Universität Rostock

Traditio et Innovatio

Integration von Anonymisierungsverfahren in ETL-Prozesse für semistrukturierte Daten

Name:	Mohammad Alabdullah
Matrikelnummer:	2212 201684
Abgabedatum:	21.03.2025
Betreuer und Gutachter:	Dr.-Ing. Hannes Grunert Universität Rostock Albert-Einstein-Str.22 18059 Rostock
Gutachter:	Felix Hauptmann, M.Sc. Universität Rostock Albert-Einstein-Str.22 18059 Rostock

Bachelorarbeit am Institut für Informatik
Universität Rostock

Zusammenfassung

Die Verarbeitung und Analyse personenbezogener Daten stellt in vielen Anwendungsbereichen eine große Herausforderung dar, insbesondere im Hinblick auf Datenschutzbestimmungen wie die Datenschutz-Grundverordnung (DSGVO). Diese Arbeit untersucht die Integration von Anonymisierungsverfahren in Extract (Extrahieren), Transform (Transformieren) und Load (Laden) (ETL)-Prozesse für semistrukturierte Daten, um sowohl den Datenschutz als auch die Datenqualität zu gewährleisten. Der Fokus liegt auf der Anonymisierung von Extensible Markup Language (XML)-Daten und deren Speicherung in einem relationalen Format sowie der Anwendung von k-Anonymität und ℓ-Diversität zum Schutz sensibler Attribute. Zur Umsetzung wurde eine Pipeline entwickelt, die Daten extrahiert, transformiert und anonymisiert, bevor sie für die weitere Verarbeitung gespeichert werden. Die Evaluierung zeigt, dass eine Kombination aus Maskierung oder Generalisierung mit den Schutzmodulen einen effektiven Schutz bietet, während die Daten weiterhin für Analysen nutzbar bleiben. Die Ergebnisse verdeutlichen, dass eine adaptive Anonymisierungsstrategie, die sich dynamisch an die Datenstruktur anpasst, den besten Kompromiss zwischen Datenschutz und Datenqualität bietet. Zukünftige Arbeiten könnten diesen Ansatz durch Machine-Learning-gestützte Klassifikationsmethoden oder Differential Privacy erweitern.

Inhaltsverzeichnis

1 Einleitung

Diese Forschungsarbeit befasst sich mit der Konzeption und Entwicklung eines ETL-Modells, das Anonymisierungsverfahren integriert, um den Datenschutz bei der Verarbeitung von personenbezogenen Daten zu gewährleisten. Der ETL-Prozess (Extraktion, Transformation und Laden) ist ein wesentlicher Schritt in der Datenverarbeitung, der die Erfassung von Daten aus verschiedenen Quellen, deren strukturierte Organisation sowie die Vorbereitung für die Analyse umfasst. In dieser Arbeit liegt der Fokus darauf, Anonymisierungsverfahren gezielt in diesen Prozess zu integrieren, um den Anforderungen an den Datenschutz gerecht zu werden, ohne die spätere Nutzung und Analyse der Daten zu stark einzuschränken. Der ETL-Prozess wird in drei Hauptphasen unterteilt [14]:

1. **Extraktion:** In diesem ersten Schritt des ETL-Prozesses werden Daten aus unterschiedlichen Quellen gesammelt. Diese Quellen können Datenbanken, Cloud-Systeme, CRM-Systeme oder bestehende Datenbanken sein. Dabei kann es sich um strukturierte, semistrukturierte oder unstrukturierte Daten handeln. Die Extraktion erfolgt in zwei Varianten:
   - **Volle Extraktion:** Die Daten werden komplett übernommen, ohne dass Änderungen vorgenommen werden.
   - **Teilweise Extraktion:** Nur ausgewählte Daten werden extrahiert, entweder mit oder ohne Aktualisierungsbenachrichtigungen. Ziel ist es, Rohdaten so zu erfassen, dass sie für den nächsten Schritt bereit sind.

2. **Transformation:** Die extrahierten Daten sind in ihrem ursprünglichen Zustand oft nicht direkt verwendbar. Im Transformationsschritt werden die Rohdaten daher bereinigt, standardisiert, dedupliziert und sortiert. Außerdem werden sie durch festgelegte Regeln oder Berechnungen angepasst, um die Datenqualität sicherzustellen. Diese Regeln können beispielsweise auf Unternehmensanforderungen basieren, etwa zur Erfüllung von Berichtsvorgaben. Daten, die keine Anpassungen benötigen, werden direkt weitergegeben. Dieser Schritt ist essenziell, um die Datenintegrität zu gewährleisten und sie für das Zielsystem vorzubereiten.

3. **Laden:** Im letzten Schritt des ETL-Prozesses werden die transformierten Daten in das Zielsystem, häufig ein Data Warehouse, geladen. Dieses kann in Formaten wie Textdateien, Excel, JSON oder XML (eXtensible Markup Language) vorliegen oder eine relationale Datenbank sein. Der Ladeprozess überträgt alle transformierten Daten in das Ziel, sodass sie dort für Analysen und Berichte genutzt werden können.

1.1 Problemstellung und Zielsetzung

Die Verarbeitung und Analyse von personenbezogenen Daten ist in vielen Bereichen der Datenverarbeitung unerlässlich, jedoch unterliegt sie strengen Datenschutzvorgaben, insbesondere in der Europäischen Union durch die Datenschutz-Grundverordnung (DSGVO). Der Extraktions-, Transformations- und Ladeprozess (ETL) stellt dabei einen grundlegenden Schritt dar, um Daten aus unterschiedlichen Quellen in eine strukturierte und für die Analyse geeignete Form zu überführen. In Fällen, in denen personenbezogene Daten verarbeitet werden, muss jedoch sichergestellt werden, dass der Datenschutz zu jedem Zeitpunkt gewährleistet ist. Ein zentrales Problem bei der Verwendung von ETL-Prozessen ist, dass die Anonymisierung von personenbezogenen Daten häufig erst nach der Datenextraktion und -transformation durchgeführt wird. Dieser Ansatz kann jedoch zu Problemen führen, da Anonymisierungsmaßnahmen in einem späten Stadium des Prozesses dazu führen können, dass wichtige Analysen erschwert oder sogar unmöglich gemacht werden. Darüber hinaus ist eine pauschale Anonymisierung, die von der eigentlichen Analyse getrennt erfolgt, möglicherweise zu restriktiv und verhindert den vollen Nutzen aus den Daten. Das Ziel dieser Bachelorarbeit ist es, eine Methode zu entwickeln und zu untersuchen, die es ermöglicht, Anonymisierungsverfahren direkt in den ETL-Prozess zu integrieren. Hierbei soll insbesondere untersucht werden, an welchem Punkt im ETL-Prozess die Anonymisierung am sinnvollsten und effektivsten durchgeführt werden kann. Die Arbeit fokussiert sich auf die Transformation von semi-strukturierten XML-Daten in ein relationales Format (CSV) und die Integration von Anonymisierungsverfahren in diesem spezifischen Transformationsprozess. Das Hauptziel ist es, ein Anonymisierungsverfahren zu entwickeln, das:

- Datenschutzanforderungen während des gesamten ETL-Prozesses berücksichtigt,
- die Datenqualität und -nutzung nicht unnötig einschränkt,
- in der praktischen Anwendung flexibel und skalierbar ist.

Durch diese Untersuchung sollen Lösungsansätze für eine datenschutzgerechte, aber gleichzeitig effiziente Handhabung von semi-strukturierten Daten im Rahmen von ETL-Prozessen erarbeitet werden.

1.2 Forschungsfragen

Im Rahmen dieser Bachelorarbeit werden die folgenden Forschungsfragen untersucht, die sich auf die Integration von Anonymisierungsverfahren in den ETL-Prozess für semi-strukturierte Daten beziehen:

1. **Welche Methoden eignen sich zur Extraktion von Daten aus verschiedenen Quellen?**
   Diese Frage untersucht die verschiedenen Ansätze und Techniken, die zur Extraktion von Daten, insbesondere aus semi-strukturierten Quellen wie XML, im Rahmen eines ETL-Prozesses verwendet werden können.

2. **An welchem Punkt des ETL-Prozesses sollte die Anonymisierung durchgeführt werden, um sowohl den Datenschutz als auch die Datenverarbeitungsziele zu wahren?**
   Hier wird ermittelt, an welchem Schritt des ETL-Prozesses – Extraktion, Transformation oder Laden – die Anonymisierung am besten integriert werden kann, um den Datenschutzanforderungen gerecht zu werden, ohne die Datenqualität zu beeinträchtigen.

3. **Welche Verfahren eignen sich am besten, um Daten im Rahmen des ETL-Prozesses zu anonymisieren?**
   Diese Frage fokussiert sich auf die Auswahl und Anwendung geeigneter Anonymisierungsverfahren wie Maskierung, Generalisierung oder Aggregation, die sicherstellen, dass personenbezogene Daten geschützt werden.

4. **Welche Auswirkungen hat die Anonymisierung auf die Qualität und Verwendbarkeit der Daten?**
   In dieser Frage wird untersucht, wie sich die Anonymisierung auf die Integrität und die spätere Nutzung der Daten auswirkt und ob es Kompromisse zwischen Datenschutz und der Nutzung der Daten für Analysen oder andere Anwendungen gibt.

1.3 Aufbau der Arbeit

Die vorliegende Arbeit ist in sechs Hauptkapitel unterteilt, die jeweils einen spezifischen Aspekt der Thematik behandeln. Zunächst wird in **Kapitel 1: Einleitung** das Thema der Arbeit eingeführt. Die Problemstellung und Zielsetzung werden dargestellt, um die Relevanz des Themas zu verdeutlichen. Außerdem werden die Forschungsfragen formuliert, die im Rahmen dieser Arbeit beantwortet werden sollen. Der Aufbau der Arbeit wird ebenfalls vorgestellt, um dem Leser eine Orientierung über die Struktur der Arbeit zu geben. In **Kapitel 2: Grundlagen** werden die grundlegenden Konzepte und Technologien vorgestellt, die für das Verständnis der Arbeit notwendig sind. Dabei wird ein Überblick über den ETL-Prozess gegeben, die Bedeutung und Merkmale von semistrukturierten Daten erklärt sowie verschiedene Anonymisierungsverfahren beschrieben, die für die Anonymisierung der Daten relevant sind. **Kapitel 3: Stand der Technik** befasst sich mit der Analyse der bestehenden Forschung und der aktuellen Werkzeuge, die im Kontext der Arbeit von Bedeutung sind. Hier wird das Vorgehen bei der Literatur- und Werkzeugsuche erläutert, und die erzielten Ergebnisse werden detailliert dargestellt, insbesondere in Bezug auf Werkzeuge zur Dateiextraktion, Datenanonymisierung und Datenspeicherung. In **Kapitel 4: Konzeption des Verfahrens** wird das Konzept für das zu entwickelnde Verfahren vorgestellt. Dabei wird beschrieben, welche Datenquellen verwendet werden, wie die Dataextraktion erfolgt und welche Schritte der Datenvorverarbeitung und -formatierung notwendig sind. Anschließend wird die Integration der Anonymisierungsverfahren erläutert, bevor der Prozess mit der Speicherung der anonymisierten Daten abschließt. **Kapitel 5: Implementierung** widmet sich der

praktischen Umsetzung des entwickelten Verfahrens. Es wird beschrieben, welche Technologien für die Implementierung genutzt wurden. Der Fokus liegt auf den konkreten Schritten der Dataextraktion, der Anonymisierung und der Speicherung der Daten. Am Ende des Kapitels werden die Ergebnisse der Implementierung zusammengefasst. In **Kapitel 6: Schlussfolgerung** werden die wesentlichen Ergebnisse der Arbeit zusammengefasst und ein Ausblick auf mögliche zukünftige Entwicklungen und Erweiterungen des Verfahrens gegeben.

2 Grundlagen

In diesem Kapitel werden die theoretischen Grundlagen behandelt, die für das Verständnis der folgenden Abschnitte essenziell sind. Zunächst steht der ETL-Prozess im Mittelpunkt, da er eine zentrale Rolle in der Datenverarbeitung spielt und die Basis für die Einbindung von Anonymisierungsverfahren bildet. Anschließend wird auf semistrukturierte Daten eingegangen, die als die Datenquellen für diese Arbeit dienen. Zum Abschluss folgt eine Übersicht der wichtigsten Anonymisierungsverfahren, die in dieser Arbeit zum Schutz personenbezogener Daten angewendet werden. Damit legt dieses Kapitel das Fundament für die späteren Ausführungen und praktischen Implementierungen.

2.1 Überblick über ETL-Prozesse

In diesem Abschnitt wird der ETL-Prozess als zentraler Bestandteil der Datenintegration beschrieben. Der Fokus liegt auf der Definition und den Hauptzielen des ETL-Prozesses sowie auf einer detaillierten Darstellung der drei zentralen Phasen: Extraktion, Transformation und Laden.

2.1.1 Definition und Ziele von ETL-Prozessen

Der Begriff **ETL** steht für *Extrahieren, Transformieren, Laden* und bezeichnet einen etablierten Prozess der Datenintegration, der Daten aus verschiedenen Quellen extrahiert, transformiert und in ein Zielsystem überführt. Ziel dieses Prozesses ist es, Daten aus heterogenen und oft unstrukturierten Quellen zu integrieren und für analytische und geschäftliche Zwecke nutzbar zu machen [20]. ETL ist ein datenverarbeitender Prozess, der es ermöglicht, große Datenmengen effizient zu bereinigen, zu konsolidieren und in ein einheitliches Format zu überführen. Dies stellt sicher, dass die Daten den Anforderungen des Zielsystems entsprechen und für Analysen sowie Entscheidungsfindungen bereitstehen [20]. Dabei spielt die Datenqualität eine entscheidende Rolle, um sicherzustellen, dass die Daten konsistent, genau und vollständig sind. Darüber hinaus wird durch den ETL-Prozess die Nutzbarkeit der Daten gewährleistet, indem sie für analytische, geschäftliche und regulatorische Anforderungen vorbereitet werden. Schließlich trägt der ETL-Prozess zur Effizienz bei, indem die Datenverarbeitung automatisiert und optimiert wird. Ein gut konzipierter ETL-Prozess bildet die Grundlage für datengetriebene Anwendungen, da er sicherstellt, dass die Daten korrekt verarbeitet und in einer Form bereitgestellt werden, die sich für Business-Intelligence-, Machine-Learning- und andere datenintensive Workflows eignet [20].

2.1.2 Phasen eines ETL-Prozesses

Der ETL-Prozess besteht aus drei zentralen Phasen: Extraktion, Transformation und Laden. Jede Phase erfüllt eine spezifische Funktion im Rahmen der Datenintegration und trägt dazu bei, dass die Daten in einer konsistenten und nutzbaren Form im Zielsystem bereitgestellt werden [20].

Extraktion

In der ersten Phase des ETL-Prozesses werden Daten aus unterschiedlichen Quellen wie Datenbanken, APIs, Dateien oder anderen Systemen extrahiert. Diese Quellen können sowohl strukturierte als auch unstrukturierte Daten enthalten. Ziel dieser Phase ist es, die Rohdaten in ihrer ursprünglichen Form zu erfassen und in einen temporären Bereich zu kopieren, um sie für die nächste Phase vorzubereiten [20].

Transformation

Nach der Extraktion werden die Daten in der Transformationsphase bereinigt, standardisiert und in eine einheitliche Struktur gebracht. Dieser Schritt umfasst verschiedene Aufgaben, die sicherstellen, dass die Daten den Anforderungen des Zielsystems und den vorgesehenen Anwendungsfällen entsprechen. Zunächst werden in der Datenbereinigung Duplikate entfernt, Inkonsistenzen behoben und fehlende Werte ergänzt. Anschließend erfolgt die Datenanpassung, bei der Datentypen, Formatierungen und Maßeinheiten konvertiert werden, um eine konsistente Darstellung zu gewährleisten. Ein weiterer zentraler Schritt ist die Aggregation und Berechnung, bei der Daten zusammengefasst oder neue Felder berechnet werden, die für weiterführende Analysen benötigt werden. Durch diese Schritte wird sichergestellt, dass die transformierten Daten in einer qualitativ hochwertigen und nutzbaren Form vorliegen [20]. Im Folgenden werden die einzelnen Schritte anhand von Beispielen erläutert:

1. **Datenbereinigung:**
   - *Entfernung von Duplikaten:* In einer Kundenliste existieren doppelte Einträge für denselben Kunden:

     ```
     1. Max Mustermann, max@example.com
     2. Max Mustermann, max@example.com
     ```

     Nach der Bereinigung:

     ```
     1. Max Mustermann, max@example.com
     ```

   - *Behebung von Inkonsistenzen:* In einer Produktdatenbank gibt es verschiedene Schreibweisen für dasselbe Produkt:

     ```
     1. 17.04.1997
     2. 04/17/1997
     3. 1997/04/17
     ```

Nach der Bereinigung:

```
17.04.1995
```

- *Ergänzung fehlender Werte:* Eine Tabelle mit Verkäufen enthält fehlende Preise:

Artikel	**Preis**
Produkt X	10 €
Produkt Y	-

Nach der Transformation: z. B. anhand eines Durchschnittswerts ergänzt.

Artikel	**Preis**
Produkt X	10 €
Produkt Y	15 €

2. **Datenanpassung:**
   - *Konvertierung von Datentypen:* Geburtsdaten im falschen Format:

     ```
     "25.12.1985" (Textformat)
     ```

     Nach der Konvertierung:

     ```
     1985-12-25 (Datumsformat)
     ```

   - *Formatierung und Maßeinheiten:* Temperaturen werden in Fahrenheit angegeben:

     ```
     Temperatur: 68°F
     ```

     Nach der Transformation:

     ```
     Temperatur: 20°C
     ```

3. **Aggregation und Berechnung:**
   - *Datenaggregation:* Tägliche Verkaufszahlen werden zu Monatsumsätzen zusammengefasst:

Tag	**Umsatz**
01.01	100 €
02.01	150 €
.....	.....

Nach der Aggregation:

Tag	Umsatz
Januar	5000 €

- *Berechnung neuer Felder:* Eine Tabelle enthält Bruttowerte, und es wird ein neues Feld für Nettowerte hinzugefügt:

Produkt	Brutto	Steuer (19 %)
Produkt A	119 €	-
Produkt B	238 €	-

Nach der Berechnung:

Produkt	Brutto	Steuer (19 %)	Netto
Produkt A	119 €	19 €	100 €
Produkt B	238 €	38 €	200

Durch diese Schritte wird sichergestellt, dass die transformierten Daten in einer qualitativ hochwertigen und nutzbaren Form vorliegen.

Laden

In der letzten Phase des ETL-Prozesses werden die transformierten Daten in das Zielsystem übertragen, z. B. ein Data-Warehouse oder einen Data Lake. Abhängig von den Anforderungen kann dies als vollständiger Ladevorgang (Initial Load) oder als inkrementeller Ladevorgang (Incremental Load) erfolgen, bei dem nur neue oder aktualisierte Daten geladen werden. Zusammen bilden diese drei Phasen die Grundlage eines effizienten ETL-Prozesses, der eine hohe Datenqualität und Konsistenz sicherstellt [20].

2.1.3 Relevanz der ETL-Prozesse in dies Arbeit

ETL-Prozesse sind für diese Arbeit von besonderer Bedeutung, da sie nicht nur die Integration und Verarbeitung semistrukturierter Daten ermöglichen, sondern auch die Einbindung von Anonymisierungsverfahren in die Datenverarbeitung unterstützen. Insbesondere wird untersucht, wie Anonymisierungsverfahren gezielt in den ETL-Prozess integriert werden können, um die Anforderungen an Datenschutz und Datensicherheit zu erfüllen, ohne die Nutzbarkeit der Daten für Analysen einzuschränken. Ein zentraler Fokus liegt dabei auf der Transformation semistrukturierter XML-Daten in relationale Daten im CSV-Format.

2.2 Semistrukturierte Daten

In diesem Abschnitt werden die Grundlagen und Eigenschaften semistrukturierter Daten behandelt, die in dieser Arbeit eine zentrale Rolle spielen. Semistrukturierte Daten sind ein zunehmend wichtiger Bestandteil moderner Datenverarbeitung. Sie zeichnen

sich durch eine flexible und hierarchische Struktur aus, die es ermöglicht, Daten sowohl inhaltlich als auch strukturell darzustellen. Ein typisches Beispiel für semistrukturierte Daten ist XML, das durch seine breite Anwendung im Web und in wissenschaftlichen sowie geschäftlichen Datenbanken hervorsticht [5]. Die Verwaltung und Analyse semistrukturierter Daten stellen jedoch besondere Herausforderungen dar, da sowohl die Struktur als auch der Inhalt berücksichtigt werden müssen. Die Fähigkeit, Beziehungen zwischen Elementen hierarchisch abzubilden und diese für Abfragen und Analysen nutzbar zu machen, ist zentral. In diesem Abschnitt werden die Merkmale semistrukturierter Daten beschrieben sowie die Transformation von XML-Daten in relationale Formate untersucht, die eine Grundlage für die spätere Nutzung dieser Daten in analytischen Systemen bildet [5].

2.2.1 Merkmale semistrukturierter Daten

Semistrukturierte Daten stellen einen Mittelweg zwischen strukturierten und unstrukturierten Daten dar. Sie besitzen keine feste Schema-Definition wie relationale Datenbanken, weisen jedoch eine hierarchische oder baumartige Struktur auf, die die Organisation und Verarbeitung erleichtert. Ein typisches Beispiel für semistrukturierte Daten ist XML (eXtensible Markup Language), das Daten sowohl inhaltlich als auch strukturell beschreibt [5]. Die wesentlichen Merkmale semistrukturierter Daten sind:

- **Flexibilität der Struktur:** Die Daten können variierende Schemata enthalten, was sie anpassungsfähig für unterschiedliche Anwendungsfälle macht [5].
- **Hierarchische Organisation:** Die Daten werden oft in einer baumartigen Struktur dargestellt, bei der Elemente und Unterelemente (Child Nodes) miteinander verknüpft sind. Diese Organisation erleichtert die Repräsentation komplexer Beziehungen zwischen Daten [5].
- **Selbstbeschreibende Natur:** Die Strukturinformationen sind in den Daten eingebettet, typischerweise in Form von Tags oder Attributen, wie sie in XML verwendet werden [55].
- **Kombination von Struktur und Inhalt:** Semistrukturierte Daten enthalten sowohl Metadaten über die Struktur als auch die eigentlichen Dateninhalte. Dies macht sie vielseitig, bringt jedoch die Herausforderung mit sich, Struktur und Inhalt gleichzeitig verarbeiten zu müssen [5].

Semistrukturierte Daten sind besonders geeignet für Anwendungen, bei denen die Daten flexibel und oft heterogen sind, wie beispielsweise in webbasierten Anwendungen oder bei der Verarbeitung von XML-Dokumenten. Ihre Flexibilität und Selbstbeschreibungsfähigkeit machen sie zu einer wichtigen Datenkategorie in der modernen Datenverarbeitung.

Das Beispiel illustriert, wie Daten durch eine saubere und gut definierte XML-Struktur organisiert werden können, um sie leicht zu lesen und weiterzuverarbeiten.

```xml
<!-- Kundenliste mit Informationen -->
<customers>
  <customer>
    <id>1</id>
    <name>Max Mustermann</name>
    <email>max.mustermann@example.com</email>
    <phone>+49 123 456789</phone>
    <address>
      <street>Musterstraße 1</street>
      <city>Berlin</city>
      <postalCode>10115</postalCode>
      <country>Germany</country>
    </address>
  </customer>
  <customer>
    <id>2</id>
    <name>Erika Musterfrau</name>
    <email>erika.musterfrau@example.com</email>
    <phone>+49 987 654321</phone>
    <address>
      <street>Beispielweg 10</street>
      <city>Hamburg</city>
      <postalCode>20095</postalCode>
      <country>Germany</country>
    </address>
  </customer>
</customers>
```

Listing 2.1: Beispiel für eine XML-Struktur

2.2.2 Transformation von XML zu relationalen Daten

Die Transformation von XML-Daten in relationale Datenstrukturen ist ein zentraler Schritt, um semistrukturierte Daten in traditionellen Datenbanksystemen zu speichern und zu verarbeiten. XML-Daten zeichnen sich durch eine hierarchische und flexible Struktur aus, während relationale Datenbanken auf festen, tabellarischen Schemata basieren. Die Herausforderung besteht darin, die hierarchischen Beziehungen und die Flexibilität von XML in das relationale Modell zu überführen, ohne dabei wichtige Informationen zu verlieren [29]. Ein gängiger Ansatz zur Bewältigung dieser Herausforderung ist die Zerlegung oder das Shredding von XML-Dokumenten. Dabei werden die hierarchischen Strukturen des XML-Dokuments in mehrere relationale Tabellen aufgeteilt, wobei Elemente und Attribute des XML in entsprechende Tabellenzeilen und -spalten überführt werden. Dieser Prozess erfordert eine sorgfältige Planung, um sicherzustellen, dass die Beziehungen zwischen den Daten erhalten bleiben und die Integrität der Informationen gewahrt wird [18]. Ein weiterer Ansatz ist die Verwendung von hybriden Datenbanksystemen, die sowohl relationale als auch XML-Datenmodelle unterstützen.

Solche Systeme ermöglichen es, XML-Daten direkt zu speichern und zu verarbeiten, während gleichzeitig relationale Datenstrukturen genutzt werden können. Dies bietet Flexibilität bei der Datenmodellierung und erleichtert die Integration unterschiedlicher Datentypen [18]. Die Wahl des geeigneten Ansatzes hängt von verschiedenen Faktoren ab, darunter die Komplexität der XML-Daten, die Anforderungen an die Abfrageleistung und die Notwendigkeit, Datenstrukturen flexibel anpassen zu können. In jedem Fall ist es wichtig, die spezifischen Anforderungen der Anwendung zu berücksichtigen, um eine effiziente und effektive Datenintegration zu gewährleisten [13]. Das XML-Beispiel aus Listing 2.1, das Kundendaten und deren Bestellungen beschreibt, kann wie folgt in relationale Tabellen transformiert werden:

Tabelle 2.1: `Kunden`

id	**name**
1	Max Mustermann
2	Erika Musterfrau

Tabelle 2.2: `Bestellungen`

orderID	**customerID**	**date**	**amount**
1001	1	2025-01-01	250.00
1002	1	2025-01-05	150.00
1003	2	2025-01-10	300.00

2.2.3 Relevanz semistrukturierter Daten in der Arbeit

Semistrukturierte Daten sind in dieser Arbeit von zentraler Bedeutung, da sie die Grundlage für die Transformation und Integration von Daten im ETL-Prozess bilden. Besonders XML-Daten, die eine flexible und hierarchische Struktur besitzen, dienen als Ausgangspunkt für die Umwandlung in relationale Formate wie CSV. Diese Transformation ist notwendig, um die Daten für weitere Schritte im ETL-Prozess, insbesondere für Anonymisierungsverfahren, aufzubereiten. Die Arbeit zielt darauf ab, semistrukturierte Daten effizient in den ETL-Prozess einzubinden, um sowohl technische als auch datenschutzrechtliche Anforderungen zu erfüllen. Die Herausforderung besteht darin, die spezifischen Eigenschaften semistrukturierter Daten so zu nutzen, dass eine präzise Datenverarbeitung und eine nahtlose Integration von Anonymisierungsverfahren gewährleistet werden können. Durch die Transformation und Anpassung der Daten wird eine Grundlage geschaffen, die sowohl die Datenqualität als auch die Einhaltung rechtlicher Vorgaben sicherstellt.

2.3 Anonymisierungsverfahren

Die Anonymisierung personenbezogener Daten ist ein zentraler Bestandteil moderner Datenverarbeitung, insbesondere wenn sensible Informationen verarbeitet und analysiert werden sollen. Ziel der Anonymisierung ist es, personenbezogene Daten so zu verändern, dass sie nicht mehr einer spezifischen Person zugeordnet werden können, während die Daten für analytische oder wissenschaftliche Zwecke weiterhin nutzbar bleiben [45].

2.3.1 Personenbezogene Daten

Bei den in diesem Projekt verwendeten Daten handelt es sich um personenbezogene Daten. Personenbezogene Daten sind gemäß der DSGVO als Informationen definiert, die sich auf eine identifizierte oder identifizierbare natürliche Person beziehen [53]. Eine identifizierbare natürliche Person ist eine Person, die direkt oder indirekt anhand von Merkmalen wie Name, Kennnummer, Standortdaten oder besonderen physischen, physiologischen, genetischen, psychischen, wirtschaftlichen, kulturellen oder sozialen Identitäten bestimmt werden kann. Beispiele für personenbezogene Daten gemäß der DSGVO sind [54]:

- **Name:** Der vollständige Vor- und Nachname einer Person dient der direkten Identifikation und ist eines der häufigsten personenbezogenen Daten.
- **Adresse:** Wohnanschrift oder Postadresse ermöglichen die Lokalisierung einer Person und sind daher besonders schützenswert.
- **Telefonnummer:** Die private oder geschäftliche Telefonnummer erlaubt die direkte Kontaktaufnahme mit einer Person.
- **E-Mail-Adresse:** E-Mail-Adressen können eine direkte Identifizierung ermöglichen, insbesondere wenn sie den Namen enthalten.
- **Geburtsdatum:** Das Geburtsdatum kann zur Identifikation und zur Bestimmung des Alters einer Person verwendet werden.
- **Personalausweisnummer:** Eindeutige Kennnummern wie die Personalausweis- oder Reisepassnummer ermöglichen eine verlässliche Identifikation.
- **Kundennummern:** Kunden- oder Vertragsnummern können Personen in Unternehmen oder Organisationen eindeutig zugeordnet werden.
- **Finanzielle Informationen:** Bankverbindungen, Kreditkartennummern oder Steuerdaten sind sensible Daten, die einer Person zugeordnet werden können.
- **Gesundheitsdaten:** Medizinische Diagnosen, Krankheitsverläufe oder Rezepte enthalten sensible Informationen über den Gesundheitszustand einer Person.

- **Genetische Daten:** DNA-Sequenzen oder genetische Marker sind eindeutig einer Person zuordenbar und ermöglichen Rückschlüsse auf Erbkrankheiten oder Verwandtschaftsverhältnisse.
- **Biometrische Daten:** Fingerabdrücke, Gesichtserkennung oder Iris-Scans dienen zur eindeutigen Identifikation einer Person.
- **Standortdaten:** GPS-Daten, Bewegungsprofile oder Aufenthaltsorte können zur Nachverfolgung von Personen genutzt werden.
- **IP-Adresse:** Die IP-Adresse eines Geräts ermöglicht die Identifikation des Nutzers im Internet.

Diese Daten sollen bei der Verarbeitung der Eingabedaten anonymisiert werden, um den Schutz personenbezogener Informationen gemäß der DSGVO zu gewährleisten.

2.3.2 Definition und Ziel der Anonymisierung

Die Anonymisierung beschreibt den Prozess, personenbezogene Daten so zu verändern, dass eine Identifikation von Individuen weder direkt noch indirekt möglich ist. Dabei werden identifizierende Informationen entfernt, verschlüsselt oder durch generische Werte ersetzt. Das Ziel der Anonymisierung ist es, den Datenschutz zu gewährleisten, indem die Rückverfolgbarkeit auf einzelne Personen verhindert wird, während die Daten gleichzeitig für analytische, statistische oder wissenschaftliche Zwecke nutzbar bleiben [45, 24]. Ein zentraler Aspekt der Anonymisierung ist die Einhaltung rechtlicher Vorgaben, wie der DSGVO, die fordert, dass personenbezogene Daten nur in einer Form verarbeitet werden dürfen, die den Schutz der Privatsphäre sicherstellt. Neben dem Datenschutz liegt ein weiteres Ziel darin, die Balance zwischen Schutz der sensiblen Daten und deren Nützlichkeit zu finden. Insbesondere in datengetriebenen Anwendungen wie Business Intelligence oder Machine Learning ist es entscheidend, dass anonymisierte Daten weiterhin aussagekräftig bleiben, um verlässliche Ergebnisse zu erzielen [53, 45].

2.3.3 Mechanismen zur Anonymisierung

Die häufigsten Mechanismen zur Datenanonymisierung sind [22]:

- **Maskierung**:

  Sensible Informationen werden durch andere Werte ersetzt (z. B. Zufallsdaten), um die Originaldaten zu verbergen.

- **Pseudonymisierung**:

  Identifizierende Informationen werden durch Pseudonyme ersetzt, die bei Bedarf zurückverfolgt werden können.

- **Generalisierung**:

  Daten werden in weniger präzise Kategorien zusammengefasst (z. B. Altersgruppen statt exaktem Alter).

- **Rauschen hinzufügen**:

  Daten werden durch das Hinzufügen zufälliger Werte verändert, um die Identifizierung zu erschweren.

Maskierung

Die **Maskierung** ist eine Technik zur Datenanonymisierung, bei der sensible Informationen durch irrelevante oder zufällige Werte ersetzt werden. Ziel ist es, den ursprünglichen Informationsgehalt zu verbergen, während die Struktur und das Format der Daten erhalten bleiben. Maskierung wird häufig in Testumgebungen, Datenfreigabeprozessen und bei der Verarbeitung von Produktionsdaten angewendet [23]. Ein Beispiel für die Maskierung von Daten zeigt die folgende Darstellung:

```
Originaldaten:
Name: Max Mustermann
Kreditkartennummer: 1234-5678-9012-3456
Telefonnummer: +49 123 456789
E-Mail: max.mustermann@example.com

Maskierte Daten:
Name: Test User
Kreditkartennummer: XXXX-XXXX-XXXX-3456
Telefonnummer: +49 000 000000
E-Mail: user@example.com
```

Funktionsweise der Maskierung

Die Maskierung erfolgt durch verschiedene Techniken, die sicherstellen, dass sensible Daten geschützt werden, während das Format und die Struktur der Daten erhalten bleiben:

- **Ersetzung sensibler Werte**: Daten wie Namen, Adressen oder Kreditkartennummern werden durch fiktive oder synthetische Werte ersetzt. Dies verhindert den Zugriff auf echte Daten durch unautorisierte Personen [23].
- **Formatbewahrung**: Die Maskierung behält oft das ursprüngliche Format der Daten bei. Zum Beispiel könnte eine Kreditkartennummer `1234-5678-9101-1121` durch eine ähnliche, aber fiktive Nummer `9876-5432-1098-7654` ersetzt werden [23].
- **Flexibilität**: Die Maskierung kann je nach Sensibilität und Anwendungsfall dynamisch oder statisch erfolgen:
  - *Statische Maskierung*: Maskierung wird einmalig durchgeführt und in der gesamten Datenbank gespeichert [26].
  - *Dynamische Maskierung*: Maskierte Daten werden bei jeder Abfrage zur Laufzeit generiert [26].

Beispielanwendungen der Maskierung

Die Maskierung findet in verschiedenen Anwendungsbereichen Einsatz, um sensible Informationen zu schützen und gleichzeitig die Nutzbarkeit der Daten zu erhalten. Testdaten können beispielsweise für Entwickler und Tester bereitgestellt werden, sodass sie mit realistischen, aber anonymisierten Daten arbeiten können, ohne dabei das Risiko einer Datenschutzverletzung einzugehen [19]. Ein weiteres Anwendungsgebiet ist die Verarbeitung von Kundendaten in Berichten. Hierbei werden sensible Informationen wie Kundennamen maskiert, wenn Daten an externe Stakeholder weitergegeben werden. Dies gewährleistet den Schutz der Privatsphäre und verhindert unbefugten Zugriff auf personenbezogene Informationen [23]. Zudem wird die Maskierung häufig für Schulungen und Trainingsdaten genutzt, um reale Szenarien nachzubilden, ohne dabei sensible Informationen preiszugeben. Dies ist besonders in Bereichen wie der IT-Sicherheit, der Datenanalyse oder im medizinischen Umfeld relevant, wo Schulungsdaten benötigt werden, die gleichzeitig den Datenschutzanforderungen entsprechen [26].

Vorteile der Maskierung

Die Maskierung bietet mehrere Vorteile im Bereich der Datensicherheit und -verarbeitung. Durch die Anwendung dieser Methode werden sensible Informationen effektiv vor unbefugtem Zugriff geschützt, da personenbezogene Daten nicht mehr in ihrer ursprünglichen Form vorliegen und somit keine direkte Identifikation möglich ist [23]. Darüber hinaus trägt die Maskierung maßgeblich zur Einhaltung regulatorischer Anforderungen bei, insbesondere im Kontext von Datenschutzgesetzen wie DSGVO, indem sie sicherstellt, dass personenbezogene Daten in einer geschützten Form gespeichert und verarbeitet werden [19]. Ein weiterer Vorteil der Maskierung liegt in der Erhaltung der Datenstruktur, wodurch anonymisierte Datensätze weiterhin für Entwicklungszwecke, Analysen oder maschinelles Lernen genutzt werden können, ohne dass das Format der Daten verändert wird [26].

Herausforderungen der Maskierung

Trotz der Vorteile birgt die Maskierung auch einige Herausforderungen. Ein wesentliches Risiko besteht in der Möglichkeit der Reidentifikation insbesondere, wenn maskierte Daten mit externen Quellen kombiniert werden. Dadurch könnten anonymisierte Informationen wieder einer Person zugeordnet werden, was die Wirksamkeit der Maskierung gefährdet [23]. Zudem kann die Datenintegrität beeinträchtigt werden, da die Maskierung unter Umständen die Qualität und Konsistenz der Daten verändert, wodurch deren Nutzbarkeit für analytische oder geschäftliche Zwecke eingeschränkt wird [19]. Ein weiterer Nachteil sind die Kosten die mit der Implementierung effektiver Maskierungstechniken verbunden sind. Der Aufwand für die Entwicklung, Integration und Wartung solcher Lösungen kann erheblich sein, insbesondere in großen Datenumgebungen mit komplexen Anforderungen [34].

Pseudonymisierung

Die **Pseudonymisierung** ist ein Verfahren zur Datenanonymisierung, bei dem personenbezogene Informationen durch Pseudonyme ersetzt werden, die keine direkte Identifikation der betroffenen Person ermöglichen. Pseudonyme sind eindeutige Ersatzwerte, wie IDs oder Codes, die mit den Originaldaten verknüpft werden können, falls dies erforderlich ist [22]. Ein Beispiel für die Pseudonymisierung von Daten zeigt die folgende Darstellung: Pseudonymisierung:

```
Originaldaten:
Patienten-ID: 12345
Name: Erika Musterfrau
Geburtsdatum: 01.01.1985
Krankheit: Diabetes Typ 2

Pseudonymisierte Daten:
Patienten-ID: PZ001X2
Name: Pseudonymisiert
Geburtsdatum: 1985 (Jahr, ohne Tag und Monat)
Krankheit: Diabetes Typ 2
```

Funktionsweise der Pseudonymisierung

- **Ersetzung durch Pseudonyme**: Anstelle von Namen oder anderen identifizierenden Daten werden Pseudonyme wie zufällig generierte Nummern oder Tokens verwendet [19].

- **Trennung der Identifikatoren**: Die Originaldaten werden separat und sicher gespeichert, um eine Rückverfolgbarkeit zu gewährleisten [34].

- **Anwendung spezifischer Algorithmen**: Algorithmen zur Generierung von Pseudonymen sorgen für die Sicherheit und Einzigartigkeit der Ersatzwerte [26].

Beispielanwendungen der Pseudonymisierung

- **Medizinische Forschung**: Pseudonymisierte Patientendaten ermöglichen die Analyse von Gesundheitsdaten, ohne die Identität der Patienten offenzulegen [22].

- **Kundendatenverwaltung**: In Kundenmanagementsystemen können Pseudonyme verwendet werden, um die Privatsphäre der Kunden zu schützen [19].

- **Analyse von Nutzerdaten**: Pseudonymisierung wird eingesetzt, um Nutzerdaten in digitalen Plattformen zu analysieren, ohne die Identität der Nutzer preiszugeben [34].

Vorteile der Pseudonymisierung

- **Datenschutz**: Pseudonymisierung reduziert das Risiko von Datenschutzverletzungen [22].
- **Rückverfolgbarkeit**: Ermöglicht die Zuordnung zu Originaldaten, wenn dies gesetzlich oder operativ erforderlich ist [19].
- **Einhaltung gesetzlicher Anforderungen**: Hilft Organisationen, Vorschriften wie die DSGVO einzuhalten [34].

Herausforderungen der Pseudonymisierung

- **Reidentifizierungsrisiken**: Wenn Pseudonyme mit anderen Datensätzen kombiniert werden, besteht die Gefahr der Reidentifikation [22].
- **Aufwand der Implementierung**: Die sichere Verwaltung und Trennung der Daten können technisch anspruchsvoll und kostspielig sein [26].
- **Begrenzte Anonymität**: Im Gegensatz zur vollständigen Anonymisierung bietet die Pseudonymisierung nur einen eingeschränkten Schutz [19].

Generalisierung

Generalisierung reduziert die Präzision von Daten, um die Identifikation einzelner Personen zu erschweren. Beispielsweise könnte eine exakte Altersangabe durch eine Altersgruppe ("30–40 Jahre") ersetzt werden. Diese Technik wird oft in Kombination mit anderen Methoden eingesetzt, um ein hohes Maß an Anonymisierung zu erreichen [45]. Ein Beispiel für die Generalisierung von Daten zeigt die folgende Darstellung:

```
Originaldaten:
Fahrzeug-ID: 98765
Position: 48.137154, 11.576124 (München Zentrum)
Zeit: 14:03:25
Geschwindigkeit: 50 km/h

Generalisierte Daten:
Fahrzeug-ID: Anonymisiert
Position: München (ohne exakte Koordinaten)
Zeit: 14:00-15:00
Geschwindigkeit: 40-60 km/h
```

Funktionsweise der Generalisierung

- **Abstraktion spezifischer Werte**: Genau definierte Daten wie Geburtsdaten werden durch größere Kategorien ersetzt, z. B. durch Altersgruppen [3].

- **Hierarchische Gruppierung**: Werte werden in einer hierarchischen Struktur zusammengefasst, z. B. wird eine Stadt zu einer Region generalisiert [52].
- **Kontrollierte Informationsreduktion**: Die Generalisierung wird so angewendet, dass die Daten nützlich bleiben, aber keine Rückschlüsse auf Einzelpersonen möglich sind [3].

Beispielanwendungen der Generalisierung

Die Generalisierung wird in verschiedenen Bereichen angewendet, um die Privatsphäre zu schützen und dennoch aussagekräftige Analysen zu ermöglichen. Gesundheitsdaten werden häufig generalisiert, indem spezifische Diagnosen durch übergeordnete Krankheitskategorien ersetzt werden, sodass individuelle Patienten nicht identifiziert werden können [47]. In der demografischen Analyse erfolgt eine Generalisierung von Merkmalen wie Alter, Einkommen oder Wohnort. Anstatt exakter Werte werden beispielsweise Altersgruppen oder Einkommensklassen gebildet, um eine detaillierte Identifikation einzelner Personen zu verhindern [52]. Auch in statistischen Berichten spielt die Generalisierung eine wichtige Rolle. Bevölkerungsumfragen oder wirtschaftliche Analysen nutzen generalisierte Daten, um die Anonymität der Befragten zu wahren und dennoch aussagekräftige Ergebnisse zu liefern [3].

Vorteile der Generalisierung

Die Generalisierung bietet mehrere Vorteile im Bereich der Datenschutztechniken. Einer der Hauptvorteile ist der verbesserte Datenschutz, da durch die Verallgemeinerung von Datenpunkten das Risiko der Identifikation einzelner Personen erheblich reduziert wird [47]. Zudem bleibt die Datenqualität erhalten, da die generalisierten Informationen weiterhin für Analysen und statistische Auswertungen genutzt werden können, ohne dass individuelle Details offengelegt werden [52]. Ein weiterer Vorteil ist die einfache Implementierung, da viele bestehende Datenschutz-Tools bereits integrierte Funktionen für Generalisierung enthalten, die ohne großen Aufwand eingesetzt werden können [3].

Herausforderungen der Generalisierung

Die Generalisierung bringt neben ihren Vorteilen auch einige Herausforderungen mit sich. Ein wesentliches Problem ist der Verlust von Details, da durch die Verallgemeinerung spezifische Informationen reduziert werden, was die Präzision von Datenanalysen beeinträchtigen kann [47]. Zudem erfordert die Generalisierung oft die Erstellung komplexer Hierarchien, da für jedes Attribut geeignete Generalisierungsstufen definiert werden müssen. Dieser Prozess kann zeitaufwendig sein und erfordert fundierte Kenntnisse über die Datenstruktur sowie die Anwendungsfälle [52]. Ein weiteres zentrales Problem ist der Ausgleich zwischen Datenschutz und Nützlichkeit. Während eine stärkere Generalisierung den Schutz der Daten erhöht, kann sie gleichzeitig deren Aussagekraft und Verwendbarkeit für Analysen und Berichte einschränken. Daher muss ein angemessenes Gleichgewicht gefunden werden, um sowohl Datenschutzanforderungen als auch analytische Anforderungen zu erfüllen [3].

Rauschen hinzufügen

Das Hinzufügen von Rauschen ist eine Technik zur Datenanonymisierung, bei der zufällige Werte zu sensiblen Daten hinzugefügt werden, um die Identifizierbarkeit von Individuen zu reduzieren. Diese Methode wird häufig in statistischen Analysen und maschinellen Lernmodellen verwendet, um Datenschutz zu gewährleisten, während die Gesamtmuster der Daten erhalten bleiben [12].

Funktionsweise des Rauschens

- **Zufallswerte hinzufügen**: Sensiblen Werten, wie Einkommen oder Alter, werden kleine zufällige Variationen hinzugefügt, sodass genaue Rückschlüsse auf Einzelpersonen erschwert werden [15].
- **Verwendung von Verteilungsmodellen**: Das Rauschen wird auf der Grundlage einer bestimmten Verteilung (z. B. normalverteiltes Rauschen) angewendet, um statistische Eigenschaften der Daten beizubehalten [30].
- **Differential Privacy**: Eine spezielle Form des Rauschens, die mathematisch garantiert, dass das Risiko der Identifikation eines Individuums minimiert wird [12].

Beispielanwendungen des Rauschens

Die Technik des Hinzufügens von Rauschen wird in verschiedenen Anwendungsbereichen eingesetzt, um die Privatsphäre zu schützen und gleichzeitig eine sinnvolle Datenanalyse zu ermöglichen. In statistischen Berichten wird Rauschen hinzugefügt, um sensible Daten in veröffentlichten Statistiken, wie Bevölkerungsumfragen, zu schützen [30]. Im Bereich des maschinellen Lernens ermöglicht das gezielte Einfügen von Rauschen, dass Modelle auf verrauschten Daten trainiert werden, wodurch der Datenschutz gewahrt bleibt und sensible Informationen nicht rekonstruiert werden können [15]. Zudem wird Rauschen in der Datenaggregation verwendet, um sicherzustellen, dass einzelne Datenpunkte in aggregierten Datensätzen nicht identifizierbar sind, wodurch das Risiko einer Reidentifikationminimiert wird [12].

Vorteile des Rauschens

Die Technik des Hinzufügens von Rauschen bietet mehrere Vorteile im Bereich des Datenschutzes. Ein wesentlicher Vorteil ist der Erhalt der Datenstruktur, da die Gesamtstatistik der Daten weitgehend unverändert bleibt, sodass Analysen und Auswertungen weiterhin aussagekräftig sind [30]. Zudem gewährleistet diese Methode einen hohen Datenschutz, da individuelle Datenpunkte schwerer zu identifizieren sind und somit das Risiko einer Reidentifikationminimiert wird [15].

Herausforderungen des Rauschens

Obwohl das Hinzufügen von Rauschen eine effektive Methode zum Schutz sensibler Daten darstellt, gibt es auch Herausforderungen. Eine der größten Herausforderungen ist

die Datenverzerrung da das eingefügte Rauschen die Genauigkeit von Analysen und statistischen Auswertungen beeinträchtigen kann [12]. Zudem erfordert diese Methode eine sorgfältige Parameterauswahl, insbesondere die Festlegung der Rauschintensität (z. B. Standardabweichung). Eine falsche Wahl kann entweder zu unzureichendem Datenschutz oder zu einem erheblichen Verlust an Datenqualität führen [30].

2.3.4 Relevanz der Anonymisierung in der Arbeit

Die Anonymisierung spielt in dieser Arbeit eine zentrale Rolle, da personenbezogene Daten verarbeitet werden, die den Anforderungen des Datenschutzes entsprechen müssen. Insbesondere im Kontext eines ETL-Prozesses, bei dem Daten aus verschiedenen Quellen extrahiert, transformiert und in ein Zielsystem geladen werden, stellt die Anonymisierung eine essenzielle Aufgabe dar. Eine frühzeitige Anonymisierung während des Transformationsschritts ist entscheidend, da eine pauschale Anonymisierung im Zielsystem die Nutzbarkeit der Daten für analytische Zwecke erheblich einschränken könnte. In dieser Arbeit wird untersucht, wie Anonymisierungsverfahren in den ETL-Prozess integriert werden können, um sowohl technische als auch datenschutzrechtliche Anforderungen zu erfüllen. Dabei liegt ein besonderer Fokus auf der Transformation semistrukturierter XML-Daten in relationale Formate wie CSV. Ziel ist es, die Anonymisierung so zu gestalten, dass sensible Informationen geschützt werden, während die Daten weiterhin für Analysen und Anwendungen nutzbar bleiben. Die Anonymisierung muss somit effizient, flexibel und an die spezifischen Anforderungen der Daten angepasst erfolgen, um die Balance zwischen Datenschutz und Datenverwendbarkeit zu gewährleisten.

2.4 Mechanismen zur Dateiextraktion

Die Extraktion von Dateien stellt einen zentralen Schritt im ETL-Prozess dar, da sie die Grundlage für die weitere Verarbeitung und Analyse der Daten bildet. Besonders bei semistrukturierten Formaten wie XML oderJavaScript Object Notation (JSON) ist der Einsatz geeigneter Mechanismen entscheidend, um relevante Informationen effizient und zielgerichtet zu extrahieren. In der Fachliteratur wird hervorgehoben, dass Mechanismen wie XPath, XQuery und reguläre Ausdrücke essenzielle Werkzeuge sind, um Daten aus diesen Formaten präzise und strukturiert zu extrahieren.

2.4.1 XPath

XPath ist eine Abfragesprache, die von der W3C entwickelt wurde, um Knoten in XML-Dokumenten präzise zu adressieren und zu extrahieren. Sie dient als Kerntechnologie für viele XML-bezogene Standards, darunter XSLT und XQuery, und ermöglicht die Navigation durch die hierarchische Struktur von XML-Dokumenten [56].

Grundlagen von XPath:

XPath betrachtet ein XML-Dokument als Baumstruktur, bei der jedes Element, Attribut, Textfragment, Kommentar und jeder Verarbeitungshinweis ein Knoten ist. Mithilfe von Pfadausdrücken können Nutzer durch den Baum navigieren, um spezifische Knoten zu identifizieren [56].

```xml
<catalog>
  <book id="1">
    <title>XML Handbook</title>
    <author>John Doe</author>
    <price>45</price>
  </book>
  <book id="2">
    <title>Learning XML</title>
    <author>Jane Smith</author>
    <price>29</price>
  </book>
</catalog>
```

Listing 2.2: Beispiel für XML-Dokument

Die folgenden XPath-Abfragen können auf dieses Dokument angewendet werden:

- **//book** – Wählt alle `book`-Elemente im gesamten Dokument.
- **/catalog/book/title** – Gibt alle Titel der Bücher im `catalog` zurück.
- **//book[@id='1']/title** – Wählt den Titel des Buches mit der ID 1.
- **/catalog/book[price > 30]** – Wählt Bücher, deren Preis über 30 liegt.

Wichtige Konzepte und Funktionen:

- **Pfadausdrücke:** Pfadausdrücke sind die grundlegende Methode, um Knoten im XML-Baum zu adressieren. Ein einfacher Ausdruck wie '//book/title' selektiert alle 'title'-Elemente, die unter einem 'book'-Element stehen, unabhängig von ihrer Position im Dokument.
- **Achsen:** XPath bietet verschiedene Navigationsmöglichkeiten durch sogenannte Achsen, zum Beispiel:
  - `child::` Wählt direkte Kinder eines Knotens aus (Standardachse).
  - `descendant::` Wählt alle Nachfahren eines Knotens aus.
  - `parent::` Wählt den Elternknoten aus.
  - `attribute::` Wählt Attribute eines Knotens.
- **Knotentests:** Mit Knotentests können spezifische Knotentypen selektiert werden, zum Beispiel `text()` für Textknoten oder `comment()` für Kommentare.

- **Prädikate:** Prädikate erlauben die Filterung von Knoten basierend auf Bedingungen, zum Beispiel:
  - `[position()=1]`: selektiert das erste Element in einer Liste von Knoten.
  - `[@lang='en']`: Wählt Knoten, die das Attribut `lang` mit dem Wert `'en'` haben, also findet alle Elemente, die auf Englisch gesetzt sind."
- **Funktionen:** XPath bietet eine Vielzahl eingehender Funktionen, darunter:
  - `count()`: Zählt die Anzahl der ausgewählten Knoten.
  - `contains()`: Überprüft, ob ein String eine bestimmte Sequenz enthält.
  - `string-length()`: Gibt die Länge eines Strings zurück.

    Zum Beispiel:
    - **count(//book)** – Gibt die Anzahl der Bücher zurück.
    - **contains(title, 'XML')** – Überprüft, ob der Titel „XML“ enthält.
    - **string-length(title)** – Gibt die Länge eines Titels zurück.

XPath-Versionen:

XPath hat sich seit seiner Einführung erheblich weiterentwickelt:

- **XPath 1.0:** Die erste Version, veröffentlicht im Jahr 1999, führte grundlegende Navigations- und Filterfunktionen ein, zum Beispiel:

  ```
  /catalog/book/title
  Selektiert alle Buchtitel aus dem Katalog.
  ```

- **XPath 2.0:** Erweiterte die Sprachmöglichkeiten durch Unterstützung für Datentypen und eine stärkere Integration in XQuery und XSLT, zum Bespiel:

  ```
  for $x in /catalog/book
  return <discount>{$x/price * 0.9}</discount>
  Berechnet einen Rabatt von 10 % auf die Buchpreise.
  ```

- **XPath 3.1:** Die aktuelle Version, veröffentlicht im Jahr 2017, fügte Unterstützung für Maps und Arrays hinzu, was die Verarbeitung komplexerer Datenstrukturen erleichtert [56], zum Bespiel:

  ```
  let $book := /catalog/book
  return array { $book/title }
  Erstellt ein Array mit allen Buchtiteln.
  ```

Anwendungsbeispiele:

XPath wird in vielen Kontexten verwendet, darunter:

- XML-Datenabfragen: Extraktion spezifischer Knoten aus großen XML-Dokumenten, zum Bespiel:

```
//project[Name="XML-Anon"]/Student
Wählt den Student des Projekts "XML-Anon"
```

- XML-Transformationen In Verbindung mit XSLT, um XML-Dokumente in andere Formate wie HTML oder JSON umzuwandeln, zum Bespiel:

```
<xsl:for-each select="//book">
   <xsl:value-of select="title"/>
</xsl:for-each>
Transformiert die XML-Daten in eine HTML-Darstellung.
```

- Validierung: Überprüfung von XML-Daten gegen bestimmte Kriterien, z. B. Struktur oder Wertebereiche.

XPath ist eine essenzielle Technologie in der XML-Datenverarbeitung und bildet die Grundlage für zahlreiche Anwendungen in der modernen Datenverarbeitung.

2.4.2 XSLT

XSLT ist eine deklarative Sprache, die speziell entwickelt wurde, um XML-Daten zu transformieren und in andere Formate wie HTML, Text, CSV oder andere XML-Formate umzuwandeln. Es arbeitet eng mit XPath zusammen, um präzise auf Teile eines XML-Dokuments zuzugreifen und diese zu verarbeiten. Die Transformation erfolgt durch die Anwendung von Vorlagen (Templates), die durch XPath-Ausdrücke gesteuert werden [59].

Funktionsweise von XSLT

Die Funktionsweise von XSLT kann in drei Hauptschritten zusammengefasst werden [16]:

- Eingabe: Ein XML-Dokument dient als Eingabedatei.
- Transformation: XSLT-Regeln und Templates, die in einer XSL-Datei definiert sind, werden auf das XML-Dokument angewendet.
- Ausgabe: Die Ausgabe kann in einem gewünschten Format wie HTML, Text oder einem anderen XML-Dokument erfolgen.

Grundlegende Bestandteile von XSLT

- XSLT-Stylesheet Ein XSLT-Stylesheet ist ein XML-Dokument, das die Transformationsregeln enthält. Es definiert, wie spezifische Teile des XML-Dokuments verarbeitet und ausgegeben werden. Beispiel für die Grundstruktur eines XSLT-Stylesheets:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/
    Transform" version="1.0">
    <xsl:template match="/">
        <!-- Transformationen hier -->
    </xsl:template>
</xsl:stylesheet>
```

Listing 2.3: XSL-Stylesheet Beispiel [59]

- **Templates**

  Templates definieren, wie Teile des XML-Dokuments transformiert werden sollen. Diese Templates werden durch XPath-Ausdrücke auf spezifische Knoten angewendet. Beispiel:

```
<xsl:template match="/root/element">
    <output>
        <xsl:value-of select="child"/>
    </output>
</xsl:template>
```

Listing 2.4: XSL-Template für spezifisches Element [51]

- **XPath-Unterstützung**

  XSLT nutzt XPath, um Knoten im XML-Dokument zu adressieren und zu selektieren. XPath ermöglicht das Navigieren durch die XML-Hierarchie und das Filtern von Daten basierend auf spezifischen Kriterien.

  Beispiel: Der Ausdruck **//element** wählt alle Knoten mit dem Namen element [56].

Anwendungsbeispiele

- **Transformation von XML in HTML**

  XSLT wird häufig verwendet, um XML-Daten in HTML umzuwandeln, beispielsweise zur Darstellung von Daten in einem Webbrowser. Zum Beispiel:

```
<xsl:template match="/">
    <html>
        <body>
            <h1>XML-Daten</h1>
            <ul>
```

```
6                   <xsl:for-each select="/root/element">
7                   <li><xsl:value-of select="name"/></li>
8                   </xsl:for-each>
9               </ul>
10          </body>
11      </html>
12  </xsl:template>
```

Listing 2.5: XSL-Template für XML-Datenanzeige [21]

- **Filterung von XML-Daten**

  Mithilfe von XSLT können nur spezifische Daten aus einem XML-Dokument extrahiert werden. Zum Beispiel kann der folgende Code alle Artikel mit einem Preis über 20 herausfiltern:

```
1  <xsl:template match="//item[price␣>␣20]">
2      <expensive-item>
3          <xsl:value-of select="name"/>
4      </expensive-item>
5  </xsl:template>
```

Listing 2.6: XSL-Template für teure Artikel [51]

2.4.3 XQuery

XQuery ist eine W3C-empfohlene Abfragesprache, die speziell für die Abfrage und Transformation von XML-Daten entwickelt wurde. Sie ermöglicht das Extrahieren, Filtern und Transformieren von Daten aus XML-Dokumenten, XML-Datenbanken sowie anderen hierarchischen Datenquellen wie JSON. XQuery basiert auf XPath und erweitert dessen Funktionen um leistungsstarke Abfrage- und Verarbeitungsmöglichkeiten [60].

Grundlagen von XQuery

- **Datenmodell**:

  XQuery verwendet ein hierarchisches Datenmodell, das XML-Daten als geordnete Baumstruktur darstellt. Jede Abfrage navigiert durch diese Baumstruktur, selektiert relevante Knoten und verarbeitet sie gemäß den angegebenen Kriterien [57].

- **Funktionale Abfragesprache**:

  XQuery ist eine funktionale Sprache, d. h., jede Abfrage besteht aus Funktionen, die Eingaben verarbeiten und Ausgaben erzeugen. Dadurch ist XQuery besonders flexibel und mächtig bei der Verarbeitung von komplexen Daten [44].

- **Integration mit XPath**:

  XQuery verwendet XPath-Ausdrücke, um spezifische Knoten innerhalb eines XML-Dokuments zu adressieren. Zum Beispiel [58]:

```xml
<catalog>
  <book>
    <title>XML Handbook</title>
    <price>45</price>
  </book>
  <book>
    <title>Learning XML</title>
    <price>29</price>
  </book>
</catalog>
```

Listing 2.7: Beispiel für eine XML-Datei

  Das obige XML-Dokument stellt eine einfache Katalogstruktur dar, die eine Liste von Büchern enthält. Jedes `book`-Element besitzt ein `title`- und ein `price`-Element, die den Titel des Buches und dessen Preis in einer strukturierten Form darstellen.

  Mit XPath können gezielt Daten aus diesem XML-Dokument extrahiert werden. Die folgenden XPath-Abfragen demonstrieren gängige Anwendungsfälle:

  – **/catalog/book** wählt alle `book`-Elemente innerhalb des übergeordneten `catalog`-Elements aus. Das Ergebnis dieser Abfrage sind alle in der XML-Datei enthaltenen Bücher.

  – **/catalog/book[price > 30]** filtert alle `book`-Elemente, bei denen der Wert des `price`-Elements größer als 30 ist. In diesem Fall wird nur das Buch mit dem Titel XML Handbook zurückgegeben.

  Diese XPath-Ausdrücke sind besonders nützlich für die Extraktion bestimmter Informationen aus XML-Dokumenten in verschiedenen Anwendungsbereichen wie Datenanalyse, Konfigurationsmanagement und Web-Services.

Funktionen und Merkmale von XQuery

- **FLWOR-Ausdrücke**:

  FLWOR steht für `"For, Let, Where, Order by, Return"` und bildet das Kernkonstrukt von XQuery-Abfragen:

```
for $book in /catalog/book
    where $book/price > 30
    order by $book/title
    return $book/title
```

Listing 2.8: XQuery-Beispiel

Dieser Ausdruck filtert Bücher mit einem Preis über 30, sortiert sie nach Titel und gibt die Titel zurück [60].

- **Datentypunterstützung**:

  XQuery unterstützt umfangreiche Datentypen, einschließlich Zahlen, Zeichenketten, Datums- und Zeitwerte, und ermöglicht komplexe Berechnungen [44].

- **XPath-Erweiterungen**:

  Neben der Unterstützung von XPath bietet XQuery zusätzliche Funktionen wie:

  – Aggregatfunktionen (sum(), avg()).
  – Zeichenkettenfunktionen (substring(), concat()).
  – Mathematische Funktionen (round(), abs()) [58].

Anwendungsbeispiele

- **Datenextraktion**:

  XQuery wird häufig verwendet, um spezifische Daten aus großen XML-Datenbanken oder Dokumenten zu extrahieren. Beispiel:

```
for $customer in /customers/customer
    where $customer/country = "Germany"
    return $customer/name
```

Listing 2.9: XQuery-Beispiel: Kunden in Deutschland [57]

- **Transformation von XML-Daten**:

  Mit XQuery können XML-Daten in neue XML-Strukturen transformiert werden:

```
for $book in /catalog/book
    return <item>
              <title>{$book/title}</title>
              <price>{$book/price}</price>
           </item>
```

Listing 2.10: XQuery-Beispiel: Buchinformationen extrahieren [60]

- **JSON-Verarbeitung**:

  XQuery 3.1 erweitert die Funktionalität um JSON-Unterstützung. Damit können JSON-Daten mit XPath-ähnlichen Ausdrücken abgefragt werden:

```
let $data := json-doc("data.json")
return $data("users")[1]("name")
```

Listing 2.11: XQuery-Beispiel: Zugriff auf JSON-Daten [61]

Vorteile von XQuery

- **Flexibilität**:

  XQuery kann sowohl XML- als auch JSON-Daten verarbeiten, wodurch es sich für viele moderne Anwendungsfälle eignet [44].

- **Leistungsstark bei Datenbanken**:

  Viele XML-Datenbanken wie BaseX, eXist-db oder MarkLogic nutzen XQuery als Hauptabfragesprache [57].

- **Standardisierung**:

  XQuery ist ein W3C-Standard, der breite Unterstützung in Tools und Datenbanken genießt [60].

2.5 Mechanismen zur Datenspeicherung

Die Speicherung anonymisierter Daten im Rahmen von ETL-Prozessen stellt eine besondere Herausforderung dar, da neben der Gewährleistung der Datenstruktur und Verfügbarkeit auch Sicherheits- und Datenschutzanforderungen erfüllt werden müssen. Nach der Anonymisierung ist es essenziell, geeignete Speicherformate und -technologien zu wählen, um die Nutzbarkeit der Daten für analytische und geschäftliche Zwecke sicherzustellen. Eine häufig genutzte Methode ist die Speicherung in relationalen Datenbanken, die insbesondere für strukturierte Daten geeignet sind. Relationale Datenbanken ermöglichen effiziente Abfragen und Analysen, indem sie die Daten in Tabellenform organisieren [49]. Eine weitere Option ist die Speicherung in textbasierten Formaten wie CSV oder JSON. Diese Formate sind besonders für kleinere Datensätze oder für die Weitergabe an externe Analysewerkzeuge nützlich. Während CSV-Dateien eine einfache tabellarische Struktur bieten, ermöglicht JSON die Speicherung hierarchischer Strukturen, die für semistrukturierte Daten wie XML von Vorteil sind [5].

2.5.1 Relationale Datenbanken

Relationale Datenbanken sind ein bewährter Ansatz zur Speicherung strukturierter Daten. Sie organisieren Daten in Tabellenform, wobei Zeilen Datensätze und Spalten Attribute darstellen. Diese Struktur ermöglicht die effiziente Abfrage und Manipulation von Daten mittels Structured Query Language (SQL). Für anonymisierte Daten eignen sich relationale Datenbanken besonders, da sie durch ihre konsistente Struktur und eingebaute Sicherheitsmechanismen die Einhaltung von Datenschutzanforderungen erleichtern [49].

Vorteile:

- **Effiziente Datenverwaltung**: Relationale Datenbanken bieten optimierte Algorithmen für das Einfügen, Aktualisieren und Abfragen großer Datenmengen [10].

- **Datenkonsistenz**: Die Verwendung von Schlüsseln und Constraints stellt sicher, dass die Datenintegrität gewahrt bleibt [49].

- **Sicherheit**: Funktionen wie Rollen- und Benutzerrechte sowie Datenbankverschlüsselung schützen anonymisierte Daten vor unberechtigtem Zugriff [49].

- **Weit verbreitete Standards**: Relationale Datenbanken wie MySQL, PostgreSQL oder Microsoft SQL Server sind gut dokumentiert und weit verbreitet [49].

Herausforderungen:

- **Skalierbarkeit**: Traditionelle relationale Datenbanken stoßen bei extrem großen Datensätzen oder verteilten Systemen an ihre Grenzen [2].

- **Anpassung an semistrukturierte Daten**: Für Daten mit unregelmäßiger Struktur (z. B. JSON oder XML) sind zusätzliche Erweiterungen oder Tools erforderlich [49].

Relationale Datenbanken sind ideal für Anwendungsfälle, bei denen strukturierte, konsistente Daten erforderlich sind, wie z. B. in Finanz- und Gesundheitsanwendungen oder bei der Verarbeitung von Berichten aus anonymisierten Daten [10].

2.5.2 Dateibasierte Speicherung in Formaten wie CSV oder JSON

Die dateibasierte Speicherung ist eine weit verbreitete Methode zur Ablage von Daten in leicht zugänglichen Formaten wie CSV oder JSON[5]. Diese Formate sind besonders geeignet für die Speicherung von anonymisierten Daten, da sie einfach zu erstellen, zu lesen und zwischen verschiedenen Systemen auszutauschen sind [17].

Vorteile:

- **Einfache Handhabung**: CSV- und JSON-Dateien sind textbasiert, was sie für eine Vielzahl von Anwendungen und Plattformen kompatibel macht [5].

- **Breite Unterstützung**: Viele Datenverarbeitungs- und Analysewerkzeuge (z. B. Python, R, Excel) bieten native Unterstützung für CSV- und JSON-Formate [17].

- **Flexibilität**: JSON ermöglicht die Speicherung von hierarchischen und komplexen Datenstrukturen, was es ideal für semistrukturierte Daten macht [5].

- **Portabilität**: Die Dateien sind leicht zwischen Systemen und Anwendungen übertragbar, ohne dass eine spezielle Software erforderlich ist [17].

Herausforderungen:

- **Fehlende Optimierung**: Im Vergleich zu Datenbanken fehlt dateibasierten Speichermethoden die Optimierung für Abfragen und Transaktionen, was die Verarbeitung großer Datenmengen verlangsamen kann [49].
- **Begrenzte Sicherheitsfunktionen**: Textbasierte Dateien bieten von Natur aus keine integrierte Verschlüsselung oder Zugriffskontrolle, wodurch zusätzliche Sicherheitsmaßnahmen erforderlich werden [5].
- **Speichereffizienz**: Bei sehr großen Datenmengen kann die Speicherung in CSV- oder JSON-Formaten zu einem höheren Speicherbedarf führen als spezialisierte Speicherlösungen [17].

Die Speicherung in CSV- oder JSON-Formaten bietet sich in mehreren Szenarien an. Zum einen eignet sie sich hervorragend für den einfachen Austausch anonymisierter Daten zwischen verschiedenen Systemen und Organisationen, da diese Formate weit verbreitet und plattformunabhängig sind [5]. Darüber hinaus ist JSON besonders vorteilhaft für die Speicherung von semistrukturierten Daten, da es hierarchische und flexible Datenstrukturen unterstützt [17]. Ein weiteres wichtiges Anwendungsfeld ist das Prototyping, bei dem schnelle Entwicklungs- und Testprozesse im Vordergrund stehen und keine komplexe Datenbankinfrastruktur erforderlich ist [49]. Diese Eigenschaften machen CSV und JSON zu vielseitigen und nützlichen Formaten für viele Anwendungen.

3 Stand der Technik

Dieses Kapitel gibt einen Überblick über den aktuellen Stand der Technik im Bereich der Datenverarbeitung und Anonymisierung im Kontext von ETL-Prozessen. Dabei werden Werkzeuge und Methoden betrachtet, die in den verschiedenen Phasen eines ETL-Prozesses – von der Dateiextraktion über die Anonymisierung bis hin zur Datenspeicherung – eingesetzt werden können. Zunächst wird das methodische Vorgehen bei der Suche nach relevanten Technologien und Werkzeugen beschrieben. Anschließend werden die erzielten Ergebnisse vorgestellt, wobei der Fokus auf spezifischen Werkzeugen zur Dateiextraktion, Datenanonymisierung und Datenspeicherung liegt. Diese Betrachtung dient als Grundlage für die spätere Konzeption und Implementierung eines angepassten ETL-Prozesses im Rahmen dieser Arbeit.

3.1 Vorgehen bei der Suche

Um den aktuellen Stand der Technik im Bereich der Dateiextraktion, Datenanonymisierung und Datenspeicherung zu ermitteln, wurde ein systematisches Vorgehen angewendet. Dabei lag der Fokus darauf, relevante Werkzeuge, Technologien und wissenschaftliche Publikationen zu identifizieren, die im Kontext von ETL-Prozessen eingesetzt werden können. Die Recherche erfolgte unter Verwendung wissenschaftlicher Datenbanken wie Google Scholar, IEEE Xplore und SpringerLink sowie durch die Analyse aktueller technischer Berichte und Open-Source-Projekte. Dabei wurden gezielte Suchbegriffe wie ETL Tools, Data Anonymization Techniques und XML to CSV Transformation verwendet. Parallel dazu wurden offizielle Dokumentationen und Anwendungsberichte von etablierten Softwarelösungen untersucht, um deren Funktionsumfang und Einsatzmöglichkeiten zu bewerten. Das strukturierte Vorgehen diente dazu, ein umfassendes Bild der verfügbaren Technologien zu erhalten und deren Eignung für die Anforderungen dieser Arbeit zu bewerten.

3.2 Erzielte Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Recherche und Analyse zum Stand der Technik vorgestellt. Der Fokus liegt auf der Identifikation und Bewertung von Werkzeugen und Technologien, die für die einzelnen Phasen eines ETL-Prozesses – von der Dateiextraktion über die Datenanonymisierung bis hin zur Datenspeicherung – relevant sind. Die Ergebnisse sind in drei Kategorien gegliedert: Werkzeuge zur Dateiextraktion, Werkzeuge zur Datenanonymisierung und Werkzeuge zur Datenspeicherung. Jede Kategorie wird detailliert betrachtet, wobei die Funktionalitäten und Einsatzmöglichkeiten der jeweiligen Werkzeuge erläutert werden. Diese Analyse bildet die Grundlage

für die spätere Konzeption und Implementierung eines ETL-Prozesses im Rahmen dieser Arbeit.

3.3 Werkzeuge zur Dateiextraktion

In diesem Abschnitt werden verschiedene Werkzeuge und Techniken vorgestellt, die zur Extraktion von Daten aus XML-Dateien eingesetzt werden können.

3.3.1 DOM

Das DOM ist eine plattform- und sprachunabhängige Programmierschnittstelle, die die Struktur und den Inhalt von XML- und HTML-Dokumenten in einer Baumstruktur im Speicher darstellt. Jeder Knoten in diesem Baum repräsentiert einen Teil des Dokuments, beispielsweise Elemente, Attribute oder Text [31].

Hauptmerkmale von DOM

- **Strukturierte Darstellung:** DOM stellt Dokumente als hierarchische Baumstrukturen dar, wobei jeder Knoten ein Objekt ist, das einen Teil des Dokuments repräsentiert.
- **Programmierbare Schnittstelle:** Es ermöglicht Programmen und Skripten, die Struktur, den Stil und den Inhalt von Dokumenten dynamisch zu ändern.
- **Sprach- und plattformunabhängig:** DOM ist nicht an eine spezifische Programmiersprache gebunden und kann in verschiedenen Umgebungen eingesetzt werden [31].

DOM in Java

In Java wird DOM über die Java API for XML Processing (JAXP) implementiert. JAXP bietet standardisierte Methoden zum Parsen und Verarbeiten von XML-Dokumenten unter Verwendung des DOM-Ansatzes [38].

Vorteile der Verwendung von DOM

- Beliebige Navigation durch die Baumstruktur, da das gesamte Dokument im Speicher geladen wird [38].
- Einfaches Hinzufügen oder Entfernen von Knoten, was die Manipulation der Dokumentstruktur erleichtert [31].

Nachteile der Verwendung von DOM

- **Hoher Speicherverbrauch:** Da das gesamte Dokument in den Speicher geladen wird, kann der Speicherbedarf bei großen XML-Dateien erheblich sein [38].

3.3.2 SAX

Die SAX ist eine ereignisbasierte Programmierschnittstelle zur Verarbeitung von XML-Dokumenten. Im Gegensatz zum DOM lädt SAX nicht das gesamte Dokument in den Speicher, sondern verarbeitet es sequenziell. Dadurch ist SAX besonders speichereffizient und für große XML-Dateien geeignet [11].

Hauptmerkmale von SAX

- **Ereignisgesteuertes Parsing:** SAX arbeitet mit Ereignissen wie dem Start und Ende von Elementen oder dem Auffinden von Text, die durch spezialisierte Methoden verarbeitet werden [28].
- **Sequentielle Verarbeitung:** Das Dokument wird zeilenweise gelesen, ohne die gesamte Baumstruktur im Speicher zu halten. Dies ermöglicht eine schnelle Verarbeitung mit minimalem Speicherverbrauch [11].

Vorteile der Verwendung von SAX

- **Geringer Speicherverbrauch:** Da SAX keine Baumstruktur im Speicher speichert, ist es ideal für große XML-Dateien [11].
- **Schnelle Verarbeitung:** Die sequentielle Abarbeitung ermöglicht eine hohe Geschwindigkeit bei der Datenverarbeitung [28].

Nachteile der Verwendung von SAX

- **Keine direkte Modifikation:** Da keine Baumstruktur existiert, ist die direkte Manipulation des Dokuments nicht möglich [39].
- **Komplexere Implementierung:** Die ereignisbasierte Natur erfordert eine sorgfältige Handhabung des Dokumentstatus, was den Code komplexer macht [39].

3.3.3 StAX

Die StAX ist eine ereignisbasierte Programmierschnittstelle zum Lesen und Schreiben von XML-Dokumenten in Java. Im Gegensatz zu SAX, das ein Push-basiertes Modell verwendet, arbeitet StAX mit einem Pull-basierten Ansatz, bei dem die Anwendung die Kontrolle über das Parsing behält und Ereignisse nach Bedarf abruft [40].

Hauptmerkmale von StAX

- **Pull-basiertes Parsing:** Die Anwendung steuert den Lesevorgang und zieht die benötigten Ereignisse aus dem Parser, was eine bessere Kontrolle über den Parsing-Prozess ermöglicht [40].
- **Bidirektionale Verarbeitung:** StAX unterstützt sowohl das Lesen als auch das Schreiben von XML-Dokumenten, was es vielseitig einsetzbar macht [40].

- **Geringer Speicherverbrauch:** Durch die sequentielle Verarbeitung wird nur ein Teil des Dokuments im Speicher gehalten, was die Effizienz bei großen Dateien erhöht [40].

StAX-APIs

StAX bietet zwei Haupt-APIs für die Verarbeitung von XML-Daten:

- **Cursor-API:** Ermöglicht den Zugriff auf XML-Daten über einen zeigerähnlichen Mechanismus, bei dem der Parser auf das nächste Ereignis im Dokument zeigt [35].

- **Iterator-API:** Behandelt XML-Daten als eine Reihe von Ereignisobjekten, die von der Anwendung durchlaufen werden können [35].

Vorteile der Verwendung von StAX

- **Hohe Kontrolle:** Entwickler können gezielt bestimmen, wann und welche Teile des Dokuments gelesen oder geschrieben werden [40].

- **Effiziente Speicherverwaltung:** Nur relevante Teile des Dokuments werden im Speicher gehalten, was besonders bei großen XML-Dateien von Vorteil ist [40].

Nachteile der Verwendung von StAX

- **Komplexität:** Die Notwendigkeit, den Parsing-Prozess manuell zu steuern, kann zu komplexerer und fehleranfälligerer Implementierung führen [40].

- **Keine automatische Validierung:** Im Gegensatz zu DOM bietet StAX keine integrierte Validierung der XML-Struktur [40].

3.3.4 JAXB

Die JAXB ist ein Framework, das es ermöglicht, Java-Klassen mit XML-Daten zu verknüpfen. Es bietet eine effiziente Möglichkeit, XML-Dokumente in Java-Objekte zu konvertieren (Unmarshalling) und umgekehrt (Marshalling) [36].

Hauptmerkmale von JAXB

- **Automatisches Mapping:** JAXB ermöglicht die automatische Zuordnung zwischen XML-Elementen und Java-Klassen, wodurch der manuelle Aufwand reduziert wird [36].

- **Anpassbare Bindings:** Durch Annotationen können Entwickler steuern, wie XML-Elemente auf Java-Felder abgebildet werden [37].

- **Schema-Unterstützung:** JAXB kann XML-Schemata XML Schema Definition (XSD)[1]) verwenden, um Java-Klassen zu generieren, die den XML-Strukturen entsprechen [36].

Vorteile der Verwendung von JAXB

- **Produktivitätssteigerung:** Automatisiertes Mapping reduziert den Entwicklungsaufwand und minimiert Fehler [36].
- **Wartungsfreundlichkeit:** Änderungen im XML-Schema können leicht in den Java-Klassen reflektiert werden [36].
- **Standardisierung:** JAXB ist ein standardisiertes API und wird von vielen Java-Entwicklungsumgebungen unterstützt [36].

Nachteile der Verwendung von JAXB

- **Komplexität bei komplexen Schemata:** Bei sehr komplexen oder dynamischen XML-Schemata kann die Generierung passender Java-Klassen herausfordernd sein [32].
- **Leistungsüberhead:** Das Marshalling und Unmarshalling kann bei großen Datenmengen zu Performance-Einbußen führen [32].

Die folgende Tabelle 3.1 vergleicht die vorgestellten Methoden zur Verarbeitung von XML-Dokumenten hinsichtlich ihrer Leistungsfähigkeit, Speicheranforderungen und Anwendungsbereiche.

[1](XSD) ist eine Sprache zur Definition der Struktur von XML-Dokumenten. `https://www.w3.org/XML/Schema`, Abgerufen am 19/01/2025

Merkmal	**DOM**	**SAX**	**StAX**	**JAXB**
Parsing-Ansatz	Baumstruktur Speicherung im Speicher	Push-basiert	Pull-basiert	Objektorientiert Binding-Mechanismus
Speicherverbrauch	Hoch Ressourcenintensiv	Niedrig	Mittel	Mittel
Geschwindigkeit	Langsam Komplettes Dokument laden	Schnell	Schnell	Mittel
Modifikationen	Einfach	Nicht möglich	Möglich	Einfach
Eignung für große Daten	Eingeschränkt	Sehr gut	Gut	Mittel
Komplexität	Hoch	Mittel	Mittel	Niedrig
API-Typ	Lese- und Schreibzugriff	Nur Lesezugriff	Lese- und Schreibzugriff	Lese- und Schreibzugriff

Tabelle 3.1: Vergleich von XML-Verarbeitungsmethoden

3.4 Werkzeuge zur Datenanonymisierung

In diesem Kapitel werden verschiedene Werkzeuge zur Datenanonymisierung vorgestellt, die eine datenschutzkonforme Bereitstellung sensibler Daten für Analysezwecke und andere Anwendungen ermöglichen. Der Schwerpunkt liegt auf der Untersuchung und dem Vergleich bestehender Tools hinsichtlich ihrer Funktionalität, Einsatzfähigkeit, Unterstützung verschiedener Anonymisierungsmethoden sowie der Einhaltung gesetzlicher Datenschutzvorgaben. Ziel dieses Kapitels ist es, einen umfassenden Überblick über die verfügbaren Lösungen zu geben und deren Eignung für unterschiedliche Anwendungsfälle zu bewerten. Dazu werden einige grundlegende Datenschutzmodelle erläutert, bevor konkrete Werkzeuge analysiert werden.

3.4.1 Syntaktische Datenschutzmodelle

Die folgenden Werkzeuge implementieren mehrere syntaktische Datenschutzmodelle wie k-Anonymität, ℓ-Diversität, t-Closeness und δ-Präsenz. Diese Modelle helfen dabei, das Risiko einer Reidentifizierung zu quantifizieren und zu minimieren [42]. Im Folgenden werden diese Modelle detailliert beschrieben.

k-Anonymität

Das Konzept der k-Anonymität stellt sicher, dass jede Kombination von Quasiidentifikatoren innerhalb eines Datensatzes mit mindestens $k-1$ weiteren Einträgen identisch ist. Dadurch kann eine Einzelperson nicht eindeutig identifiziert werden, da sie sich in einer Gruppe von mindestens k anderen Individuen befindet. Dies wird durch die Generalisierung oder Unterdrückung von Attributen erreicht, sodass eine direkte Identifikation verhindert wird. Wenn zum Beipiel $k = 3$, müssen mindestens drei Datensätze dieselben Werte in den QI-Attributen aufweisen [46]. Die folgende Tabelle 3.2 zeigt ein Beispiel für die Anonymisierung von Patientendaten mit den Quasi-Identifikatoren `Alter` und `Postleitzahl`:

Alter	**Postleitzahl**	**Diagnose**
30-35	123**	Grippe
30-35	123**	Grippe
30-35	123**	Lungenentzündung

Tabelle 3.2: Beispiel für $k = 3$-Anonymität

Da mindestens drei Personen dieselben QI-Werte besitzen, wird die Identifikation einer einzelnen Person erheblich erschwert.

ℓ-Diversität

Das Datenschutzmodell der ℓ-Diversität erweitert die k-Anonymität, indem es sicherstellt, dass innerhalb jeder Gruppe von k anonymisierten Datensätzen mindestens ℓ unterschiedliche Werte für sensible Attribute vorhanden sind. Dadurch wird das Risiko von Homogenitäts- und Hintergrundwissen-Angriffen reduziert, da es für einen Angreifer schwieriger wird, bestimmte Informationen zu extrahieren. Wenn beispielsweise ℓ=2, müssen in jeder anonymisierten Gruppe mindestens zwei verschiedene Werte für ein sensibles Attribut existieren [33]. Dies wird in der Tabelle 3.3 als Beispiel veranschaulicht:

Alter	**Postleitzahl**	**Diagnose**
30-35	123**	Grippe
30-35	123**	Lungenentzündung
30-35	123**	Lungenentzündung

Tabelle 3.3: Beispiel für ℓ-Diversität mit $\ell = 2$

Da in jeder Gruppe mindestens zwei verschiedene Diagnosen vorhanden sind, wird es für schwieriger, die Diagnose einer bestimmten Person zu erraten.

t-Closeness

Das Modell der t-Closeness erweitert die ℓ-Diversität, indem es sicherstellt, dass die Verteilung eines sensitiven Attributs innerhalb einer anonymisierten Gruppe nicht mehr als

eine festgelegte Schwelle t von der Gesamtverteilung des Attributs im gesamten Datensatz abweicht. Dadurch wird ein höheres Maß an Datenschutz erreicht, da sichergestellt wird, dass die Verteilung sensibler Attribute innerhalb jeder Gruppe der Gesamtverteilung möglichst ähnlich bleibt [1]. Ein Beispiel für die Verteilung der Diagnosen im gesamten Datensatz ist: 50% Grippe, 30% Lungenentzündung und 20% COVID-19. Eine Gruppe gilt als t-close, wenn die Verteilung der sensiblen Attribute innerhalb der Gruppe der Gesamtverteilung entspricht oder nur geringfügig davon abweicht. Eine Gruppe, die diese Bedingung nicht erfüllt, könnte beispielsweise wie folgt aussehen:

Alter	**Postleitzahl**	**Diagnose**
30-35	123**	COVID-19
30-35	123**	COVID-19
30-35	123**	COVID-19

Tabelle 3.4: Beispiel für eine nicht t-close Gruppe

Hier wäre das Risiko hoch, da 100% der Personen in dieser Gruppe COVID-19 als Diagnose haben, was erheblich von der Gesamtverteilung abweicht und eine potenzielle Rückführung auf einzelne Personen erleichtert.

δ-Präsenz

Das Modell der δ-Präsenz definiert eine untere und obere Schranke für die Wahrscheinlichkeit, mit der ein bestimmtes Individuum in einem veröffentlichten Datensatz erscheint. Dadurch wird insbesondere der Schutz vor Hintergrundwissen-Angriffen verbessert, da verhindert wird, dass eine Person mit hoher Wahrscheinlichkeit in einem veröffentlichten Datensatz identifiziert werden kann [25]. Ein anschauliches Beispiel verdeutlicht das Konzept der δ-Präsenz. Angenommen, eine Person stammt aus einer bestimmten Region und könnte in einem medizinischen Datensatz enthalten sein. Ohne δ-Präsenz könnte die Wahrscheinlichkeit, dass diese Person tatsächlich in der Datenbank existiert, bei 90% liegen. Durch die Einführung einer δ-Präsenz mit einer Schranke von 10%-50% wird diese Wahrscheinlichkeit auf einen kontrollierten Bereich begrenzt. Dies trägt dazu bei, dass die Zugehörigkeit einer Person zu einem veröffentlichten Datensatz nicht mit absoluter Sicherheit bestimmt werden kann.

Postleitzahl	**Präsenz in Datensatz**
123**	10%-50%
124**	10%-50%
125**	10%-50%

Tabelle 3.5: Beispiel für δ-Präsenz mit einer Schranke von 10%-50%

Durch diese Maßnahme wird sichergestellt, dass die Wahrscheinlichkeit des Auftretens einzelner Personen im veröffentlichten Datensatz in einem bestimmten Bereich bleibt und somit nicht exakt bestimmt werden kann.

3.4.2 ARX Data Anonymization Tool

Das ARX-Tool ist eine umfassende Open-Source-Software zur Anonymisierung sensibler personenbezogener Daten. Es unterstützt eine Vielzahl von Datenschutz- und Risikomodellen, Methoden zur Datenumwandlung sowie Verfahren zur Analyse der Nützlichkeit der Ausgabe-Daten [6]. Das ARX-Tool unterstützt mehrere Anonymisierungsmethoden, um den Schutz personenbezogener Daten zu gewährleisten und gleichzeitig die Nutzbarkeit der Daten zu erhalten. Zu den wichtigsten Verfahren gehören die **Generalisierung**, **Microaggregation** und **Unterdrückung**: [42] [6].

- **Generalisierung** ist eine Technik, bei der spezifische Werte durch allgemeinere ersetzt werden, um die Identifizierbarkeit von Personen zu verringern. Beispielsweise kann eine präzise Altersangabe wie "32 Jahre" durch eine Altersgruppe wie "30-40 Jahre" ersetzt werden. ARX bietet flexible Generalisierungsoptionen, darunter hierarchische Generalisierungsstrategien und benutzerdefinierte Hierarchien, die den Anforderungen verschiedener Anwendungsfälle gerecht werden [6].

- **Microaggregation** fasst ähnliche Datensätze in Gruppen zusammen und ersetzt sie durch aggregierte Werte, beispielsweise durch Berechnung des Durchschnitts innerhalb jeder Gruppe. Dies trägt dazu bei, Muster innerhalb der Daten zu erhalten, während die individuellen Werte vor Re-Identifikation geschützt bleiben. ARX implementiert verschiedene Algorithmen zur Microaggregation, die eine anpassbare Balance zwischen Datenschutz und Datenpräzision ermöglichen [6].

- **Unterdrückung** ist eine Technik, bei der bestimmte Datenwerte oder ganze Datensätze entfernt werden, um die Wahrscheinlichkeit einer Re-Identifikation zu reduzieren. In ARX kann die Unterdrückung selektiv angewendet werden, indem hochriskante Werte oder eindeutige Merkmale maskiert oder vollständig eliminiert werden [6].

Das Tool implementiert mehrere syntaktische Datenschutzmodelle wie k-Anonymität, ℓ-Diversität, t-Closeness und δ-Präsenz. Diese Modelle helfen dabei, das Risiko einer Re-Identifizierung zu quantifizieren und zu minimieren [42]. ARX verfügt über integrierte Funktionen zur Analyse des Reidentifikationsrisikos, die es ermöglichen, potenzielle Datenschutzverletzungen zu identifizieren und zu bewerten [43]. Das Tool bietet Mechanismen zur Bewertung der Nützlichkeit anonymisierter Daten, um sicherzustellen, dass sie für analytische Zwecke weiterhin geeignet sind [42]. ARX verfügt über eine intuitive, plattformübergreifende grafische Benutzeroberfläche, die den Import, die Konfiguration und die Anonymisierung von Daten erleichtert. Es unterstützt verschiedene Datenquellen wie CSV-Dateien, MS Excel-Tabellen und relationale Datenbanksysteme [43]. Das ARX-Tool ist als Java-Bibliothek verfügbar und kann direkt in Java-Anwendungen integriert werden. Es bietet eine umfassende Programmierschnittstelle, die eine Vielzahl von Datenanonymisierungsfunktionen bereitstellt. ARX wird als eine umfassende Softwarebibliothek mit einer klaren API beschrieben, die eine einfache Integration in bestehende Java-Programme ermöglicht [6]. Zusammenfassend ist ARX ein leistungsfähiges und flexibles Werkzeug zur datenschutzkonformen Anonymisierung sensibler Daten, das vollständig in Java implementiert ist und nahtlos in Java-basierte Systeme

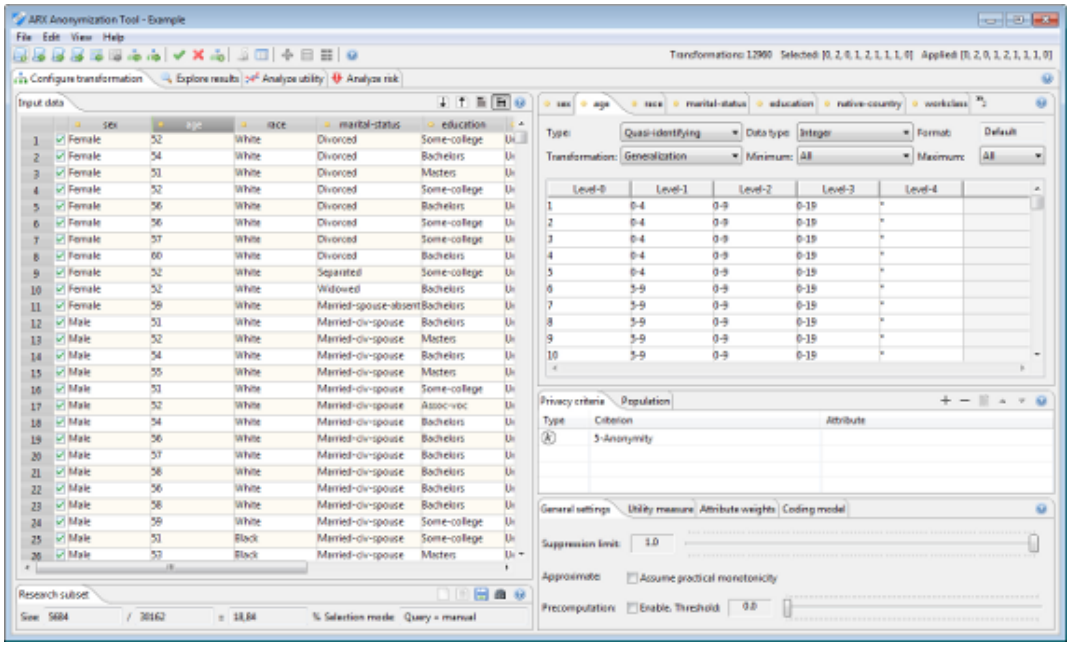

Abbildung 3.1: ARX-Grafische Nutzeroberfläche [6]

integriert werden kann. Es unterstützt verschiedene Datenschutzmodelle und Transformationstechniken und wird sowohl in der Forschung als auch in der Praxis eingesetzt [6]. Obwohl ARX keine direkte Unterstützung für die Anonymisierung von XML-Daten bietet, können diese durch vorherige Konvertierung in unterstützte Formate wie CSV oder relationale Datenbanken verarbeitet werden [48].

3.4.3 Cornell Anonymization Toolkit

Das *Cornell Anonymization Toolkit* (CAT) ist ein interaktives Open-Source-Werkzeug zur Anonymisierung sensibler Daten, entwickelt an der Cornell University. Es wurde konzipiert, um die Offenlegung von Identitäten in veröffentlichten Datensätzen zu minimieren, indem es verschiedene Angreifermodelle berücksichtigt [62]. Das Tool zeichnet sich durch die folgenden spezifischen Merkmale aus.

- **Interaktive Anonymisierung:** Cornell Anonymization Toolkit (CAT) bietet eine benutzerfreundliche Oberfläche, die es Anwendern ermöglicht, den Anonymisierungsprozess interaktiv zu steuern. Nutzer können Parameter wie den Grad der Anonymität anpassen und die Auswirkungen dieser Einstellungen auf die Datenqualität in Echtzeit evaluieren [62].
- **Unterstützte Datenschutzmodelle:** Das Toolkit implementiert mehrere Datenschutzmodelle, darunter k-Anonymität und ℓ-Diversität, um sowohl die Identitäts als auch die Attributsverknüpfung zu verhindern [42].
- **Risikobewertung:** CAT verfügt über integrierte Mechanismen zur Bewertung des Re-Identifikationsrisikos, indem es verschiedene Angreifermodelle simuliert

und die potenzielle Gefährdung der Privatsphäre der betroffenen Individuen analysiert [62].

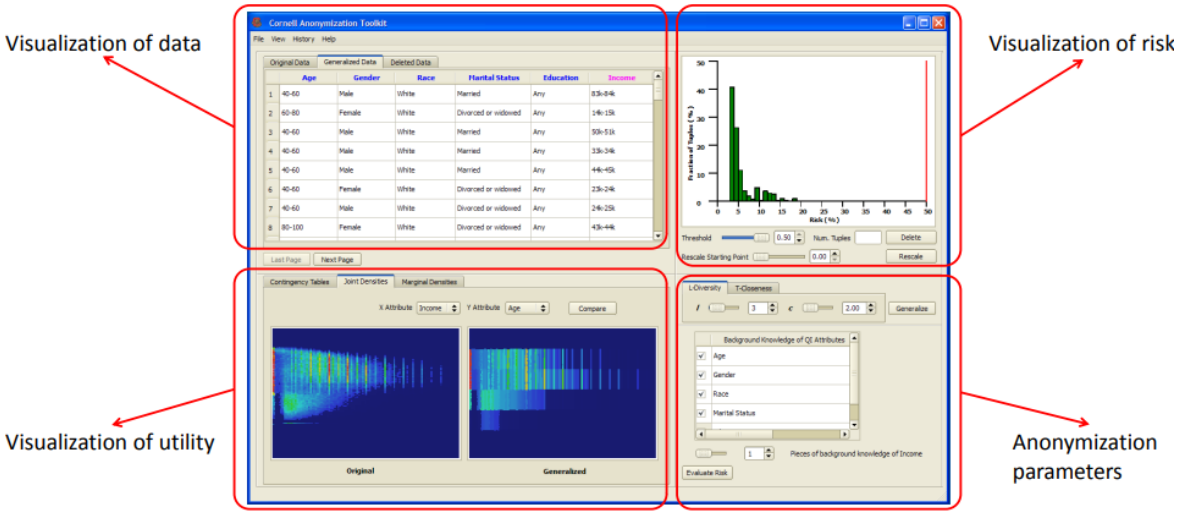

Abbildung 3.2: CAT-Grafische Nutzeroberfläche [9]

Obwohl CAT eine interaktive Anonymisierung ermöglicht, ist es als Forschungsprototyp konzipiert und wurde seit seiner Veröffentlichung im Jahr 2009 nicht mehr aktualisiert. Dies könnte zu Kompatibilitätsproblemen mit aktuellen Systemen führen und die Integration in moderne Datenverarbeitungsumgebungen erschweren [62].

3.4.4 AnonTool

Im Rahmen eines Projekts der Technologie- und Methodenplattform für die vernetzte medizinische Forschung e.V. (TMF) wurde an der Alpen-Adria-Universität Klagenfurt die Java-Anwendung *AnonTool* entwickelt. Das Tool bietet eine datenschutzkonforme Lösung zur Anonymisierung und Pseudonymisierung sensibler Daten in der medizinischen Forschung [4]. Das AnonTool ermöglicht die Verarbeitung von Daten im *XML*- und CSV-Format sowie über eine *JDBC*-Verbindung aus relationalen Datenbanken. Die anonymisierten Daten können ebenfalls in diesen Formaten gespeichert oder direkt in eine Datenbank exportiert werden. Die notwendigen Parameter und Generalisierungshierarchien werden in einer XML-Datei spezifiziert [8]. AnonTool kann entweder als eigenständige Java-Anwendung über eine *JAR*-Datei mit einer grafischen Benutzeroberfläche genutzt oder als Applikation in einen *GlassFish*-Anwendungsserver integriert werden. Dadurch wird eine flexible Nutzung in unterschiedlichen IT-Infrastrukturen ermöglicht [8]. Das Tool unterstützt bekannte Datenschutzmodelle wie k-Anonymität und ℓ-Diversität, die den Schutz personenbezogener Daten gewährleisten. Allerdings sind die verwendeten Anonymisierungsalgorithmen in der Dokumentation nicht näher beschrieben [4]. [2]

[2]Laut einer persönlichen Mitteilung der Entwickler Drepper, Johannes vom 27.01.2025 wurde die Weiterentwicklung des AnonTools bereits vor mehreren Jahren eingestellt.

3.4.5 UTD Anonymization Toolbox

An der University of Texas in Dallas wurde die *UTD Anonymization Toolbox* entwickelt, eine Sammlung von Java-Implementierungen bekannter Anonymisierungsalgorithmen. Die Software sowie der Quellcode sind unter der Adresse [50] frei verfügbar. Die Toolbox bietet verschiedene Anonymisierungsalgorithmen, darunter *Datafly*, *Mondrian*, *Incognito* (mit Unterstützung für ℓ-Diversity und t-Closeness) sowie *Anatomy* [8]. Die Toolbox erfordert, dass die zur Anonymisierung benötigten Parameter in einer XML-Konfigurationsdatei spezifiziert werden. Diese Datei enthält unter anderem die Generalisierungshierarchien der quasi-identifizierenden Attribute. Eine Ausnahme bildet der *Anatomy*-Algorithmus, der keine Generalisierungshierarchien benötigt. Als Eingangsformat werden ausschließlich *CSV*-Dateien unterstützt, und die anonymisierten Daten werden im selben Format ausgegeben.

Bei der Verwendung von Anatomy werden zwei separate Tabellen generiert [50]. Intern lädt die Toolbox die Daten in eine *SQLite*-Datenbank, die jedoch nicht plattformunabhängig ist. Abhängig von der verwendeten Plattform müssen daher die Toolbox und ein passender Datenbanktreiber möglicherweise neu kompiliert werden. Dies kann die Nutzung der Software erschweren. Darüber hinaus beschränkt die sequentielle Verarbeitung der Daten sowie das Laden aller Eingangsdaten in die eingebettete Datenbank die Fähigkeit der Toolbox, große Datenmengen effizient zu verarbeiten [8]. Ein wesentlicher Vorteil der Toolbox besteht darin, dass mehrere Algorithmen über eine einheitliche Schnittstelle auf die Daten angewendet werden können, was eine einfache Vergleichbarkeit verschiedener Anonymisierungstechniken ermöglicht [8]. Die UTD Anonymization Toolbox kann kostenlos von der offiziellen Webseite der University of Texas heruntergeladen werden [50].

3.4.6 Dynamic Data Masking

Dynamic Data Masking (DDM) ist eine Sicherheitsfunktion in Microsoft SQL Server, die den Zugriff auf sensible Daten einschränkt, indem sie bestimmte Informationen für nicht autorisierte Benutzer maskiert. Diese Maskierung erfolgt dynamisch bei der Abfrage der Datenbank, ohne dass die zugrunde liegenden Daten geändert werden [27]. DDM ermöglicht verschiedene Maskierungstypen, darunter vollständige, partielle und zufällige Maskierung. Administratoren können benutzerdefinierte Maskierungsregeln definieren, um die Anzeige vertraulicher Informationen zu steuern [27]. DDM eignet sich für Anwendungen, in denen sensible Daten dynamisch maskiert werden müssen, ohne die Struktur oder Integrität der Datenbank zu verändern, insbesondere in produktiven Echtzeitsystemen [27]. DDM ist in verschiedenen Editionen von SQL Server verfügbar, einschließlich Azure SQL-Datenbanken, und kann über SQL-Befehle oder die SQL Server Management Studio (SSMS)-Benutzeroberfläche konfiguriert werden [27].

Das folgende Beispiel zeigt, wie dynamische Datenmaskierung in SQL Server implementiert wird. Es wird eine Tabelle mit verschiedenen Maskierungsfunktionen erstellt, mit Beispieldaten befüllt und anschließend abgerufen [27].

```
1 -- Schema zur Aufnahme der Benutzertabellen
2 CREATE SCHEMA Data;
```

```
3
4
5  -- Tabelle mit maskierten Spalten
6  CREATE TABLE Data.Membership (
7      MemberID INT IDENTITY(1, 1) NOT NULL PRIMARY KEY
           CLUSTERED,
8      FirstName VARCHAR(100) MASKED WITH (FUNCTION = '
           partial(1,␣"xxxxx",␣1)') NULL,
9      LastName VARCHAR(100) NOT NULL,
10     Phone VARCHAR(12) MASKED WITH (FUNCTION = 'default()')
           NULL,
11     Email VARCHAR(100) MASKED WITH (FUNCTION = 'email()')
           NOT NULL,
12     DiscountCode SMALLINT MASKED WITH (FUNCTION = 'random
           (1,␣100)') NULL
13 );
14
15 -- Einfügen von Beispieldaten
16 INSERT INTO Data.Membership (FirstName, LastName, Phone,
       Email, DiscountCode)
17 VALUES
18 ('Roberto', 'Tamburello', '555.123.4567', '
       RTamburello@contoso.com', 10),
19 ('Janice', 'Galvin', '555.123.4568', 'JGalvin@contoso.com.
       co', 5),
20 ('Shakti', 'Menon', '555.123.4570', 'SMenon@contoso.net',
       50),
21 ('Zheng', 'Mu', '555.123.4569', 'ZMu@contoso.net', 40);
22
23 -- Benutzer erstellen und Berechtigungen erteilen
24 CREATE USER MaskingTestUser WITHOUT LOGIN;
25 GRANT SELECT ON SCHEMA::Data TO MaskingTestUser;
26
27 -- Benutzer simulieren für Testzwecke
28 EXECUTE AS USER = 'MaskingTestUser';
29 SELECT * FROM Data.Membership;
30 REVERT;
```

Listing 3.1: Erstellung einer dynamischen Datenmaskierung [27]

Das Ergebnis:

```
Originaldaten:
1 Roberto Tamburello 555.123.4567 RTamburello@contoso.com 10

Maskierte Daten:
1 Rxxxxxo Tamburello xxxx RXXX@XXXX.com 91
```

In diesem Beispiel wird die Telefonnummer vollständig maskiert, der Vorname teilweise

anonymisiert und die E-Mail-Adresse nach einem Muster verändert. Der Rabattcode (*DiscountCode*) wird mit einem zufälligen Wert ersetzt, der sich bei jeder Abfrage ändert [27].

3.4.7 Oracle Data Masking and Subsetting

Oracle Data Masking and Subsetting ist eine Funktion der Oracle-Datenbank, die zur Maskierung und Reduzierung sensibler Daten verwendet wird. Sie hilft Unternehmen, Datenschutzrichtlinien wie DSGVO einzuhalten, indem sie Mechanismen bereitstellt, um sensible Daten zu schützen und gleichzeitig die Effizienz der Datennutzung zu maximieren [41]. Die Lösung bietet verschiedene Maskierungstechniken, darunter deterministische und zufällige Maskierung, Substitution sowie Format-Preserving Encryption (FPE). Darüber hinaus ermöglicht sie das Subsetting, wodurch weniger relevante Datenmengen extrahiert werden, um Speicherplatz zu sparen und die Analysegeschwindigkeit zu erhöhen [41]. Oracle Data Masking and Subsetting eignet sich für die Nutzung in Entwicklungs- und Testumgebungen, in denen vertrauliche Informationen geschützt werden müssen, ohne die Datenintegrität und -nutzbarkeit zu beeinträchtigen [41]. Die Funktion ist als Teil der Oracle Enterprise Edition verfügbar und kann über das Oracle Enterprise Manager Tool konfiguriert werden [41].

Merkmal	**ARX**	**Cornell AnonTool**	**AnonTool**	**UTD Anonymization Toolbox**	**Oracle DMS**
Methoden	Generalisierung, Microaggregation, Unterdrückung	k-Anonymität, ℓ-Diversität	k-Anonymität, ℓ-Diversität	Datafly, Mondrian, Incognito, Anatomy	Substitution, Maskierung, FPE
Datenschutzmodelle	k-Anonymität, ℓ-Diversität, t-Closeness, δ-Präsenz	k-Anonymität, ℓ-Diversität	k-Anonymität, ℓ-Diversität	k-Anonymität, ℓ-Diversität, t-Closeness	Nicht erwähnt
Eingabeformate	CSV, Excel, DB-Integration	CSV, Datenbanken	XML, CSV, Datenbanken	CSV	Datenbanken
Plattform	Java, plattformübergreifend	Nicht erwähnt	Java-Anwendung, GlassFish	Java	Oracle Enterprise Edition
Besonderheiten	Risikobewertung, Datenqualität	Interaktive Anonymisierung	Vielfältige Ein- und Ausgabeformate	Erfordert SQLite-Datenbank	Subsetting-Funktionen
Lizenz	Open-Source	Open-Source	Unbekannt	Open-Source	Kommerziell

Tabelle 3.6: Vergleich von Datenanonymisierungstools

Die Tabelle 3.6 bietet einen Überblick über verschiedene Tools zur Datenanonymisierung.

3.5 Auswahl der einzusetzenden Tools

Zur Umsetzung der Anonymisierung in den ETL-Prozessschritten wurden die Tools DOM und ARX ausgewählt. Diese Werkzeuge wurden aufgrund ihrer spezifischen Funktionalitäten und ihrer Eignung für die jeweilige Aufgabe innerhalb des Prozesses ausgewählt.

3.5.1 Document Object Model (DOM)

Das DOM stellt eine leistungsfähige Methode zur Verarbeitung von XML-Daten bereit, da es das gesamte Dokument als Baumstruktur in den Speicher lädt. Diese Baumdarstellung ermöglicht eine beliebige Navigation durch das Dokument, wodurch gezielte Zugriffe auf einzelne Elemente und Attribute erleichtert werden [38]. Ein weiterer wesentlicher Vorteil von DOM ist die Möglichkeit, Knoten einfach hinzuzufügen, zu entfernen oder zu verändern, was die Manipulation der Dokumentstruktur besonders effizient macht [31]. Im Gegensatz dazu folgt SAX einem ereignisbasierten Ansatz und liest die XML-Daten sequentiell. Obwohl dies speicherschonender ist, erlaubt SAX daher keine direkte Modifikation der Daten [39]. Jede Manipulation oder komplexe Verarbeitung erfordert eine manuelle Implementierung, die den Zustand des Dokuments während des Durchlaufs sorgfältig verfolgt. Dies führt zu deutlich komplexerem Code und erschwert insbesondere Transformationen, bei denen auf unterschiedliche Teile des Dokuments mehrfach zugegriffen werden muss [39]. Die Entscheidung für DOM ergibt sich somit aus den Anforderungen des Anwendungsfalls: Für ETL-Prozesse, bei denen XML-Daten extrahiert, transformiert und anonymisiert werden müssen, ist eine detaillierte und flexible Kontrolle über die Datenstruktur entscheidend. Diese Flexibilität bietet SAX nicht, weshalb DOM für diese Art von Anwendung die bevorzugte Wahl ist.

3.5.2 ARX – Anonymization Tool

Das ARX-Framework ist ein Open-Source-Tool, das auf die datenschutzkonforme Anonymisierung sensibler Daten spezialisiert ist [6]. Es unterstützt eine Vielzahl von Anonymisierungstechniken, darunter Generalisierung, Pseudonymisierung und Maskierung, sowie die Implementierung syntaktischer Datenschutzmodelle wie k-Anonymität, ℓ-Diversität und t-Closeness. Diese Modelle sind entscheidend, um sensible Informationen zu schützen und sicherzustellen, dass die Anonymisierung den geltenden Datenschutzanforderungen entspricht [42] [6]. Das Framework erlaubt die Definition von Quasi-Identifikatoren, anhand derer sensible Daten anonymisiert werden und ist als Java-Bibliothek verfügbar und kann direkt in Java-Anwendungen integriert werden [6]. Zudem ermöglicht ARX eine Analyse der Risiken, die mit einer unzureichenden Anonymisierung verbunden sind, und bietet Werkzeuge, um diese Risiken zu minimieren

[42]. In diesem Projekt wird ARX verwendet, um die sensiblen Daten nach der Transformation durch DOM zu anonymisieren. Dies geschieht, indem zunächst die Quasi-Identifikatoren in den Daten identifiziert und anschließend mit den oben genannten Anonymisierungstechniken geschützt werden. Dabei sorgt ARX nicht nur für die Einhaltung datenschutzrechtlicher Vorgaben, sondern gewährleistet auch, dass die anonymisierten Daten weiterhin für analytische Zwecke nutzbar sind. ARX wurde aufgrund seiner umfangreichen Funktionalität, der Unterstützung verschiedener Datenschutzmodelle und seiner Open-Source-Natur ausgewählt. Es stellt sicher, dass die Anonymisierung effizient, nachvollziehbar und flexibel durchgeführt werden kann, was es zu einem unverzichtbaren Bestandteil der ETL-Pipeline macht.

4 Konzeption des Verfahrens

In diesem Kapitel wird die Konzeption des entwickelten Verfahrens zur Verarbeitung und Anonymisierung von Daten im Rahmen eines ETL-Prozesses vorgestellt. Ziel ist es, eine flexible und effiziente Pipeline zu entwerfen, die es ermöglicht, Daten aus einer XML-Quelle zu extrahieren, in einer Transformationsstufe sowohl strukturell als auch inhaltlich zu bearbeiten und anschließend durch verschiedene Anonymisierungstechniken zu schützen, bevor sie in ein CSV-Format exportiert werden. Die folgende Abbildung 4.1 zeigt die grundlegende Struktur des Prozesses:

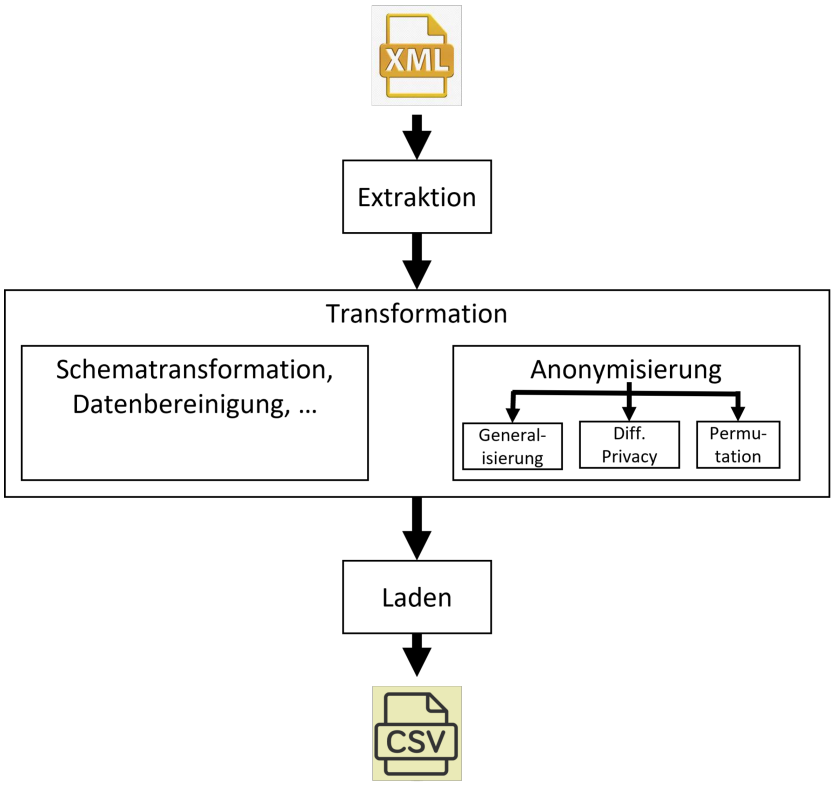

Abbildung 4.1: Übersicht über die ETL-Pipeline

Die Pipeline besteht aus drei Hauptphasen:

- **Extraktion**: Die Quelldaten liegen im XML-Format vor und werden mithilfe des DOM-Parsers in eine strukturierte Form überführt. DOM ermöglicht die Darstellung der gesamten XML-Struktur als Baum, wodurch alle enthaltenen Elemente hierarchisch erfasst und gezielt extrahiert werden können. Dies bildet die Grundlage für die nachfolgende Verarbeitung und Anonymisierung der Daten.

- **Transformation**: In diesem Schritt werden die Daten sowohl auf struktureller als auch auf inhaltlicher Ebene angepasst. Dazu gehören Schemata-Transformationen, Datenbereinigungen und insbesondere Anonymisierungsmaßnahmen. Die Anonymisierung erfolgt durch verschiedene Techniken wie Generalisierung und Maskierung, um die Datenschutzanforderungen zu erfüllen.
- **Laden**: Die transformierten und anonymisierten Daten werden in ein CSV-Format überführt, um eine weitere Nutzung in Analyse- oder Verarbeitungssystemen zu ermöglichen.

Im weiteren Verlauf dieses Kapitels werden die einzelnen Komponenten detailliert beschrieben.

4.1 Datenbeschreibung und Quellen

In diesem Abschnitt werden die für das Projekt genutzten Daten betrachtet. Dabei wird auf deren Herkunft, Format und inhaltliche Eigenschaften eingegangen. Zudem werden grundlegende Aspekte hinsichtlich der Verarbeitung und Nutzung der Daten thematisiert.

4.1.1 Datenformat

Die Eingabedaten liegen in semistrukturierten Dateien vor. In diesem Projekt werden sie im XML-Format bereitgestellt, das in 2.2.1 detailliert beschrieben ist. Nachstehend ist ein Beispiel für eine XML-Datei dargestellt, die die Struktur der in diesem Projekt verwendeten Daten verdeutlicht:

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Personen>
    <Person>
        <Vorname>Anna</Vorname>
        <Nachname>Müller</Nachname>
        <Geburtsjahr>1970</Geburtsjahr>
        <Beruf>Softwareentwicklerin</Beruf>
        <Gehalt>3000</Gehalt>
        <Kontakt>
            <Email>anna.mueller@example.com</Email>
            <Telefon>+49 989 1148 3854</Telefon>
        </Kontakt>
    </Person>
</Personen>
```

Listing 4.1: Beispieldatensatz im XML-Format

Das XML-Format ermöglicht eine hierarchische und flexible Strukturierung der Daten. Beispielsweise enthält jedes `Person`-Element in Abbildung 4.1 relevante Informationen wie Name, Geburtsjahr, Beruf sowie Kontaktdaten.

4.1.2 Datenquelle

Die ursprünglichen Datensätze sollten im bereits vorgestellten Format und in der beschriebenen Art vorliegen und stammen von einem Betrieb in Zusammenarbeit mit der Universität Rostock. Diese Daten stehen jedoch bisher nicht zur Verfügung. Aus diesem Grund wurde in dieser Arbeit nach möglichen, ähnlichen Datensätzen gesucht, die als Ersatz dienen können. Eine Quelle für solche Datensätze sind die Plattformen Kaggle[1] oder dblp[2], die eine Vielzahl öffentlich zugänglicher Datensätze bieten. Nach einer umfassenden Suche konnten jedoch keine geeigneten Datensätze gefunden werden, die personenbezogene Daten enthalten, wie in Abschnitt 2.3.1 beschrieben. Insbesondere fehlen Datensätze, die sowohl semistrukturierte Daten im XML-Format als auch relevante Attribute für die Anonymisierung enthalten. Daher wird in dieser Arbeit ein synthetischer Datensatz generiert, der den Anforderungen entspricht. Dieser wird auf Basis typischer Datenstrukturen erstellt, die in realen Szenarien auftreten, und dient als Grundlage für die Evaluierung der entwickelten Anonymisierungsmethoden. Ein Beispiel für die Struktur des synthetischen Datensatzes ist in Abbildnung 4.1 dargestellt.

4.2 Datenextraktion

Die Datenextraktion ist der erste Schritt in der Pipeline 4.1 und hat das Ziel, alle personenbezogenen Daten aus den bereitgestellten XML-Dateien zu extrahieren. Basierend auf den Ergebnissen der Literaturrecherche im Kapital 3 wird in dieser Arbeit das DOM als Methode zur Extraktion der XML-Daten verwendet. DOM ermöglicht das Laden und Verarbeiten eines gesamten XML-Dokuments in den Speicher, wodurch ein direkter Zugriff auf einzelne Elemente und Attribute möglich ist. Diese Methode ist besonders vorteilhaft, wenn eine flexible Navigation innerhalb der XML-Struktur erforderlich ist.

4.2.1 Struktur der Daten und Extraktionsansatz

XML-Daten sind hierarchisch organisiert, wobei jede Datei verschiedene Knoten aufweist, die strukturierte Informationen enthalten. Diese Knoten können Elemente mit weiteren Unterelementen oder Attributen sein. Die Extraktion erfolgt durch eine schrittweise Traversierung dieser Hierarchie, um alle relevanten Datenpunkte zu erfassen. Dabei werden insbesondere die untersten Ebenen identifiziert, da diese die eigentlichen Werte enthalten. Die Verarbeitung erfolgt über ein Modell, das die gesamte Struktur in einen Speicher lädt, um gezielte Abfragen und Extraktionen zu ermöglichen. Durch diese Herangehensweise ist es möglich, gezielt bestimmte Datensätze auszulesen, ohne zuvor eine detaillierte manuelle Konfiguration der Struktur vornehmen zu müssen. Dadurch wird eine flexible und dynamische Verarbeitung unterschiedlicher XML-Dateiformate ermöglicht.

[1] Kaggle: `https://www.kaggle.com/` Zugriff am: 31.01.2025
[2] dblp: `https://dblp.uni-trier.de/` Zugriff am: 31.01.2025

4.2.2 Hierarchische Datenstruktur vom XML mit DOM

Die DOM-Architektur basiert auf einer baumartigen Darstellung der XML-Daten, wobei jedes Element als Knoten im Baum betrachtet wird. Diese Struktur ermöglicht eine flexible Navigation und gezielte Extraktion von Daten. Die wichtigsten Knotentypen innerhalb des DOM-Baumes sind:

- **Element-Knoten:** Repräsentieren die Tags innerhalb des XML-Dokuments (z. B. `<Person>`, `<Name>`).
- **Attribut-Knoten:** Speichern zusätzliche Informationen zu Elementen (z. B. id = "123").
- **Text-Knoten:** Enthalten den eigentlichen Inhalt eines Elements (z. B. „Anna Müller“ innerhalb des `<Name>`-Tags).

Die Verwendung vom DOM bietet eine hohe Flexibilität, da die Struktur einer XML-Datei dynamisch erfasst werden kann, ohne dass eine vorherige Definition der Daten notwendig ist. Das gesamte XML-Dokument wird in einer baumartigen Struktur zwischengespeichert, wodurch alle Knoten und ihre Beziehungen, wie in Abbildung 4.2 dargestellt, abgebildet werden.

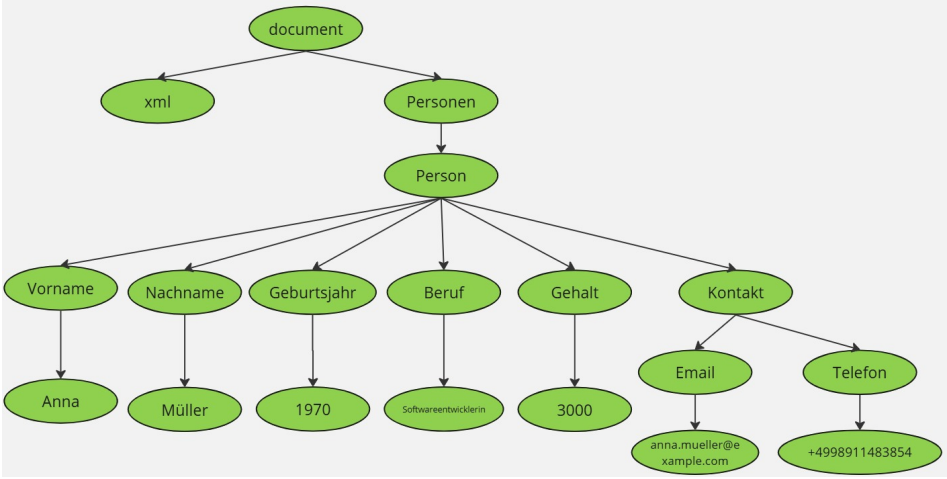

Abbildung 4.2: Hierarchische Baumstruktur der XML-Daten

4.2.3 Konzept der Datenextraktion

Die Datenextraktion erfolgt in mehreren systematischen Schritten, die eine vollständige Verarbeitung der XML-Daten ermöglichen. Dabei wird eine hierarchische Struktur erstellt, innerhalb derer die relevanten Informationen identifiziert, klassifiziert und für die weitere Verarbeitung vorbereitet werden. Die wichtigsten Schritte umfassen:

1. **Einlesen der XML-Datei:** Die extrahierten Daten werden aus der gesamten XML-Quelldatei geladen, um eine vollständige Verarbeitung zu ermöglichen.

2. **Erstellung einer hierarchischen Struktur:** Die XML-Daten werden in eine baumartige Struktur transformiert, die die Beziehungen zwischen den einzelnen Datenpunkten abbildet. Durch diese Repräsentation ist eine gezielte Navigation innerhalb der Daten möglich.

3. **Identifikation relevanter Datenpunkte:** Innerhalb der erstellten Hierarchie werden die relevanten Elemente erkannt und klassifiziert. Dabei wird insbesondere zwischen strukturellen Elementen und tatsächlichen Datenwerten unterschieden. Übergeordnete Strukturelemente dienen der Organisation, während die Blattknoten die eigentlichen Werte enthalten.

4. **Ermittlung der relevanten Attribute und Elemente:** Die extrahierten Datenpunkte werden analysiert, um personenbezogene Daten zu identifizieren und diejenigen Attribute zu bestimmen, die für die Anonymisierung relevant sind.

5. **Bereitstellung der Daten für die Weiterverarbeitung:** Nach der Extraktion der Daten, der Identifikation relevanter Elemente, der Bestimmung der personenbezogenen und zu anonymisierenden Daten sowie der sind alle Informationen in einer hierarchischen Baumstruktur organisiert, sodass eine effiziente Weiterverarbeitung, insbesondere für Transformation und Anonymisierung, gewährleistet ist.

Durch diese schrittweise Extraktion wird sichergestellt, dass die XML-Daten vollständig erfasst und effizient verarbeitet werden. Die hierarchische Struktur erleichtert die gezielte Navigation innerhalb der Daten, während die Klassifizierung eine konsistente und fehlerfreie Weiterverarbeitung ermöglichen.

4.3 Datenvorverarbeitung und -transformation

Nachdem die relevanten Daten aus den XML-Dateien extrahiert und alle notwendigen Information identifiziert wurden, erfolgt in diesem Schritt die Datenvorverarbeitung und Transformation, wie in der Pipeline 4.1. Ziel dieses Prozesses ist es, die extrahierten Informationen in eine einheitliche und strukturierte Form zu überführen, um eine konsistente Weiterverarbeitung zu ermöglichen. Die Datenvorverarbeitung umfasst verschiedene Schritte, darunter die Bereinigung fehlerhafter oder unvollständiger Werte, die Standardisierung von Formaten sowie die Umwandlung der Daten in eine geeignete Repräsentation für nachfolgende Analysen oder Anonymisierungsverfahren, sodass sie effizient für die weiteren Verarbeitungsschritte genutzt werden können. In den folgenden Abschnitten werden die einzelnen Schritte der Datenvorverarbeitung und -Transformation detailliert beschrieben.

4.3.1 Konzept zur Datenvorverarbeitung und -Transformation

Ein zentraler Schritt der Datenvorverarbeitung ist die Erkennung und Entfernung redundanter sowie fehlerhafter Daten. Durch die Analyse der extrahierten XML-Elemente

werden doppelte Datenpunkte identifiziert und eliminiert, um die Konsistenz der Datenstruktur zu gewährleisten und Speicher- sowie Verarbeitungsaufwände zu reduzieren. Zusätzlich werden inkonsistente oder fehlerhafte Werte bereinigt und standardisiert, indem Leerwerte ersetzt, falsche Formatierungen korrigiert und Zeichenfolgen vereinheitlicht werden. Im Folgenden wird der Ansatz zur Bereinigung und Transformation der XML-Daten vorgestellt, der sicherstellt, dass die extrahierten Daten korrekt, einheitlich und für weitere Analysen geeignet sind:

1. **Standardisierung von Werten und Formaten:** Alle Werte innerhalb der XML-Knoten werden auf Formatierungsabweichungen überprüft und in eine einheitliche Schreibweise überführt. Dies betrifft unter anderem Namen, Telefonnummern und Datumswerte, die oft in unterschiedlichen Formaten vorliegen. Daten, die in inkonsistenter oder unstrukturierter Form gespeichert sind (z. B. unterschiedliche Datumsformate oder Telefonnummern mit verschiedenen Schreibweisen), werden in ein konsistentes Format überführt. Durch die Standardisierung wird sichergestellt, dass alle Werte einer einheitlichen Struktur folgen, wodurch die Vergleichbarkeit und Weiterverarbeitung erleichtert werden.

2. **Korrektur und Bereinigung inkonsistenter Werte:** Werte, die in uneinheitlichen Strukturen oder Formaten vorliegen, werden erkannt und in eine standardisierte Form überführt. Falls einzelne Bestandteile von Werten fehlen oder in einer abweichenden Schreibweise vorhanden sind, werden diese ergänzt oder angepasst. Darüber hinaus werden unvollständige oder fehlerhafte Einträge überprüft und entweder korrigiert, mit passenden Werten ergänzt oder für eine spezielle Verarbeitung markiert.

3. **Entfernung unerwünschter Zeichen und Sonderzeichen:** Um die Qualität der extrahierten Daten sicherzustellen, werden unnötige Zeichen entfernt. Zusätzlich werden überflüssige Leerzeichen oder Sonderzeichen aus numerischen und textbasierten Werten entfernt, um eine konsistente Datenbasis zu gewährleisten.

4. **Erkennung und Entfernung redundanter Datensätze:** Um eine redundanzfreie und konsistente Datenbasis zu gewährleisten, werden identische Datensätze mit gleichen Werten aus der XML-Struktur entfernt. Hierbei werden die extrahierten XML-Daten auf doppelte Einträge geprüft, indem eine Kombination als sensibel eingestufter Attribute als eindeutige Signatur dient. Falls mehrere Datensätze in allen relevanten Attributen übereinstimmen, werden diese als redundant markiert und aus der XML-Struktur entfernt. Dadurch wird sichergestellt, dass nur eindeutige Datensätze erhalten bleiben, wodurch der Speicherbedarf optimiert und die Performance der nachfolgenden Verarbeitungsschritte verbessert wird.

Durch diese Bereinigung und Normalisierung entsteht eine qualitativ hochwertige und konsistente Datenbasis für die nachfolgenden Verarbeitungsschritte. Dieser mehrstufige Prozess gewährleistet eine effiziente Aufbereitung der Daten, wodurch deren Qualität erheblich verbessert wird. Gleichzeitig wird sichergestellt, dass die Daten in konsistenter und standardisierter Form für die anschließende Anonymisierung optimal bereitgestellt sind.

4.4 Datenanonymisierung

Nachdem die extrahierten Daten erfolgreich vorverarbeitet und transformiert wurden, folgt im nächsten Schritt die Anonymisierung, die zur Datentransformation gehört, wie in Abbildung 4.1 dargestellt ist. Ziel dieses Prozesses ist es, personenbezogene Informationen so zu verändern, dass eine Identifikation einzelner Personen erschwert oder unmöglich wird, während die Daten weiterhin für Analysen nutzbar bleiben. Die Anonymisierung wird durch verschiedene Methoden wie Maskierung und Generalisierung umgesetzt, die auf den Ergebnissen der Literaturrecherche in Kapitel 3 basieren und mithilfe des Werkzeugs ARX aus Abschnitt 3.4.2 realisiert werden. In den folgenden Abschnitten werden die eingesetzten Anonymisierungsverfahren ausführlich erläutert.

4.4.1 Die einzusetzenden Anonymisierungsverfahren

In dieser Arbeit werden zwei Anonymisierungsmethoden angewendet: Maskierung und Generalisierung. Beide Verfahren zielen darauf ab, personenbezogene Daten zu schützen, indem entweder direkte Identifikationsmerkmale ersetzt oder Werte in übergeordnete Kategorien zusammengefasst werden. Dadurch bleiben die Daten weiterhin für Analysen nutzbar, während die Identifizierbarkeit einzelner Personen reduziert wird.

- **Maskierung**: Sensible Informationen werden durch Platzhalter ersetzt, sodass die Originalwerte nicht mehr ersichtlich sind. Dieses Verfahren wird vor allem für eindeutig identifizierende Attribute wie Namen, Telefonnummern oder E-Mail-Adressen genutzt.
- **Generalisierung**: Die Genauigkeit von Daten wird reduziert, indem spezifische Werte in allgemeinere Kategorien überführt werden. Dadurch wird die Gruppierung von Datensätzen ermöglicht, was die Identifikation einzelner Personen erschwert.

Die folgende Tabelle 4.1 veranschaulicht den Unterschied zwischen Maskierung und Generalisierung anhand verschiedener Attribute:

Attribut	**Originaldaten**	**Maskierung**	**Generalisierung**
Vorname	Max	***	M***
Nachname	Jack	***	J***
Telefon	0123456789	0123XXXXXX	0123XXXXXX
E-Mail	m.j@example.com	***@example.com	*@example.com
G.Datum	15.07.1985	XX.XX.XXXX	XX.XX.198X
Gehalt	3500 €	3500 €	3500 €
Beruf	Softwareentwickler	Softwareentwickler	Softwareentwickler

Tabelle 4.1: Vergleich von Maskierung und Generalisierung

Diese Methoden werden gezielt auf ausgewählte Attribute angewendet, um die Privatsphäre der betroffenen Personen zu schützen und gleichzeitig die analytische Nutzbarkeit der Daten zu gewährleisten.

4.4.2 Einstellungen vom ARX

Das ARX-Framework bietet eine flexible und umfassende Konfigurationsmöglichkeit zur Anonymisierung sensibler Daten. Durch die Definition von Datenschutzmodellen, Attributtypen und Transformationsregeln ermöglicht ARX die Anpassung der Anonymisierung an spezifische Anforderungen. In diesem Abschnitt werden die wesentlichen Konfigurationsparameter erläutert [7].

Definition von Attributtypen

Jedes Attribut in der zu anonymisierenden Datenmenge muss einem bestimmten Typ zugeordnet werden. ARX unterscheidet dabei zwischen [7]:

- **Identifizierende Attribute:** Direkt personenbezogene Daten wie Namen oder eindeutige Identifikationsnummern. Diese Attribute sollten entweder vollständig entfernt oder durch starke Maskierung anonymisiert werden.
- **Quasi-identifizierende Attribute:** Attribute, die in Kombination zur Reidentifikation einer Person genutzt werden können, z. B. Geburtsdatum, Geschlecht oder Postleitzahl. Diese Attribute werden durch Generalisierung oder Suppression transformiert.
- **Sensible Attribute:** Daten, die schützenswerte Informationen enthalten, z. B. Gesundheitszustand oder Einkommen. Diese Attribute werden in der Regel nicht direkt verändert, sondern durch k-Anonymität oder andere Schutzmechanismen abgesichert.
- **Nicht-sensible Attribute:** Daten, die keine Reidentifikation ermöglichen und unverändert bleiben können.

Anonymisierungsstrategien

ARX bietet verschiedene Mechanismen zur Anonymisierung, die je nach Attributtyp konfiguriert werden [7]:

- **Generalisierung:** Werte werden in weniger spezifische Kategorien zusammengefasst, z. B. wird eine exakte Altersangabe in Altersgruppen umgewandelt.
- **Maskierung:** Sensible Informationen werden durch Platzhalter ersetzt, z. B. werden E-Mail-Adressen teilweise maskiert (`*@example.com`).
- **Rauschen hinzufügen:** Numerische Werte werden durch das Hinzufügen zufälliger Variationen modifiziert, um eine präzise Rückverfolgung zu erschweren.
- **Suppression:** Falls für bestimmte Daten keine ausreichende Anonymisierung möglich ist, werden diese vollständig entfernt.

Schutzmodelle

ARX ermöglicht die Auswahl verschiedener Datenschutzmodelle, die je nach Anforderungen kombiniert werden können [7]:

- **k-Anonymität:** Stellt sicher, dass jede Kombination von Quasi-Identifikatoren mindestens k-mal in der Datenmenge vorkommt, um die Identifizierbarkeit zu minimieren.
- **ℓ-Diversität:** Erweitert die k-Anonymität, indem sichergestellt wird, dass sensible Attribute innerhalb einer Gruppe ausreichend divers sind.
- **t-Closeness:** Stellt sicher, dass die Verteilung sensibler Attribute innerhalb einer Gruppe der ursprünglichen Verteilung in der Gesamtpopulation entspricht.

Hierarchien zur Datenanonymisierung

Für viele Anonymisierungsmethoden werden Hierarchien benötigt, um Werte schrittweise zu generalisieren. ARX unterstützt verschiedene Hierarchietypen [7]:

- **Manuelle Hierarchien:** Benutzerdefinierte Regeln zur Generalisierung von Attributen.
- **Automatische Hierarchien:** Generierung von Generalisierungsstufen basierend auf Datenverteilungen.
- **Hierarchien für numerische Werte:** Definierte Intervalle oder Cluster zur Aggregation von Zahlenwerten.

Qualitätsmetriken und Evaluierung

Um die Auswirkungen der Anonymisierung auf die Datenqualität zu messen, bietet ARX verschiedene Metriken [7]:

- **Informationsverlust:** Berechnung, wie stark die Anonymisierung die ursprünglichen Werte verändert.
- **Granularitätsstufen:** Bewertung der Generalisierungsstufen anhand definierter Hierarchien.
- **Statistische Ähnlichkeit:** Messung, wie stark sich die anonymisierten Daten von den Originaldaten unterscheiden.

Zusammenfassung

Die Konfiguration von ARX ermöglicht eine detaillierte Anpassung der Anonymisierungsstrategie an spezifische Datenschutzanforderungen. Durch die Kombination verschiedener Attributtypen, Schutzmodelle und Transformationsmechanismen kann eine Balance zwischen Datenschutz und Datenverwendbarkeit erreicht werden. Die Evaluierung der Datenqualität nach der Anonymisierung stellt sicher, dass die transformierten Daten weiterhin für analytische Zwecke nutzbar bleiben.

4.4.3 Konzept der Anonymisierung

In diesem Abschnitt wird das Konzept der Anonymisierung systematisch dargestellt. Zunächst wird erläutert, wie die extrahierten XML-Daten für den Anonymisierungsprozess aufbereitet und in eine tabellarische Struktur überführt werden. Anschließend wird die Identifikation sowohl schützenswerter als auch nicht-sensibler Attribute durchgeführt. Daraufhin erfolgt die Festlegung von Anonymisierungshierarchien, die verschiedene Generalisierungsstufen für die betreffenden Attribute enthalten. Während der Anwendung der Anonymisierung wird durch die Anwendung von Schutzmodellen wie K-Anonymität und ℓ-Diversität sichergestellt, dass eine Rückführbarkeit einzelner Datensätze erschwert wird.

Datenaufbereitung und Strukturierung

Bevor die Anonymisierung erfolgen kann, müssen die aus den XML-Dateien extrahierten Daten, die ursprünglich in einer baumartigen Struktur vorliegen, in eine relationale Form überführt werden. Dies ist erforderlich, da das Anonymisierungstool bzw. ARX eine tabellarische Datenstruktur für die Verarbeitung benötigt. Im Rahmen dieses Prozesses werden die relevanten Attribute und ihre zugehörigen Werte aus der baumartigen Struktur extrahiert. Nach der Extraktion erfolgt die Transformation der Daten von der Baumstruktur in eine relationale Tabelle. Dabei wird jedes XML-Element einer Zeile in der Zieldatenstruktur zugeordnet, während die Attribute als Spalten und ihre jeweiligen Werte als Zellen erfasst werden. Diese Umwandlung ist erforderlich, da die Eingabedaten für ARX eine tabellarische Struktur erfordern, um eine effiziente Verarbeitung und Anwendung der Anonymisierungsmethoden zu ermöglichen.

Auswahl des Schutzmodells

Da ARX verschiedene Schutzmodelle unterstützt, muss ein Modell für die Anonymisierung ausgewählt werden. In dieser Arbeit werden *k-Anonymität* sowie ℓ-Diversität als Schutzprinzipien eingesetzt, um sicherzustellen, dass anonymisierte Daten nicht durch die Kombination verschiedener Merkmale Rückschlüsse auf einzelne Personen ermöglichen. Die k-Anonymität gewährleistet, dass jede Kombination aus schützenswerten Attributen in der Datenmenge mindestens k-mal vorkommt, sodass eine eindeutige Identifikation verhindert wird. Dazu wird eine Mindestanzahl von k Datensätzen für jede Gruppe ähnlicher Merkmale festgelegt. Falls eine Gruppe diese Bedingung nicht erfüllt, werden zusätzliche Generalisierungen oder Maskierungen angewendet, bis die gewünschte Anonymitätsstufe erreicht ist. Die ℓ-Diversität erweitert die k-Anonymität, indem sie fordert, dass innerhalb jeder Gruppe von Datensätzen mindestens ℓ unterschiedliche Ausprägungen des sensitiven Attributs vorhanden sind, um Rückschlüsse auf Einzelpersonen weiter zu erschweren. Dadurch wird verhindert, dass innerhalb einer anonymisierten Gruppe eine zu hohe Homogenität sensibler Daten besteht, die Rückschlüsse auf den ursprünglichen Wert ermöglichen könnte. Da die Wahl der Parameter k und ℓ entscheidend für den Schutz der Daten sowie deren Nutzbarkeit ist, wurden diese experimentell bestimmt. Hierbei wurden verschiedene Werte getestet, um ein Gleichgewicht zwischen Datenschutzanforderungen und Datenqualität zu gewährleisten. Die

beste Werte für k und ℓ wurden durch iterative Tests ermittelt, bei denen sowohl das Risiko einer Reidentifikation als auch der Informationsverlust durch die Anonymisierung berücksichtigt wurden.

Erkennung und Klassifizierung der zu anonymisierenden Attribute

Nicht alle Attribute erfordern eine Anonymisierung. Daher müssen sensible Merkmale in den Daten identifiziert und entsprechend klassifiziert werden. Dazu gehören in der Regel direkt oder indirekt identifizierende Informationen, wie Namen, Telefonnummern, E-Mail-Adressen und Geburtsdaten. Diese Attribute werden als schützenswerte Informationen markiert und für die Anonymisierung vorbereitet. Die Bestimmung der zu anonymisierenden Attribute erfolgt nicht automatisiert, sondern basiert auf einer statischen Liste vordefinierter Attribute. Dadurch wird sichergestellt, dass ausschließlich relevante Informationen anonymisiert werden, während andere Attribute für die weitere Verarbeitung unverändert bleiben. Deswegen sollten alle Attribute entsprechend ihrer Merkmale einem der folgenden Attributtypen zugewiesen werden:

- **Direkt identifizierende Attribute:** Diese ermöglichen eine eindeutige Identifikation einer Person und stellen somit ein hohes Reidentifizierungsrisiko dar. Sie werden aus dem Datensatz entfernt oder anonymisiert. Typische Beispiele: `Name`, `Personalausweisnummer`.
- **Quasi-Identifizierende Attribute:** Diese Merkmale können in Kombination mit anderen Informationen zur Re-Identifizierung von Personen genutzt werden. Daher werden sie durch Generalisierung oder andere Transformationen geschützt. Typische Beispiele: `E-Mail`, `Geburtsdatum`, `Telefonnummer`, `Postleitzahl`.
- **Sensible Attribute:** Diese Attribute enthalten vertrauliche Informationen, die einer betroffenen Person Schaden zufügen oder zu Diskriminierung führen könnten. Sie bleiben unverändert, können aber durch zusätzliche Datenschutzmechanismen wie ℓ-Diversität oder t-Nähe geschützt werden. Typische Beispiele: `Diagnosen`, `Einkommen`, `Religionszugehörigkeit`.
- **Nicht-Sensible Attribute:** Diese enthalten keine identifizierenden oder sensiblen Informationen und stellen kein Datenschutzrisiko dar. Sie bleiben unverändert. Typische Beispiele: `Berufsbezeichnung`, `Arbeitsort`, `Kundennummer (ohne Personenbezug)`.

Auswahl der Anonymisierungsverfahren

Das ARX-Framework unterstützt verschiedene Anonymisierungstechniken, darunter insbesondere die Maskierung und die Generalisierung. Diese beiden Verfahren kommen in dieser Arbeit zum Einsatz, um personenbezogene Daten zu schützen und eine Rückführbarkeit einzelner Datensätze zu erschweren. In dieser Arbeit werden beide Verfahren mit ARX umgesetzt, wobei die Anonymisierungshierarchien entsprechend definiert werden. Während die Maskierung eine vollständige Unkenntlichmachung bestimmter Werte bewirkt, ermöglicht die Generalisierung eine schrittweise Reduktion der Spezifität.

Definition von Maskierungshierarchien

Es müssen für jedes zu anonymisierendem Attribut Generalisierungsstufen definiert werden. Diese Stufen legen fest, in welchem Umfang die Originalwerte durch Platzhalter ersetzt werden. Die Maskierung erfolgt in mehreren Abstufungen, um die Balance zwischen Datenschutz und Datenverwendbarkeit zu optimieren. Im Unterbeispiel werden einigen zu anonymisierenden Attributen mit den dazu gehörigen Generalisierungsstufen dargestellt:

- **Namen**:
  - `Max` → `M***` → `***`
  - `Mustermann` → `M*********` → `***`
- **Telefonnummern**:
  - `+49 123 456789` → `+49 XXX XXXXXX` → `+49 XXXXXXX` → `***`
- **E-Mail-Adressen**:
  - `max.mustermann@example.com` → `*******@example.com` → `*@example.com` → `***`
- **Geburtsdatum**:
  - `15.07.1985` → `XX.XX.1985` → `XX.XX.198X` → `***`

Definition von Generalisierungshierarchien

Für jedes zu anonymisierende Attribut müssen Generalisierungshierarchien definiert werden. Diese legen fest, in welchem Umfang die Originalwerte durch allgemeinere Kategorien ersetzt werden. Die Generalisierung erfolgt in mehreren Stufen, um die Balance zwischen Datenschutz und Datenverwendbarkeit zu optimieren. Nachfolgend werden einige zu anonymisierende Attribute mit den dazugehörigen Generalisierungsstufen dargestellt:

- **Geburtsjahr**:
  - `1985` → `198X` → `19XX` → `***`
  - `1992` → `199X` → `19XX` → `***`
- **Postleitzahl**:
  - `12345` → `123XX` → `12XXX` → `***`
  - `98765` → `987XX` → `98XXX` → `***`
- **Alter**:
  - `28` → `25-30` → `20-40` → `***`
  - `42` → `40-50` → `30-60` → `***`

Einsatz der Anonymisierung

Nach der Vorbereitung der Daten, der Auswahl des Anonymisierungsverfahrens sowie der Definition der zugehörigen Generalisierungshierarchien und der Klassifikation der Attribute basierend auf ihren Merkmalen erfolgt die eigentliche Anonymisierung mithilfe der ARX-Java-API und den entsprechenden ARX-JAR-Dateien. Die zuvor identifizierten schützenswerten Attribute werden entsprechend den festgelegten Generalisierungsstufen transformiert, um eine direkte oder indirekte Identifikation zu verhindern. Während der Anonymisierung gewährleistet ARX, dass:

- Die definierten Generalisierungsregeln konsistent auf alle betroffenen Attribute angewendet werden,
- Die Originalwerte durch die entsprechenden generalisierten Werte ersetzt werden,
- Die Datenstruktur erhalten bleibt, sodass eine weitere Verarbeitung und Analyse der anonymisierten Daten problemlos möglich ist.

Die Implementierung erfolgt durch die Integration der ARX-JAR-Dateien in das Projekt sowie die Nutzung der ARX-Java-API zum Einsatz des Anonymisierungsprozesses in der Programmiersprache *JAVA*. Dabei werden die Anonymisierungsmodelle und Generalisierungsstufen in den Quellcode eingebunden, um eine automatisierte und reproduzierbare Anonymisierung sicherzustellen.

4.5 Datenspeicherung

In diesem Abschnitt wird das Konzept der Datenspeicherung als letzter Schritt der ETL-Pipeline aus der Abbildung 4.1 im Kontext der Anonymisierung erläutert. Die Daten werden in einer relationalen Struktur als CSV-Dateien gespeichert, wie bereits im Kapital 2 aus Abschnitt 2.5 beschrieben. Diese Form der Speicherung gewährleistet eine strukturierte und standardisierte Ablage der anonymisierten Informationen und ermöglicht eine einfache Weiterverarbeitung.

4.5.1 Konzept zur Speicherung anonymisierter Daten

Die anonymisierten Daten werden nach der Verarbeitung in einer strukturierten und standardisierten Form gespeichert, um ihre weitere Nutzung für Analysen oder Verarbeitungsschritte zu ermöglichen.

Speicherformat

Die Speicherung erfolgt in CSV-Dateien, wobei jede Zeile einen vollständigen Datensatz repräsentiert, während die Spalten die einzelnen Attribute enthalten. Die Struktur der CSV-Datei bleibt konsistent mit den ursprünglichen Attributen, jedoch enthalten die Werte anonymisierte Informationen. Beispielhafte Speicherung der anonymisierten Daten im CSV-Format:

Vorname	Nachname	Geburtsdatum	Telefon	E-Mail	Gehalt
M***	****	XX.XX.1985	+49 XXXXX	***@example.com	3000
J***	****	XX.XX.197X	+49 XXXXX	***@example.com	3500

Tabelle 4.2: Beispielhafte Speicherung anonymisierter Daten

In diesem Beispiel wurden Namen, Geburtsdaten, Telefonnummern und E-Mails anonymisiert, während die Struktur der Daten erhalten bleibt.

Dateiorganisation

Um eine klare Trennung zwischen unterschiedlichen Anonymisierungsverfahren sicherzustellen, werden separate Dateien für jede Methode verwendet:

- **Maskierung:** Enthält die durch Maskierung anonymisierten Daten, bei denen bestimmte Werte durch Platzhalter ersetzt wurden.
- **Generalisierung:** Enthält die durch Generalisierung veränderten Daten, bei denen Werte auf einer höheren Abstraktionsebene dargestellt sind.

5 Implementierung

In diesem Kapitel wird die technische Umsetzung des Projekts ausführlich beschrieben. Es werden die eingesetzten Technologien, der Ablauf der Datenextraktion, die angewendeten Anonymisierungsmethoden sowie die Speicherung der verarbeiteten Daten erläutert. Abschließend werden die Ergebnisse der Implementierung vorgestellt und analysiert. Die Implementierung basiert auf einer klar strukturierten Architektur, die in der Abbildung 5.1 dargestellt und in vier Hauptkomponenten unterteilt ist: Datenextraktion, Transformation, Anonymisierung und Speicherung. Jede dieser Komponenten wird durch spezifische Klassen realisiert, die miteinander interagieren, um eine funktionale ETL-Pipeline mit integrierten Anonymisierungsverfahren zu gewährleisten.

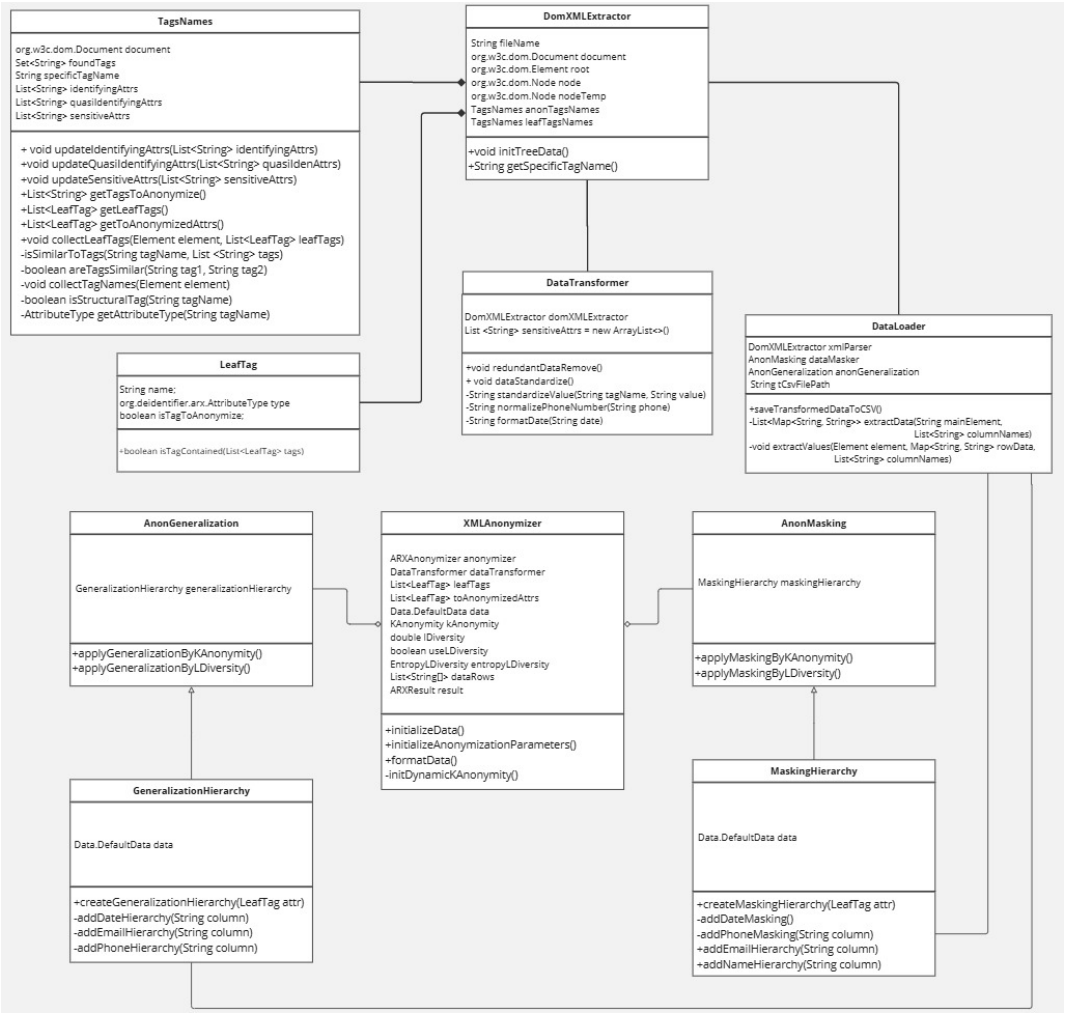

Abbildung 5.1: Klassendiagramm zur in Abschnitt 4.1 vorgestellten ETL-Pipeline

Im Folgenden wird der Inhalt der einzelnen Klassen genauer erläutert. Dabei wird beschrieben, welche Funktionalität sie innerhalb der ETL-Pipeline übernehmen und wie sie miteinander in Beziehung stehen.

5.1 Verwendete Technologien und Frameworks

In diesem Abschnitt werden die eingesetzten Technologien und Frameworks vorgestellt, die für die Umsetzung des vorgestellten Verfahrens genutzt wurden. Dazu gehören die Programmiersprache, verwendete Bibliotheken sowie relevante Software- und Entwicklungswerkzeuge.

5.1.1 Programmiersprache Java

Für die Implementierung wurde die Programmiersprache Java Version 23.0.2 verwendet. Java eignet sich besonders für datenintensive Anwendungen und bietet eine Vielzahl an Bibliotheken für die Verarbeitung, Transformation und Anonymisierung von Daten. Zudem erleichtert die objektorientierte Struktur von Java die Modularisierung der einzelnen Verarbeitungsschritte im ETL-Prozess.

5.1.2 Frameworks für Datenverarbeitung

Für die verschiedenen Verarbeitungsschritte im ETL-Prozess wurden spezialisierte Frameworks und Bibliotheken verwendet, die alle in der JAR-Datei libarx-3.9.1 enthalten sind. Diese ermöglichen eine effiziente Extraktion, Transformation und Anonymisierung der Daten, indem sie leistungsstarke Methoden zur Verarbeitung und Sicherstellung des Datenschutzes bereitstellen.

Datenvorverarbeitung und -Transformation mit dem DOM-Framework

Zur Extraktion von Daten aus XML-Dateien wurde das DOM in Kombination mit den Paketen *org.w3c.dom* und *javax.xml.parsers* verwendet. Der DOM-Parser ermöglicht das Laden und Verarbeiten von XML-Dokumenten in einer baumartigen Struktur, sodass auf die Datenknoten direkt zugegriffen werden kann. Das Paket *org.w3c.dom* enthält die folgenden Hauptklassen, die zur Verarbeitung der XML-Daten genutzt wurden:

- **Document**: Repräsentiert das gesamte XML-Dokument als Baumstruktur.
- **Element**: Stellt ein XML-Element dar.
- **NodeList**: Enthält eine Liste von XML-Knoten, die z. B. durch *getElementsByTagName* abgerufen werden.
- **Node**: Repräsentiert ein einzelnes Element, Attribut oder Textfragment innerhalb des XML-Baums.

Das Paket *javax.xml.parsers* stellt die notwendigen Werkzeuge zur Verfügung, um DOM-Dokumente zu erstellen und zu verwalten:

- **DocumentBuilderFactory**: Erstellt eine Instanz eines DOM-Parsers.
- **DocumentBuilder**: Verarbeitet eine XML-Datei und erzeugt daraus ein *Document*-Objekt.

Datenanonymisierung mit dem ARX-Framework

Für die Anonymisierung personenbezogener Daten wurde das ARX-Framework mit Versionasnummer *3.9.1* verwendet. ARX ist eine leistungsfähige Open-Source-Bibliothek, die verschiedene Datenschutzmodelle unterstützt, darunter:

- **k-Anonymität:** Sorgt dafür, dass jede Gruppe mit denselben Quasi-Identifikatoren mindestens k Einträge umfasst.
- **ℓ-Diversität:** Stellt sicher, dass innerhalb jeder Gruppe von k anonymisierten Datensätzen mindestens ℓ unterschiedliche Werte für sensible Attribute vorhanden sind.

Das Framework ermöglicht es, Generalisierung, Maskierung und Hierarchien zur Anonymisierung effizient umzusetzen. Dabei werden die folgenden Klassen vom ARX dafür verwendet:

- **org.deidentifier.arx.***: Enthält die Hauptklassen für das Anonymisierungsframework, darunter *ARXAnonymizer* zur Durchführung der Anonymisierung.
- **EntropyLDiversity**: Implementiert das ℓ-Diversitätsmodell zur Sicherstellung der Diversität sensibler Attribute.
- **KAnonymity**: Stellt sicher, dass keine eindeutige Identifikation einer Person möglich ist.
- **HierarchyBuilderRedactionBased** – Dient zur Definition einer Hierarchie durch Maskierung mit Zeichen.
- **Hierarchy**: Wird zur Verwaltung von Attributhierarchien für Generalisierung und Anonymisierung verwendet.
- **Data**: Repräsentiert die Eingabedaten für die Anonymisierung.
- **DataHandle**: Bietet Methoden zum Zugriff auf die anonymisierten Daten nach der Verarbeitung.
- **Element**: Wird aus der *org.w3c.dom*-Bibliothek verwendet, um XML-Datenknoten zu repräsentieren.
- ***NodeList*** – Erlaubt die Iteration über mehrere XML-Knoten, um z. B. alle vorhandenen Datensätze zu verarbeiten.

Datenspeicherung mit FileWriter und Streams

Nach der Verarbeitung und Anonymisierung der Daten erfolgt die Speicherung in einer CSV-Datei. Hierfür wurde die FileWriter-Klasse aus der Java-Standardbibliothek verwendet. Zusätzlich wurden Streams und Collectors genutzt, um die Daten effizient in das gewünschte Format umzuwandeln und die Speicherung zu optimieren. Die folgenden Java-Klassen wurden für die Speicherung der anonymisierten Daten eingesetzt:

- **ARXResult**: Speichert das Ergebnis der Anonymisierung, das durch ARX erzeugt wird.
- **DataHandle**: Wird genutzt, um auf die anonymisierten Daten im ARXResult zuzugreifen und diese weiterzuverarbeiten.
- **FileWriter**: Schreibt die anonymisierten Daten in eine CSV-Datei.
- **IOException**: Wird verwendet, um Fehler beim Speichern der Datei zu behandeln.
- **Collectors**: Eine Hilfsklasse aus der Stream-API, die für das Umwandeln der Daten in CSV-geeignete Formate genutzt wird.

Nach der Anonymisierung wird das Ergebnis aus dem ARXResult-Objekt extrahiert und in einer CSV-Datei gespeichert.

5.2 Datenvorverarbeitung und -Transformation

Die Datenvorverarbeitung stellt den ersten Schritt in Pipeline (siehe Abbildung 4.1) dar, um Rohdaten aus den bereitgestellten Datenquellen zu extrahieren, zu bereinigen und in ein standardisiertes Format zu überführen. Zunächst erfolgt die Extraktion der XML-Daten mithilfe des DOM-Parsers, wodurch die Datenstruktur analysiert und relevante Informationen extrahiert werden. Anschließend werden die extrahierten Daten einer Bereinigung unterzogen, um doppelte oder inkonsistente Einträge zu erkennen und zu entfernen. Nach der Bereinigung folgt die Standardisierung, um sicherzustellen, dass Werte konsistent formatiert sind und keine abweichenden Schreibweisen oder unerwarteten Zeichen enthalten. Dieser Schritt umfasst unter anderem die Vereinheitlichung von Datumsangaben, Telefonnummern und Namensformaten. Abschließend werden die Daten für die weitere Verarbeitung vorbereitet, indem sie transformiert und typisiert werden. Dabei werden Attribute entsprechend ihrer Rolle im Anonymisierungsprozess klassifiziert, beispielsweise als Quasi-Identifikatoren oder sensible Daten. Durch diese mehrstufige Datenverarbeitung entsteht eine strukturierte und bereinigte Datenbasis, die als Grundlage für die Anonymisierung dient und eine hohe Qualität der verarbeiteten Informationen gewährleistet.

5.2.1 Datenextraktion mit DOM

Die Rohdaten liegen im XML-Format vor und werden mithilfe des DOM-Parsers verarbeitet. Die Klasse *DomXMLExtractor* übernimmt die Extraktion der XML-Daten, indem sie die Datei einliest, die XML-Baumstruktur analysiert und die relevanten Elemente identifiziert. Dazu wird das DOM-Modell genutzt, um die hierarchische Struktur der XML-Datei vollständig in den Speicher zu laden und gezielt auf die benötigten Knoten zuzugreifen. Die Initialisierung der Baumstruktur erfolgt über die Methode *initTreeData* (siehe Listing 5.1). Hierbei wird die XML-Datei geöffnet und geparst, sodass eine Dokumentinstanz erzeugt wird, die anschließend weiterverarbeitet werden kann. Nach der Normalisierung der XML-Struktur werden die Blattelemente identifiziert und in einer separaten Datenstruktur gespeichert, um sie für die weiteren Verarbeitungsschritte verfügbar zu machen. Um die Struktur der XML-Daten genau zu bestimmen, analysiert die Methode *getSpecificTagName* (siehe Listing 5.2) den Inhalt der Datei und identifiziert das Hauptelement, indem sie das am häufigsten vorkommende Tag ermittelt. Diese Information wird genutzt, um die relevanten Attribute der XML-Daten zu erfassen und zu klassifizieren. Dabei werden zwei Gruppen von Attributen unterschieden: Einerseits die allgemeinen Blattelemente der Datei, die extrahiert und weiterverarbeitet werden, andererseits die Attribute, die für die Anonymisierung vorgesehen sind. Diese strukturierte Extraktion bildet die Grundlage für die weiteren Verarbeitungs- und Anonymisierungsschritte.

```
1  public void initTreeData() throws Exception {
2      File xmlFile = new File(this.getFileName());
3      DocumentBuilderFactory factory = DocumentBuilderFactory.
           newInstance();
4      DocumentBuilder builder = factory.newDocumentBuilder();
5      this.setDocument(builder.parse(xmlFile));
6      this.setAnonTagsNames(new TagsNames(this.getDocument(),
           this.getSpecificTagName()));
7      this.getAnonTagsNames().getTagsToAnonymize();
8      this.setLeafTagsNames(new TagsNames(this.getDocument()));
9      this.getLeafTagsNames().getLeafTags();
10     this.getDocument().getDocumentElement().normalize();
11     this.setRoot(getDocument().getDocumentElement());
12     this.setNode(this.getRoot());
13 }
```

Listing 5.1: Initialisierung der Baumstruktur im DomXMLExtractor

Die Bestimmung des Hauptelements der XML-Struktur erfolgt über die Methode *getSpecificTagName* (siehe Listing 5.2). Hierbei wird die Anzahl der Vorkommen jedes Tags innerhalb der obersten Hierarchieebene gezählt, um das am häufigsten vorkommende Element als Haupteintrag der Datenstruktur zu bestimmen. Dadurch kann sichergestellt werden, dass die nachfolgenden Verarbeitungsschritte mit einer konsistenten und strukturierten Datengrundlage arbeiten.

```
1  public String getSpecificTagName() {
2      Element root = this.getDocument().getDocumentElement();
3      NodeList childNodes = root.getChildNodes();
4      Map<String, Integer> tagCount = new HashMap<>();
5      for (int i = 0; i < childNodes.getLength(); i++) {
6          if (childNodes.item(i) instanceof Element) {
7              String tagName = childNodes.item(i).getNodeName();
8              tagCount.put(tagName, tagCount.getOrDefault(tagName
                   , 0) + 1);
9          }
10     }
11     return tagCount.entrySet().stream()
12                     .max(Map.Entry.comparingByValue())
13                     .map(Map.Entry::getKey)
14                     .orElse(null);
15 }
```

Listing 5.2: Bestimmung des Hauptelements in DomXMLExtractor

Nach dem Laden der XML-Daten identifiziert der *DomXMLExtractor* die relevanten Elemente und speichert diese für die Transformation und Anonymisierung ab. Durch diesen strukturierten Extraktionsprozess wird sichergestellt, dass die Daten in einer einheitlichen und standardisierten Form vorliegen, um in den nachfolgenden Schritten effizient weiterverarbeitet werden zu können.

5.2.2 Datenbereinigung und Standardisierung

Nach der Extraktion der Rohdaten erfolgt die Bereinigung, um die Qualität und Konsistenz der Daten sicherzustellen. Hierbei werden doppelte oder nicht benötigte Datensätze erkannt und entfernt, sodass eine redundanzfreie und einheitliche Datenbasis gewährleistet wird. Die Klasse *DataTransformer* übernimmt diese Aufgabe und stellt Methoden zur Erkennung redundanter Datensätze sowie zur Standardisierung von Werten bereit. Dies umfasst die Formatierung von Telefonnummern, die Normalisierung von Namen sowie die Umwandlung von Datumswerten in ein einheitliches Format. Ein zentraler Schritt der Bereinigung ist die Entfernung doppelter Datensätze. Dies erfolgt über die Methode *redundantDataRemove* (siehe Listing siehe Listing 5.3), die alle Dateneinträge anhand bestimmter Merkmale überprüft. Hierbei wird eine eindeutige Signatur generiert, indem die Werte der als sensibel eingestufter Attribute zu einer Zeichenkette kombiniert werden. Falls eine identische Signatur bereits existiert, wird der entsprechende Eintrag als Duplikat erkannt und aus der XML-Struktur entfernt. Dieser Prozess sorgt nicht nur für eine platzsparende Speicherung, sondern verbessert auch die Performance der nachfolgenden Verarbeitungsschritte. Nachdem redundante Einträge entfernt wurden, erfolgt die Standardisierung der Daten durch die Methode *dataStandardize* (siehe Listing 5.4). Um eine einheitliche Darstellung der Informationen zu gewährleisten, werden verschiedene Anpassungen vorgenommen.

```
1  public void redundantDataRemove() {
2      Set<String> uniqueEntries = new HashSet<>();
3      Element root = domXMLExtractor.getRoot();

5      NodeList allNodes = root.getChildNodes();
6      List<Element> elementsToRemove = new ArrayList<>();
7      for (int i = 0; i < allNodes.getLength(); i++) {
8          if (!(allNodes.item(i) instanceof Element element)) {
9              continue;
10         }
11         StringBuilder entrySignature = new StringBuilder();
12         NodeList childNodes = element.getChildNodes();
13         for (int j = 0; j < childNodes.getLength(); j++) {
14             if (childNodes.item(j) instanceof Element
                   childElement && this.getSensitiveAttrs().
                   contains(childElement.getTagName())) {
15                 entrySignature.append(childElement.getTagName()
                       ).append("=")
16                         .append(childElement.getTextContent().
                               trim()).append(";");
17             }
18         }
19         String signature = entrySignature.toString();
20         if (uniqueEntries.contains(signature)) {
21             elementsToRemove.add(element);
22         } else {
23             uniqueEntries.add(signature);
24         }
25     }
26     for (Element element : elementsToRemove) {
27         root.removeChild(element);
28     }
29 }
```

Listing 5.3: Entfernung redundanter Datensätze im DataTransformer

Die Standardisierung der extrahierten Werte erfolgt in der Methode *standardizeValue* (siehe Listing 5.4). Dabei wird jeder Wert anhand seines jeweiligen Attributs überprüft und in das entsprechende Format überführt.

```
1  private String standardizeValue(String tagName, String value) {
2      String originalValue = value.trim();
3      String newValue = originalValue;

5      if (tagName.equalsIgnoreCase("Vorname") || tagName.
           equalsIgnoreCase("Nachname")) {
6          newValue = value.substring(0, 1).toUpperCase() + value.
               substring(1).toLowerCase();
```

```
7       } else if (tagName.equalsIgnoreCase("Gehalt")) {
8           newValue = value.replaceAll("[^0-9.]", "");
9       } else if (tagName.equalsIgnoreCase("Geburtsjahr") ||
            tagName.equalsIgnoreCase("Geburtsdatum") || tagName.
            equalsIgnoreCase("year")) {
10          newValue = formatDate(value);
11      } else if (tagName.toLowerCase().contains("telefon") ||
            tagName.toLowerCase().contains("phone")) {
12          newValue = normalizePhoneNumber(value);
13      } else if (tagName.toLowerCase().contains("email")) {
14          newValue = value.toLowerCase();
15      }

17      return newValue;
18  }
```

Listing 5.4: Standardisierung der Werte im DataTransformer

Diese Maßnahmen stellen sicher, dass die bereinigten Daten eine einheitliche Struktur aufweisen und für die weitere Verarbeitung, insbesondere die Anonymisierung, problemlos genutzt werden können.

5.2.3 Typisierung der Daten für die Anonymisierung

Nach der Transformation werden die Daten für die Anonymisierung vorbereitet. Dabei erfolgt eine Klassifikation der Attribute, um festzulegen, welche Daten anonymisiert werden müssen und mit welcher Methode dies geschieht. Die Klasse TagsNames übernimmt diese Einordnung und unterscheidet zwischen Identifikatoren, Quasi-Identifikatoren und sensiblen Attributen. Um diese Einordnung effizient durchzuführen, sind drei vordefinierte Listen erforderlich. Zwei Listen für Identifikatoren und Quasi-Identifikatoren, die festlegen, welche Attribute generalisiert oder maskiert werden müssen und in welchem Umfang dies geschieht, sowie eine weitere Liste für sensible Attribute, die einer besonders starken Anonymisierung oder Maskierung unterzogen werden. Die Klasse *LeafTag* (siehe Listing 5.5) verwaltet die Attribute und bestimmt, ob ein bestimmtes Tag anonymisiert werden soll.

```
1  public class LeafTag {
2      private String name;
3      private AttributeType type;
4      private boolean isTagToAnonymize;
5      public LeafTag(String name, AttributeType type) {
6          this.setName(name);
7          this.setType(type);
8          this.setTagToAnonymize(false);
9          if(this.type == AttributeType.IDENTIFYING_ATTRIBUTE)
10             this.setTagToAnonymize(true);
11         if(this.type == AttributeType.
               QUASI_IDENTIFYING_ATTRIBUTE)
```

```
12              this.setTagToAnonymize(true);
13      }
14      public boolean isTagContained(List<LeafTag> tags){
15          for(LeafTag tag : tags){
16              if(tag.getName().equalsIgnoreCase(this.getName())){
17                  return true;
18              }
19          }
20          return false;
21      }
22  }
```

Listing 5.5: Tagleaf-Klasse

Die Methode *getTagsToAnonymize* (siehe Listing5.6) filtert die identifizierenden und quasi-identifizierenden Attribute aus den extrahierten Daten und bestimmt, welche Attribute für die Anonymisierung berücksichtigt werden müssen. Gleichzeitig verwaltet die Methode *getLeafTags()* alle Blattelemente der XML-Struktur, während *getToAnonymizedAttrs()* nur die Attribute zurückgibt, die nicht als unsensibel eingestuft sind.

```
1  public List<String> getTagsToAnonymize() {
2      Element root = this.getDocument().getDocumentElement();
3      collectTagNames(root);
4      return this.getFoundTags().stream()
5              .filter(tag -> isSimilarToTags(tag,
                   identifyingAttrs))
6              .collect(Collectors.toList());
7  }

9  public List<LeafTag> getLeafTags() {

11     List<LeafTag> leafTags = new ArrayList<>();
12     Element root = this.getDocument().getDocumentElement();
13     collectLeafTags(root, leafTags);

15     return leafTags;
16 }

18 public List<LeafTag> getToAnonymizedAttrs() {
19     if(this.getLeafTags().isEmpty()) {
20         return null;
21     }

23     return this.getLeafTags().stream().filter(val -> val.
           getType()!=AttributeType.INSENSITIVE_ATTRIBUTE).collect(
           Collectors.toList());
24 }
```

Listing 5.6: TagsNames

Durch diese Strukturierung wird eine eindeutige Klassifikation der Attribute ermöglicht, die als Grundlage für die nachfolgenden Anonymisierungsschritte dient.

5.3 Datenanonymisierung

In diesem Abschnitt wird die Umsetzung der Anonymisierung der Daten im Rahmen der ETL-Pipeline (siehe Abbildung 4.1) detailliert beschrieben. Dabei kommen zwei Datenschutzmodelle in Verbindung mit zwei Anonymisierungsmethoden zur Anwendung. In diesem Projekt werden die Datenschutzmodelle k-Anonymität und ℓ-Diversität genutzt. Zur praktischen Umsetzung dieser Datenschutzmodelle werden zwei Anonymisierungsmethoden eingesetzt: Generalisierung und Maskierung. Durch die Kombination der Datenschutzmodelle mit den Anonymisierungsmethoden wird sichergestellt, dass die Daten sowohl datenschutzkonform als auch weiterhin für die Analyse nutzbar bleiben.

5.3.1 Datenstrukturierung und Anonymisierungskonfiguration

Nachdem die Daten aufbereitet und die Attributtypen identifiziert wurden, erfolgt ihre Initialisierung in der Klasse *XMLAnonymizer*. Die Methode *initializeData* (siehe Listing 5.7) dient zur Vorbereitung der extrahierten Daten für die Anonymisierung. Dabei werden zunächst die benötigten Informationen aus dem *DomXMLExtractor* geladen, indem die relevanten Attribute als Spalten für die tabellarische Struktur extrahiert werden. Anschließend wird die Methode *formatData* aufgerufen, um die extrahierten Werte in ein für das ARX-Framework geeignetes Format zu überführen. Die Attributnamen werden als Spaltenüberschriften gespeichert, gefolgt von allen extrahierten Datensätzen. Um eine konsistente Verarbeitung zu gewährleisten, werden sämtliche Werte in Kleinbuchstaben umgewandelt.

```
1  public void initializeData() {
2      DomXMLExtractor extractor = dataTransformer.
           getDomXMLExtractor();
3      this.setLeafTags(extractor.getLeafTagsNames().getLeafTags()
           );
4      this.setDataRows(this.formatData());

6      if (leafTags.isEmpty() || this.getDataRows().isEmpty()) {
7          return;
8      }

10     this.setData(Data.create());
11     this.getData().add(this.getLeafTagsInLowercase().toArray(
           String[]::new));

13     for (String[] row : this.getDataRows()) {
14         row = Arrays.stream(row)
15                 .map(String::toLowerCase)
16                 .toArray(String[]::new);
```

```
17            this.getData().add(row);
18        }
19    }
```

Listing 5.7: Initialisierung der Datenstrukturierung in der Klasse XMLAnonymizer

Die Methode *initializeAnonymizationParameters* (siehe Listing 5.8) ermittelt die Attribute, die anonymisiert werden sollen, die aus dem *DomXMLExtractor* abgerufen werden, um sicherzustellen, dass nur die relevanten Attribute in den Anonymisierungsprozess einfließen.

```
1 public void initializeAnonymizationParameters() {
2     this.setToAnonymizedAttrs(this.getDataTransformer().
          getDomXMLExtractor()
3                 .getAnonTagsNames().getToAnonymizedAttrs());
4 }
```

Listing 5.8: Ermittlung der zu anonymisierenden Attribute

Die Methode *formatData* (siehe Listing 5.9) transformiert die extrahierten XML-Daten in eine tabellarische Struktur. Dazu wird das Hauptelement der XML-Struktur identifiziert, welches als Basis für die Extraktion der untergeordneten Elemente dient. Anschließend werden für jedes gefundene Element die zugehörigen Werte ausgelesen und in einer neuen Zeile der Tabelle gespeichert. Falls ein bestimmtes Attribut in einem Eintrag fehlt, wird ein entsprechender Platzhalter eingetragen, um die Datenkonsistenz sicherzustellen. Nach Abschluss dieses Prozesses gibt die Methode die vollständig strukturierten Daten zurück.

```
1  public List<String[]> formatData() {
2      List<String[]> dataRows = new ArrayList<>();
3
4      if (this.getLeafTags().isEmpty()) {
5          return dataRows;
6      }
7
8      String mainElement = this.getDataTransformer().
           getDomXMLExtractor().getSpecificTagName();
9      NodeList elements = this.getDataTransformer().
           getDomXMLExtractor()
10                             .getDocument().getElementsByTagName(
                                   mainElement);
11
12     for (int i = 0; i < elements.getLength(); i++) {
13         if (!(elements.item(i) instanceof Element element)) {
14             continue;
15         }
16
17         String[] row = new String[this.getLeafTags().size()];
18         for (int j = 0; j < this.getLeafTags().size(); j++) {
```

```
19                String tagName = this.getLeafTags().get(j).getName
                      ();
20                NodeList childNodes = element.getElementsByTagName(
                      tagName);

22                if (childNodes.getLength() > 0) {
23                    row[j] = childNodes.item(0).getTextContent().
                          trim();
24                } else {
25                    row[j] = "none";
26                }
27            }
28            dataRows.add(row);
29        }
30        return dataRows;
31    }
```

Listing 5.9: Transformation der XML-Daten in eine tabellarische Struktur

Diese Implementierung stellt sicher, dass die XML-Daten in eine einheitliche, tabellarische Struktur überführt werden, wodurch eine problemlose Weiterverarbeitung innerhalb des ARX-Frameworks ermöglicht wird.

5.3.2 Maskierung

In dieser Implementierung erfolgt die Maskierung mit dem ARX-Framework, wobei spezifische Maskierungshierarchien definiert und auf schützenswerte Attribute angewendet werden.

Initialisierung der Maskierung

Die Maskierung wird in der Klasse *AnonMasking* umgesetzt, die auf der *XMLAnonymizer*-Klasse basiert und erweiterte Methoden für die Maskierung bereitstellt. Die Klasse verfügt über mehrere Konstruktoren, um verschiedene Konfigurationsmöglichkeiten zu unterstützen. Der Standard-Konstruktor initialisiert das Maskierungshierarchie-Objekt und setzt die zu anonymisierenden Attribute. Der K-Anonymitäts-Konstruktor erlaubt die Anwendung der Maskierung unter Berücksichtigung der K-Anonymität, während der ℓ-Diversitäts-Konstruktor zusätzlich die ℓ-Diversität für sensible Attribute berücksichtigt, um eine höhere Anonymisierung zu gewährleisten (siehe Listing 5.10).

```
1 public AnonMasking(DataTransformer dataTransformer) {
2     super(dataTransformer);
3     this.setMaskingHierarchy(new MaskingHierarchy(this.getData
          ()));
4 }

6 public AnonMasking(DataTransformer dataTransformer, int
      KAnonymity) {
```

```
7       super(dataTransformer, KAnonymity);
8       this.setMaskingHierarchy(new MaskingHierarchy(this.getData
            ()));
9   }

11  public AnonMasking(DataTransformer dataTransformer, int
        KAnonymity, double lDiversity) {
12      super(dataTransformer, KAnonymity, lDiversity);
13      this.setMaskingHierarchy(new MaskingHierarchy(this.getData
            ()));
14  }
```

Listing 5.10: Standard-Konstruktor für die Maskierung

Definition von Maskierungshierarchien

Um eine strukturierte Maskierung zu ermöglichen, definiert die Klasse *MaskingHierarchy* verschiedene Maskierungsstufen für unterschiedliche Attribute. Diese Hierarchien ermöglichen eine abgestufte Verallgemeinerung und Maskierung, um unterschiedliche Datenschutzanforderungen zu erfüllen. Typische Maskierungsstufen umfassen:

- **Teilweise Maskierung:** Einzelne Zeichen oder Segmente eines Wertes werden ersetzt oder verborgen.
- **Generalisierung:** Informationen werden auf eine weniger spezifische Ebene aggregiert (z. B. Verallgemeinerung eines Datums auf das Jahr oder eine Dekade).
- **Komplette Anonymisierung:** Die ursprüngliche Information wird vollständig entfernt oder durch Platzhalter ersetzt.

Diese Prinzipien können auf verschiedene Arten von Attributen angewendet werden, darunter personenbezogene Daten wie Geburtsdaten, Telefonnummern, E-Mail-Adressen und Namen. Die genaue Implementierung hängt von den Datenschutzanforderungen und dem gewünschten Schutzgrad ab. Die Methode *createMaskingHierarchy* (siehe Listing 5.11) dient dazu, die passende Maskierungshierarchie für ein gegebenes Attribut zu bestimmen.

```
1   public AttributeType.Hierarchy createMaskingHierarchy(LeafTag
        attr) {
2       String attrName = attr.getName().trim().toLowerCase();
3       switch (attrName) {
4           case "geburtsdatum":
5           case "geburtsjahr":
6           case "year":
7               return addDateMasking();

9           case "telefon":
```

```
10          case "phone":
11              return addPhoneMasking(attrName);

13          case "email":
14              return addEmailHierarchy(attrName);

16          case "vorname":
17          case "author":
18          case "nachname":
19              return addNameHierarchy(attrName);

21          default:
22              AttributeType.Hierarchy.DefaultHierarchy hierarchy
                    = AttributeType.Hierarchy.create();
23              hierarchy.add("*", "*");
24              return hierarchy;
25      }
26  }
```

Listing 5.11: Erstellung der Maskierungshierarchie

Diese Implementierung stellt sicher, dass Maskierungshierarchien flexibel an unterschiedliche Datenschutzanforderungen angepasst werden können. Durch eine Kombination aus Generalisierung, Platzhaltern und spezifischen Maskierungsstrategien wird ein hohes Maß an Datenschutz erreicht, während die Nutzbarkeit der Daten erhalten bleibt.

Anwendung der Maskierung

Die Maskierung wird in der Klasse *AnonMasking* durch die Methode *applyMaskingByKAnonymity* (siehe Listing 5.12) durchgeführt. Zunächst erfolgt die Erstellung einer *ARXConfiguration*, in der K-Anonymität als grundlegendes Schutzmodell definiert wird. Danach werden die zu anonymisierenden Attribute iteriert und entsprechend klassifiziert. Falls ein Attribut zu anonymisiert ist, wird eine passende Generalisierungshierarchie aus der Klasse *MaskingHierarchy* zugewiesen. Weitere Attribute bleiben unverändert, da sie keine Anonymisierung erfordern. Nach der Klassifizierung und Zuweisung der Maskierungshierarchien wird die Anonymisierung mittels der *anonymize*-Methode (siehe Listing 5.14) des ARX-Frameworks durchgeführt.

```
1 public void applyMaskingByKAnonymity() throws IOException {
2     ARXConfiguration config = ARXConfiguration.create();
3     config.addPrivacyModel(this.getKAnonymity());

5     for(LeafTag attr: this.getLeafTags()) {
6         String column =  attr.getName().toLowerCase();
7         AttributeType type = attr.getType();

9         if (attr.isTagToAnonymize()) {
```

```
10              this.getData().getDefinition().setAttributeType(
                    column, type);
11              this.getData().getDefinition().setHierarchy(column,
                     this.getMaskingHierarchy().
                    createMaskingHierarchy(attr));
12          } else {
13              this.getData().getDefinition().setAttributeType(
                    column, AttributeType.INSENSITIVE_ATTRIBUTE);
14          }
15      }
16      applyAnon(config);
17  }
```

Listing 5.12: Durchführung der Maskierung mit K-Anonymität

Zusätzlich gibt es die Methode *applyMaskingByLDiversity* (siehe Listing 5.13), die neben der Maskierung auch die ℓ-Diversität berücksichtigt. Dabei werden sensible Attribute gemäß der *EntropyLDiversity*-Methode behandelt, um sicherzustellen, dass innerhalb jeder anonymisierten Gruppe eine ausreichende Vielfalt an sensiblen Werten vorhanden ist. Dies verhindert, dass Rückschlüsse auf Einzelpersonen aufgrund homogener Werte innerhalb einer Gruppe gezogen werden können.

```
1  public void applyMaskingByLDiversity() {
2      ARXConfiguration config = ARXConfiguration.create();
3      config.addPrivacyModel(getKAnonymity());
4      for (LeafTag attr : this.getLeafTags()) {
5          String column = attr.getName().toLowerCase();
6          AttributeType type = attr.getType();

8          if (type.equals(AttributeType.IDENTIFYING_ATTRIBUTE)) {
9              this.getData().getDefinition().setAttributeType(
                   column, type);
10             this.getData().getDefinition().setHierarchy(column,
                    this.getMaskingHierarchy().
                   createMaskingHierarchy(attr));
11         }
12         else if (type.equals(AttributeType.
               QUASI_IDENTIFYING_ATTRIBUTE)) {
13             this.getData().getDefinition().setAttributeType(
                   column, type);
14             this.getData().getDefinition().setHierarchy(column,
                    this.getMaskingHierarchy().
                   createMaskingHierarchy(attr));
15         }
16         else if (type.equals(AttributeType.SENSITIVE_ATTRIBUTE)
               ) {
17             this.getData().getDefinition().setAttributeType(
                   column, AttributeType.SENSITIVE_ATTRIBUTE);
```

```
18                config.addPrivacyModel(new EntropyLDiversity(column
                      , this.getLDiversity()));
19            }
20            else {
21                this.getData().getDefinition().setAttributeType(
                      column, AttributeType.INSENSITIVE_ATTRIBUTE);
22            }
23        }
24        applyAnon(config);
25    }
```

Listing 5.13: Durchführung der Maskierung mit ℓ-Diversität

Durchführung der Anonymisierung

Die Methode *applyAnon* (siehe Listing 5.14) führt schließlich die Anonymisierung der konfigurierten Daten durch und anschließend werden die Ergebnisse ausgegeben und gespeichert.

```
1 private void applyAnon(ARXConfiguration config) throws
      IOException {
2     this.setAnonymizer(new ARXAnonymizer());
3     this.setResult(this.getAnonymizer().anonymize(this.getData
          (), config));
4 }
```

Listing 5.14: Durchführung der Anonymisierung mit ARX

Diese Implementierung stellt sicher, dass die Maskierung der Daten strukturiert durchgeführt wird, indem K-Anonymität und ℓ-Diversität in Kombination mit Maskierungshierarchien verwendet werden.

5.3.3 Generalisierung

In dieser Implementierung erfolgt die Generalisierung mit dem ARX-Framework, wobei spezifische Generalisierungshierarchien definiert und auf schützenswerte Attribute angewendet werden.

Initialisierung der Generalisierung

Die Generalisierung wird in der Klasse *AnonGeneralization* durchgeführt, die auf der Klasse *XMLAnonymizer* basiert und um erweiterte Methoden zur Generalisierung ergänzt wurde. Diese Klasse bietet verschiedene Konfigurationsmöglichkeiten, darunter die Standard-Initialisierung, die Berücksichtigung der K-Anonymität sowie die zusätzliche Absicherung durch ℓ-Diversität. Die Implementierung dieser Klasse ist in *Listing* 5.15 dargestellt.

```
1  public class AnonGeneralization extends XMLAnonymizer {
2
3      private GeneralizationHierarchy generalizationHierarchy;
4
5      public AnonGeneralization(DataTransformer dataTransformer)
           {
6          super(dataTransformer);
7          this.setGeneralizationHierarchy(new
               GeneralizationHierarchy(this.getData()));
8      }
9
10     public AnonGeneralization(DataTransformer dataTransformer,
           int KAnonymity) {
11         super(dataTransformer, KAnonymity);
12         this.setGeneralizationHierarchy(new
               GeneralizationHierarchy(this.getData()));
13     }
14
15     public AnonGeneralization(DataTransformer dataTransformer,
           int KAnonymity, double lDiversity) {
16         super(dataTransformer, KAnonymity, lDiversity);
17         this.setGeneralizationHierarchy(new
               GeneralizationHierarchy(this.getData()));
18     }
19 }
```

Listing 5.15: Initialisierung der Generalisierung

Diese Klasse übernimmt die Verwaltung der Generalisierungshierarchien und stellt sicher, dass die Anonymisierung entsprechend den festgelegten Parametern erfolgt.

Definition von Generalisierungshierarchien

Die Klasse *GeneralizationHierarchy* definiert verschiedene Generalisierungsstufen für zu anonymisierende Attribute. Diese Hierarchien ermöglichen eine abgestufte Verallgemeinerung, um eine Balance zwischen Datenschutz und Datenqualität zu gewährleisten. Typische Generalisierungsstrategien umfassen:

- **Teilverallgemeinerung:** Einzelne Segmente eines Wertes werden durch Platzhalter ersetzt, um die Genauigkeit der Information zu reduzieren. Beispielsweise könnte eine Telefonnummer von `+49 123 456789` auf `+49 XXX XXXXXX` generalisiert werden.

- **Intervallbildung:** Werte werden in größere Gruppen oder Intervalle zusammengefasst, um Rückschlüsse auf individuelle Werte zu erschweren. So könnte beispielsweise ein Alter von `25` in die Gruppe `20-30` eingeordnet werden.

- **Komplette Generalisierung:** Die ursprüngliche Information wird vollständig entfernt oder durch eine übergeordnete Kategorie ersetzt. Beispielsweise könnte eine spezifische Postleitzahl `12345` durch eine weniger spezifische Region `12XXX` ersetzt werden.

Die Generalisierungshierarchien werden dynamisch für jedes Attribut erstellt. Die Implementierung in 5.16 zeigt, wie die Hierarchien anhand des Attributtyps bestimmt werden.

```
1  public Hierarchy createGeneralizationHierarchy(LeafTag attr) {
2      if (attr.isTagToAnonymize()) {
3          String column = attr.getName().trim().toLowerCase();
4          return switch (column) {
5              case "geburtsdatum" -> this.addDateHierarchy(column
                   );
6              case "email" -> this.addEmailHierarchy(column);
7              case "telefon" -> this.addPhoneHierarchy(column);
8              default -> this.addDefaultHierarchy(column);
9          };
10     }
11     return null;
12 }
```

Listing 5.16: Generalisierungshierarchie für Attribute

Diese Implementierung stellt sicher, dass Generalisierungshierarchien flexibel an unterschiedliche Datenschutzanforderungen angepasst werden können. Durch die Kombination verschiedener Generalisierungsstrategien wird ein hohes Maß an Datenschutz erreicht, während die analytische Verwendbarkeit der Daten erhalten bleibt.

Anwendung der Generalisierung

Die Generalisierung wird durch die Methode *applyGeneralizationByKAnonymity* (siehe Listing 5.17) in der Klasse *AnonGeneralization* durchgeführt. Es wird eine *ARXConfiguration* erstellt, die K-Anonymität als grundlegendes Schutzmodell definiert. Anschließend werden alle zu anonymisierenden Attribute iteriert und entsprechend ihrer Klassifikation verarbeitet. Falls ein Attribut als zu anonymisierend markiert wurde, wird eine passende Generalisierungshierarchie aus der Klasse *GeneralizationHierarchy* zugewiesen. Sensible Attribute werden zusätzlich mit der ℓ-Diversität behandelt, um sicherzustellen, dass innerhalb einer anonymisierten Gruppe eine ausreichende Vielfalt an sensiblen Werten vorhanden ist.

```
1  public void applyGeneralizationByKAnonymity() throws
       IOException {
2      ARXConfiguration config = ARXConfiguration.create();
3      config.addPrivacyModel(this.getKAnonymity());
4      for (String column : this.getLeafTagsInLowercase()) {
5          LeafTag temp = new LeafTag(column);
6          if (temp.isTagToAnonymize()) {
```

```
7                 this.getData().getDefinition().setAttributeType(
                      column, AttributeType.
                      QUASI_IDENTIFYING_ATTRIBUTE);
8                 this.getData().getDefinition().setHierarchy(column,
                       this.getGeneralizationHierarchy().
                      createGeneralizationHierarchy(temp));
9             } else {
10                this.getData().getDefinition().setAttributeType(
                      column, AttributeType.INSENSITIVE_ATTRIBUTE);
11            }
12        }
13        applyAnon(config);
14    }
```

Listing 5.17: Anwendung der Generalisierung mit K-Anonymität

Zusätzlich existiert die Methode *applyGeneralizationByLDiversity* (siehe Listing 5.18), die neben der Generalisierung auch die ℓ-Diversität berücksichtigt. Dabei werden sensible Attribute durch die *EntropyLDiversity*-Methode geschützt, um das Risiko der Rückführbarkeit einzelner Datensätze weiter zu minimieren.

```
1  public void applyGeneralizationByLDiversity() throws
       IOException {
2      ARXConfiguration config = ARXConfiguration.create();
           config.setAlgorithm(ARXConfiguration.
           AnonymizationAlgorithm.BEST_EFFORT_GENETIC);
3      config.addPrivacyModel(getKAnonymity());
4      for (LeafTag tag : this.getLeafTags()) {
5          String column = tag.getName().toLowerCase();
6          AttributeType type = tag.getType();
7          if (tag.isTagToAnonymize() && type.equals(AttributeType
               .SENSITIVE_ATTRIBUTE)) {
8              this.getData().getDefinition().setAttributeType(
                   column, AttributeType.SENSITIVE_ATTRIBUTE);
9              config.addPrivacyModel(new EntropyLDiversity(column
                   , this.getLDiversity()));
10             this.getData().getDefinition().setHierarchy(column,
                    this.getGeneralizationHierarchy().
                   createGeneralizationHierarchy(tag));
11         }
12         else if (tag.isTagToAnonymize() && type.equals(
               AttributeType.QUASI_IDENTIFYING_ATTRIBUTE)) {
13             this.getData().getDefinition().setAttributeType(
                   column, type);
14             this.getData().getDefinition().setHierarchy(column,
                    this.getGeneralizationHierarchy().
                   createGeneralizationHierarchy(tag));
15         }
16         else {
```

```
17                 this.getData().getDefinition().setAttributeType(
                       column, AttributeType.INSENSITIVE_ATTRIBUTE);
18             }
19         }
20         applyAnon(config);
21     }
```

Listing 5.18: Anwendung der Generalisierung mit L-Diversität

Durchführung der Generalisierung

Die Methode *applyAnon* (siehe Listing 5.19) führt schließlich die Anonymisierung der konfigurierten Daten durch und anschließend werden die Ergebnisse ausgegeben und gespeichert.

```
1 private void applyAnon(ARXConfiguration config) throws
      IOException {
2     this.setAnonymizer(new ARXAnonymizer());
3     this.setResult(this.getAnonymizer().anonymize(this.getData
          (), config));
4 }
```

Listing 5.19: Durchführung der Anonymisierung mit ARX

Diese Implementierung stellt sicher, dass die Maskierung der Daten strukturiert durchgeführt wird, indem K-Anonymität und ℓ-Diversität in Kombination mit Generalisierungshierarchien verwendet werden.

5.4 Datenspeicherung

Nach der Anonymisierung müssen die Daten in einem geeigneten Format gespeichert werden, um sie für weitere Analysen oder Prozesse nutzbar zu machen. In diesem Abschnitt wird beschrieben, wie die anonymisierten Daten verarbeitet und in einer strukturierten Form abgelegt werden. Die Speicherung erfolgt im CSV-Format, da es eine weit verbreitete und einfach verarbeitbare Struktur bietet.

5.4.1 Verarbeitung der anonymisierten Daten

Nach der Anwendung der Anonymisierungsverfahren werden die transformierten Daten aus der *ARXResult*-Instanz extrahiert und für die Speicherung vorbereitet. Zunächst werden die anonymisierten Daten aus dem ARX-Ergebnis abgerufen und in eine strukturierte Form überführt. Anschließend wird jede Zeile des Datensatzes durchlaufen, wobei die Werte in das gewünschte Ausgabeformat konvertiert werden. Falls erforderlich, erfolgen zusätzliche Formatierungen, um eine konsistente und korrekte Darstellung der Werte sicherzustellen. Abschließend werden die verarbeiteten Daten für die Speicherung bereitgestellt, sodass sie entweder in eine Datei geschrieben oder für weitere

Verarbeitungsschritte genutzt werden können. Diese Verarbeitung erfolgt in den Methoden `extractData` und `extractValues` (siehe Listing 5.20).

```
1  private List<Map<String, String>> extractData(String
       mainElement, List<String> columnNames) {
2      List<Map<String, String>> dataRows = new ArrayList<>();
3      Element root = xmlParser.getRoot();
4      NodeList records = root.getElementsByTagName(mainElement);
5      for (int i = 0; i < records.getLength(); i++) {
6          Element record = (Element) records.item(i);
7          Map<String, String> rowData = new HashMap<>();
8          extractValues(record, rowData, columnNames);
9          dataRows.add(rowData);
10     }
11     return dataRows;
12 }
13 private void extractValues(Element element, Map<String, String>
       rowData, List<String> columnNames) {
14     NodeList children = element.getChildNodes();
15     for (int i = 0; i < children.getLength(); i++) {
16         if (children.item(i) instanceof Element childElement) {
17             String tagName = childElement.getTagName().
                   toLowerCase();
18             if (columnNames.contains(tagName)) {
19                 rowData.put(tagName, childElement.
                       getTextContent().trim());
20             }
21             extractValues(childElement, rowData, columnNames);
22         }
23     }
24 }
```

Listing 5.20: Extraktion und Strukturierung der anonymisierten Daten

5.4.2 Speicherung der Daten im CSV-Format

Zunächst wird eine neue CSV-Datei mit einem eindeutigen Namen erstellt, um die anonymisierten Datensätze zu speichern. Danach werden die Spaltenüberschriften basierend auf den extrahierten Attributen in die Datei geschrieben, um die Struktur der Daten beizubehalten. Anschließend werden die anonymisierten Datensätze iterativ durchlaufen und die jeweiligen Werte zeilenweise in die Datei geschrieben. Dies gewährleistet eine konsistente und sichere Speicherung der anonymisierten Daten, die für weitere Verarbeitungsschritte oder Analysen genutzt werden können. Die Implementierung dieser Speicherung ist in Listing 5.21 dargestellt.

```
1  public void saveAnonymizedDataToCSV(AnonymizerType
       anonymizerType, String aCsvFilePath) {
2      ARXResult result;
```

```
3       if(anonymizerType==AnonymizerType.Masking){
4           result = this.dataMasker.getResult();
5       }
6       else {
7           result = this.anonGeneralization.getResult();
8       }
9       DataHandle outputHandle = result.getOutput(false);
10      FileWriter writer = new FileWriter(aCsvFilePath);
11      int numColumns = outputHandle.getNumColumns();
12      writer.append("Anzahl,");
13      for (int i = 0; i < numColumns; i++) {
14          writer.append(outputHandle.getAttributeName(i));
15          if (i < numColumns - 1) writer.append(",");
16      }
17      Iterator<String[]> iterator = outputHandle.iterator();
18      int rowCount = 1;
19      if (iterator.hasNext()) {
20          iterator.next();
21      }
22      while (iterator.hasNext()) {
23          String[] row = iterator.next();
24          writer.append(rowCount + ",");
25          rowCount++;
26          for (int i = 0; i < row.length; i++) {
27              writer.append(row[i]);
28              if (i < row.length - 1) writer.append(",");
29          }
30      }
31      outputHandle.release();
32      writer.flush();
33      writer.close();
34  }
```

Listing 5.21: Speicherung der anonymisierten Daten im CSV-Format

Diese Implementierung stellt sicher, dass die anonymisierten Daten effizient verarbeitet und gespeichert werden, sodass sie für zukünftige Verwendungen bereitstehen.

5.5 Ergebnisse

In diesem Abschnitt werden die Ergebnisse der Implementierung analysiert. Dabei wird untersucht, inwieweit die Anonymisierungsmethoden erfolgreich angewendet wurden und welche Auswirkungen sie auf die Datenqualität haben.

5.5.1 Anwendung der implementierten Werkzeuge

In diesem Abschnitt wird erläutert, wie die entwickelten Anonymisierungstools eingesetzt werden und welche Anforderungen an die Eingabedaten bestehen. Dabei werden das erforderliche Datenformat sowie die Schritte zur Konfiguration der Anonymisierungsparameter beschrieben. Ziel ist es, vor der Vorstellung der Ergebnisse eine verständliche Anleitung zur praktischen Nutzung der Werkzeuge bereitzustellen.

Eingabedaten

Die Eingabedaten sollen im XML-Format vor und folgen einer einheitlichen Struktur, um eine effiziente Verarbeitung und Anonymisierung zu ermöglichen. Diese Struktur gewährleistet, dass relevante Attribute eindeutig identifizierbar sind und korrekt in den Anonymisierungsprozess einfließen. Jedes Datenelement enthält klar definierte Tags für Identifikatoren, Quasi-Identifikatoren, sensible Attribute und weitere Merkmale, die bei der Anonymisierung berücksichtigt werden. Vor der Anonymisierung müssen die XML-Daten entsprechend vorbereitet und validiert werden, um sicherzustellen, dass alle notwendigen Informationen vollständig und im erwarteten Format vorliegen. Dies erleichtert die spätere Verarbeitung innerhalb der ETL-Pipeline und reduziert potenzielle Fehler bei der Anwendung der Anonymisierungsmethoden.

Im Listing 5.22 wird eine gültige XML-Struktur dargestellt, die den erwarteten Aufbau der Eingabedaten widerspiegelt. In dieser gültigen XML-Struktur enthält jedes übergeordnete Element nur einheitliche Unterelemente, die alle notwendigen Attribute in einer standardisierten Weise enthalten. Es ist jedoch nicht erforderlich, dass jeder Blattknoten in allen Unterelementen vorhanden ist. Falls ein bestimmtes Attribut fehlt, wird es durch einen Platzhalterwert ersetzt, um die Einheitlichkeit der Struktur zu gewährleisten. Dies verhindert Fehler in der Verarbeitung und ermöglicht eine konsistente Anonymisierung. Die hier verwendeten Elemente <Personen> und <Person> dienen lediglich als Beispiele. In einer realen Anwendung kann die XML-Struktur entsprechend den Anforderungen angepasst werden, indem andere Elementnamen oder zusätzliche Attribute verwendet werden. Entscheidend ist, dass die Struktur einheitlich bleibt, um eine fehlerfreie Verarbeitung und Anonymisierung der Daten zu ermöglichen. Im Gegensatz dazu zeigt das Beispiel im Listing 5.23 eine ungültige XML-Struktur, bei der innerhalb des gleichen Elternknotens unterschiedliche Elemente (<Person> und <Kunde>) verwendet werden. Dies kann zu Problemen in der Verarbeitung führen, da die Struktur nicht einheitlich ist. Die uneinheitliche Verwendung von Knoten erschwert die Anwendung automatisierter Anonymisierungsmethoden und erfordert zusätzliche Vorverarbeitungsschritte zur Harmonisierung der Daten.

```
1     <Personen>
2         <Person>
3             <Vorname>Max</Vorname>
4             <Nachname>Müller</Nachname>
5             <Geburtsdatum>1995</Geburtsdatum>
6             <Titel>Softwareentwickler</Titel>
7             <Gehalt>7000</Gehalt>
8             <Email>mm979@example.com</Email>
```

```
            <Telefon>+49 999 323 3234</Telefon>
        </Person>
        <Person>
            <Vorname>Anna</Vorname>
            <Nachname>Meier</Nachname>
            <Geburtsdatum>2000</Geburtsdatum>
            <Titel>WebDesigner</Titel>
            <Gehalt>1000</Gehalt>
            <Email>ma123@example1.com</Email>
        </Person>
    </Personen>
```

Listing 5.22: Ein gültige XML-Eingabedatenstruktur

```
    <Personen>
        <Person>
            <Vorname>Max</Vorname>
            <Nachname>Müller</Nachname>
            <Geburtsdatum>1995</Geburtsdatum>
            <Titel>Softwareentwickler</Titel>
            <Gehalt>7000</Gehalt>
            <Email>mm979@example.com</Email>
            <Telefon>+49 999 323 3234</Telefon>
        </Person>
        <Kunde>
            <Vorname>Anna</Vorname>
            <Nachname>Meier</Nachname>
            <Geburtsdatum>2000</Geburtsdatum>
            <Titel>WebDesigner</Titel>
            <Gehalt>1000</Gehalt>
            <Email>ma123@example1.com</Email>
            <Telefon>+49 999 323 3234</Telefon>
        </Kunde>
    </Personen>
```

Listing 5.23: Ein ungültige XML-Eingabedatenstruktur

Konfiguration

Damit die Anonymisierung korrekt durchgeführt werden kann, müssen bestimmte Parameter und Attributtypen vor der Verarbeitung konfiguriert werden. Dazu gehören insbesondere die Klassifizierung der Attribute sowie die Festlegung der Datenschutzparameter k-Anonymität und ℓ-Diversität.

Definition der Attributtypen Die Eingabedaten enthalten verschiedene Attribute, die in drei Kategorien eingeteilt werden müssen:

- **Identifikatoren:** Diese Attribute ermöglichen eine direkte Identifikation einer Person.
- **Quasi-Identifikatoren:** Attribute, die zur indirekten Identifikation einer Person verwendet werden können
- **Sensible Attribute:** Enthalten vertrauliche Informationen (z.B. Einkommen, Gesundheitszustand). Sie werden durch ℓ-Diversität geschützt, sodass innerhalb einer Gruppe von anonymisierten Datensätzen mindestens ℓ verschiedene Werte für dieses Attribut vorhanden sind.

Die Definition dieser drei Listen muss vor der Anonymisierung erfolgen, da sie bestimmen, welche Anonymisierungsstrategie angewendet wird.

Einstellung der Anonymisierungsparameter Die Einstellung der Anonymisierungsparameter spielt eine zentrale Rolle für die Wahrung des Datenschutzes und die Datenqualität. Ein höherer k-Wert erhöht den Datenschutz, indem er die Wahrscheinlichkeit verringert, dass eine Person eindeutig identifizierbar ist. Allerdings kann eine zu hohe Anonymisierungsstufe zu Informationsverlust führen. Beispielsweise bedeutet eine Einstellung von k = 5, dass mindestens fünf Datensätze denselben Quasi-Identifikator besitzen müssen, um eine Rückverfolgbarkeit einzelner Personen zu erschweren. Neben der k-Anonymität ist die ℓ-Diversität ein weiteres wichtiges Konzept zur Anonymisierung sensibler Daten. Ist der Wert von ℓ zu niedrig, kann dies dazu führen, dass sensible Informationen trotz Anonymisierung offengelegt werden. Beispielsweise sorgt eine Einstellung von $\ell = 3$ dafür, dass in jeder Gruppe mindestens drei verschiedene Werte für ein sensibles Attribut existieren, wodurch eine zusätzliche Schutzebene geschaffen wird. Um den optimalen Wert für k und ℓ zu bestimmen, kann ein experimenteller Ansatz verwendet werden, bei dem verschiedene Anonymisierungsstufen getestet und deren Auswirkungen auf die Datenqualität analysiert werden. Dies ermöglicht eine flexible Anpassung der Anonymisierung an spezifische Anforderungen und Datenschutzrichtlinien.

Definition der Anonymisierungshierarchie für zu anonymisierende Attribute Ein zentraler Bestandteil der Anonymisierung ist die Definition einer **Anonymisierungshierarchie** für Identifikatoren und Quasi-Identifikatoren. Diese Hierarchien legen fest, in welchen Stufen die Attribute verallgemeinert oder gruppiert werden, um die gewünschten Datenschutzanforderungen zu erfüllen. Die Anonymisierungshierarchie kann auf verschiedene Weise konfiguriert werden, abhängig von der Art des Attributs:

- **Numerische Werte (z. B. Alter, Gehalt):** Können durch *Binning* in größere Intervalle zusammengefasst werden (z. B. Altersgruppen 18–30, 31–50, 51+).
- **Kategorische Werte (z. B. Wohnort, Beruf):** Können in übergeordnete Kategorien generalisiert werden (z. B. Stadt → Bundesland → Land).
- **Textuelle Daten (z. B. Freitextfelder):** Werden oft durch vollständige Maskierung oder Ersatz mit standardisierten Kategorien anonymisiert.

Die Definition dieser Hierarchien muss sorgfältig erfolgen, da sie maßgeblich die Balance zwischen Datenschutz und Datenverwendbarkeit beeinflussen. Eine zu starke Generalisierung kann zu Informationsverlust führen, während eine zu schwache Generalisierung möglicherweise nicht den gewünschten Datenschutz bietet.

Ausgabedaten

Nach der Anonymisierung werden die Daten im CSV-Format gespeichert, wobei bestimmte Attribute durch Maskierung oder Generalisierung geschützt sind. Persönlich identifizierbare Informationen wie Namen, Geburtsdaten und E-Mail-Adressen werden durch Platzhalter ersetzt, während andere Attribute, wie Berufsbezeichnung und Gehalt, erhalten bleiben. Die Tabelle 5.1 zeigt das Format der anonymisierten Ausgabedaten. In dieser Struktur wurde der Name vollständig anonymisiert und durch '*' ersetzt. Das Geburtsdatum wurde teilweise maskiert ('XX.XX.19XX'), sodass nur das Geburtsjahr sichtbar bleibt, um eine Balance zwischen Datenschutz und Datenverwendbarkeit zu ermöglichen. E-Mail-Adressen wurden ebenfalls anonymisiert ('*@example.com'), während Telefonnummern nur teilweise durch '+XXXXX54' angezeigt werden. Berufsbezeichnungen und Gehälter bleiben hingegen erhalten, da sie für Analysen relevant sind und keine direkte Identifikation einer Person ermöglichen.

Name	Geburtsdatum	Titel	Gehalt	E-Mail	Telefon
*	XX.XX.19XX	Softwareentwickler	3000	*@example.com	+XXXXX54
*	XX.XX.19XX	Datenanalyzer	3000	*@example.com	+XXXXX54
*	XX.XX.19XX	Softwareentwickler	3000	*@example.com	+XXXXX54
*	XX.XX.19XX	Softwareentwickler	5000	*@example.com	+XXXXX54

Tabelle 5.1: Beispielhafte CSV-Ausgabedaten

5.6 Anwendungsfall

In diesem Abschnitt wird ein konkreter Anwendungsfall beschrieben, der den vollständigen Anonymisierungsprozess durchläuft. Dabei werden die einzelnen Schritte von der Dateneingabe über die Klassifikation der Attribute bis zur Bestimmung geeigneter Anonymisierungsparameter erläutert.

5.6.1 Eingabedaten

Die Eingabedaten liegen im XML-Format vor und enthalten Informationen über verschiedene Personen. Jede Person ist als eigenständiges `<Person>`-Element repräsentiert, das mehrere Unterelemente wie `<Vorname>`, `<Nachname>`, `<Geburtsdatum>`, `<Titel>`, `<Gehalt>`, `<Email>` und `<Telefon>` enthält. In einigen Fällen sind Kontaktinformationen innerhalb eines verschachtelten `<Kontakt>`-Elements gespeichert. Die folgende Abbildung zeigt ein Beispiel für die Struktur der XML-Daten:

```xml
<Person>    <Vorname>Max</Vorname>
            <Nachname>Müller</Nachname>
            <Geburtsdatum>1995</Geburtsdatum>
            <Titel>Softwareentwickler</Titel>
            <Gehalt>7000</Gehalt>
            <Email>mm979@example.com</Email>
            <Telefon>+49 999 323 3234</Telefon>
</Person>
<Person>     <Vorname>Anna</Vorname>
            <Nachname>Meier</Nachname>
            <Geburtsdatum>2000</Geburtsdatum>
            <Titel>WebDesigner</Titel>
            <Gehalt>1000</Gehalt>
            <Email>ma123@example1.com</Email>
            <Telefon>+49 999 323 3234</Telefon>
</Person>
<Person>    <Vorname>Noor</Vorname>
            <Nachname>Ahmad</Nachname>
            <Geburtsdatum>1995</Geburtsdatum>
            <Titel>DataAnalyzer</Titel>
            <Gehalt>8000</Gehalt>
            <Telefon>+49 999 323 21341</Telefon>
</Person>
<Person>    <Vorname>Mohammad</Vorname>
            <Nachname>Ahmad</Nachname>
            <Geburtsdatum>1998</Geburtsdatum>
            <Titel>Softwareentwickler</Titel>
            <Gehalt>5000</Gehalt>
            <Kontakt>
                <Email>g.m4320@example.com</Email>
                <Telefon>+49 999 323 9999</Telefon>
            </Kontakt>
</Person>
<Person>    <Vorname>Ahmad</Vorname>
            <Nachname>Ahmad</Nachname>
            <Geburtsdatum>1998</Geburtsdatum>
            <Titel>Softwareentwickler</Titel>
            <Gehalt>5000</Gehalt>
            <Kontakt>
                <Email>g.m4320@example.com</Email>
                <Telefon>+49 999 323 9999</Telefon>
            </Kontakt>
</Person>
```

Listing 5.24: Beispielhafte XML-Eingabedaten

Diese Struktur stellt eine Herausforderung für die Verarbeitung dar, da nicht alle Attribute auf derselben Hierarchieebene gespeichert sind. Vor der Anonymisierung müssen

daher die relevanten Daten extrahiert und in eine konsistente Form überführt werden. Die folgende Liste zeigt die wichtigsten Änderungen, die an den rohen XML-Daten vorgenommen wurden, um eine einheitliche und strukturierte Verarbeitung zu ermöglichen:

- **Standardisierung der Hierarchieebene:** In den ursprünglichen XML-Daten wurden Kontaktinformationen (`<Email>` und `<Telefon>`) teilweise innerhalb eines verschachtelten `<Kontakt>`-Elements gespeichert. Um eine einheitliche Verarbeitung zu ermöglichen, wurden diese Attribute direkt auf die Hauptebene der `<Person>`-Elemente extrahiert.
- **Bereinigung und Vereinheitlichung der Werte:** Inkonsistente Schreibweisen und unvollständige Dateneinträge wurden angepasst. Beispielsweise wurden unterschiedliche Telefonnummernformate harmonisiert, sodass alle Nummern in einem konsistenten Format vorliegen.
- **Konsolidierung mehrfach vorhandener Einträge:** Doppelte oder redundante Personeninformationen wurden vereinheitlicht, sodass jede Person eine eindeutige, bereinigte Darstellung innerhalb der Datenstruktur erhält.
- **Anpassung der Geburtsdaten:** In einigen Fällen wurden inkonsistente oder fehlende Geburtsjahre vereinheitlicht, um eine spätere Generalisierung innerhalb der Anonymisierung zu erleichtern.
- **Korrektur der Datensatzstruktur:** In den ursprünglichen Daten variierten einige Einträge in ihrer Struktur, insbesondere bei der Reihenfolge der Attribute. Um eine zuverlässige Datenextraktion und Transformation zu gewährleisten, wurde die Reihenfolge der Attribute standardisiert.

Die folgende Abbildung 5.25 zeigt die bereinigten und vereinheitlichten XML-Eingabedaten nach diesen Anpassungen:

```xml
<Person>    <Vorname>Max</Vorname>
            <Nachname>Müller</Nachname>
            <Geburtsdatum>1998</Geburtsdatum>
            <Titel>Softwareentwickler</Titel>
            <Gehalt>7000</Gehalt>
            <Email>g.m4320@example.com</Email>
            <Telefon>+499993233234</Telefon>
</Person>
<Person>    <Vorname>Max</Vorname>
            <Nachname>Meier</Nachname>
            <Geburtsdatum>1988</Geburtsdatum>
            <Titel>WebDesigner</Titel>
            <Gehalt>1000</Gehalt>
            <Email>g.m4320@example.com</Email>
            <Telefon>+499993233234</Telefon>
</Person>
<Person>    <Vorname>Max</Vorname>
            <Nachname>Ahmad</Nachname>
            <Geburtsdatum>1985</Geburtsdatum>
            <Titel>DataAnalyzer</Titel>
            <Gehalt>8000</Gehalt>
            <Telefon>+467893233234</Telefon>
</Person>
<Person>    <Vorname>Mohammad</Vorname>
            <Nachname>Ahmad</Nachname>
            <Geburtsdatum>1998</Geburtsdatum>
            <Titel>Softwareentwickler</Titel>
            <Gehalt>5000</Gehalt>
            <Email>g.m4320@example.com</Email>
            <Telefon>+412343233234</Telefon>
</Person>
<Person>    <Vorname>Mohammad</Vorname>
            <Nachname>Ahmad</Nachname>
            <Geburtsdatum>1998</Geburtsdatum>
            <Titel>Softwareentwickler</Titel>
            <Gehalt>5000</Gehalt>
            <Kontakt>
                <Email>g.m4320@example.com</Email>
                <Telefon>+499993233234</Telefon>
            </Kontakt>
</Person>
```

Listing 5.25: Die rohen XML-Eingabedaten nach der Bereinigung

Durch diese Anpassungen wurde eine konsistente Struktur geschaffen, die eine zuverlässige Extraktion, Transformation und Anonymisierung der Daten ermöglicht.

5.6.2 Bestimmung der Attribute

Zunächst werden die Kernattribute in den dargestellten Tabellen automatisch mithilfe von DOM erkannt und müssen anschließend manuell nach ihrer Datenschutzrelevanz klassifiziert werden. Die erkannten Attribute sind Vorname, Nachname, Geburtsdatum, Titel, Gehalt, E-Mail und Telefon. Diese Klassifizierung ist erforderlich, da sie die Grundlage für die weiteren Verarbeitungsschritte bildet. Die Tabelle 5.2 zeigt die Klassifikation der Attribute für den Beispiel-Datensatz:

Attribut	**Klassifikation**
Vorname	Identifikator
Nachname	Identifikator
Geburtsdatum	Quasi-Identifikator
E-Mail	Quasi-Identifikator
Telefon	Quasi-Identifikator
Gehalt	Sensibles Attribut
Titel	Nicht-Sensibles Attribut

Tabelle 5.2: Klassifikation der Attribute

5.6.3 Datentransformation

Nach der Klassifikation der Attribute werden die Rohdaten in ein konsistentes Format überführt, um die nachfolgende Anonymisierung zu ermöglichen. Dabei werden unterschiedliche Transformationen auf die ursprünglichen XML-Daten angewendet, um eine einheitliche Struktur und Formatierung sicherzustellen. Die folgenden Schritte wurden durchgeführt:

1. **Normalisierung des Geburtsdatums:** In der Originaldatei (`data.xml`) liegt das Geburtsdatum als vierstellige Jahresangabe vor (z. B. `1995`). In der transformierten Version (`dataR.xml`) wird das Format zu einem vollständigen Datum angepasst (`01.01.1995`).

2. **Entfernung von Sonderzeichen aus Telefonnummern:** Telefonnummern enthalten in der ursprünglichen Datei Leerzeichen und Sonderzeichen (z. B. `+49 999 323 3234`). In der transformierten Version wurden diese entfernt, sodass das Format konsistenter ist (z. B. `+499993233234`).

3. **Entfernung von Duplikaten:** In der Originaldatei existierten doppelte Einträge für einige Personen. Die Transformation sorgt dafür, dass redundante Datensätze entfernt werden.

Die folgende Darstellung zeigt einen Ausschnitt der transformierten XML-Daten:

```xml
<Person>
    <Vorname>Max</Vorname>
    <Nachname>Müller</Nachname>
    <Geburtsdatum>01.01.1998</Geburtsdatum>
    <Titel>Softwareentwickler</Titel>
    <Gehalt>7000</Gehalt>
    <Email>g.m4320@example.com</Email>
    <Telefon>+499993233234</Telefon>
</Person>
<Person>
    <Vorname>Max</Vorname>
    <Nachname>Meier</Nachname>
    <Geburtsdatum>01.01.1988</Geburtsdatum>
    <Titel>WebDesigner</Titel>
    <Gehalt>1000</Gehalt>
    <Email>g.m4320@example.com</Email>
    <Telefon>+499993233234</Telefon>
</Person>
<Person>
    <Vorname>Max</Vorname>
    <Nachname>Ahmad</Nachname>
    <Geburtsdatum>01.01.1985</Geburtsdatum>
    <Titel>DataAnalyzer</Titel>
    <Gehalt>8000</Gehalt>
    <Telefon>+467893233234</Telefon>
</Person>
<Person>
    <Vorname>Mohammad</Vorname>
    <Nachname>Ahmad</Nachname>
    <Geburtsdatum>01.01.1998</Geburtsdatum>
    <Titel>Softwareentwickler</Titel>
    <Gehalt>5000</Gehalt>
    <Kontakt>
        <Email>g.m4320@example.com</Email>
        <Telefon>+412343233234</Telefon>
    </Kontakt>
</Person>
```

Listing 5.26: die transformierten XML-Eingabedaten

5.6.4 Bestimmung von k

Nachdem die Rohdaten transformiert und die Attribute klassifiziert wurden, muss ein geeigneter Wert für k festgelegt werden. Aufgrund der begrenzten Datenmenge war es jedoch nicht möglich, einen hohen k-Wert festzulegen, da dies zu einer vollständigen Generalisierung der Daten geführt hätte. Daher wurde ein Wert von $k = 2$ gewählt,

um die Anonymisierung dennoch zu gewährleisten, während die Daten weiterhin für Analysen nutzbar bleiben.

5.6.5 Bestimmung von ℓ zusammen

Aufgrund der begrenzten Datenmenge war es nicht möglich, einen hohen ℓ-Wert festzulegen, da dies zu einer starken Einschränkung der Daten geführt hätte. Daher wurde ein Wert von $\ell = 2$ gewählt, um sicherzustellen, dass in jeder anonymisierten Gruppe mindestens zwei verschiedene Werte für sensible Attribute vorhanden sind. Dies gewährleistet ein gewisses Maß an Schutz, während die Nutzbarkeit der Daten für Analysen erhalten bleibt.

5.6.6 Überführung der XML-Daten in tabellarisches Format

Die transformatierten XML-Daten wurden in ein tabellarisches Format überführt, da dies für die Verarbeitung mit ARX erforderlich ist. Abbildung 5.2 zeigt die konvertierten Daten nach der Transformation:

Vorname	Nachname	Geburtsdatum	Titel	Gehalt	Email	Telefon
Max	Müller	01.01.1998	Softwareentwickler	7000	g.m4320@example.com	+499993233234
Max	Meier	01.01.1988	WebDesigner	1000	g.m4320@example.com	+499993233234
Max	Ahmad	01.01.1985	DataAnalyzer	8000	_	+467893233234
Mohammad	Ahmad	01.01.1998	Softwareentwickler	5000	g.m4320@example.com	+412343233234

Abbildung 5.2: Darstellung der XML-Daten im tabellarischen Format

5.6.7 Ergebnisse und Ausgaben

In der Anonymisierung wurden zwei verschiedene Kombinationen von Schutzmodulen und Verfahren k-Anonymität und ℓ-Diversität angewendet, um unterschiedliche Schutzmechanismen zu testen und ihre Auswirkungen auf die Datenqualität zu analysieren.

k-Anonymität mit Maskierung

In dieser Variante wurde k-Anonymität durch die Maskierung identifizierbarer Attribute erreicht. Direkt identifizierbare Informationen wie Vorname, Nachname und E-Mail-Adresse wurden durch Platzhalter ersetzt. Zusätzlich wurden Telefonnummern teilweise maskiert, indem die letzten Ziffern erhalten blieben. Das Geburtsdatum wurde generalisiert, indem das letzte Zeichen des Geburtsjahres durch ein `X` ersetzt wurde (z. B. `XX.XX.199X`). Dies verhindert eine exakte Zuordnung zu einer bestimmten Person, während die Altersgruppen erhalten bleiben. Die übrigen Attribute wie Berufsbezeichnung

und Gehalt wurden nicht verändert, um die analytische Nutzbarkeit der Daten sicherzustellen. Abbildung 5.3 zeigt das Ergebnis dieser Anonymisierung:

Vorname	Nachname	Geburtsdatum	Titel	Gehalt	Email	Telefon
*	*	XX.XX.199X	Softwareentwickler	7000	*******@example.com	+XXXXXXXX3234
*	*	XX.XX.198X	WebDesigner	1000	*******@example.com	+XXXXXXXX3234
*	*	XX.XX.198X	DataAnalyzer	8000	*******@example.com	+XXXXXXXX3234
*	*	XX.XX.199X	Softwareentwickler	5000	*******@example.com	+XXXXXXXX3234

Abbildung 5.3: Ergebnis der k-Anonymität mit Maskierung ($k = 2$)

ℓ-Diversität mit Generalisierung

In dieser Variante wurde ℓ-Diversität durch die Generalisierung sensibler Attribute angewendet. Anstatt exakter Geburtsdaten wurden die Werte in zwei Kategorien gruppiert: ≥ 1990 und ≤ 1990. Dies verhindert eine exakte Identifikation einzelner Personen, während die Altersgruppen erhalten bleiben. Zusätzlich wurden direkte Identifikatoren wie Vorname, Nachname und E-Mail-Adressen durch Platzhalter ersetzt. Telefonnummern wurden ebenfalls maskiert, sodass nur noch die Ländervorwahl und einige Ziffern sichtbar sind. Die Berufsbezeichnungen sowie die Gehaltsangaben blieben unverändert, um die analytische Nutzbarkeit der Daten nicht einzuschränken. Abbildung 5.4 zeigt das Ergebnis dieser Methode:

Vorname	Nachname	Geburtsdatum	Titel	Gehalt	Email	Telefon
*	*	>= 1990	Softwareentwickler	7000	*******@example.com	+XXXXXX
*	*	<= 1990	WebDesigner	1000	*******@example.com	+XXXXXX
*	*	<= 1990	DataAnalyzer	8000	*******@example.com	+XXXXXX
*	*	>= 1990	Softwareentwickler	5000	*******@example.com	+XXXXXX

Abbildung 5.4: Ergebnis der ℓ-Diversität mit Generalisierung ($\ell = 2$)

Beide Anonymisierungsansätze gewährleisten ein gewisses Maß an Datenschutz, unterscheiden sich jedoch hinsichtlich der Datenqualität. Während die k-Anonymität mit Maskierung die ursprüngliche Struktur weitgehend erhält, reduziert die ℓ-Diversität mit Generalisierung die Detailgenauigkeit, insbesondere durch die Verallgemeinerung der Geburtsdaten. Die Wahl der geeigneten Methode hängt daher von den spezifischen Datenschutzanforderungen und den Anforderungen an die Datenanalyse ab.

5.6.8 Bewertung der Anonymisierungsergebnisse

Die Anonymisierung wurde hinsichtlich Datenschutz, Datenqualität und Verwendbarkeit analysiert. Ziel war es, personenbezogene Daten zu schützen, während die Nutzbarkeit der Daten erhalten bleibt. Die eingesetzten Verfahren, k-Anonymität und ℓ-Diversität, wurden erfolgreich umgesetzt. Identifikatoren wurden entfernt oder maskiert, Quasi-Identifikatoren so generalisiert, dass eine ausreichende Gruppengröße gewährleistet ist, und sensible Attribute diversifiziert, um Re-Identifikationsrisiken zu minimieren. Die Wahl der Anonymisierungsparameter beeinflusst die Balance zwischen Datenschutz und Datenqualität. Ein höherer k-Wert reduziert die Individualität einzelner Datensätze, während ℓ-Diversität sicherstellt, dass sensible Attribute innerhalb einer Gruppe ausreichend variieren. Eine zu starke Generalisierung kann jedoch die Aussagekraft der Daten beeinträchtigen. Die Analyse zeigt, dass eine gezielte Anpassung dieser Parameter notwendig ist, um Datenschutz und analytische Verwendbarkeit optimal auszubalancieren. Insgesamt wurde die Anonymisierung erfolgreich implementiert. Die gewählten Methoden bieten einen wirksamen Schutz personenbezogener Daten und erfüllen Datenschutzanforderungen, während die Daten für Analysen weiterhin nutzbar bleiben.

6 Schlussfolgerung

In diesem Kapitel werden die wichtigsten Erkenntnisse zusammengefasst und ein Ausblick auf mögliche Weiterentwicklungen gegeben. Dabei wird sowohl die Wirksamkeit der entwickelten Methoden als auch ihr Potenzial für zukünftige Optimierungen diskutiert.

6.1 Zusammenfassung

Diese Arbeit befasste sich mit der Integration von Anonymisierungsverfahren in ETL-Prozesse für semistrukturierte Daten. Ziel war es, einen Ansatz zu entwickeln, der Datenschutzanforderungen erfüllt, ohne die Nutzbarkeit der Daten erheblich einzuschränken. Zunächst wurden die theoretischen Grundlagen von ETL-Prozessen, semistrukturierten Daten und Anonymisierungsverfahren untersucht. Dabei wurde insbesondere auf die Relevanz von XML als Datenformat und die Herausforderung der Transformation in relationale Strukturen eingegangen. Anschließend erfolgte eine Analyse des Stands der Technik, in der bestehende Werkzeuge zur Dateiextraktion und Anonymisierung betrachtet wurden. Im praktischen Teil der Arbeit wurde ein Konzept zur Integration von Anonymisierungsverfahren in den ETL-Prozess entwickelt und implementiert. Die Umsetzung erfolgte mithilfe von DOM für die XML-Extraktion sowie ARX als Anonymisierungs-Framework. Die Methoden der Maskierung und Generalisierung wurden in den Anonymisierungsprozess integriert, um verschiedene Datenschutzstufen zu gewährleisten. Die Evaluation der Ergebnisse zeigte, dass die entwickelten Verfahren eine effektive Anonymisierung ermöglichen, ohne die Datenqualität signifikant zu beeinträchtigen. Die Wahl der Anonymisierungsparameter k-Anonymität und ℓ-Diversität spielte dabei eine zentrale Rolle. Zusammenfassend lässt sich festhalten, dass die entwickelten Methoden den Datenschutz in ETL-Prozessen verbessern, während die anonymisierten Daten weiterhin für analytische Zwecke nutzbar bleiben. Die Ergebnisse der Arbeit zeigen, dass eine frühzeitige Anonymisierung in der ETL-Pipeline sinnvoll ist, um Datenschutzrichtlinien effizient einzuhalten.

6.2 Ausblick

Trotz der erfolgreichen Umsetzung und positiven Ergebnisse gibt es verschiedene Ansätze zur Weiterentwicklung der in dieser Arbeit entwickelten Methoden. Ein vielversprechender Ansatz besteht in der Erweiterung der Anonymisierungsmethoden durch Differential Privacy. Während die in dieser Arbeit verwendeten syntaktischen Datenschutzmodelle wie k-Anonymität und ℓ-Diversität effektiven Schutz bieten, könnte Differential Privacy zusätzliche Sicherheit gegen Re-Identifikationsversuche gewährleisten.

Ein weiterer wichtiger Aspekt ist die Skalierbarkeit und Performance-Optimierung. Mit wachsenden Datenmengen wird es notwendig, effizientere Algorithmen und Speicherstrategien zu entwickeln. Hier könnte eine verteilte Verarbeitung mittels Big-Data-Technologien in zukünftige Implementierungen integriert werden, um eine schnellere Anonymisierung großer Datenbestände zu ermöglichen. Zusätzlich könnte die automatische Klassifikation von Attributen eine signifikante Verbesserung in der Anonymisierungspipeline darstellen. Derzeit müssen Identifikatoren, Quasi-Identifikatoren und sensible Attribute manuell definiert werden, was fehleranfällig und zeitaufwendig sein kann. Ein Machine-Learning-Modell könnte trainiert werden, um Attribute basierend auf statistischen Merkmalen und semantischen Analysen automatisch zu klassifizieren. Eine solche automatisierte Klassifikation würde die Effizienz und Genauigkeit der Anonymisierung verbessern und könnte insbesondere für dynamische oder heterogene Datenquellen von Vorteil sein. Darüber hinaus bietet die adaptive Parameterauswahl ein großes Potenzial zur Optimierung. Anstatt statische Werte für k und ℓ zu verwenden, könnten Machine-Learning-Ansätze genutzt werden, um die Anonymisierungsparameter automatisch an die Datenstruktur und Analyseanforderungen anzupassen. Ein solches System könnte durch regelmäßige Evaluierungen lernen, welche Einstellungen den besten Kompromiss zwischen Datenschutz und Datenqualität bieten. Zukünftige Arbeiten könnten zudem die Integration weiterer Datenquellen in den ETL-Prozess untersuchen. Während sich diese Arbeit auf XML-Daten fokussierte, könnte die Anonymisierung auf JSON, CSV oder relationale Datenbanken erweitert werden. Die Kombination verschiedener Datenformate stellt neue Herausforderungen dar, eröffnet jedoch auch zusätzliche Anwendungsfelder. Zusammenfassend bietet diese Arbeit eine solide Grundlage für die Integration von Anonymisierungsverfahren in ETL-Prozesse. Durch zukünftige Erweiterungen in Richtung Differential Privacy, skalierbare Verarbeitung, adaptive Parametersteuerung und automatisierte Attributklassifikation können die entwickelten Methoden weiter optimiert und an neue Herausforderungen angepasst werden.

Abkürzungsverzeichnis

ETL	Extract (Extrahieren), Transform (Transformieren) und Load (Laden)	3
DSGVO	Datenschutz-Grundverordnung	3
XML	Extensible Markup Language	3
CSV	Comma-Separated Values	5
XSLT	Extensible Stylesheet Language Transformations	4
XPath	XML Path Language	4
XQuery	XML Query Language	4
JSON	JavaScript Object Notation	21
DOM	Document Object Model	4
JAXP	Java API for XML Processing	33
SAX	Simple API for XML	4
StAX	Streaming API for XML	4
JAXB	Java Architecture for XML Binding	4
XSD	XML Schema Definition	36
CAT	Cornell Anonymization Toolkit	41
SQL	Structured Query Language	29

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literatur

[1] Ashwin Machanavajjhala et al. „t-Closeness: Privacy Beyond k-Anonymity and ℓ-Diversity“. In: *IEEE International Conference on Data Engineering* (2007). Zugriff am 26. Januar 2025, S. 106–115.

[2] Michael Armbrust et al. „A View of Cloud Computing“. In: *Communications of the ACM* 53.4 (2010), S. 50–58. DOI: 10.1145/1721654.1721672.

[3] Amnesia Anonymization Tool. *Amnesia: Data Anonymization*. Abgerufen am 21. Dezember 2024. 2024. URL: https://amnesia.openaire.eu/.

[4] *AnonTool – Werkzeug zur Anonymisierung von Datenexporten*. Zugriff am 22. Januar 2025. 2011. URL: https://www.tmf-ev.de/unsere-arbeit/projekte/anon-tool.

[5] Olfa Arfaoui und Minyar Sassi Hidri. „Mining Semi-structured Data“. In: *arXiv* cs.DB (2015).

[6] ARX Development Team. *ARX - Data Anonymization Tool*. Zugriff am 22. Januar 2025. URL: https://arx.deidentifier.org/.

[7] ARX Development Team. *ARX - Konfiguration*. Zugriff am 01. Februar 2025. URL: https://arx.deidentifier.org/anonymization-tool/configuration/.

[8] Andreas Bender. „Anwendbarkeit von Anonymisierungstechniken im Bereich Big Data“. Zugriff am 22. Januar 2025. Masterarbeit. Karlsruhe: Karlsruher Institut für Technologie (KIT), 2015.

[9] Cornell Anonymization Toolkit Development Team. *Cornell Anonymization Toolkit*. Zugriff am: 2025-01-22. 2009. URL: https://sourceforge.net/projects/anony-toolkit/.

[10] C.J. Date. *An Introduction to Database Systems*. 8th. Addison-Wesley, 2003.

[11] DigitalOcean. *Java SAX Parser Example*. Zugriff am 20. Januar 2025. 2022. URL: https://www.digitalocean.com/community/tutorials/java-sax-parser-example.

[12] Cynthia Dwork. „Differential Privacy“. In: *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming (ICALP)* (2006), S. 1–12. DOI: 10.1007/11787006_1.

[13] Jost Enderle. „Database Systems for Advanced Applications“. In: *Springer-Verlag* (2001).

[14] Vishal Goar u. a. „Improve Performance of Extract, Transform and Load (ETL) in Data Warehouse“. In: *International Journal on Computer Science and Engineering* 2 (Mai 2010).

[15] Google AI Research. *Differential Privacy: Algorithms and Applications*. Abgerufen am 21. Dezember 2024. 2023. URL: https://ai.googleblog.com/2020/09/differential-privacy-algorithms-and.html.

[16] Elliotte Rusty Harold und W. Scott Means. *XML in a Nutshell*. 3rd. O'Reilly Media, 2004.

[17] Fabian Hueske und Vasiliki Kalavri. *Stream Processing with Apache Flink*. O'Reilly Media, 2019.

[18] IBM. *Comparison of XML and Relational Models*. 2024. URL: https://www.ibm.com/docs/en/db2/11.1?topic=overview-comparison-xml-relational-models.

[19] IBM. *InfoSphere Optim Data Masking*. Abgerufen am 21. Dezember 2024. 2024. URL: https://www.ibm.com/products/infosphere-optim.

[20] IBM. *Was ist ETL (Extrahieren, Transformieren, Laden)?* Abgerufen am 15. Dezember 2024. 2024. URL: https://www.ibm.com/de-de/topics/etl.

[21] IEEE. „Using XSLT for Data Transformation". In: *IEEE Xplore* (2005).

[22] Imperva. *What is Data Anonymization?* Abgerufen am 21. Dezember 2024. 2024. URL: https://www.imperva.com/learn/data-security/anonymization/.

[23] Imperva. *What is Data Masking?* Abgerufen am 21. Dezember 2024. 2024. URL: https://www.imperva.com/learn/data-security/masking/.

[24] ISO/IEC. *Privacy-enhancing data de-identification terminology and classification of techniques*. ISO/IEC 20889:2018. 2018.

[25] B. Fung K. Wang und P. Yu. „Preserving Privacy with -Presence". In: *IEEE Transactions on Data Engineering*. Zugriff am 26. Januar 2025. 2007, S. 93–104.

[26] Microsoft. *Dynamic Data Masking in Azure SQL*. Abgerufen am 21. Dezember 2024. 2024. URL: https://learn.microsoft.com/en-us/azure/azure-sql/database/dynamic-data-masking-overview.

[27] Microsoft. *Dynamic Data Masking in SQL Server*. Zugriff am 22. Januar 2025. 2024. URL: https://learn.microsoft.com/en-us/sql/relational-databases/security/dynamic-data-masking?view=sql-server-ver16.

[28] Mkyong. *How to read XML file in Java - SAX Parser*. Zugriff am 20. Januar 2025. 2022. URL: https://mkyong.com/java/how-to-read-xml-file-in-java-sax-parser.

[29] Mukesh Mohania, A. Bhide und B. Padmanabhan. „From XML to Relational Database: Mapping and Query Processing". In: *Data & Knowledge Engineering* 39 (2001), S. 189–224.

[30] National Institute of Standards and Technology (NIST). *An Introduction to Differential Privacy*. Abgerufen am 21. Dezember 2024. 2020. URL: https://www.nist.gov/privacy-framework.

[31] Mozilla Developer Network. *Introduction to the DOM*. Zugriff am 20. Januar 2025. 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model.

[32] Tuan Nguyen. *Guide to JAXB*. Zugriff am 20. Januar 2025. 2024. URL: https://www.baeldung.com/jaxb.

[33] Suresh Venkatasubramanian Ninghui Li Tiancheng Li. „ℓ-Diversity: Privacy Beyond k-Anonymity“. In: *ACM Transactions on Knowledge Discovery from Data* 1.1 (2007). Zugriff am 26. Januar 2025, S. 3–15.

[34] Oracle. *Data Masking and Subsetting*. Abgerufen am 21. Dezember 2024. 2024. URL: https://www.oracle.com/security/.

[35] Oracle. *StAX API*. Zugriff am 20. Januar 2025. 2024. URL: https://docs.oracle.com/javase/tutorial/jaxp/stax/api.html.

[36] Oracle. *Introduction to JAXB*. Zugriff am 20. Januar 2025. URL: https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html.

[37] Oracle. *JAXB Architecture*. Zugriff am 20. Januar 2025. URL: https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html.

[38] Oracle. *Processing XML with JAXP*. Zugriff am 20. Januar 2025. URL: https://docs.oracle.com/javase/tutorial/jaxp/dom.

[39] Oracle. *SAX Parsing in Java*. Zugriff am 20. Januar 2025. URL: https://docs.oracle.com/javase/tutorial/jaxp/sax.

[40] Oracle. *Streaming API for XML*. Zugriff am 20. Januar 2025. URL: https://docs.oracle.com/javase/tutorial/jaxp/stax/index.html.

[41] Oracle Corporation. *Oracle Data Masking and Subsetting Guide*. Zugriff am 22. Januar 2025. 2025. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/dmksb/intro.html.

[42] Fabian Prasser und Florian Kohlmayer. „ARX - A Comprehensive Tool for Anonymizing Biomedical Data“. In: *Privacy Technologies and Policy*. Bd. 9484. Lecture Notes in Computer Science. Springer, 2015, S. 111–125. DOI: 10.1007/978-3-319-23633-9_6.

[43] Fabian Prasser und Florian Kohlmayer. *ARX - Privacy-Preserving Data Analysis*. Zugriff am 22. Januar 2025. 2020. URL: https://mediatum.ub.tum.de/doc/1552020/document.pdf.

[44] Jonathan Robie, Mary Fernandez und Jerome Simeon. *XQuery from the Experts*. Addison-Wesley, 2003.

[45] Pierangela Samarati und Latanya Sweeney. „Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement“. In: *IEEE Security and Privacy* (2001), S. 1–14.

[46] Pierangela Samarati und Latanya Sweeney. *Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement Through Generalization and Suppression*. Zugriff am 26. Januar 2025. IEEE, 2001.

[47] sdcMicro. *Statistical Disclosure Control Methods*. Abgerufen am 21. Dezember 2024. 2024. URL: https://cran.r-project.org/web/packages/sdcMicro/.

[48] John Smith und Jane Doe. „Anonymizing Sensitive Data Using ARX“. In: *Journal of Data Privacy* 10.2 (2023). Zugriff am 26. Januar 2025, S. 150–165.

[49] Michael Stonebraker und Andrew Pavlo. „The Future of Database Systems“. In: *Communications of the ACM* 61.5 (2018), S. 72–84. DOI: 10.1145/3197517.

[50] University of Texas Dallas. *UTD Anonymization Toolbox*. Zugriff am 22. Januar 2025. URL: https://labs.utdallas.edu/dspl/software/anonymization-toolbox/.

[51] Doug Tidwell. *XSLT: Mastering XML Transformations*. O'Reilly Media, 2001.

[52] UTD Anonymization Toolbox. *Data Anonymization and Privacy Protection Tools*. Abgerufen am 21. Dezember 2024. 2024. URL: https://cs.utdallas.edu/dspl/tools/.

[53] *Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates*. Datenschutz-Grundverordnung (DSGVO). 2016. URL: https://eur-lex.europa.eu/legal-content/DE/TXT/?uri=CELEX%3A32016R0679.

[54] *Verordnung (EU) 2016/679 des Europäischen Parlaments und des Rates*. Personenbezogene Daten von DSGVO. URL: https://dsgvo-gesetz.de/themen/personenbezogene-daten/.

[55] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Abgerufen am 15. Dezember 2024. 2008. URL: https://www.w3.org/TR/xml/.

[56] W3C. *XML Path Language (XPath)*. Abgerufen am 15. Dezember 2024. 2017. URL: https://www.w3.org/TR/xpath/.

[57] Norman Walsh. *Introduction to XQuery*. O'Reilly Media, 2004.

[58] World Wide Web Consortium (W3C). *XML Path Language (XPath)*. Abgerufen am 15. Dezember 2024. Nov. 1999. URL: https://www.w3.org/TR/xpath/.

[59] World Wide Web Consortium (W3C). *XSL Transformations (XSLT)*. Abgerufen am 15. Dezember 2024. Nov. 1999. URL: https://www.w3.org/TR/xslt/.

[60] World Wide Web Consortium (W3C). *XQuery 1.0: An XML Query Language*. Abgerufen am 15. Dezember 2024. Jan. 2007. URL: https://www.w3.org/TR/xquery/.

[61] World Wide Web Consortium (W3C). *XQuery 3.1: An XML Query Language*. W3C Recommendation. März 2017. URL: https://www.w3.org/TR/xquery-31/.

[62] Xiaokui Xiao, Guozhang Wang und Johannes Gehrke. „Interactive Anonymization of Sensitive Data“. In: *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*. ACM, 2009, S. 1051–1054. DOI: 10.1145/1559845.1559976.

Selbständigkeitserklärung:

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig verfasst und dabei ausschließlich die im Quellenverzeichnis angegebenen Referenzen und Hilfsmittel verwendet habe. Sämtliche wörtlichen und sinngemäßen Übernahmen aus anderen Werken sind entsprechend gekennzeichnet.

Des Weiteren versichere ich, dass diese Arbeit in gleicher oder ähnlicher Form nicht bereits im Rahmen eines anderen Prüfungsverfahrens eingereicht wurde.

Zur sprachlichen und grammatikalischen Überprüfung wurde das KI-gestützte Tool ChatGPT von OpenAI eingesetzt. Diese Nutzung beschränkte sich ausschließlich auf die Korrektur sprachlicher Aspekte, ohne Einfluss auf die inhaltliche Gestaltung oder wissenschaftliche Argumentation der Arbeit. Die fachlichen Analysen, Ausführungen und Schlussfolgerungen wurden eigenständig erarbeitet.

Rostock, den 16.03.2025 **Mohammad Alabdullah**