Hochschule Wismar
University of Applied Sciences, Technology, Business and Design
Fakultät für Ingenieurwissenschaften

Master-Thesis

Entwicklung einer datensatzbasierten Zugriffsschicht innerhalb relationaler Datenbanken auf Basis hierarchischer Organisationsstrukturen

Development of a Record-Based Access Layer within Relational Databases Based on Hierarchical Organization Structures

Eingereicht am: 02. März 2020

von: Steffen Hoyer

1. Gutachter: Prof. Dr.-Ing. Antje Raab-Düsterhöft

2. Gutachter: Prof. Dr.-Ing. habil. Andreas Ahrens

Aufgabenstellung

Thema: Entwicklung einer datensatzbasierten Zugriffsschicht innerhalb relationaler Datenbanken auf Basis hierarchischer Organisationsstrukturen

Development of a Record-Based Access Layer within Relational Databases Based on Hierarchical Organization Structures

Bei der Entwicklung von neuer Software im Unternehmensumfeld wird häufig die Frage nach einem Berechtigungskonzept gestellt. Vor allem bei börsennotierten Unternehmen, die einer Vielzahl von rechtlichen Regularien unterliegen, ist die Klärung dieser Fragestellung von besonderer Bedeutung. Solche Berechtigungskonzepte sollen möglichst flexibel und vor allem robust konstruiert werden, um Sicherheitslücken vorzubeugen und Inkonsistenzen der vergebenen Berechtigungen zu verhindern.

Agile Arbeitsmethoden und der Wunsch kontinuierlicher Verbesserung sorgt in Unternehmen dafür, dass sich diese immer wieder neu ausrichten und umorganisieren. Neue Bereiche und Teams werden geschaffen um die Anforderungen des freien Marktes zu bewältigen, andere werden zusammengelegt um Synergieeffekte zu erzeugen und wieder andere verschwinden ganz. Auch die natürliche Fluktuation von Mitarbeitern, der Wechsel von Führungskräften sowie das Mitwirken einzelner Mitarbeiter an Projekten soll durch ein modernes Berechtigungskonzept abgebildet werden können.

Im Rahmen dieser Arbeit soll ein relationales Datenbankmodell entwickelt werden, welches zur Speicherung, Pflege und Validierung von Berechtigungen dient. Diese Berechtigungen sollen sich in ihrer Struktur an hierarchischen Organisationsstrukturen orientieren und diese abbilden können. Über eine Verbindungsschicht sollen fachliche Datensätze so mit den gespeicherten Berechtigungen verknüpft werden können, dass eine Berechtigungsstruktur entsteht, die auf der Ebene jedes einzelnen Datensatzes greift und die unterschiedlichen Hierarchiestufen einer Organisationsstruktur dabei berücksichtigt. Zusätzlich soll die Validierung der Rechte vollständig durch die Datenbank sichergestellt werden, um Angriffsvektoren im Frontend einer Anwendung zu minimieren.

Daraus ergeben sich die folgenden Anforderungen an das Datenbankmodell:

- Abbildung einer hierarchischen Organisationsstruktur inkl. Berücksichtigung von strukturellen Veränderungen
- Verknüpfung von fachlichen Datensätzen mit Organisationseinheiten inklusive Entwicklung einer Logik zur Sicherstellung von Zugriffsberechtigungen

Kurzreferat

Im Rahmen dieser Arbeit soll ein relationales Datenbankmodell entwickelt werden, welches sowohl hierarchische Organisationsstrukturen als auch fachliche Datensätze und deren Verknüpfung untereinander beinhaltet. Durch die Verknüpfung fachlicher Datensätze zu den urhebenden Organisationseinheiten und die Verbindung von Mitarbeitern zu diesen Organisationseinheiten soll eine Berechtigungsstruktur entstehen. Die vollständige Einhaltung dieser Berechtigungen soll, entlang der Hierarchie der Organisationseinheiten, durch eine eigene Zugriffsschicht innerhalb der Datenbank sichergestellt werden. Außerdem soll das Modell Umstrukturierungen der Organisationsstruktur möglich machen, ohne jeden Datensatz einzeln zu manipulieren.

Die Umsetzung soll unter Verwendung der Datenbanksysteme Oracle Database, Microsoft SQL-Server und MariaDB erfolgen. Die Anbindung eines externen Authentifizierungsproviders wie beispielsweise LDAP ist nicht Bestandteil dieser Arbeit.

Abstract

The scope of this thesis is to develop a relational database model which includes hierarchical organizational structures, technical data sets and their linking to each other. By linking the technical data records to the owning organizational unit and the connection of employees to their organizational units, a hierarchical authorization structure develops. The fully observance of these authorization structures, along the hierarchical organizational structures, should be completely ensured by a separate access layer within the database. In addition, the model should have the possibility to redefine the organizational structure without manipulating each technical record individually.

The implementation should be done by using the database systems Oracle Database, Microsoft SQL-Server and MariaDB. The connection of an external authentication provider such as LDAP is not part of this thesis.

Inhalt

Aufbau dieser Arbeit

Innerhalb dieser Arbeit soll ein generisches Modell zur Erstellung einer Zugriffsschicht unter Verwendung von hierarchischen Strukturen entwickelt werden. Dazu werden die Datenbanksysteme MariaDB, die Oracle Database sowie der SQL-Server eingesetzt. In diesem Kapitel soll der Aufbau dieser Arbeit kurz wiedergegeben werden.

Zu Beginn wird in der *Einleitung* die Problemstellung anhand eines praktischen Beispiels verdeutlicht, um ein Verständnis für die nachfolgenden Kapitel zu schaffen.

Im zweiten Kapitel *Theoretischer Hintergrund* werden einzelne für diese Arbeit relevante Begrifflichkeiten definiert und die Grundlagen für die darauf aufbauenden Kapitel geschaffen. Auch werden Hierarchien aus praktischer sowie mathematischer Sicht beleuchtet. Eine Vorstellung der ausgewählten Datenbanken mit ihrem geschichtlichen Hintergrund, ein Überblick über den Stand der Technik sowie eine Aufstellung relevanter Forschungen zum Thema Zugriffs-Modelle lassen das Kapitel ausklingen.

Der dritte Abschnitt *Anforderungsanalyse* dient der Aufstellung von Anforderungen an das hier zu entwickelnde Modell hinsichtlich der zu erstellenden Zugriffsschicht und mögliche Veränderungsprozesse innerhalb einer Organisation.

Anschließend findet die *Umsetzung* statt, also die Konzeption und der Aufbau des hier konzipierten Modells unter Verwendung von SQL-Befehlen. Es wird ein Berechtigungskonzept erstellt, Datenbanken sowie Schemata definiert, neue Benutzer werden angelegt und anschließend wird die Zugriffsschicht modelliert und in diesen Datenbanken implementiert. Die fertigen SQL-Statements für alle drei Systeme finden sich im Anhang dieser Arbeit. Es wird jedoch davon abgeraten, diese SQL-Statements ohne vorheriges Lesen dieser Arbeit zu nutzen.

In Kapitel 5 findet die *Überprüfung* des zuvor erstellten Modells statt. Hierzu werden Testfälle definiert und durchgeführt, Performancemessungen bei veränderter Datenmenge durchgeführt sowie spezielle Besonderheiten der einzelnen Systeme erläutert. Hier geht es inhaltlich tief in die Materie von Datenbanksystemen.

Das nachfolgende Kapitel 6 beinhaltet die *Bewertung* der Ergebnisse aus dem vorherigen Abschnitt. Hier wird in komprimierter Form auf die ermittelten Ergebnisse sowie die Besonderheiten der einzelnen Systeme eingegangen.

Im letzten Kapitel *Zusammenfassung und Ausblick* wird neben einer Zusammenfassung auch auf mögliche Fehlerquellen bei der Nutzung dieses Modells hingewiesen. Außerdem finden sich hier eine Vielzahl von Ideen und Möglichkeiten zur Weiterentwicklung.

1 Einleitung

Der Begriff Aufbauorganisation beschreibt das klassisch hierarchische Konstrukt, nach dem ein Unternehmen oder eine Behörde aufgebaut ist. Die operative Ebene, also die Mitarbeiter die das Tagesgeschäft bedienen, befinden sich hierbei auf der untersten Ebene. Ihnen direkt übergeordnet ist eine Führungskraft, die je nach Größe der Organisation einer weiteren Führungskraft unterstellt ist. Jede Position innerhalb dieser Organisation, hier im Falle einer leitenden Position, trägt Verantwortung für die darunter angeordneten Einheiten. Häufig gehen solche Verantwortlichkeiten mit höheren Berechtigungen einher. Eine Führungskraft hat Einblick in die operativen Vorgänge seiner Mitarbeiter und Teams, wohingegen diese im Falle von getrennten Funktionseinheiten keinen Einblick in die entsprechenden Daten anderer Teams haben.

In der modernen Softwareentwicklung, speziell bei der Konzeption und Entwicklung neuer Software, stellt die Frage nach einem stabilen und vor allem flexiblen Berechtigungskonzept eine zentrale Bedeutung dar.

Das *Bundesamt für Sicherheit in der Informationstechnik* (BSI) hat aufgrund der Wichtigkeit dieser Fragestellung hatte schon dem eigens bereitgestellten IT-Grundschutz-Katalog den Baustein *B 1.18 Identitäts- und Berechtigungsmanagement* [1] beigefügt. Heute findet sich dieses Kapitel im IT-Grundschutz-Kompendium des BSI unter dem Eintrag ORP.4. Doch nicht nur das BSI hat sich diesem Thema angenommen, welches mit seinem IT-Grundschutz-Katalog eher beratenden Charakter hat. Auch die *Bundesanstalt für Finanzdienstleistungsaufsicht* (BaFin), welche gemäß *Kreditwesengesetz* (KWG) § 6 eine Weisungs- und Kontrollfunktion gegenüber Finanzdienstleistern inne hat, hat sich mit dieser Thematik beschäftigt. So wurde zuletzt im Oktober 2017 die aktualisierte Fassung der Verwaltungsanweisung *Mindestanforderungen an das Risikomanagement* veröffentlicht. Darin schreibt die BaFin einen klaren Umgang mit Berechtigungen vor:

> *„Berechtigungen und Kompetenzen sind nach dem Sparsamkeitsgrundsatz (Need-to-know-Prinzip) zu vergeben und bei Bedarf zeitnah anzupassen.“* (BaFin [2], AT 3.2.1)

> *„[...] insbesondere sind Prozesse für eine angemessene IT-Berechtigungsvergabe einzurichten, die sicherstellen, dass jeder Mitarbeiter nur über die Rechte verfügt, die er für seine Tätigkeit benötigt; die Zusammenfassung von Berechtigungen in einem Rollenmodell ist möglich.“* (BaFin [2], AT 7.2)

Die Notwendigkeit funktionierender Berechtigungsstrukturen hat also auch der Gesetzgeber erkannt. Bei der Konzeption solcher Strukturen stößt man jedoch schnell an eine gewisse Komplexität, die es zu bewältigen gilt.

Dabei soll das folgende fiktive Beispiel das Problem verdeutlichen:

Ein Versicherungskonzern beschäftigt bundesweit eine Vielzahl an Maklern, die den Verkauf von Versicherungsprodukten sowie die Kundenbetreuung vor Ort übernehmen. Jeder Makler hat also seinen ganz persönlichen Kundenstamm. Alle diese Kundendaten werden konsolidiert in einer einzigen Datenbank des Versicherungskonzerns vorgehalten. Je Region gibt es sogenannte Betreuer, die wiederum Ansprechpartner für eine Vielzahl von Maklern sind. Die Provisionsabrechnung hingegen wird von einem zentralen Team im Konzern durchgeführt, das gleichermaßen für alle Makler zuständig ist. Dieses Team muss Zugriff auf alle Kunden- und Vertragsdaten haben. Wie lässt sich dies optimal abbilden?

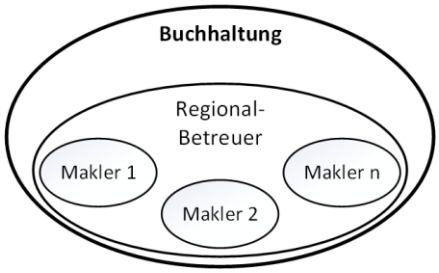

Bild 1: Venn-Diagramm zur Darstellung von Teilmengen

Die Schwierigkeit an dieser Struktur liegt in der Tatsache, dass alle Informationen innerhalb einer einzigen Datenbank vorgehalten werden. Je nach Position hat ein Mitarbeiter mehr oder weniger Datensätze, die für ihn sichtbar sind. Die einzig sinnvolle Lösung für dieses Aufgabenstellung scheint im Aufbau einer hierarchischen Struktur zu liegen. Diese muss für jeden Makler eine Berechtigungsgruppe bereitstellen, der die einzelnen Kundendatensätze zugeordnet sind. Diese Gruppen müssen nun so miteinander verknüpft werden, dass sich der Regionalbetreuer in den gleichen Gruppen befindet, wie die jeweiligen Makler, die er betreut.

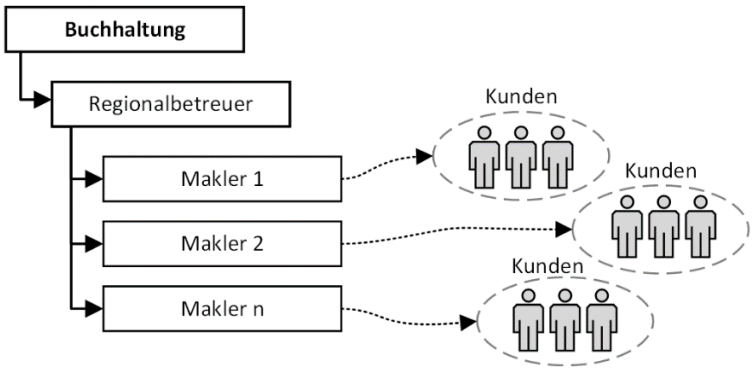

Bild 2: Überführtes Venn-Diagramm in eine hierarchische Darstellung

2 Theoretischer Hintergrund

2.1 Abgrenzung

Der Themenschwerpunkt dieser Arbeit liegt in der Entwicklung eines Datenbankmodells zur Abbildung hierarchischer Organisationsstrukturen auf der Ebene jedes einzelnen Datensatzes. Die Logik und Implementierung sind vollständig auf die Datenbank selbst beschränkt. Auch wenn es sich thematisch anbieten würde, wird auf die Anbindung eines Verzeichnisdienstes wie LDAP verzichtet. Die Verschlüsselung von Datensätzen, die EU-DSGVO konforme Löschung von Daten oder gar eine Historisierung bzw. Versionierung steht ebenfalls nicht im Fokus dieser Arbeit. Es wird keine inhaltliche Unterscheidung zwischen Angestellten in leitender Position und Angestellten ohne leitende Position vorgenommen. Alle vorangehenden und noch folgenden Unterscheidungen dienen nur dem Verständnis. Es wird davon ausgegangen, dass ein fachliches Datenbankmodell keinen strukturellen Veränderungen unterliegt, was zugegebenermaßen selten der Realität entspricht, jedoch im Rahmen dieser Arbeit die Komplexität der Aufgabenstellung reduzieren soll.

Eine Analyse der Abfrage-Performance findet nur dann statt, wenn es inhaltlich von besonderer Bedeutung für das Thema ist. Bei der Erarbeitung einer Lösung soll möglichst auf proprietäre Lösungen einzelner Datenbankhersteller verzichtet werden, um die hier erarbeiteten Ergebnisse ggf. auf andere Datenbanksysteme übertragen zu können. Eine Betrachtung von NoSQL-Datenbanken wie beispielsweise MongoDB, CouchDB oder Apache Cassandra findet nicht statt.

2.2 Begriffsdefinitionen

In diesem Kapitel sollen die verwendeten Begriffe, auch wenn diese meist in Verbindung mit einem konkreten Beispiel verwendet werden, definiert werden. Die in dieser Arbeit verwendeten Beispiele dienen nur der besseren Anschaulichkeit und sollen das Verständnis erleichtern. Die Anwendungsgebiete sind nicht auf die hier verwendeten Beispiele beschränkt.

2.2.1 Organisation

Unter einer Organisation ist die Gesamtstruktur aller Organisationseinheiten, deren Assoziationen untereinander und die Menge aller Mitarbeiter inklusive deren Zuordnungen zu den Organisationseinheiten zu verstehen.

2.2.2 Organisationseinheit

Eine Organisationseinheit (OE) stellt üblicherweise eine Position, die ein Mitarbeiter oder eine Gruppe von Mitarbeitern innerhalb seines Unternehmens innehaben kann, dar. Es kann sich dabei jedoch auch um eine verwaltungstechnische Einheit, ein funktionales Team oder eine Abteilung handeln. Selbst Tochterunternehmen in einem Konzernverbund können im Rahmen dieser Arbeit als Organisationseinheiten betrachtet werden. Am besten lässt sich das Konstrukt einer Organisationseinheiten als abstrakter Container beschreiben, dem Mitarbeiter aufgrund ihrer Tätigkeiten, Befugnisse, Aufgabengebiete oder Rolle in einer Organisation zugeordnet sind. Dieser abstrakte Container ist in eine Hierarchie, bestehend aus anderen Organisationeinheiten, eingebunden.

Eine Organisationseinheit (OE) ist durch die folgend dargestellten Eigenschaften gekennzeichnet:

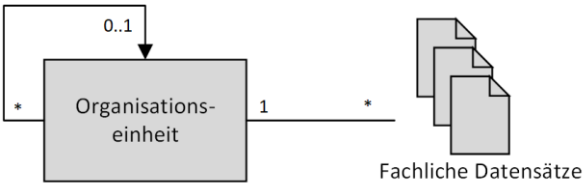

Bild 3: Logischer Aufbau einer Organisationseinheit (OE)

Einer Organisationseinheit ist hierbei eine beliebige Anzahl von fachlichen Datensätzen zugeordnet. Dabei stellt die Organisationseinheit den Ursprung oder auch Eigentümer eines Datensatzes dar. Hier gilt, dass die Anzahl der zugeordneten Datensätze 0 betragen kann, wenn die OE als ein abstrakter Container genutzt wird. In Bild 2 wird beispielsweise die Ebene „Regionalbetreuer“ als abstrakter Container genutzt. Dieser Ebene sind zwar Mitarbeiter zugeordnet, jedoch enthalten sie direkt keine fachlichen Datensätze.

Da eine OE im Rahmen dieser Arbeit mit anderen OEs in einer hierarchischen Struktur verwendet wird, ergibt sich, dass fast jede OE einer übergeordneten OE untergeordnet ist. Diese übergeordnete OE wird als *Parent* bezeichnet. Wenn es sich um das oberste Element einer Hierarchie handelt, so existiert keine übergeordnete OE. Am Beispiel von Bild 2 ist jeder „Makler“-OE die „Regionalbetreuer“-OE übergeordnet. Dieser wiederum ist die OE „Buchhaltung“ überstellt, welche als oberstes Element der Hierarchie keinen „Parent“ besitzt. Im weiteren Verlauf dieser Arbeit kommt dieser *Parent-Child*-Relation der Über- sowie Unter-Ordnung von OEs eine wesentliche Bedeutung zu, wobei die Benamung der Organisationseinheiten nur dem besseren Verständnis des Inhalts dient. Nachfolgend wird neben der Abkürzung OE auch OU verwendet, welche für die englische Bezeichnung „organizational unit“ steht und gleichbedeutend mit einer OE ist.

2.2.3 Mitarbeiter

Als Mitarbeiter (M), später auch Employee (E) genannt, wird jede Art von Benutzer oder Anwender bezeichnet, der mit den Daten innerhalb dieses Modells und somit der zugrundeliegenden Datenbank interagiert. Die Bezeichnung „Benutzer“ ist hier nicht gleichzusetzen mit der datenbankinternen Benutzerverwaltung. Es handelt sich im Rahmen dieser Arbeit ausschließlich um eine Kennung, welche sowohl von einem Serverdienst, als auch einer realen Person genutzt werden kann. Mitarbeiter werden als eigenständige Datensätze innerhalb der Datenbank betrachtet, die beim Aufbau dieses Modells in Kapitel 4.4 weiter beschrieben werden.

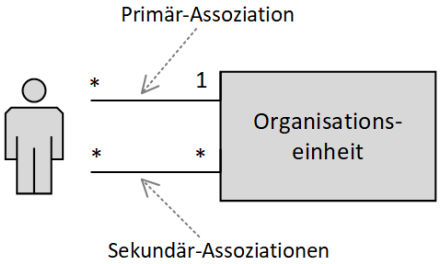

Bild 4: Zuordnung eines Mitarbeiters zu Organisationseinheiten

Ein Mitarbeiter verfügt innerhalb einer Organisation über zwei Arten von Assoziationen: Die Primär-Assoziation beschreibt die direkte Zuordnung eines Mitarbeiters zu seiner eigentlichen Organisationseinheit. Diese Zuordnung wird in der Praxis häufig vorgenommen, um eine interne Kosten- und Leistungsverrechnung zu ermöglichen. Sie beschreibt außerdem, in welchem Team oder in welcher Abteilung ein Mitarbeiter angestellt ist. Es handelt sich somit um eine starke Assoziation, die nur im Falle eines Wechsels aufgrund von Fluktuation oder interner Versetzung aufgelöst wird. Im Vergleich dazu sind die Sekundär-Assoziationen weniger stark verankert und zeigen die weitere Zuordnung zu funktionalen Teams, Arbeitsgruppen und Projekten auf. Diese Zuordnungen sind meist zeitlich begrenzt oder ändern sich im Laufe der Zeit.

Innerhalb des Modells wird nicht zwischen Mitarbeitern in leitender Funktion (Führungskräfte) und Mitarbeitern ohne leitende Funktion unterschieden. Jeder Mitarbeiter-Datensatz ist gleichwertig zu allen anderen Datensätzen. Die Position oder Stellung eines Mitarbeiters innerhalb einer Organisation wird durch die Zuordnung zu seiner primären Organisationseinheit und deren Position innerhalb der Hierarchie bestimmt. Im Gegensatz zu Organisationseinheiten haben Mitarbeiter keine Datensätze direkt zugeordnet. Dies liegt daran, dass Mitarbeiter keine Eigentümer der erhobenen Daten sind. Diese Daten gehören der Organisationseinheit, in dessen Name ein Mitarbeiter diese erfasst hat. Sollte der Mitarbeiter aus dem Unternehmen ausscheiden oder in eine andere Organisationseinheit wechseln, verbleiben die erhobenen Datensätze bei der Organisationseinheit.

2.2.4 Fachlicher Datensatz

Unter fachlichen Datensätzen (FD) werden alle Datensätze verstanden, die sich innerhalb eines bestehenden fachlichen Datenbankmodells (FDBM) befinden. Ein solches Modell besteht im Kern aus Tabellen und deren Beziehungen untereinander, beinhaltet jedoch nur die Datensätze, die für die Erfüllung der fachlichen Anforderungen benötigt werden. Folgend ein Beispiel eines solchen fachlichen Modells:

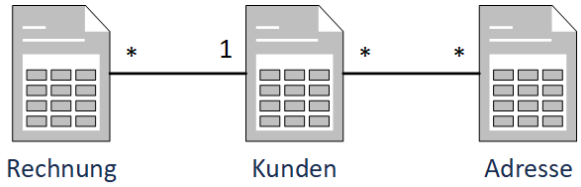

Bild 5: Beispiel eines fachlichen Datenmodells (FDBM)

Hier sehen wir ein Modell, welches Kunden, deren Adresse(-n) und die ihnen zugeordneten Rechnungen beinhaltet. Jeder hier enthaltene Datensatz, egal ob Rechnung, Adresse oder Kunde wird als fachlicher Datensatz (FD) behandelt. Auch die Zwischentabelle, die für die Abbildung der Kunde-Adresse-Relation notwendig ist, enthält fachliche Datensätze.

2.2.5 Berechtigung

Unter einer Berechtigung, oder auch Legitimation genannt, wird üblicherweise das Recht zur Nutzung einer bestimmten Ressource verstanden. Als Ressource kann der Zugriff auf einen Datensatz, Rechenzeit der CPU, eine bestimmte Menge Hauptspeicher oder Übertragungskapazität einer Netzwerkverbindung verstanden werden.

Häufig gehen moderne Softwaresysteme über das klassische „Zugriff erlaubt“ und „Zugriff nicht erlaubt“ hinaus. Betriebssysteme wie Unix oder Linux unterscheiden bereits seit vielen Jahren bei Dateisystemberechtigungen zwischen „Lesen“, „Schreiben“ und „Ausführen“. Diese Berechtigungen können beliebig kombiniert für den Besitzer einer Ressource, eine Benutzergruppe und „Andere“ gesetzt werden. [27]

Wieder andere Systeme implementieren ein Rechte- und Rollenmodell. Dabei sind Rollen mit einer bestimmten Menge an Rechten ausgestattet. Diese Rechte erlauben dem Empfänger der jeweiligen Rolle die Nutzung zugehöriger Ressourcen. Über eine hierarchische Vererbungsstruktur lassen sich Rollen um die enthaltenen Rechte anderer Rollen erweitern, wobei die Zuordnung eines Rechtes dem binären Zustand „Erlauben“ entspricht. Das Fehlen eines Rechtes heißt in diesem Fall „Verweigern“.

2.2.6 Datenbank, Instanz, Schema

Bei einer *Datenbank* handelt es sich ausschließlich um einen logisch zusammenhängenden Bestand an Datensätzen, welcher in Form einer oder mehrerer Dateien auf einem Speichermedium abgelegt sind.

Ein *Datenbankmanagementsystem (DBMS)* hingegen ist eine Software, welche den Zugriff auf diese Datenbank, also auf die abgelegten Dateien, ermöglicht. Dazu wird ein Prozess gestartet, welcher den Zugriff auf die Datenbank(-dateien) für Benutzer und Anwendungssysteme, meist über einen speziell reservierten Port des Servers, bereitstellt. In der Regel wird ein bestimmter Datenbestand vom DBMS als Puffer im Hauptspeicher des Servers vorgehalten, um Zugriffe auf die Datenbank zu beschleunigen. Neben der Steuerung von lesenden und schreibenden Zugriffen übernimmt ein DBMS auch die Aufgabe der Transaktionssteuerung, auch ACID-Prinzip genannt. Außerdem werden SQL-Abfragen an die Datenbank vor der Ausführung vom DBMS optimiert, um die Verarbeitung zu beschleunigen. Das Ausführen von *Triggern* und *Stored Procedures* wird ebenfalls vom DBMS übernommen.

Wird nun von einem DBMS der Zugriff auf eine Datenbank durch Starten der jeweiligen Prozesse ermöglicht, so spricht man von einer *Instanz*. Dabei handelt es sich um die Summe aller laufenden Prozesse des DBMS, der verwalteten Datenbank selbst sowie dem allokierten Hauptspeicher.

Ein *Schema* hingegen ist ein logisches Konstrukt zur Trennung von Objekten innerhalb einer Datenbank. Somit kann ein Schema als eine Art Namensraum betrachtet werden. Die Nutzung mehrerer Schemata erlaubt das Anlegen gleichnamiger Objekte innerhalb einer einzigen Datenbank, solange diese Objekte in unterschiedlichen Schemata abgelegt sind. Da sich Schemata in der gleichen Datenbank befinden, also auch in der gleichen Instanz im Hauptspeicher liegen, lassen sich innerhalb einer Instanz mehrere fachliche Datenmodelle parallel und unabhängig voneinander betreiben. Ein solches Konstrukt hat den Vorteil, dass gezielt Berechtigungen auf Objekte außerhalb des eigenen fachlichen Schemas vergeben werden können, ohne die übrigen Objekte freigeben zu müssen. Zusätzlich lassen sich Konsistenzbedingungen, sogenannte Constraints, über Schemata hinweg definieren. Außerdem lässt sich beispielsweise nicht nur festlegen welcher Datenbankbenutzer auf eine View außerhalb des eigenen Schemas zugreifen darf, sondern auch mit welchen Berechtigungen diese View ausgeführt werden soll.

Sowohl Oracle Database als auch der Microsoft SQL-Server unterstützen das Anlegen von Schemata innerhalb einer einzigen Datenbank. Das Datenbanksystem MariaDB hingegen verwendet die Begriffe Datenbank sowie Schema synonym zueinander. Um eine einheitliche Terminologie zu verwenden, wird im weiteren Verlauf dieser Arbeit der Begriff Schema verwendet.

2.3 Hierarchien

Unter einer Hierarchie versteht man ein Prinzip der Ordnung beziehungsweise logischen Gliederung von Elementen, welche in einer Beziehung der Über- oder Unterordnung zueinanderstehen. Eine solche Ordnung wird durch Regeln bestimmt, welche einer logischen Struktur folgen. Üblicherweise wird bei dem Aufbau einer Hierarchie ein Attribut oder eine Eigenschaft eines Elements als Kriterium für die Positionierung innerhalb einer Hierarchie verwendet.

Betrachtet man „Lebewesen“ als übergeordnetes Element, so können Elemente wie „Mensch“, „Tier“ oder „Pflanze“ diesem Element untergeordnet werden, da sie die entsprechenden Eigenschaften des übergeordneten Elements bereits mit sich bringen oder übertragen bekommen. Die zugrundeliegende Ordnung einer Hierarchie bestimmt maßgeblich deren Aufbau. Der Aufbau einer Hierarchie, welche Menschen nach ihrem Geburtsjahr geordnet enthält, ist anders aufgebaut als eine Hierarchie welche die gleichen Menschen nach ihrem Geschlecht oder ihrer Augenfarbe ordnet.

Ein Sachbuch selbst enthält sogar zwei unabhängig voneinander aufgebaute Hierarchien. Zu Beginn eines solchen Buches befindet sich ein Inhaltsverzeichnis mit den enthaltenen Kapiteln und der zugehörigen Seitenangabe. Dieses Inhaltsverzeichnis ist häufig so aufgebaut, dass bestimmte Kapitel auf dem Inhalt vorheriger oder übergeordneter Kapitel aufbauen. Auch ist eine Ordnung nach bestimmten Themengebieten möglich, welche einzelne Themen in Unterkapiteln spezialisiert behandelt. Gegen Ende eines solchen Buches befindet sich ein Stichwortverzeichnis, in dem wesentliche Begrifflichkeiten mit der entsprechenden Seitenangabe aufgeführt werden. Diese zweite Hierarchie folgt keiner thematischen Ordnung wie das eben beschriebene Inhaltsverzeichnis, sondern ordnet die enthaltenen Elemente alphabetisch an. Beide Hierarchien haben in einem Sachbuch ihre Berechtigung und finden, abhängig von den Anforderungen des Lesers, Verwendung.

Im Rahmen dieser Arbeit sollen jedoch keine Tiere, Pflanzen oder Sachbücher analysiert werden, auch wenn diese als erste Beispiele zur Einführung in diese Thematik verwendet wurden. Es werden vorwiegend Organisationseinheiten betrachtet. In der Praxis haben Organisationseinheiten die Aufgabe, Gruppen von Mitarbeitern abhängig von ihren Tätigkeiten zu ordnen. Dabei besitzen Mitarbeiter einer übergeordneten Organisationseinheit häufig Weisungsbefugnisse gegenüber den Mitarbeitern direkt oder transitiv (= indirekt) untergeordneter Organisationseinheiten.

Somit kann eine solche Hierarchie auch als eine Rangordnung der zugeordneten Mitarbeiter betrachtet werden, wobei das hier genannte Beispiel nur zur Verdeutlichung des Sachverhaltes dient. Der Inhalt dieser Arbeit befasst sich nicht mit der Abbildung der Struktur eines bestimmten Unternehmens oder einer Unternehmensgruppe. Sie erlaubt jedoch die Adaption der Ergebnisse auf reale Unternehmensstrukturen.

Die eben beschriebene Weisungsbefugnis-Relation ist hier klar erkennbar, siehe Bild 6.

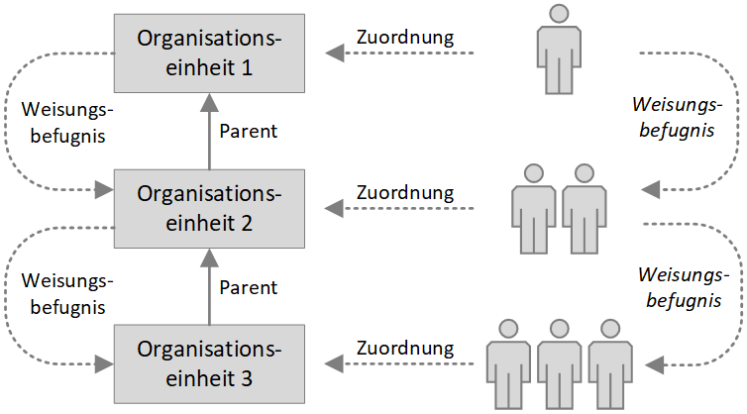

Bild 6: Weisungsbefugnis-Relation

Eine Organisationseinheit hat ein übergeordnetes Element, hier als *Parent* dargestellt. Aus dieser gerichteten *Parent-Relation* ergibt sich für die spätere Umsetzung eine ebenfalls gerichtete gegenläufige *Weisungsbefugnis-Relation*. Durch die Zuordnung eines Mitarbeiters zu einer Organisationseinheit übernimmt dieser die jeweilige gegenläufige *Weisungsbefugnis-Relation* der Organisationseinheit.

Die *Weisungsbefugnis-Relation* ist, im Gegensatz zum dargestellten Bild, jedoch nicht auf die direkt untergeordnete Organisationseinheit und die dort angesiedelten Mitarbeiter beschränkt. Sie umfasst außerdem alle darunter liegenden Organisationseinheiten sowie ihre zugeordneten Mitarbeiter. Dieser Zugriff von einer Organisationseinheit auf die darunter liegenden nicht direkt verknüpften Organisationseinheiten wird als transitiver Zugriff, oder im Falle des dargestellten Bildes, als transitive Weisungsbefugnis verstanden.

Ein wesentliches Kriterium für eine Hierarchie im weiteren Sinne dieser Arbeit ist die sogenannte Zyklenfreiheit der Organisationseinheiten innerhalb der Hierarchie. Entlang der *Weisungsbefugnis-Relation*, die entgegen der *Parent-Relation* verläuft, darf keine Schleife, auch Zyklus genannt, entstehen. Dies würde zu unklaren Weisungsbefugnissen führen, da ein Mitarbeiter seinem Vorgesetzten, ein Mitarbeiter einer übergeordneten OU, nicht gleichzeitig unterstellt und gleichzeitig überstellt sein kann.

Da es sich bei den hier verwendeten Hierarchien ausschließlich um azyklische Digraphen, also gerichtete Graphen ohne Zyklus, aus dem Bereich der Graphentheorie handelt, sollen folgend die mathematischen Grundlagen kurz umrissen werden. Anschließend wird noch das Konstrukt der Zyklenfreiheit beschrieben.

Weiterführende Informationen zu Graphen finden sich unter [28], [29] und [30].

2.3.1 Mathematische Grundlagen

Ein gerichteter Graph G besteht aus einer Knotenmenge V, einer Kantenmenge E sowie den beiden Funktionen $\alpha : E \to V$ und $\omega : E \to V$. Somit ist ein gerichteter Graph ein Quadrupel $G = (V, E, \alpha, \omega)$. Hier sei erwähnt, dass in mancher Literatur auf die Definition der beiden Funktionen α und ω verzichtet wird, was dazu führt, dass ein gerichteter Graph dort als einfaches Tupel aufgeführt ist.

Die Kantenmenge $E \subseteq V \times V$ beschreibt eine Teilmenge aller möglichen Kanten innerhalb des Graphen. Jeder Knoten $v \in V$ stellt ein Objekt innerhalb des Graphen dar. Jede gerichtete Kante wird als Tupel $e = (u, v) \in E$, als Verbindung zwischen den Objekten $u \in V$ und $v \in V$ repräsentiert.

Hierbei erlaubt die Funktion $\alpha(e) \to u \in V$ eine Abbildung auf den Startknoten und $\omega(e) \to v \in V$ auf den Zielknoten der Kante e.

Die Anzahl der Objekte innerhalb eines Graphen G wird nun als $n := |V(G)|$ und die Anzahl der Kanten als $m := |E(G)|$ angegeben. Als weitere Notwendigkeit für diese Arbeit werden reflexive sowie doppelte Kanten ausgeschlossen:

$$\forall e \in E(G) : \alpha(e) \neq \omega(e)$$

$$\nexists e, e' \in E(G) : e \neq e' \wedge \alpha(e) = \alpha(e') \wedge \omega(e) = \omega(e')$$

Im folgenden Beispiel wird auf die Erstellung einer Tabelle, wie sie exemplarisch in [30] angegeben ist, sowie auf die erneute Definition der Funktionen α und ω aus Gründen der Übersichtlichkeit verzichtet. Der in Bild 7 dargestellte Graph ist somit wie folgt definiert:

$$G = (V, E, \alpha, \omega), V = \{v1, v2, v3, v4, v5\}$$

$$E = \{(v1, v2), (v1, v3), (v2, v4), (v2, v5), (v3, v4)\}$$

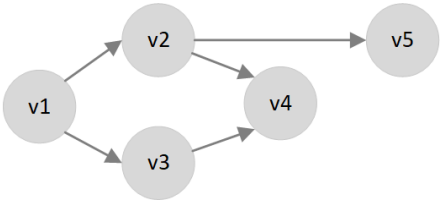

Bild 7: Digraph / Gerichteter Graph

Nachdem nun ein Graph, oder in diesem Fall ein gerichteter Graph, kurz definiert ist, soll das Thema Zyklenfreiheit angesprochen werden.

An dieser Stelle sei darauf hingewiesen, dass sich auch hier unterschiedliche Definitionen von *Pfad* und *Weg* in der Literatur finden lassen. Häufig werden diese Begriffe als Synonyme verwendet, was mit einer falschen Übersetzung der Begriffe begründet wird. Auch enthält ein *Pfad* oder *Weg* in mancher Literatur zusätzlich zu den Knoten noch die jeweiligen Kanten. Die folgende Definition entstammt daher einem entsprechenden Lehrbuch [31] und kann bei weiterer Literaturrecherche abweichend ausfallen.

Bei einem *Weg*, in diesem Beispiel sogar einem *gerichteten Weg*, handelt es sich um eine Abfolge $P = (v_0, \dots, v_l)$ von Knoten $v \in V$, wobei die jeweils benachbarten Knoten in dieser Abfolge über eine Kante miteinander verbunden sein müssen. Die Länge eines solchen *Weges* wird mit l angegeben. Die nachfolgende Definition ist auf die hier relevanten gerichteten Graphen angepasst.

$$(v_i, v_{i+1}) \in E(G) : \alpha(v_i) = \omega(v_{i+1}) \text{ für } i = 0, \dots, l-1$$

Aufgrund der zuvor ausgeschlossenen reflexiven Kanten, hat ein *Weg* somit mindestens die Länge 1, enthält also mindestens 2 Knoten. Jedoch lässt der hier beschriebene *Weg* gemäß Definition Zyklen zu beziehungsweise schließt diese nicht aus. Dieses Problem lässt sich über einen *Pfad* lösen. Ein *Pfad* ist ein *Weg* mit der zusätzlichen Eigenschaft, dass die enthaltenen Knoten paarweise verschieden sein müssen. Es existiert also kein Knoten, der mehr als ein einziges Mal erreicht wird.

$$\forall p \in P : (\forall v, v' \in p : v \neq v')$$

Wird nun der Graph um die Kanten $e_{m+1} = (v5, v1)$ und $e_{m+2} = (v5, v2)$ erweitert, ergibt sich Bild 8. Es ergibt sich ein entsprechender Zyklus, der mit der hier aufgestellten Anforderung der Zyklenfreiheit nicht kompatibel ist.

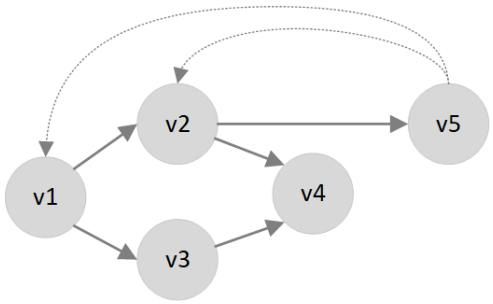

Bild 8: Digraph / Gerichteter Graph mit Zyklus

2.3.2 Monohierarchie

Eine Monohierarchie ist das meist verbreitete Konstrukt eine Hierarchie, bei der jedem Knoten innerhalb der Hierarchie maximal ein einziger Knoten übergeordnet ist. Eine solche Hierarchie wird auch „Starke Hierarchie“ genannt. [3] Jedem Knoten kann selbst eine beliebige Anzahl von Knoten untergeordnet sein. Der oberste Knoten in einer solchen Hierarchie wird Wurzel oder Wurzel-Knoten genannt.

Häufig existieren solche Strukturen im Alltag, ohne sie als solche zu erkennen sind. Ein Organigramm eines Unternehmens, egal welcher Größe, ist meist in Form einer Monohierarchie aufgebaut. So besteht ein internationaler Konzern beispielsweise aus mehreren Tochterunternehmen, welche einer Holding unterstellt sind. Diese Tochterunternehmen untergliedern sich in Bereiche, welche sich mit unterschiedlichen Produktsegmenten oder Kundengruppen beschäftigen. Diese wiederum enthalten einzelne Abteilungen oder Teams für Vertrieb, Support und Reklamation, welche sich ebenfalls in einzelne Teams untergliedern lassen. Bei den klein- und mittelständischen Unternehmen findet sich eine solche Unterteilung ebenfalls, auch wenn die Anzahl der hier vorhandenen Organisationseinheiten nicht so hoch ist, wie bei internationalen Konzernen.

Auch im privaten Umfeld, abseits der freien Marktwirtschaft, finden sich solche Strukturen. Die persönliche Aktenablage wird häufig nach bestimmten Kriterien gegliedert. So finden sich Dokumente für Steuererklärungen selten direkt neben Zertifikaten oder privaten Briefen. Hier erfolgt eine thematische Eingruppierung in spezielle Aktenordner, welche die Dokumente wiederum in einer chronologischen Reihenfolge beinhalten.

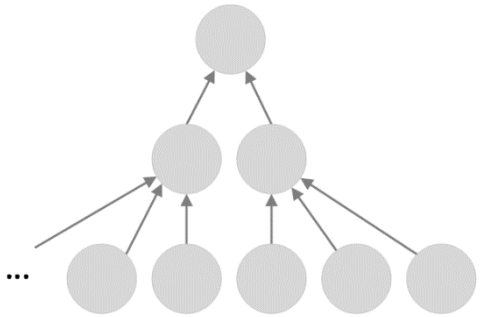

Bild 9: Monohierarchie / Starke Hierarchie

Die maximale Höhe einer solchen Monohierarchie, damit ist der *Pfad* mit der größten Länge gemeint, hat einen wesentlichen Einfluss auf die Performance eines Systems.

Je höher eine Hierarchie und somit je länger der größte *Pfad* innerhalb dieser Hierarchie, desto mehr Kanten und Knoten müssen vom Computer durchlaufen werden, was bei dem Aufbau einer Hierarchie berücksichtigt werden sollte.

2.3.3 Polyhierarchie

Gegenüber der Monohierarchie zeichnet sich die Polyhierarchie dadurch aus, dass jedes Element mehr als einen direkt übergeordneten Knoten zugewiesen haben kann. Diese Form wird auch als „Schwache Hierarchie“ bezeichnet. [3]

Anzutreffen ist diese Form der Hierarchie ebenfalls im Berufsleben. Selbst wenn Unternehmen von ihrer Struktur eine Monohierarchie darstellt, so passiert es durchaus, dass ein Mitarbeiter zeitweise an Projekten mitwirkt und damit zusätzlich einem weiteren Team angehört und folglich auch einer weiteren Führungskraft unterstellt ist. Selbst die Vertretung im Krankheitsfall oder Unterstützung bei Personalmangel resultieren in einer zusätzlichen temporären Zuordnung zu einem anderen Team. Ein Mitarbeiter ist in dieser Zeit somit gleichzeitig mehr als einer Führungskraft unterstellt.

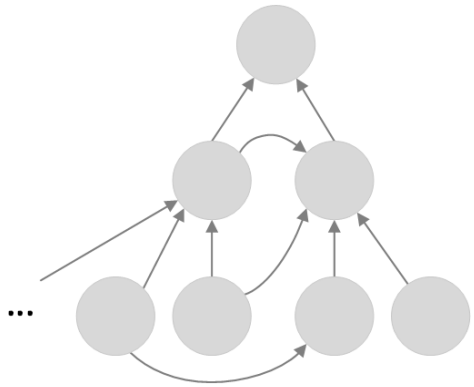

Bild 10: Polyhierarchie / Schwache Hierarchie

Eine Schwierigkeit bei der Umsetzung von Polyhierarchien ergibt sich, wenn konträre Eigenschaften eine Rolle spielen können. Dies ist unter anderem bei dem sogenannten Diamant-Problem im Rahmen der Mehrfachvererbung bei objektorientierten Programmiersprachen der Fall. Erbt eine Klasse C alle Methoden der direkt übergeordneten Klassen A und B, so ist bei identischer Spezifikation einer Methode Z mit unterschiedlicher Implementierung innerhalb von A und B nicht klar, welche der beiden Z-Methoden bei dem Aufruf über ein Objekt der Klasse C zu verwenden ist.

Ist ein Mitarbeiter direkt zwei Führungskräften unterstellt und geben diese genau gegensätzliche Anweisungen, muss durch Einigung der Führungskräfte untereinander eine Lösung erzielt werden. Aus diesem Grund ist es häufig so, dass die direkte Führungskraft mehr Weisungsrechte als der (zeitweise übergeordnete) Projektleiter besitzt. Die Weisungsbefugnis-Relation ist somit mit einer Gewichtung versehen, was bei der Entscheidungsfindung hilfreich ist. Durch die Zuhilfenahme von Computern sind Polyhierarchien jedoch auch dort möglich, wo sie in der realen Welt unmöglich wären. Innerhalb einer Bibliothek kann ein Buch nur in einem einzigen Regal stehen, auch wenn dieses im Inventar zwei unterschiedlichen Genres zugeordnet ist.

2.4 Datenbanksysteme

Im Rahmen dieser Arbeit sollen drei Datenbanksysteme untersucht werden. Die Auswahl dieser Systeme erfolgte vorwiegend anhand der Verbreitung, um eine Adaption der hier entstandenen Ergebnisse zu ermöglichen. Im Enterprise-Umfeld wurden die Oracle Database als auch der Microsoft SQL-Server ausgewählt. Um die hier entstandenen Ergebnisse auch für den privaten und non-profit Sektor nutzbar zu machen, wurde zusätzlich noch die MariaDB, welche als freies Open-Source Datenbanksystem Verwendung findet, in die Auswahl aufgenommen. [4] [5] [6]

2.4.1 Oracle Database

Als eines der beiden kommerziellen Datenbanksysteme wurde die Oracle Database in der Version 18.3 ausgewählt. Dieses Datenbanksystem wurde erstmals im Jahre 1979 auf den Markt gebracht [7] und hat sich seitdem zu einem der führenden Datenbanksysteme im Unternehmensumfeld entwickelt. Je nach betrachteter Studie oder Erhebung bewegt sich dieses Datenbanksystem immer auf den oberen Plätzen. Der Grund für die hohe Verbreitung dieser Datenbank kann einerseits in der langen Markpräsenz des Systems begründet sein, andererseits jedoch auch durch das um die Datenbank entstandene Ökosystem und die Vielzahl von unterstützten Funktionen. So bietet das System neben der reinen Datenbank-Funktionalität unter anderem auch eine Entwicklungsumgebung sowie Laufzeitumgebung für den Bau und Betrieb von Web-Applikationen, die auf dem Bestand der Datenbank aufsetzen (Oracle Application Express, kurz: APEX) [8]. Das Unternehmen unterstützt die Produkte von SAP durch neue speziell zugeschnittene Funktionen [9], erlaubt das Ausführen von Java-Anwendungen durch eine in die Datenbank integrierte JVM [10] und bringt mit PL/SQL eine eigene prozedurale Programmiersprache mit, welche die Verlagerung der Anwendungslogik in die Datenbank selbst ermöglicht [11]. Auch brachte das Unternehmen mit dem Real Application Cluster (kurz: RAC) in der Version 10g die Möglichkeit mit, ein Always-On Cluster aus Oracle Datenbanken zur Lastverteilung und Ausfallsicherheit aufzubauen. [12] Doch nicht nur die Datenbank selbst bringt eine Vielzahl von Funktionen mit sich. So bietet Oracle rund um sein Datenbanksystem auch unterschiedliche auf der Datenbank aufbauende Produkte auf den Markt, welche eine Verbreitung des Systems fördern. Der sogenannte Oracle Enterprise Manager [13] erlaubt beispielsweise die Administration einer Vielzahl von Datenbanken über eine einheitliche webbasierte Oberfläche.

Für besonders anspruchsvolle Szenarien bietet das Unternehmen die sogenannte Oracle Exadata an. [14] Dabei handelt es sich um eine auf den Betrieb von Datenbanken spezialisierte Appliance mit speziell angepasstem Betriebssystem, mehreren hundert Rechenkernen und zig Terabyte an Masse- sowie Hauptspeicher, um Abfragen auf sehr große Datenbanken performant bearbeiten zu können.

2.4.2 Microsoft SQL-Server

Das zweite kommerziell vertriebene Datenbanksystem, welches im Rahmen dieser Arbeit Verwendung findet, ist der von Microsoft entwickelte SQL-Server Version 2017. Dieser wurde erstmals im Jahre 1989 auf den Markt gebracht [15] und war zu Beginn ausschließlich für das Betriebssystem OS/2 konzipiert. Ebenso wie die Oracle Database ist mittlerweile auch der Microsoft SQL-Server im Unternehmensumfeld stark verbreitet und genießt in den letzten Jahren ein starkes Wachstum.

Mögliche Gründe für dieses Wachstum könnten einerseits die angekündigte Unterstützung des SQL-Servers 2017 für Linux basierte Betriebssysteme sein [16], oder einfach die Tatsache, dass die Lizenzkosten mehr als die Hälfte niedriger ausfallen als bei der Oracle Database. [17] [18] [19] Unternehmen, die den Microsoft SQL-Server entweder ausschließlich als Backend nutzen oder geringere Anforderungen beim Entwickeln eigener Anwendungen haben, werden sich aufgrund des Preises daher vermutlich eher für den SQL-Server als für die Oracle Database entscheiden. Doch auch das von Microsoft aufgesetzte Partner-Programm und die Vielzahl unterschiedlicher Produkte verleiten möglicherweise dazu, alle Lösungen aus einer Hand zu beziehen.

2.4.3 MariaDB

Die Datenbank MariaDB wurde hier als Vertreter der frei verfügbaren Open-Source Datenbanksysteme gewählt. Eingesetzt wird die Version 10.4.6., welche am 18.06.2019 von der gleichnamigen Firma aus Finnland veröffentlicht wurde. [32]

Bei MariaDB handelt es sich um ein vergleichsweise junges Datenbanksystem, welches sich im Jahre 2009 vom ursprünglichen Quellcode des Datenbanksystems MySQL abspaltete und seitdem unter der Führung von Michael Widenius, Hauptentwickler und Gründer von MySQL, weiterentwickelt wird. [20] [21] Der Grund für diese Abspaltung lag in der absehbaren Übernahme von Sun Microsystems durch die Oracle Corporation, welche im Jahre 2010 durchgeführt wurde. [22] Michael Widenius begründete die Entscheidung zur Abspaltung mit der Sorge, dass sich MySQL unter der Führung der Oracle Corporation als neuen Eigentümer negativ entwickeln könnte. Heute liegen die Rechte von MariaDB bei der MariaDB Foundation, welches es sich zur Aufgabe gemacht hat, MariaDB als freies Datenbanksystem zu schützen.

In den Jahren ab 2012 wechselten viele große Institutionen und Unternehmen wie beispielsweise Google [23], Red Hat [24] oder die Wikimedia Foundation [25] von MySQL auf MariaDB. Da ein Großteil der Linux Distributionen mittlerweile MariaDB statt MySQL ausliefert, steigt die Verbreitung der Datenbank vor allem als Backend für Webseiten seit Jahren stetig an, was auch an der Kompatibilität der von MySQL übernommenen Schnittstellen nach MariaDB liegt. [26]

2.5 Vorhandene Technologien

Nachdem alle in dieser Arbeit relevanten Datenbanken kurz vorgestellt wurden, soll dieses Kapitel einen Überblick über vorhandene Technologien vermitteln, die möglicherweise bei der Umsetzung der Anforderungen Berücksichtigung finden könnten. Diese Technologien sollen anhand der Aufgabenstellung, welche bereits in Kapitel 1 angesprochen wurde, hinsichtlich ihrer möglichen Verwendung bewertet werden. Der erste Ansatzpunkt zur Findung einer möglichen Lösung besteht darin, das eingebaute Rechte- und Rollenkonzept einer Datenbank zu prüfen und zu bewerten. Eine Rolle besteht aus einer Sammlung von Rechten, die einem Datenbankbenutzer innerhalb einer Datenbank zugewiesen werden kann. Dabei werden ihm die Rechte nicht direkt zugewiesen, sondern nur die übergeordnete Rolle. Über die Kapselung von Rechten in einer solchen Rolle wird der Aufwand zur Pflege von Berechtigungen reduziert. Ein Datenbankbenutzer erhält eine Rolle, in der sämtliche Rechte enthalten sind, die er für die Arbeit mit der Datenbank benötigt. Sollten sich die Anforderungen und somit die benötigten Rechte ändern, muss ausschließlich die Rolle angepasst werden. Ein Review der Rechte jedes Datenbankbenutzers entfällt. Leider greifen diese Rechte und Rollen nicht auf Datensatz-Ebene. So ist es zwar möglich, einem Datenbankbenutzer die Nutzung einer bestimmten Tabelle der Datenbank, entweder direkt oder indirekt über eine Rolle, zu erteilen, dieses Recht gilt dann jedoch nur für die gesamte Tabelle. Eine Einschränkung auf bestimmte Datensätze ist hier erst einmal nicht möglich. Die grundsätzliche Lösung dieses Problems ist die Erstellung und Nutzung von Views, welche je Datenbankbenutzer die Datensätze in der dahinter liegenden Tabelle bei der Anzeige filtert.

Seit der Version 2016 bietet der *Microsoft SQL-Server* ein Feature unter dem Namen *Row-Level-Security (RLS)* an. Im Kern handelt es sich um Funktionen, die an die Datenbank übergebene SQL-Statements um zusätzliche Filterprädikate erweitern. Laut Aussage von Microsoft ist die Nutzung von *RLS* funktional äquivalent mit der Erweiterung der *WHERE* Klausel zum Filtern von Datensätzen. [33] Das Unternehmen Oracle bietet im Rahmen seiner gleichnamigen Datenbanklösung eine Vielzahl von Features, welche den beschränkten Zugriff auf Zeilenebene ermöglichen sollen. Eines dieser Features ist die *Virtual Private Database (VPD)*, welches eine Verknüpfung von *Fine-Grained Access Control (FGAC)* zusammen mit einem sogenannten *Application Context* darstellt [34] [35]. Damit lässt sich *Row-Level-Security*, also der beschränkte Zugriff auf einzelne Zeilen innerhalb von Tabellen, realisieren. Es ist ebenfalls möglich, den Tabellenzugriff nur dann einzuschränken, wenn er auf bestimmte Spalten mit schützenswerten Informationen abzielt. Entweder wird eine Zeile aufgrund von Beschränkungen nicht zurückgegeben, vollständig zurückgegeben oder besonders schützenswerte Spalten innerhalb einer Zeile werden als Leer dargestellt. Dieses Feature steht aufgrund der Lizenzierung nur Nutzern der Enterprise Edition zur Verfügung.

Die technische Funktionsweise der *VPD* ist dabei ähnlich umgesetzt wie *RLS* des *Microsoft SQL-Servers*. Die Nutzung erfolgt über Richtlinien, auch *Policies* genannt, die in *PL/SQL* geschrieben und auf bestimmte Tabellen angewendet werden. Dies ermöglicht eine sehr flexible Steuerung der Zugriffsrechte, die nicht ausschließlich über den angemeldeten Benutzer, sondern auch über die aktuelle Uhrzeit, angemeldete IP-Adresse oder sonstige Felder definiert werden kann. Das Feature *Virtual Private Database (VPD)* wurde erstmals in der Version 8i vorgestellt und ausgeliefert. Für weiterführende Informationen, siehe: [37] [38] [39]

Ein weiteres Feature in diesem Umfeld ist *Oracle Label Security (OLS)*, welches als zusätzliches kostenpflichtiges Add-On zur bestehenden Enterprise Edition Lizenz erworben werden kann. *Oracle Label Security* wurde erstmals zusammen mit der Datenbankversion 9g veröffentlicht und basiert auf der Funktionsweise von *VPD*, wirbt jedoch damit, dass zum Aufbau und zur Nutzung keinerlei gesonderte Policies mehr entwickelt werden müssen, also eine funktionierende *VPD* „out-of-the-box“. Es steht dem Anwender grundsätzlich frei, auch über die Funktionsweise von *OLS* hinaus weitere Policies über die zugrunde liegende *VPD* anzulegen. Hierbei sollten jedoch besondere sicherheitsrelevante Ausnahmen berücksichtigt werden [40]. Die grundsätzliche Funktionsweise von *OLS* besteht darin, jeden Datensatz einer Tabelle mit einem sogenannten *Label* zu versehen, um sowohl deren Vertraulichkeitsstufe als auch deren Zuordnung klar festzulegen. Ein solches Label besteht aus bis zu drei Teilen: Der erste Teil ist das Level, welches die Vertraulichkeitsstufe des Datensatzes angibt. Solche Level sind hierarchisch geordnet und ermöglichen eine Klassifizierung wie beispielsweise „öffentlich“, „vertraulich“ oder „streng vertraulich“. Der zweite und dritte Bestandteil ist optional und definiert die zugeordnete Kategorie sowie die Gruppe des Datensatzes. Gruppen sind ebenfalls hierarchisch organisiert, während Kategorien eine lose Klassifizierung darstellen. Für weiterführende Informationen, siehe: [41] [42] [43]

Später stellte Oracle das Konzept *Real Application Security (RAS)*, deklariert als VPD der nächsten Generation, zusammen mit der Version 12c ihrer Datenbanklösung vor. Hierbei werden beispielsweise das Identity Management sowie das Session-Handling von Applikationen direkt in die Datenbank integriert, um Multi-Tier Architekturen auf Basis von Middleware Systemen, die häufig zum Einsatz kommen, besser zu unterstützen. [36] Für weiterführende Informationen, siehe: [44] [45]

Die Datenbank *MariaDB* bietet von Haus aus keine Funktionen, die den hier bisher beschriebenen Funktionen ähnlich sind oder einen ähnlich gearteten Einsatz möglich machen. Das Konzept zum rollenbasierten Zugriff, *Role-Based Access Control* (RBAC), beschränkt sich mit seinen Möglichkeiten auf den Zugriff von Objekten wie Tabellen oder Views, nicht jedoch auf Datensatzebene. Um die fehlende Funktion zum Filtern von Datensätzen zu implementieren, rät MariaDB zu einer Eigenentwicklung in Form von Views und Prozeduren. Ein Beispiel hierfür findet sich im offiziellen Blog. [76]

2.6 Stand der Forschung

Zugriffs- und Sicherheitsmodelle in heutigen Computersystemen basieren auf Konzepten, welche durch die Forschung konzipiert und publiziert wurden. Jedes heute verfügbare Modell basiert auf den Erkenntnissen der zuvor entwickelten Modelle und versucht, die dortigen Einschränkungen zu beseitigen um neue Anwendungsgebiete zu erschließen. Nachfolgend sollen einige Modelle und Strategien aufgeführt werden. Dabei gilt es zu berücksichtigen, dass dies nur Konzepte sind, welche sich in ihrer Ausgestaltung und dem Funktionsumfang je nach Implementierung unterscheiden können.

Die erste hier zu nennende Strategie ist *Discretionary Access Control (DAC)*, welche in mancher Literatur auch als *Identity Based Access Control (IBAC)* bezeichnet wird, da ein Zugriff an die Identität eines Benutzers gekoppelt ist. Hierbei werden Rechte für jede Ressourcen und jeden Datensatz über eine entsprechende *Access Control Lists (ACL)* definiert. In dieser werden Benutzer oder Benutzergruppen festgelegt, welche Zugriff auf eine Ressource erhalten sollen und wie weit dieser erteilte Zugriff reicht: Lesen, Schreiben, Ausführen. Der Begriff Ressource und Datensatz ist hierbei äquivalent zu betrachten. Bei *DAC* obliegt die Vergabe von Legitimationen beim Eigentümer der Ressource selbst. Das Freigeben von Ressourcen über das Anpassen der *ACL* für andere Benutzer setzt voraus, dass ein Benutzer der die Anpassung der Berechtigungen vornehmen möchte, die zu vergebenden Berechtigungen auf der jeweiligen Ressource selbst besitzt. Auch wenn *DAC* etwas veraltet wirken mag, so hat es sich doch auf vielen heute genutzten Betriebssystemen durchgesetzt. Betriebssysteme wie Windows, Linux sowie Mac OS arbeitet nach diesem Prinzip. [80]

Bei der Strategie *Mandatory Access Control (MAC)* wird jede Ressource nach bestimmten Regeln, meist ihrem Schutzbedarf, klassifiziert. Ein Benutzer erhält nur dann Zugriff auf eine angeforderte Ressource, wenn dieser mindestens eine gleichwertige oder höherwertige Klassifizierung aufweisen kann. Im Gegensatz zu *DAC* kann ein Benutzer die Klassifizierung einer Ressource nicht selbst festlegen oder ändern, auch wenn er selbst Zugriff auf diese Ressource hat. [77]

Eine klassische Ausprägung von *MAC* ist die sogenannte *Multi-Level-Security (MLS)*, da sich durch die Klassifizierung von Datensätzen nach ihrem Schutzbedarf mehrere Schichten ergeben. [82] Durch das Kernel-Modul *Security-Enhanced Linux (SELinux)* wird die Verwendung der *MAC* als ergänzende Security-Schicht zum bestehenden *DAC* innerhalb des Linux-Kernels möglich. Eine weitere Ausprägung der *MAC* stellt das sogenannte *Bell-LaPadula* Modell dar, welches *MAC* um die Eigenschaft erweitert, dass Mitarbeiter mit hoher Schutzstufe nicht in Datensätze niedrigerer Schutzstufe schreiben oder Datensätze mir niedrigerem Schutzbedarf erstellen dürfen, um so die Weitergabe von vertraulichen Informationen an Mitarbeiter mit geringerer Schutzstufe zu verhindern. Dieses Modell wurde jedoch mittlerweile abgelöst. [81]

Eine Weiterentwicklung der *Multi-Level-Security* ist die Erweiterung der horizontalen Schichten um vertikale Ausprägungen, welche *Lattice-Based Access Control (LBAC)* genannt wird. Hierbei ist einem Datensatz nicht nur eine Klassifikation nach ihrem Schutzbedarf, sondern zusätzlich auch eine Kategorie oder ein Label zugeordnet. Dadurch wird ein Datenbestand nicht nur horizontal durch ihre Schutzbedarf-Klassifikation, sondern auch nach ihrer Kategorie beziehungsweise ihrem Label separiert. Es bildet sich eine Gitter-Struktur, daher der Begriff „Lattice“ (engl.: Gitter). [83] Dem aufmerksamen Leser mag die Ähnlichkeit zur *Oracle Label Security* aus dem vorherigen Kapitel aufgefallen sein. Bei *OLS* handelt es sich tatsächlich um eine Implementierung des *LBAC*-Modells innerhalb der Oracle Database.

Außerdem sei an dieser Stelle noch das *Role-Based Access Control (RBAC)* beschrieben, welches in Unternehmen stark verbreitet ist, da es mittels verschachtelter Rollen die hierarchische Struktur von Unternehmen gut abbilden kann. Dieses Modell wird auch als *Rule-Based Access Control (RBAC)* bezeichnet. Ein typisches Beispiel für dieses Modell sieht man bei der Verwendung eines Verzeichnisdienstes wie dem *Active Directory (AD)*, wo mit verschachtelten Gruppen gearbeitet wird, die einem oder mehreren Benutzern zugewiesen werden können. Der Zugriff auf bestimmte Ressourcen innerhalb des Unternehmens ist an die vorherige Zuweisung der benötigten Rolle gebunden. [78] [79]

Als weiterer Vertreter soll das *Attribute-Based Access Control (ABAC)* Modell genannt werden. In diesem Modell werden Zugriffe aufgrund von Eigenschaften der angeforderten Ressource oder Eigenschaften des anfragenden Benutzers gewährt. Häufig wird nach der Authentifizierung des Benutzers ein Authentifikations-Token an den Client übertragen, welcher anschließend als Attribut für die Anforderung von Ressourcen genutzt werden kann. [84]

Die hier gestellten Zugriffs-Modelle und Strategien lassen sich in der Praxis kombinieren. Auch ist nicht immer eine scharfe Trennung der einzelnen Modelle möglich. So kann das eine Gruppenzugehörigkeit innerhalb des *RBAC* als Attribut bei *ABAC* interpretiert und genutzt werden. Auch lässt sich *DAC*, wie oben bereits kurz erwähnt, mittels *MAC* um eine zusätzliche Zugriffs-Schicht erweitern.

Aktuelle Forschungstätigkeiten zu diesem Thema findet sich zurzeit vorwiegend in den Bereichen der Adaption vorhandener Modelle auf die neu geschaffenen Cloud-Umgebungen [85], NoSQL-Datenbanken [87], die Anbindung des Internet-of-Things [86] sowie der Einsatz in verteilter Infrastruktur [88].

Aufgrund des Fokus, den diese Arbeit hat, können diese neuen Forschungsgebiete jedoch nicht als Bestandteil in diese Arbeit mit aufgenommen werden. Die Anforderung ist, dass das hier zu entwickelnde Modell mit klassischen SQL-Statements erstellt wird, was eine Adaption auf andere Datenbanksysteme möglich machen soll.

3 Anforderungsanalyse

Dieses Kapitel beschäftigt sich mit den Anforderungen an das zu entwickelnde Modell. In Kapitel 5 wird im Rahmen eines Reviews das Ergebnis mit den hier konzipierten Anforderungen verglichen.

3.1 Zugriffsschicht für fachliche Daten

Die folgenden Anforderungen werden an die zu erstellende Zugriffsschicht gestellt:

Unveränderlichkeit der Fachlichkeit: Jede Tabelle, welche fachliche Datensätze beinhaltet, soll mit einer Zugriffsschicht versehen werden können. Das Hinzufügen der Zugriffsschicht soll jedoch weder die Struktur der fachlichen Tabellen noch die darin enthaltenen Datensätze verändern oder erweitern. Daraus folgt, dass die Zugriffsschicht zwar für die Legitimation von Zugriffen auf die betroffenen Datensätze verwendet wird, die fachlichen Daten jedoch vollkommen losgelöst von der Zugriffsstruktur vorliegen und verwendet werden können.

Urheberzuordnung: Erhobene Daten werden nicht einem Mitarbeiter als urhebende Instanz zugeordnet, sondern der Organisationseinheit, welcher er primär zugeordnet ist. Damit soll sichergestellt werden, dass die betroffenen Datensätze nach dem Wechsel des Mitarbeiters in ein anderes Team oder seinem Ausscheiden aus dem Unternehmen weiterhin zur Verfügung stehen, gelesen und verarbeitet werden können.

Transitiver Zugriff: Aufgrund der hierarchischen Struktur, in der die einzelnen Organisationseinheiten angeordnet sind, soll entlang der Parent-Relation der Hierarchie der Zugriff auf die zugeordneten Datensätze Mitarbeitern in höherer Position zugänglich gemacht werden. Das heißt, dass ein übergeordneter Mitarbeiter aufgrund seiner primären oder sekundären Organisationseinheit ebenso Zugriff auf die Datensätze erhalten soll, auch wenn die urhebende Organisationseinheit seiner untergeordnet ist und er somit nicht direkt der urhebenden Organisationseinheit zugewiesen ist.

Abstraktion: Sämtliche Zugriffe auf die fachlichen Daten sollen, vorausgesetzt diese unterliegen einer Zugriffskontrolle, durch diese Zugriffsschicht abstrahiert werden. Der Softwareentwickler muss seine SQL-Statements an keine veränderten Tabellenstrukturen anpassen. Sowohl das Lesen als auch das Schreiben, Ändern oder Löschen von Daten erfolgt über die Zugriffsschicht.

3.2 Operativer Betrieb

Die folgenden Anwendungsfälle aus dem operativen Betrieb sollen durch das Modell berücksichtigt werden. Für alle diese Anforderungen gilt, dass diese nur von einem autorisierten Administrator durchgeführt werden können, was nachfolgend für die hier genannten Punkte angenommen und nicht mehr explizit erwähnt wird.

Mitarbeiterfluktuation: Mitarbeiter sollen dem Unternehmen beitreten, aus diesem ausscheiden oder ggf. Abteilungen / Teams und somit ihre primäre Organisationseinheit ändern können.

Multiple Zuordnung: Jeder Mitarbeiter hat eine primäre Organisationseinheit, der er zugeordnet ist. Es soll jedoch möglich sein, einem Mitarbeiter im Rahmen von Projektarbeit oder ähnlich gearteten Tätigkeiten mehr als eine sekundäre Organisationseinheit zuzuweisen. Ein Ausscheiden eines Mitarbeiters aus einem Projekt hat hierbei keinen Einfluss auf die erhobenen Datensätze.

Veränderungsprozesse: Die Umstrukturierung von Organisationseinheiten soll möglich sein, ohne dass eine Umstrukturierung der Mitarbeiter zu ihren zugeordneten Organisationseinheiten nötig wird. Nach dem Verschieben einer Organisationseinheit an eine andere Position innerhalb der Hierarchie sollen sämtliche Anforderungen aus dem Kapitel 3.1 weiterhin erfüllt sein.

Unveränderlichkeit der Struktur: Eine Veränderung der Organisationsstruktur sorgt nicht für eine Veränderung der Tabellenstrukturen innerhalb der Datenbank. Die Tabellenstruktur, in der die Organisationseinheiten und Mitarbeiter sowie ihre Zuordnungen untereinander abgelegt werden, bleiben unverändert. Eine Veränderung der Organisationsstruktur erfolgt ausschließlich über die Datensätze, welche ihren Aufbau beschreiben.

Zyklenfreiheit: Die Administration der Organisation sowie die Anpassung ihrer Struktur soll nicht zur versehentlichen Entstehung von Zyklen innerhalb der Hierarchie führen. Eine Organisationseinheit soll niemals eine andere Organisationseinheit direkt oder transitiv über sich haben, wenn sich diese übergeordnete Organisationseinheit unterhalb der eigentlichen Organisationseinheit befindet.

4 Umsetzung

4.1 Aufbau

Die Validierung von Zugriffsrechten bei dem Zugriff auf Datensätze einer Datenbank wird häufig innerhalb einer Applikation sichergestellt. Die dahinter liegende Datenbank hat nicht selten ausschließlich die Aufgabe, die in Konfigurationsdateien hinterlegten Zugangsdaten der Anwendung bei Öffnung einer neuen Session zu prüfen und den Zugriff zuzulassen oder eben zu verweigern. In vielen Fällen findet datenbankseitig keine weitere Überprüfung statt.

Da ein Anwender meist direkt mit einer Applikation oder über ein dafür bereitgestelltes Frontend mit der dahinter liegenden Applikation in Berührung kommt, werden fehlerhaft implementierte Validierungen innerhalb der Applikation direkt an die Datenbank weitergeleitet. Eine sogenannte SQL-Injection, das Einschleusen von schadhaften SQL-Statements, führt nicht selten zu Datenverlust oder mindestens dem Diebstahl sensibler Informationen.

Aus diesem Grund soll die Validierung von Berechtigungen einzelner Benutzer bei Zugriff auf bestimmte Datensätze in die Datenbank selbst verlegt werden. Die datenbankseitigen Sicherheitsmechanismen, welche den Zugriff auf einzelne Tabellen oder Views der Datenbank sicherstellen, sollen hierbei jedoch weiterhin Bestand haben.

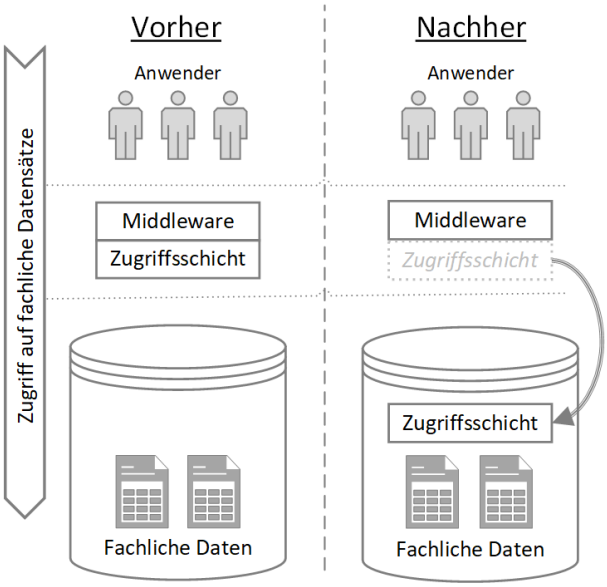

Bild 11: Aufbau – Verlagerung der Zugriffsschicht

Um eine möglichst saubere Abgrenzung der jeweiligen Inhalte zu gewährleisten, ist das Modell innerhalb der Datenbank in drei wesentliche *Schemata* unterteilt.

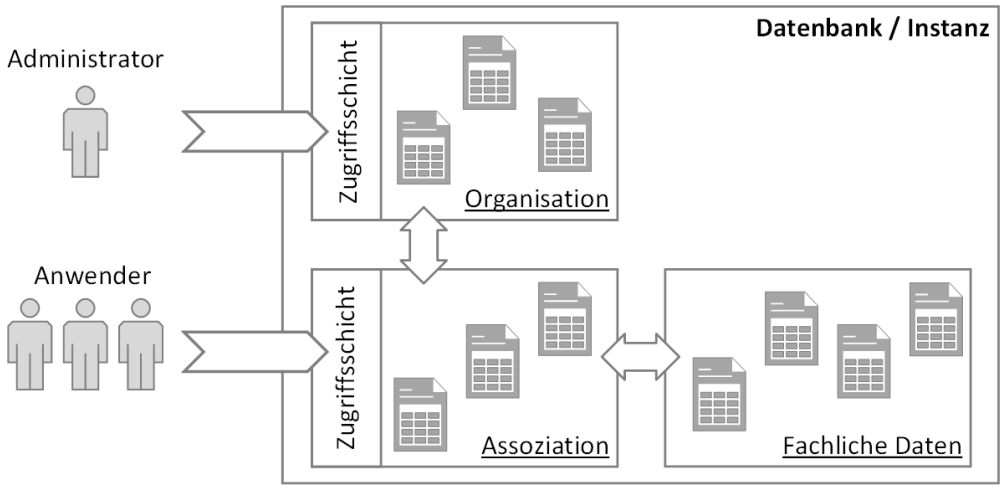

Bild 12: Aufbau – Datenbankstrukturen

Im Schema *Organisation* wird die komplette hierarchische Organisationsstruktur inklusive aller Mitarbeiter und deren Zuordnung zu ihren Organisationseinheiten hinterlegt. Eine separate Zugriffsschicht erlaubt einem Administrator die Anpassung der darin hinterlegten Daten und stellt über diese Schicht die Zyklenfreiheit innerhalb der Organisationshierarchie sicher. Dieses Schema wird je Datenbank nur einmal benötigt, unabhängig von der Anzahl der zu schützenden Schemata mit fachlichen Daten.

Der Schema *Fachliche Daten* beinhaltet das fachliche Datenmodell, welches ganz oder teilweise über eine Zugriffsschicht innerhalb von *Assoziation* abgesichert werden soll. Daraus ergibt sich, dass für jedes zu schützende fachliche Datenmodell, also jedes fachliche Schema, ein zusätzliches Assoziations-Schema benötigt wird, um den Zugriff auf das zu schützende Schema zu kapseln.

In *Assoziation* hingegen werden alle Verknüpfungen zwischen dem abstrahierten fachlichen Schema mit seinen Datensätzen und der Organisationsstruktur hinterlegt. Außerdem beinhaltet es Informationen über gesetzte Restriktionen auf Datensatzebene, was in Kapitel 7.2.3 als mögliche Weiterentwicklung des Modells angesprochen wird. Der jedoch wichtigste Abschnitt ist die hier aufsetzende Zugriffsschicht. Diese ist das Herzstück des Modells und stellt mittels eigener Funktionen und Views die Konsistenz der hinterlegten Berechtigungen sicher. Das Ziel soll sein, die zugrundeliegende fachliche Datenbankstruktur über das Assoziations-Schema so nach außen zu kapseln, dass dem Anwender (oder der Applikation) gar nicht bewusst ist, dass eine Zugriffsschicht mögliche Datensätze aufgrund von Restriktionen verschleiert.

Das Datenbanksystem MariaDB verwendet den Begriff *Schema* als Synonym für *Datenbank*, was sich in der entsprechenden Syntax zeigt. Ein sogenanntes Schema-Konzept, in dem Tabellen innerhalb der gleichen Datenbank in logische Gruppen zusammengefasst werden können, bietet MariaDB nicht. [46] Da die einzelnen Datenbanken jedoch über den gleichen Prozess verwaltet werden und dadurch ein übergreifender Zugriff möglich ist, werden hier keine gesonderten Lösungen erarbeitet werden müssen.

Der *Microsoft SQL-Server* hingegen bringt ein Schema-Konzept mit sich. Jede Datenbank besitzt hier bereits nach der Erstellung ein Default-Schema mit dem Namen *dbo*, welchem neu erstellte Tabellen automatisch zugeordnet werden, sollte kein alternatives Schema angegeben werden. Das Anlegen weiterer Schemata ist natürlich möglich. Die Nutzung von Schemata hat hier zwei besondere Vorteile: Einerseits entsteht keine Namenskollision bei der Erstellung von Objekten wie beispielsweise Tabellen, Indizes oder Views, da die Namen der einzelnen Objekte nur innerhalb ihres Schemas eindeutig sein müssen. Der jedoch wichtigere Vorteil ist das Zuweisen von Berechtigungen für dieses Schema über zuvor definierte Rollen. Die zugewiesenen Rechte vererben sich auf alle enthaltenen Objekte des Schemas. [47] [48]

Im Falle von *Oracle Database*, welche ebenfalls einen Schema-Konzept mitbringt, wird für jeden Datenbankbenutzer automatisch ein gleichnamiges Schema angelegt. Der erstellte Benutzer und sein Schema sind sehr eng miteinander verbunden. Das äußert sich einerseits darin, dass das Schema als eigenes Objekt, anders als bei dem *SQL-Server*, nicht direkt angesprochen und somit auch nicht mit Rechten oder Rollen belegt werden kann. Die zweite Auffälligkeit, auch wenn diese nicht direkt relevant für diese Arbeit ist, ist die Tatsache, dass mit dem Löschen eines Benutzers automatisch auch das verknüpfte Schema mit allen enthaltenen Objekten gelöscht wird. Bei der Löschung eines Benutzers meldet die Datenbank zwar, dass ein Schema mit Daten vorhanden ist, doch diese Meldung wird nicht selten ignoriert. Dieser Umstand hat bei einigen Oracle Datenbankadministratoren bereits zu aufwändigen Restore-Arbeiten geführt, weil versehentlich produktiv genutzte Strukturen und Daten entfernt wurden. [49] [50]

4.2 Berechtigungskonzept

Um das Modell möglichst robust und manipulationssicher zu gestalten, wird jedes Schema innerhalb der Datenbank mit gesondert berechtigten Datenbankbenutzern angelegt. In Summe kommen fünf Datenbankbenutzer zum Einsatz, die über ihre Rechte voneinander abgegrenzt werden. Dabei wird hier in allen Beispielen von einem technischen Datenbankbenutzer ausgegangen, der keine direkte oder zwingend notwendige Zuordnung zu einem Mitarbeiter der Organisation hat. Entsprechend ihrer angedachten Funktion können die Datenbankbenutzer daher in zwei Gruppen unterteilt werden: Administratoren und Benutzer, hier jeweils als Suffix an den Namen zur besseren Unterscheidung angehangen.

Ein Administrator im Sinne dieses Kapitels darf nicht mit einem Datenbankadministrator verwechselt werden, welcher die Datenbank ganzheitlich administriert und ihre Funktionsfähigkeit sicherstellt. In dem hier beschriebenen Kontext hat ein Administrator die Aufgabe, die Strukturen innerhalb eines Schemas anzulegen. Er fungiert auch als Eigentümer der erstellten Strukturen. Unter diesen Strukturen werden beispielsweise alle Tabellen, Views, Indizes, Trigger und ähnliche Objekte verstanden, welche die Verwaltung der darin gespeicherten Datensätze ermöglichen.

Im Gegensatz zu einem Administrator hat ein Benutzer die Aufgabe, die bereitgestellten Strukturen zu verwenden um die Speicherung und Verarbeitung von Datensätzen durchzuführen. Diese klare Trennung der hier beschriebenen Verantwortlichkeiten und Funktionen ist aus sicherheitstechnischen Aspekten notwendig, um bei unerwartetem Einschleusen einzelner SQL-Statements das Löschen ganzer Tabellen zu verhindern und den entstandenen Schaden auf einzelne Teilbereiche der Datenbank zu beschränken.

Schema	**Benutzer**	**Berechtigungen je Schema**		
		Organisation	**Assoziation**	**Fachl. Daten**
Organisation	mt_org_Admin	$x_1 = \{c, a, d\}$		
	mt_org_User	$x_2 = \{s, i, u, d\}$		
Assoziation	mt_ass_Admin		$x_1 = \{c, a, d\}$	$x_3 = \{s\}$
	mt_ass_User	$x_3 = \{s\}$	$x_2 = \{s, i, u, d\}$	$x_2 = \{s, i, u, d\}$
Fachl. Daten	mt_dat_Admin			$x_1 = \{c, a, d\}$

Tabelle 1: Konzept – Datenbankbenutzer mit Berechtigungen

Die verschiedenen Datenbankbenutzer können gemäß dieses Berechtigungskonzepts drei unterschiedliche Rollen innehaben. Eine Rolle definiert eine Menge von gewährten Berechtigungen, bezogen auf ganze Schemata oder einzelne Objekte innerhalb eines Schemas.

Die Rolle $\boldsymbol{x_1}$ beinhaltet die Rechte der jeweiligen Administratoren $Create, Alter, Drop$, also die Rechte zur Erstellung, Modifikation und Löschung der Strukturen innerhalb ihres jeweiligen Schemas.

Die Rolle $\boldsymbol{x_2}$ hingegen hat die Rechte *SELECT*, *INSERT*, *UPDATE*, *DELETE*. Dabei handelt es sich um die Befehlssätze zum Lesen, Einfügen, Ändern oder Löschen der vom jeweiligen Administrator angelegten Tabellenstrukturen. Bei der letzten Rolle $\boldsymbol{x_3}$ handelt es sich um eine Untergruppe von $\boldsymbol{x_2}$, welche nur *SELECT*, also das reine Lesen von Datensätzen, erlaubt. In allen genannten Schemata ist es die Aufgabe des Administrators, sich ausschließlich um den Aufbau der Strukturen zu kümmern. Der Umgang mit den darin enthaltenen Datensätzen liegt bei dem Benutzer.

Sowohl der Administrator als auch der Benutzer des Schemas *Organisation* bewegen sich ausschließlich in ihrem zugeordneten Schema der Datenbank. Sie benötigen für ihren Einsatzzweck keine schemaübergreifenden Rechte. Der Administrator legt die notwendigen Strukturen sowie die in Bild 12 dargestellte Zugriffsschicht an und der Benutzer nutzt diese zur Verwaltung der Organisationseinheiten und Mitarbeiter, welche als Datensätze innerhalb der Tabellen vorliegen.

Im Schema *Fachliche Daten* gibt es hingegen ausschließlich einen Administrator, welcher für den Aufbau der fachlichen Tabellenstrukturen zuständig ist. Hier ist kein Benutzer für den Zugriff auf die enthaltenen fachlichen Datensätze vorgesehen, da ein Zugriff über die Assoziationsschicht erfolgt und vom dort hinterlegten Benutzer durchgeführt wird.

Auch innerhalb des Schemas *Assoziation* gibt es einen Administrator, wie in den übrigen Schemata, der sowohl von seinen Rechten als auch seinem Einsatzzweck mit den anderen Administratoren identisch ist. Die wesentliche Besonderheit liegt bei dem Benutzer. Dieser hat schemaübergreifende Berechtigungen. Er besitzt die vollen Rechte zur Verwaltung der Datensätze in *Fachliche Daten*, was darin begründet ist, dass sich die steuernde Zugriffsschicht in *Assoziation* befindet und alle Operationen auf den fachlichen Datensätzen durch diese Schicht hindurch an das fachliche Datenmodell weitergeleitet werden. Außerdem besitzt er die vollen Rechte zur Verwaltung der in diesem Schema abgelegten Datensätze. In diesem Schema findet die Speicherung der Urheberschaft, siehe dazu Kapitel 3.1, zum jeweiligen fachlichen Datensatz statt. Das heißt, dass hier alle Verknüpfungen zwischen fachlichen Datensätzen und Organisationseinheiten abgelegt werden, was auch das Recht zum Lesen aus dem Schema *Organisation* erklärt.

4.3 Datenbanken und Benutzern

Alle nachfolgenden Quellcode-Ausschnitte dienen nur der Verdeutlichung. Um das in dieser Arbeit entwickelte Modell zu nutzen, sollten ausschließlich die SQL-Statements aus dem Anhang dieser Arbeit verwendet werden. Die nachfolgenden Beispiele sind, um den Umfang zu reduzieren, ausschließlich für MariaDB aufgeführt, wenn dies nicht gesondert gekennzeichnet ist. Der Quellcode für die anderen Datenbanksysteme befindet sich im Anhang des Dokuments.

Folgend werden nun die gerade eben erläuterten Benutzer, Strukturen und Berechtigungen gemäß Kapitel 4.2 angelegt.

```
CREATE SCHEMA 'mt_org';
CREATE SCHEMA 'mt_data';
CREATE SCHEMA 'mt_ass';

-- Organisation: Admin
CREATE USER 'mt_org_admin'@'localhost' IDENTIFIED BY 'mt_org_admin';
GRANT ALL PRIVILEGES ON 'mt_org'.* TO 'mt_org_admin'@'localhost';
-- Organisation: User
CREATE USER 'mt_org_user'@'localhost' IDENTIFIED BY 'mt_org_user';
GRANT INSERT, UPDATE, SELECT, DELETE ON 'mt_org'.* TO 'mt_org_user'@'localhost';

-- Assoziation: Admin
CREATE USER 'mt_ass_admin'@'localhost' IDENTIFIED BY 'mt_ass_admin';
GRANT ALL PRIVILEGES ON 'mt_ass'.* TO 'mt_ass_admin'@'localhost';
-- Assoziation: User
CREATE USER 'mt_ass_user'@'localhost' IDENTIFIED BY 'mt_ass_user';
GRANT INSERT, UPDATE, SELECT, DELETE ON 'mt_ass'.* TO 'mt_ass_user'@'localhost';
GRANT INSERT, UPDATE, SELECT, DELETE ON 'mt_data'.* TO 'mt_ass_user'@'localhost';
GRANT SELECT ON 'mt_ass'.* TO 'mt_ass_user'@'localhost';

-- Fachl. Daten: Admin
CREATE USER 'mt_data_admin'@'localhost' IDENTIFIED BY 'mt_data_admin';
GRANT ALL PRIVILEGES ON 'mt_data'.* TO 'mt_data_admin'@'localhost';
FLUSH PRIVILEGES;
```

Tabelle 2: MariaDB – Erstellen der Datenbanken, Benutzer und Berechtigungen

Wie hier zu erkennen ist, erhalten die jeweiligen Administratoren-Zugänge alle Rechte für das von ihnen zu verwaltende Datenbank-Schema. Diese Rechte sind notwendig, um nachfolgend alle relevanten Strukturen anzulegen, die später von den normalen Benutzerzugängen genutzt werden sollen.

Es versteht sich von selbst, dass die abgebildeten Kennwörter vor einer direkten Verwendung der SQL-Statements entsprechend geändert werden müssen.

Zugunsten der Lesbarkeit wurde im Rahmen dieser Arbeit auf die separate Erstellung von Rollen für die Zuweisung der Berechtigungen verzichtet. Die Erstellung eines separaten Rechte- und Rollenkonzepts wird als Ausblick in Kapitel 7.2.2 angesprochen.

4.4 Aufbau – Organisation

In diesem Kapitel wird die Struktur innerhalb des Schemas *Organisation* angelegt, in dem sich später die Datensätze befinden, welche den Aufbau der Hierarchie sowie die Mitarbeiter und ihre Zuordnung zu den Organisationseinheiten beinhalten.

```
CREATE TABLE `OrganizationUnit` (
    `ID`     INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `PARENT` INT,
    `NAME` VARCHAR (128) NOT NULL DEFAULT ''
);
ALTER TABLE `OrganizationUnit` ADD FOREIGN KEY(`PARENT`)
REFERENCES `OrganizationUnit`(`ID`) ON DELETE RESTRICT ON UPDATE RESTRICT;
```

Tabelle 3: MariaDB – Erstellen der Tabelle OrganizationUnit

Die Tabelle *OrganizationUnit* ist für die Speicherung und Verwaltung aller in der Organisation enthaltenen Organisationseinheiten vorgesehen. Sie kann grundsätzlich um zusätzliche organisatorische Aspekte erweitert werden.

Von wesentlicher Bedeutung für dieses Modell sind nur die Spalten *ID* und *PARENT*, welche ausschlaggebend für die Bildung der Hierarchie sind. Die Spalte *NAME* wurde nur aus Gründen der Übersichtlichkeit während der Modellierung mit aufgenommen, um spätere Testszenarien leichter verständlich zu gestalten.

```
CREATE TABLE `Employee` (
    `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `OU` INT NOT NULL,
    `NAME` VARCHAR (128) NOT NULL DEFAULT ''
);
ALTER TABLE `Employee` ADD FOREIGN KEY(`OU`)
REFERENCES `OrganizationUnit`(`ID`) ON DELETE RESTRICT ON UPDATE RESTRICT;
```

Tabelle 4: MariaDB – Erstellen der Tabelle Employee

Innerhalb der Tabelle *Employee* Tabelle sollen die einzelnen Mitarbeiter-Datensätze sowie deren jeweilige Zuordnung zur Primär-Organisationseinheit vorgehalten werden.

Auch bei dieser Tabelle gilt, dass *NAME* nur aus Gründen der Übersichtlichkeit aufgenommen wurde. Entscheidend für die Funktionalität des Modells sind die Spalten *ID* sowie *OU*. Diese Tabelle kann um weitere Identifikationsmerkmale erweitert werden, um in einem zukünftigen Ausbauschritt die Anbindung an einen Verzeichnisdienst zu erleichtern, was jedoch explizit nicht Bestandteil dieser Arbeit ist. Die Fremdschlüsselbeziehung der Tabelle *Employee* auf die Tabelle *OrganizationUnit* stellt die in Kapitel 2.2.3 benannte Primär-Assoziation zwischen einem Mitarbeiter und seiner zugeordneten Organisationseinheit dar.

```
CREATE TABLE `Association` (
    `EID`  INT NOT NULL,
    `OUID` INT NOT NULL,
    PRIMARY KEY (`EID`, `OUID`)
);
ALTER TABLE `Association` ADD FOREIGN KEY(`EID`)
REFERENCES `Employee`(`ID`) ON DELETE RESTRICT ON UPDATE RESTRICT;
ALTER TABLE `Association` ADD FOREIGN KEY(`OUID`)
REFERENCES `OrganizationUnit`(`ID`) ON DELETE RESTRICT ON UPDATE RESTRICT;
```

Tabelle 5: MariaDB – Erstellen der Tabelle Association

Die hier erzeugte *Association*-Tabelle dient dazu, die ebenfalls in Kapitel 2.2.3 beschriebenen Sekundär-Assoziationen zwischen Organisationseinheiten der Tabelle *OrganizationUnit* und den ihn zugeordneten Mitarbeitern der Tabelle *Employee* herzustellen. Um eine doppelte Zuweisung von Mitarbeitern zu einer Organisationseinheit zu verhindern, wurde ein kombinierter Primärschlüssel verwendet.

Aus den drei erzeugten Tabellen ergibt sich das folgende Modell:

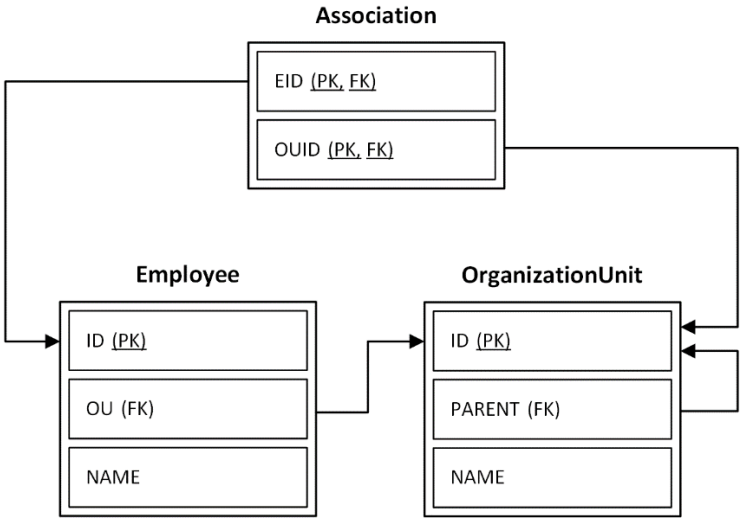

Bild 13: Tabellen, Modell Organisation

Da die Anforderung an das Modell besteht, entlang einer Hierarchie auch die Datensätze zu erfassen, die einer untergeordneten Organisationseinheit zugeordnet sind, muss nun noch eine View erstellt werden, welche abhängig vom Anwender rekursiv alle untergeordneten Organisationseinheiten ermittelt.

Dies lässt sich mittels sogenannter *Common Table Expressions (CTE)* lösen. Dabei handelt es sich um Unter-Abfragen, die in einem SQL-Statement definiert und mehrfach wiederverwenden werden können. Die zugrundeliegende Syntax entspricht der von Views, jedoch werden diese CTEs nicht mit einem Namen versehen und oder persistent innerhalb der Datenbank gespeichert. Der Zugriff ist somit ausschließlich innerhalb des definierenden SQL-Statements möglich.

4.4.1 Common Table Expressions

Sogenannte *Common Table Expressions* werden mittels *WITH* Klausel definiert und können im eigentlichen SQL-Statement anschließend beliebig oft wiederverwendet werden. Es ist ebenso möglich, mehrere CTEs direkt hintereinander zu definieren:

```
WITH temp_query_name (column1, column2, ...) AS (
    SELECT ...
), temp_query_name_2 AS (
    SELECT ...
)
SELECT ...
```

Tabelle 6: MariaDB – Aufbau Common Table Expression (CTE)

Seit dem Standard SQL:1999 existiert eine Spezifikation für CTEs, die zusätzlich die Klausel *RECURSIVE* bei der Deklaration erlauben. [51] Diese Klausel erlaubt das rekursive Ausführen von CTEs, was an dieser Stelle genutzt werden soll, um eine View zu erzeugen, die alle direkten, sekundären und transitiven Organisationseinheiten eines Mitarbeiters ausgibt. Vorher soll jedoch der grundsätzliche Aufbau von CTEs unter Verwendung der Klausel *RECURSIVE* anhand der Fakultät-Funktion dargestellt werden:

```
WITH RECURSIVE fact (n, f) AS (
    SELECT 0, 1          -- Start / Anker
    UNION ALL
    SELECT n+1, (n+1)*f -- Rekursiver Aufruf
    FROM fact
    WHERE n < 9          -- Abbruch-Bedingung
)
SELECT * FROM fact;
```

Tabelle 7: MariaDB – Beispiel, Berechnung der Fakultät mit CTE recursive

Wie sich in Tabelle 7 erkennen lässt, besteht eine rekursive Common Table Expression aus drei Teilen: Dem Start, welcher auch häufig als *Anker* bezeichnet wird, dem rekursiven Aufruf sowie der Abbruchbedingung. Das hier gezeigte Beispiel liefert die folgende Ausgabe:

n	fact
0	1
1	1
2	2
3	6
4	26
5	120
6	720
7	5.040
8	40.320

Tabelle 8: MariaDB – Ausgabe Fakultät-Funktion

4.4.2 Aufbau der View Employee_OU

Mit dem nun gewonnenen Wissen wird eine View angelegt, welche im späteren Verlauf wiederverwendet werden kann. Diese soll alle Mitarbeiter-IDs mit deren Organisationseinheiten ausgeben, unabhängig davon ob es sich um die Primär-OU des Mitarbeiters, um eine Sekundär-OU oder eine transitiv untergeordnete OU handelt. Die View ist folgend definiert:

```
CREATE VIEW `employee_ou` AS
    WITH RECURSIVE recursive_cte AS (
        -- Startpunkt 1: Primäre zugewiesene OU je Mitarbeiter ermitteln
        SELECT
            e.ID       AS "S",  -- Start E-ID
            org.ID     AS "ID"  -- OU-ID
        FROM employee e
        INNER JOIN organizationunit org ON org.ID = e.OU
        UNION

        -- Startpunkt 2: Sekundäre zugewiesene OUs je Mitarbeiter ermitteln
        SELECT
            e.ID       AS "S",  -- Start E-ID
            org.ID     AS "ID"  -- OU-ID
        FROM employee e
        INNER JOIN association a ON a.EID = e.ID
        INNER JOIN organizationunit org ON org.ID = a.OUID
        UNION

        -- Rekursiv alle darunter liegende OUs ermitteln
        SELECT
            recursive_cte.S   AS "S",  -- Start E-ID
            org.ID            AS "ID"  -- OU-ID
        FROM organizationunit org
        INNER JOIN recursive_cte ON org.PARENT = recursive_cte.ID
    )

SELECT S AS "EID", ID AS "OUID" FROM recursive_cte GROUP BY S, ID;
```

Tabelle 9: MariaDB – Rekursive View EMPLOYEE_OU

In Tabelle 9 ist erkennbar, dass hier ein *Anker* (= Startpunkt) definiert ist, welcher aus zwei einzelnen SQL-Statements besteht. Das ist notwendig, da das Start-Element der rekursiven Suche sowohl aus der primär zugewiesene OU des Mitarbeiters, als auch aus allen sekundär direkt verknüpften OU-Nummern der *Association* Tabelle besteht. Eine Abbruchbedingung wie in Tabelle 7 ist hier nicht notwendig, da das SQL-Statement aufgrund der *JOIN*-Operation im rekursiven Teil terminiert, wenn keine weiteren Organisationseinheiten unterhalb der bereits erkannten Organisationseinheiten verfügbar sind. Der *UNION* Befehl sorgt für die Verknüpfung der Rückgabewerte aller einzelnen *SELECT*-Statements und gibt ausschließlich eindeutige Wertpaare zurück. Das ist wichtig zu erwähnen, da es sonst passieren kann, dass Einträge mehrfach in der Ergebnismenge auftauchen. So etwas kann passieren, wenn ein Mitarbeiter sowohl direkt einer OU sowie auch einer ihr direkt oder indirekt übergeordneten OU zugeordnet ist.

Das nachfolgende Beispiel soll die hier beschriebene Funktionsweise grafisch verdeutlichen:

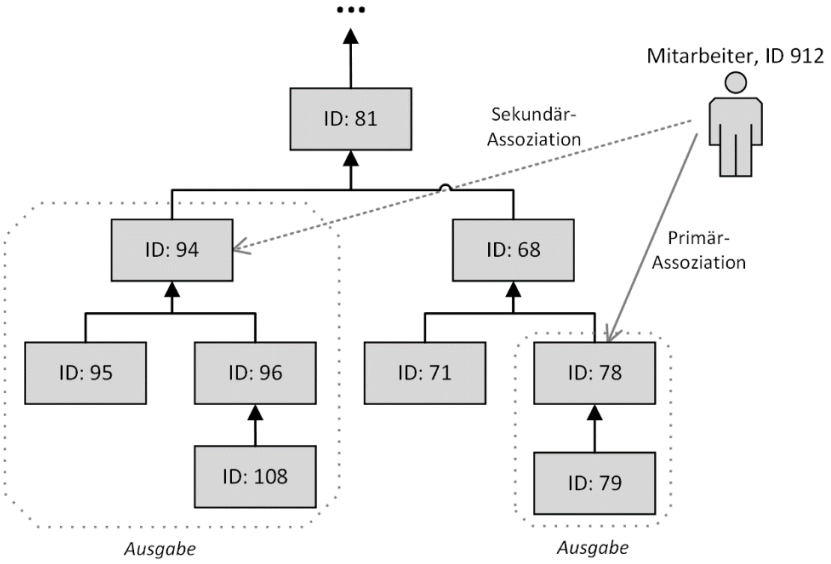

Bild 14: Grafisches Beispiel – Ergebnis der Abfrage auf EMPLOYEE_OU

Die beiden Organisationseinheiten mit den IDs 78 sowie ID 94 stellen die primär und sekundär zugewiesenen Organisationeinheiten dar. Über die View *EMPLOYEE_OU* wird nun eine Zuordnung zwischen allen verfügbaren Mitarbeitern und deren Organisationseinheiten ermöglicht. Im hier gezeigten Beispiel gibt es jedoch nur einen einzigen Mitarbeiter mit nur einer sekundären Organisationseinheit, was der Übersichtlichkeit innerhalb der Abbildung dient.

```
SELECT * FROM employee_ou;
```

Tabelle 10: MariaDB – Abfrage der View EMPLOYEE_OU

Das Ergebnis des SQL-Statements aus Tabelle 10 ist, wie bereits im Bild 14 entsprechend markiert, folgend dargestellt:

EID	OUID
912	78
912	94
912	79
912	95
912	96
912	108

Tabelle 11: MariaDB – Ausgabe der View EMPLOYEE_OU

4.4.3 Verwaltung der Organisationsstruktur

Bei der Verwaltung und Administration sollte dieses Schema gewisse Sicherheitsmechanismen mitbringen, die ein Erzeugen ungewollter Inkonsistenzen innerhalb der Organisation präventiv verhindern. Daher können die hier getroffenen Maßnahmen in zwei Kategorien einsortiert werden: Sicherheit und Konsistenz.

Unter *Konsistenz* wird in diesem Zusammenhang die Einhaltung der Fremdschlüssel-Beziehungen verstanden, welche definieren, dass zu jedem gesetzten Fremdschlüssel-Attribut ein entsprechender Primärschlüssel-Datensatz in der referenzierten Spalte der Ziel-Tabelle vorhanden sein muss. Diese Beziehung lässt sich mit datenbankseitigen Mitteln recht einfach realisieren, wie in Tabelle 3, Tabelle 4 und Tabelle 5 zu sehen ist. Diese initial eingerichteten Konsistenzbedingungen, auch Constraints genannt, werden am Ende einer jeden Transaktion von der Datenbank auf Einhaltung geprüft. So wird sichergestellt, dass kein Fremdschlüssel-Eintrag auf einen Primärschlüssel verweist, der möglicherweise nicht mehr vorhanden ist. Ausgenommen von dieser Regel sind Fremdschlüssel, welche explizit mit *NULL* befüllt werden können.

Unter *Sicherheit* fallen die Maßnahmen, welche für die die Vermeidung von Zyklen innerhalb der Organisationsstruktur sorgen oder die Bildung von Zyklen verhindern. Sollte durch administrative Aufgaben ein Zyklus innerhalb der Organisationsstruktur entstehen, hat dies fatale Folgen für das angebundene fachliche Datenmodell. Die in Kapitel 4.4.2 erstellte View *EMPLOYEE_OU* arbeitet stark rekursiv. Da sie bei jedem Zugriff auf geschützte Tabelleninhalte aufgerufen wird und das zentrale Element der Zugriffsverwaltung darstellt, würde ein Defekt dieser View durch Zyklen in der zugrundeliegenden Organisationsstruktur zum Stillstand des gesamten Modells führen. Es muss also bei allen drei Fällen *INSERT*, *UPDATE* und *DELETE* auf der Tabelle *OrganizationUnit* sichergestellt werden, dass kein Zyklus entstehen kann.

Eine Überprüfung bei *DELETE* ist an dieser Stelle nicht nötig, denn bei dem Löschen von Datensätzen kann kein Zyklus entstehen. Somit muss dieser Fall nicht näher betrachtet werden. Natürlich kann eine Inkonsistenz innerhalb der Hierarchie durch das Löschen von Organisationseinheiten entstehen, diese äußern sich jedoch nicht in einem Zyklus. Bei dem Löschen von Organisationseinheiten kann es passieren, dass sich ein Teil-Abschnitt der Hierarchie löst und somit zwei Hierarchien entstehen, die keine direkte oder transitive Verbindung über ihre Parent-Relation mehr untereinander besitzen. Die Auswirkungen sind, dass bestimmte Datensätze der neu gebildeten Hierarchie nicht mehr für Mitarbeiter der ursprünglichen Hierarchie sichtbar sind. Diese versehentliche Spaltung der Hierarchie soll durch die Fremdschlüssel-Beziehung auf der Spalte *PARENT* der Tabelle *OrganizationUnit* verhindert werden. Diese Konsistenzbedingung erlaubt das Löschen von Organisationseinheiten aus der Tabelle nur dann, wenn keine andere untergeordnete Organisationseinheit mehr über *PARENT* auf diese verweist.

Bei dem Einfügen von Datensätzen mittels *INSERT* kann nur dann ein Zyklus entstehen, wenn der neue Datensatz auf sich selbst als übergeordnetes Element verweist. Das bedeutet, dass sich bei dem Einfügen des Datensatzes in der Spalte *PARENT* der gleiche Wert wie in der Primärschlüssel-Spalte *ID* befindet.

Diese reflexive Relation lässt sich mit einer weiteren Konsistenzbedingung unterbinden, die vorgibt, dass der Primärschlüssel *ID* sowie *PARENT* unterschiedliche Werte beinhalten müssen:

```
ALTER TABLE `OrganizationUnit`
ADD CONSTRAINT OU_CHECK_ID_PARENT CHECK (ID <> PARENT);
```

Tabelle 12: MariaDB – Erstellung des Check-Constraint

Bei der Nutzung von MariaDB schlägt die Erstellung dieser Konsistenzbedingung jedoch fehl, da auf der Spalte *ID* die Funktion *AUTO_INCREMENT* liegt. Dieses Problem lässt sich durch einen zusätzlichen Trigger lösen, der auf das Einfügen neuer Werte reagiert und die Prüfung der Werte an dieser Stelle vornimmt, siehe dazu Anhang 23. So kann dieser Trigger bereits vor dem eigentlichen Einfügen von Datensätzen eine Fehlermeldung ausgeben und den Vorgang damit abbrechen. Innerhalb der Oracle Database sowie dem SQL-Server ließ sich dies in Tabelle 12 dargestellte Konsistenzbedingung jedoch ohne Hindernisse anwenden.

Bei einer Veränderung von Datensätzen innerhalb der Spalte *PARENT* unter Verwendung eines *UPDATE* Statements muss vor dem Durchführen der Veränderung geprüft werden, ob sich durch die neue zu erwartende Hierarchiestruktur ein Zyklus ergeben würde. Sollte ein potentieller Zyklus erkannt werden, so muss das Statement mit einer Fehlermeldung abgebrochen werden. Das lässt sich jedoch aufgrund der Komplexität leider nicht mittels Check-Constraint, wie am Beispiel in der oben gezeigten Tabelle, realisieren.

Als einzige Lösung hierfür hat sich die Nutzung eines Triggers herausgestellt, welcher mögliche Zyklen vor oder nach der Änderung der Daten erkennt und eine entsprechende Fehlermeldung erzeugt, was zum Abbruch des SQL-Statements führt. Dazu wird mittels rekursiver *CTE* ermittelt, ob sich die neue übergeordnete Organisationseinheit bereits unterhalb der zu ändernden Organisationseinheit befindet, was einen Zyklus bedeuten würde.

Dieses Vorgehen funktioniert aus dem Grund, da ein Trigger innerhalb der Transaktion des Benutzers läuft und somit Bestandteil der Transaktion ist. Eine Fehlermeldung innerhalb des Triggers sorgt somit dafür, dass die Änderungen innerhalb der Transaktion des Benutzers abgebrochen und zurückgerollt werden.

```
CREATE OR REPLACE TRIGGER `OrganizationUnit_Update_Trg`
BEFORE UPDATE ON `OrganizationUnit`
FOR EACH ROW
BEGIN
    -- Liste alle OU-IDs dieser und darunter liegenden OUs
    WITH RECURSIVE cte (ID) AS (
        SELECT org.ID AS "ID" FROM `OrganizationUnit` org
        WHERE org.ID = OLD.ID
        UNION ALL
        SELECT org.ID AS "ID" FROM `OrganizationUnit` org
        INNER JOIN cte ON org.PARENT = cte.ID
    )
    SELECT COUNT (*) INTO @c FROM cte WHERE cte.ID = NEW.PARENT;
    IF @c > 0 THEN
        SIGNAL SQLSTATE '20667'
        SET MYSQL_ERRNO=20667,
        MESSAGE_TEXT='Zyklus erkannt!';
    END IF;
END;
```

Tabelle 13: MariaDB – Trigger zur Vermeidung von Zyklen (korrekt)

An dieser Stelle wird darauf hingewiesen, dass der in Tabelle 13 dargestellte *UPDATE*-Trigger erst nach einem Update auf die Version 10.4.9 von MariaDB korrekt implementiert werden konnte, da sich ein bereits gemeldeter Softwarefehler [57] unter der Kennung MDEV-20730 in der hier eingesetzten Version 10.4.6 befindet, der erst ab der Version 10.4.9 behoben wurde. Der hier dargestellte Trigger lässt sich bei nahezu identischer Funktionsweise auch auf die anderen beiden Datenbanksysteme übertragen, siehe dazu Anhang 3 und 16.

Bei der Überführung des Triggers in die Syntax des SQL-Servers ist eine Besonderheit des Systems von Microsoft aufgefallen. Der SQL-Server unterstützt einerseits keine *BEFORE*-Trigger. Das führt dazu, dass der implementierte Trigger innerhalb der Transaktion des Benutzers nur auf einen bereits entstandenen Zyklus reagieren kann. Dazu wird die gleiche *CTE* wie bei den übrigen Datenbanksystemen genutzt, doch dieser rekursive Ausdruck läuft innerhalb des SQL-Servers in eine unendliche Rekursion. Das System bricht diese Rekursion nach einer bestimmten erreichten Tiefe mit dem Fehlercode 530 ab, was vom Trigger aufgefangen und mittels eigener Fehlermeldung an den Benutzer weitergeleitet wird. Außerdem unterstützt er keine *FOR-EACH-ROW* Trigger. Aus diesem Grund wurde für den SQL-Server ein speziell angepasster Trigger konstruiert.

Die Datenbanksysteme MariaDB sowie Oracle Database unterstützen *FOR-EACH-ROW* Trigger, was sich jedoch nur bei Nutzung von MariaDB als Vorteil herausgestellt hat. Leider musste auch bei der Oracle Database auf die eingebaute Erkennung von Rekursionen zurückgegriffen werden, wie bei dem SQL-Server zuvor. Weiterführende Informationen zur Rekursionserkennung finden sich nachfolgend in Kapitel 5.3.3.

Denn auch bei der Oracle Database gab es Besonderheiten, die bei dem Entwickeln des Triggers zur Vermeidung von Zyklen aufgefallen sind.

Der ursprüngliche Gedanke war, wie bei MariaDB, einen Trigger anzulegen der den Datenbestand zeilenweise auf potentielle Zyklen prüft. Siehe folgend:

```
CREATE OR REPLACE TRIGGER OU_UPDATE_TRG
BEFORE UPDATE OF "PARENT" ON ORGANIZATIONUNIT
FOR EACH ROW
DECLARE
  PRAGMA AUTONOMOUS_TRANSACTION; -- Wichtig wegen "ORA-04091"
  c NUMBER;
BEGIN
  -- Liste alle OU-IDs dieser und darunter liegenden OUs
  WITH recDat (ID) AS (
    SELECT org.ID AS "ID" FROM ORGANIZATIONUNIT org
    WHERE org.ID = :old.ID
    UNION ALL
    SELECT org.ID AS "ID" FROM ORGANIZATIONUNIT org
    INNER JOIN recDat ON org.PARENT = recDat.ID
  )
  SELECT COUNT(*) INTO c FROM recDat WHERE recDat.ID = :new.PARENT;
  IF c > 0 THEN
    RAISE_APPLICATION_ERROR(-20667, 'Potentieller Zyklus erkannt!');
  END IF;
END;
```

Tabelle 14: Oracle – Trigger zur Vermeidung von Zyklen 1 (fehlerhaft)

Der Gedanke war zu Beginn gut, doch leider funktioniert dieses Konstrukt innerhalb der Oracle Database nur bei autonomen Anweisungen oder Operationen, die einen bereits vorhandenen Datenbestand berücksichtigen. Wenn jedoch der Trigger innerhalb einer Transaktion verwendet wird, in der auf Veränderungen der darin erzeugten Datensätze reagier werden soll, scheitert er. Der Grund dafür ist einfach:

Durch den Zusatz *AUTONOMOUS_TRANSACTION*, der leider zwingend notwendig war, läuft der Trigger in einer eigenen Transaktion innerhalb der äußeren Transaktion. Diese innere Transaktion sieht die Datensätze der äußeren Transaktion nicht, weil diese Daten noch nicht mittels *COMMIT* festgeschrieben wurden. Ein Zyklus entsteht. Folgend ein Beispiel dieses Verhaltens unter Verwendung des in Tabelle 14 definierten Triggers:

```
INSERT INTO ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (1, NULL, 'OU_T1');
INSERT INTO ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (2, 1,    'OU_T2');
UPDATE ORGANIZATIONUNIT SET PARENT = 2 WHERE ID = 1;
> ORA-20667: Potentieller Zyklus erkannt!
> ORA-06512: in "MT_ORG_ADMIN.OU_UPDATE_TRG", Zeile 15
> ORA-04088: Fehler bei der Ausführung von Trigger 'MT_ORG_ADMIN.OU_UPDATE_TRG'

SET TRANSACTION READ WRITE;
    INSERT INTO ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (3, NULL, 'OU_T1');
    INSERT INTO ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (4, 3,    'OU_T2');
    UPDATE ORGANIZATIONUNIT SET PARENT = 4 WHERE ID = 3;
COMMIT;
> Commit abgeschlossen.
```

Tabelle 15: Oracle – Beispiel einer fehlerhaften Transaktion

Der Grund für dieses Inkonsistente Verhalten liegt darin, dass die innere Transaktion des Triggers die neu erzeugten Datensätze der äußeren Transaktion aufgrund der Kapselung von Transaktionen nicht sehen kann. Die Erkennung von Zyklen ist hier somit nicht gegeben. Auch ein Umbau des Triggers, so dass dieser nicht mehr innerhalb seiner eigenen Transaktion läuft, hatte leider keine Verbesserung der Situation gebracht.

```
CREATE OR REPLACE TRIGGER OU_UPDATE_TRG
AFTER UPDATE OF "PARENT" ON ORGANIZATIONUNIT
DECLARE
  c NUMBER;
BEGIN
  -- Liste alle OU-IDs und darunter liegenden OUs
  WITH recDat (ID) AS (
    SELECT org.ID AS "ID" FROM ORGANIZATIONUNIT org
    WHERE org.PARENT IS NULL -- Diese Zeile muss entfernt werden!
    UNION ALL
    SELECT org.ID AS "ID" FROM ORGANIZATIONUNIT org
    INNER JOIN recDat ON org.PARENT = recDat.ID
  )
  SELECT COUNT(*) INTO c FROM recDat WHERE recDat.ID = :new.PARENT;
EXCEPTION
  WHEN OTHERS THEN
    RAISE_APPLICATION_ERROR(-20667, 'Potentieller Zyklus erkannt!');
END;
```

Tabelle 16: Oracle – Trigger zur Vermeidung von Zyklen 2 (fehlerhaft)

Das oben aufgezeigte Beispiel, das Erzeugen und Ändern neuer Organisationseinheiten innerhalb einer Transaktion, führte noch immer zu Zyklen. Doch dieses Mal liegt die Ursache hierfür nicht in der fehlenden Sichtbarkeit der Datensätze, da der Trigger nun im Kontext der eigentlichen Transaktion läuft und die Datensätze sieht.

Das Problem war die Annahme, dass es ein oberstes Element innerhalb der Hierarchie gibt, welches als *PARENT* den Wert *NULL* besitzt. Im hier konstruierten Beispiel startet der Trigger aufgrund der *AFTER*-Bedingung jedoch erst, wenn das oberste Element kein *NULL* mehr beinhaltet. Der Zyklus wird somit für den Trigger unsichtbar, da er ein geschlossener Zyklus ohne oberstes Element ist.

Wenn nun die Zeile „*WHERE org.PARENT IS NULL*“ entfernt wird, funktioniert der Trigger korrekt wie erwartet. In seiner Funktionsweise ist dieser Trigger identisch mit dem Trigger des SQL-Servers. Der zuvor angesprochene Vorteil, der sich ursprünglich durch die Nutzung von *FOR-EACH-ROW* Triggern in Kombination mit *BEFORE* ergeben hat, ist für die Oracle Database leider dahin. Wir sind auch hier auf die interne Rekursionserkennung der Datenbank angewiesen. Die korrekte Implementierung für die Oracle Database befindet sich in Anhang 3.

Interessanterweise muss der Trigger für MariaDB nicht in einer autonomen Transaktion laufen, wie bei der Oracle Database. Der hier implementierte Trigger funktioniert korrekt, unabhängig davon ob er innerhalb einer Transaktion ausgeführt wird oder nicht.

4.5 Aufbau – Fachliche Daten

Die Entwicklung einer fachlichen Datenbankstruktur steht nicht im Fokus dieser Arbeit. Ein solches Modell zur Verfügung zu haben ist jedoch essenziell wichtig für den Aufbau des Assoziations-Schemas. Aus diesem Grund wird an dieser Stelle ein vergleichsweise einfaches Datenmodell skizziert, welches nachfolgend als Beispiel für den grundsätzlichen Aufbau des Assoziations-Schemas genutzt werden soll.

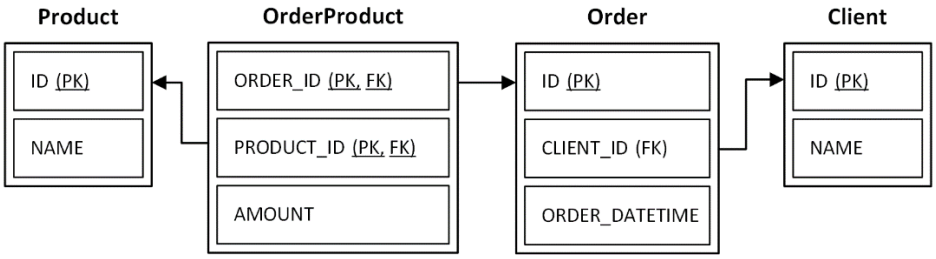

Bild 15: Beispiel – Fachliches Datenbankmodell

Wie im Bild 15 erkennbar, beinhaltet das Beispielmodell eine Kunden-, eine Bestellung- sowie eine Produkt-Tabelle. Die Tabelle *OrderProduct* dient als Zwischentabelle zwischen der Bestellung und dem Produkt.

```
-- Tabelle Product
CREATE TABLE `product` (
     `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
     `NAME` VARCHAR (128) NOT NULL DEFAULT ''
);
-- Tabelle Client
CREATE TABLE `client` (
     `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
     `NAME` VARCHAR (128) NOT NULL DEFAULT ''
);
-- Tabelle Order
CREATE TABLE `order` (
     `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
     `CLIENT_ID` INT NOT NULL,
     `ORDER_DATETIME` DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP()
);
ALTER TABLE `order` ADD FOREIGN KEY(`CLIENT_ID`) REFERENCES `client`(`ID`)
ON DELETE RESTRICT ON UPDATE RESTRICT;
-- Tabelle OrderProduct
CREATE TABLE `orderproduct` (
     `ORDER_ID` INT NOT NULL,
     `PRODUCT_ID` INT NOT NULL,
     `AMOUNT` INT NOT NULL,
     PRIMARY KEY (`ORDER_ID`, `PRODUCT_ID`)
);
ALTER TABLE `orderproduct` ADD FOREIGN KEY(`ORDER_ID`) REFERENCES `order`(`ID`)
ON DELETE RESTRICT ON UPDATE RESTRICT;
ALTER TABLE `orderproduct` ADD FOREIGN KEY(`PRODUCT_ID`) REFERENCES `product`(`ID`)
ON DELETE RESTRICT ON UPDATE RESTRICT;
```

Tabelle 17: MariaDB – Erstellung des Fachlichen Datenbankmodells

Auf dem nun erstellten Datenmodell kann das Assoziations-Schema aufgebaut werden.

4.6 Aufbau – Assoziation

Das Schema *Assoziation* ist eines der zentralen Kernelemente dieser Arbeit und soll daher möglichst ausführlich beschrieben werden. Wie bereits in Kapitel 4.2 beschrieben, wird in diesem Schema die Urheberschaft jedes fachlichen Datensatzes als Verknüpfung zwischen dem eigentlichen Datensatz und seiner urhebenden Organisationseinheit abgelegt. Daher ist der Aufbau dieses Schemas stark abhängig von der zugrundeliegenden fachlichen Datenbankstruktur, was die vorherige Konzeption des Modells in Kapitel 4.5 erklärt. Außerdem muss zuvor die Entscheidung getroffen werden, welche Tabellen im fachlichen Datenbankmodell mittels Zugriffsschicht geschützt werden sollen. Aus diesem Grund werden nachfolgend exemplarisch unterschiedliche Annahmen auf Basis des Modells in Bild 15 und Tabelle 17 getroffen. Entsprechend dieser Annahmen wird das Schema *Assoziation* modelliert.

Um an dieser Stelle nicht wahllos alle Tabellen oder möglicherweise die falschen Tabellen auszuwählen, sollte man sich die Frage stellen, an welcher Stelle im übergeordneten Programm ein möglicher Einstiegspunkt für den Abruf von Datensätzen existiert. Es gilt auch zu berücksichtigen, dass jede Tabelle die mittels Zugriffsschicht abgesichert ist, bei ihrem Aufruf eine Prüfung der angeforderten Datensätze mittels rekursiver Abfrage der Berechtigungen nach sich zieht. Dies kann sich negativ auf die Performance der Datenbank auswirken und zu einer verlangsamten Reaktionszeit führen. Somit gilt auch hier das Prinzip: So viel wie nötig, doch so wenig wie möglich.

Hier soll darauf hingewiesen werden, dass für das weitere Vorgehen in diesem Kapitel auf die Syntax der Oracle Database in PL/SQL zurückgegriffen werden muss. Die Gründe dafür sind technischer Natur und werden später in Kapitel 4.6.3 erläutert.

4.6.1 Einstiegspunkte

Unter Einstiegspunkten werden hier Tabellen verstanden, welche Datensätze beinhalten, die als Einstieg für einen fachlichen Workflow des darüber liegenden Programms dienen. So kann es für ein Programm zur Verwaltung von Kunden ausreichend sein, wenn ausschließlich die Tabelle *Client* abgesichert wird. Sollte jedoch im Rahmen der Kundenverwaltung auch über Bestellungen (Tabelle *Order*) nach Kunden gesucht werden können, beispielsweise wenn Kunden gesucht werden die in den letzten Tagen eine Bestellung getätigt haben, führt dies möglicherweise zu einem Problem. Auch kann das Öffnen eines (geschützten) Kundendatensatzes problematisch werden, wenn der aufrufende Mitarbeiter keinen Zugriff auf den Kundendatensatz hat. So besteht unter Nutzung einer weniger restriktiven *JOIN* Operation die Möglichkeit, alle Rechnungen der letzten 7 Tage angezeigt zu bekommen, auch wenn der Kundendatensatz scheinbar „fehlt“.

Korrekterweise ist hier hinzuzufügen, dass der Kundendatensatz nicht fehlt, sondern aufgrund von fehlenden Zuordnungen zur Organisationseinheiten des Mitarbeiters ausgeblendet wird. Mehr zu *JOIN* Operationen, siehe Kapitel 7.1.4.

Um das Problem zu verdeutlichen, bleiben wir bei dem oben beschriebenen Beispiel:

Eine fiktive Unternehmensstruktur gliedert sich in drei Bereiche mit je einem Mitarbeiter: Kundenverwaltung, Auftragsverwaltung und Produktverwaltung. Jeder dieser Bereiche schützt seine für ihn wesentliche Tabelle mittels Zugriffsschicht. Die Gründe hierfür sind eher nebensächlich, denn es geht um das konstruierte Beispiel.

Erstellt nun ein Mitarbeiter im Bereich Kundenverwaltung einen neuen Kundendatensatz, so ist dieser nur für ihn und seine Kollegen sichtbar, denn der neue Datensatz hat die Organisationseinheit Kundenverwaltung als Eigentümer. Möchte nun ein Mitarbeiter aus der Organisationseinheit Auftragsverwaltung telefonisch einen Auftrag für diesen Kunden entgegennehmen, ist dieser Kundendatensatz für den Mitarbeiter aus der Auftragsverwaltung nicht sichtbar. Die Ursache hierfür liegt darin, dass der Mitarbeiter selbst nicht über eine Sekundär-Assoziation in der Organisationseinheit Kundenverwaltung verfügt. Lohnt es sich daher, diese beiden Tabellen mit einer Zugriffsschicht zu schützen? Vermutlich nicht, wenn man dieses Beispiel betrachtet, denn die Datensätze Kunde und Auftrag hängen miteinander zusammen und sollten daher nicht horizontal voneinander getrennt werden. Wenn jedoch die Auftragsverwaltung als übergeordnete Organisationseinheit oberhalb der Kundenverwaltung angesiedelt ist, ist dieses Problem gelöst, ohne jeden Mitarbeiter mittels Sekundär-Assoziation mit der Kundenverwaltung zu verknüpfen.

Grundsätzlich lässt sich sagen, dass Stammdaten wie die Kundenverwaltung nur dann mittels Zugriffsschicht geschützt werden sollten, wenn die Organisationsstruktur eine solche Trennung zwingend vorsieht, weil bestimmte Bereiche des Unternehmens keinen Einblick in die Kundendaten eines anderen Bereiches haben sollen. Eine solche Trennung sollte jedoch wohl überlegt sein, denn sie führt ggf. zu einer doppelten Erfassung von Datensätzen, was eine nachgelagerte Zusammenführung der Organisationseinheiten schwierig gestaltet. Auch muss sichergestellt sein, dass das fachliche Datenbankmodell keine Einschränkungen in dieser Hinsicht mit sich bringt.

Sollte jedoch im Rahmen der Softwareentwicklung ein Dienst entwickelt werden, der eine strikte Trennung aller Datensätze vorsieht, bietet sich dieses hier entwickelte Modell fast uneingeschränkt an. Hier ist es natürlich wichtig, alle Tabellen mit einer solchen Zugriffsschicht zu versehen, um jede Art von Zugriff auf Daten anderer Kunden zu unterbinden.

4.6.2 Tabellen ohne Zugriffsschicht

Da die hier angelegten Tabellenstrukturen stark abhängig vom fachlichen Datenbankmodell sind, soll an dieser Stelle das in Kapitel 4.5 erstellte Modell als Vorlage dienen. Für jede Tabelle innerhalb des fachlichen Datenbankmodells wird eine View benötigt, welche den Zugriff auf die eigentliche Tabelle direkt weiterleitet oder eine entsprechende zur Sicherstellung der Zugriffsrechte beinhaltet. Außerdem wird, abhängig davon ob eine Tabelle mit einer Zugriffsschicht versehen werden soll oder nicht, eine Assoziations-Tabelle für jede zu schützende fachliche Tabelle benötigt. In dieser Tabelle werden die einzelnen Verknüpfungen zwischen jedem Datensatz und der jeweils urhebenden Organisationseinheit gespeichert.

Folgend ein Beispiel für die Tabelle *Product*, welche explizit nicht mittels Zugriffsschicht abgesichert werden soll. In diesem Beispiel sei angenommen, dass die Spalte *ID*, welche als Primärschlüssel der Tabelle dient, mittels Trigger und einer zugehörigen Sequence automatisch beim Einfügen eines neuen Datensatzes befüllt wird. Diese Annahme hat jedoch keinen Einfluss auf die grundsätzliche Funktionalität des Beispiels und soll nur aufgrund der Vollständigkeit erwähnt werden.

Alle hier gezeigten SQL-Statements werden im Schema *Assoziation* ausgeführt.

```
-- Erstellen der Tabelle A_PRODUCT
-- n.V.

-- Erstellen der View Product
CREATE VIEW PRODUCT
AS
    SELECT * FROM MT_DATA_ADMIN.PRODUCT;
```

Tabelle 18: Oracle – Beispiel, Abstraktion einer Tabelle ohne Zugriffsschicht

Es ist offensichtlich, dass diese hier erstellte View *Product* alle lesenden Anfragen direkt an die dahinter liegende Tabelle *Product* weiterleitet, welche sich im fachlichen Datenmodell befindet. Der Zugriff auf ein externes Schema ist hier erkennbar an dem Präfix *MT_DATA_ADMIN*, welcher vor dem Namen der Tabelle angeführt wird und ein Objekt außerhalb des eigenen Schemas adressiert. Der Punkt als Verbindungselement stellt, ähnlich wie bei gängigen objektorientierten Programmiersprachen wie Java oder C#, den Zugriff auf ein Attribut oder ein Element innerhalb eines anderen Namensraums dar. Einen solchen Präfix anzugeben ist zwingend notwendig, da innerhalb eines Schemas einer Datenbank die Namen der dort hinterlegten Strukturen eindeutig sein müssen, unabhängig von ihrem Typ. So wäre das Anlegen einer gleichnamigen View im Schema *Fachliche Daten* nicht möglich, da der Name *Product* bereits durch eine Tabelle belegt ist. Ein erster Test zeigt, dass der lesende Zugriff auf die dahinter liegende Tabelle über die gerade erstellte View ohne Schwierigkeiten möglich ist.

Da die zu erstellende Zugriffsschicht jedoch eine vollständige Abstraktion der dahinter liegenden Tabellen ermöglichen soll, müssen neben dem Lesen von Daten auch die Operationen Schreiben, Ändern und Löschen möglich sein. Ein weiterer Versuch zeigt, dass die folgenden Operationen ebenfalls von der erstellten View behandelt werden können. Folgend ein Beispiel:

```
INSERT INTO PRODUCT (NAME) VALUES ('Test_Product');
> 1 Zeile eingefügt.

UPDATE PRODUCT SET NAME = 'Test_Product_NEU' WHERE NAME = 'Test_Product';
> 1 Zeile aktualisiert.

DELETE FROM PRODUCT WHERE NAME = 'Test_Product_NEU';
> 1 Zeile gelöscht.
```

Tabelle 19: Oracle – Beispiel, Nutzung einer Tabelle ohne Zugriffsschicht

Zur Erinnerung sollte erwähnt werden, dass die in Tabelle 19 dargestellten SQL-Statements im Schema *Assoziation* ausgeführt werden und gegen die in Tabelle 18 definierte View laufen. Der Umstand, dass die hier dargestellten SQL-Statements funktionieren, obwohl sie gegen eine View laufen und die dahinter liegende Tabelle korrekt befüllen, ist auf die Arbeit des Datenbanktheoretikers Edgar Frank Codd zurückzuführen. Dieser definierte im Jahre 1985 ein Regelwerk für relationale Datenbanken, welche später als die *12 goldenen Regeln* oder auch *Codd's Rules* bekannt wurden. Eine dieser Regeln, speziell Nr. 6, beschreibt das Verhalten von Views bei nicht-lesendem Zugriff. Die Aussage von Codd ist:

„All views that are theoretically updatable are also updatable by the system." [52]

Damit sind natürlich die dahinter liegenden Tabellen und nicht die Views selbst gemeint. In einem weiteren Buch aus dem Jahre 1990 erweiterte er dieses Regelwerk von ursprünglich 12 auf mehrere hundert Elemente, um das Verhalten von relationalen Datenbanken weiter zu spezifizieren und zu präzisieren. Dem Verhalten von nicht-lesendem Zugriff auf Views widmete er hier ein ganzes Kapitel. [52] [53]

Nachdem nun eine View ohne Zugriffsschicht exemplarisch erstellt und erfolgreich auf ihre Funktionsweise hin überprüft wurde, muss nun der für diese Arbeit wesentlich wichtigere Fall erarbeitet werden: Eine View mit Zugriffsschicht

4.6.3 Tabellen mit Zugriffsschicht

Um exemplarisch die Erstellung der Zugriffsschicht für eine schützenswerte Tabelle zu demonstrieren, wählen wir die Tabelle *CLIENT* aus. Diese bietet sich an, da personenbezogene Daten nicht erst seit Inkrafttreten der Europäischen Datenschutz-Grundverordnung (DSGVO) besonderen Schutz benötigten.

```
-- Erstellen der Tabelle A_CLIENT
CREATE TABLE A_CLIENT (
    ID   INT NOT NULL PRIMARY KEY,
    OUID INT NOT NULL
);

-- Erstellen der View Product
CREATE VIEW CLIENT
AS
    -- Alle Daten lesen...
    SELECT c.*, e.EID FROM MT_DATA_ADMIN.CLIENT c
    -- ...die eine Verknüfung in der Association-Tabelle haben...
    INNER JOIN A_CLIENT a ON a.ID = c.ID
    -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
    INNER JOIN MT_ORG_ADMIN.EMPLOYEE_OU e ON e.OUID = a.OUID;
```

Tabelle 20: Oracle – Beispiel, Abstraktion einer Tabelle mit Zugriffsschicht

Für jede schützenswerte Tabelle im fachlichen Datenmodell wird eine separate Tabelle benötigt. Die hier erstellte neue Tabelle *A_CLIENT* dient dazu, die Verknüpfung zwischen einem fachlichen Datensatz in der *CLIENT*-Tabelle und der Organisationseinheit des urhebenden Mitarbeiters zu speichern. Der Aufbau des eingesetzten Primärschlüssels ist hier vollständig von der zu schützenden Tabelle zu übernehmen.

Was hier bei ersten Tests schnell auffällt, ist die Tatsache, dass die abstrahierende View *CLIENT* bei Aufruf drei statt der erwarteten zwei Spalten anzeigt. Die Spalte *EID* ist hinzugekommen und stammt nicht aus der zugrundeliegenden *CLIENT*-Tabelle. Sie enthält die *ID* jedes Mitarbeiters aus der Tabelle *EMPLOYEE*, der Zugriff auf den jeweiligen Datensatz hat. Ohne Filter auf dieser Spalte werden Datensätze mehrfach angezeigt, da üblicherweise mehrere Mitarbeiter einen Datensatz sehen dürfen. Bei der Verwendung der erzeugten View muss daher immer die neue Spalte *EID* mittels *WHERE* Bedingung auf die anfragende Mitarbeiter-ID gefiltert werden, unabhängig vom verwendeten Statement. Die Ursache hierfür ist in der Annahme begründet, dass auf einem Datenmodell, welches mittels Zugriffsschicht geschützt wird, ein Programm als Frontend aufsetzt, welches auf einem Application Server ausgeführt wird. In solchen Programmen ist es aus Performancegründen üblich, sogenannte Connection-Pools zu verwenden. Ein Nutzer verwendet also für eine beliebige Abfrage der Datenbank eine der verfügbaren Datenbankverbindungen. Er verfügt daher nicht über eine dedizierte, für ihn reservierte Verbindung.

Die Verwendung von Connection-Pools verhindert die Nutzung von Session-Variablen, die an die jeweils geöffnete Verbindung zur Datenbank gekoppelt sind und innerhalb der Datenbank jederzeit bei Abfragen genutzt werden können. Die Nutzung von Session-Variablen, um die Einschränkung der zwingend zu setzenden *WHERE* Bedingung zu beheben, wird als Ausblick in Kapitel 7.2.1 behandelt.

Die neu erstellte View *CLIENT* selbst greift auf drei unterschiedliche Objekte zu: Die zu schützende *CLIENT*-Tabelle, welche die fachlichen Daten beinhaltet, die *A_CLIENT*-Tabelle, welche die Referenz auf die urhebende Organisationseinheit enthält und die in Kapitel 4.4.2 erstellte View *EMPLOYEE_OU*.

Folgend ist der gesamtheitliche Zugriff der einzelnen Views und Tabellen, speziell für die erstellte View, in Anlehnung an Bild 12 dargestellt:

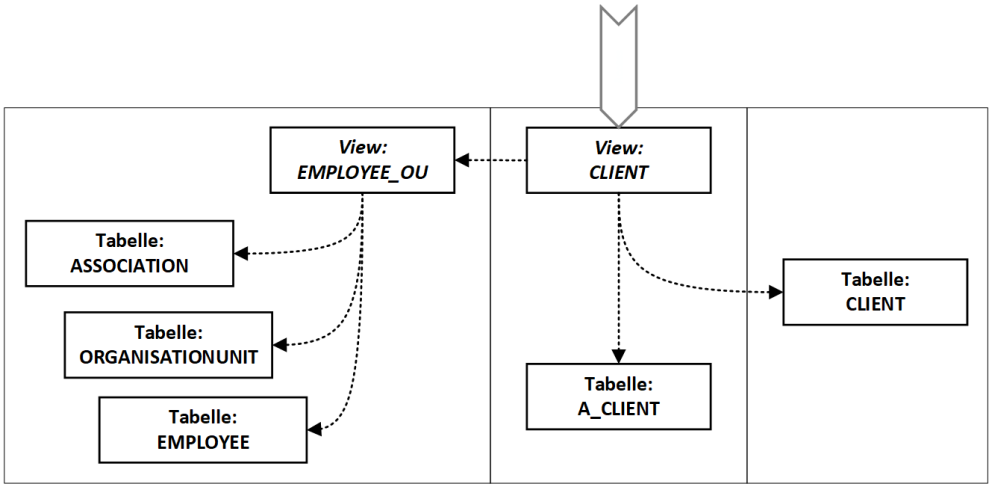

Bild 16: Zugriff auf CLIENT-View im Modell

Die Forderung, dass die wesentlichen SQL-Statement-Typen wie *SELECT*, *INSERT*, *UPDATE* und *DELETE* auch bei Views mit enthaltener Zugriffsschicht funktionieren, bleibt auch hier bestehen. Ein erster Test mit *SELECT* war erfolgreich, was wenig verwunderlich ist. Bei dem Einfügen eines ersten Datensatzes in die *CLIENT*-View unter Verwendung der Kennung des zuvor angelegten Test-Mitarbeiters *EMP_8* (ID: 8) erscheint jedoch die folgende Meldung:

```
INSERT INTO CLIENT (NAME, EID) VALUES ('TestKunde_01', 8);
> SQL-Fehler: ORA-01031: insufficient privileges
> *Cause:    An attempt was made to perform a database operation without
>            the necessary privileges.
> *Action:   Ask your database administrator or designated security
>            administrator to grant you the necessary privileges
```

Tabelle 21: Oracle – Beispiel - INSERT in View, ORA-01031

Diese Meldung mag auf den ersten Blick verwundern, lässt sich jedoch mit dem Berechtigungskonzept aus Kapitel 4.2 und den Erkenntnissen über Views aus dem Kapitel 4.6.2 erklären.

Die Datenbank versucht das *INSERT*-Statement an die dahinter liegenden Tabellen weiterzuleiten. Da der Benutzer *MT_ASS_ADMIN* jedoch keine Rechte zum Schreiben von Daten in diesen Tabellen hat, was auch so gemäß Kapitel 4.2 gewollt ist, meldet das Datenbanksystem die fehlenden Berechtigungen wie dargestellt. Nach dem temporären Zuweisen aller fehlenden Berechtigungen erscheint jedoch das eigentliche Problem:

```
INSERT INTO CLIENT (NAME, EID) VALUES ('TestKunde_01', 8);
> SQL-Fehler: ORA-01779: cannot modify a column which maps to a non key-preserved table
> *Cause:    An attempt was made to insert or update columns of a join view which
>            map to a non-key-preserved table.
> *Action:   Modify the underlying base tables directly.
```

Tabelle 22: Oracle – Beispiel - INSERT in View, ORA-01779

Hier zeigt sich der Fall, der bereits kurz in Kapitel 4.6.2 angesprochen wurde. Die Datenbank selbst kann in diesem Fall die schreibenden Vorgänge, welche an die View *CLIENT* gerichtet sind, nicht den darunterliegenden Tabellen zuordnen.

Die hier angebotene Lösung „*Modify the underlying base tables directly*“ ist aufgrund der Zielsetzung dieser Arbeit keine sinnvolle Möglichkeit, da der Anwender diese View zwingend als Abstraktionsschicht zum Filtern von Daten verwenden soll. Ein direkter Zugriff auf die dahinter liegenden Tabellen ist somit ausgeschlossen.

Erfreulicherweise bringen Oracle sowie Microsoft eine Technologie mit dem Namen „*INSTEAD-OF Trigger*“ mit. Ein normaler Trigger innerhalb einer Datenbank ist ein Objekt, welches auf Veränderungen von Tabelleneinträgen reagiert und anschließend weitere Aktivitäten anstößt. Ein solcher Trigger, welcher immer an eine Tabelle gebunden ist, wird zusätzlich zum auslösenden SQL-Statement, entweder vor oder nach der Ausführung des SQL-Statements, angestoßen und abgearbeitet.

Ein *INSTEAD-OF* Trigger hingegen reagiert ähnlich wie ein normaler Trigger auf SQL-Statements, welcher den Inhalt einer Tabelle verändert. Im Gegensatz zu normalen Triggern ersetzt er jedoch das ausführende SQL-Statement durch eine im Trigger hinterlegte Logik zur Behandlung des Statements. [54] [55]

Das Datenbanksystem *MariaDB* bietet leider keine solchen Trigger an, was auch der Grund für den in Kapitel 4.6 erwähnten Wechsel der Syntax auf Oracle darstellt. Außerdem verbietet MariaDB das Erstellen von Triggern, die mit Views verknüpft sind, unabhängig davon ob diese als normale Trigger vorgesehen sind oder nicht. [56]

Da das Lesen von Daten aus der erstellten View korrekt funktioniert, fehlen somit nur noch Trigger für die Operationen *INSERT*, *UPDATE* und *DELETE*.

Die nachfolgend dargestellten Trigger können sicherlich noch optimiert werden, sind an dieser Stelle jedoch bewusst sehr übersichtlich gehalten, um das grundlegende Prinzip zu verdeutlichen. Mögliche Erweiterungen und Optimierungen der hier dargestellten Trigger werden als Ausblick in Kapitel 7.2.3 beschrieben.

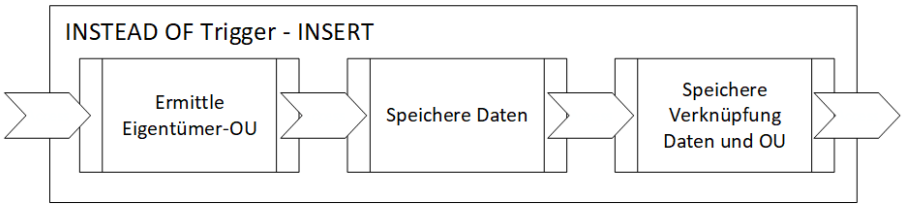

Bild 17: Ablauf des INSTEAD-OF Triggers - INSERT

Der entsprechende Trigger unter Nutzung der Oracle Syntax sieht wie folgt aus:

```
-- INSERT - Erstelle Instead-of Trigger
CREATE OR REPLACE TRIGGER CLIENT_INSERT_TRG
    INSTEAD OF INSERT ON CLIENT
DECLARE
    newID MT_DATA_ADMIN.CLIENT.ID%TYPE;
    oOUID MT_ORG_ADMIN.EMPLOYEE.OU%TYPE;
BEGIN
    -- Ermitteln der Eigentümer-OU des aufrufenden Mitarbeiters
    SELECT OU INTO oOUID FROM MT_ORG_ADMIN.EMPLOYEE WHERE ID = :new.EID;

    -- Speichern der Daten, speichern des erstellten PK in "newID"
    INSERT INTO MT_DATA_ADMIN.CLIENT (ID, NAME)
    VALUES (:new.ID, :new.NAME) RETURNING ID INTO newID;

    -- Speichern der Verknüpfung OU <--> Datensatz
    INSERT INTO MT_ASS_ADMIN.A_CLIENT VALUES (newID, oOUID);
END;
/
```

Tabelle 23: Oracle – Erstellen eines INSTEAD-OF Triggers - INSERT

Der hier dargestellte Trigger besteht aus drei wesentlichen Abschnitten: Der Ermittlung der Primär-OU des aufrufenden Mitarbeiters, der Speicherung des neuen Datensatzes sowie der Speicherung der Assoziation zwischen der Eigentümer-OU und dem Datensatz selbst. Besonders wichtig ist hier zu erwähnen, dass alle vorhandenen Spalten der fachlichen Tabelle händisch berücksichtigt und an das eigentliche INSERT-Statement innerhalb des Triggers weitergegeben werden müssen.

Eine separate Transaktion innerhalb dieses Triggers muss nicht erstellt werden, da der Trigger im Kontext der Benutzer-Transaktion ausgeführt wird. [55] Ein Fehler beim Einfügen der Daten innerhalb des Triggers führt zum Abbruch der ganzen Transaktion.

Der Aufbau des *UPDATE* Triggers hatte konzeptionell ursprünglich einen zusätzlichen Filter vorgesehen, damit kein Mitarbeiter Daten ändern kann, die ihm nicht zugeordnet sind und die dieser somit selbst nicht sehen kann.

Doch dieser zusätzliche Filter wurde verworfen. Interessanterweise war er hier überhaupt nicht notwendig, einen solchen Filter zu bauen, was sich im Nachhinein betrachtet als logisch herausstellte. Ein *UPDATE* Statement auf einer View, die einem Benutzer einen Datensatz nicht anzeigt, kann auch keinen *UPDATE* Trigger anstoßen, denn für den Benutzer existiert dieser Datensatz nicht. Aus diesem Grund sieht der nachfolgende Trigger ebenfalls recht übersichtlich aus.

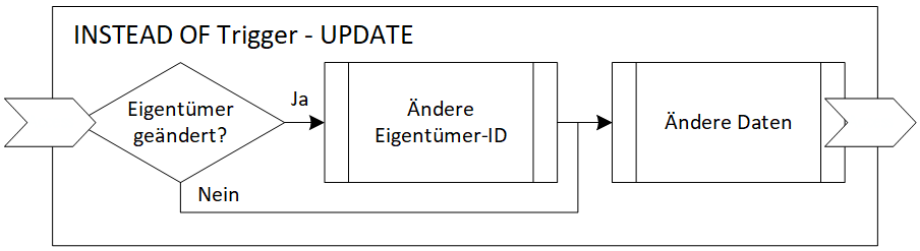

Bild 18: Ablauf des INSTEAD-OF Triggers - UPDATE

Auch hier wird nachfolgend der *INSTEAD-OF* Trigger unter Nutzung der Oracle Syntax angegeben:

```
-- UPDATE - Erstelle Instead-of-Trigger
CREATE OR REPLACE TRIGGER CLIENT_UPDATE_TRG
    INSTEAD OF UPDATE ON CLIENT
DECLARE
    NEW_OUID MT_ORG_ADMIN.EMPLOYEE.OU%TYPE;
BEGIN
    -- Eigentümer-OU ändern, wenn gewollt
    IF :new.EID <> :old.EID THEN
        SELECT OU INTO NEW_OUID FROM MT_ORG_ADMIN.EMPLOYEE WHERE ID = :new.EID;
        UPDATE MT_ASS_ADMIN.A_CLIENT SET OUID = NEW_OUID   WHERE ID = :new.ID;
    END IF;
    -- Datensatz ändern
    UPDATE MT_DATA_ADMIN.CLIENT SET NAME = :new.NAME WHERE ID = :new.ID;
END;
/
```

Tabelle 24: Oracle – Erstellen eines INSTEAD-OF Triggers - UPDATE

Dieser Trigger hat neben der reinen Aufgabe, die dahinter liegenden Daten zu ändern, auch die Funktion, den Eigentümer anzupassen. Auch hier findet, ebenso wie bei dem INSERT-Statement zuvor, eine händische Übertragung der einzelnen Spalten an das UPDATE-Statement statt.

Sollte während des *UPDATE*-Vorgangs eine abweichende Mitarbeiter-ID mit übergeben werden, dann führt dies dazu, dass die Assoziations-Tabelle *A_CLIENT* angepasst wird.

So kann ein Datensatz zwischen unterschiedlichen Organisationseinheiten verschoben werden. Sollte dies in einem Anwendungsszenario nicht gewollt sein, kann der Trigger entsprechend angepasst werden oder die darüber liegende Anwendung implementiert keinen Aufruf mit geänderter *EID*. Wichtig hierbei ist jedoch, dass es immer mindestens einen Mitarbeiter geben muss, der dieser Organisationseinheit zugeordnet ist und somit als neue *EID* mit angegeben werden kann. Dabei handelt es sich um eine konzeptionell bedingte Einschränkung dieses Modells.

Der *INSTEAD-OF* Trigger für das Löschen von Daten sieht sogar noch wesentlich kompakter aus, als die zuvor erstellten Trigger. In diesem müssen ausschließlich die gespeicherten Referenzen auf die fachlichen Daten gelöscht und anschließend das Löschen der fachlichen Daten selbst angestoßen werden.

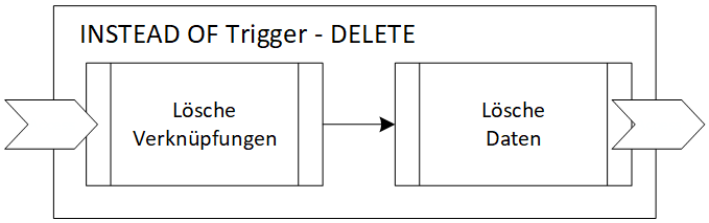

Bild 19: Ablauf des INSTEAD-OF Triggers - DELETE

Nachfolgend wird der Trigger für das Löschen der Daten in Oracle Syntax angegeben:

```
-- DELETE - Erstelle Instead-of-Trigger
CREATE OR REPLACE TRIGGER CLIENT_DELETE_TRG
    INSTEAD OF DELETE ON CLIENT
BEGIN
    DELETE FROM MT_ASS_ADMIN.A_CLIENT WHERE ID = :old.ID;
    DELETE FROM MT_DATA_ADMIN.CLIENT  WHERE ID = :old.ID;
END;
/
```

Tabelle 25: Oracle – Erstellen eines INSTEAD-OF Triggers - DELETE

Die Reihenfolge der hier enthaltenen Lösch-Befehle spielt keine Rolle, sollte der Trigger innerhalb einer Transaktion ausgeführt werden. Wenn dieser jedoch ohne Transaktion als autonome Aktion ausgeführt wird, was vermutlich der Regel entsprechen wird, muss unbedingt die Reihenfolge der Lösch-Operationen beachtet werden. Sollte die Reihenfolge verändert werden, schlägt das Löschen fehl.

Auf der Tabelle *A_CLIENT* liegt eine Fremdschlüssel-Konsistenzbedingung, welche auf das Vorhandensein von Datensätzen innerhalb der *CLIENT*-Tabelle angewiesen ist.

5 Überprüfung

Die Überprüfung des entwickelten Modells gliedert sich in vier Kategorien, wobei jede der Kategorien in die abschließende Bewertung mit einfließt. Die als grundlegend zu betrachtenden Anforderungen sind die, welche bei einer Datenbank die Basis-Operationen darstellen. Dabei sind *INSERT*, *UPDATE*, *DELETE* und *SELECT* gemeint. Unter den funktionalen Anforderungen werden die zu Beginn dieser Arbeit aufgestellten Anforderungen aus Kapitel 3.1 und 3.2 verstanden. Anschließend folgen sogenannte Nicht-Funktionale Anforderungen, in denen Besonderheiten der einzelnen Datenbanken berücksichtigt werden. Während der Entwicklung des Modells haben sich gewisse Eigenheiten der einzelnen Datenbanksysteme herauskristallisiert, die hier berücksichtigt werden sollen. Abschließend wird geprüft, wie sich die Performance des Modells, abhängig von der enthaltenen Datenmenge, verhält. Dabei findet aufgrund des Umfangs jedoch kein Stresstest mit simuliertem Mehrbenutzer-Zugriff statt.

Ein wichtiger Hinweis sei an dieser Stelle noch erwähnt: Da das Datenbanksystem MariaDB keine *INSTEAD-OF* Trigger unterstützt und somit das hier entwickelte Modell nicht auf MariaDB angewendet werden kann, wird dieses Datenbanksystem in diesem Kapitel nicht betrachtet.

Der Fokus dieses Kapitels liegt auf dem Microsoft SQL-Server und der Oracle Database.

5.1 Grundlegende Anforderungen

Um die grundlegenden Funktionen des Modells zu prüfen, wird zuerst eine fiktive Organisation mit je einem Mitarbeiter pro Organisationseinheit erstellt. Anschließend werden die oben genannten Basis-Operationen auf diesem Modell, sowohl mit einem als auch mehreren Datensätzen, angewendet und mit dem erwarteten Ergebnis verglichen. Damit soll sichergestellt werden, dass die grundlegenden Funktionen sichergestellt sind. Anbei die Erstellung der fiktiven Organisationsstruktur in Oracle-Syntax mit vorgegebenen Primär-Schlüsseln, wobei für die Vergabe eigener Primärschlüssel die in der Oracle Database hinterlegten Trigger zur Generierung neuer Primärschlüsseln zuvor deaktiviert werden müssen. Dies muss ebenso für den SQL-Server durchgeführt werden. Die Primärschlüssel sind in diesem Szenario vorgegeben, um die Mitarbeiter und Organisationseinheiten besser voneinander zu unterscheiden und das erwartete Ergebnis besser vergleichen zu können. Diese Testfälle werden in der gleichen Form auch für den SQL-Server durchgeführt. Für das Durchführen der nachfolgenden Testfälle ist der Benutzer *MT_ORG_USER* zu verwenden.

```
-- ---------------------------------------------
-- Ausführen als Benutzer MT_ORG_USER
-- ---------------------------------------------
-- Anlegen einer einfachen Organisationsstruktur mit 4 Ebenen
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (1, NULL, 'EUROPA');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (2, 1, 'Deutschland');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (3, 1, 'Frankreich');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (4, 2, 'Niedersachsen');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (5, 2, 'Hessen');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (6, 3, 'Centre');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (7, 3, 'Bretagne');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (8, 4, 'Hannover');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (9, 5, 'Frankfurt');

-- Erstellen von Mitarbeiter-Einträgen, je ein Mitarbeiter pro OU
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (11, 1, 'EMP_011');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (21, 2, 'EMP_021');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (31, 3, 'EMP_031');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (41, 4, 'EMP_041');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (51, 5, 'EMP_051');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (61, 6, 'EMP_061');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (71, 7, 'EMP_071');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (81, 8, 'EMP_081');
INSERT INTO MT_ORG_ADMIN.EMPLOYEE (ID, OU, NAME) VALUES (91, 9, 'EMP_091');

-- Der Mitarbeiter aus "Hannover" (EMP_081) unterstützt in "Frankfurt"
INSERT INTO MT_ORG_ADMIN.ASSOCIATION (EID, OUID) VALUES (81, 9);

-- Der Mitarbeiter aus "Frankreich" (EMP_031) erhält Zugriff auf "Hessen" u. "Hannover"
INSERT INTO MT_ORG_ADMIN.ASSOCIATION (EID, OUID) VALUES (31, 5);
INSERT INTO MT_ORG_ADMIN.ASSOCIATION (EID, OUID) VALUES (31, 8);

-- Der Mitarbeiter aus "Frankfurt" (EMP_091)
-- übernimmt die Vertretung des Vorgesetzten (für "Hessen")
INSERT INTO MT_ORG_ADMIN.ASSOCIATION (EID, OUID) VALUES (91, 5);
COMMIT;
```

Tabelle 26: Oracle – Erstellen von Testdaten für eine fiktive Organisation

Anbei findet sich das Beispielszenario, wie in Tabelle 26 angegeben, in grafischer Darstellungsform. Es dient dazu, die nachfolgenden Testfälle besser nachvollziehen zu können.

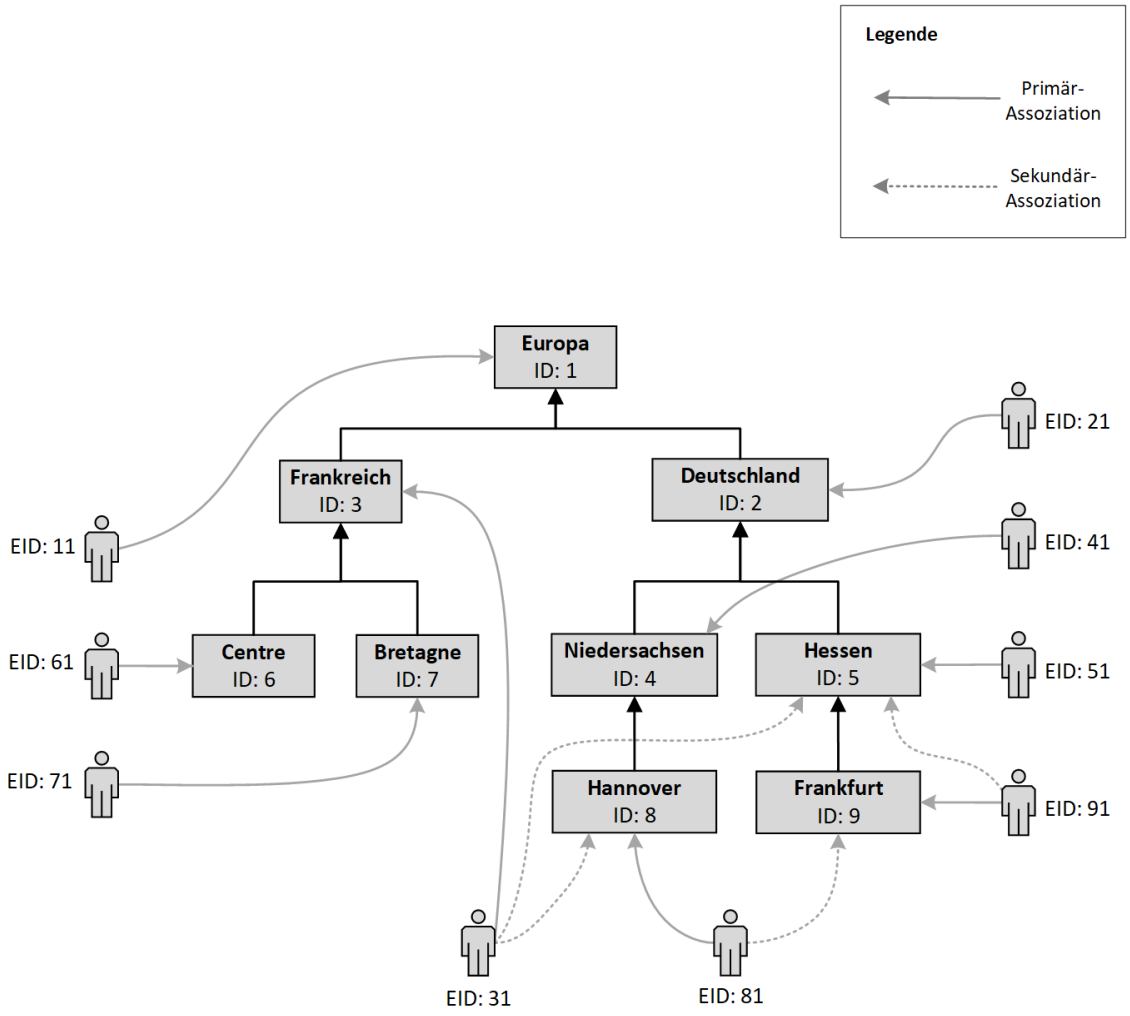

Bild 20: Grafisches Beispiel – Aufbau der fiktiven Organisation

5.1.1 Testfall 1 – Einfügen u. Lesen

In diesem Testfall erzeugt jeder Mitarbeiter der Organisation je einen Datensatz. Anschließend wird überprüft, welche Mitarbeiter diese erzeugten Datensätze sehen können. Dazu wird je Mitarbeiter ein *SELECT*-Statement mit Filter auf der jeweiligen *EID* des Mitarbeiters an die Datenbank gesendet.

```
INSERT INTO CLIENT (NAME, EID) VALUES ('C11', 11);
INSERT INTO CLIENT (NAME, EID) VALUES ('C21', 21);
INSERT INTO CLIENT (NAME, EID) VALUES ('C31', 31);
INSERT INTO CLIENT (NAME, EID) VALUES ('C41', 41);
INSERT INTO CLIENT (NAME, EID) VALUES ('C51', 51);
INSERT INTO CLIENT (NAME, EID) VALUES ('C61', 61);
INSERT INTO CLIENT (NAME, EID) VALUES ('C71', 71);
INSERT INTO CLIENT (NAME, EID) VALUES ('C81', 81);
INSERT INTO CLIENT (NAME, EID) VALUES ('C91', 91);
COMMIT;

SELECT * FROM CLIENT WHERE EID = :EID; -- Test je Mitarbeiter-ID (11, 21, ..., 91)
```

Tabelle 27: Testfall 1 – Oracle – Einfügen u. Lesen

Testergebnisse			
Mitarbeiter-ID	**Erwartung: Sichtbare Datensätze**	**Test: Oracle**	**Test: SQL-Server**
11 (Europa)	Alle	Erfolgreich	Erfolgreich
21 (Deutschland)	C21, C41, C51, C81, C91	Erfolgreich	Erfolgreich
31 (Frankreich)	C31, C51, C61, C71, C81, C91	Erfolgreich	Erfolgreich
41 (Niedersachsen)	C41, C81	Erfolgreich	Erfolgreich
51 (Hessen)	C51, C91	Erfolgreich	Erfolgreich
61 (Centre)	C61	Erfolgreich	Erfolgreich
71 (Bretagne)	C71	Erfolgreich	Erfolgreich
81 (Hannover)	C81, C91	Erfolgreich	Erfolgreich
91 (Frankfurt)	C51, C91	Erfolgreich	Erfolgreich

Tabelle 28: Testfall 1 – Ergebnisse

Während dieser Test ist unter Verwendung der *EID* 91 aufgefallen, dass der Datensatz C91 mehrfach im Ergebnis erschienen war, was sich durch eine Erweiterung der *EMPLOYEE_OU*-View um den Zusatz „*GROUP BY S, ID*“ beheben ließ. Anschließend waren alle Testfälle erfolgreich.

5.1.2 Testfall 2 – Ändern u. Löschen

Innerhalb dieses Testfalls soll überprüft werden, ob sich Datensätze ändern und löschen lassen. Dabei gilt besonders zu prüfen, ob die übrigen Datensätze von dieser Operation verschont bleiben und somit eine versehentliche oder mutwillige Manipulation der Datensätze verhindert werden kann. Auch soll überprüft werden, ob ein solche Manipulation für Datensätze möglich ist, die ein Mitarbeiter eigentlich gar nicht sehen kann. Zur Überprüfung der erwarteten Ergebnisse wird der Mitarbeiter mit der *EID* 11 verwendet, da er aufgrund seiner Position innerhalb der Hierarchie alle Datensätze sehen kann. Dieser Testfall baut auf dem vorherigen Testfall auf.

```
-- Test 1: C41 unverändert
UPDATE CLIENT SET NAME = 'NOT_ALLOWED' WHERE EID = 81 AND NAME = 'C41';
COMMIT;

-- Test 2: C51 -> C51_changed, C91 -> C91_changed
UPDATE CLIENT SET NAME = NAME || '_changed' WHERE EID = 51;
COMMIT;

-- Test 3: C41 nicht gelöscht
DELETE FROM CLIENT WHERE EID = 81 AND NAME = 'C41';
COMMIT;

-- Test 4: C51 gelöscht (und somit für EID 21, 31, 51, 91 nicht mehr sichtbar)
DELETE FROM CLIENT WHERE EID = 11 AND NAME = 'C51_changed';
COMMIT;
```

Tabelle 29: Testfall 2 – Oracle – Ändern u. Löschen

Testergebnisse			
Test	**Erwartung**	**Test: Oracle**	**Test: SQL-Server**
1, Ändern von C41	Keine Veränderung	Erfolgreich	Erfolgreich
2, Ändern von C51, C91	C51 u. C91 geändert	Erfolgreich	Erfolgreich
3, Löschen von C41	C41 nicht gelöscht	Erfolgreich	Erfolgreich
4, Löschen von C51	C51 gelöscht	Erfolgreich	Erfolgreich

Tabelle 30: Testfall 2 – Ergebnisse

5.1.3 Testfall 3 – Eigentümer-OU ändern

In diesem Testfall soll überprüft werden, wie sich das Modell verändert, wenn sich die Eigentümer-Organisationseinheit eines Datensatzes verändert. Eine solche Änderung hat einen direkten Einfluss darauf, welche Mitarbeiter einen Datensatz sehen können und welche nicht. Dazu wird exemplarisch der am Standort Frankfurt erhobene Datensatz *C91* innerhalb der Hierarchie verschoben. Vor dem Test war der Datensatz der Organisationseinheit Frankfurt zugeordnet und somit für alle Mitarbeiter sichtbar, die direkt oder indirekt dieser Organisationseinheit zugeordnet sind. Als Ergebnis wird verglichen, welche Mitarbeiter diesen Datensatz vor sowie nach jeder Veränderung sehen können. Dieser Testfall baut auf den vorherigen Testfällen auf.

```
-- Test 1: Verschieben von Frankfurt (OU: 9) nach Hannover (OU: 8)
UPDATE CLIENT SET EID = 81 WHERE EID = 81 AND NAME = 'C91_changed';
COMMIT;

-- Test 2: Verschieben von Hannover (OU: 8) nach Niedersachsen (OU: 4)
UPDATE CLIENT SET EID = 41 WHERE EID = 81 AND NAME = 'C91_changed';
COMMIT;

-- Test 3: Verschieben von Niedersachsen (OU: 4) nach Europa (OU: 1)
UPDATE CLIENT SET EID = 11 WHERE EID = 41 AND NAME = 'C91_changed';
COMMIT;
```

Tabelle 31: Testfall 3 – Oracle – Ändern des Eigentümers

Testergebnisse, Sichtbarkeit auf C91				
Test	**Sichtbar f. Mitarbeiter Vorher**	**Sichtbar f. Mitarbeiter Danach**	**Test: Oracle**	**Test: SQL-Server**
1.	11, 21, 31, 51, 81, 91	11, 21, 31, 41, 81	- *Fehlerhaft* -	Erfolgreich
2.	11, 21, 31, 41, 81	11, 21, 41	Erfolgreich	Erfolgreich
3.	11, 21, 41	11	Erfolgreich	Erfolgreich

Tabelle 32: Testfall 3 – Ergebnisse

Warum lief der erste Testfall bei der Oracle Database auf einen Fehler? Wenn man sich das *SQL*-Statement dazu anschaut, macht es auf den ersten Blick keinen Sinn, einen Wert zuzuweisen, den ein Datensatz bereits hat.

Jedoch dient die *EID* innerhalb der *WHERE*-Bedingung dazu, die Sichtbarkeit der Datensätze zu filtern, während *SET* dem Datensatz eine neue Eigentümer-Organisationseinheit zuweist, die sich indirekt aus der übergebenen *EID* des Mitarbeiters ergibt. Die Werte mögen gleich erscheinen, doch sie erfüllen unterschiedliche Aufgaben.

Um zu verstehen warum diese Operation jedoch bei dem SQL-Server korrekt umgesetzt worden ist, muss man verstehen, wie die Trigger, welche das Update-Statement behandeln, aufgebaut sind und wie die Datenbanken diese abarbeiten. Betrachtet man die beiden Update-Trigger im Vergleich, lässt sich die Ursache dafür erkennen:

SQL-Server

```
-- Eigentümer-OU ändern, wenn gewollt
IF (UPDATE(EID))
BEGIN
  -- Lese neue OU-ID
  SELECT @OUID = OU FROM [MT].[mt_org].EMPLOYEE WHERE ID = @EID;
  -- Setze neue OU-ID für diesen Datensatz
  UPDATE [MT].[mt_ass].A_CLIENT SET OUID = @OUID WHERE ID = @ID;
END;
```

Oracle

```
-- Eigentümer-OU ändern, wenn gewollt
IF :new.EID <> :old.EID THEN
  SELECT OU INTO NEW_OUID FROM MT_ORG_ADMIN.EMPLOYEE WHERE ID = :new.EID;
  UPDATE MT_ASS_ADMIN.A_CLIENT SET OUID = NEW_OUID   WHERE ID = :new.ID;
END IF;
```

Bild 21: Vergleich – Update-Trigger (Ausschnitt)

Der SQL-Server prüft über *UPDATE(EID)*, ob das Feld mittels *SET* gesetzt wurde. Dabei spielt es keine Rolle, ob sich der Wert darin verändert hat oder nicht. Die in der Tabelle *A_CLIENT* eingetragene Organisationseinheit war zuvor 9 und wird über den im Trigger enthaltenen Abschnitt nun auf 8 geändert, da die primär zugeordnete Organisationseinheit des Mitarbeiters mit der *EID* 81 die Nummer 8 besitzt.

Bei Oracle hingegen wird ein Vergleich zwischen dem alten und neuen Wert durchgeführt. Im Fall des fehlgeschlagenen Tests sind jedoch beide *EID* identisch. Es findet somit keine Neu-Zuordnung der Eigentümer-Organisationseinheit statt.

Dies kann man bei Oracle nun als Vorteil sowie Nachteil auslegen. Als Vorteil ist zu sehen, dass Mitarbeiter sich selbst und somit ihrer Organisationseinheit nicht pauschal alle sichtbaren Datensätze zuweisen können, die sie für diese sichtbar sind. Ein solches Verhalten würde ggf. dazu führen, dass untergeordnete Organisationseinheiten, welche einen Datensatz erhoben haben, den Zugriff auf diesen verlieren würden.

Als Nachteil ist ganz klar zu sehen, dass eine eigentlich valide Operation nicht durchgeführt werden kann. Die Zuweisung eines neuen Eigentümers muss somit zwingend durch andere Mitarbeiter geschehen. Das Modell verhält sich bei der Nutzung einer Oracle Database nicht durchgehend konsistent, da es diesen Fall anders behandelt als beispielsweise der SQL-Server. Dieser Sonderfall muss bei der Modellierung von Software, die dieses Modell nutzt, berücksichtigt werden.

5.1.4 Testfall 4 – Umstrukturierung

Hier soll nun überprüft werden, wie sich die Sichtbarkeit von Datensätzen durch eine Umstrukturierung der Organisationseinheit verhält. Dabei sei auf die Einschränkung aus Kapitel 7.1.8 verwiesen. Eine Veränderung der Organisationsstruktur wird über den Benutzer *MT_ORG_USER* vorgenommen, während der Benutzer *MT_ASS_USER* die Sichtbarkeit der Datensätze prüft. Für diesen Testfall wurde das Modell vollständig neu aufgebaut und mit neuen Datensätzen aus Kapitel 5.1 und 5.1.1 gefüllt, um die Ergebnisse leichter vergleichen zu können und die Einflüsse der vorherigen Testfälle zu eliminieren.

```
-- Test 1: Verschieben der Organisationseinheiten Bretagne u. Centre (OU: 6 u. 7)
--         von Frankreich (OU: 3) nach Deutschland (OU: 2)
UPDATE ORGANIZATIONUNIT SET PARENT = 2 WHERE PARENT = 3;
COMMIT;

-- Test 2: Änderungen rückgängig machen.
--         Bretagne u. Centre (OU 6: u. 7) sind nun wieder Frankreich (OU: 3) zugeordnet
UPDATE ORGANIZATIONUNIT SET PARENT = 3 WHERE ID IN (6, 7);
COMMIT;

-- Test 3: Abspaltung Frankreich (OU: 3) von Europa (OU: 1), Bildung von 2 Hierarchien
UPDATE ORGANIZATIONUNIT SET PARENT = NULL WHERE ID = 3;
COMMIT;
```

Tabelle 33: Testfall 4 – Oracle – Ändern der Organisationsstruktur

Die Prüfung der Sichtbarkeit von Datensätzen wird nur für einzelne Mitarbeiter durchgeführt, bei denen eine Veränderung zu erwarten ist oder sich keinerlei Veränderung ergeben sollte. Ausgewählt für die Validierung der Sichtbarkeit von Datensätzen wurden die Mitarbeiter mit den EIDs 11, 21 und 31. Es handelt sich um die Mitarbeiter der Organisationseinheiten Europa, Deutschland und Frankreich. Der Grund für diese Auswahl ist der folgende:

Nach dem Test 1 ist zu erwarten, dass Mitarbeiter 31 aus Frankreich den Zugriff auf die Datensätze C61 und C71 verliert. Diese sind hingegen jetzt für den Mitarbeiter 21 aus Deutschland sichtbar. Eine solche Veränderung hat gemäß Erwartung keinen Einfluss auf die Sichtbarkeit der Datensätze, die Mitarbeiter 11 aus Europa sieht. Im Test 2 soll dieser Vorgang wieder rückgängig gemacht werden.

Der dritte Test jedoch sollte sich auf die Sichtbarkeit der Datensätze von Mitarbeiter 11 aus Europa auswirken. Dieser soll den Zugriff auf die Datensätze C31, C61 und C71 verlieren, da Frankreich nun nicht mehr mit Europa über seine *PARENT*-Relation verknüpft ist. Hier ist außerdem zu erwarten, dass Mitarbeiter 31 aus Frankreich trotz aufgehobener Verknüpfung zu Europa, weiterhin Zugriff auf die Datensätze C51 und C91 aus Hessen/Frankfurt sowie C81 aus Hannover besitzt. Diese sind über eine Sekundär-Assoziation für ihn weiterhin sichtbar und nicht von der Aufspaltung der Hierarchie betroffen.

Nachfolgend sind die die Ergebnisse dieses Testfalls aufgeführt, inklusive einer daran anschließenden Delta-Betrachtung, welche die Veränderung der Sichtbarkeit zeigt.

Testergebnisse				
Test	**Mitarbeiter-ID**	**Erwartung: Sichtbare Datensätze**	**Test: Oracle**	**Test: SQL-Server**
1	11 (Europa)	Alle	Erfolgreich	Erfolgreich
	21 (Deutschland)	C21, C41, C51, C61, C71, C81, C91	Erfolgreich	Erfolgreich
	31 (Frankreich)	C31, C51, C81, C91	Erfolgreich	Erfolgreich
2	11 (Europa)	Alle	Erfolgreich	Erfolgreich
	21 (Deutschland)	C21, C41, C51, C81, C91	Erfolgreich	Erfolgreich
	31 (Frankreich)	C31, C51, C61, C71, C81, C91	Erfolgreich	Erfolgreich
3	11 (Europa)	C11, C21, C41, C51, C81, C91	Erfolgreich	Erfolgreich
	21 (Deutschland)	C21, C41, C51, C81, C91	Erfolgreich	Erfolgreich
	31 (Frankreich)	C31, C51, C61, C71, C81, C91	Erfolgreich	Erfolgreich

Tabelle 34: Testfall 4 – Ergebnisse

Test	**Mitarbeiter-ID**	**Sichtbarkeit erhalten auf…**	**Sichtbarkeit entzogen auf…**
1	11 (Europa)	-	-
	21 (Deutschland)	C61, C71	-
	31 (Frankreich)	-	C61, C71
2	11 (Europa)	-	-
	21 (Deutschland)	-	C61, C71
	31 (Frankreich)	C61, C71	-
3	11 (Europa)	-	C31, C61, C71
	21 (Deutschland)	-	-
	31 (Frankreich)	-	-

Tabelle 35: Testfall 4 – Ergebnisse – Delta-Betrachtung

5.1.5 Testfall 5 – Konsistenzbedingungen

Der Einhaltung der Konsistenzbedingungen bei der Veränderung der Hierarchie einer Organisation kommt, wie in Kapitel 3.2 angesprochen, eine besondere Bedeutung zu. Hier soll geprüft werden, ob die eingebauten Mechanismen aus Kapitel 4.4.3 korrekt funktionieren und die Konsistenz des Modells sicherstellen können.

Um die bisherigen Testdaten nicht weiter zu verändern, werden für diesen Testfall temporär andere Organisationseinheiten erstellt und im Anschluss wieder gelöscht.

Als kurzer Hinweis sei wie zuvor bereits erwähnt, dass auch für diesen Testfall unter Oracle der Trigger zum Vergeben von Primärschlüsseln deaktiviert werden muss. Dies gilt ebenso für den SQL-Server, wobei hier das Deaktivieren der automatischen Vergabe von Primärschlüsseln nur auf Session-Ebene durchgeführt werden kann und die *ALTER*-Berechtigung benötigt wird. Das heißt, unter dem SQL-Server muss der *MT_ORD_ADMIN* Benutzer verwendet werden, während unter der Oracle Database der Admin-Benutzer nur zum einmaligen Deaktivieren des Triggers verwendet werden kann und der Testfall selbst unter *MT_ORG_USER* durchgeführt werden kann.

```
-- Test 1: Erzeugen eines reflexiven Zyklus mittels INSERT
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (101, 101, 'OU_01');
COMMIT;

-- Test 2: Erzeugen eines reflexiven Zyklus mittels UPDATE
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (102, NULL, 'OU_02');
COMMIT;
UPDATE MT_ORG_ADMIN.ORGANIZATIONUNIT SET PARENT = 102 WHERE ID = 102;
COMMIT;

-- Test 3: Erzeugen eines direkten Zyklus mittels UPDATE
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (103, NULL, 'OU_03');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (104, 103, 'OU_04');
COMMIT;
UPDATE MT_ORG_ADMIN.ORGANIZATIONUNIT SET PARENT = 104 WHERE ID = 103;
COMMIT;

-- Test 4: Erzeugen eines transitiven Zyklus mittels UPDATE
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (105, NULL, 'OU_05');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (106, 105, 'OU_06');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (107, 106, 'OU_07');
COMMIT;
UPDATE MT_ORG_ADMIN.ORGANIZATIONUNIT SET PARENT = 107 WHERE ID = 105;
COMMIT;

-- Test 5: Spaltung der Hierarchie mittels DELETE
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (108, NULL, 'OU_08');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (109, 108, 'OU_09');
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (110, 109, 'OU_10');
COMMIT;
DELETE FROM MT_ORG_ADMIN.ORGANIZATIONUNIT WHERE ID = 109;
COMMIT;
```

Tabelle 36: Testfall 5 – Oracle – Einhaltung der Konsistenzbedingungen

Nachfolgend sind die Ergebnisse dieser einzelnen Test-Schritte aufgeführt.

Testergebnisse			
Test	**Erwartung**	**Test: Oracle**	**Test: SQL-Server**
1, Reflexiver Zyklus durch *INSERT*	Abbruch	Erfolgreich	Erfolgreich
2, Reflexiver Zyklus durch *UPDATE*	Abbruch	Erfolgreich	Erfolgreich
3, Direkter Zyklus durch *UPDATE*	Abbruch	Erfolgreich	Erfolgreich
4, Transitiver Zyklus durch *UPDATE*	Abbruch	Erfolgreich	Erfolgreich
5, Spaltung der Hierarchie durch *DELETE*	Abbruch	Erfolgreich	Erfolgreich

Tabelle 37: Testfall 5 – Ergebnisse

Folgend sind exemplarisch die Ergebnisse der Test-Reihe für Oracle dargestellt: Bei zwei der Tests wurde aufgrund der Constraints, die gesetzt wurden, ein Abbruch der Anweisung erzwungen. Die übrigen drei Tests sind aufgrund des gebauten Triggers abgebrochen, was dem erwarteten Verhalten entspricht.

```
-- Test 1: Erzeugen eines reflexiven Zyklus mittels INSERT
INSERT INTO MT_ORG_ADMIN.ORGANIZATIONUNIT (ID, PARENT, NAME) VALUES (101, 101, 'OU_01');
> ORA-02290: CHECK-Constraint (MT_ORG_ADMIN.OU_NO_REFLEXIVE) verletzt

-- Test 2: Erzeugen eines reflexiven Zyklus mittels UPDATE
[...]
UPDATE MT_ORG_ADMIN.ORGANIZATIONUNIT SET PARENT = 102 WHERE ID = 102;
> ORA-20667: Potentieller Zyklus erkannt!
> ORA-06512: in "MT_ORG_ADMIN.OU_UPDATE_TRG", Zeile 14
> ORA-04088: Fehler bei der Ausführung von Trigger 'MT_ORG_ADMIN.OU_UPDATE_TRG'

-- Test 3: Erzeugen eines direkten Zyklus mittels UPDATE
[...]
UPDATE MT_ORG_ADMIN.ORGANIZATIONUNIT SET PARENT = 104 WHERE ID = 103;
> ORA-20667: Potentieller Zyklus erkannt!
> ORA-06512: in "MT_ORG_ADMIN.OU_UPDATE_TRG", Zeile 14
> ORA-04088: Fehler bei der Ausführung von Trigger 'MT_ORG_ADMIN.OU_UPDATE_TRG'

-- Test 4: Erzeugen eines transitiven Zyklus mittels UPDATE
[...]
UPDATE MT_ORG_ADMIN.ORGANIZATIONUNIT SET PARENT = 107 WHERE ID = 105;
> ORA-20667: Potentieller Zyklus erkannt!
> ORA-06512: in "MT_ORG_ADMIN.OU_UPDATE_TRG", Zeile 14
> ORA-04088: Fehler bei der Ausführung von Trigger 'MT_ORG_ADMIN.OU_UPDATE_TRG'

-- Test 5: Spaltung der Hierarchie mittels DELETE
[...]
DELETE FROM MT_ORG_ADMIN.ORGANIZATIONUNIT WHERE ID = 109;
> ORA-02292: Integritäts-Constraint (MT_ORG_ADMIN.SYS_C0023391) verletzt
```

Tabelle 38: Testfall 5 – Oracle – Abbruch-Meldungen

5.2 Funktionale Anforderungen

5.2.1 Zugriffsschicht

In Kapitel 3.1 wurden wesentliche Anforderungen an die zu erstellende Zugriffsschicht aufgestellt, welche nun mit dem entstandenen Ergebnis verglichen werden.

Die erste davon ist die *Unveränderlichkeit der Fachlichkeit*, also die Unberührtheit der fachlichen Tabellenstruktur. Dies wurde erfolgreich über ein separates Assoziations-Schema mit abstrahierenden Views gelöst.

Die zweite Anforderung ist die *Urheberzuordnung*, welche die Entkopplung eines erhobenen fachlichen Datensatzes von einem Mitarbeiter vorsieht. Auch diese Anforderung wurde über die Zuordnung des Datensatzes zur Organisationseinheit sichergestellt.

Wie durch die Testfälle in Kapitel 5.1.1 gezeigt wurde, wird auch die dritte Anforderung, der sogenannte *Transitive Zugriff*, durch das erstellte Modell sichergestellt. Ein Mitarbeiter einer übergeordneten Organisationseinheit hat somit Zugriff auf Datensätze, die durch seine hierarchisch untergeordneten Mitarbeiter erhoben wurden.

Einzig die Anforderung der *Abstraktion* konnte nicht vollständig erfüllt werden. Diese sieht vor, dass SQL-Statements innerhalb einer Anwendung nicht angepasst werden müssen, wenn die darunter liegende Datenbank mittels Zugriffsschicht geschützt werden sollte. Es wurde eine weitere Spalte in die abstrahierenden Views mit aufgenommen, welche bei jedem SQL-Statement zwingend zu setzen ist. Ein weiterer Umstand der durchaus störend sein kann, ist die Tatsache, dass ausschließlich geschützte Tabellen nach außen über die View um eine separate Spalte erweitert wurden. Alle nicht geschützten Tabellen behalten nach außen hin ihre Struktur. Dieser Umstand ließe sich über Session-Variablen verhindern, siehe dazu Kapitel 7.2.1. Doch auch die Nutzung von Session-Variablen setzt zwingend mindestens einen Aufruf zum Setzen dieser Variable innerhalb der Anwendung voraus. Diese Variante funktioniert ebenfalls nicht, wenn einem Anwender keine dedizierte Datenbank-Session zur Verfügung steht, weil sich eine Anwendung aus einem Pool an Session bedient. Bei der Modellierung wurde ein Szenario angenommen, in dem von einem Session-Pool ausgegangen wurde. Somit muss die Identifikationsnummer eines Anwenders bei jedem Zugriff auf eine abstrahierende View mitgegeben werden. Außerdem muss eine Anwendung in Bezug auf die von ihr verarbeiteten Primärschlüssel angepasst werden, siehe dazu Kapitel 5.3.5.

Wie sich eine Anwendung unter Verwendung eines Objekt-Relation-Mappers wie *Hibernate* für Java oder des *Entity Frameworks* für .NET verhält, ist nicht Bestandteil dieser Arbeit und wurde daher nicht weiter untersucht.

5.2.2 Operativer Betrieb

Zusätzlich zu den Anforderungen an die Zugriffsschicht selbst wurden in Kapitel 3.2 weitere Anforderungen an das Modell hinsichtlich der Struktur einer Organisation gestellt. Diese sollen nachfolgend kurz wiederholt und ebenfalls mit dem erstellten Modell verglichen werden.

Die erste Anforderung, die Berücksichtigung der *Mitarbeiterfluktuation*, wurde durch das entstandene Modell abgebildet. Ein Mitarbeiter kann ohne Schwierigkeit von einer Organisationseinheit in eine andere verschoben werden. Die fachlichen Datensätze sowie die Zuordnung dieser Datensätze zu ihren urhebenden Organisationseinheiten bleiben hiervon unberührt. Sowohl die Aufnahme neuer Mitarbeiter als auch das Ausscheiden von Mitarbeitern kann vom Modell abgebildet und berücksichtigt werden.

Die *Multiple Zuordnung* von Mitarbeitern zu anderen Organisationseinheiten durch Projektarbeit oder temporäre Vertretungsbefugnis ist ebenfalls über das Modell sichergestellt. Hierzu wird neben der Primär-Assoziation auch eine Vielzahl von Sekundär-Assoziationen unterstützt. Jedoch sollte berücksichtigt werden, dass ein erhobener Datensatz, auch wenn er im Rahmen eines Projektes entsteht und somit eigentlich der Organisationseinheit des Projektes zugeordnet sein sollte, bei Erstellung erst einmal der Primär-Organisationseinheit des Mitarbeiters zugeordnet ist. Jedoch erlaubt das Modell die Veränderung des Eigentümers, siehe dazu Kapitel 5.1.3.

Mögliche *Veränderungsprozesse* werden innerhalb des Modells dadurch unterstützt, dass die Sichtbarkeit entlang der Hierarchie bei jedem Zugriff neu aufgebaut und geprüft wird. Das Verschieben einer Organisationseinheit im Rahmen einer Umstrukturierung hat somit sofort Einfluss auf die sichtbaren Datensätze der Mitarbeiter. Das fachliche Datenmodell bleibt von solchen Aktionen ebenso unberührt, wie deren Zuordnungen im Assoziations-Schema.

Die Anforderung an die *Unveränderlichkeit der Struktur*, bezogen auf das Modell zur Abbildung und Aufbau einer Hierarchie, wird dadurch sichergestellt, dass sämtliche Veränderungen durch Anpassung der enthaltenen Datensätze geschehen. Die Tabellenstrukturen bleiben hiervon unberührt.

Zur Sicherstellung der *Zyklenfreiheit* innerhalb der Hierarchie wurden Trigger und Constraints eingebaut, welche die Bildung von Zyklen proaktiv oder reaktiv verhindern sollen, siehe Kapitel 5.1.5. Durch die Beschränkung des Zugriffs über einen Benutzer mit limitierten Berechtigungen soll ein deaktivieren der Trigger und Constraints verhindert werden. Sollte es doch zu einem unerwarteten Zyklus kommen, greift glücklicherweise die Rekursionserkennung der jeweiligen Datenbank, auch wenn diese je nach System unterschiedlich arbeitet. Siehe dazu nachfolgend Kapitel 5.3.3.

5.3 Nicht-Funktionale Anforderungen

Während der Entwicklung des hier beschriebenen Modells sind einige Besonderheiten der einzelnen Datenbanksysteme aufgefallen, welche an dieser Stelle angesprochen werden sollen.

5.3.1 Verwendung von Triggern

Der Microsoft SQL-Server unterstützt gewisse Funktionen innerhalb von Triggern nicht. Darunter fällt beispielsweise die Definition, wann ein Trigger ausgeführt wird. Hier fehlt die Möglichkeit, einen Trigger als *BEFORE*-Trigger zu deklarieren, was sich speziell bei der Umsetzung der in Kapitel 4.4.3 beschriebenen Sicherheitsmaßnahmen als Problem herausstellt. So kann ein Trigger ausschließlich als *AFTER*-Trigger deklariert werden, was die Erkennung eines Zyklus innerhalb einer Hierarchie erst nach dessen Auftreten ermöglicht, da das darin enthaltene SQL-Statement in eine unendliche Rekursion läuft und durch die Datenbank terminiert. Auch bietet das System keine Möglichkeit, einen Trigger als *FOR-EACH-ROW* zu deklarieren, was zwangsläufig dazu führt, dass innerhalb jedes *INSTEAD-OF* Triggers mit einem Cursor gearbeitet werden muss, um einen solchen *FOR-EACH-ROW* Trigger zu simulieren und eine Verarbeitung jeder einzelnen Zeile zu ermöglichen. Dies kann ein möglicher Grund dafür sein, dass die Generierung von Testdaten im Vergleich zur Oracle Database fast die doppelte Zeit in Anspruch nimmt, was jedoch nicht weiter untersucht wurde. Das Datenbanksystem MariaDB unterstützt sowohl *FOR-EACH-ROW* als auch *BEFORE*-Trigger.

5.3.2 Schema-Zuweisung

Bei einer Oracle Database befindet sich jeder Benutzer automatisch in einem gleichnamigen Schema. Dieses Verhalten lässt sich in der Form leider nicht beeinflussen und führt bei dem hier entwickelten Modell zwangsläufig zu zwei Besonderheiten, die es zu beachten gilt.

Das aktive Schema des jeweiligen Benutzers muss, im Falle der Benutzer *MT_ASS_USER* und *MT_ORG_USER*, durch den Befehl „*SET CURRENT_SCHEMA <NAME>*“ auf das Schema des jeweiligen Admin-Benutzers umgebogen werden. Um diesen Vorgang zu automatisieren, kann auch ein sogenannter Logon-Trigger verwendet werden, welcher jedoch bei der Erweiterung der Datenbank um zusätzliche Schemata entsprechend angepasst werden muss. Ein solcher Logon-Trigger ist im Anhang 2 enthalten. Außerdem unterstützt die Oracle Database im Gegensatz zum SQL-Server keine Berechtigung auf Schema-Ebene. Sollte das fachliche Datenbankmodell erweitert werden, so ist darauf zu achten, dass die Berechtigungen für die neu hinzugekommenen Tabellen an die Benutzer des Assoziations-Schemas vergeben werden.

5.3.3 Rekursionserkennung

Innerhalb von Kapitel 4.4.3 wird durch einen Trigger versucht, die Bildung von Zyklen innerhalb einer Hierarchie durch das vorherige Prüfen der eingegebenen oder veränderten Daten zu verhindern. Doch wie verhalten sich die Datenbanksysteme, sollte es doch zu einer unerwarteten dauerhaften Rekursion kommen? Hier gestaltet sich das Verhalten hinsichtlich der Erkennung von unendlicher Rekursion durchaus unterschiedlich.

Die Oracle Database scheint eine unendliche Rekursion anhand der Zwischenergebnisse des rekursiven Aufrufs zu erkennen. Eine maximale Tiefe der Rekursion scheint es hier nach ersten Versuchen nicht zu geben.

So liefert das erste unten aufgeführte Statement bereits nach kürzester Zeit eine Fehlermeldung und terminiert, während das zweite Statement für 1.000.000 rekursive Aufrufe mehrere Sekunden lief, ohne den Vorgang abzubrechen.

```
WITH infiniteRecursion(n) AS (
    SELECT 'A' FROM DUAL
    UNION ALL
    SELECT n FROM infiniteRecursion
)
SELECT * FROM infiniteRecursion;

> SQL-Fehler: ORA-32044: Cycle bei der Ausführung der rekursiven WITH-Abfrage ermittelt
> *Cause:    A recursive WITH clause query produced a cycle and was stopped
>            in order to avoid an infinite loop.
> *Action:   Rewrite the recursive WITH query to stop the recursion or use
>            the CYCLE clause.

WITH infiniteRecursion(n) AS (
    SELECT 1 FROM DUAL
    UNION ALL
    SELECT 1 + inf.n FROM infiniteRecursion inf
    WHERE inf.n < 1000000
)
SELECT MAX(inf.n) FROM infiniteRecursion inf;

> MAX(INF.N)
> ----------
>    1000000
```

Tabelle 39: Oracle – Rekursionserkennung, ORA-32044

Der SQL-Server hingegen scheint die Tiefe der Rekursion während der Ausführung durch das Mitzählen der Aufrufe zu ermitteln und die Ausführung ab einer bestimmten Tiefe abzubrechen, da er selbst eine dauerhafte Rekursion vermutet.

Der Default-Wert für die Rekursionserkennung liegt bei 100, kann jedoch auch mittels *OPTION* auf einen Wert zwischen 0 und 32.767 angepasst werden, wobei 0 eine Deaktivierung darstellt, was in aller Deutlichkeit nicht zu empfehlen ist. [71]

Nachfolgend wird das Verhalten des SQL-Servers dargestellt:

```
WITH infiniteRecursion(dat) AS (
    SELECT 'A'
    UNION ALL
    SELECT dat FROM infiniteRecursion
)
SELECT * FROM infiniteRecursion
OPTION (MAXRECURSION 200);

> Meldung 530, Ebene 16, Status 1, Zeile 5
> Die Anweisung wurde beendet. Die maximale Rekursionstiefe 200 wurde vor
> Abschluss der Anweisung erreicht.
```

Tabelle 40: SQL-Server – Rekursionserkennung, Msg 530

Das Datenbanksystem MariaDB bildet hier leider die negative Ausnahme, da es selbst zwar eine Rekursionserkennung wie der SQL-Server mit sich bringt, diese jedoch auf einen so hohen Default-Wert von 4.294.967.295 eingestellt ist, dass man bei Ausführung von einer unendlichen Rekursion ohne Abbruch ausgehen muss. [74]

Eine Anpassung des Wertes wurde bereits unter der Nummer MDEV-17239 über den Bugtracker gefordert, ist zum jetzigen Zeitpunkt jedoch noch nicht umgesetzt. [75]

Der bei MariaDB voreingestellte Default-Wert lässt sich im Gegensatz zum SQL-Server auch nicht für einzelne SQL-Statements festlegen, sondern muss für jede Session oder vom Administrator global festgelegt werden. Diese Änderung geht jedoch verloren, sobald das Datenbanksystem neugestartet wird. Um einen dauerhaft geringeren Default-Wert zu setzen, muss die Konfigurationsdatei *my.conf* (Linux) oder *my.ini* (Windows) angepasst und um den Zusatz *max_recursive_iterations=<VALUE>* erweitert werden.

Anbei findet sich ein Beispiel zur Reduzierung des Default-Wertes für eine Session:

```
SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE
FROM   INFORMATION_SCHEMA.SYSTEM_VARIABLES
WHERE  VARIABLE_NAME = 'MAX_RECURSIVE_ITERATIONS';

> VARIABLE_NAME            | SESSION_VALUE | GLOBAL_VALUE |
> -------------------------------------------------------
> MAX_RECURSIVE_ITERATIONS |    4294967295 |   4294967295 |


SET MAX_RECURSIVE_ITERATIONS = 1000;

SELECT VARIABLE_NAME, SESSION_VALUE, GLOBAL_VALUE
FROM   INFORMATION_SCHEMA.SYSTEM_VARIABLES
WHERE  VARIABLE_NAME = 'MAX_RECURSIVE_ITERATIONS';

> VARIABLE_NAME            | SESSION_VALUE | GLOBAL_VALUE |
> -------------------------------------------------------
> MAX_RECURSIVE_ITERATIONS |          1000 |   4294967295 |
```

Tabelle 41: MariaDB – MAX_RECURSIVE_ITERATIONS

5.3.4 Kapselung über Views

Bei dem SQL-Server hat sich herausgestellt, dass es keine Möglichkeit gibt, die abstrahierenden Views im Assoziations-Schema mit den Rechten des Besitzers *MT_ASS_ADMIN* ausführen zu lassen. Dies hat zur Folge, dass *MT_ASS_USER* zwingend lesende Rechte auf dem Organisations-Schema benötigt, was aus datenschutztechnischer Sicht durchaus bedenklich ist. Dass er diese Rechte benötigt, liegt an der Nutzung der *EMPLOYEE_OU*-View, die in allen abstrahierenden Views enthalten ist und zum Filtern von Datensätzen dient.

Innerhalb einer Oracle Database werden alle Views, Prozeduren und Funktionen per Default mit den Berechtigungen des Eigentümers ausgeführt. So lässt sich eine Kapselung des dahinter liegenden Organisations-Schemas leicht umsetzen. Außerdem verhindert dies, dass der Benutzer *MT_ASS_USER* an den Views vorbei die abgelegten Assoziations-Tabellen modifiziert und so die Zuordnung zwischen Organisationseinheit und Datensatz manipuliert.

An dieser Stelle sei erwähnt, dass MariaDB ein solches Konzept ebenfalls mitbringt und konsequent umsetzt. Das Datenbanksystem erlaubt das Ausführen von Views im Kontext eines definierten Erstellers, siehe dazu Anhang 23 und 25. Dafür muss bei der Erstellung der View einerseits definiert werden, wer als Ersteller zu betrachten ist und zweitens, ob die View im Kontext des zuvor definierten Erstellers oder im Kontext des aktiven Benutzers ausgeführt werden soll. Anschließend lässt sich die Nutzung der zuvor definierten View mittels Grant an unterschiedliche Datenbankbenutzer weitergeben.

5.3.5 Primärschlüssel

Ein Primärschlüssel hat die Aufgabe, einen Datensatz eindeutig zu identifizieren. Um dies sicherzustellen, wird nicht selten die Generierung von eindeutigen Primärschlüsseln einem Automatismus der Datenbank überlassen. Bei dem SQL-Server werden solche Spalten, die den Primärschlüssel automatisch vergeben sollen, als *IDENTITY(1,1)* definiert. Die Oracle Database hingegen arbeitet mit einer Kombination aus Trigger und Sequence, welche zusammen das Setzen des Primärschlüssels ermöglichen.

Bei Oracle ist es so, dass sich der erzeugte Primärschlüssel mittels *RETURNING* ausgeben und in einer Variablen weiterverarbeiten lässt, was sich viele Softwareentwickler zunutze machen. Da sich unter Nutzung des hier entwickelten Modells das fachliche Datenmodell jedoch hinter einer View befindet und das Einfügen von neuen Datensätzen durch *INSTEAD-OF* Trigger behandelt wird, funktioniert eine Rückgabe des neu generierten Primärschlüssels nicht mehr. Dieses Verhalten lässt sich auch in der mittlerweile erschienenen Version 19c nicht verändern.

```
DECLARE
    newID CLIENT.ID%TYPE;
BEGIN
    INSERT INTO CLIENT (NAME, EID) VALUES ('Test', 11) RETURNING ID INTO newID;
END;
> SQL-Fehler: ORA-22816: unsupported feature with RETURNING clause
> *Cause:    RETURNING clause is currently not supported for object type
>            columns, LONG columns, remote tables, INSERT with subquery,
>            and INSTEAD OF Triggers.
> *Action:   Use separate select statement to get the values.
```

Tabelle 42: Oracle – RETURNING aus View, ORA-22816

Mögliche Lösungen für dieses Problem gibt es einige. Eine Variante ist, den Rückgabewert in einer Session-Variable zu speichern und durch den Benutzer nach Ausführen des Statements auslesen zu lassen. Eine weitere Alternative besteht darin, dem Benutzer *MT_ASS_USER* Lese-Rechte für die *Sequences* im fachlichen Datenmodell zu geben, so dass dieser den zuletzt vergebene Primärschlüssel auslesen kann. Jedoch ist dies ein Bruch der angestrebten Kapselung, denn ein Benutzer soll ausschließlich über das Assoziations-Schema mit dem dahinter liegenden fachlichen Datenmodell interagieren. Es besteht natürlich auch die Möglichkeit, die *Sequences* ausschließlich innerhalb des Assoziations-Schemas zu hinterlegen, auf die der Datenbankbenutzer Zugriff hat. Somit würden Primärschlüssel ausschließlich über Objekte das Assoziations-Schemas vergeben werden, was zwar keinen Bruch der Kapselung bedeutet, jedoch das fachliche Datenmodell zu gewissen Teilen verändern würde. Denkbar wäre auch ein Ablegen der neu erzeugten Primärschlüssel in einer Temporären Tabelle des Assoziations-Schemas, wo sie über eine View ausgelesen und anschließend nach dem Abruf wieder gelöscht werden könnten.

Der SQL-Server bringt einige Funktionen mit [89], um automatisch vergebene Primärschlüssel auszulesen. Zur Auswahl stehen hier *SCOPE_IDENTITY*, *@@IDENTITY* sowie *IDENT_CURRENT(xxx)*. Diese Funktionen unterscheiden sich jedoch in ihrem Wirkungsbereich, was ihre Nutzbarkeit stark einschränkt. So liefert *@@IDENTITY* zwar den letzten automatisch vergebenen Primärschlüssel, doch die Ausgabe dieser Funktion ist mit Vorsicht zu handhaben. Sollte auf einer Tabelle im fachlichen Datenmodell ein Trigger liegen, der zusätzlich einen weiteren Datensatz erzeugt, so liefert *@@IDENTITY* den vergebenen Primärschlüssel der zuletzt befüllten Tabelle. Diese entspricht jedoch nicht der eigentlich angesprochenen und durch eine View gekapselten fachlichen Tabelle. Alle drei Funktionen wurden in unterschiedlichen Konstellationen untersucht. Hierbei wurde unterschieden zwischen:

Liefert korrekten Primärschlüssel (✓), Liefert falschen/keinen Primärschlüssel (✗)

INSERT INTO …	**SCOPE_IDENTITY**	**@@IDENTITY**	**IDENT_CURRENT('T1')**
Tabelle T1 mit Zugriffsschicht ohne INSERT-Trigger	✗	✓	✓
Tabelle T1 mit Zugriffsschicht mit INSERT-Trigger T2	✗	✗	✓
Tabelle T1 ohne Zugriffsschicht ohne INSERT-Trigger	✓	✓	✓
Tabelle T1 ohne Zugriffsschicht mit INSERT-Trigger T2	✓	✗	✓

Tabelle 43: SQL-Server – Rückgabe von Primärschlüsseln

Die Funktion *IDENT_CURRENT(xxx)* erscheint hier als geeigneter Kandidat zur Rückgabe des zuletzt erstellten Primärschlüssels. Als Parameter wird der Name der Tabelle erwartet, deren letzter generierter Primärschlüssel zurückgegeben werden soll. Die Nutzung dieser Funktion gestaltet sich wie folgt:

```
SELECT IDENT_CURRENT('mt_data.CLIENT');
```

Tabelle 44: SQL-Server – Nutzung von IDENT_CURRENT

5.4 Performance

Der Fokus bei der Performanceanalyse liegt auf der Abfrage von Datensätzen. Hier stellt sich die Entwicklung eines einheitlichen Vorgehens als durchaus komplex heraus, was eine Vergleichbarkeit schwierig macht. Das liegt daran, dass die Geschwindigkeit einer Abfrage in diesem hier entwickelten Modell von sehr vielen Faktoren abhängt. Als Beispiel dafür sei die Höhe einer Hierarchie genannt, welche direkten Einfluss auf die Rekursionstiefe der zugrundeliegenden Abfrage in den gespeicherten Views hat. Um dennoch eine Vergleichbarkeit zu schaffen, wird darauf verzichtet, Sekundär-Assoziationen in den Testszenarien zu verwenden. Ebenfalls wird das bewusst einfach gehaltene fachliche Datenmodell aus Kapitel 4.5 zugrunde gelegt und nicht verändert.

Bei der Messung der Performance liegt der Fokus auf der Ausführungsgeschwindigkeit beim Lesen der gespeicherten fachlichen Datensätze. Als Aufbau der Organisation wird eine Struktur verwendet, die an einen balancierten Baum angelehnt ist.

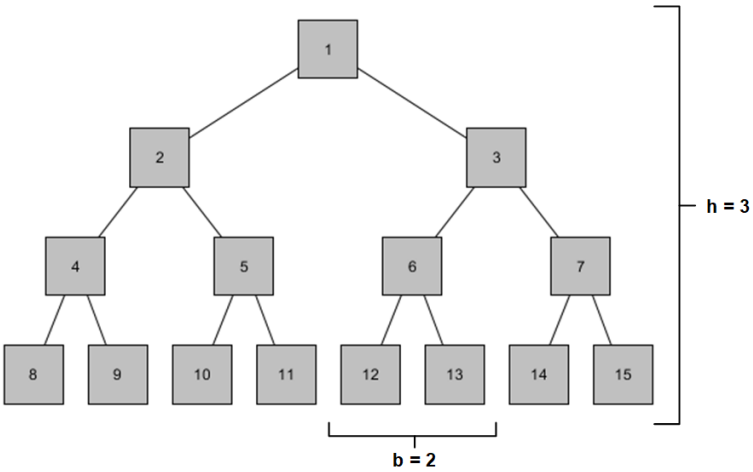

Bild 22: Beispiel – Hierarchie mit Höhe 3 und Breite 2

Die Anzahl der in Bild 22 dargestellten Organisationseinheiten lässt sich mit der nachfolgenden Formel berechnen, wobei m die maximale Höhe der Hierarchie und b die Breite, also die Anzahl der untergeordneten Organisationseinheiten, darstellt. Die Variable h dient in der nachfolgenden Formel, im Gegensatz zum dargestellten Bild, nur als Zählervariable.

$$A(m,b) := \sum_{h=0}^{m} b^h$$

Bei der Anzahl der Mitarbeiter innerhalb der Organisation gehen wir von einer gleichmäßigen Verteilung je Organisationseinheit aus. Dies entspricht zwar selten der

Realität, denn ein Team von fünf Mitarbeitern hat normalerweise keine fünf direkt übergeordneten Führungskräfte, doch dieser Umstand wird hier nicht berücksichtigt. Auch wird eine gleichmäßige Verteilung der fachlichen Datensätze auf die einzelnen Organisationseinheiten angenommen, was ebenfalls selten der Realität entspricht.

Die Anzahl der Mitarbeiter innerhalb einer Organisation berechnet sich aus der Anzahl der Mitarbeiter je Organisationseinheit, nachfolgend e genannt:

$$E(e, m, b) \coloneqq e * A(m, b)$$

Zum Vorgehen: Die Datenbank wird immer dann, wenn sich die Anzahl der enthaltenen Datensätze verändert, vollständig gelöscht und neu aufgebaut. Das soll Wechselwirkungen durch mögliche Fragmentierung innerhalb der Datenbank verhindern, die durch Löschvorgänge hervorgerufen werden können. Jede Messung wird drei Mal durchgeführt und als Mittelwert im Testprotokoll angegeben. Auch soll bei jeder Messung unterschieden werden, welcher Mitarbeiter die jeweilige Operation durchführt. Dazu wird jede Messung sowohl für einen Mitarbeiter auf oberster Ebene der Hierarchie durchgeführt, als auch für einen Mitarbeiter der sich auf der untersten Ebene der Hierarchie befindet. Im Messprotokoll werden Mitarbeiter auf der obersten Ebene als E_{FK} und Mitarbeiter auf der untersten Ebene als E_{MA} angegeben. Zu Beginn jeder Testreihe wird die Organisation generiert. Anschließend werden Testdaten erzeugt, bis die Anzahl der gewünschten fachlichen Datensätze erreicht ist.

Testszenario – Variable Anzahl fachlicher Datensätze $b = 4, m = 5, e = 4$		
Anzahl OU	**Anzahl Mitarbeiter**	**Fachl. Daten**
1365	5460	10.000
1365	5460	100.000
1365	5460	1.000.000
1365	5460	10.000.000

Tabelle 45: Testszenario – Variable Anzahl fachlicher Datensätze

Die Messungen der Performance werden ausschließlich auf der geschützten Tabelle *Client* durchgeführt, da diese mit einer Zugriffsschicht versehen wurde. Alle nachfolgenden Messungen fanden auf dem gleichen Computersystem statt, um Verfälschung der Ergebnisse durch unterschiedliche Hard- oder Software zu verhindern.

5.4.1 Messung - Oracle

Eine Oracle Database unterstützt die Bewertung eines vom Anwender entwickelten SQL-Statements mittels sogenannter Kosten, auch Abfragekosten genannt. Diese Ausführungskosten dienen als Indikator für den erwarteten Aufwand bei der Abarbeitung einer Anweisung. Dieser Wert setzt sich aus unterschiedlichen Faktoren wie beispielsweise den notwendigen I/O Operationen und der CPU-Last zusammen. [58] Die Optimierung von Abfragen oder der richtige Einsatz von Indizes kann die Kosten für eine Abfrage verringern. Hierzu stellt eine Oracle Database eine detaillierte Auflistung der Verursacher für die Kosten einer Abfrage zur Verfügung. [59] Jede der hier dargestellten Operationen wird sowohl auf einem einzelnen Datensatz als auch für einer Menge an Datensätzen durchgeführt. Damit soll ermittelt werden, welche Ausführungskosten bei dem Zugriff auf einen einzelnen bekannten Datensatz sowie bei Zugriff auf die Menge aller für den Anwender sichtbaren Datensätze entstehen.

```
-- Ausführungsplan für SQL-Statement berechnen
EXPLAIN PLAN FOR
    SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1;
-- Ausführungsplan abrufen
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
```

Tabelle 46: Oracle – Erstellen u. Abrufen von Ausführungsplänen

Bei der Bewertung eines solchen Ausführungsplans muss jedoch berücksichtigt werden, dass die angegebenen Ausführungskosten auf einer Schätzung der Datenbank basieren. So wurde bei ersten Tests für das gleiche SQL-Statement eine geschätzte Ausführungszeit von 2 Sekunden angegeben, obwohl die tatsächlich gemessene Ausführungszeit bei nur 0,2 Sekunden lag. Aus diesem Grund wird vom eigentlichen Vorgehen abgewichen, die Ausführungskosten als alleinige Messgröße zu verwenden. Stattdessen wird zusätzlich die wirkliche Ausführungszeit als Messgröße für die Performance des Systems erhoben. Neben dem zu erwarteten Aufwand, welcher ausschließlich auf einer Schätzung der Datenbank basiert, erlaubt die Oracle Database mittels *GATHER_PLAN_STATISTICS* auch die Erhebung von Statistiken während der Ausführung eines SQL-Statements. Die Ursache für diese Abweichungen kann möglicherweise an den Einstellungen des Optimizers liegen, was jedoch innerhalb dieser Arbeit nicht näher untersucht werden soll.

```
-- Messung beim Ausführen des SQL-Statements durchführen
SELECT /*+ GATHER_PLAN_STATISTICS */
COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1;

-- Ausführungsplan mit Messwerten abrufen
SELECT * FROM TABLE(DBMS_XPLAN.display_cursor(FORMAT => 'ALLSTATS LAST'));
```

Tabelle 47: Oracle – Messung u. Ausgabe eines verarbeiteten SQL-Statements

Um dieses Problem zu verdeutlichen, wird folgend ein kurzer Vergleich der einzelnen Messmethoden auf einem Datenbestand von 100.000 Datensätzen dargestellt. Die dargestellten Ergebnisse sind auf die notwendigen Informationen gekürzt.

```
EXPLAIN PLAN FOR SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1;
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);
-----------------------------------------------------------------------------------------
| Id  | Operation                | Name         |  Rows  |  Bytes | Cost (%CPU)|  Time    |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |              |      1 |     39 | 27948 (100)| 00:00:02 |
|   1 |   SORT AGGREGATE         |              |      1 |     39 |            |          |
|   2 |     NESTED LOOPS         |              |    68G |  2496G | 27948 (100)| 00:00:02 |

-- ========================================================================================

SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1;
SELECT * FROM TABLE(DBMS_XPLAN.display_cursor(FORMAT => 'ALLSTATS LAST'));
-------------------------------------------------------------------------------------
| Id  | Operation                | Name         | E-Rows | A-Rows |    A-Time   |
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |              |        |      1 | 00:00:00.35 |
|   1 |   SORT AGGREGATE         |              |      1 |      1 | 00:00:00.35 |
|   2 |     NESTED LOOPS         |              |    68G |  1000K | 00:00:00.36 |
```

Tabelle 48: Oracle – EXPLAIN PLAN / GATHER_PLAN_STATISTICS

Folgend sind die Ergebnisse der durchgeführten Messungen tabellarisch aufgeführt. Die Protokolle befinden sich in reduzierter Form im Anhang dieser Arbeit.

Messergebnisse (Oracle)					
Datenmenge	**Mitarbeiter**	**Select 1**		**Select Count(*)**	
		Kosten	**Dauer**	**Kosten**	**Dauer**
10.000	E_{FK}	887 *	0,22 s *	8.091 *	0,22 s *
	E_{MA}	887 *	0,22 s *	8.091 *	0,25 s *
100.000	E_{FK}	872	0,24 s	11.209	0,24 s
	E_{MA}	872	0,23 s	11.209	0,21 s
1.000.000	E_{FK}	872	0,22 s	27.948	0,35 s
	E_{MA}	872	0,21 s	27.948	0,22 s
10.000.000	E_{FK}	872 *	0,23 s	228.000 *	1,32 s *
	E_{MA}	872 *	0,22 s	228.000 *	0,22 s *

Tabelle 49: Performance – Messergebnisse Oracle

* Im Anhang enthalten

5.4.2 Messung - Microsoft SQL-Server

Ähnlich wie die Oracle Database zuvor, bietet auch der SQL-Server diverse Funktionen, welche sowohl die zu erwartenden Ausführungskosten vor der Ausführung eines SQL-Statements, als auch die tatsächlichen Ausführungskosten ermittelt. An dieser Stelle sei noch erwähnt, dass die Ermittlung der Ausführungskosten der beiden Datenbanksysteme nicht dazu dienen, diese miteinander zu vergleichen. Die Ausführungskosten werden hier ausschließlich dazu genutzt abzuschätzen, wie sich die gespeicherte fachliche Datenmenge bei Zugriffen über die Zugriffsschicht auf die Performance des Systems auswirkt.

Der SQL-Server arbeitet hierbei mit Session-Variablen, welche nach dem Durchführen der jeweiligen Messungen wieder deaktiviert werden sollten. Zum Durchführen von Messungen vor oder während der Ausführung von SQL-Statements muss dem jeweiligen Benutzer außerdem das Recht *SHOWPLAN* zugewiesen werden. Die Ergebnisse lassen sich entweder als XML oder Text in Tabellenformat ausgeben, wobei das von Microsoft bereitgestellte SQL-Server Management Studio die XML-Ausgabe interpretieren und grafisch aufbereitet darstellen kann.

```
USE [MT];
GO
-- Erstellen eines Ausführungsplan aktivieren
SET SHOWPLAN_XML ON;
GO
-- SQL-Statement ausführen
SELECT COUNT(*) FROM [MT].[mt_ass].CLIENT WHERE EID = 1;
GO
-- Erstellen eines Ausführungsplan deaktivieren
SET SHOWPLAN_XML OFF;
GO
```

Tabelle 50: SQL-Server – Ausführungsplan mittels SHOWPLAN_XML

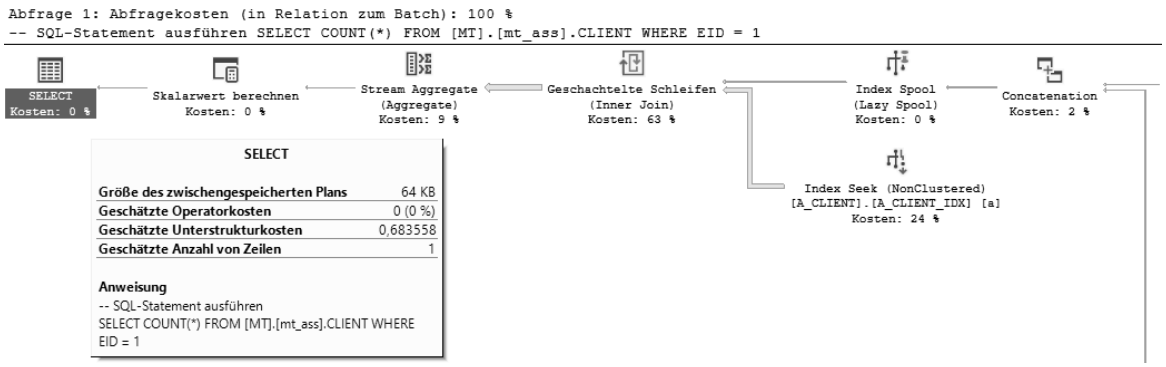

Bild 23: SQL-Server – Grafischer Ausführungsplan (Ausschnitt)

Die bereitgestellten Informationen des SQL-Servers sind derart umfangreich, dass sie leider nicht in den Anhang dieser Arbeit mit aufgenommen werden konnten. Jedoch soll der in Bild 23 dargestellte Ausschnitt die grafische Aufbereitung der XML-Ausgabe innerhalb des SQL-Server Management Studios verdeutlichen. Zu jedem enthaltenen Element des Ausführungsplans lassen sich außerdem detaillierte Informationen abrufen. Die oben dargestellten Informationen, welche über ein separates Fenster eingeblendet werden, stellen nur einen Ausschnitt der verfügbaren Informationen dar.

Bei dem Eintrag „Geschätzte Unterstrukturkosten“ aus Bild 23 handelt es sich um die aggregierten Ausführungskosten, welche von der Datenbank für die Ausführung des SQL-Statements erwartet werden. Der tatsächliche Ausführungsplan lässt sich wie folgt ermitteln:

```
USE [MT];
GO
-- Erstellen eines Ausführungsplan aktivieren
SET STATISTICS XML ON;
GO
-- SQL-Statement ausführen
SELECT COUNT(*) FROM [MT].[mt_ass].CLIENT WHERE EID = 1;
GO
-- Erstellen eines Ausführungsplan deaktivieren
SET STATISTICS XML OFF;
GO
```

Tabelle 51: SQL-Server – Ausführungsplan mittels STATISTICS

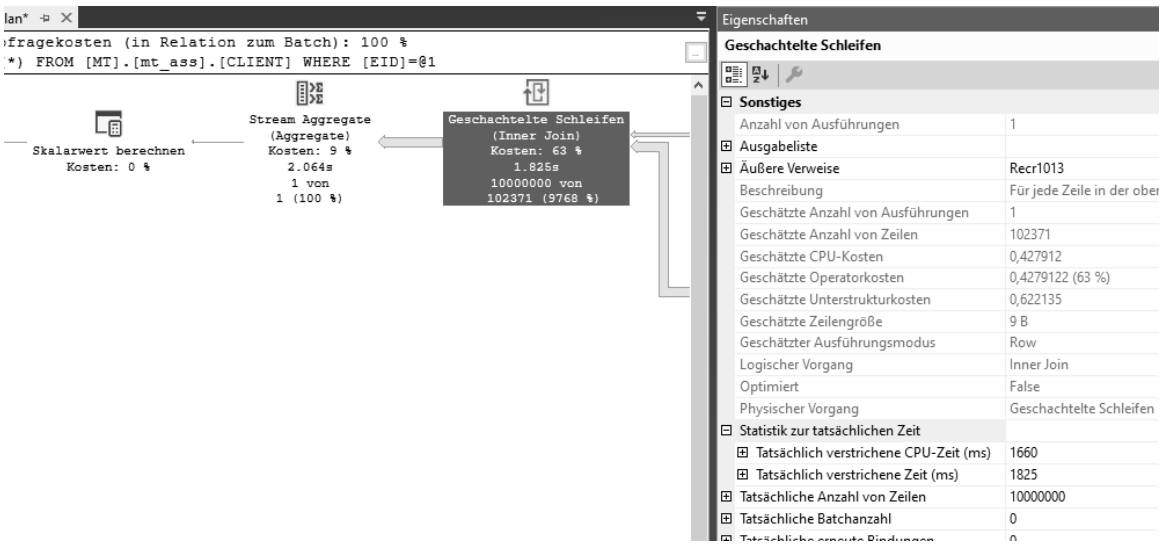

Bild 24: SQL-Server – Grafischer Ausführungsplan (Ausschnitt)

Mit dieser Möglichkeit lässt sich jede Abfrage bis ins Detail untersuchen und auf Performance-Engpässe prüfen.

Die Messwerte die für jeweiligen SQL-Statements sind entsprechend in der folgenden Tabelle aufgeführt. Dabei ist das Vorgehen bei beiden Systemen identisch. Hier sei noch einmal erwähnt, dass die Messprotokolle aufgrund ihrer Größe nicht in den Anhang aufgenommen werden konnten. Sie umfassen je Messvorgang über 40 Kilobyte an XML-Strukturen.

Messergebnisse (SQL-Server)					
Datenmenge	**Mitarbeiter**	**Select 1**		**Select Count(*)**	
		Kosten	**Dauer**	**Kosten**	**Dauer**
10.000	E_{FK}	0,038	0,01 s	0,045	0,02 s
	E_{MA}	0,038	0,01 s	0,045	0,02 s
100.000	E_{FK}	0,038	0,01 s	0,052	0,03 s
	E_{MA}	0,038	0,01 s	0,052	0,03 s
1.000.000	E_{FK}	0,038	0,01 s	0,111	0,17 s
	E_{MA}	0,038	0,01 s	0,111	0,13 s
10.000.000	E_{FK}	0,038	0,01 s	0,684	1,8 s
	E_{MA}	0,038	0,01 s	0,684	0,15 s

Tabelle 52: Performance – Messergebnisse SQL-Server

Weiterführende Informationen zur Erstellung und Nutzung von Ausführungsplänen unter Verwendung des SQL-Servers finden sich unter [72].

5.4.3 Speicherbelegung

Auch der zu belegende Speicher, speziell wenn eine geschützte Tabelle wächst und sich ihr Datenbestand stark erhöht, sollte Berücksichtigung finden. Das Schema, welche die Organisationseinheiten und Mitarbeiter beinhaltet, wird aufgrund seiner zu erwartenden Größe an dieser Stelle zu vernachlässigen. Der Blick sollte jedoch in Richtung des Assoziations-Schemas gehen. Hier hat sich nach dem Erzeugen von 10.000.000 Testdaten im Rahmen der Performance-Messungen herausgestellt, dass das Assoziations-Schema stärker gewachsen ist, als das fachliche Datenmodell. Wie in der Tabelle nachfolgend zu entnehmen ist, belegt alleine das Assoziations-Schema innerhalb der Oracle Database über 610 MB während das fachliche Datenmodell bei etwa 480 MB liegt. Der belegte Speicher des Assoziations-Schemas lässt sich sicherlich noch durch den Einsatz von Kompression und die Wahl anderer Datentypen reduzieren, jedoch sollte das Wachstum der darin liegenden Indizes und Tabellen beim Sizing einer Datenbank und der Auswahl der zu schützenden Tabellen berücksichtigt werden.

Kurz erwähnt sei an dieser Stelle, dass der Index *SYS_C0022095*, der auf dem Primärschlüssel der Tabelle *A_CLIENT* liegt, nur etwa halb so groß ist wie der zusätzlich hinzugefügte Index *A_CLIENT_IDX* auf der Spalte *OUID*, obwohl beide Indizes auf den gleichen Datentyp referenzieren und bei identischer Anzahl an Datensätzen theoretisch die gleiche Größe besitzen müssten. Dieser Umstand lässt sich damit erklären, dass der Primärschlüssel-Index als *UNIQUE* gekennzeichnet ist, was dem anderen Index fehlt. In solchen Fällen übernimmt die Oracle Database zusätzlich zum indizierten Attribut auch automatisch die sogenannte *ROWID* [67] als weiteres Attribut in den Index mit auf, um alle Elemente des Index eindeutig adressieren zu können. [68]

```
-- Ausführen als DBA / SYSTEM
SELECT
  DBA_SEGMENTS.OWNER,        DBA_SEGMENTS.SEGMENT_NAME,
  DBA_SEGMENTS.SEGMENT_TYPE, ROUND(SUM(BYTES)/1024,0) as "SIZE in KB"
FROM DBA_SEGMENTS
WHERE DBA_SEGMENTS.OWNER LIKE 'MT%'
GROUP BY DBA_SEGMENTS.OWNER, DBA_SEGMENTS.SEGMENT_NAME, DBA_SEGMENTS.SEGMENT_TYPE
ORDER BY OWNER, "SIZE in KB" DESC;

-----------------------------------------------------------------------------
| OWNER         | SEGMENT_NAME                | SEGMENT_TYPE | SIZE in KB |
-----------------------------------------------------------------------------
| MT_ASS_ADMIN  | A_CLIENT_IDX                | INDEX        |     294912 |
| MT_ASS_ADMIN  | A_CLIENT                    | TABLE        |     172032 |
| MT_ASS_ADMIN  | SYS_C0022095                | INDEX        |     163840 |
| MT_DATA_ADMIN | CLIENT                      | TABLE        |     327680 |
| MT_DATA_ADMIN | SYS_C0022084                | INDEX        |     163840 |
| MT_ORG_ADMIN  | EMPLOYEE                    | TABLE        |        192 |
| MT_ORG_ADMIN  | EMPLOYEE_OU_IDX             | INDEX        |        192 |
...[gekürzt]...
```

Tabelle 53: Oracle – Speicherbelegung aller Segmente

Interessant gestaltet sich auch der Blick auf die Speicherbelegung bei dem SQL-Server nach der Erzeugung von 10.000.000 Datensätzen. Diese liegt um fast 50% unter der Speicherbelegung der Oracle Database, wie in der nächsten Tabelle dargestellt ist. So belegt das fachliche Datenmodell mit ca. 500 MB die gleiche Menge an Speicher, während das Assoziations-Schema hingegen mit ca. 310 MB weit weniger Platz belegt. Eine Ursache für diese starke Abweichung liegt vermutlich in den verwendeten Datentypen. Eine genauere Untersuchung steht noch aus, soll jedoch kein Bestandteil dieser Arbeit sein.

Als Primärschlüssel innerhalb des SQL-Servers wurde ausschließlich der Datentyp INT verwendet. Dieser belegt innerhalb des SQL-Servers 4 Byte. [69] Für den Aufbau des Modells innerhalb der Oracle Database wurde ebenfalls INT verwendet, doch dieser Datentyp wird innerhalb der Datenbank als NUMBER gespeichert. Ein solcher Datentyp umfasst jedoch nicht die erwarteten 4 Byte, sondern er wird als numerischer Datentyp variabler Länge abgelegt, ähnlich wie ein VARCHAR eine Zeichenkette variabler Länge darstellt. Jeder gespeicherte NUMBER Datensatz kann eine Länge von bis zu 22 Byte erreichen, abhängig vom Inhalt. [70]

Da die Tabelle A_CLIENT zwei Indizes auf Spalten dieses Datentyps besitzt und jeder Index jeden indizierten Wert der Tabelle als Kopie vorhält, kann ein solches Konstrukt schnell entsprechend viel Speicherplatz beanspruchen. Die nachfolgende Übersicht unterscheidet an dieser Stelle nicht zwischen einem Index und einer Tabelle.

```
-- Ausführen als sa / Administrator
SELECT
    s.Name AS "Schema",
    t.Name AS "TableName",
    SUM(a.used_pages) * 8 AS "Size in KB"
FROM sys.tables t
INNER JOIN sys.indexes          i ON t.OBJECT_ID    = i.object_id
INNER JOIN sys.partitions       p ON i.object_id    = p.OBJECT_ID AND i.index_id = p.index_id
INNER JOIN sys.allocation_units a ON p.partition_id = a.container_id
INNER JOIN sys.schemas          s ON s.schema_id    = t.schema_id
GROUP BY s.Name, t.Name
ORDER BY s.Name, t.Name;

-------------------------------------------
| Schema  | TableName        | Size in KB |
-------------------------------------------
| mt_ass  | A_CLIENT         |     315736 |
| mt_ass  | A_ORDER          |          0 |
| mt_data | CLIENT           |     508224 |
| mt_data | ORDER            |          0 |
| mt_data | ORDERPRODUCT     |          0 |
| mt_data | PRODUCT          |          0 |
| mt_org  | ASSOCIATION      |          0 |
| mt_org  | EMPLOYEE         |        360 |
| mt_org  | ORGANIZATIONUNIT |        128 |
```

Tabelle 54: SQL-Server – Speicherbelegung aller Tabellen inkl. Indizes

6 Bewertung

Zur Bewertung der einzelnen Datenbanksysteme werden die Ergebnisse der Überprüfung aus dem vorherigen Kapitel zusammengefasst und hier noch einmal kurz wiedergegeben.

Um ein abschließendes Ergebnis zu erzielen, werden alle Einzelergebnisse mit einer Bewertung nach dem folgenden Schema versehen:

Anforderung erfüllt (✓), Anforderung teilweise erfüllt (-), Anforderung nicht erfüllt (✗)

6.1 Grundlegende Anforderungen

Die aufgestellten Anforderungen haben sowohl der SQL-Server als auch die Oracle Database grundsätzlich erfüllt. Alle abgesetzten Standard-SQL-Statements wurden innerhalb des Modells von beiden Datenbanken korrekt verarbeitet. Die Sichtbarkeit von Datensätzen entlang der Hierarchie konnte korrekt abgebildet werden, was die wesentliche Hauptfunktionalität für das hier entwickelte Modell darstellt.

Jedoch hat sich bei Oracle eine Besonderheit in Bezug auf die Änderung der Eigentümer-OU hervorgetan, was zu einem leichten Abzug in der Bewertung dieses Systems führt.

Das Datenbanksystem MariaDB unterstützt leider keine *INSTEAD-OF* Trigger auf Views, was zu einem Ausschluss aus der hier aufgestellten Bewertung führt. Grundsätzlich lässt sich das hier entwickelte Modell auch innerhalb von MariaDB realisieren, was jedoch eine Vielzahl von Prozeduren und Funktionen notwendig machte und somit nicht den Anforderungen dieser Arbeit entsprach. Die Prozeduren und Funktionen für MariaDB sind dem Anhang 25 zu entnehmen.

Auch wenn die für MariaDB entwickelten Views, ebenso wie bei der Oracle Database und dem SQL-Server, die Sichtbarkeit der gespeicherten Datensätze entlang der Hierarchie korrekt wiedergaben, wurden keine gesondert protokollierten Testfälle für MariaDB mit in diese Arbeit aufgenommen.

Grundlage	**Bemerkung**	**Oracle**	**SQL-Server**	**MariaDB**
Einfügen, Lesen Kapitel 5.1.1	Keine	✓	✓	✗*
Ändern, Löschen Kapitel 5.1.2	Keine	✓	✓	✗*
Eigentümer-OU ändern Kapitel 5.1.3	Oracle: Inkonsistentes Verhalten	-	✓	✗*
Umstrukturierung Kapitel 5.1.4	Keine	✓	✓	✗*
Konsistenzbedingungen Kapitel 5.1.5	Keine	✓	✓	✗*

Tabelle 55: Bewertung – Grundlegende Anforderungen

* Ausschluss aus der Bewertung trotz korrekter Funktionalität

6.2 Anforderungen – Zugriffsschicht

In diesem Abschnitt zeigt sich, dass die aufgestellten Anforderungen an die zu erstellende Zugriffsschicht von allen Datenbanksystemen gleichermaßen umgesetzt werden konnte. Der einzige Kritikpunkt ist die Anforderung der Abstraktion.

Hier hat sich MariaDB aufgrund der fehlenden Unterstützung für *INSTEAD-OF* Trigger und der Notwendigkeit, die Logik dieser Trigger in Prozeduren auszulagern, leider disqualifiziert.

Die Oracle Database sowie der SQL-Server können die Anforderung der Abstraktion nur teilweise erfüllen und erhalten daher ebenfalls einen leichten Abzug in der Wertung. Die erstellten Views zum Aufbau einer Zugriffsschicht enthalten eine zusätzliche Spalte, welche zwingend bei jedem Aufruf mittels *WHERE*-Bedingung zu füllen ist.

Grundlage	Bemerkung	Oracle	SQL-Server	MariaDB
Unveränderlichkeit der Fachlichkeit Kapitel 5.2.1	Keine	✓	✓	✓
Urheberzuordnung Kapitel 5.2.1	Keine	✓	✓	✓
Transitive Zugriffe Kapitel 5.2.1	Keine	✓	✓	✓
Abstraktion Kapitel 5.2.1	Keine	-	-	✗

Tabelle 56: Bewertung – Funktionale Anforderungen – Zugriffsschicht

6.3 Anforderungen – Operativer Betrieb

Sowohl die Multiple Zuordnung von Mitarbeitern zu Organisationseinheiten, als auch die Berücksichtigung von Mitarbeiterfluktuation, konnte von allen Datenbanken gleichermaßen erfüllt werden, was in dem eigens geschaffenen Modell zum Speichern und Nutzen einer Hierarchie begründet ist. Dieses Modell ist bei allen Datenbanksystemen identisch aufgebaut. Durch dieses Modell zur Abbildung einer hierarchischen Organisationsstruktur werden außerdem notwendige Veränderungsprozesse ermöglicht. Die Struktur des Modells ändert sich nicht, da sowohl Organisationseinheiten als auch Mitarbeiter als eigenständige Datensätze innerhalb dieses Modells abgelegt sind. Auch diese Anforderungen wurden, aufgrund des identischen Modells, von allen Datenbanksystemen unterstützt. Allerdings unterscheiden sich die einzelnen Systeme in ihrer proaktiven Erkennung von möglichen Zyklen innerhalb der Hierarchie. Bei dieser Bewertung erhalten der SQL-Server sowie die Oracle Database einen leichten Abzug, da die Vermeidung von Zyklen auf der datenbankseitigen Erkennung unendlicher Rekursion basiert. Der SQL-Server erlaubt es beispielsweise nicht, mittels Trigger die eingegebenen Daten vor dem Speichern zu prüfen. Die Oracle Database verlangt zwingend das Ausführen eines Triggers innerhalb einer eigenen Transaktion, wenn dieser auf den Inhalt der zugrundeliegenden Tabelle zugreifen möchte, siehe dazu Kapitel 4.4.3.

Bei MariaDB sei erwähnt, dass es nicht möglich war, einen separaten Check-Constraint auf eine *AUTO_INCREMENT* Spalte anzuwenden, was sich jedoch durch einen zusätzlichen *BEFORE-INSERT* Trigger beheben lies, siehe Anhang 23. Besonders positiv ist hier zu werten, dass MariaDB das einzige Datenbanksystem war, in dem proaktiv mögliche Zyklen mittels Trigger erkannt und korrekt verhindert werden konnten, da die hier eingesetzten Trigger sowohl mit *FOR-EACH-ROW* als auch *BEFORE* deklariert werden konnten. Selbst die Nutzung von Transaktionen hatte keinen negativen Einfluss auf die Funktionsweise dieser Trigger.

Grundlage	**Bemerkung**	**Oracle**	**SQL-Server**	**MariaDB**
Mitarbeiterfluktuation	Keine	✓	✓	✓
Multiple Zuordnung	Keine	✓	✓	✓
Veränderungprozess	Keine	✓	✓	✓
Unveränderlichkeit der Struktur	Keine	✓	✓	✓
Zyklenfreiheit	Keine	-	-	✓

Tabelle 57: Bewertung – Funktionale Anforderungen – Operativer Betrieb

6.4 Nicht-Funktionale Anforderungen

Der Nutzung von Triggern im SQL-Server gestaltet sich etwas komplizierter als bei der Oracle Database oder MariaDB, da hier keine *BEFORE*- sowie keine *FOR-EACH-ROW* Trigger möglich sind. Auch bei der Erkennung von Rekursionen verhält sich der SQL-Server etwas träge. Er erkennt Rekursionen erst beim Auftreten. Am schwersten wiegt jedoch die Tatsache, dass Views nicht im Kontext des Eigentümers ausgeführt werden können. Die Oracle Database hingegen tut sich etwas schwer bei der Zuweisung von Schemata zu einzelnen Benutzern sowie der zwingend zu setzenden Berechtigung für jedes Objekt innerhalb eines fremden Schemas. Auch die Tatsache, dass neu erzeugte Primärschlüssel nicht mittels *RETURNING* aus einer View heraus zurückgegeben werden können, ist durchaus bedauerlich. Eine endgültige Lösung hierfür steht noch aus.

Im Vergleich zum SQL-Server hat sich MariaDB besonders positiv hervorgetan. So bietet das System die Möglichkeit, Trigger sowohl als *BEFORE*- als auch *FOR-EACH-ROW* deklarieren zu können. Auch erlaubt das unterstützte Rechte-System die saubere Kapselung von Tabellen über Views, welche dann im Kontext des View-Eigentümers ausgeführt werden können. Die Rückgabe von Primärschlüsseln ist über die im Anhang 25 enthaltenen Funktionen zum Einfügen neuer Datensätze möglich und liefert auch bei nachgelagerten Insert-Triggern korrekte Ergebnisse. Leider liegt der Default-Wert für die Rekursionserkennung bei MariaDB so hoch, dass sich dies in einem Abzug der Wertung wiederspiegelt. Eine Anforderung zur Reduktion des Defaults liegt seit 2018 unter der Nummer MDEV-17239 vor, wurde zum aktuellen Zeitpunkt jedoch nicht umgesetzt. [75]

Grundlage	**Bemerkung**	**Oracle**	**SQL-Server**	**MariaDB**
Trigger Kapitel 5.3.1	SQL-Server: Kein BEFORE u. FOR-EACH-ROW	✓	-	✓
Schema-Zuweisung Kapitel 5.3.2	Oracle: Workaround mittels Logon-Trigger	-	✓	✓
Rekursionserkennung Kapitel 5.3.3	SQL-Server: Reaktiv bei Auftreten MariaDB: Default bei 4.294.967.295	✓	-	✗
Kapselung über Views Kapitel 5.3.4	SQL-Server: Technisch nicht möglich	✓	✗	✓
Primärschlüssel Kapitel 5.3.5	Oracle: Rückgabe nur via Workaround	✗	✓	✓

Tabelle 58: Bewertung – Nicht-Funktionale Anforderungen

6.5 Performance

Wie sich bei Messungen gezeigt hat, ist das Modell durchaus noch stark ausbaufähig, speziell wenn man die technischen Möglichkeiten der vorliegenden Systeme betrachtet und nutzt. Auch wenn die Oracle Database in vielen Rankings bei den relationalen Datenbanken den Platz 1 belegt, hat sich bei der Performance doch der SQL-Server positiv hervorgetan. So sind die Ausführungskosten bei dem SQL-Server vergleichsweise langsam angestiegen, was nach der Betrachtung der Messwerte von Oracle nicht zu erwarten war. Auch fällt der belegte Festplattenspeicher bei dem SQL-Server ebenfalls wesentlich geringer aus als bei der Oracle Database. Dieser Umstand wurde im Rahmen dieser Arbeit beleuchtet, jedoch bewusst nicht behoben, um einem Leser mögliche Fallstriche nicht vorzuenthalten. Der Erstellung von Testdaten ist an dieser Stelle kein eigenes Kapitel gewidmet, da dies nicht der Fokus dieser Arbeit ist. Weil sich jedoch beide Systeme in dieser Hinsicht stark voneinander unterscheiden, wurde dieser Umstand mit in die Bewertung aufgenommen. Die Messwerte befinden sich in Form von Kommentaren in den Anhängen dieser Arbeit, siehe Anhang 8 und 20.

MariaDB wird der Vollständigkeit halber mit aufgeführt. Hier fanden jedoch keine Messungen statt, da sich das konzipierte Modell auf Basis von Views nicht mit den unterstützten Technologien erstellen ließ. Ein entsprechender Test unter Verwendung der erstellten Funktionen und Prozeduren, siehe Anhang 25, war nicht vorgesehen und wurde daher nicht durchgeführt.

Grundlage	**Bemerkung**	**Oracle**	**SQL-Server**	**MariaDB**
Messung Kapitel 5.4.1 u. 5.4.2	Oracle: Höhere Ausführungskosten	-	✓	✗*
Speicherplatz Kapitel 5.4.3	Oracle: Doppelte Speicherplatzbelegung	-	✓	✗*
Erstellung Testdaten Anhang 8 u. 20	SQL-Server: Dauer für Erstellung	✓	-	✗*

Tabelle 59: Bewertung – Performance

* Keine Messungen durchgeführt

7 Zusammenfassung und Ausblick

7.1 Einschränkungen und Risiken

Bei der Nutzung des hier entwickelten Modells sollen Einschränkungen in der Handhabung sowie mögliche Risiken durch falsche Handhabung nicht außeracht gelassen werden. Dieses Kapitel erhebt keinen Anspruch auf Vollständigkeit, da die möglichen Einschränkungen und Risiken bei der Nutzung des hier entwickelten Modells stark abhängig vom jeweiligen Anwendungsszenario und den Anforderungen an die verwendete Datenbank sind.

7.1.1 Umgang mit Zwischentabellen

Ein häufig anzutreffendes Konstrukt innerhalb einer relationalen Datenbank ist die Zwischentabelle, welche die Abbildung einer N-zu-M Relation zwischen zwei Tabellen ermöglicht. Eine solche Zwischentabelle enthält häufig einen zusammengesetzten Primärschlüssel, der sich aus den Primärschlüsseln der beiden verbundenen Tabellen zusammensetzt. Sollte nun diese Zwischentabelle mittels Zugriffsschicht geschützt werden, entsteht innerhalb der dafür benötigten Assoziations-Tabelle, welche für die Funktionsweise der modellierten Zugriffsschicht benötigt wird, ebenfalls eine vollständige Kopie des referenzierten zusammengesetzten Primärschlüssels.

Zum besseren Verständnis dieses Problems soll folgendes Szenario konstruiert werden: Es wird eine Zwischentabelle T_Z erstellt, welche die Verknüpfung von Einträgen der Tabellen T_1 und T_2 ermöglicht. Die Zwischentabelle T_Z ist mittels Zugriffsschicht geschützt. Nun stellt ein Mitarbeiter A zwischen den Objekten o1 aus T_1 und o2 aus T_2 eine solche Relation über einen Eintrag in der Zwischentabelle T_Z her. Dieser neue Eintrag ist für einen Mitarbeiter B nicht sichtbar, da die Zwischentabelle ja geschützt ist und sich Mitarbeiter B nicht zwangsläufig in der benötigten Organisationseinheit befindet, um den Datensatz zu lesen. Wenn Mitarbeiter B nun versucht, eine Verknüpfung der beiden Objekte o1 und o2 herzustellen, weil er die bestehende Verknüpfung nicht sehen kann, wird dieser Versuch von der Datenbank verweigert, da der zusammengesetzt Primärschlüssel bereits in der Zwischentabelle sowie der zugehörigen Assoziations-Tabelle existiert.

Es wird daher dringend empfohlen, Zwischentabellen und ähnliche Konstrukte mit zusammengesetzten Primärschlüsseln, die durch einen Benutzer vergeben werden können, nicht mittels Zugriffsschicht zu schützen.

7.1.2 Isolations-Level

Die explizite Nutzung von bestimmten weniger restriktiven Isolations-Leveln sollte grundsätzlich nur dann erfolgen, wenn sichergestellt ist, dass keine negativen Wechselwirkungen durch parallellaufende Transaktion auftreten können.

Speziell im Fall des von „READ UNCOMMITTED“ kann, wenn sich parallel im Hintergrund durch einen Administrator die Organisationsstruktur ändert, ein inkonsistentes Modell entstehen und zum Abbruch einer Anweisung durch einen Zyklus innerhalb der Hierarchie führen. Aus diesem Grund unterstützt die Oracle Database diesen als „dirty reads“ bezeichneten Isolations-Level entgegen der offiziellen SQL-Spezifikation nicht. Der Microsoft SQL-Server hingegen erlaubt die Nutzung dieses Isolations-Levels.

7.1.3 Veränderung des Eigentümers

Es ist nicht empfehlenswert, zwei oder mehr Datensätze aus unterschiedlichen Tabellen durch Veränderung der zugeordneten Eigentümer-Organisationseinheit auf nur einer geschützten Tabelle voneinander zu trennen, wenn diese fachlich zusammengehören. Welche Datensätze eine fachliche Verknüpfung zueinander haben, sollte daher vor Ausführen der jeweiligen Anweisung bekannt sein. Wenn das fachliche Modell und sein Nutzungsszenario klar definiert ist, besteht die Möglichkeit, eine *UPDATE*-Operation welche die Eigentümer-Organisationseinheit ändert, auch an die fachlich abhängigen Tabellen über die abstrahierenden Views weiterzugeben. Eine solche Logik ließe sich innerhalb der jeweiligen *UPDATE*-Trigger platzieren. Ein solches Konstrukt würde zu einer entsprechenden Kaskade von *UPDATE*-Operationen führen, um die Sichtbarkeit für alle abhängigen und ebenfalls geschützten Datensätze konsistent zu halten.

7.1.4 Verkettung von Datensätzen

Auch der unbedachte Einsatz von *JOIN* Operationen zur Verknüpfung von Datensätzen innerhalb der SQL-Statements einer Anwendung kann durchaus zu Problemen führen. Sollte beispielsweise im Bereich Controlling die Sicht auf Rechnungsdatensätze ohne Sichtbarkeit der verknüpften Kunden-Daten möglich sein, so sollte auf die Nutzung eines *INNER JOIN* für die Verknüpfung dieser Datensätze verzichtet werden, da die fehlende Sichtbarkeit der Kunden-Daten auch zum Ausblenden der Rechnungen im Controlling führen würde. In solchen Fällen muss auf ein entsprechendes *OUTER JOIN* zurückgegriffen werden, welches die angeforderten Rechnungsdaten auch ohne die zugeordneten Kundendaten darstellt. Die Logik zur Berücksichtigung der, bei der Verwendung von *OUTER JOIN* dargestellten, *NULL* Werte obliegt ebenfalls der Anwendung, was ggf. Änderungen im Quellcode nach sich zieht.

7.1.5 Kaskadierende Operationen

Außerdem ist von der Nutzung kaskadierender Änderungs- oder Lösch-Operationen abzuraten. Diese mögen innerhalb des fachlichen Datenmodells für Ordnung in den Relationen sorgen, doch die Fremdschlüsselbeziehung innerhalb des Assoziations-Schemas, welche auf einen vorhandenen Primärschlüssel in der Tabelle des fachlichen Datenmodells angewiesen sind, können kaskadierende Operationen durchaus verhindern.

Auch ist davon abzuraten, den Primärschlüssel eines Datensatzes innerhalb des fachlichen Datenmodells zu ändern. Einerseits referenziert der entsprechende Eintrag im Assoziations-Schema auf diesen, andererseits unterstützen die bisherigen Update-Trigger das Ändern von Primärschlüssel-Attributen nicht. Dies ist bewusst so gewählt, denn ein Primärschlüssel sollte einmalig gesetzt und anschließend nicht mehr verändert werden, auch wenn Datenbanksysteme eine solche Änderung unterstützen.

7.1.6 Inkonsistenz der Sichtbarkeit

Tabellen, die mittels Zugriffsschicht geschützt sind, dürfen ausschließlich über die dafür vorgesehenen Views eine Veränderung ihres Inhalts erfahren. Der Grund dafür ist trivial: Die *INSTEAD-OF* Trigger auf den abstrahierenden Views stellen die Konsistenz des Modells sicher. Im Idealfall existiert die gleiche Anzahl an Datensätzen, sowohl in einer fachlichen Tabelle, als auch einer zugehörigen Assoziations-Tabelle. Sollte an diesem Konstrukt vorbei gearbeitet werden, indem die Assoziations-Tabelle oder die fachliche Tabelle über einen anderen Nutzer geändert wird, ist ein fachlicher Datensatz im Anschluss möglicherweise niemals wieder auffindbar.

Ein vollständiger Scan der betroffenen Tabelle mit anschließendem Abgleich der Einträge in der Assoziations-Tabelle ist die Folge. Ein solcher Vorgang zum Nachpflegen fehlender Assoziationen nimmt entsprechend viel Zeit in Anspruch und belastet die Datenbank.

7.1.7 Anpassung der Views bei Update

Über die abstrahierenden Views wird die Konsistenz des Modells sichergestellt. Innerhalb des Triggers, welcher die Insert-/Update-Statements verarbeitet, wird jedes Insert/Update Spalte für Spalte der fachlichen Tabelle zugeordnet. Sollte sich das fachliche Datenmodell ändern, muss zwingend jeder Insert- sowie Update-Trigger angepasst werden. Hier bietet sich durchaus Potential für Automatisierung durch Trigger-Generatoren, jedoch birgt die Notwendigkeit der Trigger-Anpassung auch das Risiko für verworfene Änderungen und somit unwiederbringlichen Datenverlust, wenn die Weiterleitung der neuen Spalten innerhalb des Triggers nicht eingebaut wurde.

7.1.8 Transaktionen und Konsistenz

Per Definition dient eine Transaktion dazu, einen Datenbestand einer Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand zu überführen. Innerhalb einer Transaktion kann sich der Datenbestand in einem inkonsistenten Zustand befinden, was durch die Kapselung der Transaktion grundsätzlich erst einmal kein Problem darstellt. Am Ende einer Transaktion muss der Datenbestand jedoch wieder alle zuvor definierten Bedingungen der Konsistenz erfüllen.

Was in diesem Zusammenhang Konsistenz bedeutet, gibt der Entwickler eines Datenbankmodells selbst vor. Solche Konsistenzbedingungen werden über Constraints definiert, siehe dazu beispielsweise Tabelle 12. Hier wird vorgegeben, dass innerhalb der Tabelle *ORGANIZSATIONUNIT* die Spalte *PARENT* nicht identisch mit der Spalte *ID* sein darf, um Zyklen aufgrund reflexiver Assoziationen zu vermeiden.

Während dieser Arbeit hat sich jedoch auch herausgestellt, dass diese Constraints keine komplexe Logik oder Abläufe beinhalten können. Um trotz dieser Einschränkung die Bildung von Zyklen innerhalb von Hierarchien trotzdem zu vermeiden, wurde auf Trigger zurückgegriffen, die im Falle eines Zyklus einen Fehler werfen und eine Transaktion somit abbrechen.

Hier fällt bereits auf, dass dies zu einem Problem bei der Veränderung einer Hierarchie führen kann. Innerhalb einer Transaktion werden per Constraint definierte Konsistenzbedingungen nicht geprüft, während Trigger unabhängig davon aktiviert werden und die Transaktion abgebrochen werden kann, auch wenn der Entwickler eine Sicherstellung der Konsistenz zum Ende seiner Transaktion vorgesehen hat. Die Trigger greifen somit auch innerhalb einer Transaktion, sollte hier bereits ein temporärer Zyklus entstehen, der gegen vor Ende der Transaktion wieder aufgelöst werden würde.

7.2 Ausblick

In diesem Kapitel sollen mögliche Verbesserungen und offene Fragestellungen, welche im Rahmen dieser Arbeit keine Berücksichtigung gefunden haben, betrachtet werden. Sie dienen als Denkanstoß und als Appell an alle, welche vom hier beschriebenen Konzept überzeugt sind, dieses weiterzuentwickeln und darauf aufzubauen.

7.2.1 Session-Variablen

Im Rahmen dieser Arbeit wird angenommen, dass es keine dedizierte Verbindung eines Anwenders zur Datenbank gibt. Dies hat zur Folge, dass jedes SQL-Statement, welches gegen eine geschützte Tabelle läuft, um die ausführende Mitarbeiter-ID (*EID*) erweitert werden muss. Dies führt jedoch dazu, dass die Anforderungen der Abstraktion nicht erfüllt werden können, denn eine Anwendung muss ihren Quellcodes ändern, um auf diesem Modell aufsetzen zu können.

Sollte einem Benutzer jedoch eine dedizierte Verbindung zur Datenbank zur Verfügung stehen, so ließe sich der Aufwand für Anpassungen stark reduzieren, indem die jeweilige *EID* eines Mitarbeiters nach erfolgreichem Aufbau einer Verbindung als sogenannte Session-Variable gespeichert wird. Diese *EID* steht dem Benutzer nun innerhalb seiner bestehenden Session im Speicher der Datenbank zur Verfügung. Diese Session-Variable kann anschließend innerhalb der abstrahierenden Views sowie den *INSTEAD-OF* Trigger abgerufen und genutzt werden.

Die Darstellung nach außen, dass sich eine Tabelle durch die abstrahierte View um eine Spalte EID erweitert hat, die sich im fachlichen Datenmodell nicht wiederfindet, wäre damit behoben.

Auch die Situation, dass ein Mitarbeiter zu viele Datensätze sieht auf die er nicht berechtigt ist, wenn er die *EID* nicht an die *WHERE*-Bedingung eines SQL-Statements anhängt, ist damit ebenfalls gelöst.

7.2.2 Rechte- und Rollenkonzept

Die bisher innerhalb der Datenbank vergebenen Berechtigungen wurden im Rahmen dieser Arbeit direkt an die jeweiligen Datenbank-Benutzer vergeben. Es ist durchaus möglich, die in Kapitel 4.2 beschriebenen Rechte als eigene Rollen innerhalb der Datenbanken anzulegen, was deren Wartung bei Veränderungen vereinfacht.

Außerdem bietet sich die Nutzung von Rollen innerhalb der Datenbanken an, sollten mehrere fachliche Datenmodelle über je eine Assoziationsschicht mit der Organisationsstruktur verknüpft werden sollen.

7.2.3 Erweiterte Rechte auf Datensätzen

In der aktuellen Ausbaustufe erlaubt das Modell keine Unterscheidung, welcher Benutzer mit einem Datensatz welche Aktivitäten ausführen darf. Befindet sich ein Mitarbeiter in der richtigen Organisationseinheit, so hat er vollen lesenden sowie schreibenden Zugriff auf den Datensatz.

Jedoch besteht die Möglichkeit, die Tabellen im Assoziations-Schema, welche die Verknüpfung eines Datensatzes zur jeweiligen urhebenden Organisationseinheit beinhalten, um eine zusätzliche Spalte zu erweitern. Innerhalb dieser Spalte könnte vermerkt werden, dass ein Zugriff auf einen Datensatz nur durch Mitarbeiter der urhebenden Organisationseinheit zulässig ist. So könnte innerhalb der jeweiligen abstrahierenden View die Sichtbarkeit eines Datensatzes aufgrund der gesetzten Attribute, trotz transitiver Zugehörigkeit zu einer Organisationseinheit, eingeschränkt werden. Dies würde es mit vergleichsweise geringem Aufwand ermöglichen, den jeweiligen übergeordneten Führungskräften einen sensiblen Datensatz trotz übergeordneter Weisungsbefugnis vorzuenthalten. Im Anhang 4 befindet sich bereits ein erster Prototyp einer modifizierten *EMPLOYEE_OU*-View, welche eine Unterscheidung nach dem Typ der Assoziation des Mitarbeiters zur jeweiligen Organisationseinheit ermöglicht. Hier wird bereits zwischen P = Primär-Assoziation, S = (Direkte) Sekundär-Assoziation und T = Transitiv-Assoziation unterschieden, was zum aktuellen Zeitpunkt jedoch im Modell noch keine Verwendung findet. Der Einsatz einer derartig modifizierten View müsste zuvor eingehend auf seine Auswirkungen auf das Modell untersucht werden.

Auch könnte man die Assoziations-Tabellen dazu nutzen, ausführende Aktionen fein granularer über die bestehenden *INSTEAD-OF* Trigger zu steuern und Berechtigungen wie *UPDATE* oder *DELETE*, abhängig von der Zugehörigkeit zu bestimmten Organisationeinheiten, einzuschränken oder zuzulassen. Unterschiedliche Filter-Möglichkeiten von Datensätzen nach dem Blacklist- oder Whitelist-Prinzip wären ebenfalls, bezogen auf Mitarbeiter als auch auf Organisationseinheiten, denkbar und könnten innerhalb der Views umgesetzt werden.

7.2.4 Audit über Trigger

Die *INSTEAD-OF* Trigger lassen sich dahingehend erweitern, eigene Mechanismen zur Auditierung der Datenbank einzubauen. So lässt sich erkennen, welcher Mitarbeiter zu welchem Zeitpunkt einen Datensatz verändert oder gelöscht hat, sollte dies für das jeweilige Szenario relevant erscheinen. Dazu muss im Assoziations-Schema je geschützter Tabelle eine Audit-Tabelle angelegt werden, die bei jeder Datenmanipulation einen Verweis auf den ausführenden Mitarbeiter sowie die betroffenen Daten speichert.

7.2.5 Performance

Während der Messungen in Kapitel 5.4 hat sich gezeigt, dass ein Zugriff, welcher ein indiziertes Feld als Filter-Attribut wie beispielsweise den Primärschlüssel beinhaltet, direkt durch die abstrahierende View weitergereicht und vom dahinter liegenden Index bearbeitet wird. Dies macht Zugriffe auf solche indizierten Attribute trotz der darüber liegenden logischen Schicht performant, wie bereits von relationalen Datenbanken bei Nutzung eines Index hinreichend bekannt sein sollte. Allerdings hat sich vor allem bei der Oracle Database gezeigt, dass Abfragen, welche ausschließlich die *EID* des Mitarbeiters als Filterkriterium beinhalten, eine fast lineare Steigerung der Ausführungskosten nach sich ziehen. Es greift somit kein Index, welcher den Zugriff beschleunigen könnte. Dabei muss man berücksichtigen, dass Szenarien in denen ein Anwender nach dem für ihn sichtbaren Datenbestand fragt, keine Seltenheit darstellt. Diese Situation ist für größere Datenbestände nicht vertretbar und wird auf absehbare Zeit die Performance der Datenbank auf ein unzumutbares Niveau reduzieren. Die Ursache hierfür liegt darin, dass die *EID* eines Mitarbeiters ausschließlich zur Ermittlung der zugeordneten Organisationseinheiten verwendet wird und selbst aus der *EMPLOYEE_OU*-View kommt, welche bei jeder Verwendung neu generiert werden muss.

Eine mögliche Lösung für dieses Problem liegt darin, anstatt einer volatilen View, die bei jedem Aufruf neu erzeugt wird, eine sogenannte Materialized View zu verwenden. Dabei handelt es sich um eine spezielle Art von View, welche das Ergebnis der View physikalisch als Kopie auf der Festplatte ablegt und sich bei einem Zugriff nicht erneut generieren muss. Üblicherweise finden solche Materialized Views in Modellen Verwendung, in denen die Berechnung der View selbst viel Zeit in Anspruch nimmt. In dem hier behandelten Szenario spielt die Zeit zur Berechnung der View eine untergeordnete Rolle, doch da jeder Zugriff auf geschützte Daten über sie laufen muss, summiert sich die aufgewendete Rechenzeit. Eine Materialized View hat an dieser Stelle einen nicht zu unterschätzenden Vorteil: Sie erlaubt das Erzeugen von Indizes, welche den Zugriff auf die in der View zwischengespeicherten Ergebnisse stark beschleunigen können. Ein erster Prototyp für eine solche Materialized View inklusive eine kurze Gegenüberstellung der Messwerte befinden sich im Anhang 11, 12 und 13 dieser Arbeit.

Als Ersatz für eine Materialized View kann jedoch auch ein Konstrukt aus einer separaten Tabelle und dazugehörigen Triggern dienen. Diese Tabelle würde die Aufgabe der *EMPLOYEE_OU* View übernehmen, während die Trigger auf den übrigen Tabellen dafür sorgen, dass eine Löschung und vollständige Neuberechnung der *EMPLOYEE_OU* Tabelleninhalte bei Änderung der zugrungeliegenden Tabellen *EMPLOYEE*, *ORGANIZATIONUNIT* oder *ASSOCIATION* angestoßen wird. Dieses Vorgehen bietet sich an, wenn das verwendete Datenbanksystem keine Materialized Views unterstützt oder sich der Umgang mit diesen als unhandlich herausstellen sollte.

Eine weitere Möglichkeit zur Beschleunigung des Modells besteht in der Überarbeitung der Assoziations-Tabellen, welche die Verknüpfung zwischen jedem Datensatz und seiner Eigentümer-Organisationseinheit vorhalten. Diese werden bekanntlich bei jedem Zugriff auf die geschützten fachlichen Tabellen zum Abgleich der Berechtigungen verwendet und können somit durchaus als Flaschenhals im Modell betrachtet werden. Eine denkbare Möglichkeit zur Beschleunigung dieser Tabellen besteht darin, diese als In-Memory-Tabellen zu erstellen, welche vollständig im Hauptspeicher der jeweiligen Datenbank vorgehalten werden. Diese Variante bietet sich jedoch nur für Assoziations-Tabellen an, welche aufgrund der Verwendung und Struktur des fachlichen Datenmodells nicht zu einer zu hohen Anzahl von Einträgen tendieren. Aus diesem Grund sollte ein Einsatz von In-Memory-Tabellen vorher gut überlegt sein.

Sollte sich eine Oracle Database im Einsatz befinden, besteht die Möglichkeit, die Assoziations-Tabellen als eigene sogenannte *IOTs* anzulegen. Dabei handelt es sich nicht um das sogenannte Internet-of-Things sondern um *Index-Organized Tables*. [68] Diese spezielle Form einer Tabelle wird in ihrer Struktur wie ein Index abgelegt. Eine *IOT* lässt sich jedoch auch als ein Index beschreiben, der sich wie eine Tabelle verhält. Der wesentliche Unterschied zu einer normalen Tabelle und einem darauf aufsetzenden Index ist der, dass innerhalb eines normalen Index jeder Datensatz der indizierten Spalte als Kopie mit einem Verweis auf die referenzierte Zeile der jeweiligen Tabelle vorgehalten wird. Bei einer *Index-Organized Table* ist der komplette Datensatz innerhalb des Index abgelegt, was den zusätzlichen Verweis auf den Eintrag in einer Ziel-Tabelle unnötig macht und den Zugriff auf den Datensatz beschleunigt. Solche Konstrukte eignen sich jedoch nicht für Tabellen mit vielen Spalten oder größeren Datenmengen, da ein Index regelmäßig reorganisiert und aktualisiert werden muss. Für die Assoziations-Tabellen müsste der Einsatz dieser Technologie geprüft werden. Sollte der zu verwaltende Datenbestand um diverse Größenordnungen steigen, bietet sich auch der Einsatz von Partitionierung an, sowohl für Indizes als auch Tabellen. Dabei handelt es sich, einfach ausgedrückt, um das Separieren von Datensätzen anhand zuvor definierter Kriterien. Die Tabelle und ihr Index sehen nach außen unverändert aus, doch die Datenbank unterteilt diese in logisch voneinander getrennte Segmente. So lassen sich beispielsweise Abfragen beschleunigen, wenn anhand der Filterkriterien bereits erkennbar ist, dass die gesuchten Werte sich nur in einem der Segmente befinden können und die übrigen Segmente nicht durchsucht werden müssen. Ebenfalls eine naheliegende, aber häufig unterschätzte Optimierungsmöglichkeit betrifft die physikalische Ablage der Informationen. Innerhalb dieser Arbeit wurden für jedes Schema zwei eigene physikalisch getrennte Tablespaces definiert. Es besteht bei der Oracle Database die Möglichkeit, bestimmte Tabellen oder einzelne Indizes zielgerichtet auf dedizierten Tablespaces abzulegen. So lässt sich beispielsweise die Fragmentierung eines Tablespaces verringern, welche zwangsläufig entsteht, wenn sich unterschiedliche Tabellen den gleichen Speicherplatz teilen.

7.2.6 Alternative zu MariaDB

Da die Datenbank MariaDB die erforderlichen Techniken wie *INSTEAD-OF* Trigger leider nicht unterstützt, gilt es zu prüfen, ob sich das hier beschriebene Modell auch in alternativen Systemen umsetzen lässt. Bei einer ersten Recherche zu den ebenfalls freien Datenbanksystemen PostgreSQL in der Version 12 sowie SQLite 3 ist aufgefallen, dass diese grundsätzlich alle benötigten Technologien mit sich bringen.

So unterstützt PostgreSQL v12 die bei MariaDB fehlenden *INSTEAD-OF* Trigger [60]. Trigger können hier wahlweise, im Gegensatz zum Microsoft SQL-Server, für jede Zeile aufgerufen werden oder für ein einzelnes Statement. Auch ist eine Unterscheidung zwischen *BEFORE*- und *AFTER*-Triggern möglich, was bei dem SQL-Server ebenfalls fehlt. Außerdem unterstützt PostgreSQL rekursive *Common Table Expressions* [61], welche die Grundvoraussetzung für die Arbeit mit hierarchischen Strukturen darstellt. Eine Implementierung des Schema-Konzepts zur Kapselung der Daten [62] sowie die Erstellung von *Materialized-Views* [63] zur Behebung der in Kapitel 7.2.5 angesprochenen Performanceprobleme bringt das System ebenfalls mit. Hier lohnt sich eine genauere Analyse und ein erster praktischer Versuch.

Auch SQLite in der Version 3 scheint die notwendigen Voraussetzungen für die Implementierung des hier erarbeiteten Modells mit sich zu bringen. Ebenso wie PostgreSQL unterstützt SQLite *INSTEAD-OF* Trigger [64] sowie rekursive definierte *Common Table Expressions* [65]. Ein Schema-Konzept wird aufgrund der eigentlichen Zielsetzung von SQLite nur über das Einbinden einer weiteren SQLite-Datenbank unterstützt [66]. Über diesen Umstand sowie die nicht unterstützten Session-Variablen kann man jedoch hinwegsehen, wenn man berücksichtigt, dass sich SQLite als Embedded-Datenbank für den Single-User Betrieb ohne Server-Infrastruktur versteht. Auch wenn SQLite keine *Materialized-Views* unterstützt, besteht noch immer die Möglichkeit, eine View über eine Kombination aus Triggern und einer Tabelle, wie in Kapitel 7.2.5 beschrieben, zu ersetzen.

7.3 Forensische Betrachtung

Der Forensische Aspekt soll an dieser Stelle auch Berücksichtigung finden, doch eine umfassende Betrachtung dieser komplexen Thematik kann an dieser Stelle nicht durchgeführt werden. Die Datenbanksysteme selbst bieten zwar Funktionen zur Auditierung, doch diese werden hier nicht beleuchtet. Weiterführende Informationen finden sich unter: MariaDB [90], SQL-Server [91], Oracle Database [92]

In Kapitel 7.2.4 wird kurz auf die Möglichkeiten zur eigenen Auditierung eingegangen. Die dort angesprochenen Maßnahmen würden durchaus dazu beitragen, mögliche Manipulation von Datensätzen im Nachgang aufklären zu können. Dazu bedarf es jedoch einer Anpassung der *INSTEAD-OF* Trigger, um diese Daten zur Verfügung zu haben. Den lesenden Zugriff zu auditieren gestaltet sich jedoch etwas komplizierter. Dazu bedarf es, nachfolgenden am Beispiel der Oracle Database dargestellt, eines kleinen Umbaus der entsprechenden View aus Tabelle 20 bzw. aus Anhang 6:

```
-- Ausführen als MT_ASS_ADMIN
-- Audit-Tabelle Erstellen (unvollständig wg. fehlendem PK + Trigger + Sequence)
CREATE TABLE "CLIENT_AUDIT" (
    EID INT NOT NULL,
    CID INT NOT NULL,
    ACTION_TYPE VARCHAR (16),
    DT TIMESTAMP
);
-- Funktion zum Auditieren der Client-View
CREATE FUNCTION CLIENT_AUDIT_FKT (IN_EID IN NUMBER, IN_CID IN NUMBER)
RETURN NUMBER IS
    PRAGMA AUTONOMOUS_TRANSACTION;
BEGIN
    INSERT INTO CLIENT_AUDIT (EID, CID, ACTION_TYPE, DT)
    VALUES (IN_EID, IN_CID, 'SELECT', CURRENT_TIMESTAMP);
    COMMIT;
    RETURN 1;
END;
-- Angepasste View zum Durchführen eines Audits bei Lese-Zugriffen
CREATE VIEW CLIENT
AS
    SELECT * FROM (
        -- Alle Daten lesen...
        SELECT c.*, e.EID FROM MT_DATA_ADMIN.CLIENT c
        -- ...die eine Verknüfung in der Association-Tabelle haben...
        INNER JOIN A_CLIENT a ON a.ID = c.ID
        -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
        INNER JOIN MT_ORG_ADMIN.EMPLOYEE_OU e ON e.OUID = a.OUID
    ) TBL
    WHERE CLIENT_AUDIT_FKT (TBL."ID", TBL."EID") = 1;
```

Tabelle 60: Oracle – Auditierung einer View bei lesendem Zugriff

Es wird an dieser Stelle darauf hingewiesen, dass sich dieser hier dargestellte Quellcode bewusst nicht im Anhang dieser Arbeit befindet und ein erster erfolgreicher Test ausschließlich für die Oracle Database durchgeführt wurde. Außerdem wurde bewusst, zugunsten der Lesbarkeit, auf die Erstellung eines eigenen Primärschlüssels mit zugehörigem Trigger und Sequence für die Audit-Tabelle verzichtet.

Wie dargestellt, wird eine Tabelle zum Speichern der Audit-Einträge erstellt. Außerdem wird eine erstellte Funktion erstellt, welche als Parameter den aufrufenden Mitarbeiter sowie die ID des betroffenen Datensatzes übergeben bekommt. Beide übergebenen Werte werden inklusive Zeitstempel und Operation in die Audit-Tabelle eingetragen.

Innerhalb der hier geänderten View wird nun für jeden bereits über die EID gefiltert ausgegebenen Datensatz die WHERE-Bedingung zur Validierung aufgerufen. Die Funktion gibt jedoch immer den statischen Rückgabewert Wert 1 zurück, was hier genau so gewollt ist. Wird nun die View aufgerufen, landet jeder gelesene Datensatz unter Angabe eines Zeitstempels in der Audit-Tabelle, wo er ausgelesen werden kann.

```
SELECT * FROM CLIENT WHERE EID = 21;
--------------------
|  ID | NAME | EID |
--------------------
|   2 | C21  |  21 |
|   4 | C41  |  21 |
|   8 | C81  |  21 |

SELECT * FROM CLIENT_AUDIT;
----------------------------------------------
| EID | CID | ACTION_TYPE | DT                |
----------------------------------------------
|  21 |   2 | SELECT      | 06.02.20 18:12:02 |
|  21 |   4 | SELECT      | 06.02.20 18:12:02 |
|  21 |   8 | SELECT      | 06.02.20 18:12:02 |
```

Tabelle 61: Oracle – Auditierung einer View bei lesendem Zugriff (Ergebnis)

Das hier aufgeführte Beispiel soll zeigen, dass eine solche Audit-Funktion auf Views durchaus funktionieren kann. Grundsätzlich sollte aus Performance- und Kapazitäts-Gründen von der eigenen Auditierung, wie hier dargestellt, abgesehen werden, da eine solche Audit-Tabelle durchaus schnell anwachsen kann und die Auswirkungen auf die Abfragegeschwindigkeit im Rahmen dieser Arbeit nicht untersucht wurden.

Für geschützte Tabellen mit wenigen aber dafür sehr schützenswerten Datensätzen mag sich dieses Vorgehen jedoch durchaus anbieten. Auch besteht die Möglichkeit, um die Größe der Audit-Tabelle zu beschränken, nur bestimmte Abfragen in die Audit-Tabelle zu schreiben. Am Beispiel des *Rule-Based Access Control* Modells aus Kapitel 2.6 könnte man das Protokollieren von lesenden Anfragen über die Logik innerhalb der Audit-Funktion auf eine Uhrzeit beschränken, die abseits der regulären Arbeitszeit liegt. So ließen sich Anomalien protokollieren. In solchen Fällen könnte die Funktion auch statt dem statischen Wert 1 einen abweichenden Wert 0 zurück liefern, was die Anzeige von Datensätzen für den aufrufenden Benutzer komplett verhindern würde. An dieser Stelle sei zuletzt noch darauf verwiesen, dass wir hier auf die korrekte Funktionsweise des Optimizers angewiesen sind und er alle Datensätze bereits auf die *EID* des Mitarbeiters gefiltert hat, bevor die Audit-Funktion aufgerufen wird. Sollte dem jedoch nicht so sein, entstehen Werte, die nicht den realen Gegebenheiten entsprechen und unbrauchbar sind.

7.4 Zusammenfassung

Diese Arbeit hat gezeigt, dass die Modellierung und Nutzung von hierarchischen Modellen, unter Verwendung relationaler Datenbanken, möglich ist. Aller drei untersuchten Datenbanksysteme bieten die hierfür notwendigen Funktionen. Selbst größere Datenbestände lassen sich vergleichsweise performant mit SQL-Statements abrufen und bearbeiten.

Von den drei ausgewählten Datenbanksystemen bieten jedoch nur zwei die technischen Möglichkeiten, um eine Zugriffsschicht auf Datensatzebene unter Verwendung von Views umzusetzen. Die Zukunft lässt hoffen, dass die in MariaDB noch fehlenden Funktionen bald nachgerüstet werden, um auch dieses System mit dem hier entwickelten Modell nutzen zu können. So existiert seit Februar 2017 unter der Nummer MDEV-12151 eine Anforderung zur Implementierung von *INSTEAD-OF* Triggern [73]. Grundsätzlich erlaubt auch MariaDB die Arbeit mit hierarchischen Strukturen, doch aufgrund seiner fehlenden Unterstützung für *INSTEAD-OF* Trigger war eine Implementierung des hier entwickelten Modells, unter Berücksichtigung der Anforderungen, nicht möglich.

Im direkten Vergleich zwischen der Oracle Database und dem SQL-Server hat sich gezeigt, dass die Oracle Database wesentlich mehr Funktionen zur Modellierung von Geschäftslogik mitbringt, als der SQL-Server. Auch bringt das Datenbanksystem eine feinere Berechtigungsstruktur mit sich, welche die saubere Kapselung von Tabellen und den darin gespeicherten Datensätzen ermöglicht.

Der SQL-Server hingegen hat sich durch seine klaren Datentypen, die geringere Speicherbelegung sowie die gute Abfrageperformance hervorgetan. Auch ist positiv zu werten, dass er ausreichende Funktionen zum Abrufen generierte Primärschlüssel mitbringt, was bei der Oracle Database nur durch einen Workaround zu bewerkstelligen wäre. Die eingeschränkte Unterstützung von Triggern sowie die fehlende Möglichkeit, Views in einem anderen Benutzerkontext auszuführen, sind jedoch durchaus störend.

Mit Rückblick auf Kapitel 2.6 zeigt sich, dass *RBAC* dem hier entwickelten Modell am nächsten kommt. Jedoch ließ sich keines der dort genannten Modelle direkt nutzen, da die bisherigen Forschungsergebnisse eher konzeptioneller Natur sind. Die Datenbanken unterstützen RBAC zwar, jedoch nur auf Basis der Objekte innerhalb der Datenbank, nicht auf Datensatz- bzw. Zeilenebene, was die Anforderung dieser Arbeit war.

Abschließend lässt sich sagen, dass keines der ausgewählten Datenbanksysteme die gestellten Anforderungen zur vollen Zufriedenheit erfüllt. Jedes der Systeme hat seine Stärken und seine Schwächen. Welches Datenbanksystem man für die Nutzung dieses Modells auswählt, liegt bei den persönlichen Präferenzen eines jeden Softwareentwicklers und seiner Bereitschaft, Kompromisse einzugehen.

Literaturverzeichnis

[1] BUNDESAMT FÜR SICHERHEIT IN DER INFORMATIONSTECHNIK (BSI) : IT-Grundschutz-Katalog : *B1.18 Identitäts- und Berechtigungsmanagement*, https://download.gsb.bund.de/BSI/ITGSK/IT-Grundschutz-Kataloge_2016_EL15_DE.pdf, Abgerufen 02.10.2019

[2] BUNDESANSTALT FÜR FINANZDIENSTLEISTUNGSAUFSICHT (BAFIN) : Rundschreiben 09/2017 (BA) Mindestanforderungen an das Risikomanagement – MaRisk, https://www.bafin.de/SharedDocs/Veroeffentlichungen/DE/Rundschreiben/2017/rs_1709_marisk_ba.html, Abgerufen 02.10.2019

[3] HENDRIK ARNDT : Integrierte Informationsarchitektur - Die erfolgreiche Konzeption professioneller Websites, Springer Verlag, 2006, S. 137ff

[4] I'M PROGRAMMER : Top 10 Databases You Should Learn in 2019, https://www.improgrammer.net/top-10-databases-should-learn-2015/, Abgerufen 23.01.2020

[5] STATISTICA GMBH : Marktanteile der führenden Anbieter 2016, https://de.statista.com/statistik/daten/studie/150807/umfrage/marktanteile-der-software-anbieter-mit-datenbanksystemen/, Abgerufen 23.01.2020

[6] SOLIDIT CONSULTING & SOFTWARE DEVELOPMENT GMBH : DB-Engines Ranking von Relational DBMS, https://db-engines.com/de/ranking/relational+dbms, Abgerufen 23.01.2020

[7] ORACLE CORPORATION : Brief History of Oracle Database, https://docs.oracle.com/database/121/CNCPT/intro.htm#CNCPT88784, Abgerufen 23.01.2020

[8] ORACLE CORPORATION : Oracle Application Express (APEX): Overview, https://www.oracle.com/technetwork/developer-tools/apex/overview/apex-overview-otn-4491378.pdf, Abgerufen 23.01.2020

[9] ORACLE CORPORATION : How to Improve the Efficiency and Performance of an SAP Environment - With Oracle Optimized Solution for SAP, https://www.oracle.com/technetwork/server-storage/hardware-solutions/oos-sap-efficiency-performance-1849692.pdf, Abgerufen 23.01.2020

[10] ORACLE CORPORATION : Database Embedded JVM (OJVM), https://www.oracle.com/de/database/technologies/appdev/ojvm.html, Abgerufen 23.01.2020

[11] ORACLE CORPORATION : PL/SQL, https://www.oracle.com/de/database/technologies/appdev/plsql.html, Abgerufen 23.01.2020

[12] ORACLE CORPORATION : Oracle Real Application Clusters (RAC), https://www.oracle.com/de/database/technologies/rac.html, Abgerufen 23.01.2020

[13] ORACLE CORPORATION : Oracle Enterprise Manager, https://www.oracle.com/de/enterprise-manager/, Abgerufen 23.01.2020

[14] ORACLE CORPORATION : Oracle Exadata Database Machine X8M-8, https://www.oracle.com/de/a/ocom/docs/engineered-systems/exadata/exadata-x8m-8-ds.pdf, Abgerufen 23.01.2020

[15] MICROSOFT CORPORATION : Building the Billion Dollar Database: Microsoft SQL Server Climbs to New Heights, Erschienen 28.06.2001, https://news.microsoft.com/2001/06/28/building-the-billion-dollar-database-microsoft-sql-server-climbs-to-new-heights/, Abgerufen 23.01.2020

[16] MICROSOFT CORPORATION : SQL Server unter Linux, https://docs.microsoft.com/de-de/sql/linux/sql-server-linux-overview?view=sql-server-2017, Abgerufen 23.01.2020

[17] MICROSOFT CORPORATION : Lizenzierung von SQL Server, https://www.microsoft.com/de-de/sql-server/sql-server-2017-pricing, Abgerufen 23.01.2020

[18] ORACLE CORPORATION : Oracle Technology Global Price List - December 5, 2019, https://www.oracle.com/assets/technology-price-list-070617.pdf, Abgerufen 23.01.2020

[19] TECHTARGET INC. : Oracle versus SQL Server: Microsoft SQL billiger und einfacher als Oracle, https://www.computerweekly.com/de/meinung/Oracle-versus-SQL-Server-Microsoft-SQL-billiger-und-einfacher-als-Oracle-DB, Abgerufen 23.01.2020

[20] VOGEL COMMUNICATIONS GROUP : MariaDB – die MySQL-Konkurrenz schlüpft unter ein einziges Dach, https://www.datacenter-insider.de/mariadb-die-mysql-konkurrenz-schluepft-unter-ein-einziges-dach-a-462828/, Abgerufen 23.01.2020

[21] 1&1 IONOS SE : MariaDB vs. MySQL, https://www.ionos.de/digitalguide/hosting/hosting-technik/mariadb-vs-mysql/, Abgerufen 23.01.2020

[22] IDG BUSINESS MEDIA GMBH : EU genehmigt Oracle/Sun-Übernahme, https://www.channelpartner.de/a/update-eu-genehmigt-oracle-sun-uebernahme,286691, Abgerufen 23.01.2020

[23] CBS INTERACTIVE INC. : Datenbanken: Google steigt von MySQL auf MariaDB um, https://www.zdnet.de/88169832/datenbanken-google-steigt-von-mysql-auf-mariadb-um/, Abgerufen 23.01.2020

[24] BAADER & LINDNER GBR : Red Hat steigt mit RHEL 7 auf MariaDB um, https://www.pro-linux.de/news/1/19902/red-hat-steigt-mit-rhel-7-auf-mariadb-um.html, Abgerufen 23.01.2020

[25] WIKIMEDIA FOUNDATION : Wikipedia Adopts MariaDB, https://blog.wikimedia.org/2013/04/22/wikipedia-adopts-mariadb/, Abgerufen 23.01.2020

[26] MARIADB CORPORATION: MariaDB versus MySQL – Kompatibilität, https://mariadb.com/kb/de/mariadb-vs-mysql-compatibility/, Abgerufen 23.01.2020

[27] CHRISTINE WOLFINGER, JÜRGEN GULBINS, CARSTEN HAMMER : Linux Systemadministration, Springer Verlag, 2005, S. 13

[28] MARTIN DIETZFELBINGER, KURTH MEHLHORN, PETER SANDERS : Algorithmen und Datenstrukturen, Springer Verlag, 2014, S. 62ff

[29] STEFAN HOUGARDY, JENS VYGEN : Algorithmische Mathematik, Springer Verlag, 2018, S. 100ff

[30] SVEN OLIVER KRUMKE, HARTMUT NOLTEMEIER : Graphentheoretische Konzepte und Algorithmen, Springer Verlag, 2012, Kapitel 2.1, S. 7

[31] ANGELIKA STEGER : Diskrete Strukturen – Band 1: Kombinatorik Graphentheorie Algebra, Springer Verlag, 2. Auflage, 2007, Kapitel 2.1, S. 61

[32] MARIADB CORPORATION: MariaDB 10.4.6 Official Release Notes, https://mariadb.com/kb/en/library/mariadb-1046-release-notes/, Abgerufen 29.10.2019

[33] MICROSOFT CORPORATION : Sicherheit auf Zeilenebene, https://docs.microsoft.com/de-de/sql/relational-databases/security/row-level-security?view=sql-server-2017, Abgerufen 16.11.2019

[34] JIE SHI, HONG ZHU : A fine-grained access control model for relational databases, Huazhong University of Science and Technology, China, Springer Verlag, 2010

[35] CHRISTIAN PRIETZ : Aspekte der Datensicherheit und des Datenschutzes in Datenbanksystemen : DuD – Datenschustz und Datensicherheit 31, Springer Verlag, 2007, S. 894-898

[36] ORACLE CORPORATION : Oracle Database 12c – Real Application Security, White Paper 2014, https://www.oracle.com/technetwork/database/security/real-application-security/wp-security-ras12c-2312936.pdf, Abgerufen 16.11.2019

[37] HEINZ-WILHELM FABRY, ORACLE DEUTSCHLAND GMBH : Oracle Virtual Private Database, Vortragsunterlagen Deutsche Oracle-Anwendergruppe (DOAG) vom 07.03.2005, https://www.doag.org/formes/pubfiles/59384/, Abgerufen 17.11.2019

[38] MATHIAS WEBER, MARKUS GEIS : Institut für Notfallmedizin und Medizinmanagement, Klinikum Universität München, Artikel Deutsche Oracle-Anwendergruppe (DOAG) vom 27.08.2014, https://www.doag.org/formes/pubfiles/6171545/, Abgerufen 17.11.2019

[39] ORACLE CORPORATION : Oracle Database Security Guide 18c, https://docs.oracle.com/en/database/oracle/oracle-database/18/dbseg/database-security-guide.pdf, Kapitel 11-1, S. 478ff, Abgerufen 17.11.2019

[40] PATRICIA HUEY, SUMIT JELOKA, ORACLE CORPORATION : Oracle Database Security Guide 18c, https://docs.oracle.com/en/database/oracle/oracle-database/18/dbseg/database-security-guide.pdf, Kapitel 11-48, S. 525ff, Abgerufen 17.11.2019

[41] ORACLE CORPORATION : Oracle Label Security, Technical White Paper 2018, https://www.oracle.com/technetwork/wp-dbsec-ols-201702-3634252.pdf, Abgerufen 17.11.2019

[42] ULF LÄMMERHIRT, ORACLE DEUTSCHLAND B.V. & CO. KG : Mandantentrennung von der Anwendung bis zum Storage, Weblogic Server und OLS, Artikel Deutsche Oracle-Anwendergruppe (DOAG) vom 10.12.2013, https://www.doag.org/formes/pubfiles/5308969/, Abgerufen 17.11.2019

[43] HEINZ-WILHELM FABRY, ORACLE DEUTSCHLAND B.V & CO. KG : Mandatory Access Control mit Oracle Label Security, Artikel Deutsche Oracle-Anwendergruppe (DOAG) vom 20.11.2013, https://www.doag.org/formes/pubfiles/5263842/, Abgerufen 16.11.2019

[44] ORACLE CORPORATION : Database Real Application Security Administrator's and Developer's Guide, 12c Release 1(12.1), März 2017, https://docs.oracle.com/database/121/DBFSG/toc.htm, Abgerufen 17.11.2019

[45] DR. GÜNTER UNBESCHEID, DATABASE CONSULT GMBH : 12c Real Application Security, Concepts, Techniques, Use Cases, Präsentation Deutsche Oracle-Anwendergruppe (DOAG) vom 16.11.2016, https://www.doag.org/formes/pubfiles/8573261/, Abgerufen 17.11.2019

[46] MARIADB CORPORATION: CREATE DATABASE – MariaDB Knowledge Base, https://mariadb.com/kb/en/library/create-database/, Abgerufen 21.12.2019

[47] MICROSOFT CORPORATION : Erstellen eines Datenbankschemas, https://docs.microsoft.com/de-de/sql/relational-databases/security/authentication-access/create-a-database-schema?view=sql-server-2017, Abgerufen 21.12.2019

[48] MICROSOFT CORPORATION : Objektbesitz und Trennung von Benutzer und Schema in SQL Server, https://docs.microsoft.com/de-de/dotnet/framework/data/adonet/sql/ownership-and-user-schema-separation-in-sql-server, Abgerufen 21.12.2019

[49] ORACLE CORPORATION : Schema Objects, https://docs.oracle.com/cd/B19306_01/server.102/b14220/schema.htm, Abgerufen 21.12.2019

[50] ORACLE CORPORATION : DROP USER, https://docs.oracle.com/cd/B19306_01/server.102/b14200/statements_9008.htm, Abgerufen 21.12.2019

[51] LARS E. OLSON, CARL A. GUNTER, WILLIAM R. COOK, MARIANNE WINSLETT : Implementing Reflective Access Control in SQL, University of Illinois at Urbana-Champaign, University of Texas, erschienen in Data and Applications Security XXIII, 2009, Springer Verlag, S. 21

[52] E. F. CODD : The Relational Model for Database Management : Version 2, erschienen 1990, Addison-Wesley Verlag, Kapitel 17 - View Updatability, S. 293ff

[53] ANDREAS KELZ : Kriterien für relationale Datenbanksysteme, Hochschule der Medien Stuttgart, https://www.hdm-stuttgart.de/~riekert/lehre/db-kelz/chap6.htm, Abgerufen 22.12.2019

[54] ORACLE CORPORATION : CREATE TRIGGER Statement, https://docs.oracle.com/database/121/LNPLS/create_trigger.htm#LNPLS01374, Abgerufen 22.12.2019

[55] MICROSOFT CORPORATION : CREATE TRIGGER (Transact-SQL), https://docs.microsoft.com/de-de/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-2017, Abgerufen 22.12.2019

[56] MARIADB CORPORATION: CREATE TRIGGER – MariaDB Knowledge Base, https://mariadb.com/kb/en/library/create-trigger/, Abgerufen 21.12.2019

[57] JIRA BUGTRACKER, MARIADB SERVER : Syntax error on SELECT INTO @variable with CTE, https://jira.mariadb.org/browse/MDEV-20730, Abgerufen 27.12.2019

[58] ORACLE CORPORATION : Database SQL Tuning Guide, Query Optimizer Concepts, https://docs.oracle.com/database/121/TGSQL/tgsql_optcncpt.htm, Abgerufen 18.01.2020

[59] ORACLE CORPORATION : Database SQL Tuning Guide, Generating and Displaying Execution Plans, https://docs.oracle.com/database/121/TGSQL/tgsql_genplan.htm, Abgerufen 18.01.2020

[60] POSTGRESQL GLOBAL DEVELOPMENT GROUP : Overview of Trigger Behavior, https://www.postgresql.org/docs/12/trigger-definition.html, Abgerufen 23.01.2020

[61] POSTGRESQL GLOBAL DEVELOPMENT GROUP : WITH Queries (Common Table Expressions), https://www.postgresql.org/docs/12/queries-with.html, Abgerufen 23.01.2020

[62] POSTGRESQL GLOBAL DEVELOPMENT GROUP : Schemas, https://www.postgresql.org/docs/12/ddl-schemas.html, Abgerufen 23.01.2020

[63] POSTGRESQL GLOBAL DEVELOPMENT GROUP : CREATE MATERIALIZED VIEW, https://www.postgresql.org/docs/12/sql-creatematerializedview.html, Abgerufen 23.01.2020

[64] SQLITE DEVELOPERS : CREATE TRIGGER, https://www.sqlite.org/lang_createtrigger.html, Abgerufen 23.01.2020

[65] SQLITE DEVELOPERS : WITH clause, https://www.sqlite.org/lang_with.html, Abgerufen 23.01.2020

[66] SQLITE DEVELOPERS : Database Object Name Resolution, https://www.sqlite.org/lang_naming.html, Abgerufen 23.01.2020

[67] ORACLE CORPORATION : Glossary, https://docs.oracle.com/en/database/oracle/oracle-database/18/cncpt/glossary.html, Abgerufen 18.01.2020

[68] ORACLE CORPORATION : Indexes and Index-Organized Tables, https://docs.oracle.com/en/database/oracle/oracle-database/18/cncpt/indexes-and-index-organized-tables.html, Abgerufen 30.01.2020

[69] MICROSOFT CORPORATION : Int, bigint, smallint und tinyint, https://docs.microsoft.com/de-de/sql/t-sql/data-types/int-bigint-smallint-and-tinyint-transact-sql?view=sql-server-2017, Abgerufen 30.01.2020

[70] ORACLE CORPORATION : Data Types, https://docs.oracle.com/en/database/oracle/oracle-database/18/sqlrf/Data-Types.html, Abgerufen 30.01.2020

[71] MICROSOFT CORPORATION : WITH common_table_expression (Transact-SQL), https://docs.microsoft.com/de-de/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-2017, Abgerufen 30.01.2020

[72] GRANT FRITCHEY : SQL-Server Execution Plans – What goes on beneath the surface with your queries, 3 Edition, Erschienen 2018, Redgate Publishing

[73] JIRA BUGTRACKER, MARIADB SERVER : Support for INSTEAD OF triggers, https://jira.mariadb.org/browse/MDEV-12151, Abgerufen 02.02.2020

[74] MARIADB CORPORATION: Server System Variables – MariaDB Knowledge Base, https://mariadb.com/kb/en/server-system-variables/#max_recursive_iterations, Abgerufen 02.02.2020

[75] JIRA BUGTRACKER, MARIADB SERVER : defaut max_recursive_iterations = 4G, https://jira.mariadb.org/browse/MDEV-17239, Abgerufen 02.02.2020

[76] MARIADB CORPORATION: Protect Your Data: Row-level Security in MariaDB 10.0, https://mariadb.com/resources/blog/protect-your-data-row-level-security-in-mariadb-10-0/, Abgerufen 05.02.2020

[77] PIERANGELA SAMARATI, SABRINA DE CAPITANI DI VIMERCATI : Access Control: Policies, Models, and Mechanisms, Universita di Milano Italy, erschienen 2001, Springer Verlag

[78] MATUNDA NYANCHAMA, SYLVIA OSBORN : Modeling Mandatory Access Control in Role-Based Security Systems, Department of Computer Science, University of Western Ontario, Springer Verlag, 1996

[79] SEJONG OH, SOEG PARK : An Integration Model of Role-Based Access Control and Activity-Based Access Control Using Task, Sogang University, Springer Verlag, 2001

[80] UMESH HODEGHATTA RAO, UMESHA NAYAK : The InfoSec Handbook - An Introduction to Information Security, Kapitel 4, Seite 63ff, Springer Verlag, 2014

[81] DAVID ELLIOTT BELL : Looking Back at the Bell-La Padula Model, Applied Computer Security Associates, 2005

[82] MARKUS NENTWIG : Untersuchung von MAC-Implementationen, Universität Leipzig - Fakultaet für Mathematik und Informatik - Institut für Informatik, 2010

[83] RAVI S. SANDHU : Lattice-Based Access Control Models, George Mason University, 1993

[84] QASIM MAHMOOD RAJPOOT, CHRISTIAN DAMSGAARD JENSEN, RAM KRISHNAN : Integrating Attributes into Role-Based Access Control, Technical University of Denmark, Department of Applied Mathematics and Computer Science, 2015

[85] NIHARIKA SINGH, ASHUTOSH KUMAR SINGH : Data Privacy Protection Mechanisms in Cloud, Universiti Teknology PETRONAS - Malaysia, National Institute of Technology - Kurukshetra – India, Springer Verlag, 2018

[86] HUA WANG, ZONGHUA ZHANG, TAREK TALEB : Editorial: Special Issue on Security and Privacy of IoT, Taiyuan Normal University - China, Victoria University - Australia, Institut Mines-Telecom - France, Aalto University - Finland, Springer Verlag, 2017

[87] PIETRO COLOMBO, ELENA FERRARI : Fine-Grained Access Control Within NoSQL Document-Oriented Datastores, University of Insubria - Italy, Springer Verlag, 2016

[88] HUA WANG, YONGZHI WANG, TAREK TALEB, XIAOHONG JIANG : Editorial: Special issue on security and privacy in network computing, Nanjing University of Information Science - China, Institute for Sustainable Industries & Liveable Cities - Australia, Xidian University - China, Aalto University - Finland, Future University Hakodate - Japan, Springer Verlag, 2019

[89] MICROSOFT CORPORATION : SCOPE_IDENTITY (Transact-SQL), https://docs.microsoft.com/de-de/sql/t-sql/functions/scope-identity-transact-sql?view=sql-server-2017, Abgerufen 06.02.2020

[90] MARIADB CORPORATION : MariaDB Audit Plugin, https://mariadb.com/kb/en/mariadb-audit-plugin/, Abgerufen 06.02.2020

[91] MICROSOFT CORPORATION : SQL Server Audit (Datenbank-Engine), https://docs.microsoft.com/de-de/sql/relational-databases/security/auditing/sql-server-audit-database-engine?view=sql-server-2017, Abgerufen 06.02.2020

[92] ORACLE CORPORATION : Auditing Database Activity, https://docs.oracle.com/cd/E11882_01/server.112/e10575/tdpsg_auditing.htm, Abgerufen 06.02.2020

Bilderverzeichnis

Tabellenverzeichnis

Anlagenverzeichnis und Anhang

1. Oracle – Bereinigung

```
-- ============================================
--
-- WICHTIG: Ausführen als SYSTEM / SYS
--
-- Dient der Bereinigung bestehender Strukturen und sollte nur dann ausgeführt werden,
-- wenn bereits Strukturen aus diesem Anhang erzeugt wurden.
--
-- ============================================


-- Lösche "Organisation"
DROP USER MT_ORG_ADMIN CASCADE;
DROP USER MT_ORG_USER CASCADE;
DROP TABLESPACE MT_ORG_TMP INCLUDING CONTENTS AND DATAFILES;
DROP TABLESPACE MT_ORG     INCLUDING CONTENTS AND DATAFILES;

-- Lösche "Assoziation"
DROP USER MT_ASS_ADMIN CASCADE;
DROP USER MT_ASS_USER CASCADE;
DROP TABLESPACE MT_ASS_TMP INCLUDING CONTENTS AND DATAFILES;
DROP TABLESPACE MT_ASS     INCLUDING CONTENTS AND DATAFILES;

-- Lösche "Fachliche Daten"
DROP USER MT_DATA_ADMIN CASCADE;
DROP TABLESPACE MT_DATA_TMP INCLUDING CONTENTS AND DATAFILES;
DROP TABLESPACE MT_DATA     INCLUDING CONTENTS AND DATAFILES;
```

2. Oracle – Erstelle Tablespaces, Benutzer, Logon-Trigger

```
-- ============================================
--
-- WICHTIG: Ausführen als SYSTEM / SYS
--
-- ============================================

-- Erzeuge Tablespace u. Datafiles für "Organisation"
CREATE TABLESPACE MT_ORG
    DATAFILE 'A:\DBs\ORA\MT_ORG.dbf'
    SIZE 32m AUTOEXTEND ON NEXT 32m MAXSIZE 4096m;
CREATE TEMPORARY TABLESPACE MT_ORG_TMP
    TEMPFILE 'A:\DBs\ORA\MT_ORG_TMP.dbf'
    SIZE 32m AUTOEXTEND ON NEXT 32m MAXSIZE 1024m;

-- Erzeuge Tablespace u. Datafiles für "Assoziation"
CREATE TABLESPACE MT_ASS
    DATAFILE 'A:\DBs\ORA\MT_ASS.dbf'
    SIZE 32m AUTOEXTEND ON NEXT 32m MAXSIZE 4096m;
CREATE TEMPORARY TABLESPACE MT_ASS_TMP
    TEMPFILE 'A:\DBs\ORA\MT_ASS_TMP.dbf'
    SIZE 32m AUTOEXTEND ON NEXT 32m MAXSIZE 1024m;

-- Erzeuge Tablespace u. Datafiles für "Fachliche Daten"
CREATE TABLESPACE MT_DATA
    DATAFILE 'A:\DBs\ORA\MT_DATA.dbf'
    SIZE 32m AUTOEXTEND ON NEXT 32m MAXSIZE 4096m;
CREATE TEMPORARY TABLESPACE MT_DATA_TMP
    TEMPFILE 'A:\DBs\ORA\MT_DATA_TMP.dbf'
    SIZE 32m AUTOEXTEND ON NEXT 32m MAXSIZE 1024m;

-- ==========================================================================================

-- Umbiegen des akt. Schemas
CREATE OR REPLACE TRIGGER LOGON_TRG
  AFTER LOGON ON DATABASE
BEGIN
    IF (USER = 'MT_ORG_USER') THEN
        EXECUTE IMMEDIATE 'ALTER SESSION SET CURRENT_SCHEMA = MT_ORG_ADMIN';
    END IF;
    IF (USER = 'MT_ASS_USER') THEN
        EXECUTE IMMEDIATE 'ALTER SESSION SET CURRENT_SCHEMA = MT_ASS_ADMIN';
    END IF;
END LOGON_TRG;
/

-- ==========================================================================================

CREATE USER MT_ORG_ADMIN IDENTIFIED BY "mt_org_admin"
DEFAULT TABLESPACE MT_ORG TEMPORARY TABLESPACE MT_ORG_TMP;

CREATE USER MT_ORG_USER IDENTIFIED BY "mt_org_user"
DEFAULT TABLESPACE MT_ORG TEMPORARY TABLESPACE MT_ORG_TMP;

CREATE USER MT_ASS_ADMIN IDENTIFIED BY "mt_ass_admin"
DEFAULT TABLESPACE MT_ASS TEMPORARY TABLESPACE MT_ASS_TMP;

CREATE USER MT_ASS_USER IDENTIFIED BY "mt_ass_user"
DEFAULT TABLESPACE MT_ASS TEMPORARY TABLESPACE MT_ASS_TMP;

CREATE USER MT_DATA_ADMIN IDENTIFIED BY "mt_data_admin"
DEFAULT TABLESPACE MT_DATA TEMPORARY TABLESPACE MT_DATA_TMP;

-- ==========================================================================================

GRANT CONNECT, RESOURCE, CREATE VIEW, UNLIMITED TABLESPACE TO MT_ORG_ADMIN ;
GRANT CONNECT, RESOURCE, CREATE VIEW, UNLIMITED TABLESPACE TO MT_ASS_ADMIN ;
GRANT CONNECT, RESOURCE, CREATE VIEW, UNLIMITED TABLESPACE TO MT_DATA_ADMIN ;
GRANT CONNECT, UNLIMITED TABLESPACE                        TO MT_ASS_USER ;
GRANT CONNECT, UNLIMITED TABLESPACE                        TO MT_ORG_USER ;
```

3. Oracle – Erstelle Schema – Organisation

```
-- ============================================
--
-- WICHTIG: Ausführen als MT_ORG_ADMIN
--
-- ============================================

-- Tabellen erstellen
CREATE TABLE ORGANIZATIONUNIT (
    ID INT NOT NULL    PRIMARY KEY,
    PARENT INT         REFERENCES ORGANIZATIONUNIT(ID),
    NAME VARCHAR2(128),
    CONSTRAINT OU_NO_REFLEXIVE CHECK (ID <> PARENT)
);
CREATE INDEX ORGANIZATIONUNIT_PARENT_IDX ON ORGANIZATIONUNIT (PARENT);

CREATE TABLE EMPLOYEE (
    ID INT NOT NULL    PRIMARY KEY,
    OU INT NOT NULL    REFERENCES ORGANIZATIONUNIT(ID),
    NAME VARCHAR2(128)
);
CREATE INDEX EMPLOYEE_OU_IDX ON EMPLOYEE (OU);

CREATE TABLE ASSOCIATION (
    EID INT NOT NULL   REFERENCES EMPLOYEE(ID),
    OUID INT NOT NULL  REFERENCES ORGANIZATIONUNIT(ID),
    CONSTRAINT ASSOCIATION_PK PRIMARY KEY (EID, OUID)
);
CREATE INDEX ASSOCIATION_EID_IDX  ON ASSOCIATION (EID);
CREATE INDEX ASSOCIATION_OUID_IDX ON ASSOCIATION (OUID);


-- ==========================================================================================


-- Sequences (für autom. Erzeugung der PKs) erstellen
CREATE SEQUENCE EMPLOYEE_SEQ INCREMENT BY 1 START WITH 1;
CREATE SEQUENCE ORGANIZATIONUNIT_SEQ INCREMENT BY 1 START WITH 1;

-- Trigger für EMPLOYEE (zur Vergabe der PKs) erstellen
CREATE TRIGGER EMPLOYEE_TRG
BEFORE INSERT ON EMPLOYEE FOR EACH ROW
BEGIN
    SELECT EMPLOYEE_SEQ.NEXTVAL INTO :NEW.ID FROM SYS.DUAL;
END;
/

-- Trigger für ORGANIZATIONUNIT (zur Vergabe der PKs) erstellen
CREATE TRIGGER ORGANIZATIONUNIT_TRG
BEFORE INSERT ON ORGANIZATIONUNIT FOR EACH ROW
BEGIN
    SELECT ORGANIZATIONUNIT_SEQ.NEXTVAL INTO :NEW.ID FROM SYS.DUAL;
END;
/
```

```
-- Trigger gegen Zyklen-Bildung bei UPDATE auf ORGANISATION_UNIT
CREATE OR REPLACE TRIGGER OU_UPDATE_TRG
AFTER UPDATE OF "PARENT" ON ORGANIZATIONUNIT
DECLARE
  c NUMBER;
BEGIN
    -- Liste alle OU-IDs dieser und darunter liegenden OUs
    WITH recDat (ID) AS (
        SELECT org.ID as "ID" FROM ORGANIZATIONUNIT org
        UNION ALL
        SELECT org.ID as "ID" FROM ORGANIZATIONUNIT org
        INNER JOIN recDat ON org.PARENT = recDat.ID
    )
    SELECT COUNT(*) INTO c FROM recDat;
EXCEPTION
    WHEN OTHERS THEN
        RAISE_APPLICATION_ERROR(-20667, 'Potentieller Zyklus erkannt!');
END;
/

-- View erstellen
CREATE OR REPLACE VIEW EMPLOYEE_OU
AS
WITH MySelf (S, ID) AS (
    SELECT "S","ID"  FROM (
        -- Startpunkt 1: Primäre OU ermitteln
        SELECT
            e.ID       AS "S",  -- Start E-ID
            org.ID     AS "ID"  -- OU-ID
        FROM EMPLOYEE e
        INNER JOIN ORGANIZATIONUNIT org ON org.ID = e.OU
        UNION ALL
        -- Startpunkt 2: Sekundäre OUs ermitteln
        SELECT
            e.ID       AS "S",   -- Start E-ID
            org.ID     AS "ID"   -- OU-ID
        FROM EMPLOYEE e
        INNER JOIN ASSOCIATION a ON a.EID = e.ID
        INNER JOIN ORGANIZATIONUNIT org ON org.ID = a.OUID
    )
    UNION ALL
    -- Rekursiv darunter liegende Sekundär-OUs ermitteln
    SELECT
        MySelf.S   AS "S",  -- Start E-ID
        org.ID     AS "ID"  -- OU-ID
    FROM ORGANIZATIONUNIT org
    INNER JOIN MySelf ON org.PARENT = MySelf.ID
)
SELECT S as "EID", ID as "OUID" FROM MySelf GROUP BY S, ID;


-- ==========================================================================================


-- Rechte für Nutzung der Tabellen erteilen
GRANT SELECT, INSERT, UPDATE, DELETE ON ORGANIZATIONUNIT TO MT_ORG_USER;
GRANT SELECT, INSERT, UPDATE, DELETE ON EMPLOYEE         TO MT_ORG_USER;
GRANT SELECT, INSERT, UPDATE, DELETE ON ASSOCIATION      TO MT_ORG_USER;
GRANT SELECT                         ON EMPLOYEE_OU      TO MT_ORG_USER;

-- Rechte für lesenden Zugriff erteilen
GRANT SELECT, REFERENCES ON ORGANIZATIONUNIT        TO MT_ASS_ADMIN;
GRANT SELECT, REFERENCES ON EMPLOYEE                TO MT_ASS_ADMIN;

-- Lösung für ORA-01720
GRANT SELECT, INSERT, UPDATE, DELETE ON EMPLOYEE_OU TO MT_ASS_ADMIN WITH GRANT OPTION;
```

4. Oracle – Erstelle Schema – Organisation (View-Erweiterung)

```
-- ============================================
--
-- WICHTIG: Ausführen als MT_ORG_ADMIN
--
-- Die hier enthaltenen Änderungen stellen ein Proof-of-Concept aus Kapitel 7.2.3 dar
-- und dienen als Hilfestellung für mögliche Anpassungen oder Erweiterungen der EMPLOYEE_OU-View.
-- Bitte nicht ohne vorherige Anpassung nutzen, da es sonst zu doppelten Ergebnissen kommt!
--
-- Die View wurde um eine Spalte T erweitert, welche die Herkunft der Assoziation angibt.
-- Erläuterung, siehe Kapitel 7.2.3
--
-- Die Spalte T kann folgende Ausprägungen haben:
--   P: Primär-OU
--   S: Sekundär-OU
--   T: Transitiv-OU
--
-- ============================================


-- Lösche alte View
DROP VIEW EMPLOYEE_OU;


-- Erstelle View mit zusätzlichem Attribut „T“, welches die Herkunft einer OU-Assoziation angibt
CREATE OR REPLACE VIEW EMPLOYEE_OU
AS
WITH MySelf (S, ID, T) AS (
    SELECT "S","ID","T"  FROM (
        -- Startpunkt 1: Primäre OU ermitteln
        SELECT
            e.ID       AS "S",  -- Start E-ID
            org.ID     AS "ID", -- OU-ID
            'P'        AS "T"   -- INFO: PRIMÄR-OU
        FROM employee e
        INNER JOIN organizationunit org ON org.ID = e.OU

        UNION ALL

        -- Startpunkt 2: Sekundäre OUs ermitteln
        SELECT
            e.ID       AS "S",   -- Start E-ID
            org.ID     AS "ID",  -- OU-ID
            'S'        AS "T"    -- INFO: Direkte Sekundär-OUs
        FROM employee e
        INNER JOIN association a ON a.EID = e.ID
        INNER JOIN organizationunit org ON org.ID = a.OUID
    )

    UNION ALL

    -- Rekursiv darunter liegende Sekundär-OUs ermitteln
    SELECT
        MySelf.S   AS "S",  -- Start E-ID
        org.ID     AS "ID", -- OU-ID
        'T'        AS "T"   -- INFO: Transitive Sekundär-OUs
    FROM organizationunit org
    INNER JOIN MySelf ON org.PARENT = MySelf.ID
)
SELECT S as "EID", ID as "OUID", T FROM MySelf GROUP BY S, ID, T;
```

5. Oracle – Erstelle Schema – Fachliches Datenmodell

```
-- ============================================
--
-- WICHTIG: Ausführen als MT_ DATA _ADMIN
--
-- ============================================

-- Tabellen anlegen
CREATE TABLE PRODUCT (
    ID INT NOT NULL      PRIMARY KEY,
    NAME VARCHAR(128)
);
CREATE TABLE CLIENT (
    ID INT NOT NULL      PRIMARY KEY,
    NAME VARCHAR(128)
);
CREATE TABLE "ORDER" (
        ID INT NOT NULL      PRIMARY KEY,
        CID INT              REFERENCES CLIENT (ID),
        ORDER_DATETIME TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX ORDER_CID_IDX  ON "ORDER" (CID);
CREATE TABLE ORDERPRODUCT (
    OID INT NOT NULL     REFERENCES "ORDER" (ID),
    PID INT NOT NULL     REFERENCES PRODUCT (ID),
    AMOUNT INT DEFAULT 1,
    CONSTRAINT ORDERPRODUCT_PK PRIMARY KEY (OID, PID)
);
CREATE INDEX ORDERPRODUCT_OID_IDX  ON ORDERPRODUCT (OID);
CREATE INDEX ORDERPRODUCT_PID_IDX  ON ORDERPRODUCT (PID);

-- Sequences für Auto-Increment erstellen
CREATE SEQUENCE CLIENT_SEQ  INCREMENT BY 1 START WITH 1;
CREATE SEQUENCE PRODUCT_SEQ INCREMENT BY 1 START WITH 1;
CREATE SEQUENCE ORDER_SEQ   INCREMENT BY 1 START WITH 1;

-- Trigger für Auto-Increment erzeugen
CREATE OR REPLACE TRIGGER CLIENT_TRG
BEFORE INSERT ON CLIENT FOR EACH ROW
BEGIN
    SELECT CLIENT_SEQ.NEXTVAL INTO :NEW.ID FROM SYS.DUAL;
END;
/

CREATE OR REPLACE TRIGGER PRODUCT_TRG
BEFORE INSERT ON PRODUCT FOR EACH ROW
BEGIN
    SELECT PRODUCT_SEQ.NEXTVAL INTO :NEW.ID FROM SYS.DUAL;
END;
/

CREATE OR REPLACE TRIGGER ORDER_TRG
BEFORE INSERT ON "ORDER" FOR EACH ROW
BEGIN
    SELECT ORDER_SEQ.NEXTVAL INTO :NEW.ID FROM SYS.DUAL;
END;
/

-- ==========================================================================================

-- Wichtig: "WITH GRANT OPTION" wegen ORA-01720, weil MT_ASS_ADMIN Grant für MT_ASS_USER durchführt
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON CLIENT      TO MT_ASS_ADMIN WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON "ORDER"     TO MT_ASS_ADMIN WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON PRODUCT     TO MT_ASS_ADMIN WITH GRANT OPTION;
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON ORDERPRODUCT TO MT_ASS_ADMIN WITH GRANT OPTION;
```

6. Oracle – Erstelle Schema – Assoziation

```
-- ============================================
--
-- WICHTIG: Ausführen als MT_ASS_ADMIN
--
-- ============================================

-- View für Tabelle ohne Zugriffsschicht erstellen (PRODUCT)
CREATE VIEW PRODUCT
AS
    SELECT * FROM MT_DATA_ADMIN.PRODUCT;

-- View für Tabelle ohne Zugriffsschicht erstellen (ORDERPRODUCT)
CREATE VIEW ORDERPRODUCT
AS
    SELECT * FROM MT_DATA_ADMIN.ORDERPRODUCT;


-- ==========================================================================================


-- Assoziations-Tabelle für fachliche Tabelle erstellen (CLIENT)
CREATE TABLE A_CLIENT (
    ID INT NOT NULL         PRIMARY KEY,
    OUID INT NOT NULL,
    CONSTRAINT CLIENT_OU_FK FOREIGN KEY(OUID) REFERENCES MT_ORG_ADMIN.ORGANIZATIONUNIT(ID),
    CONSTRAINT CLIENT_FK    FOREIGN KEY(ID)   REFERENCES MT_DATA_ADMIN.CLIENT(ID)
);
CREATE INDEX A_CLIENT_IDX ON A_CLIENT (OUID);


-- View für Tabelle mit Zugriffsschicht erstellen
CREATE VIEW CLIENT
AS
    -- Alle auswählen...
    SELECT c.*, e.EID FROM MT_DATA_ADMIN.CLIENT c
    -- ...die eine Verknüfung in der Association-Tabelle haben...
    INNER JOIN A_CLIENT a ON a.ID = c.ID
    -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
    INNER JOIN MT_ORG_ADMIN.EMPLOYEE_OU e ON e.OUID = a.OUID;


-- ==========================================================================================


-- Assoziations-Tabelle für fachliche Tabelle erstellen (ORDER)
CREATE TABLE A_ORDER (
    ID INT NOT NULL        PRIMARY KEY,
    OUID INT NOT NULL,
    CONSTRAINT ORDER_OU_FK FOREIGN KEY(OUID) REFERENCES MT_ORG_ADMIN.ORGANIZATIONUNIT(ID),
    CONSTRAINT ORDER_FK    FOREIGN KEY(ID)   REFERENCES MT_DATA_ADMIN."ORDER"(ID)
);
CREATE INDEX A_ORDER_IDX ON A_ORDER (OUID);


-- View für Tabelle mit Zugriffsschicht erstellen
CREATE VIEW "ORDER"
AS
    -- Alle auswählen...
    SELECT c.*, e.EID FROM MT_DATA_ADMIN."ORDER" c
    -- ...die eine Verknüfung in der Association-Tabelle haben...
    INNER JOIN A_ORDER a ON a.ID = c.ID
    -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
    INNER JOIN MT_ORG_ADMIN.EMPLOYEE_OU e ON e.OUID = a.OUID;
```

```
-- INSERT - Erstelle Instead-of Trigger für View CLIENT
CREATE OR REPLACE TRIGGER CLIENT_INSERT_TRG
    INSTEAD OF INSERT ON CLIENT
DECLARE
    newID MT_DATA_ADMIN.CLIENT.ID%TYPE;
    oOUID MT_ORG_ADMIN.EMPLOYEE.OU%TYPE;
BEGIN
    -- Ermitteln der Eigentümer-OU
    SELECT OU INTO oOUID FROM MT_ORG_ADMIN.EMPLOYEE WHERE ID = :new.EID;
    -- Speichern der Daten, speichern des erstellten PK in "newID"
    INSERT INTO MT_DATA_ADMIN.CLIENT (ID, NAME)
    VALUES (:new.ID, :new.NAME) RETURNING ID INTO newID;
    -- Speichern der Verknüpfung OU <--> Datensatz
    INSERT INTO MT_ASS_ADMIN.A_CLIENT
    VALUES (newID, oOUID);
END;
/


-- UPDATE - Erstelle Instead-of Trigger für View CLIENT
CREATE OR REPLACE TRIGGER CLIENT_UPDATE_TRG
    INSTEAD OF UPDATE ON CLIENT
DECLARE
    NEW_OUID MT_ORG_ADMIN.EMPLOYEE.OU%TYPE;
BEGIN
    -- Eigentümer-OU ändern, wenn gewollt
    IF :new.EID <> :old.EID THEN
        SELECT OU INTO NEW_OUID FROM MT_ORG_ADMIN.EMPLOYEE WHERE ID = :new.EID;
        UPDATE MT_ASS_ADMIN.A_CLIENT SET OUID = NEW_OUID   WHERE ID = :new.ID;
    END IF;
    -- Datensatz ändern
    UPDATE MT_DATA_ADMIN.CLIENT SET NAME = :new.NAME WHERE ID = :new.ID;
END;
/


-- DELETE - Erstelle Instead-of Trigger für View CLIENT
CREATE OR REPLACE TRIGGER CLIENT_DELETE_TRG
    INSTEAD OF DELETE ON CLIENT
BEGIN
   DELETE FROM MT_ASS_ADMIN.A_CLIENT WHERE ID = :old.ID;
   DELETE FROM MT_DATA_ADMIN.CLIENT  WHERE ID = :old.ID;
END;
/


-- ==========================================================================================


-- INSERT - Erstelle Instead-of-Trigger für View ORDER
CREATE OR REPLACE TRIGGER ORDER_INSERT_TRG
    INSTEAD OF INSERT ON "ORDER"
DECLARE
    newID MT_DATA_ADMIN."ORDER".ID%TYPE;
    oOUID MT_ORG_ADMIN.EMPLOYEE.OU%TYPE;
BEGIN
    -- Ermitteln der Eigentümer-OU
    SELECT OU INTO oOUID FROM MT_ORG_ADMIN.EMPLOYEE WHERE ID = :new.EID;
    -- Speichern der Daten, speichern des erstellten PK in "newID"
    INSERT INTO MT_DATA_ADMIN."ORDER" (ID, CID, ORDER_DATETIME)
    VALUES (:new.ID, :new.CID, :new.ORDER_DATETIME) RETURNING ID INTO newID;
    -- Speichern der Verknüpfung OU <--> Datensatz
    INSERT INTO MT_ASS_ADMIN.A_ORDER
    VALUES (newID, oOUID);
END;
/
```

```
-- UPDATE - Erstelle Instead-of-Trigger für View ORDER
CREATE OR REPLACE TRIGGER ORDER_UPDATE_TRG
    INSTEAD OF UPDATE ON "ORDER"
DECLARE
    NEW_OUID MT_ORG_ADMIN.EMPLOYEE.OU%TYPE;
BEGIN
    -- Eigentümer-OU ändern, wenn gewollt
    IF :new.EID <> :old.EID THEN
        SELECT OU INTO NEW_OUID FROM MT_ORG_ADMIN.EMPLOYEE WHERE ID = :new.EID;
        UPDATE MT_ASS_ADMIN.A_ORDER SET OUID = NEW_OUID   WHERE ID = :new.ID;
    END IF;
    -- Datensatz ändern
    UPDATE MT_DATA_ADMIN."ORDER"
    SET CID = :new.CID, ORDER_DATETIME = :new.ORDER_DATETIME
    WHERE ID = :new.ID;
END;
/

-- DELETE - Erstelle Instead-of-Trigger für View ORDER
CREATE OR REPLACE TRIGGER ORDER_DELETE_TRG
    INSTEAD OF DELETE ON "ORDER"
BEGIN
    DELETE FROM MT_ASS_ADMIN.A_ORDER  WHERE ID = :old.ID;
    DELETE FROM MT_DATA_ADMIN."ORDER" WHERE ID = :old.ID;
END;
/


-- ==========================================================================================


GRANT SELECT, INSERT, UPDATE, DELETE ON "CLIENT"                TO MT_ASS_USER;
GRANT SELECT, INSERT, UPDATE, DELETE ON "ORDER"                 TO MT_ASS_USER;
GRANT SELECT, INSERT, UPDATE, DELETE ON "PRODUCT"               TO MT_ASS_USER;
GRANT SELECT, INSERT, UPDATE, DELETE ON "ORDERPRODUCT"          TO MT_ASS_USER;

-- Wegen Testfällen, Messung der Performance als "SYSTEM"
GRANT SELECT, INSERT, UPDATE, DELETE ON "CLIENT"                TO SYSTEM;
```

7. Oracle – Testdaten erstellen – Hierarchie, Mitarbeiter

```
-- ============================================
--
-- WICHTIG: Ausführen als MT_ORG_USER
--
-- ============================================

DECLARE
    maxHierarchyHight INT := 5; -- Höhe der Hierarchie
    amountOuPerLevel  INT := 4; -- Anzahl untergeordneter OUs je OU
    employeePerOu     INT := 4; -- Anzahl Mitarbeiter je OU

    -- Erzeugt einen Mitarbeiter-Datensatz in der übergebenen OU
    PROCEDURE CREATE_TEST_EMPLOYEE (organizationUnit IN INT) IS
    BEGIN
        FOR ouCounter IN 1..employeePerOu LOOP
            INSERT INTO EMPLOYEE (OU, NAME)
            VALUES (organizationUnit, 'E_' || DBMS_RANDOM.string('u',10));
        END LOOP;
    END;

    -- Erzeugt für die übergebene OU die untergeordneten OUs
    PROCEDURE CREATE_TEST_HIERARY (parentID IN INT, currentLevel in INT) AS
        tmpId INT;
    BEGIN
        IF (currentLevel > maxHierarchyHight) THEN -- Abbruchbedingung
            RETURN;
        END IF;
        FOR ouCounter IN 1..amountOuPerLevel LOOP
            INSERT INTO ORGANIZATIONUNIT (PARENT, NAME)
            VALUES (parentID, 'OU_' || DBMS_RANDOM.string('u',10)) RETURNING ID INTO tmpId;
            CREATE_TEST_EMPLOYEE(tmpId);
            CREATE_TEST_HIERARY(tmpId, currentLevel + 1);
        END LOOP;
    END;

    -- Erstellt die oberste OU (Root-Element) und startet den Vorgang
    PROCEDURE CREATE_TEST_START AS
        tmpId INT;
    BEGIN
        -- Erste Organisationseinheit, Ebene: 0
        INSERT INTO ORGANIZATIONUNIT (PARENT, NAME)
        VALUES (NULL, 'FIRST_OU') RETURNING ID INTO tmpId;
        CREATE_TEST_EMPLOYEE(tmpId);
        CREATE_TEST_HIERARY(tmpId, 1);
    END;

BEGIN
    CREATE_TEST_START();
    COMMIT;
END;
```

8. Oracle – Testdaten erstellen – Fachliche Datensätze

```
-- ============================================
--
-- WICHTIG: Ausführen als MT_ASS_ADMIN (wegen Zugriff auf EMPLOYEE-Tabelle)
--
-- ============================================


/*
  Dauer: Erzeugen von Testdaten:
  ---------------------------------------------
       1.000   ->      0,2 Sekunden
      10.000   ->      2,2 Sekunden
     100.000   ->     22,5 Sekunden
   1.000.000   ->    230,4 Sekunden
  10.000.000   ->   2270,4 Sekunden
*/


DECLARE
    maxTestData        INT := 100000; -- Zu erzeugende Anzahl an Testdaten
    maxTestDataCounter INT := 0;      -- Zähler-Variable

    CURSOR allEmployee IS
        SELECT e.ID FROM MT_ORG_ADMIN.EMPLOYEE e;
BEGIN
    WHILE maxTestDataCounter < maxTestData LOOP
        FOR emp IN allEmployee LOOP
            IF maxTestDataCounter = maxTestData THEN
                EXIT;
            END IF;
            INSERT INTO "CLIENT" (NAME, EID)
            VALUES ('Client_' || DBMS_RANDOM.string('u',10), emp.ID);
            maxTestDataCounter := maxTestDataCounter + 1;
        END LOOP;
    END LOOP;

    COMMIT;
END;
```

9. Oracle – Messergebnisse – 10.000 Datensätze

```
-- ==========================================================================================
--
-- Testfall:          1 Datensatz lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:     EXPLAIN PLAN
-- SQL-Statement:     SELECT * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460 AND ID = 5460
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             | 54760 |  7005K|   887  (93)| 00:00:01 |
|   1 |  NESTED LOOPS                              |                             | 54760 |  7005K|   887  (93)| 00:00:01 |
|   2 |   NESTED LOOPS                             |                             |     1 |   105 |    17   (0)| 00:00:01 |
|*  3 |    VIEW                                    | index$_join$_003            |    16 |   416 |     1   (0)| 00:00:01 |
|*  4 |     HASH JOIN                              |                             |       |       |            |          |
|*  5 |      INDEX RANGE SCAN                      | SYS_C0022095                |    16 |   416 |     1   (0)| 00:00:01 |
|   6 |      INDEX FAST FULL SCAN                  | A_CLIENT_IDX                |    16 |   416 |     0   (0)| 00:00:01 |
|   7 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                      |     1 |    79 |     1   (0)| 00:00:01 |
|*  8 |     INDEX UNIQUE SCAN                      | SYS_C0022084                |     1 |       |     0   (0)| 00:00:01 |
|*  9 |   VIEW                                     |                             | 54760 |  1390K|   869  (95)| 00:00:01 |
|  10 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|  11 |     VIEW                                   |                             |  5461 |   138K|     9   (0)| 00:00:01 |
|  12 |      UNION-ALL                             |                             |       |       |            |          |
|  13 |       TABLE ACCESS FULL                    | EMPLOYEE                    |  5460 |   138K|     7   (0)| 00:00:01 |
|  14 |       TABLE ACCESS FULL                    | ASSOCIATION                 |     1 |    26 |     2   (0)| 00:00:01 |
|  15 |     NESTED LOOPS                           |                             |   89M|  4437M|   860  (96)| 00:00:01 |
|  16 |      NESTED LOOPS                          |                             |   89M|  4437M|   860  (96)| 00:00:01 |
|  17 |       RECURSIVE WITH PUMP                  |                             |       |       |            |          |
|* 18 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  19 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT            |     4 |   104 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("A"."ID"=5460)
   4 - access(ROWID=ROWID)
   5 - access("A"."ID"=5460)
   8 - access("C"."ID"=5460)
   9 - filter("S"=5460 AND "ID"="A"."OUID")
  18 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)


-- ==========================================================================================
--
-- Testfall:          1 Datensatz lesen (Führungskraft, EID = 1)
-- Mess-Funktion:     EXPLAIN PLAN
-- SQL-Statement:     SELECT * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1 AND ID = 5460
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             | 54760 |  7005K|   887  (93)| 00:00:01 |
|   1 |  NESTED LOOPS                              |                             | 54760 |  7005K|   887  (93)| 00:00:01 |
|   2 |   NESTED LOOPS                             |                             |     1 |   105 |    17   (0)| 00:00:01 |
|*  3 |    VIEW                                    | index$_join$_003            |    16 |   416 |     1   (0)| 00:00:01 |
|*  4 |     HASH JOIN                              |                             |       |       |            |          |
|*  5 |      INDEX RANGE SCAN                      | SYS_C0022095                |    16 |   416 |     1   (0)| 00:00:01 |
|   6 |      INDEX FAST FULL SCAN                  | A_CLIENT_IDX                |    16 |   416 |     0   (0)| 00:00:01 |
|   7 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                      |     1 |    79 |     1   (0)| 00:00:01 |
|*  8 |     INDEX UNIQUE SCAN                      | SYS_C0022084                |     1 |       |     0   (0)| 00:00:01 |
|*  9 |   VIEW                                     |                             | 54760 |  1390K|   869  (95)| 00:00:01 |
|  10 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|  11 |     VIEW                                   |                             |  5461 |   138K|     9   (0)| 00:00:01 |
|  12 |      UNION-ALL                             |                             |       |       |            |          |
|  13 |       TABLE ACCESS FULL                    | EMPLOYEE                    |  5460 |   138K|     7   (0)| 00:00:01 |
|  14 |       TABLE ACCESS FULL                    | ASSOCIATION                 |     1 |    26 |     2   (0)| 00:00:01 |
|  15 |     NESTED LOOPS                           |                             |   89M|  4437M|   860  (96)| 00:00:01 |
|  16 |      NESTED LOOPS                          |                             |   89M|  4437M|   860  (96)| 00:00:01 |
|  17 |       RECURSIVE WITH PUMP                  |                             |       |       |            |          |
|* 18 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  19 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT            |     4 |   104 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."ID"=5460)
   4 - access(ROWID=ROWID)
   5 - access("A"."ID"=5460)
   8 - access("C"."ID"=5460)
   9 - filter("S"=1 AND "ID"="A"."OUID")
  18 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         1 Datensatz lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:    GATHER_PLAN_STATISTICS
-- SQL-Statement:    SELECT /*+ GATHER_PLAN_STATISTICS */ * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460 AND ID = 5460
--
-- ==========================================================================================


SQL_ID  4jprn29d46daa, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ * FROM MT_ASS_ADMIN.CLIENT WHERE
EID = 5460 AND ID = 5460

Plan hash value: 315909755

-----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                         | Starts | E-Rows | A-Rows |   A-Time    | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                              |      1 |        |      0 |00:00:00.22 |    107K|       |       |          |
|   1 |  NESTED LOOPS                              |                              |      1 |  54760 |      0 |00:00:00.22 |    107K|       |       |          |
|   2 |   NESTED LOOPS                             |                              |      1 |      1 |      1 |00:00:00.01 |     67 |       |       |          |
|*  3 |    VIEW                                    | index$_join$_003             |      1 |     16 |      1 |00:00:00.01 |     64 |       |       |          |
|*  4 |     HASH JOIN                              |                              |      1 |        |      1 |00:00:00.01 |     64 |  1572K|  1572K|  704K (0)|
|*  5 |      INDEX RANGE SCAN                      | SYS_C0022095                 |      1 |     16 |      1 |00:00:00.01 |      2 |       |       |          |
|   6 |      INDEX FAST FULL SCAN                  | A_CLIENT_IDX                 |      1 |     16 |  10000 |00:00:00.01 |     62 |       |       |          |
|   7 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                       |      1 |      1 |      1 |00:00:00.01 |      3 |       |       |          |
|*  8 |     INDEX UNIQUE SCAN                      | SYS_C0022084                 |      1 |      1 |      1 |00:00:00.01 |      2 |       |       |          |
|*  9 |   VIEW                                     |                              |      1 |  54760 |      0 |00:00:00.22 |    107K|       |       |          |
|  10 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                              |      1 |        |  30948 |00:00:00.23 |    107K|   267K|   267K|  237K (0)|
|  11 |     VIEW                                   |                              |      1 |   5461 |   5460 |00:00:00.01 |     23 |       |       |          |
|  12 |      UNION-ALL                             |                              |      1 |        |   5460 |00:00:00.01 |     23 |       |       |          |
|  13 |       TABLE ACCESS FULL                    | EMPLOYEE                     |      1 |   5460 |   5460 |00:00:00.01 |     23 |       |       |          |
|  14 |       TABLE ACCESS FULL                    | ASSOCIATION                  |      1 |      1 |      0 |00:00:00.01 |      0 |       |       |          |
|  15 |     NESTED LOOPS                           |                              |      6 |    89M|  25488 |00:00:00.12 |   3810 |       |       |          |
|  16 |      NESTED LOOPS                          |                              |      6 |    89M|  25488 |00:00:00.09 |   3223 |       |       |          |
|  17 |       RECURSIVE WITH PUMP                  |                              |      6 |        |  30948 |00:00:00.01 |      0 |       |       |          |
|* 18 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX  |  30948 |      4 |  25488 |00:00:00.02 |   3223 |       |       |          |
|  19 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT             |  25488 |      4 |  25488 |00:00:00.01 |    587 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."ID"=5460)
   4 - access(ROWID=ROWID)
   5 - access("A"."ID"=5460)
   8 - access("C"."ID"=5460)
   9 - filter(("S"=5460 AND "ID"="A"."OUID"))
  18 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         1 Datensatz lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    GATHER_PLAN_STATISTICS
-- SQL-Statement:    SELECT /*+ GATHER_PLAN_STATISTICS */ * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1 AND ID = 5460
--
-- ==========================================================================================


SQL_ID  4v0106ua84xqh, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ * FROM MT_ASS_ADMIN.CLIENT
WHERE EID = 1 AND ID = 5460

Plan hash value: 315909755

-----------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Starts | E-Rows | A-Rows |   A-Time    | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             |      1 |        |      1 |00:00:00.22 |    107K|       |       |          |
|   1 |  NESTED LOOPS                              |                             |      1 |  54760 |      1 |00:00:00.22 |    107K|       |       |          |
|   2 |   NESTED LOOPS                             |                             |      1 |      1 |      1 |00:00:00.01 |     45 |       |       |          |
|*  3 |    VIEW                                    | index$_join$_003            |      1 |     16 |      1 |00:00:00.01 |     42 |       |       |          |
|*  4 |     HASH JOIN                              |                             |      1 |        |      1 |00:00:00.01 |     42 |  1572K|  1572K|  697K (0)|
|*  5 |      INDEX RANGE SCAN                      | SYS_C0022095                |      1 |     16 |      1 |00:00:00.01 |      2 |       |       |          |
|   6 |      INDEX FAST FULL SCAN                  | A_CLIENT_IDX                |      1 |     16 |  10000 |00:00:00.01 |     40 |       |       |          |
|   7 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                      |      1 |      1 |      1 |00:00:00.01 |      3 |       |       |          |
|*  8 |     INDEX UNIQUE SCAN                      | SYS_C0022084                |      1 |      1 |      1 |00:00:00.01 |      2 |       |       |          |
|*  9 |   VIEW                                     |                             |      1 |  54760 |      1 |00:00:00.22 |    107K|       |       |          |
|  10 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |      1 |        |  30948 |00:00:00.23 |    107K|   267K|   267K|  237K (0)|
|  11 |     VIEW                                   |                             |      1 |   5461 |   5460 |00:00:00.01 |     23 |       |       |          |
|  12 |      UNION-ALL                             |                             |      1 |        |   5460 |00:00:00.01 |     23 |       |       |          |
|  13 |       TABLE ACCESS FULL                    | EMPLOYEE                    |      1 |   5460 |   5460 |00:00:00.01 |     23 |       |       |          |
|  14 |       TABLE ACCESS FULL                    | ASSOCIATION                 |      1 |      1 |      0 |00:00:00.01 |      0 |       |       |          |
|  15 |     NESTED LOOPS                           |                             |      6 |    89M|  25488 |00:00:00.11 |   3810 |       |       |          |
|  16 |      NESTED LOOPS                          |                             |      6 |    89M|  25488 |00:00:00.09 |   3223 |       |       |          |
|  17 |       RECURSIVE WITH PUMP                  |                             |      6 |        |  30948 |00:00:00.01 |      0 |       |       |          |
|* 18 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX |  30948 |      4 |  25488 |00:00:00.02 |   3223 |       |       |          |
|  19 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT            |  25488 |      4 |  25488 |00:00:00.01 |    587 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("A"."ID"=5460)
   4 - access(ROWID=ROWID)
   5 - access("A"."ID"=5460)
   8 - access("C"."ID"=5460)
   9 - filter(("S"=1 AND "ID"="A"."OUID"))
  18 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         Anzahl aller sichtbaren Datensätze lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             |     1 |    39 |  8091 (100)| 00:00:01 |
|   1 |  SORT AGGREGATE                              |                             |     1 |    39 |            |          |
|*  2 |   HASH JOIN                                  |                             |   655M|   23G|  8091 (100)| 00:00:01 |
|   3 |    TABLE ACCESS FULL                         | A_CLIENT                    | 10000 |  126K|     7   (0)| 00:00:01 |
|*  4 |    VIEW                                      |                             |    89M| 2218M|   869  (95)| 00:00:01 |
|   5 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   6 |      VIEW                                    |                             |  5461 |  138K|     9   (0)| 00:00:01 |
|   7 |       UNION-ALL                              |                             |       |       |            |          |
|   8 |        TABLE ACCESS FULL                     | EMPLOYEE                    |  5460 |  138K|     7   (0)| 00:00:01 |
|   9 |        TABLE ACCESS FULL                     | ASSOCIATION                 |     1 |   26 |     2   (0)| 00:00:01 |
|  10 |      NESTED LOOPS                            |                             |    89M| 4437M|   860  (96)| 00:00:01 |
|  11 |       NESTED LOOPS                           |                             |    89M| 4437M|   860  (96)| 00:00:01 |
|  12 |        RECURSIVE WITH PUMP                   |                             |       |       |            |          |
|* 13 |        INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  14 |       TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |     4 |  104 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("ID"="A"."OUID")
   4 - filter("S"=5460)
  13 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         Anzahl aller sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             |     1 |    39 |  8091 (100)| 00:00:01 |
|   1 |  SORT AGGREGATE                              |                             |     1 |    39 |            |          |
|*  2 |   HASH JOIN                                  |                             |   655M|   23G|  8091 (100)| 00:00:01 |
|   3 |    TABLE ACCESS FULL                         | A_CLIENT                    | 10000 |  126K|     7   (0)| 00:00:01 |
|*  4 |    VIEW                                      |                             |    89M| 2218M|   869  (95)| 00:00:01 |
|   5 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   6 |      VIEW                                    |                             |  5461 |  138K|     9   (0)| 00:00:01 |
|   7 |       UNION-ALL                              |                             |       |       |            |          |
|   8 |        TABLE ACCESS FULL                     | EMPLOYEE                    |  5460 |  138K|     7   (0)| 00:00:01 |
|   9 |        TABLE ACCESS FULL                     | ASSOCIATION                 |     1 |   26 |     2   (0)| 00:00:01 |
|  10 |      NESTED LOOPS                            |                             |    89M| 4437M|   860  (96)| 00:00:01 |
|  11 |       NESTED LOOPS                           |                             |    89M| 4437M|   860  (96)| 00:00:01 |
|  12 |        RECURSIVE WITH PUMP                   |                             |       |       |            |          |
|* 13 |        INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  14 |       TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |     4 |  104 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("ID"="A"."OUID")
   4 - filter("S"=1)
  13 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:          Anzahl aller sichtbaren Datensätze lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:     GATHER_PLAN_STATISTICS
-- SQL-Statement:     SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460
-- Hinweis:           In Zeile 3, Spalte A-Rows, ist die Anzahl der für den Mitarbeiter „sichtbaren“ Datensätze erkennbar: 8 Stück
--
-- ==========================================================================================


SQL_ID  7mvb10m90cuvk, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM
MT_ASS_ADMIN.CLIENT WHERE EID = 5460

Plan hash value: 4186134613

-----------------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             |      1 |        |      1 |00:00:00.25 |     107K|       |       |          |
|   1 |  SORT AGGREGATE                              |                             |      1 |      1 |      1 |00:00:00.25 |     107K|       |       |          |
|*  2 |   HASH JOIN                                  |                             |      1 |    655M|      8 |00:00:00.04 |     107K|  2546K|  2546K| 2603K (0)|
|   3 |    TABLE ACCESS FULL                         | A_CLIENT                    |      1 |  10000 |  10000 |00:00:00.01 |      23 |       |       |          |
|*  4 |    VIEW                                      |                             |      1 |     89M|      1 |00:00:00.04 |     107K|       |       |          |
|   5 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |      1 |        |  30948 |00:00:00.24 |     107K|   267K|   267K|  237K (0)|
|   6 |      VIEW                                    |                             |      1 |   5461 |   5460 |00:00:00.01 |      23 |       |       |          |
|   7 |       UNION-ALL                              |                             |      1 |        |   5460 |00:00:00.01 |      23 |       |       |          |
|   8 |        TABLE ACCESS FULL                     | EMPLOYEE                    |      1 |   5460 |   5460 |00:00:00.01 |      23 |       |       |          |
|   9 |        TABLE ACCESS FULL                     | ASSOCIATION                 |      1 |      1 |      0 |00:00:00.01 |       0 |       |       |          |
|  10 |      NESTED LOOPS                            |                             |      6 |     89M|  25488 |00:00:00.11 |    3814 |       |       |          |
|  11 |       NESTED LOOPS                           |                             |      6 |     89M|  25488 |00:00:00.09 |    3227 |       |       |          |
|  12 |        RECURSIVE WITH PUMP                   |                             |      6 |        |  30948 |00:00:00.01 |       0 |       |       |          |
|* 13 |        INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |  30948 |      4 |  25488 |00:00:00.02 |    3227 |       |       |          |
|  14 |       TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |  25488 |      4 |  25488 |00:00:00.01 |     587 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("ID"="A"."OUID")
   4 - filter("S"=5460)
  13 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         Anzahl aller sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    GATHER_PLAN_STATISTICS
-- SQL-Statement:    SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
-- Hinweis:          In Zeile 3, Spalte A-Rows, ist die Anzahl der für diese oberste Führungskraft „sichtbaren“ Datensätze erkennbar: alle 10.000 Stück
--
-- ==========================================================================================


SQL_ID  dudnc2y6q163u, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM
MT_ASS_ADMIN.CLIENT WHERE EID = 1

Plan hash value: 4186134613

-------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                     | Name                        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                              |                             |      1 |        |      1 |00:00:00.22 |    107K|       |       |          |
|   1 |  SORT AGGREGATE                               |                             |      1 |      1 |      1 |00:00:00.22 |    107K|       |       |          |
|*  2 |   HASH JOIN                                   |                             |      1 |    655M|  10000 |00:00:00.08 |    107K|  2546K|  2546K| 2582K (0)|
|   3 |    TABLE ACCESS FULL                          | A_CLIENT                    |      1 |  10000 |  10000 |00:00:00.01 |     23 |       |       |          |
|*  4 |    VIEW                                       |                             |      1 |     89M|   1365 |00:00:00.01 |    107K|       |       |          |
|   5 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |      1 |        |  30948 |00:00:00.24 |    107K|   267K|   267K|  237K (0)|
|   6 |      VIEW                                     |                             |      1 |   5461 |   5460 |00:00:00.01 |     23 |       |       |          |
|   7 |       UNION-ALL                               |                             |      1 |        |   5460 |00:00:00.01 |     23 |       |       |          |
|   8 |        TABLE ACCESS FULL                      | EMPLOYEE                    |      1 |   5460 |   5460 |00:00:00.01 |     23 |       |       |          |
|   9 |        TABLE ACCESS FULL                      | ASSOCIATION                 |      1 |      1 |      0 |00:00:00.01 |      0 |       |       |          |
|  10 |      NESTED LOOPS                             |                             |      6 |     89M|  25488 |00:00:00.11 |   3810 |       |       |          |
|  11 |       NESTED LOOPS                            |                             |      6 |     89M|  25488 |00:00:00.09 |   3223 |       |       |          |
|  12 |        RECURSIVE WITH PUMP                    |                             |      6 |        |  30948 |00:00:00.01 |      0 |       |       |          |
|* 13 |        INDEX RANGE SCAN                       | ORGANIZATIONUNIT_PARENT_IDX |  30948 |      4 |  25488 |00:00:00.02 |   3223 |       |       |          |
|  14 |       TABLE ACCESS BY INDEX ROWID             | ORGANIZATIONUNIT            |  25488 |      4 |  25488 |00:00:00.01 |    587 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("ID"="A"."OUID")
   4 - filter("S"=1)
  13 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

10. Oracle – Messergebnisse – 10.000.000 Datensätze

```
-- ==========================================================================================
--
-- Testfall:         1 Datensatz lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460 AND ID = 5460
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             |    51 |  6681 |   872  (95)| 00:00:01 |
|   1 |  NESTED LOOPS                              |                             |    51 |  6681 |   872  (95)| 00:00:01 |
|   2 |   NESTED LOOPS                             |                             |     1 |   105 |     3   (0)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID             | A_CLIENT                    |     1 |    26 |     2   (0)| 00:00:01 |
|*  4 |     INDEX UNIQUE SCAN                      | SYS_C0022095                |     1 |       |     1   (0)| 00:00:01 |
|   5 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                      |     1 |    79 |     1   (0)| 00:00:01 |
|*  6 |     INDEX UNIQUE SCAN                      | SYS_C0022084                |     1 |       |     0   (0)| 00:00:01 |
|*  7 |   VIEW                                     |                             |    51 |  1326 |   869  (95)| 00:00:01 |
|   8 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   9 |     VIEW                                   |                             |  5461 |  138K|     9   (0)| 00:00:01 |
|  10 |      UNION-ALL                             |                             |       |       |            |          |
|  11 |       TABLE ACCESS FULL                    | EMPLOYEE                    |  5460 |  138K|     7   (0)| 00:00:01 |
|  12 |       TABLE ACCESS FULL                    | ASSOCIATION                 |     1 |    26 |     2   (0)| 00:00:01 |
|  13 |     NESTED LOOPS                           |                             |    89M|  4437M|   860  (96)| 00:00:01 |
|  14 |      NESTED LOOPS                          |                             |    89M|  4437M|   860  (96)| 00:00:01 |
|  15 |       RECURSIVE WITH PUMP                  |                             |       |       |            |          |
|* 16 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  17 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT            |     4 |   104 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("A"."ID"=5460)
   6 - access("C"."ID"=5460)
   7 - filter("S"=5460 AND "ID"="A"."OUID")
  16 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         1 Datensatz lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1 AND ID = 5460
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             |    51 |  6681 |   872  (95)| 00:00:01 |
|   1 |  NESTED LOOPS                              |                             |    51 |  6681 |   872  (95)| 00:00:01 |
|   2 |   NESTED LOOPS                             |                             |     1 |   105 |     3   (0)| 00:00:01 |
|   3 |    TABLE ACCESS BY INDEX ROWID             | A_CLIENT                    |     1 |    26 |     2   (0)| 00:00:01 |
|*  4 |     INDEX UNIQUE SCAN                      | SYS_C0022095                |     1 |       |     1   (0)| 00:00:01 |
|   5 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                      |     1 |    79 |     1   (0)| 00:00:01 |
|*  6 |     INDEX UNIQUE SCAN                      | SYS_C0022084                |     1 |       |     0   (0)| 00:00:01 |
|*  7 |   VIEW                                     |                             |    51 |  1326 |   869  (95)| 00:00:01 |
|   8 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   9 |     VIEW                                   |                             |  5461 |  138K|     9   (0)| 00:00:01 |
|  10 |      UNION-ALL                             |                             |       |       |            |          |
|  11 |       TABLE ACCESS FULL                    | EMPLOYEE                    |  5460 |  138K|     7   (0)| 00:00:01 |
|  12 |       TABLE ACCESS FULL                    | ASSOCIATION                 |     1 |    26 |     2   (0)| 00:00:01 |
|  13 |     NESTED LOOPS                           |                             |    89M|  4437M|   860  (96)| 00:00:01 |
|  14 |      NESTED LOOPS                          |                             |    89M|  4437M|   860  (96)| 00:00:01 |
|  15 |       RECURSIVE WITH PUMP                  |                             |       |       |            |          |
|* 16 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  17 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT            |     4 |   104 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("A"."ID"=5460)
   6 - access("C"."ID"=5460)
   7 - filter("S"=1 AND "ID"="A"."OUID")
  16 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         1 Datensatz lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:    GATHER_PLAN_STATISTICS
-- SQL-Statement:    SELECT /*+ GATHER_PLAN_STATISTICS */ * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460 AND ID = 5460
--
-- ==========================================================================================


SQL_ID  4jprn29d46daa, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ *   FROM MT_ASS_ADMIN.CLIENT WHERE
EID = 5460 AND ID = 5460

Plan hash value: 2324265528

-----------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Starts | E-Rows | A-Rows |   A-Time    | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             |      1 |        |      0 |00:00:00.22 |    107K|       |       |          |
|   1 |  NESTED LOOPS                              |                             |      1 |     51 |      0 |00:00:00.22 |    107K|       |       |          |
|   2 |   NESTED LOOPS                             |                             |      1 |      1 |      1 |00:00:00.01 |       8 |       |       |          |
|   3 |    TABLE ACCESS BY INDEX ROWID             | A_CLIENT                    |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|*  4 |     INDEX UNIQUE SCAN                      | SYS_C0022095                |      1 |      1 |      1 |00:00:00.01 |       3 |       |       |          |
|   5 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                      |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|*  6 |     INDEX UNIQUE SCAN                      | SYS_C0022084                |      1 |      1 |      1 |00:00:00.01 |       3 |       |       |          |
|*  7 |   VIEW                                     |                             |      1 |     51 |      0 |00:00:00.22 |    107K|       |       |          |
|   8 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |      1 |        |  30948 |00:00:00.23 |    107K|   267K|   267K|  237K (0)|
|   9 |     VIEW                                   |                             |      1 |   5461 |   5460 |00:00:00.01 |      23 |       |       |          |
|  10 |      UNION-ALL                             |                             |      1 |        |   5460 |00:00:00.01 |      23 |       |       |          |
|  11 |       TABLE ACCESS FULL                    | EMPLOYEE                    |      1 |   5460 |   5460 |00:00:00.01 |      23 |       |       |          |
|  12 |       TABLE ACCESS FULL                    | ASSOCIATION                 |      1 |      1 |      0 |00:00:00.01 |       0 |       |       |          |
|  13 |     NESTED LOOPS                           |                             |      6 |    89M|  25488 |00:00:00.13 |    3810 |       |       |          |
|  14 |      NESTED LOOPS                          |                             |      6 |    89M|  25488 |00:00:00.09 |    3223 |       |       |          |
|  15 |       RECURSIVE WITH PUMP                  |                             |      6 |        |  30948 |00:00:00.01 |       0 |       |       |          |
|* 16 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX |  30948 |      4 |  25488 |00:00:00.02 |    3223 |       |       |          |
|  17 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT            |  25488 |      4 |  25488 |00:00:00.01 |     587 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("A"."ID"=5460)
   6 - access("C"."ID"=5460)
   7 - filter(("S"=5460 AND "ID"="A"."OUID"))
  16 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         1 Datensatz lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    GATHER_PLAN_STATISTICS
-- SQL-Statement:    SELECT /*+ GATHER_PLAN_STATISTICS */ * FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1 AND ID = 5460
--
-- ==========================================================================================


SQL_ID  4v0106ua84xqh, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ *    FROM MT_ASS_ADMIN.CLIENT
WHERE EID = 1 AND ID = 5460

Plan hash value: 2324265528

-----------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Starts | E-Rows | A-Rows |   A-Time   | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             |      1 |        |      1 |00:00:00.23 |    107K|       |       |          |
|   1 |  NESTED LOOPS                              |                             |      1 |     51 |      1 |00:00:00.23 |    107K|       |       |          |
|   2 |   NESTED LOOPS                             |                             |      1 |      1 |      1 |00:00:00.01 |       8 |       |       |          |
|   3 |    TABLE ACCESS BY INDEX ROWID             | A_CLIENT                    |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|*  4 |     INDEX UNIQUE SCAN                      | SYS_C0022095                |      1 |      1 |      1 |00:00:00.01 |       3 |       |       |          |
|   5 |    TABLE ACCESS BY INDEX ROWID             | CLIENT                      |      1 |      1 |      1 |00:00:00.01 |       4 |       |       |          |
|*  6 |     INDEX UNIQUE SCAN                      | SYS_C0022084                |      1 |      1 |      1 |00:00:00.01 |       3 |       |       |          |
|*  7 |   VIEW                                     |                             |      1 |     51 |      1 |00:00:00.23 |    107K|       |       |          |
|   8 |    UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |      1 |        |  30948 |00:00:00.23 |    107K|   267K|   267K|  237K (0)|
|   9 |     VIEW                                   |                             |      1 |   5461 |   5460 |00:00:00.01 |      23 |       |       |          |
|  10 |      UNION-ALL                             |                             |      1 |        |   5460 |00:00:00.01 |      23 |       |       |          |
|  11 |       TABLE ACCESS FULL                    | EMPLOYEE                    |      1 |   5460 |   5460 |00:00:00.01 |      23 |       |       |          |
|  12 |       TABLE ACCESS FULL                    | ASSOCIATION                 |      1 |      1 |      0 |00:00:00.01 |       0 |       |       |          |
|  13 |     NESTED LOOPS                           |                             |      6 |    89M|  25488 |00:00:00.11 |    3810 |       |       |          |
|  14 |      NESTED LOOPS                          |                             |      6 |    89M|  25488 |00:00:00.09 |    3223 |       |       |          |
|  15 |       RECURSIVE WITH PUMP                  |                             |      6 |        |  30948 |00:00:00.01 |       0 |       |       |          |
|* 16 |       INDEX RANGE SCAN                     | ORGANIZATIONUNIT_PARENT_IDX |  30948 |      4 |  25488 |00:00:00.03 |    3223 |       |       |          |
|  17 |      TABLE ACCESS BY INDEX ROWID           | ORGANIZATIONUNIT            |  25488 |      4 |  25488 |00:00:00.01 |     587 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("A"."ID"=5460)
   6 - access("C"."ID"=5460)
   7 - filter(("S"=1 AND "ID"="A"."OUID"))
  16 - access("ORG"."PARENT"="MYSELF"."ID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         Anzahl aller sichtbaren Datensätze lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             |     1 |    39 |   228K(100)| 00:00:09 |
|   1 |  SORT AGGREGATE                              |                             |     1 |    39 |            |          |
|   2 |   NESTED LOOPS                               |                             |   666G|   23T|   228K(100)| 00:00:09 |
|*  3 |    VIEW                                      |                             |   89M| 2218M|   869  (95)| 00:00:01 |
|   4 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   5 |      VIEW                                    |                             | 5461 |  138K|     9   (0)| 00:00:01 |
|   6 |       UNION-ALL                              |                             |       |       |            |          |
|   7 |        TABLE ACCESS FULL                     | EMPLOYEE                    | 5460 |  138K|     7   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL                     | ASSOCIATION                 |    1 |   26 |     2   (0)| 00:00:01 |
|   9 |      NESTED LOOPS                            |                             |   89M| 4437M|   860  (96)| 00:00:01 |
|  10 |       NESTED LOOPS                           |                             |   89M| 4437M|   860  (96)| 00:00:01 |
|  11 |        RECURSIVE WITH PUMP                   |                             |       |       |            |          |
|* 12 |        INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |    4 |       |     0   (0)| 00:00:01 |
|  13 |       TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |    4 |  104 |     0   (0)| 00:00:01 |
|* 14 |    INDEX RANGE SCAN                          | A_CLIENT_IDX                | 7444 | 96772 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("S"=5460)
  12 - access("ORG"."PARENT"="MYSELF"."ID")
  14 - access("ID"="A"."OUID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         Anzahl aller sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             |     1 |    39 |   228K(100)| 00:00:09 |
|   1 |  SORT AGGREGATE                              |                             |     1 |    39 |            |          |
|   2 |   NESTED LOOPS                               |                             |   666G|   23T|   228K(100)| 00:00:09 |
|*  3 |    VIEW                                      |                             |   89M| 2218M|   869  (95)| 00:00:01 |
|   4 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   5 |      VIEW                                    |                             | 5461 |  138K|     9   (0)| 00:00:01 |
|   6 |       UNION-ALL                              |                             |       |       |            |          |
|   7 |        TABLE ACCESS FULL                     | EMPLOYEE                    | 5460 |  138K|     7   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL                     | ASSOCIATION                 |    1 |   26 |     2   (0)| 00:00:01 |
|   9 |      NESTED LOOPS                            |                             |   89M| 4437M|   860  (96)| 00:00:01 |
|  10 |       NESTED LOOPS                           |                             |   89M| 4437M|   860  (96)| 00:00:01 |
|  11 |        RECURSIVE WITH PUMP                   |                             |       |       |            |          |
|* 12 |        INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |    4 |       |     0   (0)| 00:00:01 |
|  13 |       TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |    4 |  104 |     0   (0)| 00:00:01 |
|* 14 |    INDEX RANGE SCAN                          | A_CLIENT_IDX                | 7444 | 96772 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("S"=1)
  12 - access("ORG"."PARENT"="MYSELF"."ID")
  14 - access("ID"="A"."OUID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:          Anzahl aller sichtbaren Datensätze lesen (Mitarbeiter, EID = 5460)
-- Mess-Funktion:     GATHER_PLAN_STATISTICS
-- SQL-Statement:     SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 5460
-- Hinweis:           In Zeile 3, Spalte A-Rows, ist die Anzahl der für den Mitarbeiter „sichtbaren“ Datensätze erkennbar: 7.324 Stück
--
-- ==========================================================================================


SQL_ID  7mvb10m90cuvk, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM
MT_ASS_ADMIN.CLIENT WHERE EID = 5460

Plan hash value: 1315294857

-----------------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Starts | E-Rows | A-Rows |   A-Time    | Buffers |  OMem |  1Mem | Used-Mem |
-----------------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             |      1 |        |      1 |00:00:00.22 |    107K|       |       |          |
|   1 |  SORT AGGREGATE                              |                             |      1 |      1 |      1 |00:00:00.22 |    107K|       |       |          |
|   2 |   NESTED LOOPS                               |                             |      1 |   666G|   7324 |00:00:00.01 |    107K|       |       |          |
|*  3 |    VIEW                                      |                             |      1 |    89M|      1 |00:00:00.22 |    107K|       |       |          |
|   4 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |      1 |        |  30948 |00:00:00.24 |    107K|   267K|   267K|  237K (0)|
|   5 |      VIEW                                    |                             |      1 |   5461 |   5460 |00:00:00.01 |      23 |       |       |          |
|   6 |       UNION-ALL                              |                             |      1 |        |   5460 |00:00:00.01 |      23 |       |       |          |
|   7 |        TABLE ACCESS FULL                     | EMPLOYEE                    |      1 |   5460 |   5460 |00:00:00.01 |      23 |       |       |          |
|   8 |        TABLE ACCESS FULL                     | ASSOCIATION                 |      1 |      1 |      0 |00:00:00.01 |       0 |       |       |          |
|   9 |      NESTED LOOPS                            |                             |      6 |    89M|  25488 |00:00:00.12 |    3810 |       |       |          |
|  10 |       NESTED LOOPS                           |                             |      6 |    89M|  25488 |00:00:00.10 |    3223 |       |       |          |
|  11 |        RECURSIVE WITH PUMP                   |                             |      6 |        |  30948 |00:00:00.01 |       0 |       |       |          |
|* 12 |        INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |  30948 |      4 |  25488 |00:00:00.02 |    3223 |       |       |          |
|  13 |       TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |  25488 |      4 |  25488 |00:00:00.01 |     587 |       |       |          |
|* 14 |    INDEX RANGE SCAN                          | A_CLIENT_IDX                |      1 |   7444 |   7324 |00:00:00.01 |      54 |       |       |          |
-----------------------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("S"=5460)
  12 - access("ORG"."PARENT"="MYSELF"."ID")
  14 - access("ID"="A"."OUID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:          Anzahl aller sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:     GATHER_PLAN_STATISTICS
-- SQL-Statement:     SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
-- Hinweis:           In Zeile 3, Spalte A-Rows, ist die Anzahl der für diese oberste Führungskraft „sichtbaren“ Datensätze erkennbar: alle 10.000.000 Stück
--
-- ==========================================================================================


SQL_ID  dudnc2y6q163u, child number 0
-------------------------------------
SELECT /*+ GATHER_PLAN_STATISTICS */ COUNT(*) FROM
MT_ASS_ADMIN.CLIENT WHERE EID = 1

Plan hash value: 1315294857

-------------------------------------------------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                    | Name                        | Starts | E-Rows | A-Rows |   A-Time    | Buffers |  OMem |  1Mem | Used-Mem |
-------------------------------------------------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |                             |      1 |        |      1 |00:00:01.32 |    146K|       |       |          |
|   1 |  SORT AGGREGATE                              |                             |      1 |      1 |      1 |00:00:01.32 |    146K|       |       |          |
|   2 |   NESTED LOOPS                               |                             |      1 |   666G|    10M |00:00:01.86 |    146K|       |       |          |
|*  3 |    VIEW                                      |                             |      1 |    89M|   1365 |00:00:01.21 |    107K|       |       |          |
|   4 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |      1 |        |  30948 |00:00:00.23 |    107K|   267K|   267K|  237K (0)|
|   5 |      VIEW                                    |                             |      1 |   5461 |   5460 |00:00:00.01 |      23 |       |       |          |
|   6 |       UNION-ALL                              |                             |      1 |        |   5460 |00:00:00.01 |      23 |       |       |          |
|   7 |        TABLE ACCESS FULL                     | EMPLOYEE                    |      1 |   5460 |   5460 |00:00:00.01 |      23 |       |       |          |
|   8 |        TABLE ACCESS FULL                     | ASSOCIATION                 |      1 |      1 |      0 |00:00:00.01 |       0 |       |       |          |
|   9 |      NESTED LOOPS                            |                             |      6 |    89M|  25488 |00:00:00.12 |    3810 |       |       |          |
|  10 |       NESTED LOOPS                           |                             |      6 |    89M|  25488 |00:00:00.09 |    3223 |       |       |          |
|  11 |        RECURSIVE WITH PUMP                   |                             |      6 |        |  30948 |00:00:00.01 |       0 |       |       |          |
|* 12 |        INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |  30948 |      4 |  25488 |00:00:00.02 |    3223 |       |       |          |
|  13 |       TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |  25488 |      4 |  25488 |00:00:00.01 |     587 |       |       |          |
|* 14 |    INDEX RANGE SCAN                          | A_CLIENT_IDX                |   1365 |   7444 |    10M |00:00:00.98 |   38943 |       |       |          |
-------------------------------------------------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("S"=1)
  12 - access("ORG"."PARENT"="MYSELF"."ID")
  14 - access("ID"="A"."OUID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

11. Oracle – Performance-Optimierung durch Materialized View

```
-- ============================================
--
-- WICHTIG: Ausführen als SYSTEM / SYS
--
-- ============================================

-- Recht wird benötigt
GRANT CREATE MATERIALIZED VIEW to MT_ORG_ADMIN;



-- ============================================
--
-- WICHTIG: Ausführen als MT_ORG_ADMIN
--
-- ============================================

-- Alte (normale) View entfernen
DROP VIEW EMPLOYEE_OU;


-- Materialized View erzeugen
CREATE MATERIALIZED VIEW EMPLOYEE_OU
BUILD IMMEDIATE
REFRESH COMPLETE ON DEMAND
AS
    WITH MySelf (S, ID) AS (
    SELECT "S","ID"  FROM (
        -- Startpunkt 1: Primäre OU ermitteln
        SELECT
            e.ID       AS "S",  -- Start E-ID
            org.ID     AS "ID"  -- OU-ID
        FROM employee e
        INNER JOIN organizationunit org ON org.ID = e.OU

        UNION ALL

        -- Startpunkt 2: Sekundäre OUs ermitteln
        SELECT
            e.ID       AS "S",   -- Start E-ID
            org.ID     AS "ID"   -- OU-ID
        FROM employee e
        INNER JOIN association a ON a.EID = e.ID
        INNER JOIN organizationunit org ON org.ID = a.OUID
    )

    UNION ALL

    -- Rekursiv darunter liegende Sekundär-OUs ermitteln
    SELECT
        MySelf.S   AS "S",  -- Start E-ID
        org.ID     AS "ID"  -- OU-ID
    FROM organizationunit org
    INNER JOIN MySelf ON org.PARENT = MySelf.ID
)
SELECT S as "EID", ID as "OUID" FROM MySelf GROUP BY S, ID;


-- Index auf Spalte EID der Materialized View erzeugen
CREATE INDEX EMPLOYEE_OU_MV_IDX ON EMPLOYEE_OU(EID);


-- Rechte neu vergeben
GRANT SELECT                        ON EMPLOYEE_OU TO MT_ORG_USER;
GRANT SELECT, INSERT, UPDATE, DELETE ON EMPLOYEE_OU TO MT_ASS_ADMIN WITH GRANT OPTION;
```

```
-- Trigger 1 / 3 für Neu-Berechnung der View (Tabelle: OrganizationUnit)
CREATE OR REPLACE TRIGGER OU_MV_REFRESH
AFTER INSERT OR UPDATE OR DELETE
ON ORGANIZATIONUNIT
DECLARE
  JOB_NO NUMBER;
BEGIN
  DBMS_JOB.SUBMIT(JOB_NO, 'DBMS_MVIEW.REFRESH(''EMPLOYEE_OU'');');
END;
/

-- Trigger 2 / 3 für Neu-Berechnung der View (Tabelle: Employee)
CREATE OR REPLACE TRIGGER EMP_MV_REFRESH
AFTER INSERT OR UPDATE OR DELETE
ON EMPLOYEE
DECLARE
  JOB_NO NUMBER;
BEGIN
  DBMS_JOB.SUBMIT(JOB_NO, 'DBMS_MVIEW.REFRESH(''EMPLOYEE_OU'');');
END;
/

-- Trigger 3 / 3 für Neu-Berechnung der View (Tabelle: Association)
CREATE OR REPLACE TRIGGER ASS_MV_REFRESH
AFTER INSERT OR UPDATE OR DELETE
ON ASSOCIATION
DECLARE
  JOB_NO NUMBER;
BEGIN
  DBMS_JOB.SUBMIT(JOB_NO, 'DBMS_MVIEW.REFRESH(''EMPLOYEE_OU'');');
END;
/

/*
Hinweis:
-------------------------------------------------------------------------------------
Die Trigger müssen über Jobs aktualisiert werden, da der direkte Aufruf der Refresh-Methode
intern ein COMMIT mit sich bringt, was innerhalb von Triggern verboten ist. Wird hingegen eine
Autonome Transaktion (AUTONOMOUS_TRANSACTION) innerhalb der Trigger verwendet, sind die
veränderten Daten für die View aufgrund des Isolations-Levels noch nicht sichtbar.

Bei der Erstellung einer Materialized View gibt es außerdem einige Einschränkungen zu beachten.
Siehe dazu den Link, offizielle Dokumentation 'Data Warehouse Guide', ab Kapitel 5.3:
https://docs.oracle.com/en/database/oracle/oracle-database/18/dwhsg/basic-materialized-
views.html

Ursprünglich war der Wunsch, eine MV über 'REFRESH COMPLETE ON COMMIT' zu nutzen, um den Umweg
über Trigger zu vermeiden. Das wurde leider mit der folgenden Meldung abgelehnt:
> ORA-12054: Cannot set the ON COMMIT refresh attribute for the materialized view
> *Cause:    The materialized view did not satisfy conditions for refresh at
>            commit time.
> *Action:   Specify only valid options.

Auch die Nutzung 'REFRESH COMPLETE ON STATEMENT' ist mit einer etwas kurzen Meldung abgebrochen:
> ORA-32428: Fehler bei Materialized Join View des Typs "on-statement":
>            Refresh fast not specified

Leider hat auch die Bedingung 'REFRESH FAST ON STATEMENT' trotz erstellter Materialized-View
Logs keine Besserung gebracht und auch eine entsprechende Meldung erzeugt:
> ORA-12015: Cannot create a fast refresh materialized view from a complex query
> *Cause:    Neither ROWIDs and nor primary key constraints are supported for
>            complex queries.
> *Action:   Reissue the command with the REFRESH FORCE or REFRESH COMPLETE
>            option or create a simple materialized view.

Da die Erstellung einer Materialized View explizit nicht Bestandteil dieser Arbeit war und eher
als kleiner Bonus mit Anreiz zur Weiterentwicklung zu betrachten ist, habe ich mich bewusst für
den Weg über den asynchronen Refresh durch Trigger und Jobs entschieden.
Dieser durchaus lange Kommentar-Block ist ebenfalls bewusst in den Anhang aufgenommen. Für den
eigentlichen Inhalt dieser Arbeit ist er nicht wirklich relevant, doch für mich persönlich und
zukünftige Leser kann er durchaus hilfreich sein. Die hier gewonnen Erkenntnisse sind meiner
Meinung nach einfach zu wertvoll, um in einer Textdatei auf einem Laufwerk zu verschwinden.
*/
```

12. Oracle – Messergebnisse – 100.000 Datensätze (Optimiert)

```
-- ==========================================================================================
--
-- Testfall:          Anzahl aller sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:     EXPLAIN PLAN
-- SQL-Statement:     SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
-- Kosten:            22.929
--
-- Hinweis:           Vor der Optimierung, Nutzung einer klassischen View aus Anhang 3
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                      | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                               |                             |     1 |    39 | 22929 (100)| 00:00:01 |
|   1 |  SORT AGGREGATE                                |                             |     1 |    39 |            |          |
|   2 |   NESTED LOOPS                                 |                             |  6740M|   244G| 22929 (100)| 00:00:01 |
|   3 |    VIEW                                        | EMPLOYEE_OU                 |   89M|  2218M| 12589 (100)| 00:00:01 |
|   4 |     HASH GROUP BY                              |                             |   89M|  2218M| 12589 (100)| 00:00:01 |
|*  5 |      VIEW                                      |                             |   89M|  2218M|   869  (95)| 00:00:01 |
|   6 |       UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   7 |        VIEW                                    |                             |  5461 |  138K|     9   (0)| 00:00:01 |
|   8 |         UNION-ALL                              |                             |       |       |            |          |
|   9 |          TABLE ACCESS FULL                     | EMPLOYEE                    |  5460 |  138K|     7   (0)| 00:00:01 |
|  10 |          TABLE ACCESS FULL                     | ASSOCIATION                 |     1 |    26 |     2   (0)| 00:00:01 |
|  11 |        NESTED LOOPS                            |                             |   89M|  4437M|   860  (96)| 00:00:01 |
|  12 |         NESTED LOOPS                           |                             |   89M|  4437M|   860  (96)| 00:00:01 |
|  13 |          RECURSIVE WITH PUMP                   |                             |       |       |            |          |
|* 14 |          INDEX RANGE SCAN                      | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  15 |         TABLE ACCESS BY INDEX ROWID            | ORGANIZATIONUNIT            |     4 |   104 |     0   (0)| 00:00:01 |
|* 16 |    INDEX RANGE SCAN                            | A_CLIENT_IDX                |    75 |   975 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - filter("S"=1)
  14 - access("ORG"."PARENT"="MYSELF"."ID")
  16 - access("E"."OUID"="A"."OUID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:          Alle sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:     EXPLAIN PLAN
-- SQL-Statement:     SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
-- Kosten:            5
--
-- Hinweis:           Nach der Optimierung, Nutzung der Materialized View aus Anhang 11
--                    Eine Reduktion der Ausführungskosten von 99,98%
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------
| Id  | Operation                                | Name                | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |                     |     1 |    21 |     5   (0)| 00:00:01 |
|   1 |  SORT AGGREGATE                          |                     |     1 |    21 |            |          |
|   2 |   NESTED LOOPS                           |                     |   427 |  8967 |     5   (0)| 00:00:01 |
|   3 |    MAT_VIEW ACCESS BY INDEX ROWID BATCHED| EMPLOYEE_OU         |     6 |    48 |     5   (0)| 00:00:01 |
|*  4 |     INDEX RANGE SCAN                     | EMPLOYEE_OU_MV_IDX  |     6 |       |     1   (0)| 00:00:01 |
|*  5 |    INDEX RANGE SCAN                      | A_CLIENT_IDX        |    75 |   975 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   4 - access("E"."EID"=1)
   5 - access("E"."OUID"="A"."OUID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

13. Oracle – Messergebnisse – 10.000.000 Datensätze (Optimiert)

```
-- ==========================================================================================
--
-- Testfall:         Anzahl aller sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
-- Kosten:           ~228.000
--
-- Hinweis:          Vor der Optimierung, Nutzung einer klassischen View aus Anhang 3
--                   Diese Messwerte wurden für eine bessere Gegenüberstellung aus Anhang 10 hierher kopiert
--
-- ==========================================================================================

-----------------------------------------------------------------------------------------------------------------------
| Id  | Operation                                  | Name                        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                           |                             |     1 |    39 |   228K(100)| 00:00:09 |
|   1 |  SORT AGGREGATE                            |                             |     1 |    39 |            |          |
|   2 |   NESTED LOOPS                             |                             |   666G|   23T|   228K(100)| 00:00:09 |
|*  3 |    VIEW                                    |                             |    89M|  2218M|   869  (95)| 00:00:01 |
|   4 |     UNION ALL (RECURSIVE WITH) BREADTH FIRST|                             |       |       |            |          |
|   5 |      VIEW                                  |                             |  5461 |   138K|     9   (0)| 00:00:01 |
|   6 |       UNION-ALL                            |                             |       |       |            |          |
|   7 |        TABLE ACCESS FULL                   | EMPLOYEE                    |  5460 |   138K|     7   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL                   | ASSOCIATION                 |     1 |    26 |     2   (0)| 00:00:01 |
|   9 |      NESTED LOOPS                          |                             |    89M|  4437M|   860  (96)| 00:00:01 |
|  10 |       NESTED LOOPS                         |                             |    89M|  4437M|   860  (96)| 00:00:01 |
|  11 |        RECURSIVE WITH PUMP                 |                             |       |       |            |          |
|* 12 |        INDEX RANGE SCAN                    | ORGANIZATIONUNIT_PARENT_IDX |     4 |       |     0   (0)| 00:00:01 |
|  13 |       TABLE ACCESS BY INDEX ROWID          | ORGANIZATIONUNIT            |     4 |   104 |     0   (0)| 00:00:01 |
|* 14 |    INDEX RANGE SCAN                        | A_CLIENT_IDX                |  7444 | 96772 |     0   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("S"=1)
  12 - access("ORG"."PARENT"="MYSELF"."ID")
  14 - access("ID"="A"."OUID")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

```
-- ==========================================================================================
--
-- Testfall:         Alle sichtbaren Datensätze lesen (Führungskraft, EID = 1)
-- Mess-Funktion:    EXPLAIN PLAN
-- SQL-Statement:    SELECT COUNT(*) FROM MT_ASS_ADMIN.CLIENT WHERE EID = 1
-- Kosten:           604
--
-- Hinweis:          Nach der Optimierung, Nutzung der Materialized View aus Anhang 11
--                   Eine Reduktion der Ausführungskosten von 99,74%
--
-- ==========================================================================================

-------------------------------------------------------------------------------------------
| Id  | Operation               | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |               |     1 |    12 |   604   (1)| 00:00:01 |
|   1 |  SORT AGGREGATE         |               |     1 |    12 |            |          |
|*  2 |   HASH JOIN             |               |   107K|  1259K|   604   (1)| 00:00:01 |
|*  3 |    MAT_VIEW ACCESS FULL| EMPLOYEE_OU   |  1467 | 11736 |    19   (6)| 00:00:01 |
|   4 |    INDEX FAST FULL SCAN| A_CLIENT_IDX  |   100K|   390K|   584   (1)| 00:00:01 |
-------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("E"."OUID"="A"."OUID")
   3 - filter("E"."EID"=1)

Note
-----
   - this is an adaptive plan
```

14. SQL-Server – Bereinigung

```
-- ============================================
--
-- WICHTIG: Ausführen als 'sa' oder lokaler Administrator
--
-- Dient der Bereinigung bestehender Strukturen und sollte nur dann ausgeführt werden,
-- wenn bereits Strukturen aus diesem Anhang erzeugt wurden.
--
-- ============================================

-- Verwende "master" für administrative Aufgaben
USE [master];
GO

-- Schließe alle Verbindungen, die uns vom Löschen der Datenbank / Benutzer abhalten würde
DECLARE KillCursor CURSOR FAST_FORWARD FOR SELECT session_id FROM sys.dm_exec_sessions WHERE
login_name LIKE 'mt%';
DECLARE @SID INT;
DECLARE @ExecSQL VARCHAR(255);
OPEN KillCursor;
FETCH NEXT FROM KillCursor INTO @SID;
WHILE @@FETCH_STATUS = 0
BEGIN
  SET  @ExecSQL = 'KILL ' + CAST(@SID AS VARCHAR(50));
  EXEC (@ExecSQL);
  FETCH NEXT FROM KillCursor INTO @SID;
END;
CLOSE KillCursor;
DEALLOCATE KillCursor;
GO

-- Lösche "alte" Konten
IF EXISTS(SELECT name FROM [master].[sys].[syslogins] WHERE NAME = 'mt_ass_admin')
BEGIN
    DROP LOGIN mt_ass_admin;
END;
IF EXISTS(SELECT name FROM [master].[sys].[syslogins] WHERE NAME = 'mt_org_admin')
BEGIN
    DROP LOGIN mt_org_admin;
END;
IF EXISTS(SELECT name FROM [master].[sys].[syslogins] WHERE NAME = 'mt_data_admin')
BEGIN
    DROP LOGIN mt_data_admin;
END;
IF EXISTS(SELECT name FROM [master].[sys].[syslogins] WHERE NAME = 'mt_ass_user')
BEGIN
    DROP LOGIN mt_ass_user;
END;
IF EXISTS(SELECT name FROM [master].[sys].[syslogins] WHERE NAME = 'mt_org_user')
BEGIN
    DROP LOGIN mt_org_user;
END;
GO

-- Lösche "alte" Datenbank
ALTER DATABASE [MT] SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
DROP DATABASE MT;
GO
```

15. SQL-Server – Erstelle Datenbank, Datafiles, Login u. Benutzer

```
-- ============================================
--
-- WICHTIG: Ausführen als 'sa' oder lokaler Administrator
--
-- ============================================

-- Verwende "master" für administrative Aufgaben
USE [master];
GO

-- Erzeuge Datenbank
CREATE DATABASE MT
ON PRIMARY (
    NAME = 'MT',     FILENAME = 'A:\DBs\MSSQL\MT.mdf',
    SIZE = 32MB, MAXSIZE = 4096MB, FILEGROWTH = 32MB
) LOG ON (
    NAME = 'MT_log', FILENAME = 'A:\DBs\MSSQL\MT_log.ldf',
    SIZE = 32MB, MAXSIZE = 2048MB, FILEGROWTH = 32MB
);
GO

-- Erzeuge Logins + Kennwort
CREATE LOGIN mt_ass_admin  WITH PASSWORD = 'mt_ass_admin',  DEFAULT_DATABASE = MT;
CREATE LOGIN mt_ass_user   WITH PASSWORD = 'mt_ass_user',   DEFAULT_DATABASE = MT;
CREATE LOGIN mt_org_admin  WITH PASSWORD = 'mt_org_admin',  DEFAULT_DATABASE = MT;
CREATE LOGIN mt_org_user   WITH PASSWORD = 'mt_org_user',   DEFAULT_DATABASE = MT;
CREATE LOGIN mt_data_admin WITH PASSWORD = 'mt_data_admin', DEFAULT_DATABASE = MT;
GO

-- Wechsel in erzeugte MT-Datenbank
USE [MT]
GO

-- Erzeuge Benutzer in MT (Verknüpfung mit Login-Kennung)
CREATE USER mt_ass_admin  FOR LOGIN mt_ass_admin  WITH DEFAULT_SCHEMA = [mt_ass];
CREATE USER mt_ass_user   FOR LOGIN mt_ass_user   WITH DEFAULT_SCHEMA = [mt_ass];
CREATE USER mt_org_admin  FOR LOGIN mt_org_admin  WITH DEFAULT_SCHEMA = [mt_org];
CREATE USER mt_org_user   FOR LOGIN mt_org_user   WITH DEFAULT_SCHEMA = [mt_org];
CREATE USER mt_data_admin FOR LOGIN mt_data_admin WITH DEFAULT_SCHEMA = [mt_data];
GO

-- Erzeuge Schemata
CREATE SCHEMA mt_ass  AUTHORIZATION  mt_ass_admin;
GO
CREATE SCHEMA mt_org  AUTHORIZATION  mt_org_admin;
GO
CREATE SCHEMA mt_data AUTHORIZATION  mt_data_admin;
GO

-- ==========================================================================================

-- Schema-Rechte
GRANT SELECT, INSERT, UPDATE, DELETE             ON SCHEMA :: mt_org  TO mt_org_user;
GRANT SELECT                                     ON SCHEMA :: mt_org  TO mt_ass_user;
GRANT SELECT, REFERENCES                         ON SCHEMA :: mt_org  TO mt_ass_admin;
GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES ON SCHEMA :: mt_data TO mt_ass_admin;
GRANT SELECT, INSERT, UPDATE, DELETE             ON SCHEMA :: mt_data TO mt_ass_user;
GRANT SELECT, INSERT, UPDATE, DELETE             ON SCHEMA :: mt_ass  TO mt_ass_user;

-- Datenbank-Rechte
GRANT CREATE TABLE, CREATE VIEW TO mt_org_admin;
GRANT CREATE TABLE, CREATE VIEW TO mt_ass_admin;
GRANT CREATE TABLE, CREATE VIEW TO mt_data_admin;

-- Wird für die Erstellung der Test-Hierarchie sowie die Performance-Messungen benötigt!
GRANT CREATE PROCEDURE TO mt_org_admin;
GRANT SHOWPLAN TO mt_ass_admin;
GO
```

16. SQL-Server – Erstelle Schema – Organisation

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_org_admin'
--
-- ============================================

USE [MT];
GO

-- Tabellen anlegen
CREATE TABLE ORGANIZATIONUNIT (
    ID INT NOT NULL    IDENTITY(1,1) PRIMARY KEY,
    PARENT INT         REFERENCES ORGANIZATIONUNIT(ID),
    NAME NVARCHAR(128),
    CONSTRAINT OU_NO_REFLEXIVE CHECK (ID <> PARENT)
);
CREATE INDEX ORGANIZATIONUNIT_PARENT_IDX ON ORGANIZATIONUNIT (PARENT);

CREATE TABLE EMPLOYEE (
    ID INT NOT NULL    IDENTITY(1,1) PRIMARY KEY,
    OU INT NOT NULL    REFERENCES ORGANIZATIONUNIT(ID),
    NAME NVARCHAR(128)
);
CREATE INDEX EMPLOYEE_OU_IDX ON EMPLOYEE (OU);

CREATE TABLE ASSOCIATION (
    EID INT NOT NULL   REFERENCES EMPLOYEE(ID),
    OUID INT NOT NULL  REFERENCES ORGANIZATIONUNIT(ID),
    CONSTRAINT ASSOCIATION_PK PRIMARY KEY (EID, OUID)
);
CREATE INDEX ASSOCIATION_EID_IDX  ON ASSOCIATION (EID);
CREATE INDEX ASSOCIATION_OUID_IDX ON ASSOCIATION (OUID);
GO


-- ==========================================================================================


-- Trigger gegen Zyklen-Bildung bei UPDATE
-- Default: Rekursionstiefe 100
CREATE OR ALTER TRIGGER OU_UPDATE_TRG
ON ORGANIZATIONUNIT FOR UPDATE AS
BEGIN
    BEGIN TRY
        WITH recDat (ID) AS (
            SELECT org.ID AS "ID" FROM ORGANIZATIONUNIT org
            UNION ALL
            SELECT org.ID AS "ID" FROM ORGANIZATIONUNIT org
            INNER JOIN recDat ON org.PARENT = recDat.ID
        )
        SELECT COUNT(*) FROM recDat; -- Zyklus provozieren
    END TRY
    BEGIN CATCH
        IF (ERROR_NUMBER() = 530)
            THROW 66642, 'Potentieller Zyklus erkannt!', 1;
        ELSE
            THROW;
    END CATCH;
END;
GO
```

```
-- View erstellen
CREATE OR ALTER VIEW EMPLOYEE_OU
AS
WITH MySelf (S, ID) AS (

    -- Startpunkt 1: Primäre OU ermitteln
    SELECT
        e.ID       AS "S",  -- Start E-ID
        org.ID     AS "ID"  -- OU-ID
    FROM employee e
    INNER JOIN organizationunit org ON org.ID = e.OU

    UNION ALL

    -- Startpunkt 2: Sekundäre OUs ermitteln
    SELECT
        e.ID       AS "S",   -- Start E-ID
        org.ID     AS "ID"
    FROM employee e
    INNER JOIN association a ON a.EID = e.ID
    INNER JOIN organizationunit org ON org.ID = a.OUID

    UNION ALL

    -- Rekursiv darunter liegende Sekundär-OUs ermitteln
    SELECT
        MySelf.S   AS "S",  -- Start E-ID
        org.ID     AS "ID"
    FROM organizationunit org
    INNER JOIN MySelf ON org.PARENT = MySelf.ID
)
SELECT S as "EID", ID as "OUID" FROM MySelf GROUP BY S, ID;
GO

-- ==========================================================================================


GRANT SELECT   ON OBJECT :: ORGANIZATIONUNIT TO mt_ass_admin;
GRANT SELECT   ON OBJECT :: EMPLOYEE_OU TO mt_ass_admin;
GO
```

17. SQL-Server – Erstelle Schema – Fachliches Datenmodell

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_data_admin'
--
-- ============================================

USE [MT];
GO

-- Tabellen anlegen
CREATE TABLE PRODUCT (
    ID INT NOT NULL      IDENTITY(1,1) PRIMARY KEY,
    NAME NVARCHAR(128)
);
CREATE TABLE CLIENT (
    ID INT NOT NULL      IDENTITY(1,1) PRIMARY KEY,
    NAME NVARCHAR(128)
);
CREATE TABLE "ORDER" (
    ID INT NOT NULL      IDENTITY(1,1) PRIMARY KEY,
    CID INT              REFERENCES CLIENT (ID),
    ORDER_DATETIME DATETIME DEFAULT CURRENT_TIMESTAMP
);
CREATE INDEX ORDER_CID_IDX  ON "ORDER" (CID);
CREATE TABLE ORDERPRODUCT (
    OID INT NOT NULL     REFERENCES "ORDER" (ID),
    PID INT NOT NULL     REFERENCES PRODUCT (ID),
    AMOUNT INT DEFAULT 1,
    CONSTRAINT ORDERPRODUCT_PK PRIMARY KEY (OID, PID)
);
CREATE INDEX ORDERPRODUCT_OID_IDX  ON ORDERPRODUCT (OID);
CREATE INDEX ORDERPRODUCT_PID_IDX  ON ORDERPRODUCT (PID);
GO
```

18. SQL-Server – Erstelle Schema – Assoziation

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_ass_admin'
--
-- ============================================

USE [MT];
GO

-- View für Tabelle ohne Zugriffsschicht erstellen (PRODUCT)
CREATE VIEW PRODUCT
AS
    SELECT * FROM [MT].[mt_data].[PRODUCT];
GO

-- View für Tabelle ohne Zugriffsschicht erstellen (ORDERPRODUCT)
CREATE VIEW ORDERPRODUCT
AS
    SELECT * FROM [MT].[mt_data].[ORDERPRODUCT];
GO


-- ==========================================================================================


-- Assoziations-Tabelle für fachliche Tabelle erstellen (CLIENT)
CREATE TABLE A_CLIENT (
    ID INT NOT NULL         PRIMARY KEY,
    OUID INT NOT NULL,
    CONSTRAINT A_CLIENT_OU_FK FOREIGN KEY(OUID) REFERENCES [MT].[mt_org].ORGANIZATIONUNIT(ID),
    CONSTRAINT A_CLIENT_FK    FOREIGN KEY(ID)   REFERENCES [MT].[mt_data].CLIENT(ID)
);
CREATE INDEX A_CLIENT_IDX ON A_CLIENT (OUID);
GO

-- View für Tabelle mit Zugriffsschicht erstellen
CREATE VIEW CLIENT
AS
    -- Alle auswählen...
    SELECT c.*, e.EID FROM [MT].[mt_data].CLIENT c
    -- ...die eine Verknüfung in der Association-Tabelle haben...
    INNER JOIN A_CLIENT a ON a.ID = c.ID
    -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
    INNER JOIN [MT].[mt_org].EMPLOYEE_OU e ON e.OUID = a.OUID;
GO

-- ==========================================================================================

-- Assoziations-Tabelle für fachliche Tabelle erstellen (ORDER)
CREATE TABLE A_ORDER (
    ID INT NOT NULL        PRIMARY KEY,
    OUID INT NOT NULL,
    CONSTRAINT ORDER_OU_FK FOREIGN KEY(OUID) REFERENCES [MT].[mt_org].ORGANIZATIONUNIT(ID),
    CONSTRAINT ORDER_FK    FOREIGN KEY(ID)   REFERENCES [MT].[mt_data]."ORDER"(ID)
);
CREATE INDEX A_ORDER_IDX ON A_ORDER (OUID);
GO

-- View für Tabelle mit Zugriffsschicht erstellen
CREATE VIEW "ORDER"
AS
    -- Alle auswählen...
    SELECT c.*, e.EID FROM [MT].[mt_data]."ORDER" c
    -- ...die eine Verknüfung in der Association-Tabelle haben...
    INNER JOIN A_ORDER a ON a.ID = c.ID
    -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
    INNER JOIN [MT].[mt_org].EMPLOYEE_OU e ON e.OUID = a.OUID;
GO
-- INSERT - Erstelle Instead-of Trigger für View CLIENT
```

```
CREATE OR ALTER TRIGGER CLIENT_INSERT_TRG
    ON [MT].[mt_ass].CLIENT INSTEAD OF INSERT AS
BEGIN
    DECLARE @OUID  INT;

    DECLARE insertCursor CURSOR FAST_FORWARD FOR SELECT EID, NAME FROM INSERTED;
    DECLARE @EID   INT;
    DECLARE @NAME  NVARCHAR(128);

    OPEN insertCursor;
    FETCH NEXT FROM insertCursor INTO @EID, @NAME;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Ermitteln der Eigentümer-OU
        SELECT @OUID = OU FROM [MT].[mt_org].EMPLOYEE WHERE ID = @EID;
        -- Speichern der Daten
        INSERT INTO [MT].[mt_data].CLIENT (NAME) VALUES (@NAME);
        -- Speichern der Verknüpfung OU <--> Datensatz
        INSERT INTO [MT].[mt_ass].A_CLIENT VALUES ((SELECT SCOPE_IDENTITY()), @OUID);
        -- Nächster Datensatz
        FETCH NEXT FROM insertCursor INTO @EID, @NAME;
    END;
    CLOSE insertCursor;
    DEALLOCATE insertCursor;
END;
GO


-- UPDATE - Erstelle Instead-of Trigger für View CLIENT
CREATE OR ALTER TRIGGER CLIENT_UPDATE_TRG
    ON [MT].[mt_ass].CLIENT INSTEAD OF UPDATE AS
BEGIN
    DECLARE @OUID INT;

    DECLARE updateCursor CURSOR FAST_FORWARD FOR SELECT ID, EID, NAME FROM INSERTED;
    DECLARE @ID   INT;
    DECLARE @EID  INT;
    DECLARE @NAME NVARCHAR(128);

    OPEN updateCursor;
    FETCH NEXT FROM updateCursor INTO @ID, @EID, @NAME;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Eigentümer-OU ändern, wenn gewollt
        IF (UPDATE(EID))
        BEGIN
            -- Lese neue OU-ID
            SELECT @OUID = OU FROM [MT].[mt_org].EMPLOYEE WHERE ID = @EID;
            -- Setze neue OU-ID für diesen Datensatz
            UPDATE [MT].[mt_ass].A_CLIENT SET OUID = @OUID WHERE ID = @ID;
        END;
        -- Datensatz ändern
        UPDATE [MT].[mt_data].CLIENT SET NAME = @NAME WHERE ID = @ID;
        -- Nächster Datensatz
        FETCH NEXT FROM updateCursor INTO @ID, @EID, @NAME;
    END;
    CLOSE updateCursor;
    DEALLOCATE updateCursor;
END;
GO
```

```
-- DELETE - Erstelle Instead-of Trigger für View CLIENT
CREATE OR ALTER TRIGGER CLIENT_DELETE_TRG
    ON [MT].[mt_ass].CLIENT INSTEAD OF DELETE AS
BEGIN
    DECLARE deleteCursor CURSOR FAST_FORWARD FOR SELECT ID FROM DELETED;
    DECLARE @ID INT;

    OPEN deleteCursor;
    FETCH NEXT FROM deleteCursor INTO @ID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        DELETE FROM [MT].[mt_ass].A_CLIENT WHERE ID = @ID;
        DELETE FROM [MT].[mt_data].CLIENT  WHERE ID = @ID;
        FETCH NEXT FROM deleteCursor INTO @ID;
    END;
    CLOSE deleteCursor;
    DEALLOCATE deleteCursor;
END;
GO


-- ==========================================================================================


-- INSERT - Erstelle Instead-of-Trigger für View ORDER
CREATE OR ALTER TRIGGER ORDER_INSERT_TRG
    ON [MT].[mt_ass]."ORDER" INSTEAD OF INSERT AS
BEGIN
    DECLARE @OUID           INT;

    DECLARE insertCursor CURSOR FAST_FORWARD FOR SELECT EID, CID, ORDER_DATETIME FROM INSERTED;
    DECLARE @EID            INT;
    DECLARE @CID            INT;
    DECLARE @ORDER_DATETIME DATETIME;

    OPEN insertCursor;
    FETCH NEXT FROM insertCursor INTO @EID, @CID, @ORDER_DATETIME;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Ermitteln der Eigentümer-OU
        SELECT @OUID = OU FROM [MT].[mt_org].EMPLOYEE WHERE ID = @EID;
        -- Speichern der Daten
        INSERT INTO [MT].[mt_data]."ORDER" (CID, ORDER_DATETIME) VALUES (@CID, @ORDER_DATETIME);
        -- Speichern der Verknüpfung OU <--> Datensatz
        INSERT INTO [MT].[mt_ass].A_ORDER VALUES ((SELECT SCOPE_IDENTITY()), @OUID);
        -- Nächster Datensatz
        FETCH NEXT FROM insertCursor INTO @EID, @CID, @ORDER_DATETIME;
    END;
    CLOSE insertCursor;
    DEALLOCATE insertCursor;
END;
GO
```

```
-- UPDATE - Erstelle Instead-of-Trigger für View ORDER
CREATE OR ALTER TRIGGER ORDER_UPDATE_TRG
    ON [MT].[mt_ass]."ORDER" INSTEAD OF UPDATE AS
BEGIN
    DECLARE @OUID           INT;

    DECLARE updateCursor CURSOR FAST_FORWARD FOR
        SELECT ID, EID, CID, ORDER_DATETIME FROM INSERTED;

    DECLARE @ID             INT;
    DECLARE @EID            INT;
    DECLARE @CID            INT;
    DECLARE @ORDER_DATETIME DATETIME;

    OPEN updateCursor;
    FETCH NEXT FROM updateCursor INTO @ID, @EID, @CID, @ORDER_DATETIME;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Eigentümer-OU ändern, wenn gewollt
        IF (UPDATE(EID))
        BEGIN
            -- Lese neue OU-ID
            SELECT @OUID = OU FROM [MT].[mt_org].EMPLOYEE WHERE ID = @EID;
            -- Setze neue OU-ID für diesen Datensatz
            UPDATE [MT].[mt_ass].A_ORDER SET OUID = @OUID WHERE ID = @ID;
        END;
        -- Datensatz ändern
        UPDATE [MT].[mt_data]."ORDER"
        SET CID = @CID, ORDER_DATETIME = @ORDER_DATETIME
        WHERE ID = @ID;
        -- Nächster Datensatz
        FETCH NEXT FROM updateCursor INTO @ID, @EID, @CID, @ORDER_DATETIME;
    END;
    CLOSE updateCursor;
    DEALLOCATE updateCursor;
END;
GO


-- DELETE - Erstelle Instead-of-Trigger
CREATE OR ALTER TRIGGER ORDER_DELETE_TRG
    ON [MT].[mt_ass]."ORDER" INSTEAD OF DELETE AS
BEGIN
    DECLARE @ID   INT;
    DECLARE deleteCursor CURSOR FAST_FORWARD FOR SELECT ID FROM DELETED;

    OPEN deleteCursor;
    FETCH NEXT FROM deleteCursor INTO @ID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        DELETE FROM [MT].[mt_ass].A_ORDER  WHERE ID = @ID;
        DELETE FROM [MT].[mt_data]."ORDER" WHERE ID = @ID;
        FETCH NEXT FROM deleteCursor INTO @ID;
    END;
    CLOSE deleteCursor;
    DEALLOCATE deleteCursor;
END;
GO
```

19. SQL-Server – Testdaten erstellen – Hierarchie, Mitarbeiter

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_ass_admin'
--
-- ============================================


USE [MT];
GO


-- Erzeugt einen Mitarbeiter-Datensatz in der übergebenen OU
CREATE OR ALTER PROCEDURE CREATE_TEST_EMPLOYEE
    (@organizationUnit INT, @employeePerOu INT)
AS
BEGIN
    DECLARE @tmpCount INT = 0;
    WHILE @tmpCount < @employeePerOu
    BEGIN
        INSERT INTO [MT].[mt_org].EMPLOYEE (OU, NAME)
        VALUES (@organizationUnit, 'E_' + left(NEWID(),5) + left(NEWID(),5));
        SET @tmpCount = @tmpCount + 1;
    END;
END;
GO


-- Erzeugt für die übergebene OU die untergeordneten OU
CREATE OR ALTER PROCEDURE CREATE_TEST_HIERARY
    (@parentID INT, @currentLevel INT, @maxHierarchyHight INT,
     @amountOuPerLevel INT, @employeePerOu INT)
AS
BEGIN
    IF @currentLevel > @maxHierarchyHight
        RETURN;

    DECLARE @tmpCount INT = 0;
    WHILE @tmpCount < @amountOuPerLevel
    BEGIN
        INSERT INTO [MT].[mt_org].ORGANIZATIONUNIT (PARENT, NAME)
        VALUES (@parentID, 'OU_' + left(NEWID(),5) + left(NEWID(),5));
        DECLARE @newId INT = SCOPE_IDENTITY();

        EXEC CREATE_TEST_EMPLOYEE
            @organizationUnit = @newId,
            @employeePerOu    = @employeePerOu;

        DECLARE @nextLevel INT = @currentLevel + 1;
        EXEC CREATE_TEST_HIERARY
            @parentID          = @newId,
            @currentLevel      = @nextLevel,
            @maxHierarchyHight = @maxHierarchyHight,
            @amountOuPerLevel  = @amountOuPerLevel,
            @employeePerOu     = @employeePerOu;

        SET @tmpCount = @tmpCount + 1;
    END;
END;
GO
```

```
-- Erstellt die oberste OU (Root-Element) und startet den Vorgang
CREATE OR ALTER PROCEDURE CREATE_TEST_START
    (@maxHierarchyHight INT, @amountOuPerLevel INT, @employeePerOu INT)
AS
BEGIN
    BEGIN TRANSACTION;
    INSERT INTO [MT].[mt_org].ORGANIZATIONUNIT (PARENT, NAME) VALUES (NULL, 'FIRST_OU');
    DECLARE @newId INT = SCOPE_IDENTITY();

    EXEC CREATE_TEST_EMPLOYEE
        @organizationUnit = @newId,
        @employeePerOu    = @employeePerOu;

    EXEC CREATE_TEST_HIERARY
        @parentID          = @newId,
        @currentLevel      = 1,
        @maxHierarchyHight = @maxHierarchyHight,
        @amountOuPerLevel  = @amountOuPerLevel,
        @employeePerOu     = @employeePerOu;
    COMMIT;
END;
GO


-- Deaktiviert die Ausgabe der Zeilen die verändert wurden und steigert so die Performance
SET NOCOUNT ON;


-- Starte Erzeugen von Test-Hierarchie für Messung
EXEC CREATE_TEST_START
    @maxHierarchyHight = 5,
    @amountOuPerLevel  = 4,
    @employeePerOu     = 4;
GO


-- Prozeduren löschen
DROP PROCEDURE CREATE_TEST_EMPLOYEE;
DROP PROCEDURE CREATE_TEST_HIERARY;
DROP PROCEDURE CREATE_TEST_START;
GO
```

20. SQL-Server – Testdaten erstellen – Fachliche Datensätze

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_ass_admin'
--
-- ============================================


/*
  Erzeugen von Testdaten:
  -----------------------------------
       1.000   ->      0,4 Sekunden
      10.000   ->      4,1 Sekunden
     100.000   ->     41,7 Sekunden
   1.000.000   ->    398,9 Sekunden
  10.000.000   ->   4110,0 Sekunden
*/


DECLARE allEmployee CURSOR FAST_FORWARD FOR SELECT ID FROM [MT].[mt_org].EMPLOYEE;
DECLARE @EID INT;
DECLARE @maxTestData INT = 100000;   -- Zu erzeugende Anzahl an Testdaten
DECLARE @maxTestDataCounter INT = 0; -- Zähler-Variable

-- Deaktiviert die Ausgabe der Zeilen die verändert wurden und steigert so die Performance
SET NOCOUNT ON;

WHILE @maxTestDataCounter < @maxTestData
BEGIN
    OPEN allEmployee;
    FETCH NEXT FROM allEmployee INTO @EID;
    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF @maxTestDataCounter = @maxTestData
            BREAK;

        INSERT INTO [MT].[mt_ass].CLIENT (NAME, EID)
        VALUES ('Client_' + left(NEWID(),5) + left(NEWID(),5), @EID);

        SET @maxTestDataCounter = @maxTestDataCounter + 1;
        FETCH NEXT FROM allEmployee INTO @EID;
    END;
    CLOSE allEmployee;
END;
DEALLOCATE allEmployee;
GO
```

21. MariaDB – Bereinigung

```
-- ============================================
--
-- WICHTIG: Ausführen als 'root'
--
-- Dient der Bereinigung bestehender Strukturen und sollte nur dann ausgeführt werden,
-- wenn bereits Strukturen aus diesem Anhang erzeugt wurden.
--
-- ============================================


USE `information_schema`;

DROP SCHEMA IF EXISTS `mt_ass`;
DROP SCHEMA IF EXISTS `mt_data`;
DROP SCHEMA IF EXISTS `mt_org`;

DROP USER IF EXISTS `mt_org_admin`@`localhost`;
DROP USER IF EXISTS `mt_org_user`@`localhost`;
DROP USER IF EXISTS `mt_ass_admin`@`localhost`;
DROP USER IF EXISTS `mt_ass_user`@`localhost`;
DROP USER IF EXISTS `mt_data_admin`@`localhost`;
```

22. MariaDB – Erstelle Datenbanken, Benutzer

```
-- ============================================
--
-- WICHTIG: Ausführen als 'root'
--
-- ============================================


DROP SCHEMA IF EXISTS `mt_ass`;
DROP SCHEMA IF EXISTS `mt_data`;
DROP SCHEMA IF EXISTS `mt_org`;
DROP USER IF EXISTS `mt_org_admin`@`localhost`;
DROP USER IF EXISTS `mt_org_user`@`localhost`;
DROP USER IF EXISTS `mt_ass_admin`@`localhost`;
DROP USER IF EXISTS `mt_ass_user`@`localhost`;
DROP USER IF EXISTS `mt_data_admin`@`localhost`;


-- ==========================================================================================


-- Schema / Datenbanken erstellen
CREATE SCHEMA `mt_org`;
CREATE SCHEMA `mt_data`;
CREATE SCHEMA `mt_ass`;


-- Benutzer erstellen
CREATE USER `mt_org_admin`@`localhost`  IDENTIFIED BY 'mt_org_admin';
CREATE USER `mt_org_user`@`localhost`   IDENTIFIED BY 'mt_org_user';
CREATE USER `mt_ass_admin`@`localhost`  IDENTIFIED BY 'mt_ass_admin';
CREATE USER `mt_ass_user`@`localhost`   IDENTIFIED BY 'mt_ass_user';
CREATE USER `mt_data_admin`@`localhost` IDENTIFIED BY 'mt_data_admin';


-- Admin Benutzer: ALL PRIVILEGES
GRANT ALL PRIVILEGES ON `mt_org`.*  TO `mt_org_admin`@`localhost`  WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON `mt_ass`.*  TO `mt_ass_admin`@`localhost`  WITH GRANT OPTION;
GRANT ALL PRIVILEGES ON `mt_data`.* TO `mt_data_admin`@`localhost` WITH GRANT OPTION;


-- Festschreiben der Rechte
FLUSH PRIVILEGES;
```

23. MariaDB – Erstelle Schema – Organisation

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_org_admin'
--
-- ============================================

USE `mt_org`;

CREATE TABLE ORGANIZATIONUNIT (
    `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `PARENT` INT REFERENCES ORGANIZATIONUNIT(ID),
    `NAME` VARCHAR(128) NOT NULL DEFAULT ''
);
CREATE INDEX ORGANIZATIONUNIT_PARENT_IDX ON ORGANIZATIONUNIT (PARENT);
-- CONSTRAINT OU_NO_REFLEXIVE CHECK (ID <> PARENT)
-- => SQL Fehler (1901): Function or expression 'AUTO_INCREMENT'
--                       cannot be used in the CHECK clause of `ID`

CREATE TABLE EMPLOYEE (
    `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `OU` INT REFERENCES ORGANIZATIONUNIT(ID),
    `NAME` VARCHAR(128) NOT NULL DEFAULT ''
);
CREATE INDEX EMPLOYEE_OU_IDX ON EMPLOYEE (OU);

CREATE TABLE ASSOCIATION (
    `EID`  INT NOT NULL  REFERENCES EMPLOYEE(ID),
    `OUID` INT NOT NULL  REFERENCES ORGANIZATIONUNIT(ID),
    `CONSTRAINT ASSOCIATION_PK PRIMARY KEY (EID, OUID)
);
CREATE INDEX ASSOCIATION_EID_IDX  ON ASSOCIATION (EID);
CREATE INDEX ASSOCIATION_OUID_IDX ON ASSOCIATION (OUID);

-- ==========================================================================================

CREATE OR REPLACE
    DEFINER='mt_org_admin'@'localhost'
    SQL SECURITY DEFINER
VIEW EMPLOYEE_OU
AS
WITH RECURSIVE MySelf (S, ID) AS (
    -- Startpunkt 1: Primäre OU ermitteln
    SELECT
        e.ID       AS "S",  -- Start E-ID
        org.ID     AS "ID"  -- OU-ID
    FROM employee e
    INNER JOIN organizationunit org ON org.ID = e.OU

    UNION ALL

    -- Startpunkt 2: Sekundäre OUs ermitteln
    SELECT
        e.ID       AS "S",  -- Start E-ID
        org.ID     AS "ID"  -- OU-ID
    FROM employee e
    INNER JOIN association a ON a.EID = e.ID
    INNER JOIN organizationunit org ON org.ID = a.OUID

    UNION ALL

    -- Rekursiv darunter liegende OUs ermitteln
    SELECT
        MySelf.S   AS "S",  -- Start E-ID
        org.ID     AS "ID"  -- OU-ID
    FROM organizationunit org
    INNER JOIN MySelf ON org.PARENT = MySelf.ID
)
SELECT S as "EID", ID as "OUID" FROM MySelf GROUP BY S, ID;
```

```
-- Trigger 1, Vermeidung von Zyklen bei INSERT
-- Ersatz für nicht unterstützten Constraint, siehe oben
DELIMITER //
CREATE OR REPLACE TRIGGER OU_INSERT_TRG
BEFORE INSERT
    ON organizationunit FOR EACH ROW
BEGIN
    IF NEW.PARENT = NEW.ID THEN
        SIGNAL SQLSTATE '20667'
        SET MYSQL_ERRNO = 20667,
        MESSAGE_TEXT = 'Potentieller Zyklus erkannt!';
    END IF;
END; //
DELIMITER ;


-- Trigger 2, Vermeidung von Zyklen bei UPDATE
DELIMITER //
CREATE OR REPLACE TRIGGER OU_UPDATE_TRG
BEFORE UPDATE
    ON organizationunit FOR EACH ROW
BEGIN
    DECLARE vUser INT;
    WITH RECURSIVE recDat (ID) AS (
        SELECT org.ID as "ID" FROM organizationunit org
        WHERE org.ID = OLD.ID
        UNION ALL
        SELECT org.ID as "ID" FROM organizationunit org
        INNER JOIN recDat ON org.PARENT = recDat.ID
    ) SELECT COUNT(*) INTO vUser FROM recDat WHERE recDat.ID = NEW.PARENT;
    IF vUser > 0 OR NEW.PARENT = NEW.ID THEN
        SIGNAL SQLSTATE '20667'
        SET MYSQL_ERRNO = 20667,
        MESSAGE_TEXT = 'Potentieller Zyklus erkannt!';
    END IF;
END; //
DELIMITER ;


-- ==========================================================================================


-- Rechte für Zugriff erteilen
GRANT INSERT, UPDATE, SELECT, DELETE ON `mt_org`.*                TO `mt_org_user`@`localhost`;
GRANT SELECT, REFERENCES             ON `mt_org`.ORGANIZATIONUNIT TO `mt_ass_admin`@`localhost`;
GRANT SELECT                         ON `mt_org`.EMPLOYEE_OU      TO `mt_ass_admin`@`localhost`;
GRANT SELECT                         ON `mt_org`.EMPLOYEE         TO `mt_ass_admin`@`localhost`;
```

24. MariaDB – Erstelle Schema – Fachliches Datenmodell

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_data_admin'
--
-- ============================================

USE `mt_data`;

-- Tabellen anlegen
CREATE TABLE PRODUCT (
    `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `NAME` VARCHAR(128) NOT NULL DEFAULT ''
);

CREATE TABLE CLIENT (
    `ID` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `NAME` VARCHAR(128) NOT NULL DEFAULT ''
);

CREATE TABLE `ORDER` (
    `ID` INT  NOT NULL PRIMARY KEY AUTO_INCREMENT,
    `CID` INT REFERENCES CLIENT(ID),
    `ORDER_DATETIME` TIMESTAMP
);
CREATE INDEX ORDER_CID_IDX  ON `ORDER` (CID);

CREATE TABLE ORDERPRODUCT (
    `OID` INT NOT NULL     REFERENCES `ORDER` (ID),
    `PID` INT NOT NULL     REFERENCES PRODUCT (ID),
    `AMOUNT` INT DEFAULT 1,
    CONSTRAINT PRIMARY KEY (OID, PID)
);
CREATE INDEX ORDERPRODUCT_OID_IDX  ON ORDERPRODUCT (OID);
CREATE INDEX ORDERPRODUCT_PID_IDX  ON ORDERPRODUCT (PID);


-- ==========================================================================================


-- Berechtigungen für Association-Admin.
-- Der Normale Benutzer nutzt später Views / Prozeduren, welche im Kontext des Admin laufen
GRANT INSERT, UPDATE, SELECT, DELETE, REFERENCES ON `mt_data`.* TO `mt_ass_admin`@`localhost`;
```

25. MariaDB – Erstelle Schema – Assoziation

```
-- ============================================
--
-- WICHTIG: Ausführen als 'mt_ass_admin'
--
-- ============================================

USE `mt_ass`;

-- View für Tabelle ohne Zugriffsschicht erstellen (PRODUCT)
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
    SQL SECURITY DEFINER
VIEW PRODUCT
AS
    SELECT * FROM mt_data.PRODUCT;

-- View für Tabelle ohne Zugriffsschicht erstellen (ORDERPRODUCT)
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
    SQL SECURITY DEFINER
VIEW ORDERPRODUCT
AS
    SELECT * FROM mt_data.ORDERPRODUCT;


-- ==========================================================================================


-- Assoziations-Tabelle für fachliche Tabelle erstellen (CLIENT)
CREATE TABLE A_CLIENT (
    `ID`   INT NOT NULL       PRIMARY KEY,
    `OUID` INT NOT NULL,
    CONSTRAINT CLIENT_OU_FK FOREIGN KEY(OUID) REFERENCES mt_org.ORGANIZATIONUNIT(ID),
    CONSTRAINT CLIENT_FK    FOREIGN KEY(ID)   REFERENCES mt_data.CLIENT(ID)
);
CREATE INDEX A_CLIENT_IDX ON A_CLIENT (OUID);

-- Assoziations-Tabelle für fachliche Tabelle erstellen (ORDER)
CREATE TABLE A_ORDER (
    ID INT NOT NULL        PRIMARY KEY,
    OUID INT NOT NULL,
    CONSTRAINT ORDER_OU_FK FOREIGN KEY(OUID) REFERENCES mt_org.ORGANIZATIONUNIT(ID),
    CONSTRAINT ORDER_FK    FOREIGN KEY(ID)   REFERENCES mt_data.`ORDER`(ID)
);
CREATE INDEX A_ORDER_IDX ON A_ORDER (OUID);


-- ==========================================================================================


-- View für Tabelle mit Zugriffsschicht erstellen (CLIENT)
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
    SQL SECURITY DEFINER
VIEW CLIENT
AS
    -- Alle auswählen...
    SELECT c.*, e.EID FROM mt_data.CLIENT c
    -- ...die eine Verknüfung in der Association-Tabelle haben...
    INNER JOIN A_CLIENT a ON a.ID = c.ID
    -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
    INNER JOIN mt_org.EMPLOYEE_OU e ON e.OUID = a.OUID;
```

```
-- View für Tabelle mit Zugriffsschicht erstellen (ORDER)
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
    SQL SECURITY DEFINER
VIEW `ORDER`
AS
    -- Alle auswählen...
    SELECT c.*, e.EID FROM mt_data.`ORDER` c
    -- ...die eine Verknüfung in der Association-Tabelle haben...
    INNER JOIN A_ORDER a ON a.ID = c.ID
    -- ...und mit einer OU verbunden sind, die unserem Employee zugeordnet ist
    INNER JOIN mt_org.EMPLOYEE_OU e ON e.OUID = a.OUID;

-- ==========================================================================================

-- INSERT - Ersatz für fehlenden Instead-Of Trigger auf "CLIENT"
DELIMITER //
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
FUNCTION CLIENT_INSERT (IN_CALLER_EID INT, IN_ID INT, IN_NAME VARCHAR(128))
    RETURNS INT
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    DECLARE OUID INT DEFAULT 0;
    DECLARE NEW_ID INT DEFAULT 0;
    -- Ermitteln der Eigentümer-OU
    SELECT OU INTO OUID FROM mt_org.EMPLOYEE WHERE ID = IN_CALLER_EID;
    -- Speichern der Daten
    INSERT INTO mt_data.CLIENT (ID, NAME) VALUES (IN_ID, IN_NAME);
    -- Speichern des erstellten PK in "NEW_ID"
    IF IN_ID IS NULL THEN
        SET NEW_ID = LAST_INSERT_ID();
    ELSE
        SET NEW_ID = IN_ID;
    END IF;
    -- Speichern der Verknüpfung OU <--> Datensatz
    INSERT INTO mt_ass.A_CLIENT VALUES (NEW_ID, OUID);
    RETURN NEW_ID;
END; //

-- UPDATE_1 - Ersatz für fehlenden Instead-Of Trigger auf "CLIENT"
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
PROCEDURE CLIENT_UPDATE_NAME (IN_CALLER_EID INT, IN_ID INT, NEW_NAME VARCHAR(128))
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    -- Ändern der Daten
    UPDATE mt_data.CLIENT
    SET NAME = NEW_NAME
    WHERE ID IN (
        SELECT ID FROM mt_ass.CLIENT WHERE ID = IN_ID AND EID = IN_CALLER_EID
    );
END; //

-- UPDATE_2 - Ersatz für fehlenden Instead-Of Trigger auf "CLIENT"
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
PROCEDURE CLIENT_UPDATE_EID (IN_CALLER_EID INT, IN_ID INT, IN_NEW_EID INT)
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    DECLARE NEW_OUID INT DEFAULT 0;
    -- Ermitteln der Eigentümer-OU
    SELECT OU INTO NEW_OUID FROM mt_org.EMPLOYEE WHERE ID = IN_NEW_EID;
    -- Ändern der Eigentümer-Assoziation
    UPDATE mt_ass.A_CLIENT
    SET OUID = NEW_OUID
    WHERE ID IN (
        SELECT ID FROM mt_ass.CLIENT WHERE ID = IN_ID AND EID = IN_CALLER_EID
    );
END; //
```

```
-- DELETE - Ersatz für fehlenden Instead-Of Trigger auf "CLIENT"
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
PROCEDURE CLIENT_DELETE (IN_CALLER_EID INT, IN_ID INT)
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    DECLARE ALLOWED INT DEFAULT 0;
    -- Prüfe, ob der Datensatz gesehen werden kann
    SELECT COUNT(*) INTO ALLOWED FROM mt_ass.CLIENT
    WHERE ID = IN_ID AND EID = IN_CALLER_EID;
    -- Datensatz löschen, wenn Aufruf zulässig
    IF ALLOWED > 0 THEN
        DELETE FROM mt_ass.A_CLIENT WHERE ID = IN_ID;
        DELETE FROM mt_data.CLIENT  WHERE ID = IN_ID;
    END IF;
END; //
DELIMITER ;

-- ==========================================================================================

-- INSERT - Ersatz für fehlenden Instead-Of Trigger auf "ORDER"
DELIMITER //
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
FUNCTION ORDER_INSERT (IN_CALLER_EID INT, IN_ID INT, IN_CID INT, IN_DT TIMESTAMP)
    RETURNS INT
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    DECLARE OUID INT DEFAULT 0;
    DECLARE NEW_ID INT DEFAULT 0;
    -- Ermitteln der Eigentümer-OU
    SELECT OU INTO OUID FROM mt_org.EMPLOYEE WHERE ID = IN_CALLER_EID;
    -- Speichern der Daten
    INSERT INTO mt_data.`order` (ID, CID, ORDER_DATETIME)
    VALUES (IN_ID, IN_CID, IN_DT);
    -- Speichern des erstellten PK in "NEW_ID"
    IF IN_ID IS NULL THEN
        SET NEW_ID = LAST_INSERT_ID();
    ELSE
        SET NEW_ID = IN_ID;
    END IF;
    -- Speichern der Verknüpfung OU <--> Datensatz
    INSERT INTO mt_ass.A_ORDER VALUES (NEW_ID, OUID);
    -- Ausgabe des neuen PK
    RETURN NEW_ID;
END; //

-- UPDATE_1 - Ersatz für fehlenden Instead-Of Trigger auf "ORDER"
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
PROCEDURE ORDER_UPDATE_DATETIME (IN_CALLER_EID INT, IN_ID INT, NEW_DT TIMESTAMP)
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    -- Ändern der Daten
    UPDATE mt_data.`order`
    SET ORDER_DATETIME = NEW_DT
    WHERE ID IN (
        SELECT ID FROM mt_ass.`order` WHERE ID = IN_ID AND EID = IN_CALLER_EID
    );
END; //
```

```
-- UPDATE_2 - Ersatz für fehlenden Instead-Of Trigger auf "ORDER"
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
PROCEDURE ORDER_UPDATE_CID (IN_CALLER_EID INT, IN_ID INT, NEW_CID INT)
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    -- Ändern der Daten
    UPDATE mt_data.`order`
    SET CID = NEW_CID
    WHERE ID IN (
        SELECT ID FROM mt_ass.`order` WHERE ID = IN_ID AND EID = IN_CALLER_EID
    );
END; //

-- UPDATE_3 - Ersatz für fehlenden Instead-Of Trigger auf "ORDER"
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
PROCEDURE ORDER_UPDATE_EID (IN_CALLER_EID INT, IN_ID INT, IN_NEW_EID INT)
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    DECLARE NEW_OUID INT DEFAULT 0;
    -- Ermitteln der Eigentümer-OU
    SELECT OU INTO NEW_OUID FROM mt_org.EMPLOYEE WHERE ID = IN_NEW_EID;
    -- Ändern der Eigentümer-Assoziation
    UPDATE mt_ass.A_ORDER
    SET OUID = NEW_OUID
    WHERE ID IN (
        SELECT ID FROM mt_ass.`order`
        WHERE ID = IN_ID AND EID = IN_CALLER_EID
    );
END; //

-- DELETE - Ersatz für fehlenden Instead-Of Trigger auf "ORDER"
CREATE OR REPLACE
    DEFINER='mt_ass_admin'@'localhost'
PROCEDURE ORDER_DELETE (IN_CALLER_EID INT, IN_ID INT)
    MODIFIES SQL DATA
    SQL SECURITY DEFINER
BEGIN
    DECLARE ALLOWED INT DEFAULT 0;
    -- Prüfe, ob der Datensatz gesehen werden kann
    SELECT COUNT(*) INTO ALLOWED FROM mt_ass.`order`
    WHERE ID = IN_ID AND EID = IN_CALLER_EID;
    -- Datensatz löschen, wenn Aufruf zulässig
    IF ALLOWED > 0 THEN
        DELETE FROM mt_ass.A_ORDER WHERE ID = IN_ID;
        DELETE FROM mt_data.`order`  WHERE ID = IN_ID;
    END IF;
END; //
DELIMITER ;

-- ==========================================================================================

GRANT EXECUTE                        ON `mt_ass`.*              TO `mt_ass_user`@`localhost`;
GRANT SELECT                         ON `mt_ass`.`client`       TO `mt_ass_user`@`localhost`;
GRANT SELECT                         ON `mt_ass`.`order`        TO `mt_ass_user`@`localhost`;
GRANT SELECT, INSERT, UPDATE, DELETE ON `mt_ass`.`orderproduct` TO `mt_ass_user`@`localhost`;
GRANT SELECT, INSERT, UPDATE, DELETE ON `mt_ass`.`product`      TO `mt_ass_user`@`localhost`;
```

Verzeichnis der Abkürzungen

ACID	Atomicity, Consistency, Isolation, Durability
APEX	Application Express
ABAC	Attribute-Based Access Control
ACL	Access Control List
AD	Active Directory, Verzeichnisdienst von Microsoft
Ass	Abkürzung für: Association / Assoziation
BaFin	Bundesanstalt für Finanzdienstleistungsaufsicht, staatliche Institution
BSI	Bundesamt für Sicherheit in der Informationstechnik, staatliche Institution
CPU	Central Processing Unit
CTE	Common Table Expressions
DAC	Discretionary Access Control
DBMS	Database Management System
DSGVO	Kurzform von EU-DSGVO
EID	Employee-Identifier, Identifikationsnummer eines Mitarbeiters
EMP	Employee, Mitarbeiter
EU-DSGVO	Europäische Datenschutz-Grundverordnung
FD	Fachlicher Datensatz
FDBM	Fachliches Datenbankmodell
FGAC	Fine-Grained Access Control
FK	Abhängig vom Kontext: Führungskraft oder Foreign Key, Fremdschlüssel
I/O	Input/Output
IBAC	Identity-Based Access Control
ID	Identifikator / Identifikationsnummer, eindeutige Kennzeichnung eines Datensatzes
JVM	Java Virtual Machine, Laufzeitumgebung für Java-Anwendungen
KWG	Kreditwesengesetz
LBAC	Lattice-Based Access Control
LDAP	Lightweight Directory Access Protocol
MA	Mitarbeiter

MAC	Mandatory Access Control
MaRisk	Mindestanforderungen an das Risikomanagement, Verwaltungsanweisung der BaFin
MLS	Multi-Level-Security
NoSQL	Not only SQL / No SQL
OE	Organisationseinheit
OLS	Oracle Label Security
Org	Organisation
OU	Organizational Unit, siehe OE
OUID	Organizational Unit Identifier, Identifikationsnummer einer OE
PK	Primary Key, Primärschlüssel einer Tabelle
PL/SQL	Procedural Language / Structured Query Language
RAC	Real Application Clusters
RAS	Real Application Security
RBAC	Role-/Rule-Based Access Control
RLS	Row-Level Security
SELinux	Security-Enhanced Linux
SQL	Structured Query Language
T-SQL	Transact-SQL
VPD	Virtual Private Database
XML	Extensible Markup Language

Thesen

Hierarchien: Eine relationale Datenbank unterstützt, unter Verwendung von rekursiven CTEs, den Aufbau und die Nutzung von hierarchischen Strukturen.

Vermeidung von Zyklen: Der Vermeidung von Zyklen innerhalb hierarchischer Strukturen kommt eine besondere Rolle zu, da es bei einer Entstehung eines Zyklus zu einem Ausfall des Modells kommen kann.

INSTEAD-OF Trigger: Die Nutzung von *INSTEAD-OF* Triggern für zuvor definierte komplexe Views ermöglichen die Verlagerung der Logik zur Manipulation von Datensätzen und dadurch die Einhaltung mehrschichtiger Konsistenzbedingungen.

MariaDB: Das Datenbanksystem MariaDB 10.4 eignet sich aufgrund der fehlenden Unterstützung von *INSTEAD-OF* Triggern nicht für die Filterung von Datensätzen durch komplexe Views.

Schnittstellen: Die Entwicklung einer Zugriffsschicht innerhalb einer relationalen Datenbank ohne Veränderung der bereitgestellten Schnittstellen ist nicht möglich.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbstständig, ohne unerlaubte fremde Hilfe und nur unter Verwendung der in der Arbeit aufgeführten Hilfsmittel angefertigt habe.

_____ _____

Ort, Datum Unterschrift