na obrázku 3.2). Poté se mu ve složce, kterou při vytváření anonymizace zadal jako zdrojovou, vytvoří nový soubor, jehož obsahem bude text ze schránky a jenž bude mít zvolený formát.



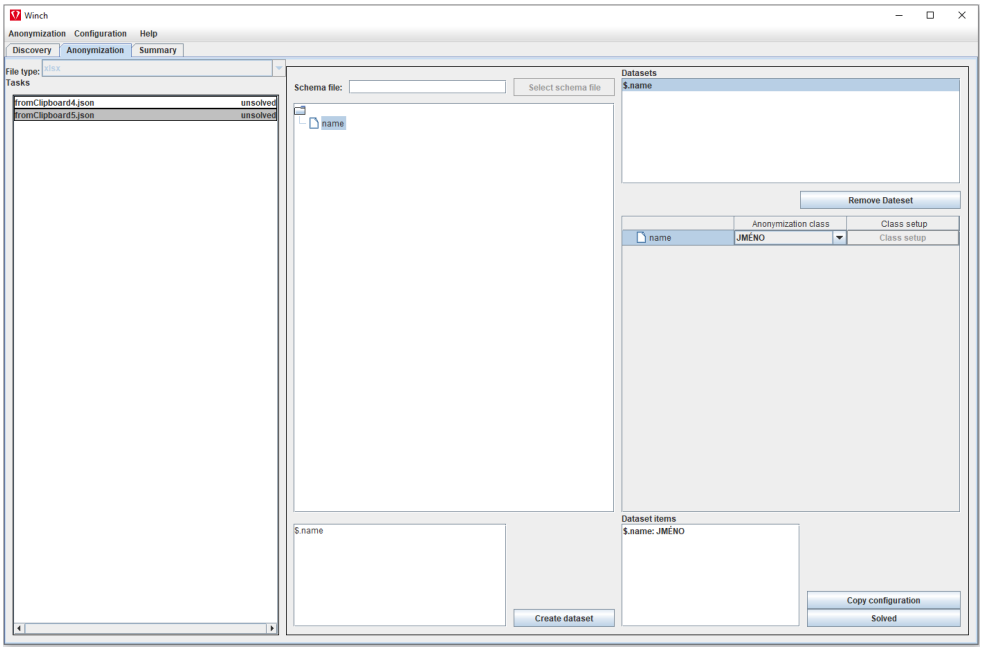
Obrázek 3.1: Záložka Discovery v aplikaci Winch s novým tlačítkem

Poté přejde na záložku Anonymization a zde si vybere daný dokument a nastaví jeho anonymizaci. Poté klikne na tlačítko Copy configuration (zobrazeno na obrázku 3.3) a dostane na výběr, jestli chce daný soubor zachovat nebo smazat (zobrazeno na obrázku 3.4). Poté, co jednu z možností vybere,

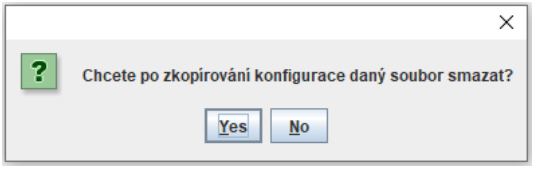


Obrázek 3.2: Vyskakovací okno na záložce Discovery v aplikaci Winch

se mu zkopíruje vygenerovaná konfigurace do schránky a případně se soubor smaže.



Obrázek 3.3: Záložka Anonymization v aplikaci Winch s novým tlačítkem



Obrázek 3.4: Vyskakovací okno na záložce Anonymization v aplikaci Winch

Poté si v aplikaci Winch pro anonymizaci v databázi otevře nastavení anonymizace přes tlačítko specialize->winch->report. Zde vybere tabulku,

ve které se nachází dokumenty k anonymizaci. Pro sloupec, ve kterém se nachází dokumenty, vybere anonymizační třídu pro anonymizaci JSON/XML dokumentu (podle toho, jakého formátu je dokument, který chce anonymizovat). Poté si ve složce, kterou zadal jako pracovní, najde konfiguraci anonymizace pro daný dokument a tu nastaví anonymizační třídě jako druhý parametr. Poté klasicky spustí anonymizaci databáze.

3.3 Návrh implementace

Tato sekce se zabývá návrhem samotné implementace. Nejprve se zabývá potřebnými úpravami v balíčku *disl-winch-filesystem*, dále úpravami v *disl-winch-connector* a poté se zabývá *disl-winch-postgresql*

3.3.1 Balíček disl-winch-filesystem

Nejprve bude nutné přidat tlačítka Create Task From Clipboard a Copy configuration a logiku k nim.

O uživatelské rozhraní záložky Discovery se stará třída *DiscoveryPanel*, která dále obsahuje *DiscoveryButtonPanel*, což je třída, která se stará o tlačítka na dané záložce. Do třídy *DiscoveryButtonPanel* bude přidáno tlačítko Create Task From Clipboard.

Dále do kontroleru *DiscoveryButtonPanelController*, který obsluhuje tuto třídu, bude přidána nová metoda *createTaskFromClipboard*, která bude zajišťovat obsluhu tlačítka Create Task From Clipboard.

O uživatelské rozhraní záložky Anonymization se stará třída *AnonymizationPanel*, která dále obsahuje třídu *TreeStructuredFilePanel*, která zobrazuje uživatelské rozhraní, pomocí kterého se konfiguruje anonymizace pro strukturované dokumenty. Do této třídy bude přidáno tlačítko Copy configuration.

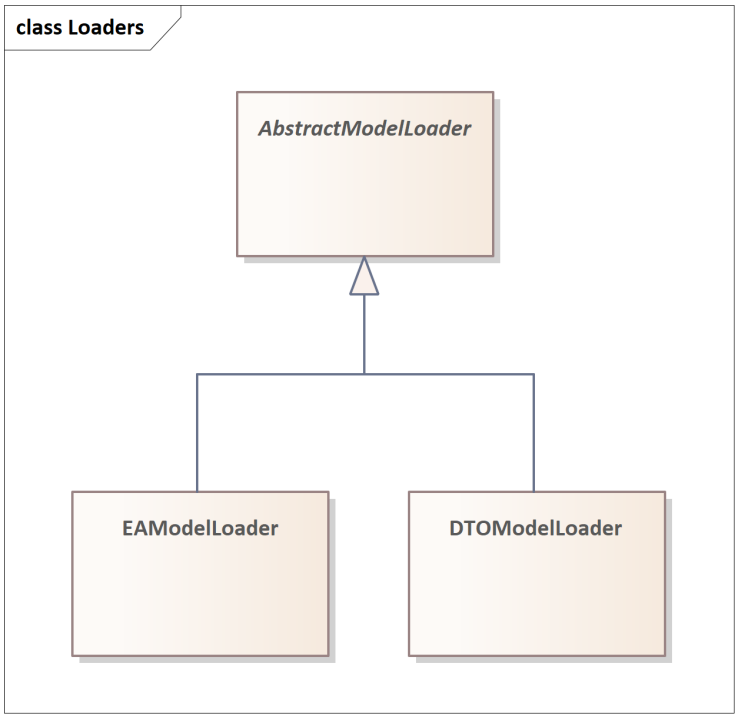
Dále do kontroleru *TreeStructuredAnonymizationController*, který obsluhuje třídu *TreeStructuredFilePanel*, bude přidána nová metoda *handleCopyConfiguration*, která bude zajišťovat obsluhu tlačítka Copy configuration.

3.3.2 Balíček disl-winch-connector

V balíčku *disl-winch-postgresql* bude třída *FilesystemConfigurationToDatabaseConfiguration*. Této třídě ve fázi generování SQL skriptů anonymizační funkce pro strukturované dokumenty předá konfiguraci, kterou uživatel nastavil, a upraví ji do zjednodušené podoby. K tomu, aby byla schopna převést konfiguraci vygenerovanou aplikací na souborovém systému na konfiguraci vhodnější pro databázi, bude potřebovat list anonymizačních tříd. Tato informace není nikde v aplikaci uložena ve vhodné podobě a bude nutné ji načíst z konfigurace a nastavit této třídě jako statický atribut.

Třídě *AbstractModelLoader* (zobrazená na obrázku 3.5), která definuje vlastnosti společné pro všechny třídy načítající konfiguraci, bude přidána abs-

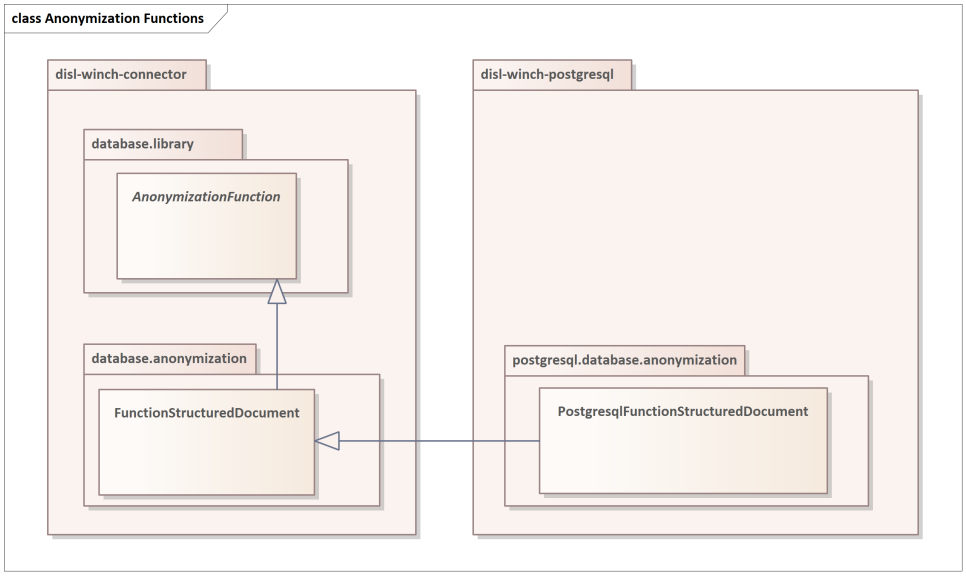
traktní metoda *loadAnonymizationClassMapping*, která bude načítat anonymizační třídy z konfigurace a vracet je jako list. V potomcích, jimiž jsou aktuálně pouze *EAModelLoader* pro načítání konfigurace ze souboru formátu eap a *DTOModelLoader* pro načítání konfigurace ze souboru formátu JSON, bude tato metoda implementována.



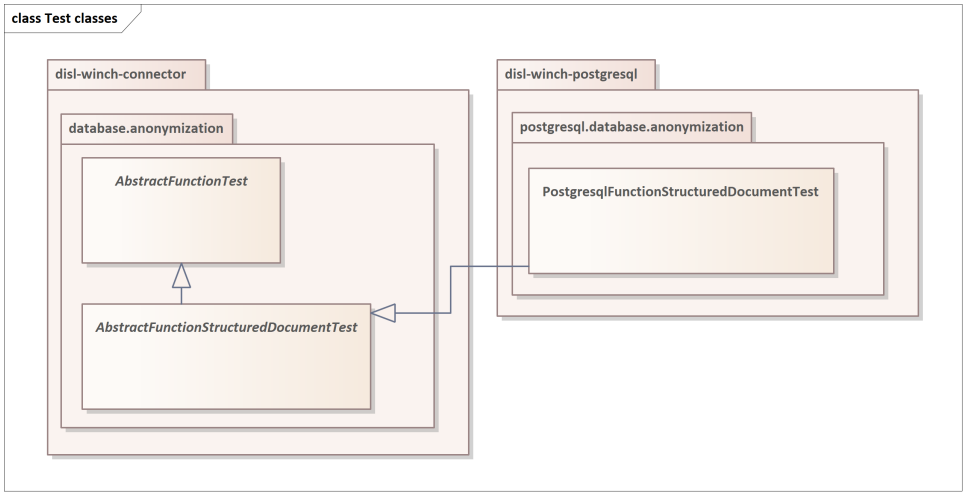
Obrázek 3.5: Hierarchie tříd načítajících konfiguraci

Dále bude nutné do balíčku *database.anonymization* přidat třídu *FunctionStructuredDocument*. V tomto balíčku jsou třídy, které implementují pouze část anonymizačních funkcí, která je společná pro všechny databáze, a dědí od nich třídy, které implementují část anonymizačních funkcí, která je specifická pro konkrétní databázi. Třída *FunctionStructuredDocument* bude tedy implementovat část anonymizace strukturovaného dokumentu, která bude společná pro všechny databáze, a bude od ní dědit třída *PostgresqlFunctionStructuredDocument* (zobrazeno na obrázku 3.6), která bude v balíčku *dislwinch-postgresql* a bude implementovat část specifickou pro databázi PostgreSQL.

Do balíčku *database.anonymization* v balíčku test bude přidána třída *AbstractFunctionStructuredDocumentTest* (zobrazeno na obrázku 3.7). V tomto balíčku se nachází společná část implementace testů, které poté testují jednotlivé implementace dané anonymizační funkce. Hlavní částí implementace této funkce je caseList, který obsahuje společná testovací data.



Obrázek 3.6: Hierarchie tříd anonymizačních funkcí



Obrázek 3.7: Hierarchie tříd testujících anonymizační funkce

Ve třídě *ScriptHelper* bude nutné upravit metodu *evaluate*. Tato metoda přijímá jako parametr *WinchColumn*. Z něj si vezme anonymizační třídu. Z ní získá jméno třídy anonymizační funkce a informace o jejích parametrech. Tyto parametry nastaví jako proměnné do instance třídy *GroovyShell*. Poté zadá do *GroovyShell* výraz, který na dané třídě anonymizační funkce zavolá metodu *getFunctionSql*, která vrátí SQL kód pro zavolání SQL funkce se zadanými parametry. Metoda *getFunctionSql* využívá ve třídách *Postgresql-*

FunctionXML a *PostgresqlFunctionJSON* uložené mapování anonymizačních tříd. Aby metoda evaluate fungovala i pro anonymizaci XML a JSON dokumentů, bude nutné přidat do *AnonymizationFunction*, která je předkem všech anonymizačních funkcí, boolean atribut needMappingEvaluation, který je nastaven na true, pokud daná anonymizační funkce vyžaduje mapování anonymizačních tříd. Nejprve se před voláním metody *getFunctionSql* v *GroovyShell* ověří, zda daná anonymizační funkce vyžaduje mapování anonymizačních tříd. Pokud ano, tak se před voláním metody *getFunctionSql* v *GroovyShell* zavolá metoda setMapping a nastaví se mapování anonymizačních tříd.

3.3.3 Balíček disl-winch-postgresql

V tomto balíčku se nachází implementace anonymizace, která je specifická pro anonymizaci v PostgreSQL. Bude zde nutné přidat SQL skripty a kód v Groovy potřebný k anonymizaci strukturovaných dokumentů v PostgreSQL databázi.

3.3.3.1 Groovy

Groovy část bude zajišťovat vygenerování SQL skriptů a předzpracování konfigurace.

Do balíčku *database.anonymization* bude nutné přidat třídu *PostgresqlFunctionStructuredDocument*, která bude dědit od *FunctionStructuredDocument*. V tomto balíčku jsou třídy, které implementují pouze část anonymizačních funkcí, která je specifická pro konkrétní databázi, v tomto případě pro PostgreSQL. Společnou část pro všechny databáze dědí.

Do balíčku *database.anonymization* v balíčku *test* bude nutné přidat třídu *PostgresqlFunctionStructuredDocumentTest*, která bude testovat třídu *PostgresqlFunctionStructuredDocument*. Konkrétně tím způsobem, že zavolá metodu *getResourcesAsSqlString* na třídě *PostgresqlFunctionStructuredDocument*, která vrátí SQL skript pro vytvoření anonymizační funkce v databázi PostgreSQL. Poté skript předá do metody třídy *PostgresqlTestManager executeInDb*, která ho spustí v databázi, čímž se vytvoří anonymizační funkce v databázi. Poté třída *AbstractFunctionStructuredDocumentTest* v metodě *runTest* vytvořenou funkci otestuje.

3.3.3.2 SQL

SQL část bude zajišťovat úpravu dokumentů přímo v databázi a kontrolu vstupních hodnot.

Jedním z problémů, které SQL část musí vyřešit je, že není dopředu známé, jak bude mít zákazník uložené JSON/XML dokumenty. JSON dokumenty může mít uložené jako varchar, json nebo jsonb. XML dokumenty může mít uložené jako varchar nebo xml. Při řešení tohoto problému lze využít, že PostgreSQL umožňuje přetěžování funkcí, tj. vytvořit stejně pojmenované funkce,

které se budou lišit typem parametru a návratové hodnoty. Zde je popis jednotlivých SQL skriptů, které bude nutné implementovat.

Obalující funkce pro anonymizaci strukturovaného dokumentu se bude v databázi nacházet ve čtyřech variantách, které se budou lišit pouze typem prvního parametru a návratové hodnoty. Typ prvního parametru a návratové hodnoty bude stejný a bude varchar, xml, json nebo jsonb. Tato funkce bude jako parametry brát strukturovaný dokument k anonymizaci a konfiguraci anonymizace vytvořenou přes Winch pro souborový systém a upravenou třídou *FilesystemConfigurationToDatabaseConfiguration*. Tato funkce zkontroluje, že vstupní dokument je validní JSON nebo XML a že předaná konfigurace je validní JSON. Poté převede podle předané konfigurace dokument k anonymizaci na datový typ jsonb nebo xml (zjistí z konfigurace). Následně zavolá příslušnou anonymizační funkci (fc_xml nebo fc_json). Poté vezme výsledek anonymizační funkce a převede ho na původní datový typ a vrátí anonymizovaný strukturovaný dokument.

fc_xml bude brát jako parametry XML dokument k anonymizaci a konfiguraci anonymizace vytvořenou přes Winch pro souborový systém. Vracet bude anonymizovaný XML dokument.

fc_json bude brát jako parametry JSON dokument k anonymizaci a konfiguraci anonymizace vytvořenou přes Winch pro souborový systém. Vracet bude anonymizovaný JSON dokument.

fc_is_json bude brát jako parametr řetězec. Vrátí boolean podle toho, jestli zadaný řetězec reprezentuje validní JSON dokument. Bude sloužit k ověření, že zadaná konfigurace a dokument zadaný jako text jsou validními JSON dokumenty

fc_is_xml bude brát jako parametr řetězec. Vrátí boolean podle toho, jestli zadaný řetězec reprezentuje validní XML dokument. Bude sloužit k ověření, že dokument zadaný jako text je validním XML dokumentem.

3.3.4 Transformace konfigurace

Konfigurace vygenerovaná aplikací pro anonymizaci na souborovém systému obsahuje mnoho informací zbytečných pro anonymizační funkci v databázi, avšak některé naopak chybí. Anonymizační funkce bude potřebovat seznam jednotlivých položek, které se budou anonymizovat. U každé položky bude nutné uložit, jaká anonymizační funkce bude použita k její anonymizaci a parametr, který se předá anonymizační funkci. Dále je nutné, aby obsahovala dotaz buď jako XPath nebo JSONPath podle toho, jestli se jedná o XML nebo JSON dokument.

3.3.4.1 Původní konfigurace

Původní konfigurace vygenerovaná aplikací pro anonymizaci na souborovém systému obsahuje cestu k souboru, název souboru, boolean, zda se má aplikovat i na podsložky, ID aplikace, v rámci které byla konfigurace vytvořena, stromové schéma strukturovaného souboru a formát strukturovaného souboru (JSON nebo XML). Všechny tyto atributy jsou nepotřebné. Dále obsahuje atribut datasets. Jedná se o pole obsahující jednotlivé datasety. Každý dataset obsahuje atribut query, což je dotaz (JSONPath nebo XPath), kterým je definován. Dále obsahuje atribut datasetSubTree popisující strukturu definovanou dotazem. Jedná se o rekurzivní strukturu složenou z uzlů. Každý uzel obsahuje pole uzlů, které jsou jeho přímými potomky, a jméno. Uzel může obsahovat atribut anonymizableItem, který obsahuje atributy anonymizationClass, ve kterém je uloženo jméno anonymizační třídy, a fullQuery, což je dotaz (JSONPath nebo XPath), kterým je anonymizovaná položka definována. Viz ukázka 3.1.

3.3.4.2 Upravená konfigurace

Upravená konfigurace bude též validní JSON objekt. Avšak na rozdíl od původní konfigurace bude mít pouze dva atributy. Prvním z nich bude formát strukturovaného souboru (XML nebo JSON) a datasets, což bude pole obsahující anonymizované položky. Každá anonymizovaná položka bude obsahovat název anonymizační funkce, která bude použita k její anonymizaci, typ druhého parametru anonymizační funkce, hodnotu druhého parametru anonymizační funkce a cestu k anonymizované položce ve formátu JSONPath nebo XPath (podle toho, jestli se bude jednat o JSON nebo XML dokument). Viz ukázka 3.2.

3.4 Omezení

Navržené řešení je omezené tím, že všechny strukturované dokumenty v daném sloupci musí mít stejnou strukturu. Navržené řešení bude fungovat, pokud bude konfigurace nastavena pro strukturovaný dokument, jenž obsahuje všechny nepovinné atributy. Například uživatel bude mít sloupec Kontakty, ve kterém je strukturovaný dokument, jenž bude tvořen JSON objektem se dvěma nepovinnými atributy. Prvním z nich bude e-mailová adresa. Druhým atributem bude telefonní číslo. Pokud bude nastavena anonymizace prostřednictvím dokumentu, ve kterém jsou oba nepovinné atributy, tak anonymizace proběhne v pořádku i pro dokument, který některé z těchto atributů neobsahuje. Atribut, který bude chybět, se nebude anonymizovat.

Pokud by se však v jednom sloupci vyskytovaly strukturované dokumenty, které by měly úplně jinou strukturu (nelišily by se pouze tím, že některé atributy nebo uzly by byly nepovinné), tak takový sloupec nelze anonymizovat

Výpis kódu 3.1: JSON dokument a původní konfigurace

```
{
    "name":"Honza"
}

{
     "destinationURLAndPath":
     "C:\\working\\output\\test.json",
     "applyToSubfolders": "false",
     "relatedApplicationContextId": "7",
     "filePath": "fromClipboard3.json",
     "datasets": [{
          "query": "$.name",
          "datasetSubTree": {
               "isAggregated": false,
               "childNodes": [{
                    "isAggregated": false,
                    "childNodes": [],
                    "anonymizableItem": {
                         "anonymizationClass": "JMENO",
                         "fullQuery": "$.name"
                    },
                    "name": "name"
               }]
          }
     }],
     "treeSchema": {
          "isAggregated": false,
          "childNodes": [{
               "isAggregated": false,
               "childNodes": [],
               "name": "name"
          }]
     },
     "fileType": "JSON"
}
```

Výpis kódu 3.2: Upravená konfigurace

```
{
    "datasets":[
        {
            "anonymizationFunction":"fc_firstname_name",
            "hasSecondParameter":true,
            "secondParameter":"F",
            "fullQuery":"$.name"
        }
    ],
    "fileType":"JSON"
}
```

Výpis kódu 3.3: Dva JSON dokumenty s rozdílnou strukturou a jejich spojení

```
{
    "emails":[
        "anonym@google.com",
        "anonym2@google.com"
    ]
}

{
"email":"anonym@google.com"
}

{
    "emails":[
        "anonym@google.com",
        "anonym2@google.com"
    ],
    "email":"anonym@google.com"
}
```

jednoduchým způsobem. Pokud by oba byly stejného formátu (JSON nebo XML), tak by bylo možné zkopírovat atributy nebo uzly z kořene jednoho dokumentu a vložit je do druhého, čímž by vznikl dokument, který by obsahoval atributy a uzly obou dokumentů. Výsledek spojení by se použil pro vytvoření konfigurace. Zobrazeno na 3.3

3.5 Rozšíření

Jednou z možností, jak by se dalo navržené řešení upravit, je rozšířit modul v Enterprise Architect o možnost stáhnout si do schránky strukturovaný dokument z vybraného sloupce. Uživatel by poté nemusel kromě Enterprise Architect a Winch pro anonymizaci na souborovém systému spouštět navíc program pro práci s databází, např. pgAdmin, aby získal strukturovaný dokument, pro který chce nastavit anonymizaci. Jedním z problémů, který by musel být vyřešen je, jak vybrat dokument, který obsahuje všechny nepovinné atributy nebo uzly.

Kapitola 4

Realizace

Tato kapitola se zabývá implementací chybějících částí anonymizace pro databázi PostgreSQL. Dále se zabývá implementací navrženého způsobu anonymizace strukturovaných dokumentů a odchylkami od návrhu.

4.1 Anonymizační funkce

Chybějící anonymizační funkce byly doplněny. Stačilo pouze zkopírovat kód anonymizačních funkcí z databáze, pro kterou již byly implementovány, a přepsat ho do syntaxe PostgreSQL. Testy pro chybějící anonymizační funkce byly již připraveny.

Během implementace byla objevena chyba v SQL funkci isNumeric. Tato funkce má za úkol rozhodnout, zda řetězec je číslem, tj. je složen pouze z číslic a obsahuje alespoň jeden znak. Chybou bylo, že označila jako číslo i řetězec, který začínal znakem +. Tato chyba byla opravena a byl přidán test této funkce.

4.2 Balíček pattern.codegenerator

Všechny chybějící třídy byly implementovány. Třídy, které již byly implementovány, byly otestovány a zjištěné chyby opraveny. Dále byly implementovány testy pro tyto třídy

4.2.1 Procedury

V ostatních databázích jsou generátory SQL kódu implementovány tak, že některé vrací SQL kód, který se v rámci anonymizace rovnou spustí v databázi, a zbytek vrací SQL kód, který vytvoří v databázi proceduru, která se poté během anonymizace zavolá. Avšak v PostgreSQL jsou procedury dostupné až od verze 11. Z tohoto důvodu generátory SQL kódu, které u ostatních databází vracely kód pro vytvoření procedury, zde vrací kód pro vytvoření

funkce. Jelikož ve většině databází se s funkcemi pracuje jinak než s procedurami, byly nutné další změny. V třídě *PostgresqlDatabaseHelper* byla upravena *getProcedureNameInSql* tak, že vrací volání funkce pomocí PERFORM obalené v anonymním bloku. Funkce se volá příkazem PERFORM, jelikož takto nic nevrací. Příkaz PERFORM je konstrukt PL/pgSQL, a proto musí být obalen anonymním blokem. Pokud by se volala pomocí SELECT, tak i přesto, že ve funkci je RETURNS VOID, tak by něco vrátila, což by vedlo k chybě.

4.2.2 Testy

Z důvodů uvedených v kapitole 2.4.2 jsem testy napsal vlastním způsobem. V každém testu se nejprve provede potřebná inicializace hodnot ze souboru WinchTest.json, který obsahuje konfiguraci anonymizace pro účely testování. Tyto hodnoty se použijí pro vytvoření testovaného generátoru kódu a případně pomocných generátorů, které inicializují databázi. Poté se v rámci transakce v databázi spustí SQL kód vygenerovaný testovaným generátorem a zkontroluje se, že tento kód provedl (cílová tabulka má stejná omezení jako původní, v cílové tabulce jsou správně anonymizované řádky, atd.) to, co měl. Poté se použije příkaz rollback, který všechny změny, které se v rámci transakce v databázi vykonaly, vrátí zpět. Takto se spolehlivě otestuje, že vygenerovaný SQL kód má požadovanou funkcionalitu bez toho, aniž by se narušila testovací databáze.

Kvůli zjednodušení testů na třídy *PostgresqlAnonymizeDeleteSourceDataTableCodeGenerator* a *PostgresqlApplyAsDeleteAnonymizeTableCodeGenerator*, které mažou řádky z databáze, se musí dočasně upravit omezení nad některými tabulkami. Kvůli právům jednotlivých uživatelů v databázi se úprava a obnovení omezení děje mimo transakci.

4.3 Anonymizace strukturovaných dokumentů

Sekce se věnuje implementaci anonymizace strukturovaných dokumentů a odlišnostem od návrhu.

4.3.1 Balíček disl-winch-filesystem

Změny v balíčku byly implementovány dle návrhu.

Metoda *createTaskFromClipboard* třídy *DiscoveryButtonPanelController* nejprve uživateli zobrazí okno, ve kterém z dropdown menu vybere, jestli zadává XML nebo JSON. Poté vezme text ze schránky a ve složce zadané jako zdrojová při vytváření anonymizace vytvoří soubor XML nebo JSON, jehož obsahem bude text ze schránky. Dále zavolá metodu *addTask* na třídě implementující interface *ITaskManager*, čímž se daný soubor přidá do seznamu úloh, aby se zobrazil v seznamu na záložce Anonymization.

Metoda *handleCopyConfiguration* třídy *TreeStructuredAnonymizationController* nejprve zavolá metodu *getCurrentConfiguration* na třídě *TreeDataSelectionService*, která vrátí konfiguraci aktuálně vybraného strukturovaného souboru. Poté vytvoří instanci *ModelFileSaverXml* nebo *ModelFileSaverJson* (podle typu vybraného souboru) a využije ji k získání řetězce, který reprezentuje konfiguraci souboru ve formátu JSON. Dále metoda uživateli zobrazí okno, ve kterém uživatel bude moci zaškrtnout, zda chce daný soubor smazat nebo zachovat. Po potvrzení se uživateli zkopíruje vygenerovaná konfigurace do schránky a daný soubor se případně smaže. Před zkopírováním vygenerované konfigurace jsou z ní odstraněny atributy destinationURLAndPath, applyToSubfolders, relatedApplicationContextId, filePath, treeSchema a fileType, jelikož nejsou potřebné k anonymizaci v databázi.

4.3.2 Balíček disl-winch-connector

Největší změnou oproti návrhu je rozdělení třídy *FunctionStructuredDocument* na třídy *FunctionJSON* a *FunctionXML* (zobrazeno na obrázku 4.1) kvůli přehlednosti. Jednak je pro uživatele jasnější, že *FunctionJSON* slouží k anonymizaci JSON dokumentů a *FunctionXML* k anonymizaci XML dokumentů. Navíc je zdrojový kód přehlednější. Následně je rozdělena i třída *AbstractFunctionStructuredDocumentTest* na *AbstractFunctionJSONTest* a *AbstractFunctionXMLTest*, kde každá z nich obsahuje testovací data pro anonymizace JSON, resp. XML dokumentu.

Další změnou je, že místo třídy *FilesystemConfigurationToDatabaseConfiguration* zajišťují transformaci konfigurace metody v anonymizačních třídách pro konkrétní databáze, tedy ve třídách *PostgresqlFunctionJSON* a *PostgresqlFunctionXML*. Důvodem je, že v jiných databázích by mohlo být nutné konfiguraci transformovat jiným způsobem. Transformace konfigurace je implementována pro JSON a XML dokumenty odděleně, protože každý používají jiný formát pro cestu (JSONPath a XPath).

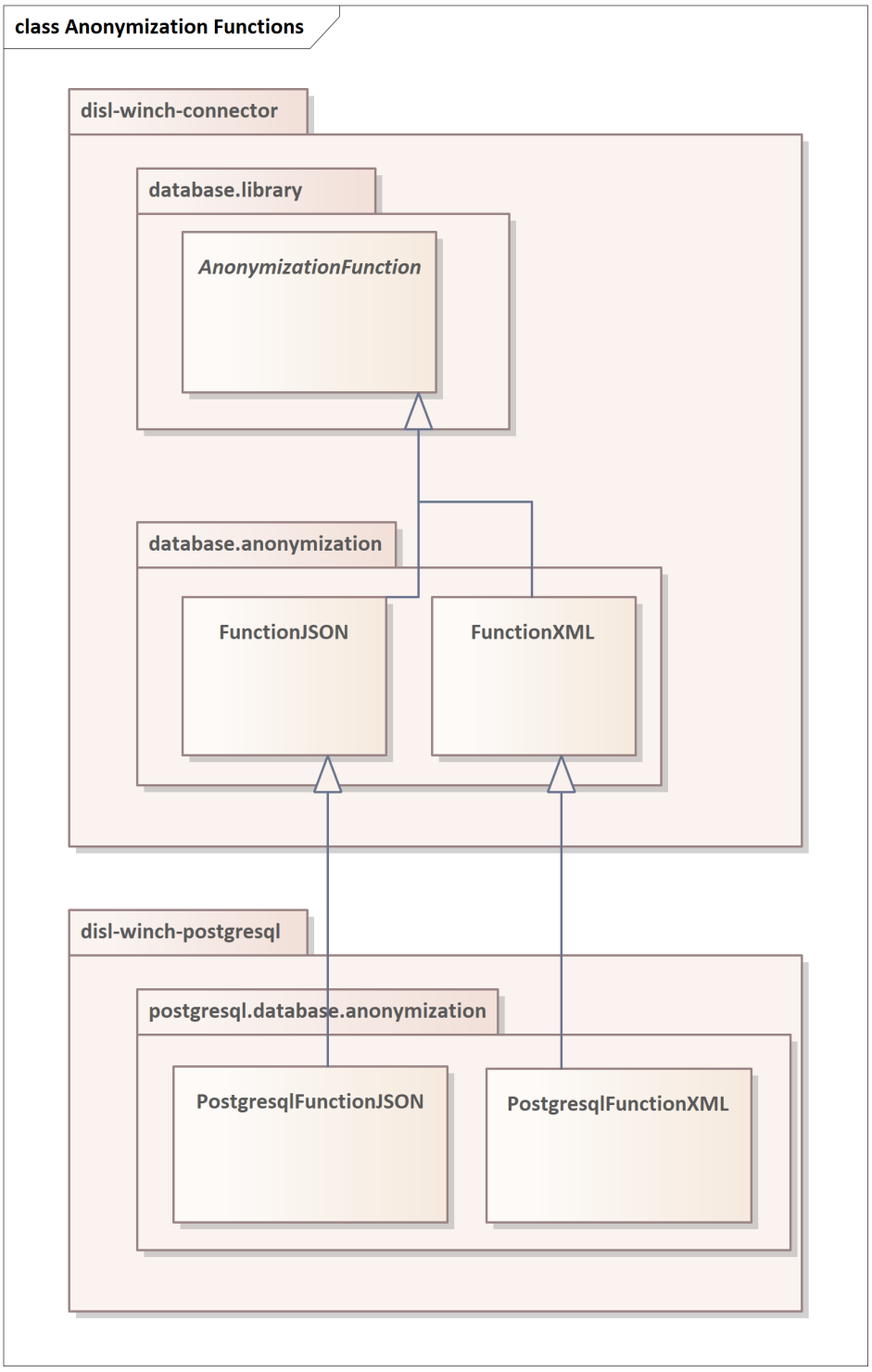
Dále do metody *loadModel* třídy *ModelLoadingUtil*, která načítá konfiguraci aplikace, bylo přidáno načtení anonymizačních tříd z konfigurace a jejich vložení do statické proměnné typu List tříd *FunctionJSON* a *FunctionXML*.

4.3.3 Balíček disl-winch-postgresql

Část popisuje změny v balíčku *disl-winch-postgresql* oproti návrhu. Nejprve se zabývá změnami v Groovy kódu a poté změnami v SQL.

4.3.3.1 Groovy

Třída *PostgresqlFunctionStructuredDocument*, jež měla dědit od třídy *FunctionStructuredDocument*, byla rozdělena na *PostgresqlFunctionJSON*, která dědí od *FunctionJSON*, a *PostgresqlFunctionXML*, jež dědí od *FunctionXML*.



Obrázek 4.1: Hierarchie implementovaných anonymizačních funkcí

Následně byla testovací třída *PostgresqlFunctionStructuredDocumentTest* rozdělena na třídy *PostgresqlFunctionJSONTest* a *PostgresqlFunctionXMLTest* (zobrazeno na obrázku 4.2).

Dále třídy *PostgresqlFunctionJSON* a *PostgresqlFunctionXML* mají metody na transformaci konfigurace pro souborový systém do formy vhodné pro databázi, k čemuž slouží metoda *transformConfiguration*. V kapitole 3.3.4 je popsána transformace konfigurace. Metodě *transformConfiguration* je předán jako parametr řetězec, jenž reprezentuje konfiguraci získanou z aplikace Winch pro souborový systém ve formátu JSON. Nejprve odstraní obalení řetězce. Poté řetězec převede na JSON objekt, ze kterého vezme atribut datasets, který je typu JSONArray. Dále z každého prvku tohoto pole vezme atribut datasetSubTree a z něj vezme atribut childNodes, což je pole potomků daného uzlu, na který zavolá metodu *getAnonymizableItems*.

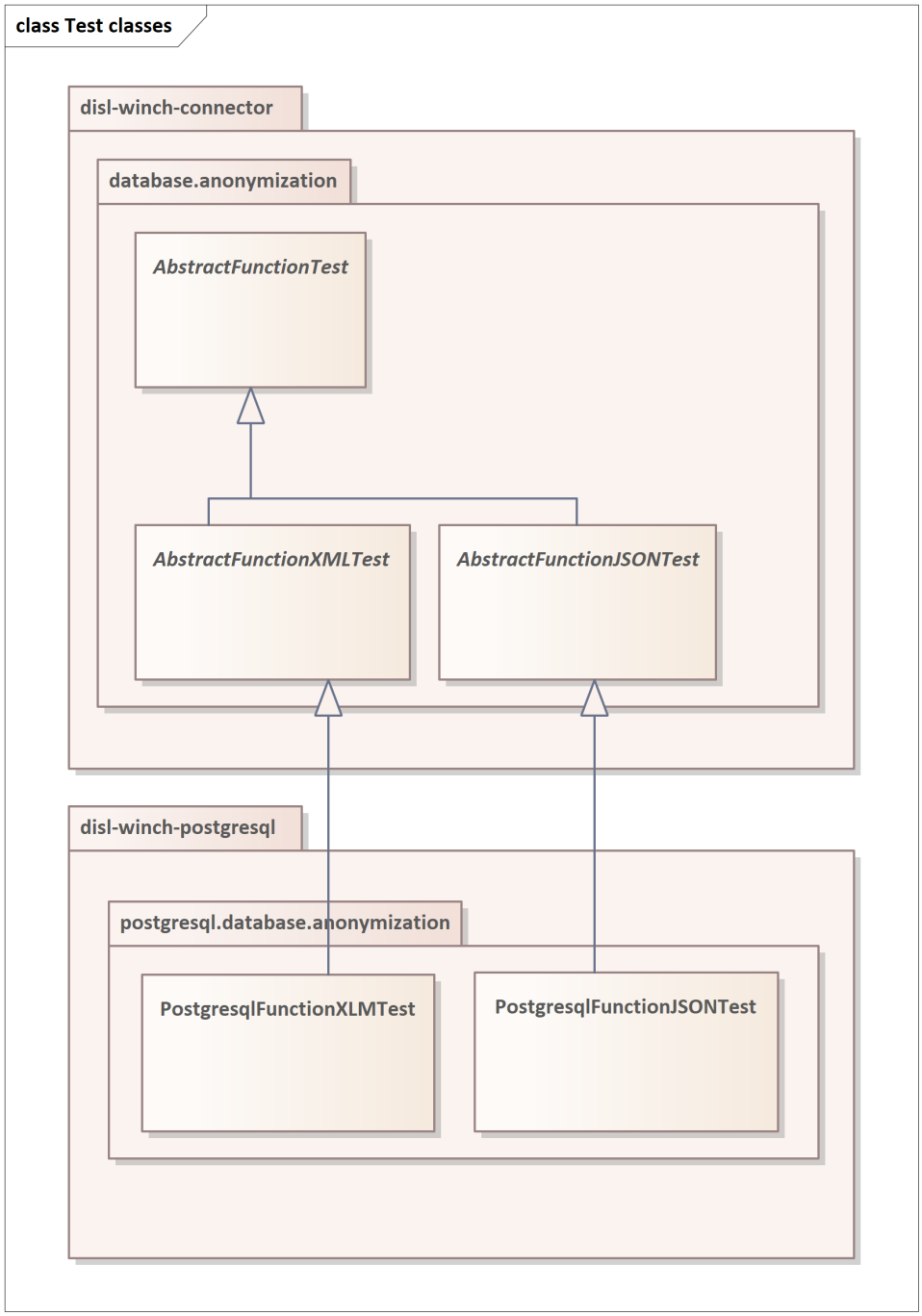
Jedná se o rekurzivní metodu, která projde předané pole uzlů a vrátí řetězec reprezentující JSONArray anonymizovaných položek. U každého uzlu zkontroluje, zda obsahuje atribut anonymizableItem, který reprezentuje položku k anonymizování. Pokud ji obsahuje, tak ji předá jako parametr metodě *transformAnonymizableItem*, která ji transformuje do formy vhodnější pro anonymizaci v databázi.

Metoda *transformAnonymizableItem* nejprve upraví atribut fullQuery. Pokud se jedná o JSON dokument, řetězec fullQuery rozdělí podle znaku „.“ a výsledné pole řetězců vloží jako hodnotu do fullQuery. V případě XML dokumentu nahradí první lomítko za dvě lomítka a odstraní podřetězec „/text()“ kvůli provedení anonymizace v SQL. Pro JSON i XML dokumenty platí, že název anonymizační třídy nahradí za název anonymizační funkce a dále přidá anonymizableItem atributy secondParameterType, který specifikuje typ druhého parametru, a secondParameter, který specifikuje hodnotu druhého parametru anonymizační funkce. Tyto informace získá tak, že zavolá svoji metodu *findAnonymizationClass*, které předá řetězec reprezentující název anonymizační třídy a následně získá objekt typu WinchAnonymizationClass, který všechny potřebné informace obsahuje.

4.3.3.2 SQL

Stejně jako v části Groovy byla třída *PostgresqlFunctionStructuredDocument* rozdělena na *PostgresqlFunctionJSON* a *PostgresqlFunctionXML*, byla rozdělena anonymizace i v SQL části.

Anonymizace JSON dokumentu v PostgreSQL databázi probíhá tak, že je nejprve předán JSON dokument a konfigurace anonymizace SQL funkci fc_anon_json_document. Tato funkce je přetížená, je definovaná třikrát s tím, že se liší typ prvního parametru (způsob uložení JSON dokumentu, který může být uložen jako json, jsonb nebo text). Funkce ověří, že zadaný dokument je validní JSON dokument pomocí SQL funkce fc_is_json. Poté zavolá funkci fc_anon_json_document_starter, které předá zadaný dokument a konfiguraci.



Obrázek 4.2: Hierarchie tříd testujících anonymizační funkce

Funkce nejprve zkontroluje, že zadaná konfigurace je validní JSON dokument pomocí SQL funkce fc_is_json (nekontroluje se, že obsahuje všechny

atributy, jelikož uživatel bude konfiguraci pouze kopírovat a pokud ji zkopíruje špatně, tak se nebude jednat o validní JSON dokument). Poté z konfigurace vezme pole datasets. Z každého datasetu vezme pole fullQuery a z něj první prvek. Pokud prvek je název atributu typu pole, tak projde všechny jeho prvky a zavolá funkci fc_anon_json_document_recursive a předá jí anonymizovaný dokument, pole obsahující první uzel na cestě k anonymizovanému prvku a jeho pořadí v poli, pole obsahující zbývající uzly, název anonymizační funkce, typ druhého parametru a hodnotu druhého parametru. Pokud prvek není název atributu typu pole, tak zavolá funkci fc_anon_json_document_recursive a předá jí anonymizovaný dokument, pole obsahující první uzel na cestě k anonymizovanému prvku, pole obsahující zbývající uzly, název anonymizační funkce, typ druhého parametru a hodnotu druhého parametru.

Funkce nejprve zkontroluje, zda je již v cílovém uzlu. Pokud již je v cílovém uzlu, použije pole current_path obsahující seznam uzlů k cílovému uzlu ke kontrole, že anonymizovaný prvek není null. Pokud je null, tak vrátí nezměněný dokument. Jinak ho anonymizuje a vrátí upravený dokument. Pokud není v cílovém uzlu, zkontroluje, zda další prvek cesty k anonymizovanému prvku je pole. Pokud se jedná o pole, projde všechny jeho prvky a zavolá funkci fc_anon_json_document_recursive a předá jí anonymizovaný dokument, již navštívené uzly na cestě k anonymizovanému prvku spolu s aktuálním prvkem a jeho pořadím v poli, pole obsahující zbývající uzly, název anonymizační funkce, typ druhého parametru a hodnotu druhého parametru. Pokud není, zavolá funkci fc_anon_json_document_recursive a předá jí anonymizovaný dokument, pole obsahující již navštívené uzly na cestě k anonymizovanému prvku, pole obsahující zbývající uzly, název anonymizační funkce, typ druhého parametru a hodnotu druhého parametru. Tato funkce je rekurzivní, protože PostgreSQL funkce pro úpravu JSON dokumentů umožňuje měnit pouze jeden prvek.

Anonymizace XML dokumentu v PostgreSQL databázi probíhá tak, že je nejprve funkci fc_anon_xml_document předán XML dokument a konfigurace anonymizace. Tato funkce je přetížená, je definovaná dvakrát s tím, že se liší typ prvního parametru (způsob uložení XML dokumentu, který může být uložen jako xml nebo text). Poté zavolá funkci fc_anon_xml_document_starter, které předá zadaný dokument a konfiguraci.

Funkce nejprve zkontroluje, že předaná konfigurace je validní JSON dokument pomocí SQL funkce fc_is_json a pomocí funkce fc_is_xml zkontroluje, že předaný dokument je validní. Poté z konfigurace vezme pole datasets. Pro každý dataset zavolá funkci fc_anon_xml_document_helper, které předá anonymizovaný dokument, fullQuery (XPath), jméno anonymizační funkce, typ druhého parametru anonymizační funkce a hodnotu druhého parametru anonymizační funkce.

Funkce fc_anon_xml_document_helper je na rozdíl od ostatních napsána v jazyce PL/Python. Pomocí předaného XPath vybere uzly z XML dokumentu, vezme z nich anonymizované hodnoty (atributy nebo text), aplikuje na ně předanou anonymizační funkci s parametry a vrátí upravený dokument.

4.3.4 Testování

Část se věnuje testování anonymizačních funkcí pro strukturované dokumenty a chybám, které byly při jejich testování objeveny.

4.3.4.1 Anonymizační funkce pro strukturované dokumenty

Pro každou anonymizační funkci byly napsány testy. Pro XML dokumenty se testuje anonymizace textu v jednotlivých uzlech i anonymizace hodnot atributů jednotlivých uzlů. Dále se testuje, pokud se anonymizuje nepovinný uzel. Navíc je otestováno zadání nevalidního XML dokumentu.

Pro JSON dokumenty se testuje anonymizace hodnot atributů. Dále se testuje, pokud se anonymizuje nepovinný atribut. Navíc je otestováno zadání nevalidního JSON dokumentu.

Dále byly upraveny soubory s testovacími daty a testovací konfigurace anonymizace WinchTest.eap a WinchTest.json tak, aby se během testování celé anonymizace otestovaly i anonymizační funkce pro strukturované dokumenty. Do testovacích dat byla přidaná tabulka s JSON dokumenty a tabulka s XML dokumenty. V testovací konfiguraci bylo nastaveno, aby se sloupce těchto tabulek anonymizovaly pomocí anonymizačních funkcí pro strukturované dokumenty.

4.3.4.2 Zjištěné chyby již implementovaných testů

Při přidání testů pro anonymizační funkce se změnilo pořadí provádění testů, a tak bylo zjištěno, že mezi testy je skrytá závislost, v jakém pořadí se provádějí. Aktuálně byly testy opraveny pouze tak, aby procházely, jelikož jiná práce se zabývá společným řešením pro všechny databáze.

Dále se zjistilo, že testy, které dědily z *AbstractDiscovererTest*, po sobě neuzavíraly spojení s databází. Následkem této chyby mohla nastat (a nastala) situace, kdy jeden z testů nechal otevřené spojení s databází, ve kterém měl neukončenou transakci, kterou nepotvrdil ani nevrátil. Pokud následující test potřeboval upravit v databázi něco, s čím se pracovalo v neukončené transakci, vedlo to k zablokování databáze, jelikož se čekalo, až se předchozí transakce dokončí. Bylo velmi obtížné tuto chybu objevit, protože když byly testy spuštěny samostatně, tak fungovaly.

4.3.5 Omezení

Část se zabývá omezeními implementovaného řešení. Nejprve se zabývá nemožností anonymizace JSON dokumentu, který obsahuje tečku v názvu atributu. Dále se zabývá nutností shodného nastavení anonymizačních tříd v aplikaci Winch pro souborový systém a pro databáze.

Výpis kódu 4.1: JSON dokument s tečkou

```
{
    "name.firstname":"Honza"
}
```

Výpis kódu 4.2: Ukázkový JSON dokument

```
{
    "name":"Honza"
}
```

4.3.5.1 Tečka v názvu atributu v JSON dokumentu

Implementované řešení nijak neošetřuje situaci, pokud je v názvu atributu JSON dokumentu tečka, což je problém, jelikož tečka je speciálním znakem v JSONPath. Avšak tato situace není nijak ošetřena v implementaci anonymizace JSON dokumentů na souborovém systému, jež poskytuje konfiguraci pro anonymizaci. Například JSON dokument v ukázce 4.1 nelze se současnou implementací anonymizovat.

4.3.5.2 Nastavení anonymizačních tříd

Při nastavování anonymizace strukturovaných dokumentů je nutné nastavit všechny anonymizační třídy, které se při anonymizaci strukturovaných dokumentů využijí stejně, jak ve Winch pro souborový systém, tak i pro databázi. V konfiguraci vygenerované na souborovém systému jsou pouze názvy anonymizačních tříd. Při exportování konfigurace z Winch pro souborový systém nelze názvy anonymizačních tříd přepsat na anonymizační funkce s jejich parametry, což by tento problém odstranilo. V ukázce 4.2 je JSON dokument, který obsahuje JSON objekt s atributem name. Pokud se nastaví v aplikaci Winch pro souborový systém, že k anonymizaci se použije anonymizační třída JMENO, tak vygeneruje konfiguraci v ukázce 4.3.

V této konfiguraci je atribut anonymizableItem, který má atribut anonymizationClass, který má hodnotu JMENO. Pokud se tato konfigurace použije ve Winch Add-in, je nutné, aby v něm byla nadefinována anonymizační třída JMENO, jelikož exportovaná konfigurace obsahuje pouze název anonymizační třídy, ale už neobsahuje informace o její konfiguraci (např. jaká anonymizační funkce v databázi se má využít a jaký se má použít druhý parametr).

Výpis kódu 4.3: Vygenerovaná konfigurace

```
{
    "datasets":[
        {
            "query":"$.name",
            "datasetSubTree":{
                "isAggregated":false,
                "childNodes":[
                    {
                        "name":"name",
                        "isAggregated":false,
                        "childNodes":[

                        ],
                        "anonymizableItem":{
                            "anonymizationClass":"JMENO",
                            "fullQuery":"$.name"
                        }
                    }
                ]
            }
        }
    ]
}
```

4.4 Testování

Během analýzy bylo zjištěno, že chybí návod, jak si nastavit testovací prostředí pro spouštění testů testujících anonymizaci v databázi PostgreSQL. Jelikož bylo velmi pracné zjistit, jak ho nastavit, byl sepsán návod, jak si testovací prostředí nastavit.

Dále bylo zjištěno, že test na inicializaci databáze vypíše do konzole informaci o tom, že inicializace databáze se nezdařila, ale označí se jako úspěšný. Toto bylo opraveno.

Kapitola 5

Testování výkonu implementace

Kapitola se zabývá testováním výkonu anonymizace strukturovaných dokumentů uložených v databázi. Testuje se rychlost anonymizace přímo v databázi a rychlost anonymizace pomocí implementace na souborovém systému. Účelem testování je ověřit, zda zvolené řešení, které anonymizuje strukturované dokumenty přímo v databázi, je rychlejší, než kdyby se využila již implementovaná anonymizace strukturovaných dokumentů na souborovém systému.

5.1 Způsob testování

V databázi byla vytvořena tabulka se sloupcem typu text, ve kterém byly uloženy JSON dokumenty. Ve všech řádcích byl stejný JSON dokument, který je zachycen na 5.1. Lišil se pouze tím, že v poli book bylo 12 položek. Tabulka měla celkem 100 000 řádků. U dokumentů se anonymizovala pouze jména autorů pomocí funkce fc_firstname_name. Obdobně byla vytvořena testovací data pro XML.

Výpis kódu 5.1: Testovaný JSON dokument

```
{
   "catalog":{
      "book":[
         {
            "@id":"bk101",
            "title":"XML Developer's Guide",
            "genre":"Computer",
            "price":"44.95",
            "publish_date":"2000-10-01",
            "description":"An in-depth look at create."
         },
         {
            "@id":"bk102",
```

```
            "author":"Ralls, Kim",
            "title":"Midnight Rain",
            "genre":"Fantasy",
            "price":"5.95",
            "publish_date":"2000-12-16",
            "description":"A former architect battles."
        }
    ]
  }
}
```

Poté byl v databázi spuštěn SQL příkaz, který vybral všechny řádky z tabulky, anonymizoval JSON dokumenty pomocí implementované anonymizační funkce a vložil je do cílové tabulky.

Pro otestování anonymizace strukturovaných dokumentů uložených v databázi s využitím implementace anonymizace na souborovém systému bylo nutné napsat obalující kód. Nejprve pomocí SELECT příkazu vybere z databáze JSON dokumenty. Dále pro každý dokument vytvoří soubor, do kterého ho uloží. Poté se nastaví, jak se má anonymizovat a anonymizuje se. Nakonec se načte obsah anonymizovaného souboru a nahraje se do cílové tabulky v databázi. Obdobně se testovalo XML. Prostředí, ve kterém probíhalo testování mělo následující parametry:

- Java 8 32bit
- PostgreSQL 10
- OS - Windows 10 Home 64bit
- Procesor - AMD Ryzen 5 3600
- Operační paměť - 16GB, 3200MHz
- Disk - Samsung SSD 960 EVO 500GB

5.2 Výsledky

Z naměřených hodnot v tabulkách 5.1 a 5.2 lze vyčíst, že doba anonymizace strukturovaných dokumentů roste lineárně s počtem anonymizovaných dokumentů. Pro souborový systém nebylo provedeno měření pro 10 000 řádků, jelikož by trvalo přibližně 30 minut, ani 100 000 řádků, pro něž by trvalo přibližně 5 hodin. Dále lze vyčíst z měření, že anonymizace strukturovaných dokumentů s pomocí implementace pro souborový systém je výrazně pomalejší.

S největší pravděpodobností je to způsobeno tím, že pro každý anonymizovaný řádek se musí vytvořit soubor, poté se anonymizuje (čtení a zápis

do souboru) a nakonec se z něj musí načíst výsledek a vložit do databáze. Vytváření souboru, čtení a zápis jsou velmi pomalé operace. Další příčinou je, že databáze byla schopna zpracování SQL příkazu rozdělit mezi více jader procesoru. Při reálném využití by pravděpodobně byla anonymizace s pomocí implementace pro souborový systém ještě pomalejší, jelikož by se data z databáze musela posílat po síti (při testování databáze běžela na stejném počítači), což by při velkých objemech dat anonymizaci dále zpomalilo.

Navíc v databázi může být tolik strukturovaných dokumentů, že se všechny nevejdou do paměti a při pokusu o jejich načtení z databáze dojde k OutOfMemoryError. Bylo by tedy nutné data z databáze načítat po částech.

Tabulka 5.1: Test výkonu JSON, doba běhu v sekundách

Počet řádků	100	1 000	10 000	100 000
Souborový systém	19.5	188	x	x
Databáze	0.18	1.15	11.3	112.1

Tabulka 5.2: Test výkonu XML, doba běhu v sekundách

Počet řádků	100	1 000	10 000	100 000
Souborový systém	20.9	199	x	x
Databáze	0.11	0.58	4.9	49.5

Závěr

Cílem práce bylo analyzovat aktuální stav implementace anonymizace osobních údajů v databázi PostgreSQL v nástroji Winch a implementovat identifikované chybějící části. Dalším cílem bylo analyzovat nástroje databáze PostgreSQL pro práci s XML a JSON dokumenty a na základě této analýzy navrhnout a implementovat rozšíření anonymizace v databázi PostgreSQL o možnost anonymizovat XML a JSON dokumenty.

Implementace anonymizace v databázi PostgreSQL byla doplněna o identifikované chybějící části a nyní je ekvivalentní s implementacemi pro ostatní databáze. Na základě analýzy nástrojů databáze PostgreSQL bylo navrženo a implementováno rozšíření umožňující anonymizovat XML a JSON dokumenty uložené v databázi PostgreSQL. Jelikož nástroje databáze PostgreSQL nebyly dostatečné pro práci s XML dokumenty, což bylo diskutováno v kapitole 2.7.3, tak pro anonymizaci XML dokumentů bylo nutné využít rozšíření PL/Python. Nově implementované třídy a metody jsou pokryty testy. Implementované řešení bylo otestováno z hlediska rychlosti proti variantě, kdy by samotná anonymizace dokumentů probíhala na souborovém systému, a bylo zjištěno, že implementované řešení je výrazně rychlejší.

Implementace bakalářské práce byla již integrována do hlavní vývojové větve Winch.

Literatura

[1] Perner, P.: *Refaktoring a metodika testování nástroje pro anonymizaci dat*. Praha, 2022. Bakalářská práce. Fakulta informačních technologií. Ing. Jiří Mlejnek.

[2] Zákon č. 110/2019 Sb. o zpracování osobních údajů.: In: *Sbírka zákonů*. 24. 4. 2019.

[3] CloverDX: *Data Anonymization: 7 Essential Use Cases* [online]. 2019, [cit. 2023-04-06]. Dostupné z: `https://www.cloverdx.com/blog/data-anonymization-7-essential-use-cases`

[4] Refsnes Data: *XML and XPath* [online]. [cit. 2023-02-07]. Dostupné z: `https://www.w3schools.com/xml/xml_xpath.asp`

[5] WWW Consorcium: *XML Essentials* [online]. [cit. 2023-02-07]. Dostupné z: `https://www.w3.org/standards/xml/core`

[6] Refsnes Data: *XML Tutorial* [online]. [cit. 2023-02-07]. Dostupné z: `https://www.w3schools.com/xml/default.asp`

[7] Refsnes Data: *XPath Tutorial* [online]. [cit. 2023-04-06]. Dostupné z: `https://www.w3schools.com/xml/xpath_intro.asp`

[8] WWW Consorcium: *XML Path Language (XPath) 3.1* [online]. [cit. 2023-04-06]. Dostupné z: `https://www.w3.org/TR/xpath-31/`

[9] JSON.org: *Introducing JSON* [online]. [cit. 2023-02-07]. Dostupné z: `https://www.json.org/json-en.html`

[10] Refsnes Data: *JSON - Introduction* [online]. [cit. 2023-02-07]. Dostupné z: `https://www.w3schools.com/js/js_json_intro.asp`

[11] Goessner, S.: *JSONPath - XPath for JSON* [online]. 2007, [cit. 2023-04-06]. Dostupné z: `https://goessner.net/articles/JsonPath/`

[12] The PostgreSQL Global Development Group: *8.13. XML Type* [online]. [cit. 2023-02-10]. Dostupné z: https://www.postgresql.org/docs/8.3/datatype-xml.html

[13] The PostgreSQL Global Development Group: *9.14. XML Functions* [online]. [cit. 2023-02-10]. Dostupné z: https://www.postgresql.org/docs/8.3/functions-xml.html

[14] Tada AB: *Building PL/Java* [online]. [cit. 2023-03-15]. Dostupné z: https://tada.github.io/pljava/build/build.html

[15] The PostgreSQL Global Development Group: *Chapter 46. PL/Python — Python Procedural Language* [online]. [cit. 2023-03-16]. Dostupné z: https://www.postgresql.org/docs/15/plpython.html

[16] The PostgreSQL Global Development Group: *8.14. JSON Types* [online]. [cit. 2023-02-10]. Dostupné z: https://www.postgresql.org/docs/9.4/datatype-json.html

[17] The PostgreSQL Global Development Group: *9.15. JSON Functions and Operators* [online]. [cit. 2023-02-10]. Dostupné z: https://www.postgresql.org/docs/9.5/functions-json.html

Příloha **A**

Seznam použitých zkratek

DTD Document Type Definition

GUI Graphical user interface

JSON Extensible markup language

SQL Structured Query Language

XML Extensible markup language

Příloha **B**

Obsah zip archivu

- `readme.txt` ..... stručný popis obsahu archivu
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `BP_Prusek_Tomas_2023.pdf` ..... text práce ve formátu PDF