Practical Regulatory Compliance in Database Systems

Alexander Nykjær, Ane Søgaard Jørgensen,
and Jakob Sønderby Kristensen

Summary

The General Data Protection Regulation (GDPR), which came into effect in 2018, regulates how natural or legal persons such as companies, public authorities, agencies, or other bodies process the personal data of natural persons [European Commission, 2016]. As a result, many companies have had to rework their approach to processing personal data, as typical database practices, such as storing data forever or reusing data for different purposes, are at odds with GDPR [Agarwal et al., 2022] [Shastri et al., 2019a] [Shastri et al., 2019b].

The objective of this project is to design and prototype a Data Protection Compliance Tool (DPCT) that supports companies in making their existing database systems GDPR compliant. This is achieved by allowing a user of DPCT to modify an existing database system in such a manner, that the following requirements are fulfilled:

1. Personal data is only processed for legitimate and specific purposes.
2. Personal data is only processed on a legal basis.
3. Personal data is associated with an individual.
4. Personal data is only stored for as long as it is necessary to fulfil a purpose, and it is deleted as soon as that is no longer the case.
5. Operations affecting personal data are logged such that regulators can inspect compliance.

DPCT supports the fulfillment of these requirements by enabling its users to associate all personal data with metadata that define the purposes for storing it, its associated individual, and when it should be deleted.

Existing studies propose much of the functionality needed to fulfill the requirements. Kraska et al. [2019] propose a system that fulfills many of the requirements, but provide only an abstract design and no implementation. The contribution of DPCT is to implement some of this functionality and to combine it with the data vacuuming, presented by Palmer and Srikandarajah [2022], in order to provide companies with a tool that helps them ensure that all personal data in their existing database systems is compliant with GDPR, and is removed as soon as that is no longer the case. Although there are preconditions for DPCT to be applicable, existing work on schema evolution by Curino et al. [2008, 2010] and Nykjær et al. [2023] can be used to fulfill these preconditions.

DPCT allows a user to register columns in an existing database as containing personal data. After a column is registered as containing personal data, DPCT can give an overview of the

metadata needed to ensure that the personal data is stored in compliance with GDPR. DPCT enables its users to register the metadata and vacuuming policies needed to document that the personal data is being processed for legitimate and specific purposes, can be associated with a natural person, and is deleted when it is no longer being processed for a valid purpose.

The prototype of DPCT fulfills the requirements defined for the system, except for the requirement *Legal processing*, which requires that personal data must be stored with a legal basis. It is only possible to determine the legal basis on which personal data is being processed using DPCT if the legal basis is a legal obligation.

The design and implementation of DPCT presented in this report can assist companies in complying with specific aspects of GDPR. However, this design can be extended to provide support for additional aspects. Several extensions are presented as future work, including fulfilling the missing requirement, providing built-in support for the rights of individuals, and extending logging to support verification of external changes to personal data.

Agarwal, A., George, M., Jeyaraj, A. and Schwarzkopf, M. [2022], 'Retrofitting GDPR compliance onto legacy databases', *Proceedings of the VLDB Endowment.* **15**(4), 958–970.
URL: *https://doi.org/10.14778/3503585.3503603*

Curino, C. A., Moon, H. J., Deutsch, A. and Zaniolo, C. [2010], 'Update rewriting and integrity constraint maintenance in a schema evolution support system: PRISM++', *Proceedings of the VLDB Endowment* **4**(2), 117–128.

Curino, C. A., Moon, H. J. and Zaniolo, C. [2008], 'Graceful database schema evolution: The prism workbench', *Proceedings of the VLDB Endowment* **1**(1), 761–772.

European Commission [2016], '2016 reform of eu data protection rules'.
URL: *https://eur-lex.europa.eu/eli/reg/2016/679/oj*

Kraska, T., Stonebraker, M., Brodie, M., Servan-Schreiber, S. and Weitzner, D. [2019], Schengendb: A data protection database proposal, *in* 'Heterogeneous Data Management, Polystores, and Analytics for Healthcare', Springer International Publishing, pp. 24–38.
URL: *https://doi.org/10.1007/978-3-030-33752-0_2*

Nykjær, A. M., Jørgensen, A. S. and Kristensen, J. S. [2023], 'Towards practical regulatory compliance in database systems'.

Palmer, A. H. and Srikandarajah, S. [2022], 'Design and implementation of a system for rule-based data retention compliance'. Report not publicly available.

Shastri, S., Banakar, V., Wasserman, M., Kumar, A. and Chidambaram, V. [2019], 'Understanding and benchmarking the impact of GDPR on database systems', *arXiv preprint arXiv:1910.00728* .

Shastri, S., Wasserman, M. and Chidambaram, V. [2019], The seven sins of personal-data processing systems under GDPR, *in* '11th USENIX Workshop on Hot Topics in Cloud Computing', USENIX Association.
URL: *https://www.usenix.org/conference/hotcloud19/presentation/shastri*

Practical Regulatory Compliance in Database Systems

Master's Thesis

CS-22-DT-10-03

Aalborg University
Computer Science

AALBORG UNIVERSITY
STUDENT REPORT

Computer Science
Aalborg University
http://www.aau.dk

Title:
Practical Regulatory Compliance in Database Systems

Theme:
Database Technology

Project Period:
Spring Semester 2023

Project Group:
CS-22-DT-10-03

Participants:
Alexander Nykjær
Ane Søgaard Jørgensen
Jakob Sønderby Kristensen

Supervisor:
Christian S. Jensen

Page Numbers without appendix: 62
Page Numbers with appendix: 82

Date of Completion:
June 15, 2023

Abstract:

The General Data Protection Regulation (GDPR), which came into effect in 2018, regulates the processing of personal data. This meant that companies have had to rework their approach to processing personal data. Understanding of, and compliance with, GDPR remains a problem in 2023.
This project analyses GDPR and existing work to determine the effect it has on database systems and proposes five requirements for a system that can help companies make their existing database systems GDPR compliant. A tool, called Data Protection Compliance Tool (DPCT), that satisfies four of these requirements is then proposed. DPCT enables its users to register metadata and vacuuming policies needed to document that personal data is being processed for legitimate and specific purposes, can be associated with a natural person, and is deleted when it is no longer being processed for a valid purpose.
A prototype of DPCT is implemented and is evaluated using a database for a fictional web shop storing personal data about customers. Finally, extensions to DPCT are presented that provide additional GDPR support.

Contents

Preface

This report was written by group CS-22-DT-10-03 as a master's thesis in Software at Aalborg University.

The citations in this report are based on the Harvard standard [Mendeley, 2021], where parentheses are replaced with square brackets. GDPR articles are referred to by their article number, and possibly paragraph and point.

We want to thank our supervisor Christian S. Jensen, professor at the Department of Computer Science, for his help during the semester.

The public source code for the project can be found at:

`https://github.com/P9-P10/DataProtectionComplianceTool`.

Aalborg University, June 15, 2023

Alexander Nykjær
<anykjal18@student.aau.dk>

Ane Søgaard Jørgensen
<asja18@student.aau.dk>

Jakob Sønderby Kristensen
<jkr18@student.aau.dk>

Chapter 1

Introduction

Section 1.1 presents the motivation and problem statement for the project, Section 1.2 describes the context of the project by presenting existing studies that analyse the effects of GDPR on database systems, Section 1.3 formalises the requirements for DPCT based on GDPR and the existing studies, and Section 1.4 presents a database system that is used as a running example throughout the report. Finally, the outline of this report is presented in Section 1.5.

1.1 Motivation and Problem Statement

The General Data Protection Regulation (GDPR), which came into effect in 2018, regulates how natural or legal persons such as companies, public authorities, agencies, or other bodies process the personal data of natural persons [European Commission, 2016]. As a result, many companies have had to rework their approach to processing personal data, as typical database practices, such as storing data forever or reusing data for different purposes, are at odds with GDPR [Agarwal et al., 2022; Shastri et al., 2019a,b]. Personal data, in the context of GDPR, is defined in Article 4(1) as *"Any information relating to an identified or identifiable natural person"* and processing is defined in Article 4(2) as *"Any operation [...] which is performed on personal data [...] such as [...] storage, [...] alteration, [...] erasure"*. Throughout the report, when referring to a natural person, the term *individual* is also used.

It is still a problem in 2023 for companies to understand and comply with GDPR [Datatilsynet, 2023]. The biggest fines, totalling 14.9M DKK, have been given to:

- Gyldendal, for not deleting personal data regarding 685,000 book club members, was fined 1.1M DKK [Datatilsynet, 2022b].
- Arp-Hansen Hotel Group A/S, for not deleting 500,000 customer profiles, was fined 1.1M DKK [Datatilsynet, 2020].
- Taxa 4x35, for not deleting 9M taxa trips containing personal data, was fined 1.2M DKK [Datatilsynet, 2019b].
- IDdesign A/S, for not deleting personal data regarding 385,000 customers, was fined 1.5M DKK [Datatilsynet, 2019a].
- Danske Bank, for not being able to document procedures for deleting personal data, and not being able to show that the personal data of millions of individuals was manually deleted, was fined 10M DKK [Datatilsynet, 2022a].

The objective of this project is to design and prototype a **Data Protection Compliance Tool (DPCT)** that supports companies in making their existing database systems GDPR compliant. In the context of this project, a database system is said to be GDPR compliant if it can be documented that the following apply to all personal data stored in the database system:

- **i** Personal data is associated with an individual.
- **ii** Personal data is only processed for legitimate and specific purposes.
- **iii** Personal data is only processed on a legal basis.
- **iv** Personal data is only stored for as long as it is necessary to fulfil a purpose, and it is deleted as soon as that is no longer the case.

These are the aspects of GDPR that this project focuses on, as **i** is needed for the data to be personal data, and **ii**, **iii**, and **iv** are needed in order for the company to lawfully process the personal data.

While creating a GDPR compliant database system from the ground up presents an interesting challenge, it will do little to help the companies that have existing database systems that need to be made GDPR compliant. This project therefore focuses on creating a tool for making existing database systems GDPR compliant.

1.2 Context

Existing studies analyse the effects of GDPR on the design and use of of database systems. These studies, along with GDPR itself, form the basis for the requirements presented in Section 1.3.

Shastri et al. [2019a] present a set of requirements for GDPR compliant database systems and proposes a benchmark, called GDPRBench, that evaluates the performance of a given database system that fulfills these requirements. The requirements are based on legal cases arising from GDPR as well as an analysis of GDPR. The study adopts a strict interpretation of GDPR to define their requirements, such that they propose the worst-case scenario for a compliant database system.

According to Shastri et al. [2019a], GDPR introduces several behavioural characteristics for personal data, which they represent as metadata. These are used as a basis for the requirements for DPCT as they define the properties for how to treat personal data in accordance with GDPR.

Shastri et al. [2019b] analyse GDPR and determine how it conflicts with modern database system design. Based on this analysis, seven "sins" of processing personal data are presented, each of which is a common practice that is in violation of GDPR.

Two of these sins are relevant to the aspects of GDPR considered in this project. The first is storing personal data forever. The second is reusing personal data indiscriminately, which happens when personal data is perceived and used as a resource that can be used without restriction to help in accomplishing the goals of the company. The remaining sins fall outside the scope of DPCT as described in Section 1.1.

Agarwal et al. [2022] present a tool called GDPRizer. The purpose of this tool is to generate queries that extract or delete personal data associated with an individual from a legacy database. This requires the ability to identify all personal data in a database related to a particular individual, which may not be possible, as legacy databases may have schemas that lack the necessary information.

The study shows the difficulty of automatically associating personal data with the owning individual in legacy databases, and it highlights the necessity for user input to handle all possible relations, which DPCT also utilises.

1.3 Requirements

GDPR [European Commission, 2016] presents requirements for lawfully processing personal data, and Shastri et al. [2019a], Shastri et al. [2019b], and Kraska et al. [2019] each study how these requirements affect personal data stored in relational databases. The requirements for DPCT are extracted from these four sources. Table 1.1 presents the requirements for DPCT and how they relate to the GDPR articles, and Table 1.2 presents how the requirements are related to the three other sources, Kraska et al. [2019]; Shastri et al. [2019a,b].

Next, Section 1.4 presents an example of a database system that has the necessary data for DPCT to make it GDPR compliant and fulfill these requirements.

#	**Name**	**Description**	**GDPR Article**
1	Purposeful processing	Personal data is only processed for legitimate and specific purposes.	Article 5(1)(b)
2	Legal processing	Personal data is only processed on a legal basis.	Article 6(1)
3	Associated individual	Personal data is associated with an individual.	Article 15
4	Storage limitation	Personal data is only stored for as long as it is necessary to fulfil a purpose, and it is deleted as soon as that is no longer the case.	Article 5(1)(e)
5	Audit trail	Operations affecting personal data are logged such that regulators can inspect compliance.	Article 5(2)

Table 1.1: Requirements for a system that facilitates compliance with GDPR. The referenced articles are from GDPR [European Commission, 2016]

#	Kraska et al. [2019]	Shastri et al. [2019a]	Shastri et al. [2019b]
1	Their **Queries** requirement	Their **Purpose** requirement	**3.2 Reusing Data Indiscriminately**
2	Their **Controlled storage access** requirement		**3.2 Reusing Data Indiscriminately**
3	**3.2 Existing Schema Within a Single System**, **3.3 Across Systems in the Enterprise**	Their **Associated person** requirement	
4		Their **Time to live** and **Timely Deletion** requirements	**3.1 Storing Data Forever**
5	**5 The Audit Process**	Their **Audit trail** requirement	

Table 1.2: Sources of the requirements from Table 1.1 beyond GDPR. The absence of a value indicates that the requirement is not related to the given source.

1.4 Running Example

This section presents a case of a fictional company with an existing database system that the company wants to make GDPR compliant. This case is used as a running example throughout the report.

The company is a web shop based in Denmark, which stores account and purchase information about their customers. As the web shop is based in Denmark, it must comply with Danish law. To limit the scope of this example, only Bogføringsloven [Retsinformation, 2006], in addition to GDPR, are considered.

Bogføringsloven regulates the data that must be stored to document transactions between individuals and companies. The paragraph considered in this example is §12, which states that information relevant for bookkeeping should be kept for five fiscal years after a transaction.

The web shop's database has three tables storing information relevant to this example, namely the `users`, `orders`, and `newsletter` tables. The schemas of these tables are shown in Figure 1.1, and excerpts of the data they contain are shown in Table 1.3, Table 1.4, and Table 1.5.

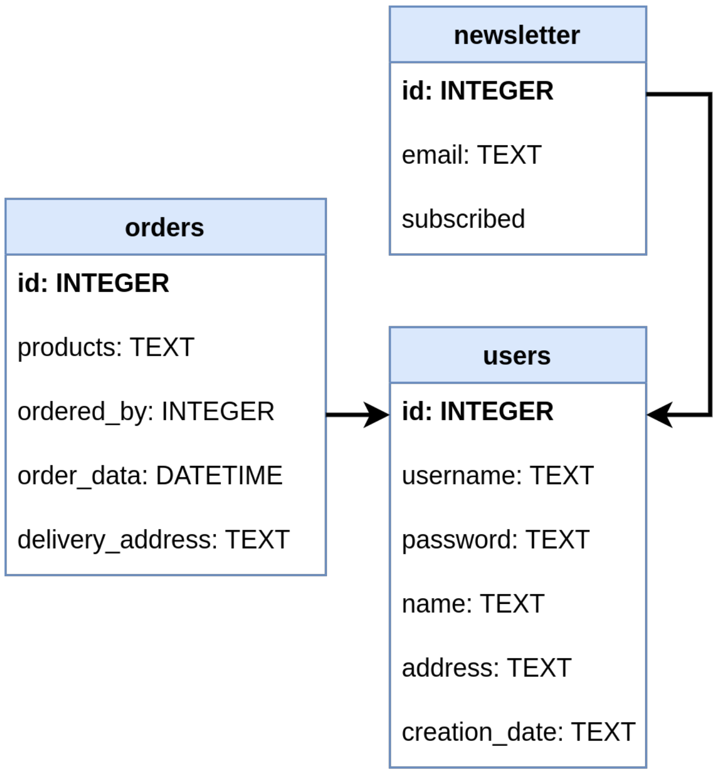

Figure 1.1: An overview of the schema of the database used throughout the report.

An excerpt of the data stored in the database system can be seen in the following tables:

id	username	password	name	address	creation_date
1	yellowgorilla847	31a4d45[..]	Emil Olsen	6325 Højagervej	2018-01-20 00:16:00
2	biggoose777	f058301[..]	Signe Pedersen	5593 Bøgebakken	2022-10-06 15:43:00
3	goldenbird592	8c2a9f1[..]	Emily Sørensen	4479 Neptunvej	2020-04-24 20:54:00

Table 1.3: An excerpt of the `users` table, where the passwords have been truncated such that all fields can be seen.

id	email	subscribed
1	emil.olsen@example.com	0
2	signe.pedersen@example.com	0
3	emily.sorensen@example.com	1

Table 1.4: An excerpt of the `newsletter` table. Attribute `id` is a foreign key referencing `users.id`.

id	products	order_date	delivery_address	ordered_by
1	Leverpostej	2017-08-26 10:34:00	2761 Hestehavevej	825
2	Rødgrød med fløde	2018-05-06 21:13:00	85 Ellevej	951
3	Rødgrød med fløde	2016-09-10 12:54:00	1858 Gammel Køge Landevej	248

Table 1.5: An excerpt of the `orders` table. Attribute `ordered_by` is a foreign key referencing `users.id`.

The following columns contain personal data: `username`, `name`, and `address` in the `users` table; `email` in the `newsletter` table; and `delivery_address` in the `orders` table.

The personal data is associated with individuals identified internally in the database system by the surrogate key `id` in the `users` table.

The web shop processes the personal data for two purposes: `username`, `name`, `address`, and `delivery_address` for the purpose of `Bookkeeping`, as well as `name` and `email` for the purpose of `Marketing`.

Processing personal data for the purpose of `Bookkeeping` is valid for five years after the last purchase by the customer with whom the personal data is associated, in accordance with Bogføringsloven [Retsinformation, 2006]. After the five years have passed, the personal data must be deleted at the end of the fiscal year, unless it is being processed for another purpose.

Processing personal data for the purpose of `Marketing` is only valid so long as the individual, with which the personal data is associated, is subscribed to the newsletter, i.e., as long as the value of the `subscribed` column in the `newsletter` table is true for that individual. Once the individual unsubscribes, the personal data must be deleted as soon as possible, unless it is being processed for another purpose.

The web shop processes the personal data for the purpose of `Marketing` on the legal basis of consent, in accordance with point a of GDPR Article 6(1), and processes the personal data for the purpose of `Bookkeeping` on the legal basis of legal obligation, in accordance with GDPR Article 6(1)(c).

1.5 Report Outline

Chapter 2 presents existing work related to GDPR compliance in existing database systems. The design of DPCT is covered in Chapter 3, and a prototype implementation of this design is described in Chapter 4. How well the prototype fulfills the requirements defined in Section 1.3 is evaluated in Chapter 5. The decisions made in the design and implementation of DPCT, as well as the results of the evaluation of DPCT, are discussed in Chapter 6. Extensions of DPCT to provide additional support for GDPR and other future work is the subject of Chapter 8. The conclusion of the project is presented in Chapter 7.

Chapter 2

Related Work

There is existing work that presents solutions or support functionality related to the requirements of DPCT. This chapter presents three areas that provide useful tools or functionality for GDPR compliance in existing database systems.

2.1 Extending Databases With Compliance

Kraska et al. [2019] propose a data management system called SchengenDB that is meant to help companies comply with GDPR. The purpose of the system is to protect personal data, by only allowing the data to be processed for specific purposes with the owning individual's permission, and to support individuals' right to be forgotten by being able to delete the personal data stored about an individual. SchengenDB also supports auditing to document compliance with GDPR by extending the existing Database Management System (DBMS) log.

SchengenDB proposes functionality similar to what is needed to fulfill requirements 1, 2, 3, and 5 from Section 1.3.

Each attribute containing personal data is associated with one or more purposes for processing the personal data. Each purpose has a legal basis that allows the processing of the personal data for the associated purposes. These purposes are used to restrict how and why personal data can be accessed and processed, by requiring all access to be associated with specific purposes. Only personal data associated with a matching purpose can be accessed. Individuals are able to opt-out and opt-in of each purpose. This functionality ensures that personal data is processed for specific purposes, and that the processing has a legal basis. This is the functionality needed to fulfill requirements 1 and 2 from Section 1.3.

Supporting individuals' right to be forgotten requires associating personal data with the owning individual, and being able to delete it. This is similar to requirement 3. In order for SchengenDB to be able to support this, it requires that individuals are unique within the database systems of the company. A similar precondition is necessary for DPCT.

SchengenDB extends the DBMS log to include read operations as well as the queries that were invoked and the associated purposes. This information is also added to a log entry created to record update operations. This additional information allows the DBMS log to be used to audit compliance with GDPR by inspecting the log and ensuring that the purposes are enforced. This extension of the DBMS log is similar to the audit trail specified in requirement 5.

SchengenDB presents functionality relevant to the fulfillment of the requirements. DPCT builds on this work by providing an implementation of some of this functionality. DPCT associates each element of personal data with a purpose, a legal basis for processing, and the owning individual. DPCT also logs updates to metadata associated with the personal data, which can be used to verify compliance.

SchengenDB and DPCT differ on their focus in regards to deleting personal data. SchengenDB focuses on supporting the individuals' right to be forgotten (GDPR Article 17), while DPCT focuses on deleting data when it is no longer necessary for fulfilling any purpose (GDPR Article 5(1)(e)). Both approaches can be extended to support the other.

2.2 Schema Evolution

This section presents existing work on schema evolution, which can be used to support GDPR compliance of existing databases. Compliance with GDPR in an RDBMS requires a single notion of individuals [Kraska et al., 2019], which in this report is interpreted as each individual having a unique identifier. Agarwal et al. [2022] show that this is not always the case in legacy database systems, and that it can be difficult to identify all related information. Schema evolution can be used to create such unique identifiers in existing databases where they do not already exist. Modifying the schema such that individuals can be represented with a unique identifier and migrating the database supports associating personal data with individuals according to requirement 3.

Curino et al. [2008] present a tool for schema evolution in relational databases called PRISM. The tool provides a language for schema modifications and means of evaluating the effect of changes to the schema, translation of queries to the transformed schema, automatic migration of data to the transformed schema, and documentation of changes made to the schema.

Curino et al. [2010] extend this functionality, allowing for the evolution of integrity constraints and supporting updates performed on previous versions of a schema. The evolution of integrity constraints adds further support for creating unique identifiers for individuals, by allowing for the removal and addition of foreign key constraints. This enables the removal of the unconventional representations of relationships mentioned by Agarwal et al. [2022].

Nykjær et al. [2023] propose a system using schema evolution to improve the GDPR compliance of an existing database system. The proposed system allows owners to change the schema of their relational databases to better support associating personal data with individuals and reducing duplication of personal data. Query rewriting is used to allow applications using the database to continue to function while the schema changes.

2.3 Data Vacuuming

Palmer and Srikandarajah [2022] propose an approach for rule-based data retention compliance by defining an approach in which a user can specify a set of keep and removal rules, which are associated with specific attributes in a database. A removal rule defines a Boolean condition according to which the associated attribute values should be deleted. If multiple removal rules exist, they are executed one by one. A keep rule defines a Boolean condition for associated attributes according to which they should be kept, which ensures that the data is not deleted by any of the removal rules. If multiple keep rules exist for some attribute, it is kept if any of the rules evaluate to true. The purpose of this approach is to be able to define arbitrary removal and keep rules for data in a database, and ensuring that all data, that should be kept, is kept.

An adapted version of this functionality is incorporated into DPCT, as requirement 4 implies that it should be possible to remove personal data when certain conditions are met.

2.4 Summary

Existing studies propose much of the functionality needed to fulfill the requirements defined in Section 1.3. Kraska et al. [2019] propose a system that fulfills many of the requirements, but provide only an abstract design and no implementation. The contribution of DPCT is to implement some of this functionality and to combine it with the data vacuuming, presented by Palmer and Srikandarajah [2022], in order to provide companies with a tool that helps them ensure that all personal data in their existing database systems is compliant with GDPR, and is removed as soon as that is no longer the case.

Although there are preconditions for DPCT to be applicable, existing work on schema evolution by Curino et al. [2008, 2010] and Nykjær et al. [2023] can be used to fulfill these preconditions.

Chapter 3

Design

3.1 Overview

The core of DPCT is three components, marked in blue in Figure 3.1, each of which has a distinct set of responsibilities.

The **Metadata** component is responsible for keeping track of personal data, including where it is stored, and what metadata is defined for each piece of personal data. This component is described in further detail in Section 3.2.

The **Vacuuming** component is responsible for deleting personal data that no longer has any valid purpose for processing. To do this, the component facilitates the specification of vacuuming policies for when data should be deleted. Data is then deleted periodically according to these policies. This component is described in further detail in Section 3.3.

The **Logging** component is responsible for documenting compliance. This is done by logging the operations performed by DPCT. Logging when data is registered as personal data, changes to metadata, and the creation and execution of vacuuming policies for deleting personal data allows for documenting that the database system is compliant with GDPR. This component is described in further detail in Section 3.4.

To facilitate interaction with users and databases, there are two layers of functionality, marked in green in Figure 3.1, that serve to separate these interactions from the core functionality of DPCT.

The **Command** layer provides a set of commands for interacting with the core. Users can access these commands through a **User Interface**, which in the case of the prototype implementation of this design presented in Section 4.4 is a command line interface. This layer is described in more detail in Section 3.5.

The **Data Access** layer provides an abstraction over database access that can be changed depending on the database management system in use. All interactions with the Data Access layer are routed through the Logging component. This ensures that every interaction with the data is logged. This layer is described in more detail in Section 4.1.

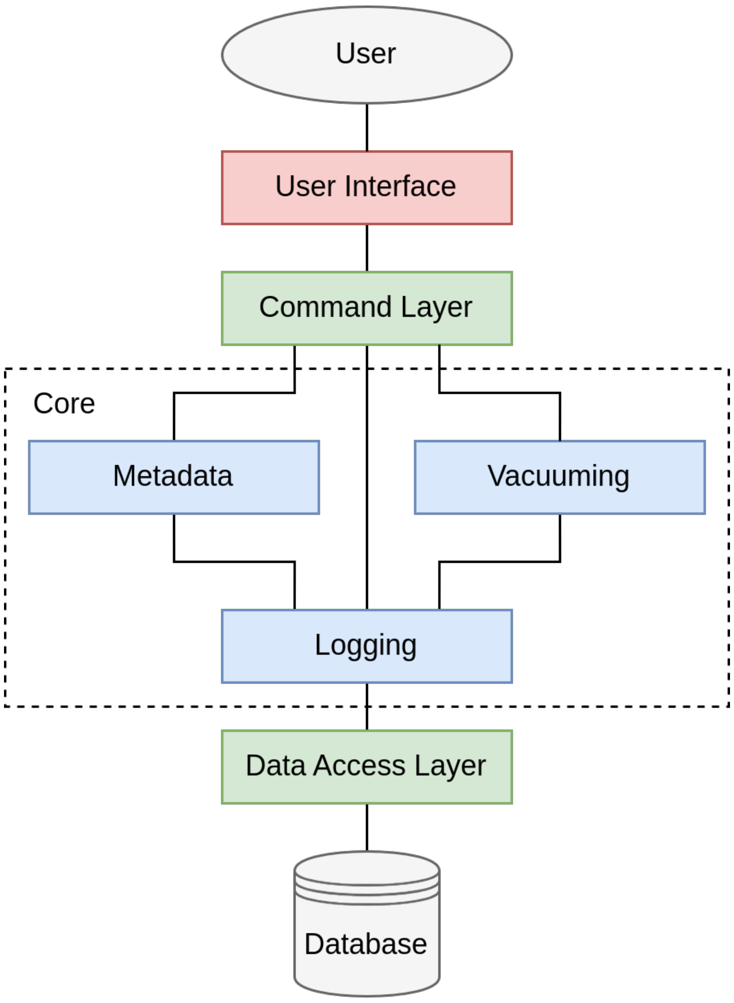

Figure 3.1: Architecture diagram of DPCT.

3.2 Metadata

The requirements in Section 1.3 necessitate registering and maintaining additional metadata about personal data stored in the database. The metadata is needed to determine whether the personal data can be stored in compliance with GDPR. GDPR compliance requires that (**a**) personal data is used for one or more explicit and lawful purposes, as stated by requirements 1 and 2, that (**b**) the associated individual of the personal data is known, as stated by requirement 3, and that (**c**) the personal data is stored only as long as it is needed to fulfill at least one of the associated purposes, as stated by requirement 4.

Determining whether a piece of personal data can continue to be stored requires the following information:

- A purpose for processing the personal data.
- The personal data's time to live.
- Whether the personal data is required to fulfill a legal obligation.
- The individual the personal data belongs to.

When personal data is no longer needed, GDPR Article 6(1)(e) requires that it is no longer possible to associate it with an individual. The process of removing the personal data that is no longer needed is presented in Section 3.3.

The remainder of this section explains the nature of the metadata and how it is used to ensure that the storage of personal data in the database is compliant with GDPR.

3.2.1 Registering Personal Data

When DPCT is first deployed on an existing database system, none of the data is assumed to be personal data. DPCT allows users to register data stored in the database as personal data. As an example, a user could register the data stored in the `name` column of the `users` table from Section 1.4 as personal data. DPCT then stores a reference to this table and column, such that it is possible to associate it with the required metadata.

It is worth emphasising that personal data is managed at the column level. This is based on the assumptions that a single column storing personal data in a relational database contains data of the same nature, and that the data stored in that column is being processed for the same purposes. The nature of personal data stored in the column may be apparent from the name. An example of this is the `address` column in the `users` table shown in Table 1.3 of the example. However, there is no guarantee that the nature of the personal data will be clear to a user based on the names of the table and column. Therefore, DPCT allows the user to also provide a description of the data.

Section 6.1 presents further discussion of the granularity of personal data.

3.2.2 Creating Purposes

GDPR Article 5(1)(b) requires a legitimate purpose for storing and processing personal data. As such, DPCT requires that one or more purposes is defined for each column storing personal data.

A user can create a purpose, which requires a name and description of the purpose, and specifying whether or not processing data for this purpose is based on a legal obligation. The name and description serves to inform users about the nature of the purpose. The specification of a legal obligation is a simplification of the legal basis required by GDPR Article 6(1), and is used to determine whether personal data should be deleted as the result of the owning individual objecting to the processing. If personal data has a purpose indicating that it is being processed based on a legal obligation, it must not be deleted based on objections by the individual. As an example, personal data stored with the purpose of `Bookkeeping` from Section 1.4 is being processed based on a legal obligation and must not be deleted even if an individual objects, while personal data stored for the purpose of `Marketing` is not being processed based on a legal obligation and must be deleted if an individual requests it. The consequences of this simplified approach are discussed in Section 6.1 and a more elaborate approach to representing the legal basis of a purpose is discussed in Section 8.1.

Purposes are also used to determine how long personal data can be stored. The conditions for which data can be stored is referred to as the Time to Live (TTL), and when it expires the purpose is no longer valid for a given piece of personal data. An example of the TTL for a purpose is that Bogføringsloven [Retsinformation, 2006] mentioned in Section 1.4 requires that relevant personal data is kept for five fiscal years since the last transaction, and as such must only be deleted once those five years have elapsed. Another example is that personal data may need to be deleted when an individual withdraws consent. These TTLs and their use in deleting data are explained in greater detail in Section 3.3.

3.2.3 Associating Personal Data with Individuals

DPCT assumes that:

- **I** Individuals are represented in the database system by a unique identifier.
- **II** The identifiers representing individuals are single-column, i.e, not multi-column.
- **III** The identifiers are stored in a single table known by DPCT.
- **IV** A one-to-many relationship exists between an identifier and the personal data of the individual that identifier represents.

These assumptions result in a simplified approach, from both the user and DPCT's point of view. Methods for satisfying requirement **III** are presented in Chapter 2. How to relax these assumptions and the effects on DPCT are discussed in Section 6.1.

Constructing the one-to-many relationship in requirement **IV** is achieved by defining an association expression for the columns that are registered as storing personal data. This association expression must describe how data stored in the column relates to the identifiers of the individuals and must result in a one-to-many relationship. Under the current assumptions, the association expression is the column in the table storing the personal data that holds the foreign keys to the table storing the individuals' identifiers.

Using the example in Section 1.4, when registering the `email` column in the `newsletter` table as storing personal data, and assuming that the individuals' identifiers are stored in the `id` column of the `users` table, the user must also provide the association expression "`newsletter.id`". Using this information, it is possible to construct the one-to-many relationship between the `id` column of the `users` table and the `email` column of the `newsletter` table using the following query:

```sql
SELECT individuals.id, newsletter.email
FROM users AS individuals
JOIN newsletter ON individuals.id = newsletter.id
```

The column storing the individuals' identifiers is always aliased to avoid name clashes in cases where the personal data and the identifiers are stored in the same table.

3.3 Vacuuming

As mentioned in Section 3.2, personal data is associated with one or more purposes. Requirement 4 in Section 1.3 states that personal data should not be stored if it is no longer necessary for fulfilling an associated purpose. To ensure this, DPCT deletes personal data that has no valid purpose. This process is referred to as vacuuming, and the general principles of the vacuuming process are based on Palmer and Srikandarajah [2022].

Deletion of personal data can be achieved in different ways, either by removing the data from the database, by anonymising the data, or by replacing it with a value defined by the user of the system (GDPR Article 5(1)(e)). DPCT uses a value that can be defined by the user.

As personal data can be used for multiple purposes, e.g., emails being used for both bookkeeping and marketing, it is necessary to determine whether all purposes have elapsed for a given value before deleting it. This is achieved by combining the set of conditions defined by the assigned purposes into a statement that updates the values to the value defined by the user, if all conditions are met. The structure of such a combined statement can be seen in Listing 3.1. If more conditions are present, these conditions will be appended to the end, using an `AND` operator for each condition.

```sql
UPDATE Table SET Column = 'Value' WHERE (Condition1) AND (Condition2);
```

Listing 3.1: The structure of a statement that selects the data to be deleted and replaces it with a user defined value. `Condition1` and `Condition2` are conditions that evaluate to true if the value of the column in a given row should be deleted. If more conditions are defined from different purposes, each condition is appended with another `AND` operator.

The structure defined in Listing 3.1 imposes limitations on the queries used in the conditions, namely that the queries should result in Boolean values and that the conditions should reference the outer query.

An example of such a reference can be seen in Listing 3.2. The condition contains `users.id = u.id`, where `users.id` refers to the `users` table in the outer query, while `u.id` refers to the `users` table in the inner query.

If the query in the condition does not reference the outer query with a constraint similar to the example described above, the condition would evaluate to true for all rows if there exists one row fulfilling that condition. The update statement described in Listing 3.1 would then be equivalent to `UPDATE Table SET Column = 'Value' WHERE (true);`, which would then update all values in the column. However, if the condition references the outer query in a constraint, it would validate each row to ensure that the id matches the current row, and therefore only update the values that no longer have a valid purpose.

```
SELECT id FROM users WHERE
EXISTS(
    SELECT * FROM users as u
    WHERE u.creation_date > datetime('now', '-2 year')
    AND users.id = u.id
);
```

Listing 3.2: A condition containing a selection query. This query selects all rows from `users` that are more than two years old.

DPCT places two restrictions on the content of the conditions, the first being that it should reference the outer query, and the second being that it should be a Boolean condition. Aside from these, the user can create arbitrary conditions, allowing them to create conditions which fit their needs. As a result, it is possible to define contradicting conditions, such that the combined condition will never evaluate to true. As an example, if one condition specifies that users created more than two years ago should be deleted, and another condition specifies that users created less than two years ago should be deleted, no values will ever be deleted, as all conditions must be true for a value to be deleted. However, preventing this from occurring is beyond the scope of this project.

As a single purpose can be used for personal data stored in different tables, each purpose has a condition for each of the associated tables. Each condition specifies when the purpose is no longer valid for the values in the particular table, such as the condition in Listing 3.2 which is specific to the `users` table.

For example, the `Marketing` purpose may be used in different tables, where the name of the individual is stored in one table while the email of the individual is stored in another table. Each of these tables would then have a specific condition to validate the data. This relation can be seen in Figure 3.2.

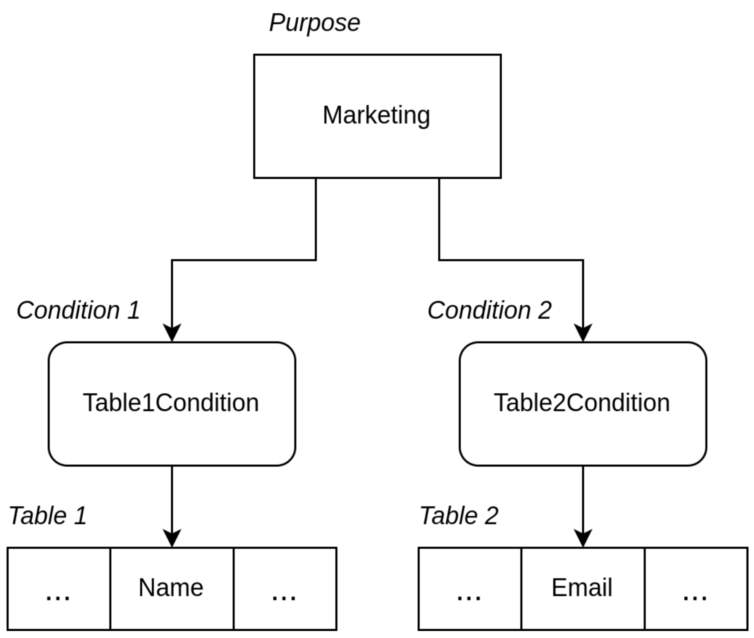

Figure 3.2: Figure showing the relation between a purpose and a set of conditions on different tables.

3.3.1 Deletion of Personal Data Related to a Purpose

As there is a many-to-many relation between columns and purposes, the deletion of personal data stored with a given purpose requires that all columns relating to a purpose are validated, which in turn requires evaluating all conditions for each column. As a result, the deletion of personal data related to a particular purpose will evaluate conditions from other purposes as well. In Algorithm 1, the pseudocode describing the vacuuming process can be seen.

Algorithm 1 Pseudocode for Vacuuming

```
procedure CreateUpdateStatement($Purpose$)
    statement = ""
    for column in ColumnsWithPurpose($Purpose$) do
        for condition in ConditionsForColumn(Column) do
            statement.Append(condition)
        end for
    end for
end procedure
procedure ColumnsWithPurpose($Purpose$)
    Returns a list of columns with the given $Purpose$
end procedure
procedure ConditionsForColumn($Column$)
    Returns a list of Conditions associated with the given $Column$.
end procedure
Execute(CreateUpdateStatement(Purpose))
```

By iterating through all purposes related to a specific column, and combining the related conditions as described in Listing 3.1, DPCT ensures that all personal data with no valid purpose is deleted, and that only that data is deleted.

3.3.2 Periodic Deletion

It should be possible to define a periodic execution that automatically verifies specific purposes, based on a user-defined duration, such that personal data is deleted as soon as it becomes unnecessary to store it. As the conditions of some purposes may take a significant amount of time to evaluate, or are only required to be verified at specific dates, it should be possible to define different durations for different purposes. One such example is personal data stored with the purpose of `Bookkeeping`, which only needs to be deleted at the end of the fiscal year. This is discussed in more detail in Section 8.2.2.

3.4 Logging

As stated in requirement 5 in Section 1.3, DPCT must log all operations on personal data in order to create an audit trail. An audit trail must contain the information necessary for regulators to inspect the compliance of the database system. In this project, this is interpreted to mean that the audit trail should be able to show that the database system complies with requirements 1 to 4 from Section 1.3.

DPCT logs all changes made to metadata and the effects of all vacuuming executions, i.e., all updates carried out by DPCT are logged. As an example, a new purpose being created in DPCT is logged, and the vacuuming of personal data is also logged. Queries made by DPCT are not logged, as DPCT only has read access to metadata and not to personal data.

Queries and updates of personal data performed outside of DPCT are currently not logged, which is discussed in Section 6.3. Ideally, external queries and updates would also be available for auditing.

A log entry has the following content:

- The log id. A number that is incremented each time a new log entry is created.
- The time and date of the log entry.
- The type of event the log entry is recording. In DPCT this is limited to metadata changes, called `Metadata`, and vacuuming executions, called `Vacuuming`.
- The subject of the log entry, i.e. the entity that is affected. As an example, if a new purpose is created, the subject of the log entry is the name of the new purpose, or if personal data is being vacuumed, the subject is the name of the column that is affected.
- The log message. The content of this field depends on what is being logged, but at minimum contains information about the operation that has been performed, which in DPCT is limited to `create`, `update`, `delete`, and, in the case of vacuuming policies, `execute`. In a log entry regarding updating metadata, the new value of the metadata is also recorded in the message. In a log entry regarding vacuuming policy executions, the message contains the statement that was sent to the database, which column was affected, and the reason for that column being affected. The reason is always that the personal data in the column is stored under a purpose.
- The format of the log message. Two different formats can be used, these being `Plaintext` and `JSON`.

The syntax of a log entry is the values of the files separated by a delimiter. The delimiter can be configured. Using an example delimiter of '|', the syntax is:

```
id | timestamp | logType | subject | messageFormat | message
```

Examples of what an actual log entry looks like in the implementation can be seen in Section 4.7.

Using these log entries, it is possible to follow the history of a piece of metadata and the effects of vacuuming executions, which is covered in more detail in Section 5.4.

Being able to follow the history of metadata and effects of vacuuming over time is necessary for showing compliance. As an example, showing that the purpose for processing a piece of personal data was specified at a certain point in time, and has never changed since then, would show that a company is in compliance with requirement 1. Another example is showing that vacuuming executions are run periodically, and showing the effects of these executions, would document that the company is in compliance with requirement 4.

In order to make it easy for regulators to inspect the log, i.e., the audit trail, basic search functionality is also available. The user can limit the logs to a time frame, a set of log types, or a set of subjects. As an example, it is possible for the user to specify that they only want to see log entries from the month of June 2022, of the type `Vacuuming`, and with the subject "(users, address)" i.e., the `address` column in the `users` table. This would result in the log entries for vacuuming executions that have affected the `address` column in the month of June 2022.

3.5 Command Layer

The Command layer presents the functionality provided by the core components, Metadata, Vacuuming, and Logging, in an API that allows a user to interact with DPCT in the form of a set of commands. These commands operate on metadata entities in DPCT, such as purposes and columns storing personal data, and are based on Create, Read, Update, and Delete operations, with some additions.

The **Create** command creates a new metadata entity in DPCT using the user provided information, and is of the form:

```
Create(entity-type, entity-key)
```

where `entity-type` is the type of entity that should be created, e.g., purpose or a column storing personal data, and `entity-key` is how this entity can be referenced, e.g., in the case of purposes, it is a name, and in the case of a column storing personal data, it is a pair of the table name and the column name. The definitions of these two parameters are the same for the other commands.

The **Update** command updates a metadata entity with the information provided by the user, and is of the form:

```
Update(entity-type, entity-key, entity-value)
```

where `entity-value` is the new value that the entity referenced by the given `entity-key` should be updated to. This `entity-value` can contain a new key for the referenced entity, and after the command has been run, the old key should no longer reference an entity.

The **Delete** command deletes a metadata entity in DPCT, and is of the form:

```
Delete(entity-type, entity-key)
```

The **List** command shows all metadata entities of a user provided type, and is of the form:

```
List(entity-type)
```

The **Show** command shows details about a specific metadata entity, and is of the form:

```
Show(entity-type, entity-key)
```

The **Status** command shows the current GDPR compliance status of either all entities of a user provided type or all entities in DPCT, and is of the form:

`Status(entity-type)` **or** `Status()`

These commands are available to use for all types of metadata entities in DPCT. The user can access the commands through the user interface, which is described in more detail in Section 4.4.

Chapter 4

Implementation

4.1 Technical Details

Data Protection Compliance Tool (DPCT) is implemented in the C# language using the .NET framework and the implementation is publicly available at `https://github.com/P9-P10/DataProtectionComplianceTool`. The Command Line Interface is implemented using the System.CommandLine package [Microsoft, 2022].

DPCT interacts with relational databases using the SQLite, as it is small, fast, and self-contained [SQLite, 2023a]. The implementation of DPCT described in this report contains 9754 lines of code. It has been tested through 176 tests.

In addition to this, DPCT makes use of the object-relational mapper Entity Framework Core.

4.2 Domain Model Terminology

This section introduces the domain model representation of the concepts introduced in Chapter 3. This terminology is used when describing and discussing the implementation of the design.

Columns storing personal data are referred to as `personal data columns`, and they are identified by a pair of the form `(tableName, columnName)`.

Purposes for processing personal data are referred to as `purposes`, and they are identified by a name.

Individuals whose personal data is being processed are referred to as `individuals`, and they are identified by an integer id.

Conditions for when personal data stored with a specific purpose should be deleted are referred to as `storage policiess`, and they are identified by a name.

Specifications of the duration between the vacuuming of personal data stored with a specific purpose are referred to as `vacuuming policies` and they are identified by a name.

The attribute on a purpose describing whether or not the data is being processed is based on a legal obligation is represented by the Boolean `legally required`.

The user-defined value that is used when vacuuming personal data is referred to as the `default value`, and one is defined for each personal data column.

4.3 Setup and Configuration

DPCT is meant to be used on an existing database system, and the user must therefore provide a connection string to a database at startup. When DPCT is first started it creates additional tables in the database. These tables are used to store details about the domain entities listed in Section 4.2. The schema for these tables is shown in Figure 4.1.

The user must also provide a path to the file they want the log to be stored.

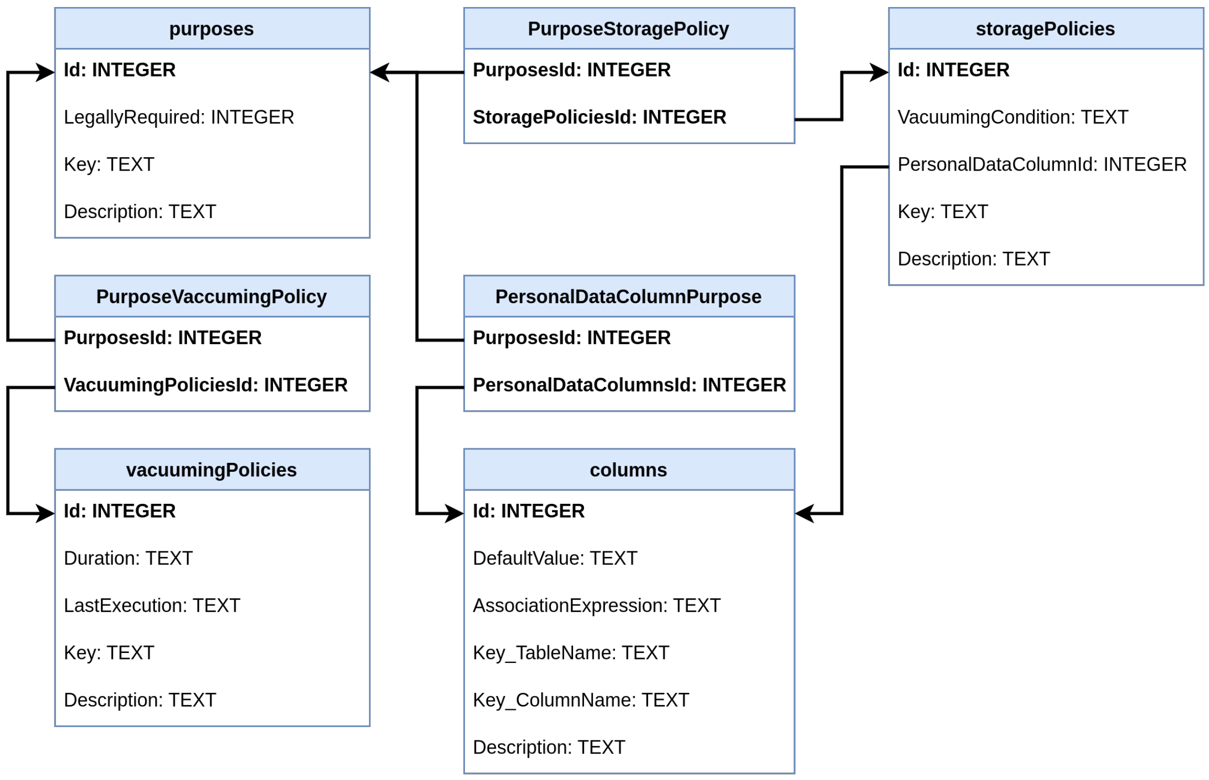

Figure 4.1: A figure showing the tables used by DPCT to store information on the domain entities.

4.4 Command Line Interface

The commands described in Section 3.5 have been implemented as a Command Line Interface (CLI) in the prototype. All commands available to the user are of the form:

```
[metadata-entity-type] [command] [options],
```

where `[metadata-entity-type]` can be any of the domain entities described in Section 4.2, and `[command]` can be any of a set of available commands based on the chosen metadata-entity-type. Options are of the form:

```
--[option-name] [value]
```

As an example, if a new purpose called `Marketing` should be created in DPCT, the following command could be used:

```
purpose create --name Marketing
```

The same command could also be used to provide more information in the initial creation using more options:

```
purpose create --name Marketing --description "This data is used for mar-
keting" --legally-required false --storage-policies MarketingUsers
```

This creates a `Marketing` purpose with a description, and the information that personal data stored for this purpose is not legally required, and that the vacuuming of data under this purpose should follow the storage policy `MarketingUsers`.

All commands and options also have aliases, so the previous commands could also be:

```
p c -n Marketing -d "This data is used for marketing" -lr false -sps Mar-
ketingUsers
```

This is useful for quickly using the CLI once the user is familiar with the available commands. However, for the purposes of readability, only the proper names will be used in examples in this report.

More examples of commands will be given throughout the report. However, Appendix A presents a complete list of the available commands.

The following sections contain examples of commands and their output. Lines starting with the symbol $ indicate that the line is entered by the user. The lines following the user input are the output produced by DPCT. The following interaction is an example of an interaction with the prototype using the CLI, which demonstrates this notation.

```
$ purpose create --name Marketing

Purpose 'Marketing' successfully created
Purpose 'Marketing' is missing a legally required value
Purpose 'Marketing' is missing a vacuuming policy
Purpose 'Marketing' is missing a storage policy
```

4.5 Metadata

This section demonstrates how to use DPCT to manage the personal data in the example database described in Section 1.4, following the process described in Section 3.2.

4.5.1 Registering Personal Data

The first step is to register a column in the database as containing personal data. The command and resulting output can be seen in the following interaction:

```
$ personal-data-column create --table-column users address

Personal data column '(users, address)' successfully created
Personal data column '(users, address)' is missing a purpose
Personal data column '(users, address)' is missing a default value
Personal data column '(users, address)' is missing an association expression
```

This interaction results in the creation of a row in the `columns` table as shown in Table 4.1.

id	DefaultValue	AssociationExpression	TableName	ColumnName	Description
1	NULL	NULL	users	Address	NULL

Table 4.1: The contents of the `column` table after `users.address` has been registered as containing personal data.

The output of the command informs the user that information is missing about the purpose for storing the address, about the default value to use when deleting the information, and about the association expression used to associate individuals with their personal data. A description of the column can also be added to describe the nature of the data stored in the column, but it is not required.

4.5.2 Updating Personal Data

The following interaction shows a command that updates the entity describing the `address` column by adding a default value, a description, and an association expression.

```
$ personal-data-column update --table-column users address --default-value
 ↪  "removed" --description "The customer's address" --association-expression
 ↪  "users.id"

Personal data column '(users, address)' successfully updated to '(users,
 ↪  address), The customer's address, removed, users.id, Empty'
Personal data column '(users, address)' is missing a purpose
```

The changes to the `columns` tables as a result of this interaction can be seen in Table 4.2.

id	DefaultValue	Association-Expression	TableName	ColumnName	Description
1	removed	users.id	users	Address	The customer's address

Table 4.2: The contents of the `columns` table after `users.address` has been updated with appropriate values for default value, description, and association expression.

4.5.3 Creating a Purpose

The output of the update command once again reminds the user that no purpose has been defined for the column. To specify a purpose for the column, it is first necessary to create a purpose in DPCT. The following interaction shows a command that creates a new purpose that can be used to indicate that personal data is used for marketing. The output of the command informs the user about additional information needed about the purpose.

```
$ purpose create --name Marketing

Purpose 'Marketing' successfully created
Purpose 'Marketing' is missing a legally required value
Purpose 'Marketing' is missing a vacuuming policy
Purpose 'Marketing' is missing a storage policy
```

4.5.4 Defining the Fields of a Purpose Immediately

To remove the need for the user to go through several create commands for the other entities required to define a entity, it is possible to define all the necessary values when the entity is created.

The following interaction shows a command that creates a purpose used to indicate that personal data is used for bookkeeping. The command specifies a reference to a storage policy that does not exist, so the user is presented with the option of creating it.

```
$ purpose create --name Bookkeeping --legally-required true --storage-policies
 ↪  delete-after-five-years

delete-after-five-years storage policy does not exist. Would you like to create
 ↪  one? (y/n)
$ y

Storage policy 'delete-after-five-years' successfully created
Not reporting status when creating on demand
Purpose 'Bookkeeping' successfully created
Purpose 'Bookkeeping' successfully updated to 'Bookkeeping, None, True, [
 ↪  delete-after-five-years ], Empty, Empty'
Purpose 'Bookkeeping' is missing a vacuuming policy
```

The state of the purposes table as a result of this interaction is shown in Table 4.3, and the storage policy `delete-after-five-years` is added to the table `storagePolicies` as shown in Table 4.4. The relations between purposes and storage policies are stored in a join table shown in Table 4.5.

id	LegallyRequired	Key	Description
1	NULL	Marketing	NULL
2	true	Bookkeeping	NULL

Table 4.3: The contents of the `purposes` table after creating the `Marketing` and `Bookkeeping` purposes.

id	VacuumingCondition	PersonalDataColumnId	Key	Description
1	NULL	NULL	delete-after-five-years	NULL

Table 4.4: The contents of the `storagePolicies` table after creating the `delete-after-five-years` storage policy.

PurposesId	StoragePoliciesId
2	1

Table 4.5: The contents of the join table representing the many-to-many relation between purposes and storage policies. It contains a single entry relating purpose `Bookkeeping` to the storage policy `delete-after-five-years`.

4.5.5 Associating Personal Data With Purposes

When purposes have been created it is possible to associate them with personal data. The following interaction shows a command that associates the personal data stored in the column `users.address` with the purposes `Marketing` and `Bookkeeping`:

```
$ personal-data-column add-purpose --table-column users address --purposes
 ↪  Marketing Bookkeeping

Personal data column '(users, address)' successfully updated to '(users,
 ↪  address), The customer's address, removed, users.id, [ Marketing,
 ↪  Bookkeeping ]'
```

4.5.6 Seeing the System Status

An overview of all the necessary information that is still missing can be retrieved with the `status` command. The following interaction shows the result of this command given the state of DPCT produced by the preceding commands:

```
$ status

Purpose 'Marketing' is missing a legally required value
Purpose 'Marketing' is missing a vacuuming policy
Purpose 'Marketing' is missing a storage policy
Purpose 'Bookkeeping' is missing a vacuuming policy
Storage policy 'delete-after-five-years' is missing a vacuuming condition
Storage policy 'delete-after-five-years' is missing a personal data column
```

4.5.7 Listing Existing Entities

As the status shows, the `Marketing` purpose is still missing information on whether it is legally required and the policies for when associated personal data should be deleted. The user can create a new storage policy, but it is possible that an existing policy could be used. The following interaction shows how the `list` command can be used to see the existing entities of a given type:

```
$ storage-policy list

Key, Description, Vacuuming Condition, Personal Data Column, Purposes
delete-after-five-years, None, None, None, [ Bookkeeping ]
```

4.5.8 Creating a Storage Policy

The previous interaction shows that there is only the storage policy created earlier, which is not applicable to the `Marketing` purpose. Therefore, a new storage policy can be created in the same way as with purposes:

```
$ storage-policy create --name marketing-policy --description "data used for
 ↪  marketing should be deleted if the user is not subscribed"

Storage policy 'marketing-policy' successfully created
Storage policy 'marketing-policy' successfully updated to 'marketing-policy,
 ↪  data used for marketing should be deleted if the user is not subscribed,
 ↪  None, None, Empty'
Storage policy 'marketing-policy' is missing a vacuuming condition
Storage policy 'marketing-policy' is missing a personal data column
```

The newly added storage policy has a description. The reason for this is that the name should serve as a concise description that can be used to identify the entity. With `Bookkeeping` the description was sufficiently concise to act as the name. In the case of the storage policy for `Marketing` a longer description is required.

Now it is possible to update the `Marketing` purpose with the newly added storage policy. It is also possible to define that it is not legally required as the use of personal data for marketing is based on consent.

```
$ purpose add-storage-policy --name Marketing --storage-policies
 ↪  marketing-policy

Purpose 'Marketing' successfully updated to 'Marketing, None, None, [
 ↪  marketing-policy ], [ (users, address) ], Empty'
Purpose 'Marketing' is missing a legally required value
Purpose 'Marketing' is missing a vacuuming policy

$ purpose update --name Marketing --legally-required false

Purpose 'Marketing' successfully updated to 'Marketing, None, False, [
 ↪  marketing-policy ], [ (users, address) ], Empty'
Purpose 'Marketing' is missing a vacuuming policy
```

4.6 Vacuuming

After a set of purposes and personal data columns have been defined, the storage policies can be defined. In Section 4.5 a new storage policy named `marketing-policy` was created as a result of creating the `Marketing` purpose. For this storage policy to be fully applicable by the vacuumer, and for it to be usable for vacuuming, it has to be updated with the following command:

```
$ storage-policy update --name marketing-policy --vacuuming-condition
 ↪  "EXISTS(SELECT u.id FROM users AS u JOIN newsletter AS n ON u.id = n.id
 ↪  WHERE subscribed = 0 AND u.id = users.id)" --table-column users address

Storage policy 'marketing-policy' successfully updated to 'marketing-policy,
 ↪  None, EXISTS(SELECT u.id FROM users AS u JOIN newsletter AS n ON u.id =
 ↪  n.id WHERE subscribed = 0 AND u.id = users.id), (users, address), [
 ↪  Marketing ]'
```

This updates the value of the `VacuumingCondition` column for the marketing storage policy. The relevant row of the `StoragePolicies` tables can be seen in Table 4.6.

id	VacuumingCondition	PersonalDataColumnId	Key	Description
1	*MarketingCondition*	1	marketing-policy	NULL

Table 4.6: The contents of the `StoragePolicies` table after the storage policy `marketing-policy` has been updated. The query making up the `VacuumingCondition` for `marketing-policy` has been omitted. This condition can be seen in Listing 3.2.

4.6.1 Creating Vacuuming Policies

The purpose of the vacuuming policy is to define the duration between deleting personal data associated with specific purposes. This allows for the periodic deletion of personal data, as described in Section 3.3.2. Vacuuming policies also act as the entity in the command line interface used for deletion of expired data related to purposes, meaning that if the user wants to delete all expired data with the `Marketing` purpose, the vacuuming policies associated with `Marketing` should be executed.

After the storage policy has been updated, the vacuuming policy should then be defined, which is achieved by executing the following command:

```
$ vacuuming-policy create --name MarketingVacuuming --purposes Marketing
 ↪  --duration "1d"

Vacuuming policy 'MarketingVacuuming' successfully created
Vacuuming policy 'MarketingVacuuming' successfully updated to
 ↪  MarketingVacuuming, , 1d, None, [ Marketing ]'
```

This creates a vacuuming policy which is associated with `Marketing` and has a vacuuming duration of one day, meaning that the data stored with the `Marketing` purpose, that is no longer valid, is deleted with one day intervals. The above command results in a new entry in the `vacuumingPolicies` table, which can be seen in Table 4.7.

id	Duration	LastExecution	Key	Description
1	1d	NULL	MarketingVacuuming	NULL

Table 4.7: The contents of the `vacuumingPolicies` table after the vacuuming policy `MarketingVacuuming` has been added.

4.6.2 Executing Vacuuming Policies

After the storage policy `delete-after-five-years` has been fully defined, it can be enforced by executing the vacuuming policy `MarketingVacuuming`. This is achieved through the following command:

```
$ vacuuming-policy execute --vacuuming-policies MarketingVacuuming

Executing MarketingVacuuming...
Vacuuming policy 'MarketingVacuuming' executed
```

As a result `LastExecution` seen in Table 4.7 is updated with the current timestamp, and all values in the column `users.address` matching the condition in Table 4.6 are then set to the value `removed`, as dictated by the personal data column's default value.

4.7 Logging

The Logging component described in Section 3.4 is implemented as a plaintext-logger in DPCT. That is, for every action performed by DPCT, a log entry is appended to a plaintext file, using the syntax described in Section 3.4. This file can then be searched using the CLI.

The command that is used to search the log is `list` with options to narrow the search. These options are:

- `limit`, which limits the number of shown results to a given amount.
- `numbers`, which restricts the result to an inclusive range of log id values.
- `date-times`, which restricts result to an inclusive range of date-times.
- `log-types`, which restricts the result to a given list of log-types. In DPCT these are `Metadata` and `Vacuuming`.
- `log-formats`, which restricts the result to a given list of message formats. In DPCT these are `Plaintext` and `Json`.
- `subjects`, which restricts the result to a given list of subjects, i.e., the names or ids of metadata entities in DPCT.

The resulting log entries are sorted by their id, from lowest to highest. If the output is limited to less than the total amount of log entries in the result, the newest log entries, i.e., the log entries with the greatest ids are shown.

An example of a narrowed search is:

```
$ log list --limit 100 --numbers 5 250 --log-formats Plaintext Json
↪  --date-times 2013/04/26T12:00 2019/06/02T15:45:13 --log-types Vacuuming
↪  --subjects "(users, address)" "(users, name)"
```

which will show the 100 newest log entries that have ids between 5 and 250, have messages formatted using `Plaintext` or `Json`, were made between the 25th of April 2013 at 12:00 and the 2nd of June 2019 at 15:45:13, and were recording events related to vacuuming of the two columns `address` and `name` in the `users` table.

Examples of the logs produced by running commands are shown in the following two interactions:

```
$ personal-data-column create --table-column users address --default-value
 ↪  "removed" --description "The customer's address" --association-expression
 ↪  "users.id" --purposes Marketing

$ log list --numbers 4 5 --log-types Metadata

4   25-05-2023 10:10:53   Metadata   (users, address)   Plaintext   Personal
 ↪  data column '(users, address)' created
5   25-05-2023 10:10:53   Metadata   (users, address)   Plaintext   Personal
 ↪  data column '(users, address)' updated to '(users, address), The customer's
 ↪  address, removed, users.id, [ Marketing ]'
```

```
$ vacuuming-policy execute --vacuuming-policies BookkeepingVacuuming

$ log list --numbers 13 14 --log-types Vacuuming

13   25-05-2023 14:44:32   Vacuuming   MarketingVacuuming   Plaintext
 ↪  Vacuuming policy 'MarketingVacuuming' executed
14   25-05-2023 14:44:32   Vacuuming   (users, address)   Plaintext   Execution
 ↪  of vacuuming policy 'MarketingVacuuming' possible affected (users, address)
 ↪  because it is stored under the following purpose(s): Marketing. The
 ↪  following query was executed: "UPDATE users SET address = 'removed' WHERE
 ↪  EXISTS(SELECT u.id FROM users AS u JOIN newsletter AS n ON u.id = n.id
 ↪  WHERE subscribed = 0 AND u.id = users.id);"
```

Chapter 5

Evaluation

This chapter evaluates whether the Data Protection Compliance Tool (DPCT) satisfies the requirements defined in Section 1.3. This is done on a component basis, and each component is evaluated with respect to the requirements it must satisfy.

5.1 Setup

The running example introduced in Section 1.4 has been used throughout this report. To demonstrate and evaluate the functionality of the prototype, a database with the same schema has been seeded with 1000 random users generated using the tool Random User Generator [Armstrong, 2022]. This database contains a variety of customers, which have placed random orders on random dates and are subscribed at random to the newsletter.

The database file can be found at `https://github.com/P9-P10/DataProtectionComplianceTool/blob/main/EndToEndCommands/database.sqlite`.

The database initially contains the tables shown in Figure 1.1.

5.2 Metadata

As stated in Section 3.2, the metadata component must satisfy requirements 1, 2, and 3. It must also support fulfillment of requirement 4, by making it possible to determine when storing personal data is no longer necessary.

1. Personal data is only processed for legitimate and specific purposes.
2. Personal data is only processed on a legal basis.
3. Personal data is associated with an individual.
4. Personal data is only stored for as long as it is necessary to fulfil a purpose, and it is deleted as soon as that is no longer the case.

This section assesses the extent to which the metadata component of DPCT, described in Section 4.5, satisfies these requirements. In the following interactions, some of the output of the commands is omitted for brevity.

5.2.1 Processing With Specific Purposes

The first step is to register all the columns storing personal data, and can be seen in the following interaction:

```
$ personal-data-column create --table-column users username
$ personal-data-column create --table-column users name
$ personal-data-column create --table-column users address
$ personal-data-column create --table-column newsletter email
$ personal-data-column create --table-column orders delivery_address
```

Then it is possible to check whether all columns containing personal data are associated with at least one purpose using the `status` command. In the output from the interaction below "`...`" represents additional information that has been omitted.

```
$ status

Personal data column '(users, username)' is missing a purpose
...
Personal data column '(users, name)' is missing a purpose
...
Personal data column '(users, address)' is missing a purpose
...
Personal data column '(newsletter, email)' is missing a purpose
...
Personal data column '(orders, delivery_address)' is missing a purpose
...
```

Personal data with no purpose is not deleted because DPCT operates under the assumption that the data is used for a legitimate purpose that has not yet been specified. Should this turn out not to be the case, the names of the table and column are sufficient to enable users to delete this information manually.

In the running example presented in Section 1.4, the purpose for processing information about customers' email addresses is `Marketing`, and the purpose for processing customers' usernames, names, and addresses is `Bookkeeping`. These purposes can be created as demonstrated by the following interaction:

```
$ purpose create --name Marketing --legally-required false
$ purpose create --name Bookkeeping --legally-required true
```

These purposes can then be associated with the columns containing personal data, which is demonstrated by:

```
$ personal-data-column add-purpose --table-column users username --purposes
 ↪  Bookkeeping
$ personal-data-column add-purpose --table-column users name --purposes
 ↪  Bookkeeping Marketing
$ personal-data-column add-purpose --table-column users address --purposes
 ↪  Bookkeeping
$ personal-data-column add-purpose --table-column newsletter email --purposes
 ↪  Marketing
$ personal-data-column add-purpose --table-column orders delivery_address
 ↪  --purposes Bookkeeping
```

It is then possible to determine that all columns containing personal data are being processed for at least one purpose using the status command, thus fulfilling requirement 1. This also supports fulfillment of requirement 4 by the vacuuming component. This is covered in more detail in Section 5.3.

Requirement 2 is not fulfilled, as it is not possible to determine whether the processing of personal data has a legal basis. DPCT is based on the implicit assumption that all created purposes have a legal basis. The flag `legally-required` indicates whether the data must be stored in order to comply with legislation, i.e., the processing is based on a legal obligation. When this value is false, it is still legal to process the personal data, but it is not required by law. The consequences of this are discussed in Section 6.1.

5.2.2 Associating Personal Data With Individuals

Fulfilling requirement 3 requires associating the registered personal data with specific individuals. This is done by specifying an association expression, which under the current assumptions consists of the column in a table storing personal data that can be used to identify a specific individual, as described in Section 3.2.3. This must be done for each column storing personal data, as seen in the following interaction. In the case where a single table has multiple columns storing personal data, these columns will have identical association expressions. This is the case for the columns `users.username`, `users.name`, and `users.address`.

```
$ personal-data-column update --table-column users username
↪  --association-expression "users.id"

$ personal-data-column update --table-column users name
↪  --association-expression "users.id"

$ personal-data-column update --table-column users address
↪  --association-expression "users.id"

$ personal-data-column update --table-column newsletter email
↪  --association-expression "newsletter.id"

$ personal-data-column update --table-column orders delivery_address
↪  --association-expression "orders.ordered_by"
```

These association expressions can be used to construct queries that associate the values of a column containing personal data with the identifiers of the associated individuals. An example of such a query is shown in the following interaction with a DBMS managing the database described in Section 1.4.

```
SELECT individuals.id, newsletter.email
FROM users AS individuals
JOIN newsletter ON individuals.id = newsletter.id
ORDER BY individials.id;
```

The first ten rows of the result of the query are shown in the following table.

id	email
1	emil.olsen@example.com
2	signe.pedersen@example.com
3	emily.sorensen@example.com
4	naja.moller@example.com
5	katrine.hansen@example.com
6	albert.moller@example.com
7	ella.poulsen@example.com
8	nanna.petersen@example.com
9	victoria.olsen@example.com
10	freja.kristensen@example.com

The following query shows the case of associating personal data stored in the same table as the identifiers for individuals. As mentioned in Section 3.2 the table storing identifiers for individuals is given an alias when used in queries. It is not given an alias when used as a table containing a column storing personal data.

```sql
SELECT individuals.id, users.name
FROM users AS individuals
JOIN users ON individuals.id = users.id
ORDER BY individuals.id;
```

The first ten rows of the result of the query are shown in the following table.

id	name
1	Emil Olsen
2	Signe Pedersen
3	Emily Sorensen
4	Naja Moller
5	Katrine Hansen
6	Albert Moller
7	Ella Poulsen
8	Nanna Petersen
9	Victoria Olsen
10	Freja Kristensen

As these queries show, specifying an association expression for each column containing personal data allows for associating personal data with individuals thus fulfilling requirement 3 under the assumptions mentioned in Section 3.2.3.

5.3 Vacuuming

Section 3.3 specifies that the vacuuming component must fulfill requirement 4, which states:

4. Personal data is only stored for as long as it is necessary to fulfil a purpose, and it is deleted as soon as that is no longer the case.

Using the metadata for personal data shown in Section 5.2, it should be possible to define policies for when data should be deleted.

To evaluate whether DPCTis capable of this, a set of cases is presented. The cases describe three different scenarios, which show the base cases the system can handle:

1. Vacuuming data in a single table with a single purpose.
2. Vacuuming data in a single table with multiple purposes.
3. Vacuuming data in multiple tables with multiples purposes.

These cases use the purposes and columns defined in Section 1.4, which are sufficient to show the different cases the system can be used in. The cases use the database described in Section 1.4, and the entire database, including the stored metadata, is reset between each case.

These evaluations are not formal proofs of the system, as they are practical evaluations on a set of data. Therefore this evaluation is used to show that the implementation works as intended on the data used in the sample database. The formal proofs are beyond the scope of this project.

As the focus of this project is creating a prototype of a system fulfilling the requirements specified in Section 1.3, performance metrics, such as run-time of queries, the amount of updates etc. are not evaluated. These have not been measured as the implementation used for vacuuming in DPCT has not been optimized for any of these metrics.

5.3.1 Vacuuming of Data With a Single Purpose

The first case to verify, is whether the vacuuming process works for a single purpose defined on a set of columns in the same table. This is achieved by assigning a storage policy to the purpose `Marketing` through the following commands:

```
$ purpose create --name Marketing --description "Purpose for marketing"
↪  --legally-required false

$ personal-data-column create --table-column users name --description "User's
↪  name" --default-value "removed" --purposes Marketing

$ storage-policy create --name MarketingUsers --description "Storage policy for
↪  the use of customers' names for marketing" --table-column users name
↪  --vacuuming-condition "EXISTS(SELECT u.id FROM users u JOIN newsletter n ON
↪  u.id = n.id WHERE subscribed = 0 AND u.id = users.id )"

$ vacuuming-policy create --name MarketingVacuuming --description "Policy for
↪  vacuuming Marketing purpose" --purposes Marketing --duration "1d"

$ purpose add-storage-policy --name Marketing --storage-policies MarketingUsers
```

These commands create the purpose `Marketing`, and the personal data column `users.name` with a reference to the purpose, and a default value of 'removed'. The commands then create a storage policy, and a vacuuming policy for the purpose. After these have been created, the final command associates the storage policy with the purpose.

The storage policy specifies that values in the column `users.name` should be deleted if the associated customer is no longer subscribed to the newsletter, i.e., where subscribed is 0. The duration specified in the creation of the vacuuming policy, specifies that it should be automatically executed once every day.

The next step is to execute the vacuuming policy using the following command:

```
$ vacuuming-policy execute --vacuuming-policies MarketingVacuuming
```

After the vacuuming policy is executed, the database described in Section 1.4 should contain an amount of customers where the name has been replaced with ’removed’ equal to the amount of customers that are not subscribed, i.e., where subscribed is 0.

This can be verified by running the following queries. The first query selects all costumers from `users` where `subscribed = 0`. The second query selects the costumers which are not subscribed, and where the name equals ’removed’, as this is only the case once the vacuuming process has been executed. The last query then selects all rows where the name equals ’removed’. If these queries return the same values, the vacuuming has deleted only the correct tuples.

```
SELECT users.id, subscribed, name
FROM users
         JOIN newsletter AS n ON users.id = n.id
WHERE subscribed = 0
ORDER BY users.id;

SELECT users.id, subscribed, name
FROM users
         JOIN newsletter AS n ON users.id = n.id
WHERE subscribed = 0
  AND name = 'removed'
ORDER BY users.id;

SELECT users.id, subscribed, name
FROM users
         JOIN newsletter AS n ON users.id = n.id
WHERE name = 'removed'
ORDER BY users.id;
```

When executing the queries on the database all return the same result, showing the vacuuming component works as expected. The result of the queries consists of a total of 506 rows, the first ten of which can be seen in the following table:

id	subscribed	name
1	0	removed
2	0	removed
5	0	removed
17	0	removed
19	0	removed
21	0	removed
29	0	removed
30	0	removed
31	0	removed
32	0	removed

5.3.2 Vacuuming of Data With Multiple Purposes

To verify that vacuuming works with multiple purposes in effect at the same time, multiple purposes should be assigned to the same column. The purpose `Marketing` and personal data column `users.name` are defined in the same way as the previous case, and the following commands define a new purpose called `Bookkeeping` and associates it with `users.name`.

```
$ purpose create --name Bookkeeping --description "Purpose for bookkeeping"
 ↪  --legally-required true

$ personal-data-column add-purpose --table-column users name --purposes
 ↪  Bookkeeping

$ storage-policy create --name BookkeepingUsers --description "Storage policy
 ↪  for the use of customers' names for bookkeeping" --table-column users name
 ↪  --vacuuming-condition "EXISTS(SELECT u.id AS uid, MAX(order_date) AS
 ↪  last_order_date FROM orders JOIN users AS u ON u.id = ordered_by WHERE u.id
 ↪  = users.id GROUP BY ordered_by HAVING last_order_date <
 ↪  datetime('2023-06-02 13:00:00', '-5 year'))"

$ vacuuming-policy create --name BookkeepingVacuuming --description "Policy for
 ↪  vacuuming Bookkeeping purpose" --purposes Bookkeeping --duration "1d"

$ purpose add-storage-policy --name Bookkeeping --storage-policies
 ↪  BookkeepingUsers
```

The vacuuming condition defined for `Bookkeeping` above selects the costumers, where the date of the last order they made is more than five years old. A specific date has been used to ensure repeatability.

The rows that will be affected by the vacuuming conditions can be seen in the following table. These are the customers that are not subscribed to the newsletter and have not ordered anything within the last five years.

id	username	name	subscribed	last_order_date
55	angrygoose622	Lærke Olsen	0	2017-01-28 05:51:00
148	goldendog216	Marius Madsen	0	2018-03-31 07:16:00
223	blackswan387	Caroline Thomsen	0	2016-12-04 06:56:00
386	orangebird847	Mathias Christensen	0	2017-11-01 16:02:00
452	orangeladybug614	Tobias Johansen	0	2017-12-02 06:55:00
471	crazybear894	Rosa Mortensen	0	2016-09-06 22:43:00
600	sadfish549	Mille Madsen	0	2018-04-23 10:50:00
652	purpleladybug911	Marie Madsen	0	2018-04-27 21:45:00
662	ticklishduck171	Josefine Larsen	0	2018-01-28 13:18:00
710	happypeacock624	Simon Pedersen	0	2016-03-13 19:04:00
929	bigwolf576	Nanna Andersen	0	2017-08-15 19:56:00
949	silverkoala729	Frederikke Sørensen	0	2017-12-31 05:40:00

Then, the vacuuming policies for the associated purposes are executed by running the following command:

```
$ vacuuming-policy execute --vacuuming-policies MarketingVacuuming
↪  BookkeepingVacuuming
```

To verify that the correct columns have been vacuumed the following queries can be used:

```sql
SELECT users.id, username, name, subscribed, MAX(order_date) as last_order_date
↪
FROM users
         JOIN newsletter n ON users.id = n.id
         JOIN orders o ON users.id = o.ordered_by
WHERE subscribed = 0
GROUP BY ordered_by
HAVING last_order_date < datetime('2023-06-02 13:00:00', '-5 year') ORDER BY
↪  users.id;

SELECT users.id, username, name, subscribed, MAX(order_date) as last_order_date
↪
FROM users
         JOIN newsletter n ON users.id = n.id
         JOIN orders o ON users.id = o.ordered_by
WHERE subscribed = 0 AND name = 'removed'
GROUP BY ordered_by
HAVING last_order_date < datetime('2023-06-02 13:00:00', '-5 year') ORDER BY
↪  users.id;

SELECT *
FROM users
WHERE name = 'removed'
ORDER BY users.id;
```

The first query selects all rows that should have been vacuumed by the vacuuming process and is the combination of vacuuming conditions defined in the storage policies `BookkeepingUsers` and `MarketingUsers`. The method for combining vacuuming conditions is described in Section 3.3. The second query is identical, except it also checks whether the values of `name` have been updated to 'removed'. The third query selects all rows from `users`, where the `name` is set to 'removed'. As these three queries return the exact same costumers, i.e., the same values for `id`, the vacuuming component has updated exactly the rows that should be updated according to the vacuuming conditions.

The result of the queries is shown in the following table:

id	**username**	**name**	**subscribed**	**last_order_date**
55	angrygoose622	removed	0	2017-01-28 05:51:00
148	goldendog216	removed	0	2018-03-31 07:16:00
223	blackswan387	removed	0	2016-12-04 06:56:00
386	orangebird847	removed	0	2017-11-01 16:02:00
452	orangeladybug614	removed	0	2017-12-02 06:55:00
471	crazybear894	removed	0	2016-09-06 22:43:00
600	sadfish549	removed	0	2018-04-23 10:50:00
652	purpleladybug911	removed	0	2018-04-27 21:45:00
662	ticklishduck171	removed	0	2018-01-28 13:18:00
710	happypeacock624	removed	0	2016-03-13 19:04:00
929	bigwolf576	removed	0	2017-08-15 19:56:00
949	silverkoala729	removed	0	2017-12-31 05:40:00

5.3.3 Vacuuming of Data Across Multiple Tables

For the case of vacuuming multiple tables, the two purposes from the previous sections are defined in the same manner. To evaluate vacuuming across multiple tables, the column `newsletter.email` is registered as containing personal data and the purposes are associated with it:

```
personal-data-column create --table-column newsletter email --default-value
 ↪  "removed" --purposes Bookkeeping Marketing
```

Then storage policies are created for the `Marketing` and `Bookkeeping` purposes. As mentioned in Section 3.3, each table should have its own storage policies, as the vacuuming condition changes depending on the table on which it is used. After the storage policies have been created, they are related to the `Marketing` and `Bookkeeping` purposes, as seen in the following interaction:

```
storage-policy create --name MarketingNewsletter --table-column newsletter
 ↪  email --vacuuming-condition "EXISTS(SELECT * FROM newsletter AS n WHERE
 ↪  n.subscribed = 0 AND n.id = newsletter.id)"

storage-policy create --name BookkeepingNewsletter --table-column newsletter
 ↪  email --vacuuming-condition "EXISTS(SELECT *, MAX(order_date) AS
 ↪  last_order_date FROM newsletter AS n JOIN orders ON n.id = ordered_by WHERE
 ↪  n.id = newsletter.id GROUP BY ordered_by HAVING last_order_date <
 ↪  datetime('2023-06-02 13:00:00', '-5 year'))"

purpose add-storage-policy --name Marketing --storage-policies
 ↪  MarketingNewsletter

purpose add-storage-policy --name Bookkeeping --storage-policies
 ↪  BookkeepingNewsletter
```

The vacuuming condition defined for `BookkeepingNewsletter` selects the costumers that have made their last order more than five years from the defined date. A specific date has been used to ensure repeatability.

Before the vacuuming is executed, the rows that will be affected by the vacuuming conditions can be seen in the following table:

id	**email**	**subscribed**
55	laerke.olsen@example.com	0
148	marius.madsen@example.com	0
223	caroline.thomsen@example.com	0
386	mathias.christensen@example.com	0
452	tobias.johansen@example.com	0
471	rosa.mortensen@example.com	0
600	mille.madsen@example.com	0
652	marie.madsen@example.com	0
662	josefine.larsen@example.com	0
710	simon.pedersen@example.com	0
929	nanna.andersen@example.com	0
949	frederikke.sorensen@example.com	0

The policies can then be executed using:

```
$ vacuuming-policy execute --vacuuming-policies MarketingVacuuming
 ↪  BookkeepingVacuuming
```

To verify whether or not the vacuuming executed correctly, the content of the table `users` is verified in the same manner as in the last section, while the content of the `newsletter` table is verified using the following queries:

```
SELECT *
FROM newsletter
WHERE EXISTS(SELECT *, MAX(order_date) AS last_order_date
             FROM newsletter AS n
                      JOIN orders ON n.id = ordered_by
             WHERE n.id = newsletter.id
             AND n.subscribed = 0
             GROUP BY ordered_by
             HAVING last_order_date < datetime('2023-06-02 13:00:00',
 ↪  '-5 year'))
ORDER BY id;

SELECT *
FROM newsletter
WHERE EXISTS(SELECT *, MAX(order_date) AS last_order_date
             FROM newsletter AS n
                      JOIN orders ON n.id = ordered_by
             WHERE n.id = newsletter.id
             AND n.subscribed = 0
             AND email = 'removed'
             GROUP BY ordered_by
             HAVING last_order_date < datetime('2023-06-02 13:00:00',
 ↪  '-5 year'))
ORDER BY id;

SELECT * FROM newsletter
WHERE email = 'removed'
ORDER BY id;
```

The above queries are used to evaluate the result of the vacuuming in the same manner as in Section 5.3.2.

The output of the first two queries is shown in the following table:

id	email	subscribed
55	removed	0
148	removed	0
223	removed	0
386	removed	0
452	removed	0
471	removed	0
600	removed	0
652	removed	0
662	removed	0
710	removed	0
929	removed	0
949	removed	0

5.4 Logging

As stated in Section 3.4 the logging component should fulfill requirement 5:

5. Operations affecting personal data are logged such that regulators can inspect compliance.

That is, the log should be able to show that the database system complies with requirements 1 to 4 from Section 1.3. This will be shown through an examination of a log that has been produced after a number of commands, which can be found at the end of this section, have been executed. In the following interactions, the log entries contained in the examined log are referred to by their id number and the vacuuming conditions for the created storage policies have been replaced with keywords for brevity.

In this example, a user wants to know whether the personal data in the `name` column of the `users` table is stored in a compliant manner, i.e., whether it complies with the requirements 1 to 4.

5.4.1 Documenting Purpose

The user first inspects if the data is being processed for a valid purpose, and searches the log by looking for log entries associated with the personal data column (users, name):

```
$ log list --subjects "(users, name)"

1   05-06-2023 14:03:30   Metadata   (users, name)   Plaintext   Personal data
 ↪  column '(users, name)' created
2   05-06-2023 14:03:30   Metadata   (users, name)   Plaintext   Personal data
 ↪  column '(users, name)' updated to '(users, name), None, removed, users.id,
 ↪  Empty'
7   05-06-2023 14:03:30   Metadata   (users, name)   Plaintext   Personal data
 ↪  column '(users, name)' updated to '(users, name), None, removed, users.id,
 ↪  [ Marketing , Bookkeeping ]'
18   05-06-2023 14:03:30   Vacuuming   (users, name)   Plaintext   Execution of
 ↪  vacuuming policy 'AllVacuumingPolicy' possibly affected (users, name)
 ↪  because it is stored under the following purpose(s):  MarketingInformation ,
 ↪  Bookkeeping. The following query was executed: "UPDATE users SET name =
 ↪  'removed' WHERE (MarketingUsersCondition) AND
 ↪  (BookkeepingUsersCondition);"
```

From this interaction the user can see that the personal data column was (1) created, (2) updated with a default-value removed and an association expression users.id, (7) updated with two purposes Marketing and Bookkeeping, and (18) the subject of an execution of the vacuuming policy AllVacuumingPolicy. The user could conclude that the personal data is being processed under two purposes, Marketing and Bookkeeping. However, the user can also see that there is a discrepancy between the logged purposes for the personal data column in log entry 7 and 18, marked in yellow, as the purpose Marketing has been replaced with the purpose MarketingInformation. The user can inspect this:

```
$ log list --subjects Marketing MarketingInformation

3   05-06-2023 14:03:30   Metadata   Marketing   Plaintext   Purpose
 ↪  'Marketing' created
4   05-06-2023 14:03:30   Metadata   Marketing   Plaintext   Purpose
 ↪  'Marketing' updated to 'Marketing, None, False, Empty, Empty, Empty'
12   05-06-2023 14:03:30   Metadata   Marketing   Plaintext   Purpose
 ↪  'Marketing' updated to 'Marketing, None, False, [ MarketingUsers ], [
 ↪  (users, name) ], Empty'
16   05-06-2023 14:03:30   Metadata   Marketing   Plaintext   Purpose
 ↪  'Marketing' updated to 'MarketingInformation, None, False, [ MarketingUsers
 ↪  ], [ (users, name) ], [ AllVacuumingPolicy ]'
19   05-06-2023 14:03:30   Metadata   MarketingInformation   Plaintext
 ↪  Purpose 'MarketingInformation' deleted
```

From this interaction the user can see that a purpose called `Marketing` was (3) created, (4) updated with a legally required value `False`, (12) updated with a storage policy `MarketingUsers`, and (16) updated with a new name `MarketingInformation` and a vacuuming policy `AllVacuumingPolicy`. From this it is possible to determine why log entry 18 has a different purpose listed, as `Marketing` has been renamed to `MarketingInformation`. Furthermore, from log entry 19 the user can see that the purpose `MarketingInformation` has been deleted. As a result, the user can conclude that the personal data stored in the `name` column of the `users` table is no longer being processed for the purpose of `Marketing`. However, from the previous interaction, the user found that the personal data was also being processed for the purpose of `Bookkeeping`. The user can continue the inspection to see if the purpose `Bookkeeping` still exists:

```
$ log list --subjects Bookkeeping

5   05-06-2023 14:03:30   Metadata   Bookkeeping   Plaintext   Purpose
 ↪  'Bookkeeping' created
6   05-06-2023 14:03:30   Metadata   Bookkeeping   Plaintext   Purpose
 ↪  'Bookkeeping' updated to 'Bookkeeping, None, True, Empty, Empty, Empty'
13   05-06-2023 14:03:30   Metadata   Bookkeeping   Plaintext   Purpose
 ↪  'Bookkeeping' updated to 'Bookkeeping, None, True, [ BookkeepingUsers ], [
 ↪  (users, name) ], Empty'
```

From this interaction the user can see that the purpose `Bookkeeping` has been created and updated with various values, but it has not been renamed or deleted. Therefore the user can conclude that the personal data stored in the `name` column of the `users` table is still being processed under the `Bookkeeping` purpose.

5.4.2 Documenting Vacuuming

Now that it has been established that the personal data is being processed for a valid purpose, the user can inspect the log to see if the personal data is being regularly vacuumed. From log entry 18 the user can see that the personal data column "(users, name)" was affected by the execution of the vacuuming policy AllVacuumingPolicy. The user can inspect this vacuuming policy further:

```
$ log list --subjects AllVacuumingPolicy

14   05-06-2023 14:03:30   Metadata   AllVacuumingPolicy   Plaintext
 ↪  Vacuuming policy 'AllVacuumingPolicy' created
15   05-06-2023 14:03:30   Metadata   AllVacuumingPolicy   Plaintext
 ↪  Vacuuming policy 'AllVacuumingPolicy' updated to 'AllVacuumingPolicy, None,
 ↪  1d, None, [ Marketing, Bookkeeping ]'
17   05-06-2023 14:03:30   Vacuuming   AllVacuumingPolicy   Plaintext
 ↪  Vacuuming policy 'AllVacuumingPolicy' executed
```

From this interaction the user can see that the vacuuming policy AllVacuumingPolicy was (14) created, (15) updated to have a duration of "1d", i.e., one day and have the two purposes Marketing and Bookkeeping, and (17) executed. Because the id of log entry 17 is less than the id of log entry 19, the user can also conclude that personal data processed for the purpose of Marketing was vacuumed once before the purpose was deleted.

From log entry 13 and 16 the user can see that two storage policies have been in use, namely MarketingUsers and BookkeepingUsers. The user can inspect these further:

```
$ log list --subjects MarketingUsers BookkeepingUsers

8   05-06-2023 14:03:30   Metadata   MarketingUsers   Plaintext   Storage
 ↪  policy 'MarketingUsers' created
9   05-06-2023 14:03:30   Metadata   MarketingUsers   Plaintext   Storage
 ↪  policy 'MarketingUsers' updated to 'MarketingUsers, None,
 ↪  MarketingUsersCondition , (users, name), Empty'
10   05-06-2023 14:03:30   Metadata   BookkeepingUsers   Plaintext   Storage
 ↪  policy 'BookkeepingUsers' created
11   05-06-2023 14:03:30   Metadata   BookkeepingUsers   Plaintext   Storage
 ↪  policy 'BookkeepingUsers' updated to 'BookkeepingUsers, None,
 ↪  BookkeepingUsersCondition , (users, name), Empty'
```

From this interaction the user can see that the two storage policies were (8 and 10) created and (9 and 11) updated with a vacuuming condition, excluded here for brevity, and that they are associated with the personal data column "(users, name)".

5.4.3 Summary

From this example examination, the user can conclude the following about the personal data stored in the `name` column of the `users` table:

1. The personal data can be associated with an individual, as the association expression `users.id` is available (see log entry 2).
2. The personal data is being processed for the purpose of `Bookkeeping`. Previously it was also being processed for the purpose of `Marketing`.
3. The personal data is being processed on the legal basis of a legal obligation, because the `legally-required` flag on the `Bookkeeping` purpose is set to `True` (see log entry 13).
4. The personal data is regularly vacuumed, because the vacuuming policy `AllVacuuming-Policy` has a duration of one day. The user can inspect the two vacuuming conditions *MarketingUsersCondition* and *BookkeepingUsersCondition* to see if they are behaving correctly.

This example examination shows that it is possible to verify that the personal data is being processed in compliance with GDPR. This was just one example, but it shows that the necessary information is available to verify compliance, within the context of the data DPCT has access to.

These are the commands that had been run prior to the example examination:

```
$ personal-data-column create --table-column users name
 ↪  --association-expression "users.id" --default-value "removed"

$ purpose create --name Marketing --legally-required false
$ purpose create --name Bookkeeping --legally-required true

$ personal-data-column add-purpose --table-column users name --purposes
 ↪  Marketing Bookkeeping

$ storage-policy create --name MarketingUsers --table-column users name
 ↪  --vacuuming-condition MarketingUsersCondition

$ storage-policy create --name BookkeepingUsers --table-column users name
 ↪  --vacuuming-condition BookkeepingUsersCondition

$ purpose add-storage-policy --name Marketing --storage-policies MarketingUsers

$ purpose add-storage-policy --name Bookkeeping --storage-policies
 ↪  BookkeepingUsers

$ vacuuming-policy create --name AllVacuumingPolicy --purposes Marketing
 ↪  Bookkeeping --duration "1d"

$ purpose update --name Marketing --new-name MarketingInformation

$ vacuuming-policy execute --vacuuming-policies AllVacuumingPolicy

$ purpose delete --name MarketingInformation
```

Chapter 6

Discussion

Based on the design described in Chapter 3 and the evaluation of DPCT seen in Chapter 5. The components of DPCT are discussed in the following sections. First the implementation and design of metadata is discussed, followed by a discussion of the vacuuming process. Finally logging is discussed.

6.1 Metadata

6.1.1 Legal Basis

DPCT implements a simplified representation of the legal basis required by GDPR to process personal data lawfully. It is simplified as it can only describe whether or not personal data is being processed on the legal basis of "legal obligation". However, GDPR Article 6(1) presents a number of possible legal bases for the processing of personal data that can be used by companies. As a result, if personal data managed by DPCT is not being processed on the basis of a legal obligation, i.e., the `legally required` flag on the `purpose` is `false`, users of DPCT have no way of specifying which, if any, of the other available legal bases are in use. DPCT currently assumes that some legal basis other than legal obligation is in use, such as consent (GDPR Article 6(1)(a)), but it has no way of verifying this.

A more elaborate approach to representing legal bases is presented in Section 8.1.

6.1.2 Associating Personal Data with Individuals

In Section 3.2.3, four assumptions were made to make it possible to associate personal data with individuals:

- **I** Individuals are represented in the database system by a unique identifier.
- **II** The identifiers representing individuals are single-column, i.e, not multi-column.
- **III** The identifiers are stored in a single table known by DPCT.
- **IV** A one-to-many relationship exists between an identifier and the personal data of the individual that identifier represents.

Relaxing assumptions **I** and **IV** is not possible, as they are fundamental to the notion of associating unique individuals with their personal data. Without a relationship between an individual and that individual's personal data, the association is impossible to make.

Relaxing assumptions **II** and **III** could possibly require more work on the user's part when constructing the one-to-many relationship, as the user would also have to define a query for combining the tables storing the, now possibly multi-column, identifiers, before the relationship can be built. In many cases, this could be done by using one or more UNION and FULL JOIN operations, but there might be cases where that would not be enough, and the user would have to define a more complex operation. Another option would be to make the database conform to the assumptions, using schema evolution tools such as Curino et al. [2008, 2010]; Nykjær et al. [2023].

6.1.3 Circumventing DPCT

DPCT does not force a company to use it. As an example, it would be possible for a company to process data for a different purpose than the one registered in DPCT. However, it is in the best interest of the company to register these purposes in DPCT, as it allows them to show compliance.

In the future, it would be prudent to expand DPCT to function as a layer between the DBMS and the database, requiring all access to go through DPCT, which is then logged for auditing purposes.

6.1.4 Personal Data Granularity

As mentioned in Section 3.2, personal data is managed at the column level. This is based on the assumption that all personal data stored in a column has the same nature, e.g., every value stored in an `address` column is an address, and it was therefore assumed that they would also share the same purposes for processing.

However, this assumption may not always hold, and it may be necessary to manage personal data at the level of individual cells in a column. As an example, it might be necessary to specify that one half of the addresses is stored under the `Marketing` purpose and the other half is stored under the `Bookkeeping` purpose, which would not be possible under the current assumption.

Another option is managing personal data at the table level. This is a more practical representation for some metadata. An example of this is the association expression. The association expression is the same for all columns in the same table, as each row in a table containing personal data must be associated with a single individual.

As this discussion shows, there are arguments for managing personal data at different levels of granularity. As a result, a third option is to combine these different granularities, such that information relevant to all columns in a table can be stored in a single place while it is also possible to specify details about individual columns and cells in a column.

An extension supporting this is presented in Section 8.1.

6.2 Vacuuming

6.2.1 Run-Time of Combined Conditions

As shown in Algorithm 1, the algorithm responsible for creating the combined vacuuming conditions for a purpose iterates through every column related to a purpose, and then iterates through all purposes related to those columns. As a result, the run time of the system is proportional with the amount of columns associated with a purpose, and the amount of purposes associated with each column. As optimization of DPCT is beyond the scope of this project, it has not been a focus to optimize this algorithm.

6.2.2 Query Analysis

DPCT does not perform any analysis of the combined vacuuming conditions. Therefore, if a user was to create an equivalent vacuuming condition for multiple purposes associated with the same column, i.e., multiple purposes expiring after two years, these conditions will both be added to the combined vacuuming condition for the column. This results in the same condition being checked multiple times, resulting in a larger overhead. Therefore it would be prudent analyze and optimize the combined queries, as described in Section 8.2.

6.2.3 Deletion Guarantees

A user can define arbitrary queries as discussed in Section 3.3, and as a result, they can create combinations of vacuuming policies which never delete any data. However, this is a result of allowing a user to create the conditions that fit their needs, without any restrictions, and is therefore an accepted side effect.

DPCT also allows a user to define arbitrary default values for columns. This allows a user more freedom, but can result in them attempting to set a column containing integers to a string. To alleviate this issue, it would be prudent to allow the user to define the datatype of a column, such that DPCT can determine whether the value is of the correct type. For certain database systems, such as SQLite and PostgreSQL, this issue is not as prevalent, as SQLite has dynamic types [SQLite, 2023b] and PostgreSQL has built-in casting between certain types [The PostgreSQL Global Development Group, 2023a]. However, relying on the specific DBMSs and their way of handling types is not as extendable as implementing the functionality in DPCT.

6.3 Logging

6.3.1 Logging Vacuuming Executions

DPCT is currently not able to determine if a vacuuming execution actually affected data, which is why the log entry for vacuuming says the vacuuming *possibly* affected data. This is due to DPCT not accessing the personal data that is stored in the database system. As a result, when running the vacuuming statements, it can not detect if the data was affected by the deletion statement.

As an example, if a user creates a vacuuming condition which is set to `false`, no data in the system is effected by that condition. However, DPCT will still state that the data has *possibly* been affected.

One way of detecting whether data is affected by vacuuming, is to execute a query that selects the data that should be deleted, and then determining whether the result is empty. If this is not the case, the vacuuming statement updates the stored data. However, this would result in extra queries made on the database, and therefore a larger vacuuming overhead.

Another option is to use a function such as `RETURNING`, which returns the updated columns [SQLite, 2023c]. This would enable the DPCT to verify whether the result is non empty.

Implementing one of these methods would result in more complete logs which would be able to state that a specific purpose has been validated and a certain number of columns were modified.

6.3.2 External Data Manipulation

DPCT does not log external queries and updates to the database. This makes it possible for someone with access to the database to, as an example, update the `creation_dates` in the `users` table to the current date, then execute vacuuming, which is logged, and then set the `creation_dates` to their original values. The result would be an unchanged database, but the log would show that vacuuming has been run. This way, it would be possible to build an audit trail that shows compliance, in this instance that vacuuming has been run, but in reality the data would not be in compliance.

As a result of the logging component in DPCT not logging external queries and updates, it is not possible to utilize it as the sole documentation of compliance for a company, as an auditor would need to compare the DPCT log with the DBMS log.

Methods for expanding DPCT to include logging external queries and updates are described in Chapter 8.

6.3.3 Logging Guarantees

DPCT creates a new log entry each time a command has been successfully executed. If the command failed to execute, no log entry is created. This differs from the standard write-ahead logging technique, where every operation is logged before it is performed [The PostgreSQL Global Development Group, 2023b]. Write-ahead logging is good for crash-recovery as it is possible to rebuild the data in the database should something fail during an operation. However, the log maintained in DPCT is used to document the changes made to the data, and not the planned operations, as the latter can lead to tampering. As an example, if a user ran a command and then deliberately made the database system crash after the log entry had been made, but before the command had an effect on the database, the log would show that the command had gone through, which the user could then present to regulators as documentation for compliance.

However, a problem also arises when using this technique. If a database operation is successful, but appending to the log fails, the log and the state of the data in the database system would be out of sync, and the log would represent an incorrect audit trail.

Ideally a transaction-style system should be used, where either both the database operation and log append succeed, or both fail.

6.4 Support for Complying with the Rights of Individuals

This project was approached from the perspective of a company trying to make their database system GDPR compliant. Providing built-in support for individuals to exercise their rights was therefore not a priority. However, it is also in the interest of a company to be able to comply with individuals' requests, as this is necessary in order to be GDPR compliant.

Using DPCT as it is, it is possible for the company to comply with individuals' requests, as DPCT allows for a user of the system to associate individuals with their personal data using the association expressions. However, this requires some work on the company's part, as the process of complying with individuals' requests regarding their GDPR rights has not been automated. Enabling this automation is described in further detail in Chapter 8.

6.5 Process

While the previous sections discuss aspects of the design and implementation of DPCT, this section discusses relevant aspects of the process of developing DPCT.

The development of DPCT started with a specification of the schema for the database storing the metadata, and the functionality that should use this information. The information represented by the schema was correct, however the structure and relations were based on assumptions that turned out to be incorrect. This resulted in the schema being a hindrance to the development of the functionality. Several iterations of the database schema have been developed, and the problems have been mitigated. The vacuuming process is an example of functionality that remains more complex than necessary because the data structures it requires underwent multiple iterations and the component had to be adapted.

During the initial stages of the project, the project group reflected on the progress of the project every second week. However, once development of the prototype began, these reflection days stopped. The initial reflection days made this possible, as little changed regarding the project direction during development, but it would still have been beneficial during development, with regards to the final stages of the project, by providing more opportunities for refinement.

The prototype was developed iteratively using test driven development. The practice of test driven development provided benefits, such as revealing inconsistencies between components early. However, the resulting interfaces are more complex and in some cases more tightly coupled than necessary. The reason for this is that insufficient attention was paid to refactoring and simplifying the interfaces.

The prototype has more functionality than is presented in the report. This functionality partially implements or supports some of the features described in Chapter 8, and has been omitted to limit the scope of the report to functionality that has been fully implemented.

Chapter 7

Conclusion

The General Data Protection Regulation (GDPR) requires that companies only store and process personal data so long as they have specific purposes with valid legal bases for doing so. In this project, a Data Protection Compliance Tool (DPCT) is designed, implemented, and evaluated. DPCTaims to help companies make their existing database systems GDPR compliant. Requirements for this system are based on GDPR and existing research on GDPR's effects on database systems. These requirements are used to determine the additional information that must be provided about personal data in order to ensure that it is stored in compliance with GDPR.

DPCT supports the capture of this additional information by associating all personal data with metadata that define the purposes for storing it, its associated individual, and when it should be deleted.

DPCT allows a user to register columns in an existing database as containing personal data. After a column is registered as containing personal data, DPCT can give an overview of the metadata needed to ensure that the personal data is stored in compliance with GDPR. DPCT enables its users to register the metadata and vacuuming policies needed to document that the personal data is being processed for legitimate and specific purposes, can be associated with a natural person, and is deleted when it is no longer being processed for a valid purpose.

The prototype of DPCT fulfills the requirements defined for the system, except for the requirement *Legal processing*, which requires that personal data must be stored with a legal basis. It is only possible to determine the legal basis on which personal data is being processed using DPCT if the legal basis is a legal obligation.

The design and implementation of DPCT presented in this report can assist companies in complying with specific aspects of GDPR. However, this design can be extended to provide support for additional aspects. Several extensions are presented as future work, including fulfilling the missing requirement, providing built-in support for the rights of individuals, and extending logging to support verification of external changes to personal data.

Chapter 8

Future Work

8.1 Metadata

8.1.1 Personal Data Granularity

DPCT represents personal data as references to specific columns. As already discussed in Section 6.1, this representation has limitations. It requires information about a table storing personal data to be repeated across all references to columns in the table, and it is not possible to represent metadata about individual values or subsets of values in a column.

These limitations can be removed by using a different representation of personal data that supports describing personal data with multiple levels of granularity. Figure 8.1 shows a schema with tables that allows relating metadata to tables, columns, rows, and cells containing personal data.

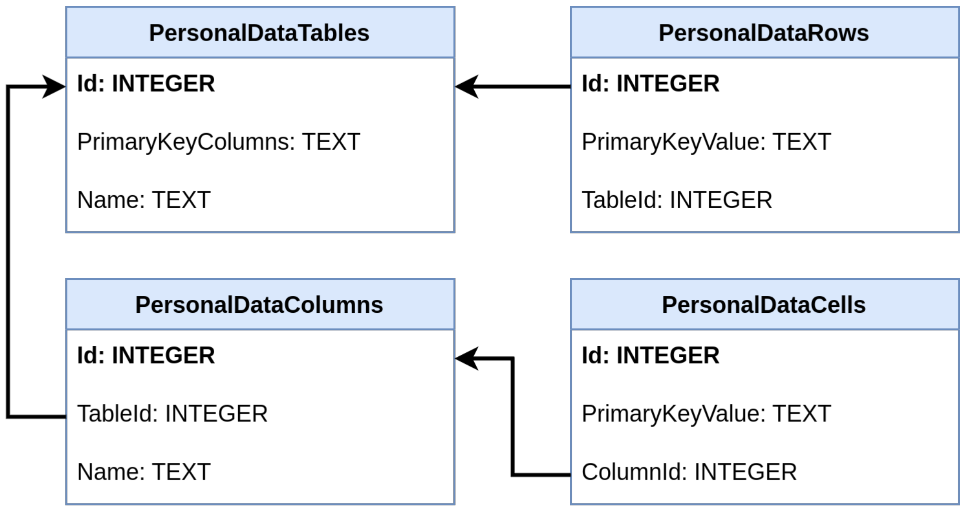

Figure 8.1: An overview of a schema that allows for representing multiple granularities of personal data.

8.1.2 Extended Metadata

Only the metadata necessary to determine whether personal data can continue to be stored is included in DPCT. To fully comply with GDPR, additional metadata is required.

A more elaborate representation of legal basis is needed, as the current representation of the legal basis for the processing of personal data is simplified as mentioned in Section 6.1. A complete representation of legal basis requires an auditor be able to determine which legal basis applies to a given purpose, and verify that it is valid. This can be accomplished by replacing the `legally_required` flag in the table `purposes` with a reference to a specific legal basis.

Information about the source of personal data about an individual is also required (GDPR Article 15). As the source may differ between individuals, this requires a way to relate metadata to a specific individual. In the case where the source differs between values of personal data for the same individual, it requires being able to relate metadata to a specific value of personal data. This would be possible using the more granular representation of personal data described previously.

Similarly to the source of personal data, it is also required that a company keep track of what personal data is shared, and what entities it is shared with (GDPR Article 15). This can be done by creating a table with the possible entities, and relating these to the personal data shared with them.

It also required that a company be able to inform individuals about how their personal data is processed (GDPR Article 15), and if it is used for automated decision making (GDPR Article 22). This requires changing and extending the current representation of metadata. A table recording the usages of personal data needs to be added. Personal data should then be associated with one or more usages, and purposes should be associated with usages instead of personal data. It would then be possible to determine how personal data is processed, and for what purpose. A flag indicating whether it performs automated decision making can be added to the table recording usages.

8.2 Vacuuming

8.2.1 Optimization of Combined Conditions

As mentioned in Section 6.2, DPCT currently does not conduct any analysis on the combined conditions created in the vacuuming component. As a result, the combined condition can result in a statement containing the same condition multiple times. This could be alleviated using the chase and back-chase algorithm described in [Curino et al., 2008].

The pseudo code presented in Algorithm 1 describes the procedure for generating the combined condition. It is not part of the scope of the project to optimize this algorithm, and it would therefore be prudent to focus on reducing the run-time of the system in the future work. This could be done using memoisation techniques.

8.2.2 Periodic Vacuuming

Periodic vacuuming is not implemented in DPCT as this was deemed outside the scope of the project. However, there are different methods of implementing this functionality.

One of these methods is using an in-process scheduler, which can be implemented by starting a scheduled thread running within DPCT that periodically deletes data. However, this method requires that the process is constantly running to ensure that the data is deleted.

Another method is to generate a set of cron jobs, which is a functionality used for running commands at regular intervals on a Unix system [Kubernetes, 2023]. These cron jobs should be set to execute based on the duration defined in the vacuuming policy, and contain the combined query which is needed to vacuum the database. This query will have the same structure as the one seen in Listing 3.2. However, this means that whenever a new purpose, and vacuuming policies are added, the cron jobs must be updated. This requires system access to the device executing the cron jobs.

A third option is to develop a system that runs on a server and periodically executes the vacuuming policies. This process must have access to the database, and so that it is able to adapt to new purposes and vacuuming policies being added.

Of the three above mentioned possibilities, the third option is preferred for expanding DPCT, as it allows for periodic vacuuming without being dependent on DPCT running in the background, and without having to create, manage, and update cron jobs.

8.3 Logging

8.3.1 Logging External Queries and Updates

As mentioned in Section 6.3, DPCT does not log external queries and updates. This could be achieved by recording queries in the DBMS log, copying and merging it with the information logged by DPCT, similarly to the approach of Kraska et al. [2019]. Not every DBMS supports logging, such as SQLite. However, several relevant DBMSs do, such as the ones analysed by Shastri et al. [2019a].

Without copying the DBMS log, it would only be possible to log external queries and updates if the system had access to every external query and update. The method of implementing this depends on the DBMS. Some, such as PostgreSQL, can be extended to support this functionality. Alternatively, a proxy can be used to provide access to queries and updates.

Logging all queries would enable DPCT to determine who is using which data. Logging every update would allow DPCT to prevent the circumvention mentioned in Section 6.1. An example of how this prevents circumvention, is that it enables determining if a value was changed just before a vacuuming execution or status report, and then changed back afterwards.

8.3.2 Personal Data and Logging

If all queries and updates made to a database system were logged, the logs would also contain personal data about the individuals stored in the system. As a result, there would be a need to ensure that personal data is removed from the logs as well. This could be achieved by either vacuuming the log or anonymising the personal data stored in the log, such that it cannot be referred back to particular individuals.

8.4 Better Support for the Rights of Individuals

As mentioned in Section 6.4, providing support for the rights of individuals is not a priority in this project. However, the requirements already fulfilled by DPCT provides the foundation necessary to provide better support for complying with these rights. Being able to respond to users exercising their rights is also in the interest of the company, as it is needed in order to be fully GDPR compliant. Automating the responses can save the company time and money, as they will not have to manually respond to each request.

Because the rights of individuals are based on the individuals making requests to the company, functionality supporting individuals' ability to make such requests is required. This can be done by expanding DPCT to allow individuals access to a limited set of functionality.

The rights to data access (GDPR Article 15), data portability (GDPR Article 20), and to being informed of automated decision making (GDPR Article 22), require that the individuals can make requests to receive their personal data and the associated metadata. DPCT stores the relevant GDPR metadata, and it is possible to construct queries using the association expressions on `personal data columns` to extract all personal data for a given individual from the database system.

The rights to erasure (GDPR Article 17) and rectification (GDPR Article 16) require that the individuals can make requests to have their personal data changed. In both cases the purpose for processing the personal data must be taken into account before the changes are made, and the company should therefore be able to review the requests before they are acted upon. However, there may also be many cases, where the data can be rectified or deleted without needing approval. As an example, if personal data is being processed on the legal basis of consent only, the data can be deleted immediately as the company is obligated to comply with the request.

The rights to object (GDPR Article 21), restriction of processing (GDPR Article 18), and to not be subject to automated processing (GDPR Article 22), require that individuals can make requests to limit the processing of their personal data. This may be difficult to fully automate, as it is up to the company how exactly they will react. As an example, it might be the case that the company has legitimate reasons for not complying with a request. However, it would be possible for DPCT to log the date and time of the arrival of the request, and notify the company that they need to react in a timely manner.

Bibliography

Archita Agarwal, Marilyn George, Aaron Jeyaraj, and Malte Schwarzkopf. Retrofitting GDPR compliance onto legacy databases. *Proceedings of the VLDB Endowment.*, 15(4):958–970, apr 2022. URL `https://doi.org/10.14778/3503585.3503603`.

Keith Armstrong. Random user generator, 2022. URL `https://randomuser.me/`. Accessed 02/06-2023.

Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo. Graceful database schema evolution: The prism workbench. *Proceedings of the VLDB Endowment*, 1(1):761–772, 2008.

Carlo A. Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. Update rewriting and integrity constraint maintenance in a schema evolution support system: PRISM++. *Proceedings of the VLDB Endowment*, 4(2):117–128, 2010.

Datatilsynet. Møbelfirma indstillet til bøde, 2019a. URL `https://www.datatilsynet.dk/afgoerelser/afgoerelser/2019/jun/moebelfirma-indstillet-til-boede`. Accessed 07/06-2023.

Datatilsynet. Datatilsynet indstiller taxaselskab til bøde på 1,2 mio. kr., 2019b. URL `https://www.datatilsynet.dk/afgoerelser/afgoerelser/2019/mar/datatilsynet-indstiller-taxaselskab-til-boede-paa-12-mio-kr`. Accessed 07/06-2023.

Datatilsynet. Arp-hansen hotel group a/s indstilles til bøde, 2020. URL `https://www.datatilsynet.dk/afgoerelser/afgoerelser/2020/jul/arp-hansen-hotel-group-as-indstilles-til-boede`. Accessed 07/06-2023.

Datatilsynet. Danske bank indstilles til bøde, 2022a. URL `https://www.datatilsynet.dk/afgoerelser/afgoerelser/2022/apr/danske-bank-indstilles-til-boede`. Accessed 07/06-2023.

Datatilsynet. Gyldendal indstilles til bøde, 2022b. URL `https://www.datatilsynet.dk/afgoerelser/afgoerelser/2022/jun/gyldendal-indstilles-til-boede`. Accessed 07/06-2023.

Datatilsynet. Gdpr for små virksomheder, 2023. URL `https://www.datatilsynet.dk/presse-og-nyheder/nyhedsarkiv/2023/maj/gdpr-for-smaa-virksomheder`. Accessed 07/06-2023.

European Commission. 2016 reform of eu data protection rules, 2016. URL `https://eur-lex.europa.eu/eli/reg/2016/679/oj`.

Tim Kraska, Michael Stonebraker, Michael Brodie, Sacha Servan-Schreiber, and Daniel Weitzner. Schengendb: A data protection database proposal. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare*, pages 24–38. Springer International Publishing, 2019. URL https://doi.org/10.1007/978-3-030-33752-0_2.

Kubernetes. Cronjob, 2023. URL https://kubernetes.io/docs/concepts/workloads/controllers/cron-jobs/. Accessed 30/05-2023.

Mendeley. Harvard format citation guide, 2021. URL https://www.mendeley.com/guides/harvard-citation-guide. Accessed 11/11-2022.

Microsoft. System.commandline overview, 2022. URL https://learn.microsoft.com/en-us/dotnet/standard/commandline/. Accessed 11/06-2023.

Alexander Mundbjerg Nykjær, Ane Søgaard Jørgensen, and Jakob Sønderby Kristensen. Towards practical regulatory compliance in database systems, 2023.

Anton Hinsby Palmer and Sujeepan Srikandarajah. Design and implementation of a system for rule-based data retention compliance, 2022. Report not publicly available.

Retsinformation. Bekendtgørelse af bogføringslov, 2006. URL https://www.retsinformation.dk/eli/lta/2006/648.

Supreeth Shastri, Vinay Banakar, Melissa Wasserman, Arun Kumar, and Vijay Chidambaram. Understanding and benchmarking the impact of GDPR on database systems. *arXiv preprint arXiv:1910.00728*, 2019a.

Supreeth Shastri, Melissa Wasserman, and Vijay Chidambaram. The seven sins of personal-data processing systems under GDPR. In *11th USENIX Workshop on Hot Topics in Cloud Computing*. USENIX Association, 2019b. URL https://www.usenix.org/conference/hotcloud19/presentation/shastri.

SQLite. What is sqlite?, 2023a. URL https://www.sqlite.org/index.html. Accessed 29/05/2023.

SQLite. Datatypes in sqlite, 2023b. URL https://www.sqlite.org/datatype3.html. Accessed 05/06-2023.

SQLite. Returning, 2023c. URL https://sqlite.org/lang_returning.html. Accessed 03/06-2023.

The PostgreSQL Global Development Group. Value storage, 2023a. URL https://www.postgresql.org/docs/current/typeconv-query.html. Accessed 05/06-2023.

The PostgreSQL Global Development Group. 30.3. write-ahead logging (wal), 2023b. URL https://www.postgresql.org/docs/current/wal-intro.html. Accessed 07/06-2023.

Appendix A

Command Line Interface

This appendix presents the commands that are available in DPCT. Commands that have similar structure and functionality are not repeated, but a reference to the similar command is given.

- Section A.1 presents an overview of all available commands.
- Section A.2 covers the `personal-data-column` command.
- Section A.3 covers the `purpose` command.
- Section A.4 covers the `vacuuming-policy` command.
- Section A.5 covers the `storage-policy` command.
- Section A.6 covers the `log` command.

The commands are presented using the built-in help command, which is invoked by using the `?` option, such as:

```
$ personal-data-column create ?
```

A.1 Overview

```
$ ?

Commands:
  pdc, personal-data-column
  p, purpose
  vacuuming-policy, vp
  sp, storage-policy
  lg, log
  q, quit                      Quits the program
  stat, status                 Shows the status of all entities in the system
```

Listing A.1: Overview of the available commands.

A.2 Personal Data Column Command

A.2.1 Overview

- Section A.2.2 covers the `personal-data-column delete` command.
- Section A.2.3 covers the `personal-data-column list` command.
- Section A.2.4 covers the `personal-data-column show` command.
- Section A.2.5 covers the `personal-data-column status` command.
- Section A.2.6 covers the `personal-data-column create` command.
- Section A.2.7 covers the `personal-data-column update` command.
- Section A.2.8 covers the `personal-data-column add-purpose` command.
- Section A.2.9 covers the `personal-data-column remove-purpose` command.

```
$ personal-data-column ?
Description:

Usage:
  ! personal-data-column [command] [options]

Options:
  ?, h, help  Show help and usage information

Commands:
  d, delete           Deletes the given personal data column from the system
  list, ls            Lists the personal data columns currently in the system
  sh, show            Shows details about the given personal data column
  stat, status        Shows the statuses of the personal data columns currently
   ↪  in the system
  c, create           Creates a new personal data column in the system
  u, update           Updates the given personal data column with the given
   ↪  values
  add-purpose, ap     Adds the given purposes to the personal data column
  remove-purpose, rp  Removes the given purposes from the personal data column
```

Listing A.2: Overview of the `personal-data-column` command.

A.2.2 Delete

```
$ personal-data-column delete ?
Description:
  Deletes the given personal data column from the system

Usage:
  ! personal-data-column delete [options]

Options:
  -tc, --table-column <table-column> (REQUIRED)  The table and column in which
   ↪  the personal data is stored
  ?, h, help                                     Show help and usage
   ↪  information
```

Listing A.3: The `personal-data-column delete` command.

A.2.3 List

```
$ personal-data-column list ?
Description:
  Lists the personal data columns currently in the system

Usage:
  ! personal-data-column list [options]

Options:
  ?, h, help  Show help and usage information
```

Listing A.4: The `personal-data-column list` command.

A.2.4 Show

```
$ personal-data-column show ?
Description:
  Shows details about the given personal data column

Usage:
  ! personal-data-column show [options]

Options:
  -tc, --table-column <table-column> (REQUIRED)  The table and column in which
   ↪  the personal data is stored
  ?, h, help                                     Show help and usage
   ↪  information
```

Listing A.5: The `personal-data-column show` command.

A.2.5 Status

```
$ personal-data-column status ?
Description:
  Shows the statuses of the personal data columns currently in the system

Usage:
  ! personal-data-column status [options]

Options:
  ?, h, help  Show help and usage information
```

Listing A.6: The `personal-data-column status` command.

A.2.6 Create

```
$ personal-data-column create ?
Description:
  Creates a new personal data column in the system

Usage:
  ! personal-data-column create [options]

Options:
  -tc, --table-column <table-column> (REQUIRED)           The table and column
   ↪  in which the personal data is stored
  -d, --description <description>                         The description of
   ↪  the personal data column
  -dv, --default-value <default-value>                    The default value
   ↪  that attributes in the column should receive upon deletion
  -ae, --association-expression <association-expression>  The expression that
   ↪  describes how this personal data column should be associated with the
   ↪  table containing individuals
  -ps, --purposes <purposes>                              The purpose(s) under
   ↪  which the personal data is stored
  ?, h, help                                              Show help and usage
   ↪  information
```

Listing A.7: The `personal-data-column create` command.

A.2.7 Update

```
$ personal-data-column update ?
Description:
  Updates the given personal data column with the given values

Usage:
  ! personal-data-column update [options]

Options:
  -tc, --table-column <table-column> (REQUIRED)          The table and column
   ↪  in which the personal data is stored
  -d, --description <description>                        The description of
   ↪  the personal data column
  -dv, --default-value <default-value>                   The default value
   ↪  that attributes in the column should receive upon deletion
  -ae, --association-expression <association-expression>  The expression that
   ↪  describes how this personal data column should be associated with the
   ↪  table containing individuals
  ?, h, help                                             Show help and usage
   ↪  information
```

Listing A.8: The `personal-data-column update` command.

A.2.8 Add Purpose

```
$ personal-data-column add-purpose ?
Description:
  Adds the given purposes to the personal data column

Usage:
  ! personal-data-column add-purpose [options]

Options:
  -tc, --table-column <table-column> (REQUIRED)  The table and column in which
   ↪  the personal data is stored
  -ps, --purposes <purposes>                     The purpose(s) under which the
   ↪  personal data is stored
  ?, h, help                                     Show help and usage
   ↪  information
```

Listing A.9: The `personal-data-column add-purpose` command.

A.2.9 Remove Purpose

```
$ personal-data-column remove-purpose ?
Description:
  Removes the given purposes from the personal data column

Usage:
  ! personal-data-column remove-purpose [options]

Options:
  -tc, --table-column <table-column> (REQUIRED)  The table and column in which
   ↪  the personal data is stored
  -ps, --purposes <purposes>                     The purpose(s) under which the
   ↪  personal data is stored
  ?, h, help                                     Show help and usage
   ↪  information
```

Listing A.10: The `personal-data-column remove-purpose` command.

A.3 Purpose Command

A.3.1 Overview

- Section A.3.2 covers the `purpose delete` command.
- Section A.3.3 covers the `purpose list` command.
- Section A.3.4 covers the `purpose show` command.
- Section A.3.5 covers the `purpose status` command.
- Section A.3.6 covers the `purpose create` command.
- Section A.3.7 covers the `purpose update` command.
- Section A.3.8 covers the `purpose add-storage-policy` command.
- Section A.3.9 covers the `purpose remove-storage-policy` command.

```
$ purpose ?
Description:

Usage:
  ! purpose [command] [options]

Options:
  ?, h, help  Show help and usage information

Commands:
  d, delete                    Deletes the given purpose from the system
  list, ls                     Lists the purposes currently in the system
  sh, show                     Shows details about the given purpose
  stat, status                 Shows the statuses of the purposes currently in
   ↪  the system
  c, create                    Creates a new purpose in the system
  u, update                    Updates the given purpose with the given values
  add-storage-policy, as       Adds the given storage-policys to the purpose
  remove-storage-policy, rs  Removes the given storage-policys from the purpose
```

Listing A.11: Overview of the `purpose` command.

A.3.2 Delete

```
$ purpose delete ?
Description:
  Deletes the given purpose from the system

Usage:
  ! purpose delete [options]

Options:
  -n, --name <name> (REQUIRED)  The name of the purpose
  ?, h, help                    Show help and usage information
```

Listing A.12: The `purpose delete` command.

A.3.3 List

Same as Section A.2.3 but with the `purpose` keyword instead of `personal-data-column`.

A.3.4 Show

```
$ purpose show ?
Description:
  Shows details about the given purpose

Usage:
  ! purpose show [options]

Options:
  -n, --name <name> (REQUIRED)  The name of the purpose
  ?, h, help                    Show help and usage information
```

Listing A.13: The `purpose show` command.

A.3.5 Status

Same as Section A.2.5 but with the `purpose` keyword instead of `personal-data-column`

A.3.6 Create

```
$ purpose create ?
Description:
  Creates a new purpose in the system

Usage:
  ! purpose create [options]

Options:
  -n, --name <name> (REQUIRED)                  The name of the purpose
  -d, --description <description>               The description of the purpose
  -lr, --legally-required                       Whether the purpose falls under
   ↪  any legal obligations
  -sps, --storage-policies <storage-policies>  The storage policies which
   ↪  personal data stored under this purpose should follow
  ?, h, help                                    Show help and usage information
```

Listing A.14: The `purpose create` command.

A.3.7 Update

```
$ purpose update ?
Description:
  Updates the given purpose with the given values

Usage:
  ! purpose update [options]

Options:
  -n, --name <name> (REQUIRED)     The name of the purpose
  -nn, --new-name <new-name>       The new name of the purpose
  -d, --description <description>  The description of the purpose
  -lr, --legally-required          Whether the purpose falls under any legal
   ↪  obligations
  ?, h, help                       Show help and usage information
```

Listing A.15: The `purpose update` command.

A.3.8 Add Storage Policy

```
$ purpose add-storage-policy ?
Description:
  Adds the given storage-policys to the purpose

Usage:
  ! purpose add-storage-policy [options]

Options:
  -n, --name <name> (REQUIRED)                 The name of the purpose
  -sps, --storage-policies <storage-policies>  The storage policies which
   ↪  personal data stored under this purpose should follow
  ?, h, help                                   Show help and usage information
```

Listing A.16: The `purpose add-storage-policy` command.

A.3.9 Remove Storage Policy

```
$ purpose remove-storage-policy ?
Description:
  Removes the given storage-policys from the purpose

Usage:
  ! purpose remove-storage-policy [options]

Options:
  -n, --name <name> (REQUIRED)                  The name of the purpose
  -sps, --storage-policies <storage-policies>  The storage policies which
   ↪  personal data stored under this purpose should follow
  ?, h, help                                   Show help and usage information
```

Listing A.17: The `purpose remove-storage-policy` command.

A.4 Vacuuming Policy Command

A.4.1 Overview

- Section A.4.2 covers the `vacuuming-policy delete` command.
- Section A.4.3 covers the `vacuuming-policy list` command.
- Section A.4.4 covers the `vacuuming-policy show` command.
- Section A.4.5 covers the `vacuuming-policy status` command.
- Section A.4.6 covers the `vacuuming-policy create` command.
- Section A.4.7 covers the `vacuuming-policy update` command.
- Section A.4.9 covers the `vacuuming-policy add-purpose` command.
- Section A.4.10 covers the `vacuuming-policy remove-purpose` command.

```
$ vacuuming-policy ?
Description:

Usage:
  ! vacuuming-policy [command] [options]

Options:
  ?, h, help  Show help and usage information

Commands:
  d, delete           Deletes the given vacuuming policy from the system
  list, ls            Lists the vacuuming policys currently in the system
  sh, show            Shows details about the given vacuuming policy
  stat, status        Shows the statuses of the vacuuming policys currently in
   ↪  the system
  c, create           Creates a new vacuuming policy in the system
  u, update           Updates the given vacuuming policy with the given values
  e, execute          Executes the given vacuuming policies
  add-purpose, ap     Adds the given purposes to the vacuuming policy
  remove-purpose, rp  Removes the given purposes from the vacuuming policy
```

Listing A.18: Overview of the `vacuuming-policy` command.

A.4.2 Delete

Same as Section A.3.2 but with the `vacuuming-policy` keyword instead of `purpose`.

A.4.3 List

Same as Section A.2.3 but with the `vacuuming-policy` keyword instead of `personal-data-column`.

A.4.4 Show

Same as Section A.3.4 but with the `vacuuming-policy` keyword instead of `purpose`.

A.4.5 Status

Same as Section A.2.5 but with the `vacuuming-policy` keyword instead of `personal-data-column`

A.4.6 Create

```
$ vacuuming-policy create ?
Description:
  Creates a new vacuuming policy in the system

Usage:
  ! vacuuming-policy create [options]

Options:
  -n, --name <name> (REQUIRED)     The name of the vacuuming policy
  -d, --description <description>  The description of the vacuuming policy
  -dur, --duration <duration>      The duration between vacuuming policy
   ↪  executions
  -ps, --purposes <purposes>       The purpose(s) under which the personal data
   ↪  is stored
  ?, h, help                       Show help and usage information
```

Listing A.19: The `vacuuming-policy create` command.

A.4.7 Update

```
$ vacuuming-policy update ?
Description:
  Updates the given vacuuming policy with the given values

Usage:
  ! vacuuming-policy update [options]

Options:
  -n, --name <name> (REQUIRED)     The name of the vacuuming policy
  -nn, --new-name <new-name>       The new name of the vacuuming policy
  -d, --description <description>  The description of the vacuuming policy
  -dur, --duration <duration>      The duration between vacuuming policy
   ↪  executions
  ?, h, help                       Show help and usage information
```

Listing A.20: The `vacuuming-policy update` command.

A.4.8 Execute

```
$ vacuuming-policy execute ?
Description:
  Executes the given vacuuming policies

Usage:
  ! vacuuming-policy execute [options]

Options:
  -vps, --vacuuming-policies <vacuuming-policies> (REQUIRED)  The names of the
   ↪  vacuuming policies that should be executed
  ?, h, help                                                  Show help and
   ↪  usage information
```

Listing A.21: The `vacuuming-policy execute` command.

A.4.9 Add Purpose

Same as Section A.2.8 but with the `vacuuming-policy` keyword instead of `personal-data-column`

A.4.10 Remove Purpose

Same as Section A.2.9 but with the `vacuuming-policy` keyword instead of `personal-data-column`

A.5 Storage Policy Command

A.5.1 Overview

- Section A.5.2 covers the `storage-policy delete` command.
- Section A.5.3 covers the `storage-policy list` command.
- Section A.5.4 covers the `storage-policy show` command.
- Section A.5.5 covers the `storage-policy status` command.
- Section A.5.6 covers the `storage-policy create` command.
- Section A.5.7 covers the `storage-policy update` command.

```
$ storage-policy ?
Description:

Usage:
  ! storage-policy [command] [options]

Options:
  ?, h, help  Show help and usage information

Commands:
  d, delete     Deletes the given storage policy from the system
  list, ls      Lists the storage policys currently in the system
  sh, show      Shows details about the given storage policy
  stat, status  Shows the statuses of the storage policys currently in the
   ↪  system
  c, create     Creates a new storage policy in the system
  u, update     Updates the given storage policy with the given values
```

Listing A.22: Overview of the `storage-policy` command.

A.5.2 Delete

Same as Section A.3.2 but with the `storage-policy` keyword instead of `purpose`.

A.5.3 List

Same as Section A.2.3 but with the `storage-policy` keyword instead of `personal-data-column`.

A.5.4 Show

Same as Section A.3.4 but with the `storage-policy` keyword instead of `purpose`.

A.5.5 Status

Same as Section A.2.5 but with the `storage-policy` keyword instead of `personal-data-column`

A.5.6 Create

```
$ storage-policy create ?
Description:
  Creates a new storage policy in the system

Usage:
  ! storage-policy create [options]

Options:
  -n, --name <name> (REQUIRED)                       The name of the storage
   ↪  policy
  -d, --description <description>                    The description of the
   ↪  storage policy
  -vc, --vacuuming-condition <vacuuming-condition>  The condition that must be
   ↪  fulfilled for data to be deleted
  -tc, --table-column <table-column>                 The data that will be
   ↪  vacuumed under the condition
  ?, h, help                                         Show help and usage
   ↪  information
```

Listing A.23: The `storage-policy create` command.

A.5.7 Update

```
$ storage-policy update ?
Description:
  Updates the given storage policy with the given values

Usage:
  ! storage-policy update [options]

Options:
  -n, --name <name> (REQUIRED)                         The name of the storage
   ↪  policy
  -nn, --new-name <new-name>                           The new name of the storage
   ↪  policy
  -d, --description <description>                      The description of the
   ↪  storage policy
  -vc, --vacuuming-condition <vacuuming-condition>  The condition that must be
   ↪  fulfilled for data to be deleted
  -tc, --table-column <table-column>                   The data that will be
   ↪  vacuumed under the condition
  ?, h, help                                           Show help and usage
   ↪  information
```

Listing A.24: The `storage-policy update` command.

A.6 Log Command

A.6.1 Overview

- Section A.6.2 covers the `log list` command.

```
$ log ?
Description:

Usage:
  ! log [command] [options]

Options:
  ?, h, help  Show help and usage information

Commands:
  list, ls  Lists the logs that fall within the given constraints
```

Listing A.25: Overview of the `log` command.

A.6.2 List

```
$ log list ?
Description:
  Lists the logs that fall within the given constraints

Usage:
  ! log list [options]

Options:
  -li, --limit <limit>                                       Limits results to
   ↪  the number given [default: 100]
  -n, --numbers <numbers>                                    Limits results to
   ↪  the specified numbers range (inclusive). Must provide two numbers as
   ↪  range (e.g. -n 3 6), first minimum then maximum [default: 02147483647]
  -d, --date-times <date-times>                              Limits results to
 ↪  the specified time range (inclusive). Must provide two date times as range
 ↪  (e.g. -d 2000/04/28T12:34:56 3000/06/16T09:38:12), first minimum then
 ↪  maximum [default: 01/01/0001 00:00:0031/12/9999
                                                             23:59:59]
  -lt, --log-types <MetadataSchemaChangeSystemVacuuming>  Limits results to the
   ↪  specified log type(s). [default: SchemaChangeVacuumingMetadataSystem]
  -s, --subjects <subjects>                                  Limits results to
   ↪  the specified subject(s). []
  -lf, --log-formats <JsonPlaintextTurtle>                 Limits results to
   ↪  the specified log format(s) [default: JsonPlaintextTurtle]
  ?, h, help                                                 Show help and
   ↪  usage information
```

Listing A.26: The `log list` command.