BACHELOR-ARBEIT

Einrichten sicherer Datenbankanwendungen mit SQL

29. APRIL 2024

Hasan Korhan Köroglu

Einrichten sicherer Datenbankanwendungen mit SQL

Bachelorarbeit zur Erlangung des akademischen Grades
Bachelor of Science (B.Sc.)
im Studiengang Wirtschaftsinformatik
an der Fakultät für Informatik und Ingenieurswissenschaften
der Technischen Hochschule Köln

vorgelegt von: Hasan Korhan Köroglu
Matrikel-Nr.:
Adresse:

eingereicht bei: Prof. Dr. Birgit Bertelsmeier
Zweitgutachter*in: Prof. Dr. Heide Faeskorn-Woyke

Gummersbach, 29.04.2024

Kurzfassung/Abstract

Diese Arbeit befasst sich mit der immer vorhandenen Gefahr durch Cyberkriminelle auf Anwendungen und Informationssysteme. Die Informationssicherheit ist ein sehr umfangreiches Thema und sieht sich in der heutigen Zeit besonders starken Gefahren ausgesetzt, die durch die voranschreitende Digitalisierung immer gängiger werden. Durch ein gut eingerichtetes Datenbanksystem und einer gut entwickelten Datenbankanwendung soll es Anwendern ermöglicht werden, die Sicherheit dieser zu bewahren. Hierbei spielen Themen wie Authentifizierung, Zugriffskontrolle, PL/SQL Best Practices und Audits sowie Backups eine zentrale Rolle. Das Einrichten interner und externer Schutzmaßnahmen, explizit auch der Schutz vor SQL-Injektionen, sind die in dieser Arbeit zu behandelnden Themen. Durch den korrekten Einsatz der prozeduralen Erweiterung von SQL sollen Schutzmechanismen erforscht und demonstriert werden, die eine Umfangreiche Bandbreite von potenziellen Angriffen abdecken können. Das Verständnis für den Einsatz der korrekten Verfahren soll über eine Methode, genannt Systematisches Risikomanagement, erörtert werden. In diesem Kontext werden Risiken identifiziert, um passende Lösungsvorschläge zu erforschen und zu präsentieren, um im Nachhinein ihren genauen Einsatz zu diskutieren. Durch den Einsatz der vorgestellten Methoden, Konzepte und Modelle soll die Sicherheitslandschaft im Bereich Informations- und Kommunikationstechnik optimiert werden, um einen soliden Schutz gegen präsente Gefahren zu leisten.

Inhalt

Tabellenverzeichnis

Codeverzeichnis

1 Einleitung

Bedrohungen steigen kontinuierlich an. Die Arbeitsweise von Unternehmen hat sich in den vergangenen Jahren vor allem durch die Digitalisierung deutlich verändert. Aufgaben und Prozesse, die einst auf analoge Art ausgeführt wurden, werden immer stärker digitalisiert durchgeführt. Die zunehmende Vernetzung und Globalisierung erlebt einen rasanten Anstieg, vor allem mit der verstärkten Einführung von Künstlicher Intelligenz und auch durch den Ausbruch der COVID-19-Pandemie. Die damit verbundene Umstellung der Arbeitswelt macht Arbeitnehmer sowie Unternehmen heute stärker abhängig von der digitalen Welt als jemals zuvor. Selbst nach dem Ende der Pandemie sind Laut dem statistischen Bundesamt noch viele Arbeitnehmer vom Homeoffice aus tätig: „24,2 % aller Erwerbstätigen in Deutschland waren im Jahr 2022 zumindest gelegentlich im sogenannten Homeoffice“ [110]. In der Zeit vor der Pandemie sah es anders aus: „Gegenüber dem Vor-Corona-Niveau hat sich der Anteil nahezu verdoppelt: 2019 hatten noch 12,8 % der Erwerbstätigen im Homeoffice gearbeitet, im ersten Corona-Jahr 2020 waren es 21,0 %.“ [110].

Gleichzeitig stieg während der Pandemie auch die Cyberkriminalität mit an. Laut Bitkom war der Schaden an Unternehmen durch Cyberkriminalität folgendermaßen einzuordnen: „206 Milliarden Euro Schaden entstehen der deutschen Wirtschaft jährlich durch Diebstahl von IT-Ausrüstung und Daten sowie digitale und analoge Industriespionage und Sabotage. Damit liegt der Schaden zum dritten Mal in Folge über der 200-Milliarden-Euro-Marke (2022: 203 Milliarden Euro, 2021: 223 Milliarden Euro) und pendelt sich auf sehr hohem Niveau ein“ [112]. Demnach kann behauptet werden, dass mit steigender Digitalisierung die Cyberkriminellen bessere Chancen sehen, Systeme zu infiltrieren und an relevante Informationen zu gelangen. Die Informationssicherheit der deutschen Wirtschaft steht derzeit vor einem überwältigenden Angriff durch internationale Cyberkriminelle, und mit der voranschreitenden Digitalisierung ist es nicht unwahrscheinlich, dass die Zahlen noch weiter steigen werden.

1.1 Motivation

Laut aktuellen Umfragen fühlen sich Unternehmen durch Cyberattacken in ihrer Existenz bedroht. Rund 52% aller deutschen Unternehmen behaupten, dass sie Angriffsziel von kriminellen und feindlich gesonnenen Staaten sind [112]. Dabei sollen die Grenzen fließend sein. Gleichzeitig erlebt die künstliche Intelligenz und die damit verbundene Digitalisierung einen starken Zuwachs. Laut Bitkom nutzen bereits 15% der Unternehmen KI, vor einem Jahr waren es nur 9 % [111]. Das Bundeskriminalamt (BKA) stellt

Cybercrime als ein dauerhaftes Problem dar: „Cybercrime ist eines der sich am dynamischsten verändernden Kriminalitätsphänomene“ [15]. Zudem ist es sehr lukrativ. Geldwäsche, das Stehlen sensibler Informationen von Unternehmen und damit verbundene Erpressungen, das Nutzen dieser Informationen für die eigenen wirtschaftlichen Ziele und Technologien. All das sind Beweggründe, die die Kriminellen haben. Dementsprechend wurden auch in den vergangenen Jahren einige erfolgreiche Angriffe ausgeübt [119]. Ein Beispiel ist eine Russische Hackergruppe Lockbit, die in Deutschland im Sommer 2022 40 Terabyte Daten von dem Autozulieferer Continental gestohlen haben und dafür Lösegeld in Höhe von 50 Millionen US-Dollar forderten. Das Unternehmen zahlte den Betrag nicht und nahm den Schaden in Kauf [90].

Ein anderes Beispiel ist der Angriff auf das Versicherungsunternehmen CNA Financial. Im Mai 2021 zahlte das Unternehmen ein Lösegeld in Höhe von 40 Millionen US-Dollar. Dabei stahlen Cyberkriminelle durch den Einsatz von Ransomware sensible Daten in Form von Sozialversicherungsnummern und medizinischen Informationen. CNA Financial sah sich infolgedessen gezwungen, das geforderte Lösegeld zu zahlen, um höhere Schäden zu vermeiden [19]. Das sind nur zwei von etlichen Beispielen der vergangenen Jahre, in denen regelmäßig große Angriffe auf Unternehmen stattfanden. Cyberkriminalität entwickelt sich immer stärker weiter, und die Unternehmen haben Schwierigkeiten, gegen die diversen Angriffe standzuhalten. Das der Staat neue IT-Sicherheitsgesetzte verabschiedet, um dem entgegenzuwirken, scheint nicht auszureichen: Das Bundesamt für Sicherheit in der Informationstechnik hat zuletzt am 07. Mai 2021 ein Gesetz gebilligt, das die Erhöhung der Sicherheit informationstechnischer Systeme gewährleisten soll [14].

Diese Situation hat dazu geführt, dass die Informationssicherheit für Unternehmen, aber auch für privat Personen, wieder einmal viel stärker in den Vordergrund gerückt ist. Aktuelle betriebliche Abläufe verlangen ein viel höheres Maß an Datenschutzstandards, um sich besser vor der Vielzahl an IT-Angriffen zu schützen. Demnach besteht eine besonders hohe Anforderung an die IT-Sicherheit, insbesondere die Informationssicherheit. Es ist von essenzieller Bedeutung, eine umfassende Betrachtung verschiedener Sicherheitsaspekte bei der Einrichtung von Datenbankanwendungen und damit verbunden Datenbanksystemen mit einzubeziehen. Basierend auf die heutige Situation muss sichergestellt werden, dass sensitive Informationen hinsichtlich ihrer Integrität, Vertraulichkeit und Verfügbarkeit geschützt sind. Laut OWASP sind die Top 10 der größten Sicherheitsrisiken für Anwendungen folgende:

1. Broken Access Control
2. Cryptographic Failures
3. Injection

4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Outdated Components
7. Identification and Autehntifaction Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server Side Request Forgery

[76].

Wenn ein genauer Blick auf Datenbankanwendungen geworfen wird, vertreten die vorgestellten Punkte folgende Aspekte, die relevant für die Informationssicherheit sind:

➢Zugriffskontrolle und Verschlüsselungen in Form von der Implementierung strenger Zugriffsrichtlinien und Verschlüsselungs-Algorithmen, um einen hohen internen Schutz zu gewährleisten.

➢Authentifizierungs- und Identitätsmanagement sowie das Einrichten einer sicheren Umgebung, um nur identifizierten Benutzern Zugriffsrechte zu gewähren.

➢Die Implementierung von Audits und Audit-Trails, um ungewöhnliche Aktivitäten zu erkennen und auf potenzielle Bedrohungen rechtzeitig reagieren zu können.

➢Die Implementierung von gut durchdachtem Code zum Schutz vor fremden Angriffen, beispielsweise in Form von SQL-Injektionen.

➢Der Einsatz von Backups zur Datensicherung und Wiederherstellung bei Datenexposition oder Katastrophen.

Die Integration dieser Sicherheitsaspekte in den Entwicklungsprozess von Datenbankanwendungen ist entscheidend, um eine robuste, widerstandsfähige und sichere Datenbankinfrastruktur zu schaffen. Sie sind wesentlicher Bestandteil des IT-Sicherheitsmanagements [44, S.44-47]. Es ist ratsam, regelmäßige Sicherheitsüberprüfungen durchzuführen und die Sicherheitsrichtlinien bei Bedarf zu aktualisieren, sowie Schulungen für Mitarbeiter einzuführen, um auf neue Bedrohungen und Entwicklungen in der Sicherheitslandschaft reagieren zu können.

Laut DB-Engines ist Oracle der Anbieter der beliebtesten Datenbankplattform [24]. Die Prozedurale Erweiterung von SQL, die Oracle Database anbietet, PL/SQL, wird auch primär in dieser Arbeit verwendet werden um mit der IDE von Oracle, dem Oracle SQL

Developer Version 23.1.1, eine Beispiel Datenbank erzeugen, so dass die Umsetzung einiger Aspekte nachvollziehbar dargestellt werden kann.

1.2 Zielsetzung

Diese Arbeit soll sich mit dem Entwickeln einer sicheren Datenbankanwendung befassen und erforschen, mit welchen Methoden Sicherheitsaspekte innerhalb eines Datenbanksystems umgesetzt werden können. Zudem sollen bezüglich der Datenbankanwendung SQL Best Practices betrachtet und ihre Vor- und Nachteile genauer erforscht werden. Diese Aspekte sollen gemeinsam das Thema der Einrichtung von sicheren Datenbankanwendungen mit SQL vertreten. Der genaue Einsatz soll in einer Oracle Umgebung betrachtet werden. Die eingesetzten Konzepte und Methoden können mit einigen Anpassungen aber auch auf anderen Datenbankplattformen eingesetzt werden.

Dabei ist die Arbeit so strukturiert, dass erst Risiken identifiziert werden, die solch eine Datenbankanwendung potenziell gefährden, um daraufhin Methoden vorzustellen, die die identifizierten Risiken minimieren und/oder limitieren können. Nach der Identifikation und der Erforschung von passenden Gegenmaßnahmen ist die Vorgehensweise Folgende: Erst soll die Umgebung vorgestellt werden, in der diese Maßnahmen umgesetzt werden, um anschließend dann damit zu beginnen abzuwägen, welche Maßnahmen gegen welche Bedrohungen wie am effektivsten eingesetzt werden können.

1.3 Aufbau der Arbeit

Die Arbeit ist eingeteilt in insgesamt 6 Kapitel, wobei Kapitel 1 die Einleitung und Kapitel 6 eine abschließende Zusammenfassung mit einem Fazit und einem Ausblick in die Zukunft abbilden soll.

In Kapitel 2 werden Grundlagen vorgestellt und gewisse Begrifflichkeiten erklärt. Es wird veranschaulicht werden, was ein Datenbanksystem überhaupt ist, wie es arbeitet, aus welchen Komponenten es besteht und was es umsetzen soll. Daraufhin wird die deskriptive Programmiersprache SQL genauer angeleuchtet, um ihre Relevanz deutlich zu machen. Abschließend wird die Informationssicherheit im IT-Umfeld definiert, um diesbezüglich Standards und Best Practices vorzustellen.

Kapitel 3 befasst sich dann mit Problemen in Datenbankanwendungen selbst. An dieser Stelle soll der Einsatz der systematischen Risikoanalyse dabei helfen, diese zu

identifizieren. Daraufhin sollen dann Konzepte und Methoden erforscht werden, die die identifizierten Risiken theoretisch bewältigen können.

In Kapitel 4 sollen die prozedurale Erweiterung von SQL und die Zugriffskontrolle vorgestellt werden. Das Kapitel soll dazu dienen, ein besseres Verständnis für die Einsatzmöglichkeiten der Prozeduralen Erweiterung von SQL (in diesem Fall PL/SQL von Oracle) durch die Erforschung der gängigen Funktionalitäten zu entwickeln und die Zugriffssteuerung in diesem Umfeld zu erforschen.

Kapitel 5 wird sich dann mit dem genauen Einsatz von PL/SQL von Oracle Database zum Schutz vor Angriffen befassen. Es sollen interne- und externe-Schutzmaßnahmen erforscht und ihre Einsatzmöglichkeiten, mit Vor- und Nachteilen, vorgestellt werden. Dieses Kapitel soll damit verbunden auch die in Kapitel 3 vorgestellten Konzepte und Methoden mit einbeziehen und ihren Einsatz verdeutlichen.

Folgende Themen werden in den Kapiteln behandelt:

➢ Identitäts- und Authentifikations Management (Kapitel 3 und 5).

➢ Zugriffskontrolle und Rollenkonzepte (Kapitel 3 und 4).

➢ Audit-Trail und Backups (Kapitel 3 und 5).

➢ Verschlüsselung und Hashing (Kapitel 3 und 5).

➢ Einrichten von Verfügbarkeitsmaßnahmen (Kapitel 5).

➢ Integritäts-Maßnahmen (Kapitel 5).

➢ Bewährte Praktiken bei der Programmierung mit prozeduralen Erweiterungen (Kapitel 5).

Die Kapitel sind so aufgebaut, dass sie beginnend bei der Vorstellung der Grundlagen über die Identifizierung der Risiken und der Erforschung von Gegenmaßnahmen bis hin zu dem genauen Einsatz dieser Gegenmaßnahmen einen logischen roten Faden verfolgen soll.

2 Grundlagen

Dieses Kapitel soll dazu dienen, dem Leser eine solide Grundlage zu schaffen, auf die der weitere inhaltliche Verlauf der Arbeit aufbauen kann. Bezogen auf die Fragestellung werden dementsprechend erst einmal Datenbanksysteme vorgestellt und Begrifflichkeiten definiert. Anschließend wird auf die Entwicklung einer Datenbank eingegangen und die Programmiersprache SQL vorgestellt. Zum Ende hin wird das Thema der Informationssicherheit genauer werden, um basierend darauf Standards und Best Practices bezüglich dieser vorzustellen.

2.1 Datenbanksysteme

Ein Datenbanksystem (DBS) ist ein Rechnergestütztes System zur Beschreibung, Speicherung und Abfrage von Daten [66, S.2]. Es besteht aus einer Datenbasis, die in einer oder mehreren Datenbanken zusammengefasst werden kann, sowie einem Datenbankmanagementsystem (Data Base Management System, DBMS) [33, S.25].

Datenbanksysteme (DBS) unterteilen sich in relationale und nicht relationale DBS. In dieser Arbeit werden ausschließlich relationale DBS behandelt. Im relationalen Datenbankansatz wird im Gegensatz zu nicht-relationalen Systemen die Speicherung von Daten in einfachen Tabellen umgesetzt [66, S.6]. Bei Transaktionen vertreten Relationale Datenbanken zusätzlich als Grundlage die ACID-Eigenschaften (Atomacity, Consistency, Isolation, Durability) [115, S.17-18].

Die Datenbanken (DB), die die Datenbasis bereitstellen, enthalten Daten, die durch verschiedene Entitäten kontextualisiert werden. Demnach ist eine Datenbank im relationalen Modell eine strukturierte Sammlung von Daten, die in einer logischen Beziehung stehen und in Tabellen eingeordnet werden, um eine einfache Verwaltung mittels des Datenbankmanagementsystems zu ermöglichen [102, S.3].

Ein Datenbankmanagementsystem (DBMS) ist ein Softwaresystem, mit dessen Hilfe die Daten innerhalb einer Datenbank manipuliert werden können [102, S.3]. Indem auf Benutzeranfragen reagiert wird und Ergebnismengen in Form von Tabellen ausgegeben werden, kann eine übersichtliche Koordination der gespeicherten Daten erfolgen.

Ein Datenbankanwendungsprogramm, oder auch Datenbankanwendung, ist ein anwendungsspezifisches Programm, das auf Daten einer oder mehrerer Datenbanken zugreift, um sie zu verarbeiten und eventuell Änderungen aktualisiert zurückzuschreiben [114,

S.1]. Es wird im Kontext von Webanwendungen eingesetzt, um beispielsweise Benutzerdaten für eine Anmeldung zu speichern oder ähnliche Interaktionen zu ermöglichen. Es hat als Grundlage das DBS, welche diese Interaktion ermöglicht. Das DBS ist besonders gegen fremde oder interne Angriffe zu schützen, um den Zugriff nur autorisierten Anwendern zu erlauben.

Komponenten eines Datenbankverwaltungssystems

Ein DBMS bestehen aus einer Vielzahl von Komponenten, die miteinander interagieren [114, S.29]. Anfragen werden in mehreren Phasen bearbeitet und auf ihre Korrektheit überprüft, um anschließend eine Optimierung und Ausführung zu ermöglichen. Datenbanksysteme basieren auf einer Client/Server-Architektur, wobei der Client mit dem Server über eine SQL-Datenbankschnittstelle kommunizieren kann [114, S.29]. Bei Änderungen sendet der Client die Anfrage in Form von deklarativen SQL-Anweisungen an den Server, der diese auswertet und Ergebnisse in Form von Tabellen wiedergibt.

Physische und logische Unabhängigkeiten

Das Datenbanksystem hat zur Aufgabe, die Datenunabhängigkeit zu gewährleisten. Das ist bedeutsam, damit Sicherheitsmaßnahmen passend eingerichtet werden können.

Die physische Unabhängigkeit in einem Datenbanksystem wird umgesetzt, indem die physische Speicherung der Daten von der Verwaltung dieser mit einem Anwendungsprogramm getrennt wird [39, S.15]. Sie bezieht sich auf die interne Ebene und kann verwendet werden, um DBMS und DB voneinander zu trennen. Dadurch soll eine Änderung der Datenstruktur ermöglicht werden, ohne das damit verbundene Datenbanksystem zu beeinflussen.

Die logische Datenunabhängigkeit erlaubt es, zwischen einer logischen Gesamtstruktur der Datenbank und den anwenderspezifischen Sichten auf die Daten zu unterscheiden [39, S.15]. Sie ist wichtig für die Sicherheit der Datenbank, damit der Entwickler Änderung an der logischen Struktur der Datenbank vornehmen kann, ohne dass das DBMS von diesen betroffen ist.

2.2 Konzipieren von Datenbanken

Ein Datenbanksystem enthält Informationen über Objekte aus der realen Welt. Das relationale Modell besteht aus der Definition dieser Objekte, in Verbindung mit Operationen und Regeln [100, S.19]. Beziehungen und Objekte, die in einem Modell abgebildet werden, werden in relationalen Datenbankmanagementsystemen (RDBMS) zu Tabellen mit Zeilen und Spalten umgewandelt und abgespeichert. Es existieren verschiedene Methoden zum Erstellen von Datenmodellen, an denen sich der Entwickler orientieren kann. Dabei spielen die Entity- und die Referentielle Integrität eine herausragende Rolle.

Entity Integrität

Die Entity Integrität besagt, dass eine Entität eindeutig identifizierbar sein soll [104, S.76]. Das gelingt durch den Einsatz von Primärschlüssel-Attributen. Als Attribut wird ein Eintrag in ein Feld eines Tupels (einer Zeile der Tabelle) bezeichnet, der einen konkreter Attributwert abbildet [98, S.19-20]. Ein Primärschlüssel ist eindeutig identifizierbar und besteht entweder aus einem einzelnen oder mehreren zusammengesetzten Attributen [60, S.81-82]. Die Vorgabe einer eindeutigen Identifizierung mithilfe eines Primärschlüssels wird als Bedingung bei der Entity Integrität vorausgesetzt. Hierzu darf keine Komponente des Primärschlüssels einer Tabelle den NULL-Wert enthalten [104, S.173]. Die Verbindungen zu anderen Entitäten mithilfe von Fremdschlüssel-Beziehungen wird als referentielle Integrität bezeichnet.

Referentielle Integrität

Bei dieser Art der Integrität geht es darum, die Beziehungen zwischen Objekten in einem Datenbanksystem Konsistent zu halten. Die Definition besagt, dass zu jedem Wert eines Fremdschlüssels ein entsprechender Primärschlüssel in einer anderen Tabelle existieren muss [104, S.168-169]. Folgendes Beispiel kann diesbezüglich betrachtet werden:

Es existieren Tabelle T1 und Tabelle T2, wenn T1 ein Fremdschlüssel-Attribut besitzt, muss dieser in T2 ein Primärschlüssel-Attribut sein.

Die Referentielle Integrität hat den Zweck, dass die Konsistenz einer Datenbank bewahrt wird. Es existieren folgende Bedingungen:

1. Jede Entität hat einen eindeutigen Primärschlüssel (Entity-Integrität) [104, S.76].

2. Jeder Wert des Fremdschlüssel-Attributs in T1 muss gleich einem Wert des Primärschlüssel-Attributs in der Tabelle T2 sein [104, S.168].
3. Fremdschlüssel-Attribute dürfen dementsprechend nicht auf Werte geändert werden, die keinen Primärschlüssel-Attributen entsprechen [104, S.168].

Beim Konzipieren einer Datenbank soll bereits bei der Entwicklung darauf geachtet werden, dass die Datenintegrität gewahrt wird, um später in dem Datenbanksystem keine Komplikationen herbeizurufen. Ziel ist an dieser Stelle, dass die Daten, die in eine Tabelle eingetragen werden, in der gleichen Form wieder ausgegeben werden können, um so ihre Konsistenz beizubehalten und später eine widerspruchsfreie Datenbankanwendung zu haben.

2.3 SQL

SQL ist eine standardisierte und deskriptive Programmiersprache, mit deren Hilfe Informationen auf relationale Datenbanken manipuliert und bereitgestellt werden können [9, S.7].

Mit SQL kann über die Datenbankanwendung mithilfe von Programmierschnittstellen (APIs) dem Datenbanksystem mitgeteilt werden, welche Daten abgerufen, aktualisiert, eingefügt oder gelöscht werden sollen, nicht aber wie es zu diesen Daten gelangt. Es ist eine Abfrage- und Manipulationssprache, die Sprachkomponente ist deskriptiv [66, S.7]. Der Einsatz der Sprache erfolgt, indem SQL-Ausdrücke in relationale Algebra umgewandelt werden, die wiederum in prozedurale Anwendungslogik umgewandelt wird [31, S.348]. Wenn SQL eine Anweisung ausführt, muss daher eine Umsetzung in prozedurale Anweisungen erfolgen, was zu einer Kettenreaktion führen kann, die mehrere prozedurale Ereignisse aufruft, nur damit eine einzelne SQL-Ausgabe ausgeführt wird. Relationale Datenbankmanagementsysteme und SQL greifen dementsprechend ineinander. Die Verwaltung bei dem relationalen Datenbankmanagement wird immer mit SQL-Anweisungen umgesetzt. Datenabfrage, Datenmanipulation, Datendefinition und Datensteuerung können in RDBMS nur mithilfe von SQL erfolgen [31, S.348].

Data-Definition-Language (DDL): Befasst sich mit dem Erstellen von Datenbanken, Tabellen und Indizes [55, S.783].

Data-Manipulation-Language (DML): Behandelt alle manipulativen Aktionen: Anlegen, Ändern und Löschen von Datensätzen [55, S.783].

Data-Query-Language (DQL): Befasst sich mit der Abfrage von Daten aus einer Datenbank [46, S.16].

Data-Control-Language (DCL): Fasst das Anlegen von Benutzern und die Vergabe von Zugriffsrechten zusammen [55, S.783].

SQL erlaubt eine direkte Ausführung von Abfragen (Ad-Hoc), um eine schnelle und einfache Bearbeitung von Daten zu ermöglichen [39, S.68]. Wie genau eine Anweisung ausgeführt wird, bestimmt in SQL die Optimierung [9, S.9]. Sie betrachtet SQL-Anweisungen und ermittelt den effizientesten Ausführungspfad. Die Optimierung kann eingestellt werden durch Optimierungshinweise (Optimizer Hints), was zum Themenbereich des SQL-Tunings gehört. Durch den Einsatz von Best Practices in SQL kann der Anwender jedoch selbst Einfluss durch möglichst präzise Ausdrücke auf die Geschwindigkeit der Abfragen nehmen.

Um die Datenbanksicherheit zu gewährleisten, ist das richtige Verwenden von SQL von entscheidender Bedeutung. Aufgrund der deklarativen Eigenschaft haben unterschiedliche Plattformen Erweiterungen in Form von neuen, prozedural orientierten Programmiersprachen bereitgestellt.

2.4 Prozedurale Erweiterungen von SQL

Wenn erweiterte Sicherheitsvorkehrungen oder komplexe Geschäftslogik innerhalb der Datenbank eingerichtet werden soll, ist eine Prozedurale Programmiersprache, bei der Anweisungen vom Entwickler erstellt werden können, von großer Bedeutung. Zu diesem Zweck wurden Prozedurale Erweiterungen von SQL entwickelt. Diese bieten zusätzlich zu den SQL-Abfragen prozedurale Anweisungen an, die das Datenbanksystem funktional erweitern können. Hierbei verwenden verschiedene Plattformen unterschiedliche Programmiersprachen. Dabei sind PL/SQL von Oracle, T-SQL von Microsoft, SQL/PSM für IBM DB2, PL/pgSQL für PostgreSQL und PL/SQL für DB2 von IBM SQL/PSM für MySQL die gängigsten [24].

Informationssicherheit

Das Datenbank-Security-Management befasst sich mit dem Schutz vor unautorisiertem Zugriff auf die in einer Datenbank gespeicherten Daten, um interne Datenexpositionen

und Datenlecks zu vermeiden. Das gelingt durch eine Einteilung in folgende drei Kategorien:

- Datenbankmanagementsystem
- Betriebssystem
- Netzwerk

[51, S.168].

Ausschlaggebend ist die Identifizierung von Bedrohungen und Schwachstellen in diesem Kontext. Eine zentrale Rolle spielen die Begriffe Bedrohung, Schwachstelle und Schwachstellenausnutzung.

Eine Bedrohung ist ein Ereignis oder ein Zustand, durch den ein Schaden entstehen kann [22, S.37]. Eine Schwachstelle ist ein Sicherheitsrelevanter Fehler eines IT-Systems oder einer Institution [22, S.37]. Eine Schwachstelle kann dazu führen, dass eine Bedrohung wirksam wird. Diese Realisierung wird als eine Schwachstellenausnutzung oder auch Exploit bezeichnet [22, S.37]. Um die Sicherheit zu bewahren, müssen diese Exploits vermieden werden. Für dieses Vorhaben kann die „Triade der Informationssicherheit“ verwendet werden [22, S.46]. Die Triade definiert die Aspekte der Verfügbarkeit, Vertraulichkeit und Integrität im Umfeld einer Datenbankanwendung. Diese können als Überbegriffe für detailliertere Maßnahmen wie die Zugriffskontrolle, Verschlüsselungen, Authentifikationen und Backups betrachtet werden.

➢Die Verfügbarkeit, beschreibt die Zugänglichkeit von Dienstleistungen, Funktionen oder Informationen. Sie müssen von Anwendern wie vorgesehen genutzt werden können [22, S.46].

➢Die Vertraulichkeit, beschreibt den Schutz vor unbefugtem Zugriff auf Informationen. Ausschließlich Befugte dürfen Zugriff auf die besagten Informationen haben [22, S.48].

➢Die Integrität, bezieht sich auf die Korrektheit und Unversehrtheit von Daten. Diese müssen so abrufbar sein, wie sie eingetragen wurden. Es darf keine unerlaubte Änderung der Daten stattfinden [22, S.49].

Betriebssystemsicherheit

Zum Betriebssystem gehören Prozessor-, Speicher-, Ein/Ausgabeverwaltung sowie die grafische Benutzeroberfläche und Netzwerk-Funktionen [12, S.2]. Das Betriebssystem

ist ein entscheidender Faktor für die Informationssicherheit, da es als Schnittstelle zwischen Hardware und Software operiert. Ein Betriebssystem sollte durch Zugriffskontrollen, einer Sandbox und andere Mechanismen wie dem Monitoring (zum Aufzeichnen von Systemaktivitäten) geschützt werden und nur autorisierten Zugriff zulassen [118, S.273-274].

Das Missachten der Betriebssystemsicherheit kann dazu führen, dass Hardware und Software von Angriffen betroffen sind und dadurch das gesamte Netzwerk und die Datenbanksysteme gefährdet werden können.

Netzwerksicherheit

Die Netzwerksicherheit ist der Zustand, in dem ein Rechnernetzwerk samt seinen Komponenten exakt den vom Netzwerkbetreiber angedachten Zweck erfüllt [118, S.87]. Diese Netzwerksicherheit kann durch Angriffe beeinflusst werden. Angriffe können entweder passiv, zur Gewinnung von Informationen, oder aktiv, zur Beeinflussung von Zielsystemen eingesetzt werden [118, S.188].

Es existieren viele Methoden, um die Netzwerksicherheit zu bewahren. Im Kontext der Entwicklung von sicheren Datenbankanwendungen spielt sie eine entscheidende Rolle, um die Informationssicherheit einzuhalten. Es ist ebenso wie die Betriebssystemsicherheit ein umfangreiches Themengebiet. Methoden zur Umsetzung werden in Kapitel 5 genauer betrachtet.

2.5 Richtlinien, Standards und Best Practices

Sicherheitsstandards sind alle Sicherheitsregeln, -richtlinien und -standards eines Unternehmens, die dazu beitragen, ein gewünschtes Sicherheitsniveau zu erreichen [71, S.153]. Für die Informations- und Kommunikations-Technik (IKT) existieren übergreifende Regeln zum Zugang und Zugriffsschutz. IKT-Sicherheitsmanagement beschreibt die Planung, Konzeption und Umsetzung sowie die kontinuierliche Prüfung und Überwachung des Sicherheitsniveaus in einer IKT-Umgebung [71, S.123]. Hierbei wird vor allem Wert auf die Netzwerk- und Betriebssystem-Sicherheit gelegt und damit verbunden die Sicherheit von Datenbankanawendungen. Der Einsatz dieser kann die komplexen Themengebiete Betriebssystem- und Netzwerksicherheit komplett abdecken. Im Folgenden werden die Begriffe Richtlinien und Standards erklärt:

Richtlinien: Richtlinien Erstellen eine Basis für ein Unternehmensweites Sicherheitsniveau, das weitgehend einheitlich und zentral steuerbar ist [71, S.153-154]. Somit geben Richtlinien die Richtung vor, auf die sich dann die weitere Entwicklung von Konzepten befassen kann.

Standards: Ein Standard wird auf der BSI-Webseite folgendermaßen definiert: „Sie enthalten Empfehlungen zu Methoden, Prozessen und Verfahren sowie Vorgehensweisen und Maßnahmen zu unterschiedlichen Aspekten der Informationssicherheit." [13].

In Bezug auf Kontext und Umfeld sind Standards aber häufig unterschiedlich und können regelmäßig Änderungen unterliegen, weil sich angebotene Systeme und Technologien permanent weiterentwickeln. Sie unterteilen sich in generische und spezifische Standards.

Generische Standards: Diese können Betriebssysteme oder Netzwerke betreffen und sind Normen und Richtlinien zur Erfüllung bestimmter Zielvorgaben, beispielsweise der Informationssicherheit [71, S.153].

Spezifische Standards: Solche Standards sind auf spezifische Situationen zugeschnitten. Ein Beispiel ist der Notfall-, Krisen- und Katastrophenplan (NKK-Plan). Durch diese Art von Standards kann eine einheitliche Struktur für bestimmte Situationen hergestellt werden [71, S.154].

Best Practices

Best Practices sind ähnlich wie Standards und unterscheiden sich lediglich in den Abstraktionsebenen [71, S.124]. Sie bilden mit Rahmenwerken zusammen Vorgaben zur Umsetzung bestimmter Prinzipien und Strategien. Beispielhaft kann das Best Practice *„Database Security"* von IBM erwähnt werden [54]. Im Vergleich zu Standards sind sie aber nur „Empfehlungen" von Organisationen wie unter anderem auch der internationalen National Institute of Standards and Technology (NIST) oder in Deutschland von dem Bundesamt für Sicherheit und Informationstechnik (BSI).

Die Umsetzung dieser Richtlinien, Standards und Best Practices wird als Sicherheitsmaßnahme definiert. Im Folgenden sind einige aktuelle und weit verbreitete Sicherheitsstandards aufgelistet:

- ISO/IEC 27001 (Informationssicherheit, Cybersicherheit und Schutz der Privatsphäre – Informationssicherheitsmaßnahmen)

- IT-Grundschutz (BSI IT-Grundschutz)
- COBIT (Control O**b**jectives for Information and Related Technology)
- ITIL (Information Technology Infrastructure Library)
- ITSEC/CC (Information Technology Security Evaluation Criteria/Common Criteria)
- NIST Cybersecurity Framework (NIST CSF)
- Zero Trust Architecture – National Institute of Standards and Technology
- Database Security – IBM

Damit eine Datenbank bestmöglich geschützt wird, müssen zuallererst sichere Rahmenbedingungen geschaffen werden. Potenziellen Angreifen den Zugang zu der Datenbank zu erschweren, ist ein entscheidender Faktor. Eine sichere Umgebung bereit zu stellen, dient als Grundlage für das darauf aufbauende Datenbanksystem und sollte unbedingt umgesetzt werden. Der Einsatz von Standards und Best Practices kann deshalb sehr hilfreich sein und ist stark zu empfehlen, wenn hohe Sicherheitsanforderungen bestehen.

3 Sicherheitsaspekte von Datenbankanwendungen

Um eine sichere Datenbankanwendungen zu entwickeln, muss erst einmal identifiziert werden, an welchen Stellen Probleme auftreten können, was also potenzielle Risiken sind. Eine etablierte Methode diesbezüglich ist die Risikoanalyse. Risikoanalysen werden verwendet, um Schwachstellen im Kontext zu der Sicherheit, Reinheit, Funktionalität und Zuverlässigkeit einer bestimmten Situation oder eines Umfeldes zu definieren [28, S.7]. Die Identifizierung von Problemen und ihre Analyse werden in diesem Kapitel behandelt, um darauf basierend Methoden zur Bewältigung vorstellen zu können.

3.1 Systematisches Risikomanagement

Jede Anwendung beziehungsweise jede Organisation hat Schwachstellen, die es zu definieren gilt. Das Identifizieren, Analysieren und Bewältigen dieser Schwachstellen kann innerhalb der einzelnen Prozessschritte des Risikomanagements erfolgen [27, S.91]. Das Risikomanagement wird in dem Kontext dieser Arbeit in drei Phasen eingeteilt: Risikoidentifizierung, Risikobeurteilung/Risikoanalyse und Risikobewältigung (auch genannt Risikosteuerung).

Phase 1: Risikoidentifikation

Bei der Risikoidentifikation geht es darum, Daten zu sammeln und diese zu verarbeiten, so dass aussagekräftige Informationen gewonnen werden können. Dieser Prozess wird auch als Risikobewertung oder Risikoaggregation bezeichnet [41, S.156]. Die gesammelten Informationen müssen denen der realen Welt entsprechen und bilden die Grundlage für weitere Analysen.

Phase 2: Risikoanalyse/Risikobeurteilung

Nach der Risikoidentifizierung sollen die Risiken analysiert und beurteilt werden [41, S.135]. Ziel ist an dieser Stelle herauszufinden, wie sich die identifizierten Risiken bei einem Eintreffen verhalten beziehungsweise welche Reaktion sie auslösen, um besser zu verstehen, wie diese vermieden werden können.

Phase 3: Risikobewältigung

Die Phase der Risikobewältigung befasst sich mit den Lösungen zu den identifizierten und analysierten Risiken. Damit ist eine Limitierung des potenziellen Schadens sowie eine Minimierung der Eintrittswahrscheinlichkeit gemeint [41, S.158]. Ziel ist somit das

positive Verändern der Risikolage und das Vermeiden zukünftiger Risiken durch den Einsatz passender Methoden.

Der hier beschriebene Prozess über 3 Phasen soll bewirken, dass passende Methoden für Gegenmaßnahmen identifiziert und eingesetzt werden können.

3.1.1 Risikoidentifizierung

Die Risikoidentifizierung befasst sich mit Schwachstellen und Risiken. Schwachstelle und Risiko zusammen ergeben eine potenzielle Gefahr, die es zu limitieren oder komplett zu vermeiden gilt. Das setzt voraus, dass alle gefährdenden Gebiete betrachtet und analysiert werden, um Schwachstellen zu verstehen und die Risikowahrscheinlichkeit gering zu halten. Durch den präventiven Einsatz der richtigen Abwehrstrategie kann dieses Vorhaben umgesetzt werden. Bezogen auf die Informationssicherheit von Datenbankanwendungen sind hierbei das Datenbanksystem, das Betriebssystem, die Netzwerksicherheit und alle teilnehmenden Personen betroffen. Tabellarisch abgebildet kann das folgendermaßen aussehen:

Tabelle Risikoidentifizierung 1

Identifiziertes Risiko	Potenzielle Gefahren
Datenbanksystem	*Fehlkonfiguration, kein Einsatz der Triade der Informationssicherheit.*
Betriebssystem	*Fehlende Sicherheitsvorkehrungen- und Maßnahmen.*
Netzwerk	*Fehlende Sicherheitsvorkehrungen- und Maßnahmen.*
Personen	*Fahrlässiger Umgang mit vertraulichen Informationen, Missbrauch der eingeräumten Rechte.*

Tabelle 1: Tabelle Risikoidentifizierung 1 (eigene Tabelle)

Durch eine Einteilung in die Triade der Informationssicherheit können die oben erwähnten Bereiche für potenzielle Gefahren spezifisch zugeordnet werden: Mithilfe einer Einteilung der einzelnen Gefahren in die Gebiete Verfügbarkeit, Vertraulichkeit und Integrität werden Aspekte der Informationssicherheit Tabellarische veranschaulicht. Weil sich die definierten Risiken zum Großteil auch auf Personen-Interkationen beziehen, sind

diese passiv in allen 3 Gebieten der Informationssicherheit vertreten, weswegen auf das explizite erwähnen dieses Aspekts in der folgenden Tabelle verzichtet wurde.

Tabelle Risikoidentifizierung 2

Identifiziertes Risiko	**Potenzielle Gefahren**	
Verfügbarkeit	*1. Angriff auf die Netzwerksicherheit* *2. Angriff auf die Betriebssystem-Sicherheit* *3. Keine Back-Up Strategie* *4. Keine Maßnahmen gegen SQL-Injektion*	*Bei nicht beachten der Gefahren riskiert der Anwender den Verlust aller Informationen*
Vertraulichkeit	*1. Keine Verschlüsselung* *2. Kein Auditing* *3. Keine ausreichende Authentifizierung* 4. *Keine Sichere Passwort Verwaltung* *5. Keine Rollenbasierte Zugriffskontrolle* *6. Keine Maßnahmen gegen SQL-Injektion*	*Das Fehlen der aufgelisteten Punkte gefährdet in einem besonders hohen Maße die Vertraulichkeit der Datenbank.*
Integrität	*1. Inkorrekte Datentypen und Datenstrukturen* 2. *Keine Konsistenzprüfung*	*Datensicherheit wird gefährdet, da nicht sichergestellt ist, dass eingefügte Daten in erwarteter Form ausgegeben werden.*

	3. *Transaktionsprobleme* 4. *Keine Maßnahmen gegen SQL-Injektion*	

Tabelle 2: Tabelle Risikoidentifizierung 2 (eigene Tabelle)

Die aufgestellte Tabelle verdeutlicht, dass eine Einrichtung der eingeordneten Punkte vital für den Schutz der Daten in einem Datenbanksystem ist. Zusätzlich wird auch klar, dass SQL-Injektionen in jeder Spalte vertreten sind. SQL-Injektionen sind Angriffe auf ein Datenbanksystem mithilfe eingeschleusten Codes über Eingabefelder in einer Datenbankanwendung [121, S.215]. Falls erfolgreich, haben Angreifer die Möglichkeit, die Verfügbarkeit, Vertraulichkeit und Integrität des Systems zu gefährden. Neben SQL-Injektionen existieren noch andere potenziell gefährliche Angriffe, wie beispielsweise das Cross-Site-Scripting (XSS) [11, S.4850]. Diese Arbeit soll einige mögliche alternative Angriffe besprechen, jedoch nicht im Detail auf jeden potenziellen Angriff eingehen. Primär sollen Konzepte vorgestellt werden, die die Entwicklung einer allgemein sicheren Datenbankanwendung ermöglichen.

3.1.2 Risikoanalyse/Risikobeurteilung

Bei der Risikobeurteilung soll das mögliche Schadensausmaß eines eingetroffenen Riskos bestimmt werden [27, S.135]. Ziel ist in diesem Kontext die nachvollziehbare Beurteilung der identifizierten Risiken. An dieser Stelle wurde die Risikobaumanalyse eingesetzt, um Probleme logisch nachvollziehbar darzustellen und das Schadensausmaß von kombiniert eintreffenden Risiken zu veranschaulichen [30, S.3].

Risikobaumanalyse

Die Risikobaumanalyse eignet sich gut, wenn mehrstufige Probleme existieren [97, S.83-84]. Sie wurde entwickelt, um Kausalketten nachvollziehen zu können und durch die Kombination von verschiedenen Problemen entstehende Schwachstellen durch deren Analyse zu bewältigen. Die Risikobaumanalyse wurde in diesem Fall für die Triade der Informationssicherheit eingesetzt, da sich im Vorfeld herausgestellt hat, dass diese das Hauptaugenmerk der Analyse bildet. Zusätzlich wurde ein eigener Risikobaum für SQL-Injektionen erstellt. Die Risikobäume werden jeweils in 3 Ebenen aufgeteilt, der

Aufbau ist Bottom-Up: ganz oben steht das eigentliche Problem, welches aus einer Kombination der Risiken aus den vorherigen Ebenen entsteht.

Datenintegrität:

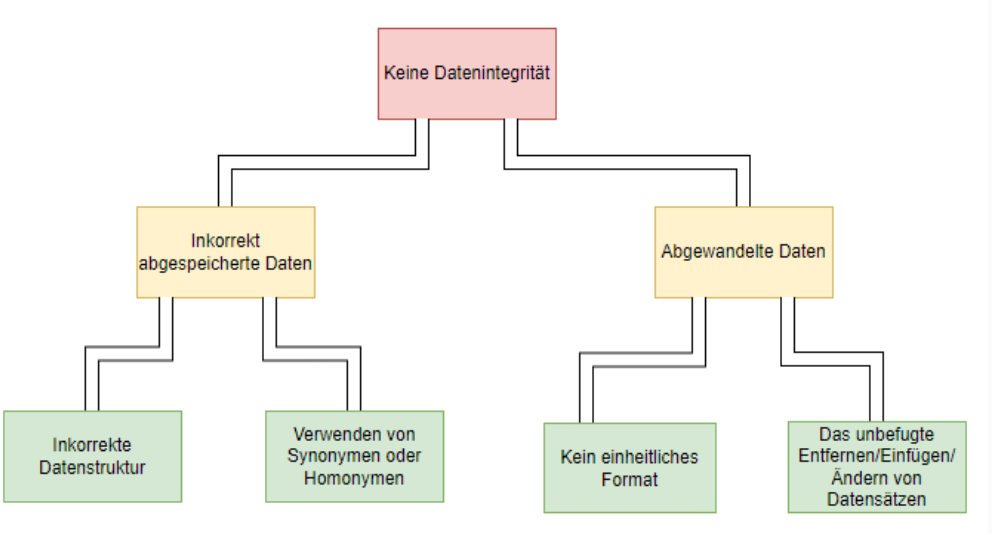

Tabelle 3: Fehlerbaumanalyse 1 – Datenintegrität (eigene Tabelle)

In dem ersten Risikobaum wird die Datenintegrität betrachtet. Faktoren, die zu fehlender Datenintegrität führen, werden visuell dargestellt. Zu diesen Faktoren zählen: Eine Inkorrekte Datenstruktur, das Verwenden von Synonymen und Homonymen, die zu inkorrekt abgespeicherten Datensätzen verhelfen, sowie das Fehlen eines einheitlichen Formats, was zu Problemen bei der Datenumwandlung führen kann. Inkorrekt abgespeicherte- sowie Abgewandelte Daten führen zum Verlust der Datenintegrität.

Verfügbarkeit

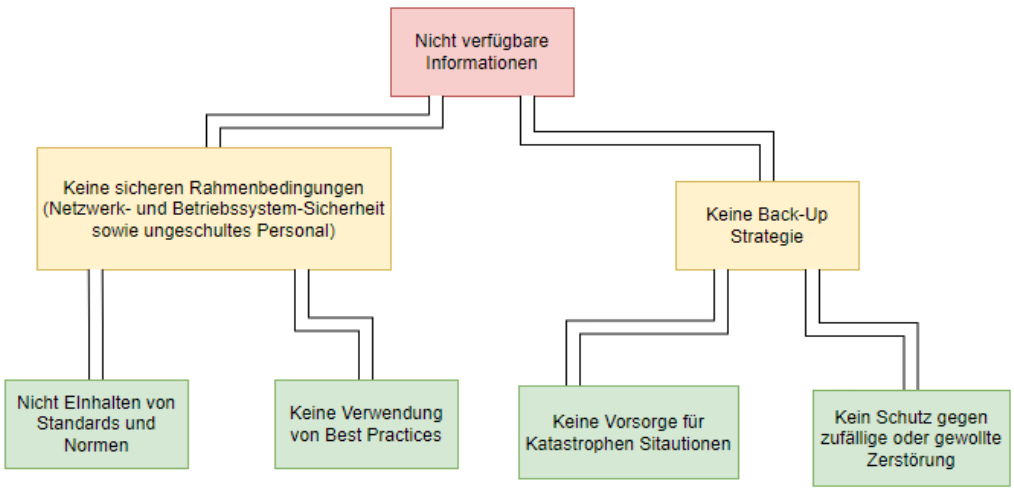

Tabelle 4: Fehlerbaumanalyse 2 – Verfügbarkeit (eigene Tabelle)

Bei der Verfügbarkeit werden in erster Linie die Rahmenbedingungen der Datenbanksysteme analysiert. Wenn für die zu der Risikoidentifizierung genannten Punkte keine Sicherheitsvorkehrungen eingerichtet werden, hat das zur Folge, dass die Informationssicherheit sehr anfällig für Angriffe oder internen Missbrauch wird. Das kann auch die Folge von einer schwachen Netzwerksicherheit sein. Weiter ergibt die Analyse, dass ohne eine gute Back-Up Strategie die Gefahr besteht, durch zufällig eintretende Ereignisse einen Informationsverlust zu riskieren.

Vertraulichkeit:

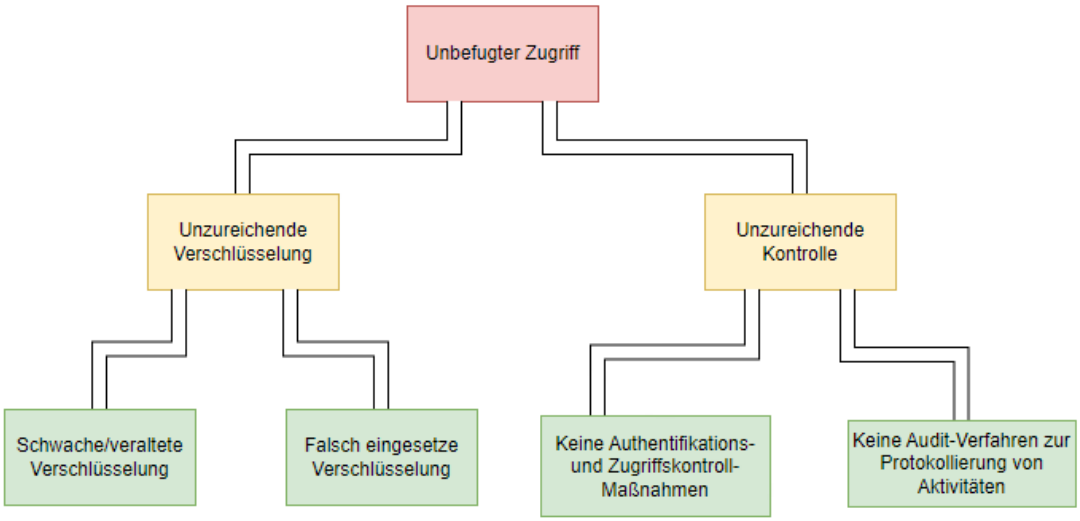

Tabelle 5: Fehlerbaumanalyse 3 – Vertraulichkeit (eigene Tabelle)

Der dritte Risikobaum behandelt Verschlüsselungen zum Schutz der gespeicherten Daten. Dabei sollen die Modernsten Verschlüsselungsmechanismen betrachtet werden, um eine unzureichende Verschlüsselung vorzubeugen. Bezüglich der Kontrolle innerhalb eines Datenbanksystems müssen Aktivitäten von Mitarbeitern zurückverfolgt werden können, um Fahrlässigkeit aufzuzeichnen beziehungsweise unbefugte Aktionen zu registrieren. Zusätzlich spielen die Themenfelder Authentifikation und Zugriffskontrolle in dieser Hinsicht eine Rolle, auch wenn sie übergreifend auf die anderen beiden Aspekte der Triade der Informationssicherheit zutreffen. Ohne den Einsatz der notwendigen Mittel gegen diese Probleme kann ein unbefugter Zugriff nicht verhindert werden.

SQL-Injektion:

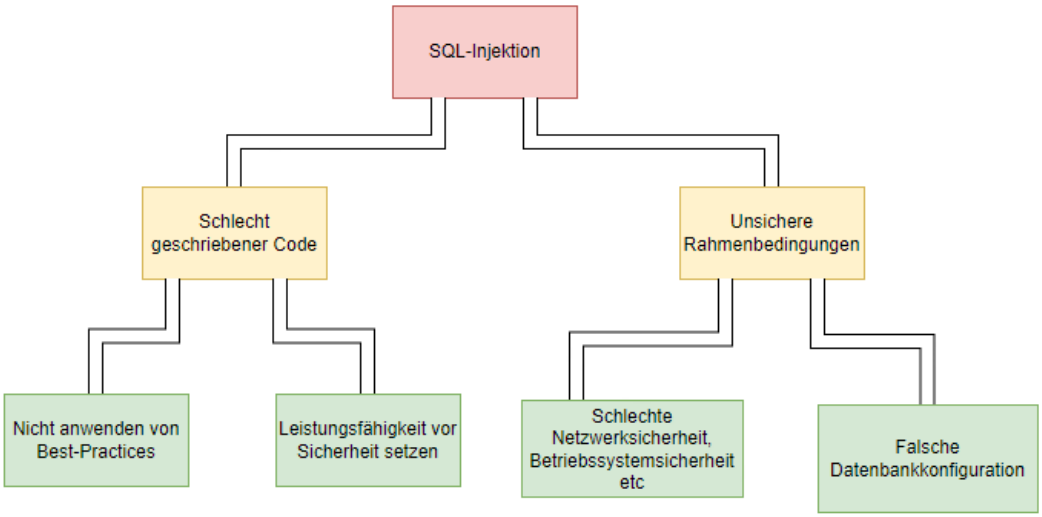

Tabelle 6: Fehlerbaumanalyse 4 – SQL-Injektion (eigene Tabelle)

SQL-Injektionen wurden an dieser Stelle neben anderen möglichen Angriffen explizit betrachtet, weil sie die am weitesten verbreitete Methode von Angriffen bilden, die auf Datenbankanwendungen eingesetzt werden können [122, S.215]. Der Einsatz von Gegenmaßnahmen, wie beispielsweise der Validierungen von Benutzereingaben, kann dabei helfen, auch andere potenzielle Angriffe zu verhindern.

Bei SQL-Injektionen spielen vor allem schlechter Code und schlechte Rahmenbedingungen eine entscheidende Rolle. Ein schlechter Code wird geschrieben, wenn entweder Best-Practices nicht umgesetzt oder die falschen Prioritäten gesetzt werden. Als Beispiel kann in diesem Kontext der bevorzugte Einsatz von Mitteln für eine besonders hohe Leistung von Anfragen im Austausch gegen eine solide Sicherheit sein. Hierdurch nimmt der Entwickler für eine bessere Performance willigend Risiken in Kauf, die die Informationssicherheit gefährden.

Als unsichere Rahmenbedingung kann eine ungenügende Netzwerk- beziehungsweise Betriebssystem-Sicherheit sowie eine schlechte Datenbankkonfiguration genannt werden. Das Missachten dieser bietet eine potenziell große Angriffsfläche und sollte unbedingt vermieden werden.

Die Risikobaumanalyse bezüglich der Informationssicherheit und der damit verbundenen Entwicklung von sicheren Datenbanksystemen hat ergeben, dass das Abdecken der drei aus der Risikoidentifizierung gefundenen Überbegriffe, Vertraulichkeit, Verfügbarkeit und Integrität, unbedingt erfolgen muss, um die Sicherheit einer Datenbankanwendung zu gewährleisten. Zusätzlich spielen SQL-Injektionen und damit verbunden auch

andere Angriffe eine entscheidende Rolle und können alle drei Aspekte betreffen, wenn sie nicht durch passende Maßnahmen vermieden werden. Auch wenn die Triade der Informationssicherheit eine solide Grundsicherheit bietet, muss durch Best Practices in der Codierung noch einmal für verschiedene Angriffe speziell ein gewisser Schutz eingerichtet werden. Für den korrekten Einsatz von Code müssen dementsprechend auch die richtigen Prioritäten gesetzt werden.

3.1.3 Risikobewältigung

Bei der Risikobewältigung sollen die identifizierten und analysierten Risiken dazu beitragen, passende Konzepte und Methoden zu finden., um eine positiv ausfallende Risikoverteilung zu erlangen [97, S.283-284]. Das bedeutet, dass mögliche Schwachstellen und damit verbunden Eintrittswahrscheinlichkeiten minimiert und bei einem Eintritt das Schadensausmaß stark limitiert werden soll. Somit wird eine Veränderung der Risikolage angestrebt [41, S.158].

Die Risikoanalyse hat ergeben, dass eine vermiedene Umsetzung von Maßnahmen gegen die identifizierten Risiken die Triade der Informationssicherheit verletzt, was es zu vermeiden gilt. Dementsprechend sollen Methoden in Form von Maßnahmen eingesetzt werden, um die Risikolage positiv zu verändern.

3.2 Methoden & Konzepte für Sicherheitsmaßnahmen

Dieser Abschnitt soll sich damit beschäftigen, Methoden und Konzepte zu erforschen und vorzustellen, die die Sicherheitsaspekte von Datenbankanwendungen unterstützen und die Risikobewältigung ermöglichen. Hierzu werden die identifizierten Risiken genauer betrachtet, um passende Maßnahmen für ihre Limitierung zu erforschen. Die Vorstellung der Methoden erfolgt über die Zuordnung in die Triade der Informationssicherheit mit einem zusätzlichen Abschnitt für das Management der Identitäts- und Zugriffskontrolle. Der Kontext zu der Risikobewältigung wird am Ende eines jeden Abschnittes mittels eines Fazits definiert. Der genaue Einsatz der Methoden wird in Kapitel 4 und 5 vorgestellt.

3.2.1 Verfügbarkeit

Die Verfügbarkeit kann mithilfe einer gut eingerichteten Betriebssystem- und Netzwerksicherheit bewahrt werden. Wie bereits in den Grundlagen angemerkt, werden diese Themen aber nicht zu detailliert behandelt, weil sie für sich selbst sehr umfangreich sind. Kapitel 5 stellt diesbezüglich einige Maßnahmen vor. Im Folgenden werden Möglichkeiten erforscht, um die Verfügbarkeit für Datenbankanwendungen einzurichten.

Datenbankkonfiguration

Die Datenbankkonfiguration kann dabei helfen, den Code der Datenbankanwendung zu sichern. Ein Datenbankserver aktiviert oft standardmäßig Funktionen, die nicht unbedingt erforderlich sind und von Angreifern ausgenutzt werden können [17, S.50]. Eine Gegenmaßnahme ist an dieser Stelle deshalb eine Zusammenarbeit zwischen Entwickler und DBA, um darauf zu achten, dass solche Funktionen deaktiviert beziehungsweise individuell angepasst werden. Diese können von Datenbankplattform zu Datenbankplattform variieren und sollten deshalb in jeder Umgebung explizit betrachtet werden.

Back-Up

Backup-Systeme dienen dazu, gespeicherte Informationen im Katastrophenfall verfügbar zu halten. Sie sind passiv und häufig geografisch entfernt stationiert [93, S.303]. Eine Backup Strategie funktioniert so, dass regelmäßige Kopien der Datenbank erstellt werden, die möglichst aktuell sind, um sie dauerhaft zu speichern und bei einer Notfallsituation verfügbar zu halten. Die meisten Datenbankplattformen bieten eigene Funktionen für das Einrichten einer Backup-Strategie an.

Golden Image

Ein Golden Image kann verwendet werden, um Datenbank von DBMS zu trennen und dadurch ein zuverlässiges Umziehen und damit verbunden eine höhere Verfügbarkeit der Datenbank zu gewährleisten [38, S.29-30]. Es ist im Wesentlichen eine Vorlage oder ein Musterbild einer vollständig konfigurierten und funktionsfähigen Computerumgebung, das als Ausgangspunkt für die Bereitstellung von identischen Kopien dieser Umgebung verwendet wird [46, S.36]. Durch den Einsatz eines Golden Image wird die unabhängige Verwaltung der beiden Elemente ermöglicht. Es ist sehr zu empfehlen, auf

eine Trennung zwischen Software und Datenbank selbst wert zu legen, so dass bei einer Notfallsituation die Datenbank mit einer anderen Software verwendet werden kann.

Protokollierung (Audit)

Die Überwachung von Zugriffen und Aktionen ist eine wichtige Maßnahme für die Sicherheit eines Systems. Das in diesem Kontext notwendige Protokollieren von Zugriffen wird als Auditing bezeichnet [47, S.500]. Es ist eine systematische Prüfung von Aufzeichnungen, Prozessen oder Systemen zum Abgleich mit Richtlinien für interne Kontrollen. Bei der Protokollierung können unterschiedliche Attribute wie das Datum, die Uhrzeit, der Benutzer und die Netzadresse vermerkt werden [102, S.169]. Audits haben den Vorteil, dass durch sie Benutzer identifizierbar werden und somit ihre Aktivitäten zurückverfolgt werden können. Es soll als Abschreckung gegenüber internen Angreifern dienen und erfüllt diesen Zweck auch gut. Der Nachteil ist aber, dass laufende Audits sehr viel Rechnerleistung verbrauchen. Das ist der Grund, weswegen sie nur in besonderen Verdachtsfällen oder auf besonders gut zu sichernden Objekten anzuwenden sind.

Fazit

Die Verfügbarkeit befasst sich hauptsächlich mit der Datenbankkonfiguration, Backup-Strategien und Auditing. Sie soll dazu beitragen, die in einer Datenbank enthaltenen Daten verfügbar zu halten. Zur Risikobewältigung dieses Aspekts der Triade der Informationssicherheit sind dementsprechend die vorgestellten Methoden sehr hilfreich und sollten für eine sichere Datenbankanwendung unbedingt eingesetzt werden.

3.2.2 Integrität

Die Integrität kann bereits beim Definieren einer Tabelle mit SQL durch das Einrichten von Bedingungen bewahrt werden. Hierbei spielen die in Kapitel 2 definierten Begriffe Entity- und Referentielle Integrität eine besondere Rolle. Im Kontext dieser Begriffe stehen Transaktionen, die die Integritätsregeln berücksichtigen und nur bei Erfüllung dieser ausgeführt werden können. Zusätzlich spielt aber auch der Schutz von Passwörtern und anderen wichtigen Informationen eine wichtige Rolle für die Integrität. Der folgende Abschnitt stellt diesbezüglich Einschränkungen auf Tabellen- und Spalten-Ebene sowie

Transaktionssteuerungen und Hashfunktionen inklusive des Salzens vor, die für die Erhaltung der Integrität notwendig sind.

Einschränkungen/Constraints

Constraints werden verwendet, um bestimmte Vorgaben in Tabellen umzusetzen. Es sind Einschränkungen, die für eine oder mehrere Spalten einer Tabelle gelten [9, S.241-242]. Sie dienen dazu, die referentielle Integrität in einem Datenbanksystem zu bewahren. Die Festlegung von Constraints erfolgt entweder bei der Definition einer Spalte oder aber einer ganzen Tabelle als Tabellen-Constraint [39, S.81]. Neben den üblichen Einschränkungen für die referentielle Integrität existieren noch weitere Bedingungen, mit denen die Datenbankintegrität bewahrt werden kann. Eines von denen ist der Check-Constraint.

Check-Constraint

Der Check-Constraint kann verwendet werden, um zulässige Werte für eine Spalte einzuschränken. Somit kann eine Bedingung eingerichtet werden, die eingehalten werden muss, damit eine Transaktion ausgeführt werden kann [39, S.82].

Der SQL-Standard hat sehr viele unterschiedliche Operatoren, die Bedingungen für die Integrität festlegen können. Dazu gehören Vergleichsoperatoren, Bereichsprüfungen, Elementprüfungen, Mustervergleich, NULL-Wertprüfung, Logische Operatoren und Existenzbedingungen [39, S.107]. Durch einen gezielten Einsatz dieser Operatoren kann eine hohe Integrität umgesetzt werden, um Transaktionsprobleme zu vermeiden.

Transaktionssteuerung

Eine Transaktion ist ein zusammenhängender Arbeitsprozess, der entweder vollständig durchgeführt oder abgebrochen werden muss [96, S.470]. Das bedeutet, dass die Transaktion nicht frühzeitig beendet werden darf. Zu Transaktionen zählen verschiedene DML-Befehle zur Veränderung von Tabelleninhalten [60, S.224]. Transaktionen sind wichtig für die Datenintegrität einer Datenbankanwendung und müssen konsistent bleiben, also die ACID-Eigenschaften vertreten. Um Transaktionen abzuschließen oder rückgängig zu machen, werden die grundlegende Befehle zur Transaktionssteuerung COMMIT und ROLLBACK verwendet.

COMMIT und ROLLBACK

Mit COMMIT können Transaktionen begonnen/abgeschlossen und mit ROLLBACK rückgängig gemacht werden [39, S.184-185]. Optional kann der Befehl BEGIN TRANSACTION für den Beginn der Transaktion definiert werden, was aber nicht notwendig ist, da COMMIT die Transaktion endgültig abschließt und ein Beginn dementsprechend nicht definiert werden muss [60, S.224]. Weil Transaktionen immer abgeschlossen werden müssen, um die Datenbankkonsistenz beizubehalten, ist der Einsatz von COMMIT essenziell. ROLLBACK ist hilfreich, wenn eine Transaktion rückgängig gemacht werden soll, was in bestimmten Situationen vorkommen kann. Wichtig in diesem Kontext sind mögliche Probleme, die bei Transaktionen auftreten können. Diese werden in einem späteren Kapitel vorgestellt.

Hashfunktionen

Eine Hashfunktion komprimiert Nachrichten einer beliebigen Länge. Hierdurch entsteht ein Code, auch genannt Digest oder Hashwert, der in dem Datenbankserver hinterlegt wird [105, S.289]. Diese Hashwerte sind eindeutig identifizierbar, was bedeutet, dass die Eingabe einer identischen Nachricht denselben Hashwert erzeugt [10, S.17]. Sie nehmen Daten in Form von Eingangswerten auf um sie, im Gegensatz zu Verschlüsselungen, dauerhaft zu verändern [105, S.289]. Der Einsatz von Hashfunktionen zum Umwandeln von Eingabewerten in Hashwerte kann auch als Hashing bezeichnet werden und dient zur Bewahrung der Integrität der gespeicherten Daten. Dank der eindeutigen Identifizierbarkeit der Hashwerte kann sichergestellt werden, dass die Daten denen entsprechen, die im System hinterlegt sind. Es existieren verschiedene Hash-Algorithmen, die verwendet werden können, um Hashfunktionen auszuführen. Im Folgenden wird der Secure Hash-Algorithmus 2 (SHA2) vorgestellt.

Secure Hash-Algorithmus 2 (SHA2)

SHA-2 ist ein Algorithmus, der mehrere kryptografische Hashfunktionen beinhaltet. Diese unterscheiden sich primär in ihrer Größe und werden zur Erzeugung von Hashwerten für Daten verwendet. Die Hashfunktionen in der SHA-2-Familie umfassen SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224 und SHA-512/256. Diese Funktionen basieren auf die Merkle-Damgard-konstruktion [121, S.39ff.].

Die Merkle-Damgard-Konstruktion verwendet eine Komprimierungsfunktion [121, S.41]. Die Kompirimierungsfunktion ist grundlegend deterministisch, das heißt dieselbe Eingabe erzeugt dieselbe Ausgabe. Das ist wichtig, um Probleme wie die Kollision zu vermeiden. Die Kollision ist ein Ereignis, bei dem durch eine Eingabe zweimal derselbe Hash-Wert entstehen kann [26, S.157]. Dabei spielt die Größe des verwendeten Hash-Algorithmus bezüglich der Kollision eine Rolle: Je größer die Anzahl der Bits ist, desto sicherer ist der Hashwert gegen zwei verschiedene Klartextdatensätze, die denselben Wert erzeugen und eine Kollision herbeirufen. Anfällig für solche Kollisionen können ältere Hash-Algorithmen sein, wie beispielsweise MD5 und SHA-1, wobei diese aber anfällig für Kollisionen sein können [121, S.39].

Salzen/Salting

Beim Salzen wird ein sogenanntes Salz erstellt, das den Eingabewert um zusätzliche Zeichen erweitern soll. Es funktioniert so, dass das Salz an den Eingabewert angeheftet wird, um den entstandenen neuen Wert mit der Hash-Funktion umzuwandeln [65, S.270]. Der Salz-Wert wird zufällig generiert und mit dem Hash-Wert zusammen gespeichert, so dass bei einer erneuten Abfrage derselbe Salz verwendet werden kann, um den gleichen Hash-Wert zu erzeugen.

Salzen Ist eine zusätzliche Sicherheitsanwendung, die Hashwerte von Passwörtern weniger anfällig gegen Brute-Force Angriffe machen soll. Brute-Force Angriffe sind Attacken, bei denen wiederholt verschiedene, zufällig generierte Passwörter eingegeben werden, um das richtige Passwort zu erraten [11, S.4850].

Außerdem hilft das Salzen durch die Erweiterung des Hash-Wertes dabei, die Kollision zu vermeiden. Das ist besonders bei den schwächeren Hash-Algorithmen SHA-1 und MD5 wichtig, die heute noch häufig verwendet werden.

Fazit

Deutlich wird nach diesem Abschnitt, dass für die Integrität einer Datenbankanwendung sind Einschränkungen, Transaktionssteuerungen und Hash-Funktionen essenziell sind. Durch den Einsatz dieser Mittel können der Zugriffsschutz bewahrt und Datenlecks vermieden werden. Konzepte für die Bewältigung von integritäts-Risiken wurden dementsprechend vorgestellt und sollten implementiert werden.

3.2.3 Vertraulichkeit

Bei der Vertraulichkeit bezüglich Datenbankanwendungen geht es darum, die Zugänglichkeit der gespeicherten Informationen zu bewahren. Um Maßnahmen für die Vertraulichkeit vorzustellen, werden Verschlüsselungen genauer betrachtet. Daraufhin werden Verschlüsselungsmethoden und in diesem Zusammenhang der aktuell sicherste Verschlüsselungs-Algorithmus Advanced Encryption Standard (AES) vorgestellt. Abschließend werden Zugriffskontroll-Modelle angesprochen.

Verschlüsselungen

Verschlüsselungen sind Teil der Kryptografie und gehören zum Aspekt der Vertraulichkeit. Die Kryptografie ist eine Wissenschaft, die sich mit Methoden beschäftigt, die durch den Einsatz von Verschlüsselungen Daten vor unbefugtem Zugriff schützen sollen [103, S.39].

Bei einem Verschlüsselungsverfahren werden Informationen umgewandelt in verschlüsselte Dateien. Sie verwenden gewisse Algorithmen, um diese Umwandlung zu ermöglichen. In diesem Kontext bezieht sich der Begriff Algorithmus darauf, Informationen als Eingabe zu verwenden und diese dann in abgewandelter Form als Ausgabe zu liefern [21, S.5]. Für dieses Konzept existieren verschiedene Methoden, die Hauptsächlich in zwei Kategorien unterteilt werden können: symmetrische und die asymmetrische Kryptografie.

Symmetrische Kryptographie

Bei der symmetrischen Kryptographie existiert eine Eingabe, die Verschlüsselung dieser Eingabe, und eine verschlüsselte Ausgabe. Die Verschlüsselung wird mithilfe eines Schlüssels durchgeführt, der mit den Operatoren Exklusiv-Oder-Verknüpfung (XOR-Verknüpfung), Substitution und Permutation einen normalen Text in einen verschlüsselten umwandeln kann [103, S.39].

Verwendet wird die symmetrische Verschlüsselung in Datenbanksystemen, wenn bestimmte Daten oder ganze Entitäten aufgrund einer hohen Vertraulichkeit verschlüsselt werden müssen. Dazu können Tabellen, Spalten oder auch Backup-Daten gehören. Die symmetrischen Verschlüsselungen unterteilen sich in Block- und Strom-Chiffren.

Blockchiffre

Die Blockchiffre ist als Verschlüsselungs-Methode in der Datenspeicherung stark vertreten und spielt eine bedeutende Rolle bei dem Bewahren der Informationssicherheit. Bei der Blockchiffre werden Klartext-Eingaben in verschiedene Blöcke aufgeteilt, die alle eine einheitliche Größe haben [10, S.11]. Durch den Einsatz der Feister- oder der SPN-Struktur (abhängig von dem verwendeten Algorithmus) wird eine besonders hohe Sicherheit gewährleistet [50, S.3]. Sie baut bei der Verschlüsselung auf eine spezielle Betriebsart, das Cipher Block Chaining (CBC), auf [121, S.82]. Diese stellt sicher, dass die Verschlüsselung über gleich große Blöcke erfolgt. Dabei kann die Funktionsweise folgendermaßen beschrieben werden: Es existiert ein Initialisierungsvektor (IV), ein Klartext, der Schlüssel und der verschlüsselte Ausgabetext. Der Klartext wird in Blöcke von fester Größe aufgeteilt, mit dem IV randomisiert und mit einem Verschlüsselungsalgorithmus und Schlüssel dann in Blöcken als verschlüsselter Text ausgegeben. Diese Blöcke werden zusammengefügt und es entsteht ein verschlüsselter Ausgabetext. Der gesamte Prozess geschieht über mehrere Runden, so dass eine sehr sichere Verschlüsselung entstehen kann. In jeder Runde wird eine Hälfte durch den Verschlüsselungsalgorithmus transformiert und das Ergebnis mit der anderen Hälfte XOR-Verknüpft. Dieser Prozess nennt sich auch Diffusion [121, S.79]. Während der Diffusion verschlüsselt der ausgewählte Algorithmus, beispielsweise AES, den Klartext-Block. Die Entschlüsselung gelingt dann nur noch mit dem jeweiligen passenden Schlüssel.

Die Diffusion kann beispielhaft folgendermaßen veranschaulicht werden:

Eingabetext: 11001110

Schlüssel: 11110110

Ergebnis Runde 1: 00111000

An dieser Stelle wird mithilfe einer XOR-Verknüpfung das Ergebnis erzeugt. Wenn die Ziffern von dem Eingabetext und dem Schlüssel gleich sind, wird bei dem Ergebnis eine 0 eingesetzt. Falls die beiden Ziffern unterschiedlich sind, wird eine 1 eingesetzt. Die Anzahl der Runden wird durch die Schlüsselgröße bestimmt, die wiederum abhängig vom verwendeten Algorithmus ist. Eine vollständige Substitution der eingegebenen Datei beendet den Prozess.

Durch die Aufteilung in Blöcke bieten Blockchiffren eine besonders gute Verschlüsselung, die es Angreifern sehr schwer macht, eine Gegenmaßnahme zu finden. In Datenbanksystemen eigenen sie sich gut für die Verschlüsselung von statischen Informationen, in Form von ganzen Tabellen oder auf Zeilen bzw. Spaltenebene.

Advanced Encryption Standard (AES)

Der Advanced Encryption Standard, auch bekannt als Rijandael, ist ein Blockchiffre-Algorithmus zum Verschlüsseln von Informationen. Im Jahr 2001 wurde er von der NIST als offizieller Standard festgelegt [34].

Es gibt drei Versionen von AES, AES-128 (128 Bit), AES-192 (192 Bit), AES-256 (256 Bit) [1, S.1]. Diese unterscheiden sich lediglich darin, dass sie unterschiedliche Schlüssellängen anbieten. Wie bereits erwähnt bestimmt die Länge des Schlüssels den Grad der Sicherheit, was sich vor allem auf Brute-Force Angriffe bezieht.

Die Funktionsweise von AES baut auf den Substitution-Permutation-Network-Stil auf, bei dem über mehrere Runden eine Diffusion stattfindet, die den Eingabewert mithilfe der XOR-Verbindung verändert [4, S.1-2, S.5-6]. In diesem Kontext spielt die Anzahl der Runden eine entscheidende Rolle, die den Sicherheitsgrad bestimmen. Die Anzahl der Runden bezieht sich hierbei immer auf die Schlüsselgröße [4, S.4].

Aktuelle Studien belegen, dass Angriffe auf AES bis zu der 4. Und 5. Runde fast entschlüsselt werden können [58, S.339]. Da AES aber bei der kleinsten Variante (AES-128) jedoch 10 Runden durchläuft, besteht derzeit noch keine Gefahr für den Algorithmus.

Dennoch hat AES einige Nachteile, wie dass ohne eine separate Hardware-Komponente die Geschwindigkeit von diesem stark verringert wird [59, S.7]. Deshalb ist der Einsatz immer Situationsabhängig und kann in manchen Umgebungen nicht empfehlenswert sein.

Stromchiffre

Stromchiffren sind im Vergleich zu Blockchiffren etwas anders: Sie können einen Chiffretext direkt verschlüsseln, anstelle ihn erst in Blöcke aufteilen zu müssen. Das macht den Prozess der Verschlüsselung effizienter. Stromchiffren setzen die Verschlüsselung durch einen Schlüsselstrom um [103, S.320]. Dieser ist einzigartig und wird mithilfe eines Zufallszahlengenerators erstellt. Jeder Schlüsselstrom darf nur einmal verwendet werden. Die Größe der Schlüsselströme hat hierbei keinen einheitlichen Wert, sondern ist variabel.

Die Sicherheit von Stromchiffren hängt jedoch stark von der Generierung der Zufallszahlen ab [31, S.154]. Diese müssen Angreifern gegenüber geheim gehalten werden, was

schwierig sein kann. Der Einsatz von Pseudo-Zufallszahlengeneratoren zu Erzeugung von Pseudozufallszahlen kann die Sicherheit stark beeinträchtigen [31, S.155]. Das hat den Grund, dass diese im Gegensatz zu echten Zufallszahlen deterministisch sind, sie sind also nur Pseudo-Zufällig. Aktuell ist RC4 der gängigste Stromchiffre-Algorithmus [57, S.698]. Andere Algorithmen können in ihrer stärke variieren und passend nach der jeweiligen Situation eingesetzt werden. Einsatzgebiet von Stromchiffren könnten theoretisch dynamische SQL-Anweisungen sein, oder auch Passwort-Verschlüsselungen mit dem modernen SCKHA-Algorithmus, der das Hashen als zusätzliche Funktionalität anbietet [107, S1ff.], sowie dem ChaCha20-Algorithmus, der in Kombination mit dem Poly1305 Authentifizierer als besonders effizient und sicher eingestuft wird [74, S.4, 14].

Asymmetrische Verschlüsselung

Die asymmetrische Verschlüsselung funktioniert im Vergleich zu der symmetrischen Verschlüsselung etwas anders: Schlüssel werden nicht ausgetauscht, es existiert aber neben dem geheimen Schlüssel noch ein weiterer, öffentlicher, der für die Entschlüsselung benötigt wird [10, S.14]. Diese Art der Verschlüsselung wird besonders bei der Kommunikation zwischen zwei Anwendern verwendet. Sie arbeitet mit Einwegfunktionen, einem mathematischen Begriff, bei dem eine Funktion ohne die korrekten Parameter sehr schwer umzukehren ist [120, S.166]. Durch den Einsatz der privaten Schlüssel auf die öffentlichen, frei zugänglichen Schlüssel kann eine Verschlüsselte Kommunikation zwischen zwei Parteien ermöglicht werden. In Datenbanksystemen kann die asymmetrische Verschlüsselung bei der Datenübertragung sowie bei der Schlüsselverwaltung für die Symmetrische Verschlüsselung eine entscheidende Rolle spielen.

Hybride Verschlüsselung

Auf den Anwendungsfall bezogen können hybride Verschlüsselungen eingesetzt werden, die symmetrische mit asymmetrischen Methoden verbinden [120, S.174-175]. Je nach Kontext kann das die Sicherheit einer Datenbankanwendung um einiges erhöhen. Die Hybride Verschlüsselung funktioniert so, dass die Schlüssel, die bei der symmetrischen Verschlüsselung entstehen, mithilfe der asymmetrischen Verschlüsselung zusätzlich verschlüsselt werden. So kann nur mit dem privaten Schlüssel des Empfängers der symmetrische Schlüssel entschlüsselt werden, um die verschlüsselte Nachricht zu lesen. Diese Methode gilt als besonders sicher für die Kommunikation von Schlüsseln zwischen zwei Anwendern als auch für den Austausch kurzer Nachrichten.

Fazit

Bezüglich der Vertraulichkeits-Aspekte aus der Risikoanalyse sind Verschlüsslungen ganz klar effektiv, wenn auch der Einsatz einiger Methoden nicht ganz so einfach umzusetzen ist. Am gängigsten sind Blockchiffren vertreten, durch deren Einsatz der Schutz von statischen Daten gut eingerichtet werden kann. Stromchiffren werden aufgrund der beschriebenen Sicherheitsprobleme weniger häufig eingesetzt, sind aber in der Theorie potenziell stark und haben vielleicht mit besseren Algorithmen für die Generierung der Zufallszahlen in der Zukunft einen festen Platz bei der Einrichtung von Sicherheitsmaßnahmen. Asymmetrische Verschlüsselungen hingegen sind in ihrem Einsatz spezieller und können bei Bedarf genau die richtige Wahl sein.

Im Kontext der Risikobewältigung sind die vorgestellten Methoden aber ganz klar effektiv und ermöglichen eine starke Limitierung der Schwachstelle der Vertraulichkeit. Zusätzlich gehören neben Verschlüsselungen noch Zugriffskontrollen inhaltlich zu diesem Aspekt. Diese werden an dieser Stelle jedoch Aufgrund des inhaltlichen Umfanges und der übergreifenden Relevanz auf die anderen zwei Aspekte (Verfügbarkeit und Integrität) separat aufgelistet

3.2.4 Management der Identitäts- und Zugriffskontrolle

Das Management der Identitäts- und Zugriffskontrolle (auf Englisch ***Identity and Access Management (IAM))*** wird verwendet, um Berechtigungen zu verwalten [106, S.30]. Zugriffskontrollen und damit verbunden die Zugriffssteuerung sind wesentlicher Bestandteil des IAM. Dieses unterteilt sich in Authentifikation, Autorisation, Benutzerkontrollen und Benutzerverzeichnisse [53, S.12]. Im Folgenden werden Konzepte des IAM vorgestellt und genauer beleuchtet. Es ist wesentlicher Bestandteil der Informationssicherheit und spielt dementsprechend eine entscheidende Rolle in der Entwicklung von sicheren Datenbankanwendungen.

Zugriffskontrolle

Die Zugriffskontrolle kann alle drei Aspekte der Triade der Informationssicherheit vertreten, ganz stark aber die Vertraulichkeit. Die Verfügbarkeit kann beispielsweise durch zu viele Authentifikations-Methoden beeinträchtigt und die Integrität durch eine mangelnde Zugriffskontrolle mit unbefugten Datenmanipulationen geschädigt werden. Die Vertraulichkeit läuft jedoch ohne eine eingerichtete Zugriffskontrolle Gefahr, vertrauliche Daten

in Form von Expositionen preiszugeben oder überhaupt erst Manipulationen von unbefugten zuzulassen. Zugriffskontrollen stellen demensprechend sicher, dass nur autorisierte Benutzer Zugriff auf die Daten bekommen dürfen.

Um die Zugriffskontrolle zu realisieren, existieren verschiedene Prinzipien und Zugriffskontroll-Modelle. Dazu zählen das Least-Privilege-Prinzip, Zero Trust und allgemein die digitale Identität. Bezüglich der Modelle spielen Rollenbasierte- und Attributbasierte-Zugriffskontroll-Modelle eine ganz wichtige Rolle in Kombination mit verschiedenen Authentifikations-Techniken.

Least-Privilege

Das Least-Privilege-Prinzip besagt, dass ein Nutzer nicht mehr administrative Rechte besitzen sollte als die, die ihm zustehen [49, S.64]. Das bedeutet, dass wenn ein Nutzer eine Aufgabe hat, er diese mit der geringsten Menge an verfügbarer Autorität verrichten soll. Ein Beispiel kann wie folgt aussehen: Drei Nutzer existieren, N1, N2 und N3, ebenso wie zwei Privilegien P1 und P2. Wenn N1 eine Aufgabe zugewiesen werden soll, ist es besser diesem nur Privileg P1 zuzuweisen, als P1 und P2, wenn P1 ausreicht, um die Aufgabe zu erfüllen. Der DBA spielt bei der Umsetzung der Zugriffskontrolle eine entscheidende Rolle und muss darauf Achten, einem Benutzer nur die Privilegien zu geben, die er auch benötigt.

Zero Trust

Zero Trust ist ein Paradigma der Informationssicherheit, das besagt, dass die Sicherheit nie vollständig gegeben ist und immer geprüft beziehungsweise neu eingerichtet und ausgewertet werden muss [108].

Es wird an dieser Stelle erwähnt, weil nicht unbedingt die Umsetzung Pflicht ist, sondern es vielmehr in dem dafür definierten Kontext eines Paradigmas betrachtet werden sollte. Es kann als eine Denkweise angesehen werden, deren Einsatz in bestimmten Situationen hilfreich sein kann. Die Grundidee hinter Zero Trust ist, dass nur die notwendigsten Rechte dem Nutzer zugeteilt werden, dass eine Zugriffskontrolle dementsprechend stattfindet und die wichtigsten Aktivitäten auf dem Betriebssystem und im Netzwerk kontrolliert werden müssen. Zero Trust kann in einer sehr ausgeprägten Form einige Nachteile mit sich bringen, wie zum Beispiel den Verlust von Produktivität durch zu viele Kontrollen

oder den Verbrauch von sehr viel Speicherkapazität, weswegen die Umsetzung gut durchdacht werden muss.

Digitale Identität

Die Digitale Identität ist das Nachweisen einer Identität im digitalen Bereich durch das Umwandeln dieser aus der realen Welt in eine digitale Welt [64, S.1]. Eine solche Umwandlung erlaubt es, menschliche Identitäten in maschinell lesbare digitale Identitäten zu verarbeiten. Die digitale Identität ermöglicht eine komplett digitale Durchführung der Identifizierung, Authentifizierung und Autorisierung. Heute, mit der stark voranschreitenden Digitalisierung, spielt sie eine entscheidende Rolle in der Informationssicherheit. Bezüglich Datenbanksystemen ist sie ein entscheidender Faktor, vor allem vor dem Hintergrund des Zero Trust. Sie repräsentiert Benutzer, die in der Datenbank registriert sind und reguliert so die Rechteverteilung.

Authentifikation

In einem Datenbanksystem existieren Profile, die mit bestimmten Rechten versehen sind. Jeder Benutzer hat sein eigenes Profil. Damit ein Benutzer Zugang zu einem gesicherten Bereich erhält, findet eine Authentifikation statt [67, S.19]. Diese stellt sicher, dass der angemeldete Benutzer auch wirklich die Person ist, der die Rechte zugewiesen wurden. Es existieren verschiedene Methoden für die Authentifikation, darunter die DBMS-eigene Passwort-Basierte Authentifikation, das Kerberos Protokoll oder die Betriebssystem-Authentifikation. Häufig werden Kombinationen von ein- oder mehreren Authentifikations-Techniken verwendet, um auf diese Weise die Sicherheit zu erhöhen.

Single-Sign-On (SSO):

Ein Single-Sign-On Dienst ermöglicht es Benutzern, sich über den besagten Authentifikationsdienst einmalig anzumelden und anschließend ohne weitere Anmeldung auf andere Anwendungen im Netzwerk zuzugreifen [8, S.18]. Durch eine einmalige Legitimation der Anmeldedaten wird die Anmeldung für Anwender erleichtert und weniger Spielraum für Fehler gelassen. Durch das SSO kann das Auditing, also die Überwachung von Benutzeraktivitäten, vereinfacht und eine zentrale Benutzerverwaltung eingeführt werden. Das hat den Vorteil, dass Rechte übersichtlich eingerichtet werden können.

3.2.5 Rollenmodelle

Rollenkonzepte befassen sich mit der Vergabe von Rechten an einen Benutzer und gehören zur Zugriffskontrolle. Sie unterteilen sich in 4 Modelle:

- Das diskrete Sicherheitsmodell [47, S.495]
- Das Obligatorische Sicherheitsmodell [47, S.495]
- Die Rollenbasierte Zugriffskontrolle und [94, S.36]
- Die Attribut basierte Zugriffskontrolle [48, S.6]

Der Einsatz von Rollenkonzepten ist neben den anderen Aspekten des IAM von entscheidender Bedeutung, um die mit der Integrität und der Vertraulichkeit verbundenen Risiken zu minimieren. Kapitel 4 geht näher auf dieses Thema ein.

Fazit

Bei dem Einrichten von sicheren Datenbankanwendungen spielen die vorgestellten Konzepte und Modelle eine entscheidende Rolle. Durch ihre Erforschung und ihren Einsatz kann die Gefahr, alle drei Aspekte der Informationssicherheit im Umfeld einer Datenbankanwendung zu verletzten, stark verringert werden. Ohne eingerichtete Zugriffssteuerungen sind potenzielle (gewollte oder ungewollte) Gefahren durch beteiligte Personen zu hoch. Somit sind sie für die Einrichtung sicherer Datenbankanwendungen notwendig und Teil der Risikobewältigung.

Abschließend kann behauptet werden, dass die identifizierten Risiken durch die Erforschung und den Einsatz der passenden Konzepte, Methoden und Modelle minimiert sowie limitiert werden kann.

4 Einführung in Prozedurale Erweiterungen & Einsatz der Zugriffskontrolle

Nach dem Einsatz des Risikomanagements und den vorgestellten Modellen und Konzepten zu der Risikobewältigung erfolgt nun der Einsatz von Sicherheitsmaßnahmen. Die hierfür notwendige prozedurale Erweiterung von SQL soll in diesem Kapitel vorgestellt werden, um basierend darauf Modelle für die Zugriffskontrolle zu besprechen.

4.1 Einführung in Prozedurale Erweiterungen von SQL

Um den Einsatz von PL/SQL zu verdeutlichen, wurde eine provisorische Datenbank mit Oracle SQL Developer entwickelt. Diese soll die prozedurale Erweiterung von Oracles PL/SQL verwenden, um Sicherheitsmaßnahmen zu implementieren. Diese Datenbank dient lediglich zur Vorstellung von einigen, aber nicht allen Maßnahmen. Der komplette Quellcode wird im Anhang präsentiert. Bei der Erstellung der Datenbank spielt die Datenbankkonfiguration eine wichtige Rolle, in der Einstellungen bezüglich der Authentifizierung und den Zugriffsrechten eingerichtet werden können.

4.1.1 Datenbankkonfiguration

Um eine Datenbank auf der Oracle Plattform erstellen zu können, braucht der Entwickler die passende integrierte Entwicklungsumgebung (IDE). Oracle bietet hierfür den SQL Developer an. Nach einer erfolgreichen Installation von Oracle SQL Developer Version 23.1.1 kann der Anwender eine Datenbank hinzufügen. Wenn er sich hierfür entscheidet, wird er aufgefordert, der Datenbank einen Namen zu geben. Diese ist für die eindeutige Identifizierung der Datenbank wichtig. Nach der Namensvergabe kann der Anwender den Authentifizierungstypen auswählen: Oracle bietet hierfür das Passwort basierte Authentifizierungs-Modell, die Betriebssystem-Authentifizierung und Kerberos als Authentifizierungsprotokoll an.

Anschließend können zusätzlich noch Rollen ausgewählt werden, die der Anwender einnehmen kann. Hierbei ist normalerweise die Rolle Standard als Anfangswert eingerichtet. Diese hat grundlegende Berechtigungen, die zusätzlich von dem DBA erweitert oder beschränkt werden können. Andere Rollen in dem SQL Developer der Version 23.1.1 sind SYSDBA, SYSOPER, SYSBACKUP, SYSDG, SYSKM, SYSASM. Jede dieser Rollen hat eigene, spezifische Rechte. Wenn eine Anmeldung mit den Root-Anmeldedaten

erfolgt, wird der Benutzer als SUPERUSER angemeldet [87]. Der SUPERUSER kann einen DBA erstellen und die Verteilung von beliebigen Rechten an beliebige Rollen übernehmen.

Daraufhin kann der Verbindungstyp ausgewählt werden. Dieser bezieht sich auf die Art der Verbindung, die in Oracle SQL Developer Version 23.1.1 für den Zugriff auf eine Datenbank verwendet wird. Zur Auswahl stehen die Verbindungstypen: Einfach, Cloud Wallet, LDAP, Benutzerdefinierte JDBX-Verbindung, SSH, TNS. Die Auswahl des Verbindungstypen ist umgebungsabhängig und kann situationsbedingt entschieden werden. Eine Einfache Verbindung ist eine übliche TCP/IP Verbindung, was den Vorteil eines geringen Aufwandes für die Einrichtung mitbringt. Bei dem Einsatz einer Cloud kann das Cloud Wallet ausgewählt werden. Wenn ein Verzeichnis integriert werden soll, wäre LDAP die bessere Wahl etc.

Abschließend werden Hostname und Port sowie System Identifier (SID) verlangt, damit die Verbindung zu der Datenbankinstanz hergestellt werden kann. Die Einrichtung einer Verbindung ermöglicht es einem Anwender, eine Datenbank zu erstellen.

4.1.2 Datenbankadministrator (DBA)

Eine DBS benötigt immer einen Datenbankadministrator (DBA). In Oracle SQL-Developer ist das die Rolle des SYSDBA. Dieser ist dafür zuständig, die administrativen Aufgaben eines DBS zu übernehmen. Dazu gehören: Festlegen der physischen Speicherstrukturen und Zugriffsmechanismen, beraten des Anwenders, definieren von Zugriffsberechtigungen, Integritätsregelungen, Datensicherungs- und Rücksicherungsroutinen sowie das Überwachen der Geschwindigkeit des Datenbanksystems [100, S.26]. Die Befehle zur Zugriffssteuerung kann somit nur ein DBA (oder ein SUPERUSER) definieren. Die Rolle des DBA wird meistens bei dem ersten Login von dem SUPERUSER zugewiesen [87].

4.1.3 Datenabfrage/SELECT-Abfragen

In SQL werden für die Arbeit mit den vorhandene Datensätzen SELECT-Anweisungen verwendet. Diese werden im Kontext einer Abfrage (*Query*) ausgeführt [101, S6]. SELECT-Abfragen sind grundlegender Bestandteil von Standard-SQL und dienen dazu, Daten aus einer Datenbank abzurufen und Geschäftslogik umzusetzen. Sie können JOINS

verwenden, um unterschiedliche Tabellen miteinander zu verknüpfen [46, S.133]. Ein Beispiel für eine einfache SELECT-Abfrage kann so aussehen:

```
SELECT mitarbeiter_id, nachname, geburtsdatum

FROM ut_mitarbeiter

GROUP BY mitarbeiter_id, nachname, geburtsdatum

HAVING (SYSDATE - geburtsdatum) > 35*365;

ORDER BY mitarbeiter_id ASC;
```

Codebeispiel 1 (eigener Code)

Die abgebildete Anweisung sucht alle Mitarbeiter aus der Tabelle ut_mitarbeiter raus und filtert sie nach dem Alter über 35 aus. Daraufhin gibt sie diese aufgelistet in Form von einer neuen Tabelle mit den Ergebniswerten aus.

Mit SELECT-Anweisungen können Zugriffskontrollen und Bedingungen für die Vertraulichkeit eingerichtet werden. Somit tragen Abfragemöglichkeiten zu der Gewährleistung der Sicherheit von Datenbanken bei. In Kombination mit Prozeduren, Funktionen und Triggern bilden sie einen wesentlichen Aspekt der Sicherheit von Datenbankanwendungen. Zusätzlich ermöglichen sie auch den Einsatz von Datenbanksichten, auch genannt Views.

4.1.4 Datenbanksichten (Views)

Zum Bewahren der Vertraulichkeitsaspekte können Sichten (Views) eingesetzt werden. Diese sind gespeicherte SELECT-Anweisungen, die eine virtuelle Tabelle erstellen und neben der eigentlichen Definition keinen zusätzlichen Speicherplatz innerhalb des Datenbanksystems benötigen [39, S.156]. Sie erlauben eine höhere Übersichtlichkeit von verknüpften Tabellen und können durch die Implementierung zusätzlicher Zugriffskontrollmaßnahmen die Rechte für Benutzer einschränken. Das gelingt, indem sie mehrere Tabellen in einer einzelnen virtuellen Tabelle zusammenfassen können. In Datenbanksystemen können Views erzeugt werden, die sich gegenseitig referenzieren [16, S.1-2].

Der Einsatz von Views kann bezüglich der Sicherheitsanforderungen dazu beitragen, die Integrität der Referenz-Tabelle zu bewahren. Sie können verwendet werden, damit keine ungewollten Änderungen auf der eigentlichen Tabelle stattfinden und abfragen über die View geschehen. Negativ wirkt sich jedoch das dynamische Erstellen dieser Views auf

die Sicherheitsaspekte von Datenbankanwendungen aus: Es kann sein, dass ein Verlust der Zugriffskontrolle durch die ständige Erzeugung neuer Views einhergehen kann.

Bei einem strategischen Einsatz von Views werden oft Materialized Views verwendet, weil diese ähnlich wie normale Tabellen in den Datenbanken selbst gespeichert werden können und es dementsprechend mehr Sinn macht, Zugriffsrechte auf diese Art von Views zu vergeben.

Materialzied View

Eine View kann in Form von einer Materialized View in physischer Form abgespeichert und somit „Materialisiert“ werden. Abfragen können ganz einfach über die Materiazied View erfolgen, ohne die Ursprungstabelle zu verändern [16, S.1]. Das hat den Vorteil, dass keine Leistungseinbuße durch ständige Abfragen geduldet werden müssen. Sie sind eine Erweiterung von Views und können die referenzierten Tabellen ersetzen.

Der Nachteil ist aber, dass die erstellten Materialized Views im Datenbankpuffer gespeichert werden und dadurch zusätzlichen Speicher verbrauchen [39, S.163; 47, S.53-54]. Die Kapazität des Puffers ist jedoch begrenzt. Bei dem Einsatz von zu vielen Sicherheitsmaßnahmen kann es passieren, dass dieser überlastet wird. Deswegen sollte der Entwickler der Datenbankanwendung wie auch der DBA Situationsbedingt einschätzen, an welcher Stelle es Sinn macht, Materialized Views zu verwenden.

Durch den Einsatz von Views kann die Verwaltung von Berechtigungen auf Tabellen eingeschränkt werden [60, S.237-238.]. Es können Rechte auf eine View vergeben werden, ohne Rechte auf die Referenztabelle zu vergeben, wodurch nur gewisse, ausgewählte Attribute auf der View manipuliert oder betrachtet und andere aus der Ursprungstabelle verborgen gehalten werden können.

4.1.5 Prozeduren und Funktionen

Prozeduren und Funktionen können als benutzerdefinierte Routinen bezeichnet werden [114, S.134-136.]. Diese sind Unterprogramme, die eine Reihe vordefinierter SQL-Anweisungen enthalten. Prozeduren wie auch Funktionen werden im Datenbankpuffer gespeichert und können im Laufe einer Sitzung immer wieder verwendet werden [2, S.327].

Jede Routine besteht aus einer Signatur und einem Rumpf. Die Signatur enthält den Namen der Routine und die Parameter [114, S.134-136]. Im Rumpf einer Routine steht

der Programmcode, der eine bestimmte Funktionalität hat. Allein diese Zusammensetzung kann bereits eine Anwendungsfunktionalität haben und wird als Anonymous Block bezeichnet [35, S.307].

Funktionen beziehungsweise Prozeduren funktionieren so, dass der Code geschrieben, kompiliert und ausgeführt wird, bevor er von der Webanwendung verwendet werden kann, um mit der Datenbank zu kommunizieren. Durch die Speicherung in der Datenbank selbst wird die Sicherheit des Anwendungsprogrammcodes verbessert, da eine weitere Abstraktionsschicht vor der Interaktion mit der Datenbank hinzugefügt werden kann. Häufig verwendete Abfragen können in einer Prozedur gespeichert und bei Bedarf abgerufen werden [37, S.415-416.]. Durch den Einsatz von Prozeduren kann eine hohe Sicherheit der Datenbankanwendung garantiert werden.

Oracle hat den Vorteil, dass alle Prozeduren direkt in der Datenbank selbst gespeichert werden, wohingegen andere DBMS diese erst als gespeicherte Prozedur deklarieren müssen [84].

Hauptsächlich unterscheiden sich Prozeduren und Funktionen nur darin, dass eine Funktion einen Rückgabewert besitzt und eine Prozedur nicht. Der Einsatz dieser kann in vielerlei Hinsicht hilfreich sein und die Umsetzung von bestimmten Anforderungen flexibel einrichten. Sie werden bei Integritätskontrollen, Zugriffskontrollen oder für standardisierte Abfragen verwendet und bieten die Option für individuelle funktionale Erweiterungen, die nach den Vorstellungen des Entwicklers gestaltet sein können.

4.1.6 Trigger

Trigger sind speziell gespeicherte Prozeduren oder Funktionen, die eine automatische Reaktion einer Datenbank auf benutzerdefinierte Ereignisse ermöglichen [39, S.198]. Trigger haben den Vorteil, dass sie dynamisch anpassbar sind. Sie können Bedingungen enthalten, die die gesamte Anwendungslogik der Datenbank ändert, aber sie können auch ohne weiteres entfernt werden, ohne die Funktionalität dieser zu beeinflussen. Der Hauptgrund für Trigger ist eine Abbildung komplexer Geschäftsregeln, die über andere Integritätsbedingungen nicht abgebildet werden können [20, S.303-304.].

Das Einsatzgebiet von Triggern ist groß und die genaue Implementierung dem Entwickler überlassen. Sie erlauben Integritäts- und Zugriffskontrollen und können komplexe Anwendungslogik umsetzen. Trigger können so gestaltet werden, dass sie entweder vor oder nach einer Aktion feuern. Zusätzlich kann die genaue Aktion als DML-Befehl spezifiziert werden (Insert, Update, Delete) [20, S.304]. Der innerhalb des Triggers

eingebettete Anonymous Block kann verwendet werden, um eine prozedurale Logik zu erstellen. Ein möglicher Einsatz dieser Logik kann dann beispielsweise das Auditing sein, bei dem nach einer bestimmten Aktion die durchgeführten Änderungen sowie die Informationen von dem Benutzer in einer separaten Tabelle automatisch hinterlegt werden.

Wichtig bei Triggern ist, dass sie gut programmiert werden und eine passende Terminierung haben. Schlimmstenfalls können sie sich gegenseitig rekursiv aufrufen und zu einer Reihe von Problemen führen.

Bei der Frage an welcher Stelle Trigger Prozeduren oder Funktionen vorgezogen werden sollen, ist die Antwort meistens, dass es von der jeweiligen Situation abhängt. Trigger sind sehr hilfreich dabei, komplexe Integritätsregeln umzusetzen und können auch erweiterte Funktionalitäten bereitstellen, wie das Audit-Trailing von einzelnen Tabellen [46, S. 362]. Jedoch kann es Situationen geben, in denen die automatisch feuernden Aktionen nicht unter den jeweiligen Umständen als angebracht betrachtet werden können. Das wären Situationen, in denen der Entwickler dann auf Prozeduren oder Funktionen zurückgreifen würde.

4.1.7 Packages/Pakete

Ein "Package" (Paket) kann als ein Container für Prozeduren, Funktionen, Konstanten, Variablen etc. betrachtet werden [36, S.129]. Pakete bilden eine Zusammensetzung aus Code-Elementen, durch deren Einsatz die Nutzung von Code stark vereinfacht wird [79]. Sie helfen bei der Wartung durch eine höhere Übersichtlichkeit und können dafür sorgen, dass Code nach einmaligem kompilieren im Datenbankpuffer zwischengespeichert wird [36, S.131-132.]. Ihr Einsatzgebiet sind funktionale Erweiterungen, die bereits existierende Prozesse und Funktionen nutzen sollen, um komplexe Geschäftslogik wie beispielsweise Verschlüsselungen umzusetzen.

4.2 Einsatz der Zugriffskontrolle

Die Zugriffskontrolle ist wichtig, um verschiedene Angriffe von außen sowie auch internen Missbrauch zu vermeiden. Dieser Abschnitt befasst sich mit der Data Control Language (DCL). Sie ist Teil des Datenbank-Security-Managements und kann durch die Vergabe von Zugriffsrechten einen internen Schutz sicherstellen [51, S.167-168]. In Kombination mit der Authentifizierung ist die Zugriffskontrolle Kernbestandteil von

sicheren Datenbankanwendungen und reguliert die Rechte, die einem Anwender innerhalb dieser zustehen.

In SQL wird die Zugriffskontrolle durch das Einrichten von Benutzern umgesetzt, denen gewisse Rechte zugewiesen werden können. Die Benutzererstellung- und Verwaltung wird von dem DBA eingestellt. Die Zugriffskontrolle lässt sich grob in zwei Modell-Kategorien einteilen: Das sind die statischen- und dynamischen Modelle der Zugriffskontrolle [6, S.188].

4.2.1 Statische Modelle der Zugriffskontrolle

Der Begriff Statisch bezeichnet weniger flexible Modelle mit einer bestimmten Rollen- und Rechteverteilung. Vorteilhaft sind sie Aufgrund einer guten Übersichtlichkeit und weitreichender Sicherheit, die durch individuell definierte Bedingungen umgesetzt werden kann.

Rollenbasierte Zugriffskontrolle (RBAC)

Die Rollenbasierte Zugriffskontrolle wird durch die Rollenvergabe an Benutzer umgesetzt. Es beinhaltet die 5 Konzepte Benutzer, Rolle, Objekt, Operationen und Rechte [94, S.16]. Hierbei werden Benutzer angelegt, die bestimmte Rechte haben und Rollen zugeordnet werden können. Auch können Gruppen erstellt werden, denen die Benutzer mit ihren Rollen zugewiesen werden, um die Rechtevergabe zu vereinfachen [113, S.69]. Weiterhin können Untergruppen existieren, die Teil einer Hauptgruppe sind, wodurch eine hierarchische Struktur eingerichtet werden kann. Für diese Rollen-Vergabe sind administrative Rechte notwendig [113, S.70-74]. Somit ist der DBA für die Rechte und Rollenverteilung zuständig. Aspekte aus anderen Modellen können in das RBAC integriert werden, um es zu erweitern.

Welcher Benutzer welche Rolle zugewiesen bekommt, ist häufig in Organisationen über bereits existierende Benutzer-Berechtigungs-Zuweisungen geregelt. Diese werden verwendet, um passende Rollen zu definieren. Dieser Prozess wird auch als Role-Mining bezeichnet [23, S.24-25]. Das Role-Mining ist dementsprechend hilfreich, um bereits bestehende Strukturen in die Zugriffssteuerung der Datenbank zu integrieren, was auch ein Grund für die Beliebtheit des RBAC ist. Im Folgenden werden die Bestandteile eines RBAC-Modells vorgestellt.

Benutzer

Oracles PL/SQL bietet Möglichkeiten zum Anlegen neuer Benutzer seitens des DBA an. Hierbei kann mit einfachen Befehlen wie CREATE USER ein Benutzer angelegt werden. Ein Beispiel hierfür wäre:

```
CREATE USER user_1 IDENTIFIED BY passwort123;
```

Codebeispiel 2 (eigener Code)

Dieser Befehl erstellt in Oracle SQL einen Nutzer, dem anschließend Rechte zugewiesen werden können [38, S.267]. Diese Rechte können allgemeiner Natur sein oder sich auf gewisse Tabellen oder sogar Spalten beziehen.

Rechte

Die Vergabe von Rechten (Privilegien) an Benutzer oder Rollen erfolgt mit den GRANT-Befehl (für die Zuweisung von Rechten) und dem REVOKE-Befehl (Für die Entnahme von Rechten) [39, S.167]. Zwei Beispiele sollen diese Prozesse verbildlichen:

```
GRANT privileg_Bezeichnung TO ut_MA_1;
```

Codebeispiel 3 (eigener Code)

Entzogen kann das Recht dann wieder folgendermaßen:

```
REVOKE privileg_Bezeichnung FROM ut_MA_1;
```

Codebeispiel 4 (eigener Code)

An der Stelle „privileg_Bezeichnung“ können unterschiedliche Privilegien eingesetzt werden, wie DML- oder DQL-Berechtigungen. Ein Benutzer kann mehrere Berechtigungen haben. Diese Privilegien können dem Benutzer direkt oder über Rollen zugewiesen werden [38, S.267].

Profile

Profile werden einem Benutzer beim Erstellen automatisch zugewiesen. Sie sind für die Passwort-Policy zuständig und können Datenbankressourcen beschränken [38, S.272]. Dieses Profil kann dann in seiner Einsatzfähigkeit mit dem Befehl LIMIT eingeschränkt werden [88]. Das Einsatzgebiet von Limit ist weitreichend: Ein Beispiel kann sein, dass es so eingerichtet wird, dass nach einer bestimmten Anzahl an Tagen ein Passwort geändert werden muss. Weitere Beispiele wären:

- Die maximale Anzahl an Zeichen für das Passwort
- Die Länge des Passworts
- Maximale Anzahl an Anmeldungen etc.

[88].

Die Möglichkeiten hier sind zahlreich und können die Sicherheit der Datenbankanwendung um ein Vielfaches verstärken. Andere DBMS bieten in diesem Kontext ähnliche Lösungen.

Rollen

Rollen sind Gruppierungen mit bestimmtem Rechten. Sie sind Kernbestandteil des RBAC und spielen eine wichtige Rolle bei der Zugriffsverwaltung, indem sie eine Benutzer-unspezifische Privilegien-Vergabe ermöglichen. Eine Rolle kann wie folgt definiert werden:

```
CREATE ROLE ut_mitarbeiter;
```

Codebeispiel 5 (eigener Code)

Der Vorteil von Rollen ist eine übersichtliche Verwaltung der Rechte. Es ist stark zu empfehlen, dass Privilegien nur über Rollen zugewiesen werden und nicht direkt an den Benutzer selbst, so dass das Prinzip des Least-Privilege erfüllt ist. Das Verwenden von Benutzer-Profilen eingeteilt in Rollen ist bezüglich der Zugriffskontrolle maßgebend und sollte bei der Einrichtung der Datenbankanwendung berücksichtigt werden.

Diskrete (DAC) und Obligatorische (MAC) Zugriffskontrolle

Bei der DAC werden Regeln für den Zugriff auf Objekte auf der Basis der Identität des Benutzers festgelegt [47, S.495]. DAC allein wird in Datenbanken selten verwendet und tritt viel mehr im Kontext des RBAC auf. Die Erstellung von Rollen und Gruppen und die Zuweisung von Privilegien an diese setzen das DAC-Konzept um.

Die obligatorische Sicherheitskontrolle (MAC) ordnet Subjekte und Objekte sogenannten Sicherheitsklassen zu [47, S.495]. Nutzerklassen wird somit ein Zugriff auf klassifizierte Informationen gewährt. Dieser Ansatz wird auch als Sicherheitsklassenansatz bezeichnet. Hierbei werden allen Entitäten innerhalb einer Datenbank Sicherheitsklassen zugewiesen. Mögliche Sicherheitsklassen sind: öffentlich < vertraulich < geheim < streng geheim [40, S.10].

Die genauen Kriterien werden durch den Administrator festgelegt und können nicht verändert werden. Durch die festgelegte Einordnung von Benutzern in Sicherheitsklassen können diese nur mit Objekten Interagieren, bei denen entsprechende Rechte ausreichen. Diese Form gehört zu den sichersten Zugriffssystemen, hat aber den Nachteil, dass sie sehr starr ist und dadurch wenig Flexibilität in der Praxis zulässt.

Der Einsatz von MAC ist in einer Oracle Umgebung nicht so einfach. Die Umsetzung von MAC kann in Oracle Database über die Erweiterung Oracle Label Security mithilfe von Trusted Extensions erfolgen [89]. Andere DBMS haben ähnliche Erweiterungen zur Umsetzung dieser Prinzipien. Durch den Einsatz von Erweiterungen können höhere Sicherheitsstandards gesetzt werden. Häufig reicht jedoch das RBAC aus, um eine starke Zugriffssteuerung zu implementieren.

4.2.2 Dynamische Modelle der Zugriffskontrolle

Dynamische Modelle der Zugriffskontrolle sind Modelle, die ein hohes Grad an Flexibilität aufweisen. Sie sind sehr zu empfehlen, wenn eine besonders sichere Zugriffskontrolle etabliert werden soll, die stark flexibel und außerordentlich sicher ist und auf Attributen und anderen Feingranularen-Eigenschaften aufbaut. Dynamische Berechtigungen ermöglichen es, mithilfe von Prozeduren, Funktionen und Triggern bestimmte Zugriffslogik zu implementieren, um eine individuelle Einrichtung der eigenen Vorstellungen für die Zugriffssteuerung zu ermöglichen.

Hauptsächlich besteht das dynamische Modell aus der Attributbasierten-Zugriffskontrolle und damit verbrunden der Row-Level-Security sowie der Feingranularen-Zugriffskontrolle.

Attributbasierte-Zugriffskontrolle/Attribute Based Access Control (ABAC)

Attributbasierte-Zugriffskontrolle (ABAC) verwendet beliebige Eigenschaften von Entitäten zur Formulierung von Zugriffsberechtigungen. Es repräsentiert ein Spektrum von logischen Zugriffskontrollen über einfache Zugriffsteuerungslisten, die zu der hochgradig flexiblen Methode des ABAC erweitert werden können [48, S.6].

Eine Zugriffsentscheidung in ABAC (Attribute-Based Access Control) erfolgt, indem die Attribute eines Benutzers gewisse Zugriffsrichtlinien erfüllen. Die Beschränkung der

Erweiterbarkeit traditioneller Modelle in mehreren Domänen kann in ABAC durch eine Auswahl geeigneter Attribute für diese Domänen überwunden werden [56, S.61-62].

ABAC kann entweder manuell oder über die von einem Anbieter bereitgestellten Funktionen eingerichtet werden. Für die Rechteverteilung wird ein Zugriffsrichtlinien-Mechanismus verwendet [68, S.1-2]. Um Richtlinien zu entwerfen, müssen alle notwendigen Informationen diesem Mechanismus bereitgestellt werden. Der Einsatz von externen Authentifikationsmethoden kann jedoch beispielsweise wichtige Informationen verbergen [68, S.1-2]. Somit ist ABAC nicht in allen Umgebungen ohne weiteres anwendbar.

Bei der Manuellen Variante können mithilfe von PL/SQL-Funktionen und Prozeduren Zugriffsrichtlinien implementiert werden. Das gelingt in SQL, indem abfragen programmiert werden, die sich auf die Attribute beziehen. Durch einen gezielten Einsatz kann dann der Zugriff auf ein Objekt über die Attribute ermöglicht werden.

Viele DBMS-Anbieter haben aber auch eigene Erweiterungen für die Zugriffskontrolle entwickelt. Bei Oracle Database wäre das beispielsweise das Virtual Private Database (VPD). Dieses ermöglicht einen einfachen und automatisierten Ablauf der Zugriffskontroll-Verteilung. VPD ist eine Funktion von Oracle Datenbanken, die durch den Einsatz von Richtlinien auf Tabellen, Views oder Synonymen eine Zugriffskontrolle implementiert [38, S.277-278]. Sie fügt dynamisch WHERE-Klauseln zu jedem SQL-Statement hinzu, das gegen eine Tabelle oder eine Ansicht verwendet wird, um festzulegen, welche Zeilen und Spalten einem bestimmten Benutzer angezeigt werden dürfen [109, S.211].

Mithilfe von VPD können Richtlinien definiert werden, die auf Attribute basieren, wodurch das Implementieren einer Feingranularen-Zugriffskontrolle vereinfacht wird. Diese Richtlinien können dann verwendet werden, um eine automatisierte Attributvergabe für Attributbasierte Zugriffskontrollen (ABAC) einzurichten.

Wenn entschieden werden soll, ob das manuelle Einrichten oder das automatisierte Zuweisen der Zugriffsrechte besser ist, kann gesagt werden, dass automatisierte Prozesse manuellen bevorzugt werden sollten, wenn es allein um die Informationssicherheit geht. Jedoch ist auch anzumerken, dass die manuelle Variante ein hohes Maß an Flexibilität bei der Einrichtung und Erweiterbarkeit bietet und den Wünschen der jeweiligen Organisation besser entsprechen kann.

Sicherheit auf Zeilen-Ebene/Row-Level-Security (RLS)

Row-Level-Security (RLS) ist das Vergeben von Zugriffsrechten auf verschiedene Datensätze auf Zeilen- und Spaltenebene [92, S.281]. Es ermöglicht das granulare Einrichten von Sicherheitsvorgaben und vergibt Zugriffsrechte basierend auf Benutzerprofilen. RLS ist demnach eine hilfreiche Zugriffskontrolle und kann mit RBAC und ABAC kombiniert werden. Die Kombination von mehreren Methoden für die detaillierte Zugriffssteuerung wird in der Implementierung als Feingranulare Zugriffskontrolle bezeichnet.

Feingranulare-Zugriffskontrolle/Fine Grained Access Control (FGAC)

Die Feingranulare Zugriffskontrolle ermöglicht es dem System, verschiedenen Benutzern innerhalb derselben Gruppe unterschiedliche Zugriffsrechte zu gewähren [3, S.749].

FGAC befasst sich mit einer detaillierten, dynamischen Zugriffskontrolle und wird von PL/SQL unterstützt. Es ist eng mit der RLS verbunden. Die Umsetzung erfolgt in Form von Richtlinien. Diese Richtlinien hängen nicht nur von statischen Informationen ab, nämlich den Zuweisungen von Benutzern und Berechtigungen zu Rollen, sondern auch von dynamischen Informationen, nämlich der Erfüllung von Autorisierungsbeschränkungen durch den aktuellen Zustand des Systems [7, S.2-3].

Die Feinabstimmung der Zugriffskontrolle wird ermöglicht, indem dynamische Informationen in den Sicherheitsrichtlinien berücksichtig werden. Hierdurch können kontextsensitive Entscheidungen getroffen werden.

Einsatz der passenden Methoden

Zusammengefasst bietet die Oracle PL/SQL Umgebung viele Methoden für die Zugriffskontrolle. Der genaue Einsatz ist Situationsabhängig und kann nicht pauschal definiert werden. Es kommt auf die Anforderungen der Anwender an, wobei aber eines der Methoden immer aktiv integriert sein sollte, um eine sichere Datenbankanwendung zu entwickeln.

Alle vorgestellten Ansätze haben ihre Vor- und Nachteile. Diese werden im Folgenden präsentiert:

So kann zum Beispiel in einem Unternehmen, das bereits eine gewisse Struktur hat, das RBAC verwendet werden, um diese auch in dem Datenbanksystem zu etablieren. Das einfache Abbilden von Hierarchien und übernehmen bereits vorhandener Richtlinien macht es zu einer beliebten Option für die Zugriffskontrolle.

Wenn die Organisation eine höhere Flexibilität bei der Zugriffskontrolle wünscht, so dass auch Objekte in anderen Zuständen berücksichtigt werden, kann ABAC bevorzugt werden. ABAC kann in komplexen Systemen schwieriger zu implementieren sein, bietet dafür aber einen besonders hohen Schutz durch die Definition der Zugriffskontrolle basierend auf Attributen. Zusätzlich bieten Funktionen wie VPD, die in ähnlicher Form auch bei anderen Anbietern vertreten sind, eine simplere Umsetzung.

FGAC ist noch eine Ebene feiner als ABAC und bietet die Möglichkeit von granularen Zugriffsregeln auf einzelne Datensätze. Vor allem in Kombination mit RLS bildet FGAC das feinste Modell der Zugriffskontrolle und kann besonders hohen Ansprüchen gerecht werden. Allerding ist damit auch ein hoher Verwaltungsaufwand und eine besonders hohe Komplexität verbunden.

5 Einsatz von Schutzmaßnahmen

Cyberattacken können mit unterschiedlichen Methoden durchgeführt werden. Einige dieser Methoden sind Man-in-the-Middle Angriffe, SQL-Injektionen, Passwort Attacken, Cross-site Scripting (XSS) und allgemeine Malware Angriffe [11, S.4849-4851]. Das Behandeln aller Angriffsmöglichkeiten im Detail würde die Kapazität der Arbeit überlasten, weswegen im Folgenden der Fokus auf SQL-Injektionen und das Einrichten der in Kapitel 3 definierten Maßnahmen gesetzt wird. Jedoch kann behauptet werden, dass die eingesetzten Maßnahmen eingeteilt in Integrität, Verfügbarkeit und Vertraulichkeit einen guten grundlegenden Schutz gegen diese und diverse andere Angriffe bieten.

5.1 Einhalten der Verfügbarkeit

Um die Verfügbarkeit eines Datenbanksystems zu gewährleisten, müssen die Rahmenbedingungen korrekt eingerichtet werden. Zu den Rahmenbedingungen zählen die Betriebssystem- und Netzwerksicherheit. Zusätzlich werden Backups und Audits betrachtet und die Implementierung dieser wird besprochen. Eine solide eingerichtete Verfügbarkeit kann dazu beitragen, Angriffe wie DDOS oder Malware-Attacken zu vermeiden.

5.1.1 Arten von Sicherheitsmaßnahmen

Sicherheitsmaßnahmen können in ihrer Art und ihrem Einsatzgebiet variieren. Die wichtigsten Komponenten in diesem Kontext sind die Betriebssystem- und die Netzwerksicherheit. Bezüglich dieser ist der Einsatz von präventiven-, Detektiven(passiven)- und kompensative-Maßnahmen von großer Bedeutung [45, S.29].

Kompensative Maßnahmen

Kompensative Maßnahmen haben die Aufgabe, eingetretenen Schaden zu limitieren und das Ausmaß somit zu begrenzen [45, S.29].

Eine gute kompensative Maßnahme für die Netzwerk und Betriebssystemsicherheit ist der Einsatz einer Sandbox: Eine Sandbox schränkt das Netzwerk ein, auf das zugegriffen werden kann [118, S.211]. Sie kann zusätzlich auch als eine präventive- und detektive Maßnahme betrachtet werden. Über eine Sandbox wird festgelegt, auf welche

anderen Entitäten wie Verzeichnisse, Bibliotheken oder Server ein Zugriff gestattet ist. Hierdurch kann bei einem Angriff nur innerhalb dieser Sandbox Schaden angerichtet werden und andere wichtige Entitäten bleiben unbetroffen.

Präventive Maßnahmen

Präventive Maßnahmen sollen vorbeugend die Eintrittswahrscheinlichkeit eines Schadens reduzieren [45, S.29].

Eine Möglichkeit für eine präventive Maßnahme sind Firewalls [118, S.205]. Diese sind im Allgemeinen dafür zuständig, den Zugriff auf das Netzwerk zu kontrollieren. Sie überwachen Verbindungen durch eingestellte Zugriffsbedingungen. In SQL können Firewalls aber auch über die Konfigurations-Einstellungen eingerichtet werden. Das ermöglicht eine Unterteilung in Datenbank- und Webanwendungs-Firewalls.

Datenbankfirewall

Die Datenbankfirewall ist ein Datenbank-Proxyserver, der sich zwischen Datenbank und Anwendung befindet. Hierzu nimmt die Anwendung erst einmal eine Verbindung mit der Firewall auf. Daraufhin können Abfragen gesendet werden, so als wäre die Anwendung direkt mit der Datenbank verbunden [17, S.533]. Falls die Anfrage von der Firewall als sicher eingestuft wird, kann sie so an den Datenbankserver weitergegeben werden. Ist die Anfrage schädlich, verhindert die Firewall ihre Ausführung. Zum Schutz vor SQL-Injektionen aber auch vor anderen Angriffen wie Malware-Attacken sind Datenbankfirewalls sehr wirkungsvoll. Es können Regeln aufgestellt werden, die bei ungewöhnlichen Abfragen gewisse Maßnahmen ergreifen. Oracle speziell bietet in diesem Kontext eine eigene Lösung mit dem Namen Audit Vault and Database Firewall (AVDF) an [89]. Andere Datenbank-Plattformen stellen ähnliche Lösungen bereit.

Webanwendungsfirewalls (WAF)

Das sind entweder Netzwerkgeräte oder Softwarelösungen, die eine Webanwendung mit zusätzlichen Sicherheitsfunktionen unterstützen sollen [17, S.517]. Diese betreffen generell die Netzwerksicherheit und sollten in jedem Fall eingesetzt werden. Ähnlich wie Datenbank-Firewalls bieten sie Schutz vor verschiedenen Angriffen, nur betrifft dieser Schutz nicht nur die Datenbank-Sicherheit, sondern auch die Betriebssystem- und Netzwerksicherheit.

Detektive Maßnahmen

Detektive Maßnahmen sind dazu gedacht, Schadens- oder Risikoeintritt zu erkennen und zu protokollieren [45, S.29].

Eine detektive- sowie kompensative-Maßnahme ist der Einsatz eines Intrusion Detection System (IDS) [118, S.206]. Das sind Anwendungen, die im Hintergrund laufen und den Zweck haben, Angriffe zu erkennen, zu protokollieren und sie zu melden. Ihr Nachteil ist aber, dass sie nicht aktiv gegen erkannte Angriffe vorgehen können.

Verschiedene Standards wie die ISO/IEC27002 behandeln dieses Thema weit ausführlicher und bieten gute Vorgaben zum Einrichten eines sicheren Umfeldes und damit verbunden der Einhaltung der Verfügbarkeit.

5.1.2 Die Implementierung von Audits

Der Einsatz von Audits in Oracle Database ist wichtig zum Protokollieren von Aktionen innerhalb der Datenbankanwendung selbst [70, S.47]. Es muss von den zuständigen Personen gut eingeschätzt werden, welche Aktivitäten auditiert werden sollen und welche nicht. Das hat den Grund, dass Auditing viel Leistung verbrauchen kann.

Protokollierung (Auditing)

PL/SQL von Oracle kann verwendet werden, um Audits zu aktivieren. Die Implementierung von Audits kann durch Trigger erfolgen. In PL/SQL können Tabellen beispielsweise so eingerichtet werden, dass bei einer Aktion direkt ein vorher eingerichteter Trigger feuert. Dieser Trigger kann dann die Aktion in eine extra für das Auditing angelegte Tabelle speichern und somit ein Verzeichnis für das Audit-Trail anlegen [46, S. 362]. Hierfür hat der Entwickler die Möglichkeit, zwischen verschiedenen DML-Befehlen zu entscheiden. Durch diese Art des Auditing ist es möglich, eine einfache Protokollierung der Interaktion eines Benutzers mit dem System zu schaffen.

Der Nachteil an dieser Methode ist, dass die Trigger auf Tabellen erstellt werden müssen und somit die Informationen verteilt sind [70, S.49]. Dem kann aber mit dem Unified Audit-Trail, das Aktivitäten aus verschiedenen Quellen zusammenführt, entgegengewirkt werden [80].

Ein weiteres Problem kann sein, dass Rekursive Trigger-Aufrufe stattfinden oder die Trigger sich gegenseitig blockieren. Das könnte eventuell die größte Schwachstelle von Triggern sein und wäre ein Grund, ein anderes Verfahren für die Protokollierung zu verwenden. Weitere Audit Methoden sind Normale Audits, API-Audits, Unified Audits und Fine-Grained Audits [70, S.47-53]. Jedes dieser Methoden hat seine eigenen Vor- und Nachteile. Die Entscheidung für die richtige Audit-Methode hängt von dem Anwender ab.

Auditing Best Practices

Wichtig ist bei der Protokollierung, dass solche Aktivitäten gewählt werden, die die größten Sicherheitsrisiken darstellen [80]. Auditing verbraucht einiges an Leistung und muss deshalb mit Bedacht eingerichtet werden. Es kann verwendet werden, um verdächtige Aktivitäten zurückzuverfolgen, spezifische zu überwachen und unautorisierten Zugriff zu melden [80].

Wenn Aufgrund der hohen Komplexität von Triggern beim Aufrufen von Daten andere Audit-Methoden gewählt werden, kann es dazu kommen, dass nur die Benutzer protokolliert werden, nicht aber, was genau an Informationen diese manipuliert haben [72, S.268]. Für solche Situationen wurde das Fine Grained Auditing entwickelt.

Fine Grained Auditing (FGA)

Durch ein Fine Grained Auditing wird eine erhöhte Sicherheit gewährleistet, indem alle Informationen über Aktivitäten detailliert protokolliert werden [72, S.268]. Es kann eine Richtlinie mithilfe des Befehls add_policy erstellt werden, indem hinterlegt wird, welches Attribut von welcher Tabelle kontrolliert werden soll [72, S.268]. Hierdurch kann mithilfe von vordefinierten Bedingungen überprüft werden, wie Anfragen auf die ausgewählte Tabelle protokolliert werden. Das ermöglicht eine besonders detaillierte Protokollierung und auch einen Informativen Audit-Trail. FGA ist besonders nützlich für die Überwachung sensibler Datenzugriffe und die Einhaltung von Standards oder anderen internen Richtlinien.

5.1.3 Das Erstellen von Backups

Ein Backup ist eine Kopie der in dem Datenbanksystem gespeicherten Daten [78]. Backups dienen dazu, eine hohe Verfügbarkeit gegen Notfallsituationen und damit

verbunden Systemausfällen und Datenverlusten zu erreichen. Das gelingt durch eine Wiederherstellungsmöglichkeit um das Schadensausmaß zu minimieren. Dieser Vorgang dient zum Schutz gegen Komplettausfälle eines Datenbanksystems [93, S.302-303]. Backups können in Physische und logische Backups unterteilt werden.

Physische Backups

Physische Backups duplizieren das Medium, also die Festplatte oder die Festplatten-Arrays, ohne die darauf gespeicherten Daten zu verändern [52, S.7].

Diese Art von Backups wird in der Regel nicht mithilfe von PL/SQL eingerichtet. Hierfür bieten unterschiedliche DBMS eigene Anwendungen an. Beispielsweise kann eine Standby-Datenbank erstellt werden, die immer aktuell gehalten wird und alle Daten aus der primären Datenbank enthält [62, S.116]. So können Datenbankanwendungen gespeichert werden, damit bei einer Notfall-Situation noch ein Zugriff möglich ist.

Logische Backups

Logische Backups konzentrieren sich stärker auf die Daten selbst und nicht auf das Medium. Sie ermitteln, welche Daten gespeichert werden sollen, und hinterlegen diese dann in einem seperaten Verzeichnis, das auf einer anderen Hardware liegen kann [52, S.6].

Solche Daten können beispielsweise Tabellen, Prozeduren oder Trigger sein [78]. Logische Backups sollen die physischen unterstützen, indem sie die Daten und Meta Daten in einem verständlichen Format sichern, das unabhängig von der Speicherstruktur der Datenbank sein soll.

Backup-Operationen sind Aufgabe des DBAs, der für Probleme wie Backup, Recovery, Installation und Wartung zuständig ist [46, S.86]. Verschiedene DBMS haben auch explizit für das Backup eigene Anwendungen entwickelt, die hierfür Abhilfe schaffen. In Oracle Database wäre das zum Beispiel das Tool Recovery Manager (RMAN) [62, S.1].

Es ist sehr zu empfehlen, die für Backups entwickelten Tools zu verwenden, um eine automatisiert aktualisierende Backup-Datei zu erzeugen. Manuelle, mit PL/SQL eingerichtete Optionen, sind an dieser Stelle nicht zu empfehlen. Das hat den Grund, dass die Backup-Strategie besonders wichtig ist und optimiert werden sollte. Wie bereits in vorherigen Abschnitten erwähnt, haben manuell eingerichtete Funktionalitäten den Nachteil, dass Komplikationen durch Entwicklerfehler entstehen können. Ihr Vorteil ist häufig, dass sie individuell besser anpassbar sind als automatisierte Prozesse, was aber bei der Backup-Strategie aber keine vorrangige Rolle spielt. Zusätzlich sollten Standards und

bewährte Verfahren unbedingt mit berücksichtig werden. Diese enthalten ausführliche Informationen über die korrekte Vorgehensweise bezüglich der Einrichtung.

5.2 Einrichten des Authentifizierungs- und Identitätsmanagement

Das Authentifizierung- und Identitätsmanagement behandelt im Allgemeinen das Erstellen von Benutzer-Profilen mit einer Benutzerkennung und einem Passwort sowie der Zuweisung von Zugriffsrechten an diese Profile. Diese Einteilung ermöglicht das Verwalten von Rechten und das einfache Protokollieren von Aktivitäten [106, S.30].

Im SQL-Kontext gehören zu diesen Zugriffsrechten Data Definition Language (DDL) und Data Control Language (DCL). DDL erstellt Profile und Benutzer und der Einsatz von DCL weist diesem Rechte zu oder enteignet ihn von Rechten.

Hierbei repräsentieren Benutzer Digitale Identitäten. Für diese kann die Identität aus einem anderen System übernommen oder eine neue erstellt werden. Diese können dann mit Profilen verknüpft werden, die Sicherheitsmaßnahmen enthalten. Wird der Benutzer übernommen, können Ihm zusätzliche Zugriffsrechte vergeben oder vorhandene entzogen werden. Nach dem Erstellen oder Übernehmen der digitalen Identität können dann verschiedene authentifikations-Methoden verwendet werden, um eine Anmeldung im Datenbanksystem zu ermöglichen. Erst dann kann eine Manipulation der Informationen mithilfe von SQL erfolgen.

5.2.1 DBMS-Basierte Authentifikation

Bei dieser Art der Authentifikation muss sich die Person gegenüber einer Zentrale mit einem individuellen Geheimnis (Passwort) authentifizieren [10, S.31-32]. Hierfür wird für die Person ein Account oder auch genannt Benutzer erstellt. Nach dem Erstellen eines Benutzers hat dieser erst einmal nur die Out-of-the-box-Security [38, S.272]. Dazu gehören das Anlegen eines Benutzernamens und eines Passworts mit mindestens 4 stellen. Dieser Sicherheitsstandard ist aber für die meisten Umgebungen nicht ausreichend.

Durch die Aktivierung von im Profil existierenden Sicherheitsmaßnahmen können zusätzliche Richtlinien implementiert werden. Wie in Kapitel 4 bereits erwähnt, können über Profile Passwort-Policy Einstellungen eingerichtet werden. Hierdurch können für die Passwortvergabe bestimmte Bedingungen hinzugefügt werden:

- Länge des Passworts
- Unterschiedlicher Benutzername und Passwort
- Keine einfachen Passwörter
- Das Passwort muss eine Kombination aus Zahlen und Buchstaben sein

[38, S.272].

Der Einsatz der im Profil existierenden Sicherheitsmaßnahmen verbessert die Informationssicherheit der Datenbank um ein Vielfaches. Durch die Einrichtung von gewissen Vorgaben, wie beispielsweise der Länge eines Passworts oder der Zusammensetzung, kann die Wahrscheinlichkeit für einen erfolgreichen Angriff stark minimiert werden.

Zusätzlich kann durch den Einsatz von Hash-Funktionen die Sicherheit der in der Datenbank gespeicherten Benutzerinformationen bewahrt werden. Hashing und Salting sind wesentlicher Bestandteil zur Bewahrung der Integrität von Datenbankanwendungen. Sie wurden in Kapitel 3 vorgestellt. Im Folgenden wird ihr Einsatz in einer PL/SQL Umgebung näher betrachtet.

Einsatz von Hash-Funktionen mit DBMS_Crypto.hash und Ora_Hash

Hashfunktionen werden überall eingesetzt, wo eine eindeutige Zuordnung von Informationen notwendig ist. Ein Einsatzgebiet ist in den meisten Fällen das Hashen von Passwörtern. Bei einer Registrierung trägt ein Anwender seinen Benutzernamen und sein Passwort ein. Das verwendete Benutzer-Passwort kann dann durch den Einsatz einer Hashfunktion dauerhaft umgewandelt und somit geschützt werden. Bei einer erneuten Anmeldung werden dann bei korrekter Kombination von Benutzernamen und Passwort diese erneut mithilfe von Hash-Algorithmen umgewandelt und mit dem gespeicherte Hash-Wert verglichen. Ist der Abgleich korrekt, wird das Passwort akzeptiert.

In verschiedenen DBMS existieren für das Hashen selbst spezifische Befehle. Da diese Arbeit auf die Entwicklung mit PL/SQL von Oracle Database beruht, werden an dieser Stelle das Paket DBMS_Crypto.Hash und die Funktion Ora_Hash vorgestellt und miteinander verglichen.

Vor- und Nachteile

Ora_Hash ist eine PL/SQL-Funktion die ein einfaches erstellen eines Hash-Wertes innerhalb der Datenbank angewendet werden [83]. Vorteilhaft ist Ora_Hash, weil es einen

flexiblen Einsatz ermöglicht. Es ist eine integrierte SQL-Funktion und kann innerhalb von Prozeduren und Funktionen aufgerufen werden. Das ermöglicht einen flexiblen Einsatz.

Der Nachteil ist jedoch der Algorithmus, den Ora-Hash verwendet. Dieser ist nicht offiziell bekannt, weswegen angenommen werden kann, dass es kein sicherer Algorithmus ist. Aus diesem Grund sollte bei dem Erstellen von besonders sicheren Umgebungen die Verwendung von DBMS_Crypto.Hash bevorzugt werden.

DBMS_Crypto.Hash ist ein PL/SQL Paket und unterstützt den aktuell sichersten Hash-Algorithmus SHA-2 [81]. Das Paket DBMS_CRYPTO ist Oracle Database spezifisch. Andere DBMS bieten ähnliche Alternativen. Der genaue Einsatz von DBMS_Crypto.Hash wird unter dem hierauf folgenden Abschnitt erklärt.

Salzen/Salting in PL/SQL

Auch wenn Hash-Werte als sicher eingestuft werden, existieren wie bei anderen Sicherheitsmaßnahmen auch hier Möglichkeiten, diese anzugreifen. Eine Möglichkeit, Hash-Werte anzugreifen, ist der Rainbow-Table-Angriff, bei dem bereits erstellte Hash-Werte verwendet werden, um diese mit den im System hinterlegten Werten abgleichen zu können [123, S.49]. Hierbei denkt sich der Angreifer Passwörter aus, von denen er meint, sie könnten übereinstimmen, und erzeugt ihren Hash-Wert. Die entstandenen Werte werden dann mit den im System gespeicherten Hash-Werten abgeglichen. An diese kann der Angreifer über andere Mittel wie Phishing oder Datenlecks (Englisch: Data Dumps) gelangen. Phishing ist ein Angriff, bei dem vertrauliche Informationen beispielsweise über das Abfangen von Emails erlangt werden [11, S.4850]. Data Dumps enthalten vertrauliche Informationen und können entweder privat oder auf dem Schwarzmarkt verkauft beziehungsweise ersteigert werden [91, S.446]. Sollte nun bei einem Abgleich mit den vorhandenen Daten eine Übereinstimmung vorliegen, kann davon ausgegangen werden, dass das ausgedachte Passwort dem Hash-Wert entspricht und somit dieses das Passwort für die Anmeldung ist.

Zusätzlich besteht das in Kapitel 3 vorgestellte Problem der Kollision noch, bei der zweimal derselbe Hash-Wert entstehen kann. Um Kollisionen und Rainbow-Table-Angriffe zu verhindern, kann das Salzen verwendet werden.

Das Salzen wird auch durch Pakete in PL/SQL umgesetzt. Hierfür wird mithilfe von PL/SQL eine Funktion erstellt, die als Parameter eine gewisse Länge hat. Um ein passendes Salz zu erstellen, ist es wichtig zu wissen, mit welchem Datentyp der Entwickler arbeitet.

Bei dem Datentyp Raw kann das Paket DBMS_CRYPTO.RANDOMBYTES(Länge) eine zufällige Länge an BYTES generieren, die einer Variable zugewiesen werden, um den Rückgabewert als gesalzten Wert zu erhalten. VARCHAR2 hingegen ermöglicht durch eine einfache Konkatenation mit einem anderen, zufälligen VARCHAR2-Wert das Erstellen eines Salzes, welches dann mit dem Hash-Algorithmus umgewandelt werden kann. Der Prozess des Salzens erfolgt über 4 Schritte und wird im Folgenden durch den Einsatz von DBMS_Crypto.Hash vorgestellt:

1. Es wird eine Funktion in PL/SQL erstellt, die dafür verantwortlich ist, ein als Parameter eingegebenes Passwort zu salzen
2. Basierend auf den Datentypen wird dann ein Salz erstellt, das eine zufällige Zeichenkombination ist, welches an das Passwort angehängt wird.
3. Nun wird DBMS_Crypto.Hash verwendet, um den neu entstandenen Wert zu einem Hash-Wert umzuwandeln. Hierfür benötigt die Funktion zwei Parameter, den Eingabewert und den Algorithmus, der den Hash-Wert erzeugen soll. Der Aufruf kann folgendermaßen aussehen: Es wird eine Variable erstellt, der der zu entwickelnde Hash-Wert zugewiesen wird.
4. Hash_Var := DBMS_CRYPTO.HASH(input => 'passwort123', typ => DBMS_CRYPTO.HASH_SH256);

An dieser Stelle soll ein Eingabewert, das Passwort, mit dem SHA-2 (256) Algorithmus verschlüsselt werden. Welchen Algorithmus der Entwickler verwenden möchte, hängt von der jeweiligen Situation ab. Empfehlenswert ist jedoch immer ein SHA-2 Algorithmus, da wie in Kapitel 3 erwähnt, andere Algorithmen anfällig für Kollisionen sind.

5.2.2 Externe Authentifikation

Externe Authentifikation-Methoden können die DBMS-Basierte Authentifikation entweder erweitern oder ersetzen. Hierfür werden unterschiedliche Authentifikations-Ansätze präsentiert: Zu denen zählen das LDAP-Protokoll, das Kerberos-Protokoll und die Betriebssystem-Authentifikation.

Lightweight Directory Access Protocol (LDAP)

Das LDAP ist ein Internetprotokoll, welches Anwendungen verwendet, um Informationen abzurufen [116, S.252]. Es unterstützt das Prinzip des SSO und ist in vielerlei Hinsicht anwendbar.

Es wird verwendet, um Benutzer-Verzeichnisse zentral zu verwalten [109, S.162]. Das ist wichtig für das IAM, weil alle Zugänge zentral überwacht werden können. Somit ist das hinzufügen oder entfernen von Benutzern sehr einfach umzusetzen. LDAP ist in vielen DBMS einsetzbar und ein guter erster Ansatz für die Authentifikation. Häufig werden aber in der Praxis IAM-Lösungen verwendet, die LDAP erweitern oder auch komplett ersetzen. Das hat den Grund, dass eine höhere Sicherheitsanforderung in den meisten Umgebungen besteht und LDAP diesen nicht genügt. Das gelingt durch das Einrichten erweiterte Funktionalitäten, darunter SSO, MFA, Zugriffskontrollen und weitere.

Kerberos

Kerberos ist ein reiner Authentifizierungs- und Schlüsselaustauschdienst. Es basiert auf dem Konzept des Single-Sign-On (SSO) und funktioniert folgendermaßen: Bei der Anmeldung meldet sich ein Benutzer mit Benutzernamen und Passwort an. Während dieser Anmeldung erstellt der Client-Rechner einen Session-Key sowie ein Ticket mit Benutzerinformationen für die Kommunikation mit dem Key Distribution Center (KDC) [73, S.6]. Das bei der Anmeldung des Benutzers generierte Ticket wird an das KDC gesendet, welches der zentrale Anlaufpunkt ist, der die Ticket-Verteilung verwaltet [69, S.109291-109295].

Der KDC entschlüsselt die Nachricht, überprüft die aktuelle Uhrzeit (die nicht älter als 5 Minuten sein darf) und stellt dann ein Initialticket aus, das als Ticket-Granting-Ticket (TGT) bekannt ist. Dieses enthält die Benutzerinformationen und das Session-Key.

Das TGT ermöglicht es dem Benutzer, weitere Tickets ohne erneute Anmeldung anzufordern, um Authentifizierungen durchzuführen. Der KDC überprüft dann die Benutzerinformationen und authentifiziert den Benutzer. Nach erfolgreicher Authentifizierung erstellt der KDC ein Zugriffsticket für den Dienst und verschlüsselt diesen mit einem geheimen Schlüssel, der nur dem Benutzer und der Anwendung bekannt ist, auf die der Benutzer zugreifen möchte [73, S.6]. Dieses kann dann wieder an den Client-Rechner gesendet werden, der das verschlüsselte Ticket einfach an die Anwendung weiterleitet. Die Entschlüsselung erfolgt dann von dem Dienst der Anwendung, welches das Ticket überprüft und Zugriff gewährt.

Durch diese Aufstellung gilt Kerberos als ein sehr sicheres Protokoll und kann mithilfe der Authentifikations-Methodik die Sicherheit von LDAP-Protokollen stark verbessern. Aufgrund dessen ist eine Integration von Kerberos in LDAP möglich.

Kerberos in LDAP integrieren

Die Integration von Kerberos in LDAP macht insoweit Sinn, dass für den Zugriff auf die Verzeichnisse eine Authentifizierung seitens des Anwenders erfolgen muss. Somit kann sichergestellt werden, dass diese nicht von unbefugten Anwendern manipuliert werden.

Kerberos bietet hierfür eine hohe Sicherheit: Das Protokoll ist anfällig für einige Angriffe, die aber komplex umzusetzen sind [69, S.109294-109295]. Durch die Möglichkeit des Single-Sign-On-Prinzips kann es ohne Probleme im gesamten Netzwerk verwendet und auch in LDAP integriert werden [117, S.123-124]. LDAP und Kerberos zusammen bieten eine hohe Sicherheit sowie eine übersichtliche Zugänglichkeit, was diese Kombination besonders stark macht.

Betriebssystemauthentifizierung

Verschiedene DBMS bieten Möglichkeiten zur Authentifizierung über das Betriebssystem an. Wichtig ist bei jedem Betriebssystem, dass in den SQL-Einstellungen hinterlegt ist, dass der Benutzer zu einer Authentifizierung durch die Betriebssystem-Daten berechtigt ist [42, S.158-159]. Viele Datenbankverwaltungssysteme, darunter auch Oracle Database, bieten die Möglichkeit der Betriebssystemauthentifizierung [82]. Die Benutzer können einzeln von dem DBA in den Stammdaten gespeichert und dann Gruppen zugewiesen werden, um die Authentifizierung zu erleichtern. Passwörter werden durch in Windows eingerichtete Passwort-Richtlinien geschützt. Dazu gehören die Passwortlänge und die Sperre bei häufiger falscher Eingabe [42, S.152].

Der Vorteil ist, dass sich ein Benutzer ohne Probleme mit seinen üblichen Anmeldedaten auch in dem DBS anmelden kann. Somit ist die Anwenderfreundlichkeit besonders hoch und die zentrale Verwaltung ermöglicht eine bessere Protokollierung und damit stärkeren Schutz. Genau das kann aber auch als Nachteil angesehen werden: Ein Angreifer hat theoretisch nach einmaliger Identifizierung des Passworts die Möglichkeit, auf alle Anwendungen zuzugreifen. Situationsbedingt kommt es deshalb auf das Umfeld an, in dem diese Art von Authentifikation stattfindet.

5.2.3 Multi-Faktor-Authentifizierung (MFA)

Alle Authentifizierungsmethoden können mittels MFA verbessert werden. MFA kann in Datenbanksystemen eingesetzt werden, um eine besonders sichere Authentifikation zu

gewährleisten. Es wird erforderlich, wenn Sicherheitsanforderungen höher eingestuft werden sollen [75, S.2]. Die MFA-Ist eine Kombination aus zwei Techniken: Es zeichnet sich häufig daraus aus, dass die sich zu authentifizierende Person etwas Wissen und Besitzen muss.

Zum Beispiel werden häufig die Passwort Authentifikation mit einem Gerät zusammen verwendet, über das ein Code freigegeben werden muss [26, S.166]. Hierbei wird die Passwort-basierte Authentifikation verwendet, die aber zusätzlich noch einen Code verlangt, dass auf ein vorher eingerichtetes vertrauliches Gerät geschickt wurde. Um dieses Prinzip umzusetzen, bieten verschiedene DBMS unterschiedliche Erweiterungen an. Durch ihren Einsatz können Entsperrcodes beispielsweise via SMS verschick, die dann als zusätzliche Abfrage an den Benutzer gestellt werden, um diesen eindeutig zu identifizieren [75, S.1-3.].

Ein weiterer Ansatz ist das Einrichten eines Virtual Private Network (VPN). Eingerichtet wird es, indem in der Datenbankkonfiguration hinterlegt wird, dass nur eine bestimmte IP-Adresse für die Verbindung zugelassen ist. Die Verbindung zu dieser VPN ist dann Passwortverschlüsselt. Die Firewall-Einstellungen von SQL ermöglichen es, jeglichen Zugriff auf das Datenbanksystem zu verweigern, wenn die eingerichtete Adresse nicht übereinstimmt.

Bei Oracle kann als Erweiterung beispielsweise der Oracle Access Manager (OAM) erwähnt werden [95, S.205]. Diese Erweiterungen sind aber häufig kostenpflichtig und Plattform bezogen, so dass eine Erweiterung von einem Anbieter X nicht auf Plattform Y verwendet werden kann.

5.3 Einhalten der Integrität

Dieser Abschnitt soll Transaktionsprobleme bei einem parallelen Zugriff genauer betrachten und die in Abschnitt 3.2.2 vorgestellten Lösungen umsetzen. Bezüglich der Transaktionen ist in erster Linie die Integrität der Daten wichtig, die mithilfe von Constraints eingehalten werden kann.

Weitere, Transaktionsspezifische Sicherheitsmaßnahmen sollten darauf abzielen, reibungslos verlaufende Transaktionen zu ermöglichen, um auf Datenbankanwendungen Systemausfälle zu vermeiden.

5.3.1 Einsatz von Einschränkungen

Die Integrität von Tabellen wird mithilfe von Einschränkungen in PL/SQL umgesetzt. Wichtig ist die Bewahrung der Integrität. In diesem Kontext bewahren Primärschlüssel- und Fremdschlüssel-Bedingungen die Referentielle Integrität und Check-Constraints die Domänenintegrität. Mit ihnen können Datentypen und Wertebereiche überprüft werden. Ein Beispiel für ein Check-Constraint wäre wie folgt:

```
ALTER TABLE streetwear

ADD CONSTRAINT check_groesse

CHECK (groesse != 'XXL');
```

Codebeispiel 6 (eigener Code)

Der Einsatz von Constraints ist in PL/SQL für die Integrität unerlässlich. Sei es für die Referentielle Integrität, die Entity Integrität oder um nur gewisse Datensätze zuzulassen. Sie stellen sicher, dass nur gültige Daten in die Datenbank eingefügt werden können. Dadurch kann versichert werden, dass die Datenbank korrekt und konsistent bleibt.

Eine weitere Möglichkeit ist der Einsatz von Triggern. Trigger können komplexere Integritätsregeln umsetzen. Ein Beispiel hierfür wäre, wenn eine Tabelle Sportartikel existieren würde, und diese alle Artikel akzeptieren soll, bis auf blaue Sportanzüge, die von Benutzer „user1“ eingefügt wurden. Dieser Constraint wäre mit einfachen Mitteln schwer umzusetzen, ein Trigger kann das jedoch ohne weiteres ermöglichen. Das hat den Grund, dass Trigger sich auf den Benutzer beziehen können, Constraints bieten diese Möglichkeit der Zugriffskontrolle nicht direkt.

Auch wenn Constraints und Trigger dabei helfen, die Integrität der Datenbank zu bewahren, kann es trotzdem dazu kommen, dass bei mehreren Benutzern gewisse Probleme bei gleichzeitiger Bearbeitung von Tabellen entstehen. Im Folgenden werden einige dieser Probleme aufgelistet und darauf bezogen Lösungen präsentiert.

5.3.2 Transaktionsprobleme bei parallelem Zugriff

Transaktionsprobleme entstehen, wenn mehrere Benutzer an derselben Tabelle Änderungen vornehmen wollen [60, S.227]. Das kann dazu führen, dass Änderungen nicht korrekt abgespeichert und somit auch nicht planmäßig umgesetzt werden.

An dieser Stelle sollen bei dem Einsatz von Transaktionen entstehende Probleme genauer betrachtet werden. Bei einem parallelen Zugriff können verschiedene Komplikationen auftreten. Im Folgenden werden einige dieser Komplikationen aufgelistet.

Lost Update: Es kann sein, dass bei gleichzeitiger Bearbeitung einer Tabelle Änderungen nicht komplett registriert werden können. Wenn beispielsweise Person A etwas an der Tabelle ändert, vor der Persistierung aber Person B eine Änderung vornimmt, dann wird die Änderung von Person A gar nicht erst registriert. Dieses Phänomen wird auch als Lost Update bezeichnet [60, S. 227].

Phantom Read: Ein Phantom Read tritt auf, wenn eine Transaktion ausgeführt wird, während dieser weitere, neue Datensätze auftauchen die bei der eigentlichen Transaktion nicht berücksichtigt werden können, weil sie zu dem Zeitpunkt nicht existiert haben [60, S.228]. Eine solche Situation kann kritisch sein und tritt auf, wenn eine parallele Nutzung stattfindet.

Unrepeatable Read: Das Unrepeatable Read geschieht, wenn eine Transaktion ausgeführt wird, die denselben Datensatz zweimal liest, zwischen den Lesevorgängen aber ein anderer Nutzer eine Transaktion an der Tabelle vorgenommen hat [39, S.229]. In so einem Fall kann eine erneute Ausgabe dieses Datensatzes nicht erfolgen, da eine Änderung stattgefunden hat.

Deadlock: Der Deadlock Ist das gegenseitige Blockieren von Transaktionen [60, S.231]. Diesen treten auf, wenn beispielsweise zwei unterschiedliche Transaktionen versuchen zwei unterschiedliche Tabellen gleichzeitig zu ändern und diese dabei sperren. Hierdurch kann keine Transaktion ausgeführt werden, da beiden Tabellen gesperrt sind.

Maßnahmen gegen diese Probleme sind Prozeduren oder auch Funktionen, die so erstellt werden, dass sie gleichzeitige Änderungen nicht zulassen sowie eine gute Rechteverteilung. In der in Anhang beigefügten Test-Datenbank wurde beispielsweise für das Lost Update ein Trigger erstellt, der den Bestand vor einem Update abgleichen soll. Falls dieser nicht mit dem aktuellen Bestand übereinstimmt, wird eine Fehlermeldung von dem Trigger ausgegeben, so dass das Update nicht ausgeführt werden kann. Somit kann bei einem parallelen Zugriff das Problem des Lost-Update vermieden werden. Derselbe Trigger kann dann auch für das Problem des Unrepeatable Read verwendet werden.

Bezüglich Phantom Reads kann ein einfacher AFTER INSERT|UPDATE|DELETE TRIGGER helfen, der nach der Anwendung einer Transaktion auf eine Tabelle automatisch ein COMMIT ausführt. Somit wird die Vollständigkeit der Transaktion sichergestellt.

Deadlocks sind etwas komplizierter zu lösen. Das Problem hierbei ist, dass keine Tabelle die Ressourcen freigeben kann, da sie selbst eine Anfrage geschickt haben und auf eine Freigabe warten. Sie werden am besten durch Prävention, Detektion oder Vermeidung umgangen [96, S. 484]. Bei Prävention wird bei einem möglichen Eintritt eines Deadlocks die gesamte Transaktion rückgängig gemacht. Die Detektion testet das DBMS in regelmäßigen Zeitabständen auf potenzielle Deadlocks und die Vermeidung erstellt eine Bedingung, bei der alle Transaktionen nacheinander abgeschlossen werden. Hierfür werden Protokolle verwendet, die den Zugriff auf eine Ressource nur dann gewähren, wenn das Protokoll entschieden hat, dass es sicher ist [99, S.12]. Eine effektive Möglichkeit ist hierbei die Deadlock-Detection, bei der Deadlocks zwar entstehen können, nach dem Identifizieren aber dann mittels eines einfachen ROLLBACK-Befehls aufgelöst werden.

5.4 Einhalten der Vertraulichkeit

Die Vertraulichkeit kann durch Verschlüsselungen bewahrt werden. Verschlüsselungen werden in Oracle Database PL/SQL unterschiedlich umgesetzt. Sie bieten Schutz vor internen und externen Angriffen. Zu diesen Angriffen zählen zum Beispiel Man-in-the-Middle (MITM) Attacken, bei denen ein Angreifer sich zwischen zwei Kommunikationspartnern positioniert und versucht, die ausgetauschten Nachrichten abzuhören [11, S.4850].

Dieser Abschnitt soll sich mit dem Einsatz von Verschlüsselungen befassen und festlegen, welche Alternative die Bedürfnisse für eine solide Sicherheit am besten erfüllt.

5.4.1 Manuelle Verschlüsselung

Bei der Manuellen Verschlüsselung ist der Entwickler dafür verantwortlich, diese einzurichten. Durch eigenen Code wird die Funktionalität somit in das Datenbankverwaltungssystem implementiert.

Vor dem Eintragen von sensiblen Daten in eine Tabelle müssen diese immer verschlüsselt und vor dem Abrufen entschlüsselt werden. Die Verschlüsselung an sich erfolgt so, dass ein zu verschlüsselnder Text definiert wird, häufig in VARCHAR2. Der eingesetzte Schlüssel wird dann mit der Größe des Algorithmus, bei AES-128 wäre das beispielsweise der Schlüssel Key RAW (128) für 128 Bits, erstellt. Da bei herkömmlichen Verschlüsselungen die symmetrische Verschlüsselung eingesetzt wird, und Blockchiffre die

sicherere Alternative ist, kann davon ausgegangen werden, dass der Ausgabewert dieselbe Größe wie der Eingabewert hat.

Dann beginnt der prozedurale teil, bei dem das Paket DBMS_CRYPTO.ENCRYPT aufgerufen wird. Mithilfe von einer Funktion kann der Einsatz der Verschlüsselung in PL/SQL erfolgen. Der Vorteil ist dabei, dass diese Funktion funktional erweiterbar und anpassbar sein kann. Folgender Aufbau kann beispielhaft eine solche Funktion darstellen:

Der Schlüssel kann als Parameter aufgenommen werden, genauso wie die zu verschlüsselnde Datei. Ausgabewert ist dann die verschlüsselte Datei. Innerhalb des Anonymous Block wird diese durch den Einsatz von DBMS_CRYPTO und den ausgewählten Verschlüsselungsalgorithmus verschlüsselt. Angemerkt wird an dieser Stelle noch, dass die zu verschlüsselnde Datei umgeformt werden muss in einen binären Wert, damit die Rundenfunktion wie in Kapitel 3 beschrieben die Verschlüsselung mittels der Diffusion durchführen kann.

Der Einsatz von DBMS_CRYPTO erfolgt in Gebieten, in denen Informationen entweder nur privilegierten Benutzern zugänglich gemacht werden sollen oder wenn besonders relevante Sicherheitsinformationen möglichst effektiv gesichert werden müssen.

Durch den Einsatz der asymmetrischen Verschlüsselung in Kombination mit der symmetrischen Verschlüsselung, auch genannt hybride Verschlüsselung, können Schlüssel dann auch verschlüsselt ausgetauscht werden. Das hat den großen Vorteil, dass Angreifer keine Möglichkeit haben, Schlüssel abzufangen und so an die verschlüsselten Daten zu gelangen. Sie lassen sich auch manuell einrichten oder durch den Einsatz von DBMS-bezogenen Erweiterungen integrieren.

Vor- und Nachteile der manuellen Methode

Ein großer Nachteil ist, dass die Speicherung in einer SQL-Datenbank es erfordert, die Dateien in RAW oder BLOB umzuwandeln [109, S.256]. Das binäre Format sorgt für einen guten Schutz und ist dem VARCHAR2-Format vorzuziehen, weil VARCHAR2 eine unerwünschte Offenlegung sensibler Informationen preisgeben kann. Jedoch muss die Tabelle, auf der die Verschlüsselung stattfinden soll, extra hierfür eine Spalte besitzen, die die Binären Daten aufnehmen kann. Damit verbunden müssen dementsprechend Tabellen aktualisiert werden, was bei großen Datenmengen umständlich sein kann.

Eine weitere Herausforderung ist das Abspeichern des Schlüssels [109, S.256]. Wenn dieser in die falschen Hände gelangt, macht die Verschlüsselung wenig Sinn. Dementsprechend muss die sichere Verwahrung des Schlüssels eine herausragende Rolle spielen. Ein Ansatz ist, diesen innerhalb der Datenbank zu lagern. Somit kann eine Tabelle erstellt werden, die die Schlüsselinformationen bewahrt und auf die nur Prozesse oder Funktionen den Zugriff ermöglichen können. Wenn eine Datei verschlüsselt oder entschlüsselt werden muss, würde immer die dafür zuständige Prozedur oder Funktion aufgerufen werden. Hierdurch wird jedoch der Zugriff verkompliziert, was die Leistung der Datenbank durch einen hohen Arbeitsaufwand mittels ständiger Aufrufe stark beeinträchtigen kann. Zudem kann die hybride Verschlüsselung nicht ohne weiteres eingesetzt werden, da das Schützen des Schlüssels mittels Prozeduren keinen absolut zuverlässigen Schutz bietet. Das liegt daran, dass Aufgrund der ständigen Aufrufe von Prozeduren das Risiko einer Datenexposition besteht.

Vorteilhaft ist jedoch, dass der Entwickler die Verschlüsselung nach seinen eigenen Wünschen gestalten kann. In diesem Kontext kann der Abstraktionsgrad der Implementierung vom Anwender personalisiert und potenziell auch erweitert werden.

5.4.2 Funktionale Erweiterungen für die Verschlüsselung

Um die negativen Aspekte zu vermeiden haben verschiedene Anbieter eigene Lösungen entwickelt, die die aufgelisteten Probleme vermeiden sollen. In Oracle Database ist eine solche Erweiterung das Oracle Advanced Security Options-Paket [63, S.723] oder auch das Oracle Crypto Software Development Kit (SDK) [86]. Grundsätzlich bieten diese Erweiterungen integrierte Funktionen an, die Verschlüsselungen vereinfachen und die Probleme der manuellen Verschlüsselung bekämpfen sollen.

Eines dieser Lösungen zur Vereinfachung der oben aufgelisteten Probleme ist ein sogenanntes Wallet: ein Ort zum Lagern von Schlüsseln für die Ver- und Entschlüsselung [32]. Ein Wallet ist sehr hilfreich, weil in der manuellen Verschlüsselung diese Schlüssel-Lagerung sehr umständlich einzurichten ist und nur mit sehr viel Erfahrung seitens des Entwicklers relativ zuverlässig sein kann.

Eine weitere Option ist in Oracle Database das Transparent Data Encryption (TDE) zum automatischen Verschlüsseln [32]. TDE und Wallets stehen in enger Verbindung zueinander: Durch TDE kann die Verschlüsselung automatisiert eingerichtet werden, die dabei entstehenden Schlüssel werden dann in der Wallet gelagert.

Vorteilhaft an TDE ist, dass ganze Tabellen automatisch verschlüsselt werden können [109, S.263]. DBMS_CRYPTO kann diese Option nicht anbieten. Zusätzlich kann TDE, Ebenso wie DBMS_CRYPTO, Block- sowie Stromchiffre anwenden. Zudem kann TDE aber auch durch den Einsatz der Wallet-Funktion im Gegensatz zu DBMS_CRYPTO asymmetrische Verschlüsselungen einrichten. In diesem Kontext spielen Wallets eine wichtige Rolle. Somit ist die Hybride Verschlüsselung mit dieser Erweiterung auch sehr unproblematisch einzusetzen.

Der größte Nachteil an Erweiterungen von diversen Anbietern ist, dass sie kostenpflichtig sind. Wenn das primäre Ziel jedoch ein sicheres Datenbank-System und damit verbunden eine sichere Datenbankanwendung ist, dann wäre eine solche Investition an dieser Stelle angebracht.

Dafür spricht auch ein Grundprinzip der Informationssicherheit, das besagt, dass der Einsatz automatisierter Prozesse für die Sicherheit der Daten die klügere Alternative ist [46, S.32]. Als Fazit kann dementsprechend gezogen werden, dass wenn die Entwickler besonderen Wert auf die Informationssicherheit legen, sie die vom Anbieter bereitgestellten Funktionen und Erweiterungen einsetzen sollten.

5.5 PL/SQL Best Practices

PL/SQL Best Practices werden eingesetzt, um verschiedene externe Angriffe durch den Einsatz von gut durchdachtem Code zu vermeiden. Zu diesen Angriffen zählen in erster Linie SQL-Injektionen. SQL-Injektionen sind ein verheerender Angriff, bei dem der Angreifer durch die Eingabe von SQL-Abfragen die dahinter liegende Datenbasis manipuliert [17, S.27-28.]. Hierbei ist jedes System, das Eingaben aus nicht vertrauenswürdigen Quellen annimmt, anfällig für diese Art von Angriff. Dazu zählen Anwendungen, die:

- Eingabewerte nicht validieren, filtern oder reinigen
- Dynamische SQL anfragen falsch einsetzen
- Sensible Informationen nicht durch Prozeduren oder andere Verfahren schützen

[77].

Die wichtigsten Maßnahmen gegen SQL-Injektionen sowie weitere Angriffe, wie beispielswese Cross-Site-Scripting (XSS) [11, S.4851], ist die Validierung der Benutzereingaben. Diese muss geschehen, bevor Abfrage an SQL weitergegeben werden kann. Falls das nicht geschieht, können Anwendungen erstellt werden, die anfällig für Injektionen sind.

5.5.1 Einsatz von PL/SQL zur Validierung

Der Einsatz von SQL ist in jedem Fall unverzichtbar, wenn es um die Manipulation von Daten in einem Relationalen Datenbanksystem geht. Weil Angreifer das Wissen, nutzen sie schlecht gestalteten Code aus, um sich Zugang zu diesen Systemen zu verschaffen. Im Folgenden werden Methoden vorgestellt, die einen sicheren SQL-Code definieren sollen. Ziel ist es, Angreifern eine möglichst geringe Chance zu geben, erfolgreiche Angriffe auf die Datenbankanwendung durchzusetzen.

Validierte Benutzereingaben/Input Validierung

Die Input Validierung hat die Aufgabe, Benutzereingaben vor der Bearbeitung zu validieren, das bedeutet, nach gesetzten Vorgaben zu kontrollieren [61, S.2]. Der Entwickler muss wissen, welche Art von Informationen er erwartet, so dass dementsprechend Vorgaben gesetzt werden können [25, S.5]. Die Datenvalidierung beschäftigt sich also mit der Verifizierung von Daten, so dass eine Kombination aus Werten bestimmten vorgegebenen Bedingungen entsprechen.

Die Input Validierung ist wichtig, weil in SQL-Zeichen mehrere Bedeutungen haben können. Solche Zeichen werden auch als Escape-Zeichen bezeichnet [18, S.14]. Als Beispiel kann das Anführungszeichen „ ‘ “ gegeben werden. Es dient dazu, die Grenze zwischen Code und Daten zu kennzeichnen. In dem Beispiel würde bei einer Eingabe in ein dafür vorgesehenes Feld das Programm davon ausgehen, dass alles, was auf ein Anführungszeichen folgt, code ist, den es ausführen muss. Das hätte zur Folge, dass SQL-Befehle über Eingabefelder ausgeführt werden und so Informationen aus der Datenbank entnommen werden können. Es existieren noch weitere Escape-Zeichen, die so von SQL interpretiert werden. Dazu gehören: Das Leerzeichen ( ), der doppelter Senkrechtsstrich (||), das Komma(,), der Punkt (.), der Befehl für Kommentare (*/) und das doppelte Anführungszeichen („) [18, S.15].

Das hätte verheerende Folgen und sollte bei einer sicheren Datenbankanwendung unbedingt verhindert werden. Dafür ist die Input Validierung gedacht. Für die Input Validierung existieren zwei unterschiedliche Methoden, die Whitelist-Validierung und die Blacklist-Validierung.

Whitelist-Validierung und Blacklist-Validierung

Bei der Whitelist-Validierung werden Eingabewerte mit bekannten Werten abgeglichen, die gewisse Formatierungsstandards erfüllen sollen [17, S.479]. In einer sogenannten Whitelist werden nur Daten aufgelistet, deren Verwendung erlaubt ist. Das kann so aussehen, dass beispielsweise die Bedingung gesetzt wird, dass alle Zeichen von a-z und Sonderzeichen wie “!“,“?“#“@“,“.“ erlaubt sind. Allgemein zählen zu den Erlaubten Eingaben Werte, Typen, Längen, Größen oder andere Formatierungsstandards [17, S. 479]. Alle weiteren Zeichen werden bei der Eingabe nicht akzeptiert und werfen Fehler auf. Diese Art der Validierung ist besonders effektiv, weil erlaubte Zeichen limitiert werden und somit den Freiraum für Angriffe minimieren.

Dem gegenüber steht die Blacklist-Validierung. Bei dieser Art werden alle Zeichen erlaubt, bis auf diejenigen, die in der Blacklist aufgeführt wurden [109, S.145]. Diese Art der Validierung ist Aufgrund dessen aber nicht besonders effektiv und bietet einem Angreifer einen großen Freiraum für Experimente.

Einsatzmöglichkeiten

Die Blacklist-Validierung kann zusammen mit der Whitelist Validierung eingerichtet werden, so dass alle Zeichen erlaubt sind, außer denjenigen, die in der Blacklist stehen. Jedoch macht das im Allgemeinen wenig Sinn, da die Whitelist aufgrund ihrer Funktionsweise priorisiert wird und diese keine Zeichen außer denen, die erlaubt sind, zulässt.

Der Einsatz einer Blacklist in Kombination mit einer Whitelist kann höchstens noch einmal verdeutlichen, welche Zeichen nicht erlaubt sind, bewirken kann die Blacklist in dieser Kombination aber sonst nichts.

Bei der Entscheidung, ob Blacklist- oder Whitelist-Validierung ist die Whitelist-Validierung dementsprechend die bessere Wahl, da das zulassen von bestimmten Zeichen weniger Raum für Angriffe zulässt als das Verbieten bestimmter Zeichen.

In PL/SQL kann die Whitelist- sowie die Blacklist-Validierung mit Tabellen, Prozeduren und Triggern eingesetzt werden. Hierfür wird beispielsweise bei der Whitelist-Validierung eine Tabelle mit den erlaubten Werten erstellt (es kann auch eine einfache Variable mit erlaubten Werten innerhalb der Prozedur verwendet werden, die genaue Vorgehensweise ist Situationsbedingt), die dann mithilfe einer Prozedur, die Benutzereingaben aufnimmt, abgeglichen werden. Sollte ein Wert enthalten sein, der nicht in der Whitelist aufgelistet ist, kann eine Fehlermeldung ausgeworfen werden. Bei einer Whitelist für die

Datenbankanwendung selbst, um Werte zu limitieren die eingegeben werden können, eignet sich der Einsatz eines Triggers besonders gut.

Durch die Validierungen kann also ein guter Schutz vor dem missbräuchlichen Einsatz von Escape-Zeichen ermöglicht werden. Jedoch ist es bei nummerischen Werten nicht notwendig, die Daten in Anführungszeichen einzuschließen [18, S.15]. Diese würden sonst als Zeichenketten behandelt werden. Um diese Schwachstelle zu eliminieren, ist der Einsatz von Prozeduren und parametrisierten Abfragen entscheidend. Vorher werden aber in diesem Kontext dynamische SQL-Abfragen vorgestellt.

5.5.2 Dynamische- & Parametrisierte -SQL-Abfragen

Dynamische Abfragen

Dynamische SQL-Abfragen sind eine Programmierungstechnik, die es Entwicklern ermöglicht, SQL-Anweisungen zur Laufzeit zu erstellen [17, S.40]. Hierdurch können flexible Anwendungen geschrieben werden, die bei unterschiedlichen Situationen andere Ergebnismengen erzeugen können.

Das hat aber auch seine Nachteile: Abfragen können nicht geprüft oder ausreichend gesichert werden, wenn sie zur Laufzeit erstellt werden. Zwar kann eine Validierung eingerichtet werden, die einen guten Schutz bietet. Jedoch bleiben Ziffern wie bereits erwähnt bei einer Validierung außen vor, so dass ein Angreifer diese Schwachstelle nutzen kann, um die Validierung zu umgehen und dynamische SQL-Anfragen zu seinem Vorteil zu nutzen. An dieser Stelle wird stark der Einsatz von Prozeduren, Funktionen oder parametrisierten Abfragen anstelle der dynamischen SQL-Abfragen empfohlen.

Dynamische SQL-Strings sollten nur eingesetzt werden, wenn dem Entwickler die Struktur der Tabelle bis zur Laufzeit nicht bekannt ist, und nicht wenn einfach nur einige Werte nicht bekannt sind [46, S.353]. Aber wenn sie eingesetzt werden, kann dieser Einsatz in einigen Situationen dementsprechend unumgänglich sein. Das sind Situationen, in denen Informationen über verschiedene Kriterien für Abfragen oder Tabellen notwendig sind, und diese zur Laufzeit variieren sollen [46, S.349]. In den meisten anderen Situationen ist der Einsatz von parametrisierten Abfragen sicherer und demnach empfehlenswerter.

Parametrisierte Abfragen

Eine parametrisierte Abfrage ist eine vorkompilierte SQL-Anweisung, bei der eine oder mehrere Parameter in die Anweisung eingebettet werden müssen, damit sie ausgeführt werden kann [2, S.327]. Diese werden nicht als auszuführende Befehle interpretiert und lassen somit keinen Code durch SQL-Injektion einschleusen [2, S.327].

Parametrisierte Abfragen haben die Aufgabe, dynamische Abfragen zu ersetzen, um somit für eine höhere Sicherheit in der Datenbankanwendung zu sorgen. Das ist in den meisten auftretenden Situationen möglich, weil viele Abläufe vorhersehbar sind und dementsprechend Vorbereitungen in Form von parametrisierten Abfragen getroffen werden können [17, S.470].

Es ist aber anzumerken, dass wenn parametrisierte Abfragen in Kombination mit Prozeduren und Funktionen verwendet werden, und diese Code beinhalten, der anfällig für SQL-Injektionen ist, kein Schutz versichert werden kann. Das sind beispielsweise Situationen, in denen keine Validierungen eingerichtet wurden. Im Vergleich zu den dynamischen Methoden sind parametrisierte Abfragen in der Laufzeit etwas langsamer, dafür aber um einiges sicherer. Bei der Einrichtung von möglichst sicheren Datenbanksystemen sollten in jedem Fall deshalb parametrisierte Abfragen bevorzugt werden. Ein Beispiel für eine parametrisierte Abfrage kann wie folgt aussehen:

```
SELECT column_name

FROM table

WHERE your_column = :input_value;
```

Codebeispiel 7 (eigener Code)

Dennoch kann es sein, dass einige komplexe Anwendungen mit dynamischen SQL-Anweisungen codiert werden müssen. Ein Beispiel hierfür ist eine nicht existierende Tabelle oder ein nicht existierendes Feld, das eine variable Struktur haben soll. Wenn dieses während einer Bearbeitung erstellt wird, müssen eine dynamische SQL-Anweisungen eingesetzt werden. Diese Situation Angreifer nutzen, um bestimmte Anfragen zu manipulieren und Felder zu ersetzen. Schutzmaßnahmen sind an dieser Stelle die Whitelist-Validierungen, Auditing und die Zugriffssteuerung. Zusätzlich können Fehlermeldungen eine wichtige Rolle spielen.

5.5.3 Fehlerbehandlung (Exception Handling)

Fehlermeldungen sind dazu gedacht, Informationen über eine gewisse Komplikation über die Konsole anzuzeigen. Typischerweise werden Fehlermeldungen mit einem Code und einer darauffolgenden Definition ausgegeben. Eine Fehlermeldung kann beispielsweise so aussehen:

```
SQL-Fehler: ORA-20001: Ungültiges Material: Kunstleder
```

Codebeispiel 8 (eigener Code)

Es ist zu beachten, dass eine Fehlermeldung nicht zu viele Informationen preisgibt. Angreifer können dieses Wissen nutzen, um daraus Informationen für weitere Angriffe zu gewinnen. Die Fehlermeldung aus dem Beispiel basiert auf einer Blacklist-Validierung, die das Material Kunstleder auf dieser hinterlegt hat. Wenn der Angreifer die Fehlermeldung liest, weiß er, dass Kunstleder auf der Blacklist steht. Hierdurch kann er diesen Begriff ausschließen und ist einen Schritt weiter.

Um das zu vermeiden, müssen Fehlermeldungen so eingestellt werden, dass sie möglichst wenig Informationen preisgeben. An dieser Stelle kann das Exception Handling eingesetzt werden: Es erlaubt Entwicklern, Fehlermeldungen spezifisch zu definieren, wodurch eine Ausgabe mit falschen Informationen vermieden werden kann [36, S.54-55]. Das Exception-Handling hat seine eigene prozedurale Logik innerhalb eines eigenen, dem Anonymus Block ähnelnden prozeduralen Bereiches. Innerhalb dieser Logik wird bei einem Fehlereintritt die vom Anonymous Block ausgeführte Operation übernommen um anschließend die in der Exception definierte Logik auszugeben. Sobald eine Exception ausgegeben wird, bricht die ursprüngliche prozedurale Logik an der Stelle ab [36, S.59] und es wird eine Fehlermeldung ausgegeben. Hierbei kann der Einsatz von RAISE_APPLICATION_ERROR die ausgegebene Fehlermeldung spezifizieren. Das kann so aussehen:

```
RAISE_APPLICATION_ERROR(-20002, 'Ein Fehler ist aufgetreten: ');
```

Codebeispiel 9 (eigener Code)

Exceptions sind ein bewährtes Mittel für die Fehlerbehandlung in einer PL/SQL von Oracel Umgebung [85]. Diese können potenzielle Fehler auffangen und als Meldung ausgeben.

Für alle Fehlermeldungen, die außerhalb der Exception definiert sind, kann die WHEN OTHERS-Klausel eingesetzt werden. Diese fängt alle potenziellen Fehlermeldungen ab, die nicht spezifisch definiert wurden [60, S.257].

Durch den Einsatz von SQLERRM und SQLCODE innerhalb der WHEN OTHERS-Klausel können Informationen nur in dem Maße preisgegeben werden, der für die Informationssicherheit notwendig ist.

SQLCODE: SQLCODE gibt den nummerischen Code der jeweiligen Exception wieder, die eine Fehlermeldung hervorgerufen hat. Wenn keine Fehler existieren, wird eine 0 zurückgegeben. Werte unter 0 geben an, dass ein Fehler aufgetreten ist. Der exakte Wert gibt zusätzliche Informationen zum Typ des Fehlers. Werte über null geben aus, dass die SQL-Anweisung erfolgreich war, aber eine Warnung erzeugt hat. Der genaue Wert gibt weitere Informationen zum Typ der Warnung [80]. Somit wird durch den Einsatz von SQLCODE lediglich ein Fehlercode ausgegeben. Die Ausgabe kann wie folgt aussehen:

```
Fehlercode: 0
```

Codebeispiel 10 (eigener Code)

Für die darauffolgende Definition können bei Detaillierteren internen Fehlermeldungen wie Datenbankdumps und Fehlercodes dem Benutzer weitere Implementierungsdetails angezeigt werden [14]. Die Ausgabe erfolgt in Oracles PL/SQL mittels SQLERRM.

SQLERRM(): SQLERRM erweitert SQLCODE und gibt eine Beschreibung bezüglich der Fehlermeldung aus [81]. Verbunden können beide Werte durch eine einfache Konkatenation bei der Ausgabe, so dass alle möglichen Informationen ausgegeben werden. Die Ausgabe mit SQL-CODE und zusammen SQLERRM kann beispielsweise folgende sein:

```
Fehlercode: 0

Fehlermeldung: ORA-0000: normal, successful completion
```

Codebeispiel 11 (eigener Code)

Für den korrekten Einsatz von Fehlermeldungen muss der Entwickler darauf achten, dass diese möglichst wenige Informationen preisgeben, um die Sicherheit der Datenbank hierdurch gegenüber potenziellen Angreifern zu bewahren.

6 Zusammenfassung

Durch den Einsatz des Risikomanagements für die Identifizierung, Analyse und Bewältigung möglicher Risiken lässt sich eine gut strukturierte Herangehensweise für die Einrichtung von sicheren Datenbankanwendungen ermöglichen. Mittels der in diesem Kontext präsentierten Lösungsvorschläge soll das Datenbanksystem, auf das die Anwendung aufbaut, sicher eingerichtet werden. Ziel ist dabei, alten sowie neuen Angriffen durch Cyberkriminelle entgegenzuwirken. Der Einsatz unterschiedlichen Konzepten, Methoden und Modellen soll das ermöglichen. Die Triade der Informationssicherheit bietet hierbei einen soliden Leitfaden, an den sich Entwickler sowie Datenbankadministrator halten können, um eine solide Grundsicherheit gegen viele unterschiedliche Angriffe zu gewährleisten.

Zusätzlich ist der korrekte Einsatz von SQL und damit verbunden auch prozeduralen Erweiterungen entscheidend. Die Datenbankanwendung stellt Abfragen an das Datenbanksystem und diese Abfragen müssen sicher ausgeführt werden können, so dass gängige Angriffe wie SQL-Injektionen geringe Erfolgsaussichten haben. Diesbezüglich ist der entsprechende Einsatz von Best Practices und auch die Befolgung von Standards für die Umgebung, in der die Anwendung arbeiten soll, besonders entscheidend. Ein Datenbanksystem ist sehr komplex und potenzielle Angriffe Zahlreich. Dementsprechend erfordern unterschiedliche Gefahren unterschiedliche Methoden für die Bewältigung. Durch die Erforschung vieler verschiedener Modelle und die Einteilung in die Triade der Informationssicherheit kann behauptet werden, dass der Einsatz mindestens einer Methode für jeden definierten Aspekt unbedingt erforderlich ist. Eine auf die gängigsten Gefahren angepasste Einrichtung von Sicherheitsmaßnahmen kann eine solide Grundsicherheit für Datenbankanwendungen gewährleistet. Für die detailliertere Einrichtung abgestimmt auf explizite Gefahren ist es in diesem Kontext besonders wichtig, aktuelle potenzielle Angriffe mit einzubeziehen und diese nach Möglichkeit im Einzelnen zu betrachten. Entwickler müssen bestimmten Vorgaben gegenüber von Stakeholder gerecht werden, um ein wirtschaftlich effizientes Modell zu erstellen. Somit sollte bei der Überlegung nach den richtigen Verfahren eine Absprache für Prioritäten unbedingt erfolgen.

Bezüglich der einzusetzenden Methoden und Modelle ist es besonders wichtig, modernste Technologien zu verwenden und darauf zu achten, dass verwendete Algorithmen nicht neuste Lücken aufweisen. Mit der rasant voranschreitenden Entwicklung künstlicher Intelligenz ist dieser Aspekt besonders hervorzuheben. Neue Möglichkeiten, altbewährte Algorithmen zu widerlegen, können bei der exponentiellen Entwicklung von KI tagtäglich entwickelt werden.

Auch im Kontext der Zugriffssteuerung und Authentifizierung und damit verbunden der digitalen Identität spielt dieser Aspekt eine besonders wichtige Rolle. Neben sehr modernen Methoden zu der Identitätsfälschung und neuen, raffinierten Möglichkeiten, künstliche Intelligenz für die eigenen Interessen einzusetzen, existieren weitere Probleme, die mit den altbewährten Methoden einhergehen. Beispielsweise wurden einige gängige Algorithmen für Verschlüsselungen lange durch erfolgreiche Angriffe widerlegt, werden aber noch immer eingesetzt.

Jedoch ist die Entwicklung von künstlicher Intelligenz nicht nur negativ zu betrachten: Ihr Einsatz sollte eine entscheidende Rolle in der Einrichtung von sicheren Datenbankanwendungen in der Zukunft spielen. So kann KI beispielsweise dabei helfen, die Feingranulare Zugriffskontrolle effektiver einzurichten und hierdurch eine besonders flexible und hohe Sicherheit für die Datenbankanwendung zu ermöglichen, was die weit verbreitete Rollenbasierte Zugriffskontrolle komplett ablösen könnte. Auch wenn die Möglichkeiten in diesem Gebiet sehr neu und unerforscht sind, sieht es mit einem Blick in die Zukunft danach aus, dass solche Modelle eine zentrale Rolle in kommenden Sicherheitsrichtlinien spielen werden.

Somit ist der Schutz der Anwendungsumgebung mithilfe des Einsatzes von Verschlüsselungen und Methoden für die Bewahrung der Integrität ein wesentlicher Bestandteil, wenn es darum geht, eine sichere Datenbankanwendung einzurichten. Das Missachten dieser Aspekte kann dazu führen, dass die Anwendung anfällig für diverse Angriffe ist, die in ihrer Anzahl und ihrer Ausführung ständig variieren. Auch deshalb ist eine möglichst solide Grundlage besonders wichtig, die sich auf die besagte Triade stützt. Zusätzlich ist ein vielversprechender Ansatz für die Zukunft auch die weitere Erforschung und Entwicklung der prozeduralen Erweiterungen von SQL. Auch wenn diese Aufgabe den einzelnen Entwicklern überlassen ist, ist davon auszugehen, dass zukünftig einige revolutionäre Erweiterungen kommen werden, die in Kombination mit künstlicher Intelligenz besonders effektiv sein können. Beispielsweise wäre es möglich, eine integrierte Authentifizierung- und Zugriffssteuerungs-Mechanismus in das Datenbankmanagementsystem selbst zu integrieren, damit die Einrichtung dieser um ein Vielfaches vereinfacht werden kann. Während bereits einige Fortschritte in diesem Bereich getätigt wurden, gibt es noch viel Raum für Verbesserungen und Innovationen. Darüber hinaus könnte die Integration von maschinellem Lernen und KI in prozedurale SQL-Erweiterungen neue Möglichkeiten eröffnen, um Bedrohungen zu erkennen und präventive Maßnahmen zu ergreifen. Indem Machine-Learning-Algorithmen Anomalien im Datenbankverhalten erkennen, könnten proaktive Sicherheitsmechanismen implementiert werden, um potenzielle Angriffe abzuwehren.

Insgesamt wird die Zukunft der Sicherheit von Datenbankanwendungen eng mit der Weiterentwicklung prozeduraler Erweiterungen von SQL verbunden sein. Durch kontinuierliche Forschung und Entwicklung können innovative Lösungen geschaffen werden, um die Sicherheit von Datenbankanwendungen zu verbessern und sie widerstandsfähiger gegenüber den sich ständig weiterentwickelnden Bedrohungen der Cyberwelt zu machen.

Literaturverzeichnis

[1] Abdullah, A. M. (2017). Advanced encryption standard (AES) algorithm to encrypt and decrypt data. *Cryptography and Network Security*, *16*(1), 11.

[2] Ahmad, K., & Karim, M. (2021). A method to prevent SQL injection attack using an improved parameterized stored procedure. *International Journal of Advanced Computer Science and Applications*, *12*(6).

[3] Albulayhi, K., Abuhussein, A., Alsubaei, F., & Sheldon, F. T. (2020, January). Fine-grained access control in the era of cloud computing: An analytical review. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 0748-0755). IEEE.

[4] Aragona, R., Calderini, M., Civino, R., Sala, M., & Zappatore, I. (2017). Generalised Round Functions for Block Ciphers and their Security. *arXiv preprint arXiv:1708.08814*.

[5] Bales, D. J. (2015). *Beginning Oracle PL/SQL* (Second Edition). Apress. https://doi.org/10.1007/978-1-4842-0737-6

[6] Bamberger, A., & Fernández, M. (2023, October). From Static to Dynamic Access Control Policies via Attribute-Based Category Mining. In *International Symposium on Logic-Based Program Synthesis and Transformation* (pp. 188-197). Cham: Springer Nature Switzerland.

[7] Bao, H. N. P., & Clavel, M. (2021). A model-driven approach for enforcing fine-grained access control for SQL queries. *SN Computer Science*, *2*(5), 370.

[8] Bazaz, T., & Khalique, A. (2016). A review on single sign on enabling technologies and protocols. *International Journal of Computer Applications*, *151*(11), 18-25.

[9] Beaulieu, A. (2009). *Einführung in SQL* (2. Auflage). O'Reilly.

[10] Beutelspacher, A., Schwenk, J., & Wolfenstetter, K.-D. (2022). *Moderne Verfahren der Kryptographie: Von RSA zu Zero-Knowledge und darüber hinaus* (9th ed. 2022). Springer Berlin Heidelberg, Imprint: Springer Spektrum. https://doi.org/10.1007/978-3-662-65718-8

[11] Biju, J. M., Gopal, N., & Prakash, A. J. (2019). Cyber attacks and its different types. *International Research Journal of Engineering and Technology*, *6*(3), 4849-4852.

[12] BRAUSE, Rüdiger, 2017. *Betriebssysteme: Grundlagen und Konzepte* [online]. 4. Aufl. 2017. Berlin, Heidelberg: Springer Berlin Heidelberg, Imprint: Springer Vieweg. ISBN 9783662541005. Verfügbar unter: https://doi.org/10.1007/978-3-662-54100-5

[13] Bundesamt für Sicherheit in der Informationstechnik," IT-Grundschutz", (2024). [Online]. Verfügbar: https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/BSI-Standards/bsi-standards_node.html Zugriff am: 06. März, 2024.

[14] Bundesamt für Sicherheit in der Informationstechnik, "Zweites Gesetz zur Erhöhung der Sicherheit informationstechnischer Systeme (IT-Sicherheitsgesetz 2.0)," Bundesamt für Sicherheit in der Informationstechnik, [Online]. Verfügbar: https://www.bsi.bund.de/DE/Das-BSI/Auftrag/Gesetze-und-Verordnungen/IT-SiG/2-0/it_sig-2-0_node.html Zugriff am: 03. März, 2024.

[15] "Bundeskriminalamt - Cybercrime," Bundeskriminalamt, [Online]. Verfügbar: https://www.bka.de/DE/UnsereAufgaben/Deliktsbereiche/Cybercrime/cybercrime_node.html Zugriff am: 05. März, 2024.

[16] Chirkova, R., & Yang, J. (2012). Materialized views. *Foundations and Trends® in Databases*, *4*(4), 295-405.

[17] Clarke, J. ([2016]). *SQL Hacking: SQL-Injektion auf relationale Datenbanken im Detail verstehen und abwehren : Anfälligkeit für SQL-Injektion erkennen, ausnutzen und Schäden beseitigen : Angriffe auf Datenbanken detailliert im Quellcode nachvollziehen : Tools kennen und nutzen: automatisierte Quellcodeprüfung, Automatisierung der blinden SQL-Injektion und mehr*. Franzis.

[18] Clarke-Salt, J. (2009). *SQL injection attacks and defense*. Elsevier.

[19] Cook, Sam, "Ransomware-Statistiken 2018–2022: Zahlen und Fakten," Comparitech, [Online]. Verfügbar: https://www.comparitech.com/de/antivirus/ransomware-statistiken/ Zugriff am: 01. März, 2024.

[20] Cordts, S., Blakowski, G., & Brosius, G. (2011). *Datenbanken für Wirtschaftsinformatiker: Nach dem aktuellen Standard SQL:2008*. Vieweg+Teubner Verlag / Springer Fachmedien Wiesbaden GmbH, Wiesbaden. https://doi.org/10.1007/978-3-8348-8192-2

[21] Cormen, T. H., Krieger-Hauwede, M., Leiserson, C. E., Lippert, K., Molitor, P., Rivest, R., & Stein, C. ([2017]; [2013]). *Algorithmen - Eine Einführung* (4., durchges. und korr. Aufl). De Gruyter Oldenbourg ([2017]) ; MIT Press ([2013]). https://doi.org/10.1515/9783110522013

[22] Darms, M., Haßfeld, S., & Fedtke, S. (2019). *IT-Sicherheit und Datenschutz im Gesundheitswesen: Leitfaden für Ärzte, Apotheker, Informatiker und Geschäftsführer in Klinik und Praxis*. Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-658-21589-7

[23] Das, S., Mitra, B., Atluri, V., Vaidya, J., & Sural, S. (2018). Policy Engineering in RBAC and ABAC. *From Database to Cyber Security: Essays Dedicated to Sushil Jajodia on the Occasion of His 70th Birthday*, 24-54.

[24] DB-Engines, "DB-Engines Ranking," DB-Engines, [Online]. Verfügbar: https://db-engines.com/de/ranking# Zugriff am: 05. März, 2024.

[25] Di Zio, Marco, et al. "Methodology for data validation 1.0." *Essnet Validat Foundation* (2016)

[26] Diaz, C. ([2022]). *Database Security: Problems and Solutions*. Mercury Learning and Information. https://doi.org/10.1515/9781683926627

[27] Diederichs, M. ([2023]). *Risikomanagement und Risikocontrolling* (5., vollständig überarbeitete und ergänzte Auflage). Verlag Franz Vahlen.

[28] Ebert, B. (2018). *Prozessoptimierung bei Industrie 4.0 durch Risikoanalysen: Gefährdungen erkennen und minimieren*. Springer Berlin Heidelberg, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-662-55729-7

[29] Eckert, C. ([2023]). *IT-Sicherheit: Konzepte – Verfahren – Protokolle*. De Gruyter Oldenbourg. https://doi.org/10.1515/9783110985115

2234

[30] Edler, F., Soden, M., & Hankammer, R. (2015). *Fehlerbaumanalyse in Theorie und Praxis*. Springer Berlin Heidelberg.

[31] Ernst, H., Schmidt, J., & Beneken, G. (2023). *Grundkurs Informatik: Grundlagen und Konzepte für die erfolgreiche IT-Praxis – Eine umfassende Einführung* (8th ed. 2023). Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-658-41779-6

[32] Fabry, H.-W., "Daten verschlüsseln mit Transparent Data Encryption (TDE)," Oracle, [Online]. Verfügbar: https://www.oracle.com/webfolder/technetwork/de/community/dbadmin/tipps/tde/index.html Zugriff am: 06. März, 2024.

[33] Faeskorn-Woyke, H., Bertelsmeier, B., & Riemer, P. (2007). Datenbanksysteme: Theorie und Praxis mit SQL2003, Oracle und MySQL. Pearson Studium.

[34] Federal Inf. Process. Stds. (NIST FIPS) – 197

[35] Ferle, M. (2023). *SnowPro™ Core Certification Companion: Hands-on Preparation and Practice* (1st ed. 2023). Apress, Imprint: Apress. https://doi.org/10.1007/978-1-4842-9078-1

[36] Feuerstein, S., Pribyl, B., & Dawes, C. (2015). *Oracle PL/SQL language pocket reference: [a guide to Oracle's PL/SQL language fundamentals]* (5. ed.). O'Reilly.

[37] Foggon, D. (2006). Stored Procedures. *Beginning ASP. NET 2.0 Databases: From Novice to Professional*, 415-457.

[38] FRÖHLICH, Lutz, 2021. *Oracle 19c/20c: Das umfassende Praxis-Handbuch*. 2021. Frechen: mitp-Verlag. ISBN 9783747500583

[39] Fuchs, E. (2021). *SQL Grundlagen und Datenbankdesign* (1. Ausgabe, Juli 2021). Herdt.

[40] Gedam, M. N., & Meshram, B. B. (2021). Database private security jurisprudence: a case study using oracle. *International Journal of Database Management Systems*, *13*(3), 01-21.

[41] Gleißner, W., & Romeike, F. (2005). Anforderungen an die Softwareunterstützung für das Risikomanagement. *Zeitschrift für Controlling & Management*, *49*(2), 154-164.

[42] Glörfeld, T. (2013). *SQL Server Security*. MITP-Verlags GmbH & Co. KG.

[43] Haakegaard, R., & Lang, J. (2015). The elliptic curve diffie-hellman (ecdh). *Online at https://koclab. cs. ucsb. edu/teaching/ecc/project/2015Projects/Haakegaard+ Lang. pdf*.

[44] Harich, T. W. (2021). *IT-Sicherheitsmanagement: das umfassende Praxis-Handbuch für IT-Security und technischen Datenschutz nach ISO 27001*. MITP-Verlags GmbH & Co. KG.

[45] Haufe, K., & Dzombeta, S. (2023). *Managementsystem zur Informationssicherheit: Aufbau und Betrieb gemäß Prozess-Referenzmodell der ISO/IEC TS 27022* (1. Auflage 2024). Schäffer-Poeschel.

[46] HELLER, Jon, 2023. *Pro Oracle SQL Development: Best Practices for Writing Advanced Queries* [online]. 2nd ed. 2023. Berkeley, CA: Apress, Imprint: Apress. ISBN 9781484288672. Verfügbar unter: https://doi.org/10.1007/978-1-4842-8867-2

[47] Heuer, A., Sattler, K.-U., & Saake, G. (2018). *Datenbanken – Konzepte und Sprachen* (2018. Aufl.). mitp-Verlag.

[48] Hu, V. C., Ferraiolo, D., Kuhn, R., Friedman, A. R., Lang, A. J., Cogdell, M. M., ... & Scarfone, K. (2013). Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, *800*(162), 1-54.

[49] Huang, H., Shao, F., Liu, J. & Du, H. (2013). Handling least privilege problem and role mining in RBAC. *Journal Of Combinatorial Optimization*, *30*(1), 63–86. https://doi.org/10.1007/s10878-013-9633-9

[50] Huang, J., & Lai, X. (2014). What is the effective key length for a block cipher: an attack on every practical block cipher. *Science China Information Sciences*, *57*, 1-11.

[51] Huawei Technologies Co., L. (2023). *Database Principles and Technologies - Based on Huawei GaussDB*. Springer Nature.

[52] Hutchinson, N. C., Manley, S., Federwisch, M., Harris, G., Hitz, D., Kleiman, S., & O'Malley, S. (1999, February). Logical vs. physical file system backup. In *OSDI* (Vol. 99, pp. 239-249).

[53] Hüffmeyer, M. (2019). Effiziente Gestaltung und Anwendung von attributbasierter Zugriffskontrolle für RESTful Services.

[54] IBM, "Was ist Datenbanksicherheit?," IBM, [Online]. Verfügbar: https://www.ibm.com/de-de/topics/database-security Zugriff am: 03. März, 2024.

[55] Ikromovna, A. Z. (2023). SQL (STRUCTURED QUERY LANGUAGE) STATISTICAL PACKAGES OF CAPABILITIES. *Best Journal of Innovation in Science, Research and Development*, *2*(12), 781-787. [GNM] Gigih Forda Nama – Implementation of Two-Factor Authentication

[56] Jajodia, S., & Mazumdar, C. (2015). *Information Systems Security: 11th International Conference, ICISS 2015, Kolkata, India, December 16-20, 2015. Proceedings* (1st ed.

2015). Springer International Publishing, Imprint: Springer. https://doi.org/10.1007/978-3-319-26961-0

[57] Jindal, P., & Singh, B. (2015). RC4 encryption-A literature survey. *Procedia Computer Science*, *46*, 697-705.

[58] Kara, O. (2024). Lower data attacks on Advanced Encryption Standard. *Turkish Journal of Electrical Engineering and Computer Sciences*, *32*(2), 338-357.

[59] Khairallah, M. (2022). *Hardware Oriented Authenticated Encryption Based on Tweakable Block Ciphers*. Springer.

[60] Kleuker, S. (2024). *Grundkurs Datenbankentwicklung: Von der Anforderungsanalyse zur komplexen Datenbankanfrage* (5th ed. 2024). Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-658-43023-8

[61] Køien, G. M., & Øverlier, L. (2023). A Call for Mandatory Input Validation and Fuzz Testing. *Wireless Personal Communications*, 1-13.

[62] Kuhn, D. (2015). *Oracle RMAN Database Duplication*. Apress. https://doi.org/10.1007/978-1-4842-1112-0

[63] Malcher, M. & Kuhn, D. (2019). Pro Oracle Database 18C Administration. In *Apress eBooks*. https://doi.org/10.1007/978-1-4842-4424-1

[64] Masiero, S., & Bailur, S. (2021). Digital identity for development: The quest for justice and a research agenda. *Information Technology for Development*, *27*(1), 1-12.

[65] McGiffen, M. (2022). *Pro Encryption in SQL Server 2022: Provide the Highest Level of Protection for Your Data* (1st ed. 2022). Apress, Imprint: Apress. https://doi.org/10.1007/978-1-4842-8664-7

[66] Meier, A., & Kaufmann, M. (2016). *SQL- & NoSQL-Datenbanken* (8., überarb. u. erw. Aufl. 2016). Springer Berlin Heidelberg, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-662-47664-2

[67] Meinel, C., & Sack, H. (2014). *Sicherheit und Vertrauen im Internet: Eine technische Perspektive*. Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-04834-1

[68] Morisset, C., Willemse, T. A. C. & Zannone, N. (2019). A framework for the extended evaluation of ABAC policies. *Cybersecurity*, *2*(1). https://doi.org/10.1186/s42400-019-0024-0

[69] Motero, C. D., Higuera, J. R. B., Higuera, J. B., Montalvo, J. A. S., & Gómez, N. G. (2021). On Attacking Kerberos Authentication Protocol in Windows Active Directory Services: A Practical Survey. *IEEE Access*, *9*, 109289-109319.

[70] Mustafa, O. & Lockard, R. P. (2019). Oracle Database Application Security. In *Apress eBooks*. https://doi.org/10.1007/978-1-4842-5367-0

[71] Müller, K.-R. (2018). *IT-Sicherheit mit System: Integratives IT-Sicherheits-, Kontinuitäts- und Risikomanagement – Sichere Anwendungen – Standards und Practices* (6. Aufl. 2018). Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-658-22065-5

[72] Nanda, A., & Feuerstein, S. (2005). *Oracle PL/SQL for DBAs: Security, Scheduling, Performance & More*. " O'Reilly Media, Inc.".

[73] Neuman, C., Yu, T., Hartman, S., & Raeburn, K. (2005). *The Kerberos network authentication service (V5)* (No. rfc4120).

[74] Nir, Y., & Langley, A. (2018). *ChaCha20 and Poly1305 for IETF Protocols* (No. rfc8439).

[75] Ometov, A., Bezzateev, S., Mäkitalo, N., Andreev, S., Mikkonen, T., & Koucheryavy, Y. (2018). Multi-factor authentication: A survey. *Cryptography*, *2*(1), 1.

[76] Open Web Application Security Project, "Top 10 Web Application Security Risks," OWASP, [Online]. Verfügbar: https://owasp.org/www-project-top-ten/ Zugriff am: Mar. 5, 2024.

[77] Open Web Application Security Project, "A03:2021 - Injection," OWASP, [Online]. Verfügbar: https://owasp.org/Top10/A03_2021-Injection/ Zugriff am: 05. März, 2024.

[78] Oracle Corporation, (n.d.), "Backup and Recovery," Oracle Help Center, [Online]. Verfügbar: https://docs.oracle.com/cd/B19306_01/server.102/b14220/backrec.htm Zugriff am: 06. März, 2024.

[79] Oracle Corporation, (n.d.), "Database PL/SQL Language Reference - PL/SQL Packages," Oracle Help Center, [Online]. Verfügbar:

https://docs.oracle.com/en/database/oracle/oracle-database/19/lnpls/plsql-packages.html#GUID-8D02540E-C697-4498-9261-848F6D4E5CB5 Zugriff am: 07. März, 2024.

[80] Oracle Corporation, (n.d.), "Database Security Guide," Oracle Help Center, [Online]. Verfügbar: https://docs.oracle.com/en/database/oracle/oracle-database/12.2/dbseg/introduction-to-auditing.html#GUID-8F354963-F0E6-4B8F-BFF4-891278E954D5 Zugriff am: Mar. 6, 2024.

[81] Oracle Corporation, (n.d.), "PL/SQL Packages and Types Reference - DBMS_CRYPTO," Oracle Help Center, [Online]. Verfügbar: https://docs.oracle.com/en/database/oracle/oracle-database/19/arpls/DBMS_CRYPTO.html#GUID-1C98C203-29EF-488D-A5FA-42AD4BD7718D Zugriff am: 06. März, 2024.

[82] Oracle Corporation, (n.d.), "Security Guide - Operating System Authentication of Users," Oracle Help Center, [Online]. Verfügbar: https://docs.oracle.com/en/database/oracle/oracle-database/19/dbseg/configuring-authentication.html#GUID-37BECE32-58D5-43BF-A098-97936D66968F Zugriff am: 06. März, 2024.

[83] Oracle Corporation, (n.d.), "SQL Language Reference - ORA_HASH," Oracle Help Center, [Online]. Verfügbar: https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/ORA_HASH.html#GUID-0349AFF5-0268-43CE-8118-4F96D752FDE6 Zugriff am: 06. März, 2024.

[84] Oracle Corporation, (n.d.), “Database 2 Day Developer's Guide: Developing and Using Stored Procedures.” Oracle Help Center, [Online]. Verfügbar: https://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.htm Zugriff am 15. März, 2024.

[85] Oracle Corporation, (n.d.), “Database PL/SQL User's Guide and Reference: Handling PL/SQL Errors. Oracle Help Center.“, [Online]. Verfügbar: https://docs.oracle.com/cd/B19306_01/appdev.102/b14261/errors.htm Zugriff am 15. März, 2024.

[86] Oracle Corporation, (n.d.), “Oracle Security Developer Tools Reference, 10g Release 2 (10.1.2).”, [Online]. Verfügbar: https://docs.oracle.com/cd/B14099_19/idmanage.1012/b15975/crypto.htm Zugriff am 17. März, 2024.

[87] Oracle Corporation, (n.d.), “Oracle Documentation Home: System Administration Guide: Basic Administration, Chapter 2: Working with the Solaris Management Console

(Tasks), Becoming Superuser (root) or Assuming a Role.", [Online]. Verfügbar: https://docs.oracle.com/cd/E19120-01/open.solaris/819-2379/smcover-149/index.html Zugriff am 15. März 2024.

[88] Oracle Corporation, (n.d.), "SQL Language Reference: Create Profile." Oracle Help Center, [Online]: Verfügbar. https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/CREATE-PROFILE.html#GUID-ABC7AE4D-64A8-4EA9-857D-BEF7300B64C3 Zugriff am 15. März, 2024.

[89] Oracle Corporation, (n.d.),"Trusted Extensions User's Guide," Oracle, [Online]. Verfügbar: https://docs.oracle.com/cd/E36784_01/html/E36841/ugintro-10.html#scrolltoc Zugriff am: 06. März, 2024.

[90] Plickert, Philip, "Russische Hacker erpressen Royal Mail," Frankfurter Allgemeine Zeitung, [Online]. Verfügbar: https://www.faz.net/aktuell/wirtschaft/unternehmen/russische-hacker-erpressen-royal-mail-18601163.html Zugriff am: 28. Februar, 2024.

[91] Porcedda, M. G., & Wall, D. S. (2019, June). Cascade and chain effects in big data cybercrime: Lessons from the TalkTalk hack. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)* (pp. 443-452). IEEE.

[92] RAD, Reza, 2018. *Pro Power BI Architecture: Sharing, Security, and Deployment Options for Microsoft Power BI Solutions* [online]. Berkeley, CA: Apress, Imprint: Apress. ISBN 9781484240151. Verfügbar unter: https://doi.org/10.1007/978-1-4842-4015-1

[93] Rahm, E., Saake, G., & Sattler, K.-U. (2015). *Verteiltes und Paralleles Datenmanagement: von verteilten Datenbanken zu Big Data und Cloud*. Springer Vieweg.

[94] Raimunda Matulevičius & Henri Lakk – (2015) – A Model-driven Role-based Access Control for SQL Databases – Complex Systems Informatics and Modeling Quarterly (CSIMQ), Issue 3, Pages 35-62

[95] RAMEY, Kenneth, 2016. *Pro Oracle Identity and Access Management Suite* [online]. Berkeley, CA: Apress, Imprint: Apress. ISBN 9781484215210. Verfügbar unter: https://doi.org/10.1007/978-1-4842-1521-0

[96] Rob, P., & Coronel, C. (1997). *Database systems: design, implementation and management* (3. ed.). Thomson.

[97] Rosenkranz, F., & Missler-Behr, M. (2005). *Unternehmensrisiken erkennen und managen: Einführung in die quantitative Planung*. Springer-Verlag Berlin Heidelberg. https://doi.org/10.1007/b137724

[98] Saake, G., Sattler, K.-U., & Heuer, A. (2018). *Datenbanken: Konzepte und Sprachen* (Sechste Auflage). mitp.

[99] Sánchez, C., Sipma, H. B., Manna, Z., & Gill, C. D. (2006, October). Efficient distributed deadlock avoidance with liveness guarantees. In *Proceedings of the 6th ACM & IEEE International conference on Embedded software* (pp. 12-20).

[100] Sauer, H., & Grieger, K. (2002). *Relationale Datenbanken - Theorie und Praxis: [berücksichtigt ausführlich die Neuerungen des SQL-3-Standards]* (5., aktualisierte und erw. Aufl.). Addison-Wesley.

[101] Schendera, C. F. ([2015]). *SQL mit SAS: Band 1: PROC SQL für Einsteiger*. Oldenbourg Wissenschaftsverlag. https://doi.org/10.1524/9783486704563

[102] Schicker, E. (2017). *Datenbanken und SQL: Eine praxisorientierte Einführung mit Anwendungen in Oracle, SQL Server und MySQL* (5. Aufl. 2017). Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-658-16129-3

[103] SCHMEH, Klaus, 2016. *Kryptografie: Verfahren, Protokolle, Infrastrukturen*. 6., aktualisierte Auflage. Heidelberg: dpunkt.verlag. ISBN 9783864919084

[104] Schubert, M. (2007). *Datenbanken: Theorie, Entwurf und Programmierung relationaler Datenbanken* (2., ⁿberarbeitete Auflage). B. G. Teubner Verlag / GWV Fachverlage GmbH, Wiesbaden. https://doi.org/10.1007/978-3-8351-9108-2

[105] SIEBEN, Jürgen, 2023. *Oracle PL/SQL: Das umfassende Handbuch*. 4th ed. Bonn: Rheinwerk Verlag. ISBN 3-8362-9632-2

[106] Singh, C., Thakkar, R., & Warraich, J. (2023). IAM identity Access Management—importance in maintaining security systems within organizations. *European Journal of Engineering and Technology Research*, *8*(4), 30-38.

[107] Souror, S., El-Fishawy, N., & Badawy, M. (2021, July). SCKHA: a new stream cipher algorithm based on key hashing and Splitting technique. In *2021 International Conference on Electronic Engineering (ICEEM)* (pp. 1-7). IEEE.

[108] Special Publication (NIST SP) - 800-207 – Zero Trust Architecture

[109] SPENDOLINI, Scott, 2013. *Expert Oracle Application Express Security* [online]. Berkeley, CA: Apress. ISBN 9781430247326. Verfügbar unter: https://doi.org/10.1007/978-1-4302-4732-6

[110] Statistisches Bundesamt – Zahl der Woche: Knapp ein Viertel aller Erwerbstätigen arbeitete im Jahr 2022 im Homeoffice

https://www.destatis.de/DE/Presse/Pressemitteilungen/Zahl-der-Woche/2023/PD23_28_p002.html Zugriff am 04. März, 2024.

[111] Streim, Andres & Beerlink, Kai Pascal, "Deutsche Wirtschaft drückt bei Künstlicher Intelligenz aufs Tempo," Bitkom, [Online]. Verfügbar: https://www.bitkom.org/Presse/Presseinformation/Deutsche-Wirtschaft-drueckt-bei-Kuenstlicher-Intelligenz-aufs-Tempo Zugriff am: 05. März, 2024.

[112] Streim, Andres & Kuhlenkamp, Felix, "Organisierte Kriminalität greift verstärkt die deutsche Wirtschaft an," Bitkom, [Online]. Verfügbar: https://www.bitkom.org/Presse/Presseinformation/Organisierte-Kriminalitaet-greift-verstaerkt-deutsche-Wirtschaft-an Zugriff am: 05. März, 2024.

[113] Tsolkas, A., & Schmidt, K. (2017). *Rollen und Berechtigungskonzepte: Identity- und Access-Management im Unternehmen* (2. Aufl. 2017). Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-658-17987-8

[114] Türker, C., & Saake, G. (2006). *Objektrelationale Datenbanken: ein Lehrbuch* (1. Aufl.). dpunkt-Verl.

[115] Unterstein, M., & Matthiessen, G. (2013). *Anwendungsentwicklung mit Datenbanken* (5. Aufl.). Springer Vieweg.

[116] Vasiliadis, D. C., Rizos, G. E., Stergiou, E., & Margariti, S. V. (2007, August). A trusted network model using the lightweight directory access protocol. In *Proceedings of AIC* (pp. 252-256).

[117] VAZQUEZ, Antonio, 2019. *Practical LPIC-3 300: Prepare for the Highest Level Professional Linux Certification* [online]. 1st ed. 2019. Berkeley, CA: Apress, Imprint: Apress. ISBN 9781484244739. Verfügbar unter: https://doi.org/10.1007/978-1-4842-4473-9

[118] Wendzel, S. (2021). *IT-Sicherheit für TCP/IP- und IoT-Netzwerke: Grundlagen, Konzepte, Protokolle, Härtung* (2nd ed. 2021). Springer Fachmedien Wiesbaden, Imprint: Springer Vieweg. https://doi.org/10.1007/978-3-658-33423-9

[119] Wilkinson, Marilyn, "Die fünf teuersten Ransomware-Attacken & was wir daraus lernen können," Prolion, [Online]. Verfügbar: https://prolion.com/de/blog/die-fuenf-

teuersten-ransomware-attacken-was-wir-daraus-lernen-koennen/ Zugriff am: Feb. 28, 2024.

[120] Witt, K. U., & Witt, K. U. (2014). Asymmetrische Verschlüsselung. *Algebraische und zahlentheoretische Grundlagen für die Informatik: Gruppen, Ringe, Körper, Primzahltests, Verschlüsselung*, 165-180.

[121] Wong, D., & Langenau, F. (2023). *Kryptografie in der Praxis: Eine Einführung in die bewährten Tools, Frameworks und Protokolle*. dpunkt.verlag.

[122] Yunus, M. A. M., Brohan, M. Z., Nawi, N. M., Surin, E. S. M., Najib, N. A. M., & Liang, C. W. (2018). Review of SQL injection: problems and prevention. *JOIV: International Journal on Informatics Visualization*, *2*(3-2), 215-219.

[123] Zhang, L., Tan, C., & Yu, F. (2017). An improved rainbow table attack for long passwords. *Procedia Computer Science*, *107*, 47-52.

Anhang

--DROP-Befehle zum löschen

DROP TABLE Ut_Vorgesetzte CASCADE CONSTRAINTS;

DROP TABLE Ut_Mitarbeiter CASCADE CONSTRAINTS;

DROP TABLE streetwear CASCADE CONSTRAINTS;

DROP TABLE sportartikel CASCADE CONSTRAINTS;

DROP TABLE Nicht_Veroeffentlichte_Projekte CASCADE CONSTRAINTS;

DROP TABLE Bestellungen CASCADE CONSTRAINTS;

DROP TABLE Lagerbestand CASCADE CONSTRAINTS;

DROP TABLE Abteilung CASCADE CONSTRAINTS;

DROP SEQUENCE ma_sq;

DROP SEQUENCE vg_sq;

DROP SEQUENCE sw_sq;

DROP SEQUENCE sp_sq;

DROP SEQUENCE np_sq;

DROP SEQUENCE bst_sq;

DROP SEQUENCE lg_sq;

DROP SEQUENCE abt_sq;

----------Erstellen von Sequenzen und Tabellen

```
CREATE SEQUENCE ma_sq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE vg_sq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE sw_sq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE sp_sq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE np_sq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE bst_sq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE lg_sq START WITH 1 INCREMENT BY 1;

CREATE SEQUENCE abt_sq START WITH 1 INCREMENT BY 1;



CREATE TABLE Streetwear (
    SW_Artikel_ID INT PRIMARY KEY,
    Bezeichnung VARCHAR2(50),
    Kategorie VARCHAR2(50),
    Beschreibung VARCHAR2(200),
    Preis NUMBER,
    Bestand NUMBER,
    Material VARCHAR2(100),
    Groesse VARCHAR2(25),
    Farbe VARCHAR2(50),
    Marke VARCHAR2(100)
);

ALTER TABLE streetwear
```

```
ADD CONSTRAINT check_groesse
CHECK (groesse != 'XXL');



CREATE TABLE Sportartikel (
    SP_Artikel_ID INT PRIMARY KEY,
    Bezeichnung VARCHAR2(50),
    Kategorie VARCHAR2(50),
    Beschreibung VARCHAR2(200),
    Preis NUMBER,
    Bestand NUMBER,
    Material VARCHAR2(100),
    Groesse VARCHAR2(25),
    Farbe VARCHAR2(50),
    Marke VARCHAR2(100)
);


CREATE TABLE Lagerbestand (
    Lagerbestand_ID INT GENERATED BY DEFAULT AS IDENTITY PRIMARY KEY,
    Bezeichnung VARCHAR2(50),
    Kategorie VARCHAR2(50),
    Beschreibung VARCHAR2(200),
    Preis NUMBER,
```

```
    Bestand NUMBER,
    Material VARCHAR2(100),
    Groesse VARCHAR2(25),
    Farbe VARCHAR2(50),
    Marke VARCHAR2(100),
    sw_artikel_id INT,
    sp_artikel_id INT
);
ALTER TABLE Lagerbestand
ADD(CONSTRAINT lg_sw_fk FOREIGN KEY (sw_artikel_id)
REFERENCES streetwear(sw_artikel_id));

ALTER TABLE Lagerbestand
ADD(CONSTRAINT lg_sp_fk FOREIGN KEY (sp_artikel_id)
REFERENCES Sportartikel(sp_artikel_id));


CREATE TABLE Abteilung (
    Abteilungs_ID INT PRIMARY KEY,
    Bezeichnung VARCHAR2(50),
    Standort VARCHAR2(50)
);
```

```
CREATE TABLE Ut_Mitarbeiter (
    Mitarbeiter_ID INT PRIMARY KEY,
    Vorname VARCHAR2(50),
    Nachname VARCHAR2(50),
    Position VARCHAR2(100),
    Gehalt NUMBER,
    Abteilungs_ID INT,
    Geburtsdatum DATE
);
ALTER TABLE Ut_Mitarbeiter
ADD(CONSTRAINT abt_fk FOREIGN KEY (Abteilungs_ID)
REFERENCES Abteilung(Abteilungs_ID));

CREATE TABLE Ut_Vorgesetzte (
    Vorgesetzte_ID INT PRIMARY KEY,
    Vorname VARCHAR2(50),
    Nachname VARCHAR2(50),
    Position VARCHAR2(100),
    Gehalt NUMBER,
    Abteilungs_ID INT,
    Geburtsdatum DATE,
    Mitarbeiter_ID INT
);
```

```
ALTER TABLE Ut_Vorgesetzte

ADD(CONSTRAINT mi_fk FOREIGN KEY (Mitarbeiter_ID)

REFERENCES Ut_Mitarbeiter(Mitarbeiter_ID));



ALTER TABLE Ut_Vorgesetzte

ADD(CONSTRAINT abt_fk2 FOREIGN KEY (Abteilungs_ID)

REFERENCES Abteilung(Abteilungs_ID));




CREATE TABLE Bestellungen (

   Bestell_ID INT PRIMARY KEY,

   Menge NUMBER,

   Einzelpreis DECIMAL(10, 2),

   Gesamtpreis DECIMAL(10, 2) DEFAULT NULL,

   SP_Artikel_ID INT,

   SW_Artikel_ID INT,

   Mitarbeiter_ID INT

   );

ALTER TABLE Bestellungen

ADD(CONSTRAINT ma_fk FOREIGN KEY (Mitarbeiter_ID)

REFERENCES Ut_Mitarbeiter(Mitarbeiter_ID));



ALTER TABLE Bestellungen
```

```
ADD(CONSTRAINT sp_fk FOREIGN KEY (sp_artikel_id)

REFERENCES Sportartikel(sp_Artikel_ID));


ALTER TABLE Bestellungen

ADD(CONSTRAINT sw_fk FOREIGN KEY (sw_artikel_id)

REFERENCES streetwear(sw_artikel_ID));



CREATE TABLE Nicht_Veroeffentlichte_Projekte (

    ProjektID INT PRIMARY KEY,

    Bezeichnung VARCHAR2(50),

    Beschreibung VARCHAR2(500),

    Startdatum DATE,

    Enddatum DATE,

    Status VARCHAR2(50),

    Verantwortlicher INT,

    Mitarbeiter INT

);

ALTER TABLE Nicht_Veroeffentlichte_Projekte

ADD(CONSTRAINT vp_fk FOREIGN KEY (Verantwortlicher)

REFERENCES Ut_Vorgesetzte(Vorgesetzte_ID));


ALTER TABLE Nicht_Veroeffentlichte_Projekte
```

```
ADD(CONSTRAINT mp_fk FOREIGN KEY (Mitarbeiter)

REFERENCES Ut_Mitarbeiter(Mitarbeiter_ID));


ALTER TABLE Nicht_Veroeffentlichte_Projekte

ADD (hashwert NUMBER);

-------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------

--Erzeugen von TRIGGERn


--TRIGGER für Gesamtpreis

CREATE OR REPLACE TRIGGER berechne_gesamtpreis

BEFORE INSERT OR UPDATE ON Bestellungen

FOR EACH ROW

BEGIN

   IF :NEW.sp_artikel_id IS NOT NULL THEN

      :NEW.gesamtpreis := :NEW.einzelpreis * :NEW.menge;

   ELSIF :NEW.sw_Artikel_ID IS NOT NULL THEN

      :NEW.gesamtpreis := :NEW.einzelpreis * :NEW.menge;

   END IF;

END;


--TRIGGER zum automatischen einfügen von Datensätzen in die Tabelle Lagerbe-
stand

CREATE OR REPLACE TRIGGER Streetwear_After_Insert
```

```
AFTER INSERT ON Streetwear

FOR EACH ROW

BEGIN

   INSERT INTO Lagerbestand (Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

   VALUES (:NEW.Bezeichnung, :NEW.Kategorie, :NEW.Beschreibung, :NEW.Preis, :NEW.Bestand, :NEW.Material, :NEW.Groesse, :NEW.Farbe, :NEW.Marke);

END;



CREATE OR REPLACE TRIGGER Sportartikel_After_Insert

AFTER INSERT ON Sportartikel

FOR EACH ROW

BEGIN

   INSERT INTO Lagerbestand (Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

   VALUES (:NEW.Bezeichnung, :NEW.Kategorie, :NEW.Beschreibung, :NEW.Preis, :NEW.Bestand, :NEW.Material, :NEW.Groesse, :NEW.Farbe, :NEW.Marke);

END;



--Trigger um ein Lost-Update zu verhindern

CREATE OR REPLACE TRIGGER lost_update

BEFORE UPDATE ON Lagerbestand

FOR EACH ROW
```

BEGIN

IF :OLD.Bestand != :NEW.Bestand THEN

RAISE_APPLICATION_ERROR(-20001, 'Datensatz wurde von einer anderen Transaktion geändert.');

END IF;

END;

--Inserts in Abteilungen

INSERT INTO Abteilung (Abteilungs_ID, Bezeichnung, Standort)

VALUES

(abt_sq.NEXTVAL, 'Management', 'Köln');

INSERT INTO Abteilung (Abteilungs_ID, Bezeichnung, Standort)

VALUES

(abt_sq.NEXTVAL, 'Verkauf', 'Köln');

INSERT INTO Abteilung (Abteilungs_ID, Bezeichnung, Standort)

VALUES

(abt_sq.NEXTVAL, 'Marketing', 'Gummersbach');

--Inserts in Mitarbeiter

INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES(ma_sq.NEXTVAL, 'Anna', 'Schmidt', 'Manager', 3600, 1, TO_DATE('1988.09.20', 'YYYY-MM-DD'));

INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES(ma_sq.NEXTVAL,'Peter', 'Müller', 'Vertrieb', 3500, 1, TO_DATE('1985-02-10', 'YYYY-MM-DD'));

INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Hans', 'Schulz', 'Consultant', 3800, 1, TO_DATE('1987-07-12', 'YYYY-MM-DD'));

INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Carina', 'Wagner', 'Kassierer', 2100, 2, TO_DATE('1986-03-08', 'YYYY-MM-DD'));

INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Carina', 'Wagner', 'Verkäufer', 2400, 2, TO_DATE('1986-03-08', 'YYYY-MM-DD'));

```
INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES

(ma_sq.NEXTVAL, 'Max', 'Meier', 'Verkäufer', 2400, 2, TO_DATE('1990.05.15', 'YYYY-MM-DD'));


INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL,'Laura', 'Meier', 'Ideation', 3200, 3, TO_DATE('1992-11-25', 'YYYY-MM-DD'));


INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Julia', 'Klein', 'Ideation', 3200, 3, TO_DATE('1991-06-18', 'YYYY-MM-DD'));


INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Erik', 'Lange', 'Analyst', 3400, 3, TO_DATE('1989-10-30', 'YYYY-MM-DD'));


INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Sarah', 'Schneider', 'Abteilungsleiter', 7500, 1, TO_DATE('1978-08-15', 'YYYY-MM-DD'));
```

```
INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Jannick', 'Jamann', 'Abteilungsleiter', 6000, 2, TO_DATE('1992-02-23', 'YYYY-MM-DD'));


INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Lothar', 'Schwarz', 'Abteilungsleiter', 6600, 3, TO_DATE('1980-04-25', 'YYYY-MM-DD'));


--------------------------------------------------------------------------------------------------------------------------------

--Inserts in Vorgesetzte

INSERT INTO Ut_Vorgesetzte(Vorgesetzte_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum, Mitarbeiter_ID)

VALUES

(vg_sq.NEXTVAL, 'Sarah', 'Schneider', 'Abteilungsleiter', 7500, 1, TO_DATE('1978-08-15', 'YYYY-MM-DD'), 1);


INSERT INTO Ut_Mitarbeiter (Mitarbeiter_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum)

VALUES (ma_sq.NEXTVAL, 'Jannick', 'Jamann', 'Abteilungsleiter', 6000, 2, TO_DATE('1992-02-23', 'YYYY-MM-DD'));


INSERT INTO Ut_Vorgesetzte(Vorgesetzte_ID, Vorname, Nachname, Position, Gehalt, Abteilungs_ID, Geburtsdatum, Mitarbeiter_ID)

VALUES
```

(vg_sq.NEXTVAL, 'Lothar', 'Schwarz', 'Abteilungsleiter', 6600, 3, TO_DATE('1980-04-25', 'YYYY-MM-DD'), 3);

--Inserts in Streetwear

INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES

(sw_sq.NEXTVAL, 'T-Shirt', 'Oberteile', 'Ein bequemes T-Shirt aus Baumwolle.', 20.00, 50, 'Baumwolle', 'M', 'Weiß', 'Puma');

INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Hoodie', 'Oberteile', 'Ein warmer Kapuzenpullover.', 40.00, 40, 'Baumwolle', 'L', 'Grau', 'Nike');

INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Sneakers', 'Schuhe', 'Stylische Sneakers für den Alltag.', 60.00, 25, 'Textil', '39', 'Schwarz', 'Adidas');

INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Shorts', 'Hosen', 'Kurze Shorts für den Sommer.', 30.00, 35, 'Baumwolle', 'S', 'Grün', 'Jako');

```
INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis,
Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL,'Trainingshose', 'Hosen', 'Ein luftdurchlässiger Trainingsan-
zug.', 35.00, 30, 'Polyester', 'M', 'Schwarz', 'Hummel');


INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis,
Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Cap', 'Accessoires', 'Eine stylische Baseballkappe.', 25.00,
45, 'Textil', 'Einheitsgröße', 'Rot', 'Ellesse');


INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis,
Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Jacke', 'Oberteile', 'Eine leichte Jacke für den Übergang.',
70.00, 20, 'Polyester', 'XL', 'Blau', 'New Era');


INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis,
Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'T-Shirt', 'Oberteile', 'Ein sportliches T-Shirt.', 20.00, 50,
'Baumwolle', 'L', 'Weiß', 'Under Armour');


INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis,
Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Shorts', 'Hosen', 'Leichte Shorts für Training und Freizeit.',
30.00, 40, 'Polyester', 'M', 'Blau', 'Champion');


INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis,
Bestand, Material, Groesse, Farbe, Marke)
```

VALUES(sw_sq.NEXTVAL, 'Hoodie', 'Oberteile', 'Ein bequemer Kapuzenpullover.', 45.00, 35, 'Baumwolle', 'XL', 'Grau', 'Puma');

--Inserts in Sportartikel

INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Fußball', 'Fußball', 'Ein gut verarbeiteter Fußball.', 25.00, 50, 'Leder', '5', 'Weiß', 'Adidas');

INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Laufschuhe', 'Schuhe', 'Leichte Laufschuhe mit guter Polsterung.', 80.00, 30, 'Mesh', '42', 'Schwarz', 'Nike');

INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Tennisschläger', 'Tennis', 'Ein hochwertiger Tennisschläger für Wettkämpfe.', 120.00, 20, 'Graphit', 'M', 'Rot', 'Wilson');

INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Handball', 'Handball', 'Ein hochwertiger Handball mit festem Griff.', 30.00, 40, 'Synthetik', '1', 'Blau', 'Molten');

```
INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Basketball', 'Basketball', 'Ein schwarzer Basketball mit orangenen Streifen.', 40.00, 35, 'Gummi', '7', 'Orange', 'Spalding');


INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Golfbälle', 'Golf', 'Eine Packung mit hochwertigen Golfbällen.', 20.00, 25, 'Kunststoff', NULL, 'Weiß', 'Titleist');


INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Fahrradhelm', 'Fahrrad', 'Ein sicherer Fahrradhelm für Radfahrer', 50.00, 45, 'Polycarbonat', NULL, 'Rot', 'Bell');


INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Schwimmbrille', 'Schwimmen', 'Eine bequeme Schwimmbrille.', 15.00, 60, 'Kunststoff', NULL, 'Blau', 'Speedo');


INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sp_sq.NEXTVAL, 'Skateboard', 'Skateboarding', 'Ein hochwertiges Skateboard.', 70.00, 20, 'Holz', '8', 'Blau', 'Element');


INSERT INTO Sportartikel (sp_Artikel_ID, Bezeichnung, Kategorie, Beschreibung, Preis, Bestand, Material, Groesse, Farbe, Marke)
```

VALUES(sp_sq.NEXTVAL, 'Jogginganzug', 'Laufen', 'Ein bequemer Jogginganzug.', 60.00, 30, 'Baumwolle', NULL, 'Grau', 'Nike');

--Inserts in Nicht veröffentlichte Projekte

INSERT INTO Nicht_Veroeffentlichte_Projekte(ProjektID, Bezeichnung, Beschreibung, Startdatum, Enddatum, Verantwortlicher, Mitarbeiter)

VALUES(np_sq.NEXTVAL, 'Sportartikel-Innovation', 'Forschungs- und Entwicklungsprojek', TO_DATE('2024-06-01', 'YYYY-MM-DD'), TO_DATE('2024-12-31', 'YYYY-MM-DD'), 1, 2);

INSERT INTO Nicht_Veroeffentlichte_Projekte(ProjektID, Bezeichnung, Beschreibung, Startdatum, Enddatum, Verantwortlicher, Mitarbeiter)

VALUES(np_sq.NEXTVAL, 'Sportartikel-Innovation', 'Forschungs- und Entwicklungsprojek', TO_DATE('2024-06-01', 'YYYY-MM-DD'), TO_DATE('2024-12-31', 'YYYY-MM-DD'), 1, 3);

INSERT INTO Nicht_Veroeffentlichte_Projekte(ProjektID, Bezeichnung, Beschreibung, Startdatum, Enddatum, Verantwortlicher, Mitarbeiter)

VALUES(np_sq.NEXTVAL, 'Sportartikel-Innovation', 'Forschungs- und Entwicklungsprojek', TO_DATE('2024-06-01', 'YYYY-MM-DD'), TO_DATE('2024-12-31', 'YYYY-MM-DD'), 1, 4);

INSERT INTO Nicht_Veroeffentlichte_Projekte(ProjektID, Bezeichnung, Beschreibung, Startdatum, Enddatum, Verantwortlicher, Mitarbeiter)

VALUES(np_sq.NEXTVAL, 'Neues Sneaker-Design', 'Entwicklung eines neuen Sneaker-Designs.', TO_DATE('2024-04-01', 'YYYY-MM-DD'), TO_DATE('2024-09-30', 'YYYY-MM-DD'), 2, 1);

INSERT INTO Nicht_Veroeffentlichte_Projekte(ProjektID, Bezeichnung, Beschreibung, Startdatum, Enddatum, Verantwortlicher, Mitarbeiter)

VALUES(np_sq.NEXTVAL, 'Neues Sneaker-Design', 'Entwicklung eines neuen Sneaker-Designs.', TO_DATE('2024-04-01', 'YYYY-MM-DD'), TO_DATE('2024-09-30', 'YYYY-MM-DD'), 2, 5);

--Inserts in Bestellungen

INSERT INTO bestellungen (bestell_id, menge, einzelpreis, sp_artikel_id, sw_artikel_id, mitarbeiter_id)

VALUES (bst_sq.NEXTVAL, 5, 30.00, 3, NULL, 2);

INSERT INTO bestellungen (bestell_id, menge, einzelpreis, sp_artikel_id, sw_artikel_id, mitarbeiter_id)

VALUES (bst_sq.NEXTVAL, 2, 15.00, NULL, 1, 4);

INSERT INTO bestellungen (bestell_id, menge, einzelpreis, sp_artikel_id, sw_artikel_id, mitarbeiter_id)

VALUES (bst_sq.NEXTVAL, 3, 45.00, NULL, 1, 4);

--Anzeige aller Tabellen mithilfe der SELECT-Anweisung

SELECT*FROM streetwear;

SELECT*FROM Sportartikel;

SELECT*FROM Nicht_Veroeffentlichte_Projekte;

SELECT*FROM ut_mitarbeiter;

SELECT*FROM ut_vorgesetzte;

SELECT*FROM Bestellungen;

SELECT*FROM Lagerbestand;

SELECT*FROM Abteilung;

--SELECT-Abfrage zur Ausgabe von Mitarbeitern über 35

SELECT DISTINCT mitarbeiter_id, nachname, geburtsdatum

FROM ut_mitarbeiter

GROUP BY mitarbeiter_id, nachname, geburtsdatum

HAVING (SYSDATE - geburtsdatum) > 35*365

ORDER BY mitarbeiter_id ASC;

--Ein einfacher Join zur Ausgabe der Vorgesetzten über 44

SELECT DISTINCT m.mitarbeiter_id, m.nachname, m.geburtsdatum

FROM ut_mitarbeiter m

JOIN ut_vorgesetzte v ON m.nachname = v.nachname

WHERE v.vorgesetzte_id IS NOT NULL

```
AND (SYSDATE - m.geburtsdatum) > 44*365

ORDER BY m.mitarbeiter_id ASC;



--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------

--VIEWS

DROP VIEW vw_ut_mitarbeiter;

CREATE VIEW vw_ut_mitarbeiter AS

SELECT*FROM ut_mitarbeiter;

SELECT*FROM vw_ut_mitarbeiter;

GRANT SELECT ON vw_ut_mitarbeiter TO user_1;



DROP VIEW Mitarbeiter_View;

CREATE VIEW Mitarbeiter_View AS

SELECT * FROM ut_Mitarbeiter;



--Eingeschränkte Spalten

DROP VIEW Mitarbeiter_Namen;

CREATE VIEW Mitarbeiter_Namen AS

SELECT Vorname, Nachname FROM ut_Mitarbeiter;





--Aggregation in der View

DROP VIEW gehalt_sum_abt;
```

```
CREATE VIEW gehalt_sum_abt AS

SELECT Abteilungs_ID, SUM(Gehalt) AS Gesamtgehalt

FROM ut_Mitarbeiter

GROUP BY Abteilungs_ID;


SELECT * FROM gehalt_sum_abt;


--Verknüpfte Tabellen in der View

DROP VIEW Mitarbeiter_Information;

CREATE VIEW Mitarbeiter_Information AS

SELECT M.Vorname, M.Nachname, A.Bezeichnung

FROM ut_Mitarbeiter M

JOIN Abteilung A ON M.Abteilungs_ID = A.Abteilungs_ID;


SELECT * FROM Mitarbeiter_Information;


--View mit Bedingung

DROP VIEW hochbezahlte_Mitarbeiter;

CREATE VIEW Hochbezahlte_Mitarbeiter AS

SELECT Vorname, Nachname, Gehalt

FROM ut_Mitarbeiter

WHERE Gehalt > 6000;
```

SELECT * FROM hochbezahlte_Mitarbeiter;

--Ein DBA-Befehl zum erstellen eines Benutzers inklusive Profils mit den notwendigen Berechtigungen

CREATE USER user_1 IDENTIFIED BY password;

--UPDATE-Befehele

-- Update des Lagerbestands der Artikel

UPDATE Lagerbestand SET bestand = bestand - 3 WHERE Lagerbestand_ID = 1;

UPDATE Lagerbestand SET bestand = bestand - 1 WHERE Lagerbestand_ID = 2;

SELECT privilege

FROM user_tab_privs

WHERE table_name = 'DBMS_CRYPTO';

SELECT * FROM Bestellungen;

--Fehlende Berechtigung für DBMS_Crypto

SELECT *

FROM USER_TAB_PRIVS

WHERE TABLE_NAME = 'DBMS_CRYPTO';

Ausgabe: Keine Zeilen ausgewählt

```
SET SERVEROUTPUT ON;

--Whitelist-Validierung für das Material

DROP TABLE erlaubtes_Material;

--Eine Tabelle mit dem erlaubten Material

CREATE TABLE erlaubtes_Material (

   Material VARCHAR2(50) PRIMARY KEY

);



INSERT INTO erlaubtes_Material (Material) VALUES ('Textil');

INSERT INTO erlaubtes_Material (Material) VALUES ('Polyester');

INSERT INTO erlaubtes_Material (Material) VALUES ('Baumwolle');



--Material das hinzugefügt werden soll kommt in den Parameter

--wenn es auf der Whitelist steht, kann es  hinzugefügt werden, sonst nicht

CREATE OR REPLACE PROCEDURE material_validierung (sw_material IN VAR-
CHAR2) IS

   sw_count NUMBER;

BEGIN

   SELECT COUNT(*)

   INTO sw_count
```

```
    FROM erlaubtes_material
    WHERE Material = sw_material;

-- Wenn die Farbe nicht in der Whitelist ist, wird eine Fehlermeldung ausgelöst
    IF sw_count = 0 THEN
       RAISE_APPLICATION_ERROR(-20001, 'Ungültiges Material.');
    END IF;

      IF sw_material IS NULL THEN
            RAISE_APPLICATION_ERROR(-20002, 'Ein Fehler ist aufgetreten: ' ||
SQLERRM);
    END IF;
END;

DROP TRIGGER material_check_white;

--Trigger der vor einer Eingabe in die Tabelle die Blacklist überprüft
CREATE OR REPLACE TRIGGER material_check_white
BEFORE INSERT ON Streetwear
FOR EACH ROW
BEGIN
    material_validierung(:NEW.material);
END;
```

```
--Testdaten

INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung,
Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Jacke', 'Oberteile', 'Eine schicke Jacke.', 45.00, 35,
'Kunstleder', 'XL', 'Grau', 'Puma');



--------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------



--Blacklist-Validierung für das Material

DROP TABLE nicht_erlaubtes_Material;

--Eine Tabelle mit dem verbotenen Material

CREATE TABLE nicht_erlaubtes_Material (

   Material VARCHAR2(50) PRIMARY KEY

);



INSERT INTO nicht_erlaubtes_Material (Material) VALUES ('Kunstleder');

INSERT INTO nicht_erlaubtes_Material (Material) VALUES ('Daunen');

INSERT INTO nicht_erlaubtes_Material (Material) VALUES ('Leder');



--Material das hinzugefügt werden soll kommt in den Parameter

--wenn es auf der Blacklist steht, kann es nicht hinzugefügt werden

CREATE OR REPLACE PROCEDURE material_validierung (sw_material IN VAR-
CHAR2) AS
```

```
    sw_count NUMBER;
BEGIN
-- Überprüfen, ob das Material in der Liste der nicht erlaubten Materialien ist
    SELECT COUNT(*)
    INTO sw_count
    FROM nicht_erlaubtes_Material
    WHERE Material = sw_material;

-- Wenn die Farbe in der Blacklist ist, wird eine Fehlermeldung ausgelöst
    IF sw_count > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Ungültiges Material.');
    END IF;

    IF sw_material IS NULL THEN
            RAISE_APPLICATION_ERROR(-20002, 'Ein Fehler ist aufgetreten: ' ||
SQLERRM);
    END IF;
END;

DROP TRIGGER material_check_black;

--Trigger der vor einer Eingabe in die Tabelle die Blacklist überprüft
DROP TRIGGER material_check_black;
CREATE OR REPLACE TRIGGER material_check_black
```

```
BEFORE INSERT ON Streetwear

FOR EACH ROW

BEGIN

   material_validierung(:NEW.material);



END;



--Testdaten

INSERT INTO Streetwear (sw_Artikel_ID, Bezeichnung, Kategorie, Beschreibung,
Preis, Bestand, Material, Groesse, Farbe, Marke)

VALUES(sw_sq.NEXTVAL, 'Jacke', 'Oberteile', 'Eine schicke Jacke.', 45.00, 35,
"Kunstleder", 'XL', 'Grau', 'Puma');



--------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------

--Fehlermeldungen mit SQLERRM und SQLCODE

SET SERVEROUTPUT ON;

BEGIN

DBMS_OUTPUT.PUT_LINE('Fehlercode: ' || SQLCODE);

DBMS_OUTPUT.PUT_LINE('Fehlermeldung: ' || SQLERRM);

END;



--------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------
```

Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer oder der Verfasserin/des Verfassers selbst entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Anmerkung: In einigen Studiengängen findet sich die Erklärung unmittelbar hinter dem Deckblatt der Arbeit.

Köln, 24.04.2024
Ort, Datum

Unterschrift