

Datenbankadministration

Microsoft SQL Server 2017

Florian Weidinger

1. Ausgabe: 05.03.2020
Letzte Änderung: 31. März 2020

Inhaltsverzeichnis

Teil I

Grundlagen des Datenbankdesigns

1 Entwicklung eines Datenmodells

Inhaltsangabe

In diesem Abschnitt erfolgt die Vermittlung der Kenntnisse darüber, wie man aus einer großen Menge von Informationen und Anforderungen an eine neue Datenbank eine Datenstruktur entwirft. Im Zuge der Datenmodellierung soll ein exaktes und vollständiges Modell des betrachteten Realitätsausschnitts erarbeitet werden, welches den Rahmen für die Entwicklung neuer oder die Erweiterung bestehender Anwendungssysteme bildet.

Als Werkzeug für die Erstellung konzeptioneller Datenmodelle wird die Entity-Relationship Modellierung (ER-Modellierung), zuerst in ihrer einfachsten Art und später in einer erweiterten Fassung, verwendet. Es werden dabei alle vier Phasen des Datenmodellierungsprozesses anhand eines durchgängigen Beispiels beschrieben.

Die ER-Modellierung stellt eine spezielle „Information-Engineering-Technik“ dar, die zur Erstellung von Datenmodellen hoher Qualität benutzt wird. Entworfen in den 70er Jahren von Peter Pin-Shan Chen wurde sie seither vielfach erweitert und verbessert.

Ziel eines konzeptionellen Datenmodells ist es, die typmäßige Struktur der Daten in der Datenbank ohne deren Inhalt zu beschreiben. Als Beispiel aus der realen Welt wäre die Ausstattung eines Büros mit Möbeln zu nennen. Wer später das Büro nutzt und mit welchem Inhalt die Schränke gefüllt werden, ist für die Erstellung des Modells Bedeutungslos.

Die Entwicklung eines Datenmodells teilt sich in die folgenden vier Phasen ein:

1. Klassifizierung von Objekten
2. Festlegung relevanter Eigenschaften
3. Bestimmung identifizierender Eigenschaften
4. Beschreibung sachlogischer Zusammenhänge zwischen den einzelnen Objekten

1.1 Die Modellierungsinformationen

Die Struktur der Bundeswehr soll in einem ER-Modell dargestellt werden. Es wird jedoch nur ein Ausschnitt aus der Realität betrachtet, um die Übersichtlichkeit der Datenstrukturen zu gewährleisten. Im Folgenden werden die dafür notwendigen Objekte festgelegt.

1. Die Bundeswehr besitzt zahlreiche Dienststellen an den unterschiedlichsten Standorten, welche sich untereinander über- oder untergeordnet sind. Jede Dienststelle besteht aus Dienstposten. Diese

wiederum werden mit Soldaten besetzt. Jeder Soldat empfängt bei Einstieg in die Bundeswehr seine persönliche Ausrüstung und besitzt diese dann für die Dauer seines Dienstverhältnisses.

2. Jeder Dienststelle der Bundeswehr ist eine Dienststellennummer zugeordnet, über die diese identifiziert werden kann. Des Weiteren hat jede Dienststelle eine bestimmte Größe sowie Bezeichnung, wie z. B. Führungsunterstützungsschule der Bundeswehr.
3. Für die Standorte, an denen sich die Dienststellen befinden, müssen in der Datenbank Postleitzahl (PLZ), Ort, Straße und die Hausnummer hinterlegt sein. Hierbei ist zu beachten, dass sich eine Dienststelle auch an mehreren Standorten befinden kann.
4. Die den Dienststellen untergeordneten Dienstposten werden durch ihre Dienstposten_ID und ein Beginn- und Enddatum charakterisiert. Als eine weitere Eigenschaft soll eine kurze Dienstpostenbeschreibung hinterlegt werden.
5. Jeder in der Datenbank aufgeführte Soldat soll dort mit seiner Personanlnummer, einer Personen-kennziffer, sowie Vor- und Nachnamen und Dienstgrad aufgeführt werden. Weiterhin kann der Anwender auch die aktuelle Adresse, bestehend aus PLZ, Wohnort, Straße und Hausnummer, aus der Datenbank heraussuchen.
6. Das letzte Objekt in der Datenbank stellt die persönliche Ausrüstung des Soldaten dar. Diese besitzt eine Bezeichnung, ist aus einem bestimmten Material gefertigt und hat eine Farbe. Für eine bessere Zuordnung ist jeder Aurüstungsgegenstand mit einer Versorgungsnummer versehen.

1.2 Klassifizierung von Objekten

Das im vorigen Abschnitt vorgestellte Beispiel stellt nur einen kleinen Ausschnitt aus der Realität dar. Komplexer gestaltete Datenbanken können aus weit mehr Objekten bestehen. Um dabei nicht den Überblick über diese Flut von Objekten zu verlieren, werden diese in Klassen gruppiert. Diese Objektklassen enthalten dann die Objekte, die von ihrer Art her gleich sind und über die die gleichen Informationen gesammelt werden. Damit wird eine Abstraktionsebene gebildet, die es ermöglicht, von den Besonderheiten der einzelnen Objekte abzusehen und nur das typische der gebildeten Objektklassen zu berücksichtigen.

1.2.1 Definitionen und Syntaxregeln

Für die Darstellung der Objekttypen gelten folgende syntaktische Regeln:

1. Ein Objekttyp wird durch ein Rechteck dargestellt, in dessen Mitte der Objekttypname eingetragen wird.



2. Die Größe und Position des Rechtecks sind bedeutungslos.
3. Der Objekttypname steht im Singular und muss für das gesamte Datenmodell eindeutig sein.



- **Objekt:** Ein Objekt (engl. Entity) ist ein Exemplar von Personen, Gegenständen oder nicht-materiellen Dingen, über das Informationen gespeichert wird, z. B. der konkrete Soldat Max Mustermann.
- **Objekttyp:** Ein Objekttyp (engl. Entity type) ist eine durch einen Objekttyp-namen eindeutig benannte Klasse von Objekten, über die dieselben Informationen gespeichert und die prinzipiell in gleicher Weise verarbeitet werden, wie z. B. die benannte Klasse bzw. der Objekttyp SOLDAT.

Die Bildung von Objekttypen hängt entscheidend von den Anforderungen des jeweils zu modellierenden Gegenstandsbereiches ab. Aus der Sicht eines Großhändlers kann ein Unternehmen mit all seinen Bereichen als ein einziger Objekttyp gesehen werden. Dasselbe Unternehmen dagegen wird aus seiner eigenen Sicht detailliert mit seinen Bereichen, Abteilungen, Mitarbeitern, Werkshallen, Fahrzeugen usw. zu modellieren sein. Diese Modellierung ist ebenfalls nicht eine einmalige, in sich abgeschlossene Tätigkeit, denn im Laufe der Zeit müssen Änderungen der Realität auch im Modell eingearbeitet werden.

1.2.2 Traditionelles Pendant

Die Informationsverarbeitung mit Hilfe elektronischer Datenverarbeitung hat hinsichtlich der Datenspeicherung nur wenig prinzipiell neue Methoden entwickelt. Fast alle Konzepte, in Bezug auf das Entity-Relationship-Modell, haben ihr Pendant in der traditionellen Informationsspeicherung. Zum besseren Verständnis der eingeführten Begriffe wird auf diese Zusammenhänge an den entsprechenden Stellen hingewiesen.

So entsprechen die Objekttypen den traditionellen Karteikästen und die Objekte eines Objekttyps den Karteikarten, die in einem Karteikasten eingeordnet sind. In der traditionellen Arbeitsweise würde für „Axel Schweiss“ eine Karteikarte angelegt werden und z. B. im Karteikasten „Soldat“ abgelegt werden.

1.3 Festlegung der relevanten Eigenschaften

Im ersten Schritt, der Klassifizierung von Objekten, wurden Objekte, die in gleicher Art und Weise verarbeitet werden, in Objekttypen zusammengefasst. Um die Verarbeitung dieser Objekte automatisiert durchführen zu können, ist es Voraussetzung, dass jeder Objekttyp bestimmte Angaben speichert. Diese Angaben werden für die elektronische Verarbeitung entweder als Eingabeinformation oder als Ausgabeinformation benötigt.

Aus diesem Grund ist es notwendig, für jeden Objekttyp die relevanten Eigenschaften der Objekte, die in ihm zusammengefasst werden, anzugeben. Damit wird der durch den Objekttyp definierte Begriff auf einen Satz relevanter Eigenschaften reduziert. Die Festlegung dieser Eigenschaften ist aber nur bei genauer Kenntnis der Geschäfts- und Verarbeitungsprozesse des Auftraggebers möglich. Ohne diese Kenntnisse befindet man sich in jedem Falle im Bereich von Spekulationen.

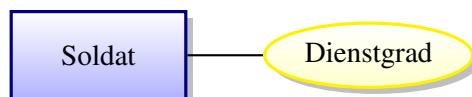
1.3.1 Definitionen und Syntaxregeln



- **Eigenschaft:** Eine Eigenschaft (engl. Attribute) ist die Benennung für ein relevantes Merkmal aller Objekte, die in einem Objekttyp zusammengefaßt werden, z. B. Soldaten haben die Eigenschaft Dienstgrad.
- **Eigenschaftswert:** Ein Eigenschaftswert (engl. Attribute value) ist eine spezielle Ausprägung, die eine Eigenschaft für ein konkretes Objekt annimmt, z. B. der Dienstgrad Hauptfeldwebel.

Für die Darstellung der Eigenschaften gelten folgende Regeln:

1. Die Benennung der Eigenschaft wird als Blase an den Objekttyp angehängt.



2. Die Reihenfolge der Eigenschaften ist bedeutungslos.
3. Die Benennung der Eigenschaft steht im Singular und muss für den Objekttyp eindeutig sein.

Auch an dieser Stelle wird eine Abstraktion der realen Welt vorgenommen, in dem statt der Eigenschaftswerte, die jedes einzelne Objekt besitzt, nur die Eigenschaften der Objekttypen angegeben werden.

Um diesen Abstraktionsprozess korrekt durchführen zu können, bedarf es der Einhaltung einiger Regeln:

- Es darf niemals ein Eigenschaftswert als Eigenschaftsname verwendet werden (beispielsweise anstatt 02.05.1985 die Bezeichnung „Geburtsdatum“ oder statt „männlich“ die Bezeichnung „Geschlecht“).
- Die Bezeichnung eines Objekttyps sollte niemals im Eigenschaftsnamen auftauchen, da dieser immer nur im Kontext des Objekttyps gilt (z. B. Person mit der Eigenschaft Name, nicht „Personname“, sondern die Eigenschaft „Name“ wählen).
- Bei komplexen Eigenschaften wie z. B. einer Adresse oder einer PK stellt sich immer wieder die Frage, ob diese zerlegt werden müssen oder ob sie als atomar¹ betrachtet werden. Die Antwort auf diese Frage ist abhängig von den einzelnen Verarbeitungsprozessen, denen diese Daten unterliegen. Muss beispielsweise die Information verfügbar sein, von welchem Kreiswehrersatzamt ein Soldat betreut wird, so muss die PK in ihre Bestandteile zerlegt werden. Ist diese Information irrelevant, kann die PK im ganzen als atomar betrachtet werden.
- Problematisch ist auch die Entscheidung, ob speicherwürdige Informationen als Eigenschaften oder als ein eigenständiger Objekttyp betrachtet werden sollen. Um einen eigenständigen Objekttyp handelt es sich immer dann, wenn er für das Unternehmen bedeutsame Objekte enthält, die relevante individuelle Eigenschaften besitzen.

1.3.2 Traditionelle Datenspeicherung

Die Eigenschaften eines Objekttyps A entsprechen den Feldern, die auf einer Karteikarte zur Aufnahme der relevanten Informationen angelegt werden. Durch die gewählten Eigenschaften wird also die einheitliche Struktur aller Karteikarten eines Kastens festgelegt.

Welche Eigenschaften können Sie, bezogen auf das Beispiel „Bundeswehr“, den identifizierten Objekttypen zuordnen?

Lösungsvorschlag

- Objekttyp „Dienststelle“: Dienststellennummer, Bezeichnung
- Objekttyp „Standort“: PLZ, Ort, Straße, Hausnummer

¹atomare Information = eine einzige, nicht mehr teilbare Information

- Objekttyp „Dienstposten“: Dienstposten_ID, Beginndatum, Enddatum, Dienstpostenbeschreibung
- Objekttyp „Soldat“: Personalnummer, PK, Name, Vorname, Dienstgrad, PLZ, Ort, Straße, Hausnummer
- Objekttyp „Ausrüstung“: Versorgungsnummer, Material, Farbe

1.4 Festlegung der Identifizierung

Ein Objekttyp stellt die Zusammenfassung mehrerer gleichartiger Objekte dar. Gleichartig bedeutet, dass diese Objekte die gleichen Eigenschaften aufweisen und die Verarbeitungsprozesse für all diese Objekte gleich sind. Die einzelnen Objekte eines Objekttyps müssen aber voneinander unterscheidbar sein. Deshalb muss festgelegt werden, auf welche Weise ein Objekt innerhalb des Objekttyps identifiziert werden kann.

1.4.1 Identifizierungsvarianten

Für die Identifizierung eines Objekts innerhalb eines Objekttyps stehen die konkreten Eigenschaftswerte des Objekts zur Verfügung.

Es werden drei Varianten zur Identifizierung von Objekten unterschieden.

Identifizierung eines Objekts durch eine einzelne Eigenschaft

Es kann vorkommen, dass die Eigenschaftswerte einer Eigenschaft eines Objekttyps eindeutig ist. D. h. jeder Eigenschaftswert dieser Eigenschaft ist so geartet, dass jedes Objekt dieses Objekttyps einen unterschiedlichen Wert für diese Eigenschaft hat. Durch Angabe dieses Eigenschaftswertes ist dann das Objekt eindeutig identifiziert.

Beispiel: Soldaten werden i. d. R. durch ihre PK eindeutig identifiziert

Identifizierung eines Objekts durch eine Kombination mehrerer Eigenschaften

In einigen Fällen ist keine der Eigenschaften eines Objekttyps geeignet, alleine als identifizierende Eigenschaft zu fungieren. Um dieses Problem zu lösen kann versucht werden, eine minimale Kombination von Eigenschaften eines Objekttyps als Identifikationsmerkmal zu benutzen. Dies funktioniert dann, wenn die Kombination der Werte der entsprechenden Eigenschaften bei jedem Objekt eindeutig sind.

Beispiel: Die Kombination der beiden Attribute PLZ und Ortsbezeichnung ist eindeutig, eine Eigenschaft alleine nicht.

Identifizierung eines Objekts durch eine organisatorische Eigenschaft

Sollte es dennoch vorkommen, dass weder eine einzelne Eigenschaft, noch eine Kombination von Eigenschaften zur Identifikation der Objekte eines Objekttyps geeignet ist, kann eine künstliche Eigenschaft eingeführt werden, bei der die Eindeutigkeit durch organisatorische Maßnahmen gewährleistet wird.

Ein Beispiel hierfür wäre eine Ort_ID, die es möglich macht, einen Ort eindeutig zu bestimmen ohne die Kombination aus PLZ und Ortsnamen heranziehen zu müssen.

Es ist aber auch möglich, dass eine organisatorische Eigenschaft gewählt wird, da abzusehen ist, dass eine identifizierende Eigenschaft durch eine andere ersetzt werden soll. Die Umstrukturierung der Datenbank wäre zu aufwendig und zu komplex.

Ein Beispiel aus der Praxis ist die PK und die Personalnummer eines jeden Soldaten. Die Personalnummer wird in geraumer Zeit die PK ersetzen. Es ist aus diesem Grund von Vorteil eine organisatorische Eigenschaft, wie die Personen_ID zu wählen, um eine Umstrukturierung zu vermeiden.

Weitere Beispiele für organisatorische Eigenschaften sind die beiden Attribute Artikelnummer und Lfd-Nr.

Die Identifizierung der Objekte eines Objekttyps mittels einer „organisatorischen Eigenschaft“ ist bei der automatisierten Datenverarbeitung eine beliebte Vorgehensweise. Sie wird häufig selbst dann angewendet, wenn natürliche identifizierende Eigenschaften vorhanden sind. Meist handelt es sich um eine laufende Nummer.

Dies hat den Vorteil, dass kurze identifizierende Eigenschaftswerte entstehen. Der Nachteil ist, dass Werte wie z. B. eine laufende Nummer meist keinerlei Aussagekraft haben und somit Gefahren wie Verwechslung oder Fehleingabe entstehen.

Die Festlegung der Identifizierungsform für einen Objekttyp muss mit großer Sorgfalt erfolgen. Relationale Datenbank-Managementsysteme für die wir unsere Modellierung durchführen, lassen es nämlich nicht zu, dass zwei Objekte desselben Objekttyps in der Kombination ihrer identifizierenden Merkmale übereinstimmen. Bleibt ein Restrisiko hinsichtlich der Unikalität der identifizierenden Merkmale und treten dann bei der praktischen Datenbankarbeit tatsächlich zwei Objekte mit übereinstimmenden Werten ihrer identifizierenden Merkmale auf, lehnt das Datenbank-Managementsystem die Speicherung des zweiten Objekts ab.

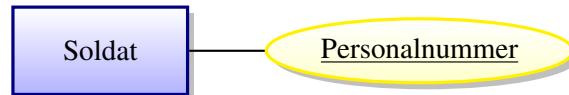


Um ein Modell übersichtlich und verständlich zu halten, sollte konsequent immer nur eine der drei zur Verfügung stehenden Methoden für die Identifizierung von Objekten genutzt werden!

1.4.2 Markierung der Identifizierenden Eigenschaft

Für die Markierung der (teil)identifizierenden Eigenschaft gelten die folgenden syntaktischen Regeln (diese werden u. U. in Kombination mit anderen angewendet):

1. Eine identifizierende Eigenschaft (bzw. jede teilidentifizierende Eigenschaft) wird durch Unterstreichung kenntlich gemacht.



2. Die Position der (teil)identifizierenden Eigenschaft innerhalb der Liste der Eigenschaften ist bedeutungslos. Sie sollte jedoch aus Gründen der Übersichtlichkeit immer zu erst genannt werden.

1.4.3 Klassisches Pendant

Bei der traditionellen Informationsspeicherung mit Hilfe von Karteikästen entspricht der Identifizierung die Festlegung eines Kennbegriffs: Üblicherweise wird im Kopf einer Karteikarte für das betreffende Objekt ein Wert angegeben, der das Objekt innerhalb des Karteikastens identifiziert². Um eine bestimmte Karteikarte manuell schneller finden zu können, werden die Karteikarten des Karteikastens nach diesem Begriff sortiert.

Betrachten wir wieder das Beispiel „Bundeswehr“. Welche Objekteigenschaften können Ihrer Meinung nach als identifizierende Merkmale verwendet werden?

Lösungsvorschlag

- Objekttyp „Dienststelle“: Dienststellennummer
- Objekttyp „Standort“: PLZ, Ort
- Objekttyp „Dienstposten“: Dienstposten_ID
- Objekttyp „Soldat“: Personalnummer
- Objekttyp „Ausrüstung“: Versorgungsnummer

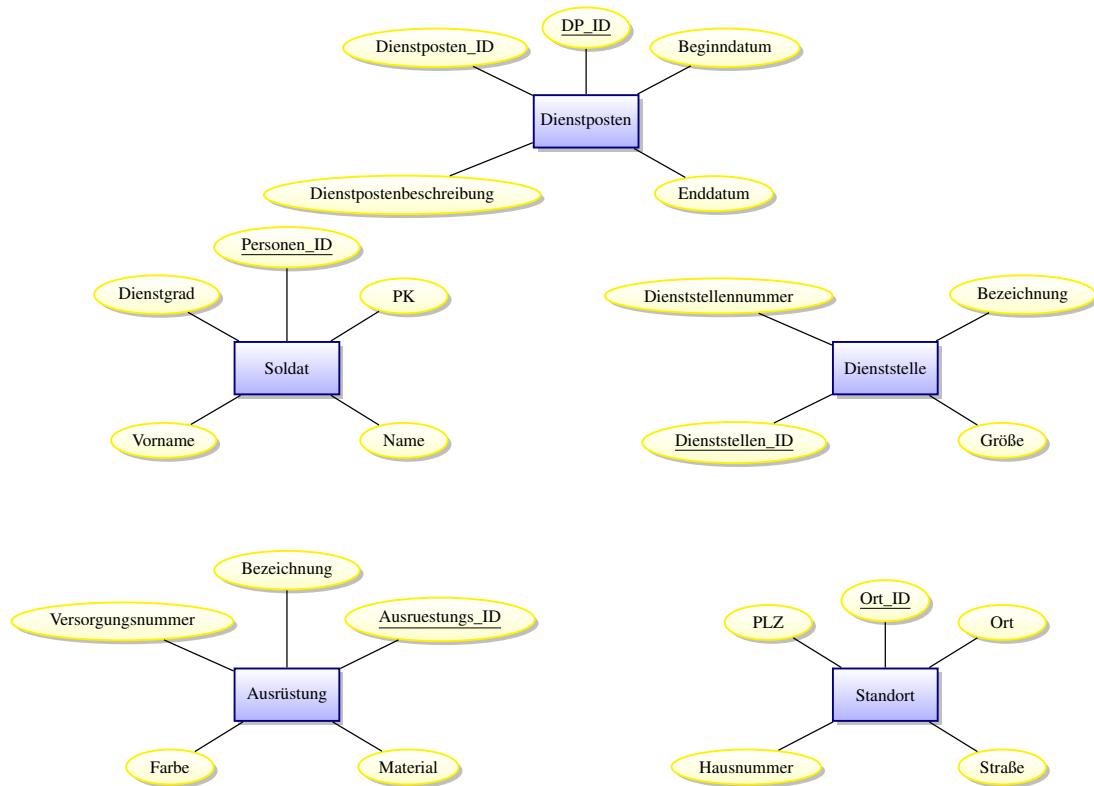
An dieser Stelle wird die Entscheidung getroffen, dass für die Fortführung dieses Modells organisatorische Einheiten als identifizierende Eigenschaften eingeführt werden!

- Objekttyp „Dienststelle“: Dienststellen_ID

²Kennbegriff wird oft auch als „Reiter“ bezeichnet

- Objekttyp „Standort“: Ort_ID
- Objekttyp „Dienstposten“: DP_ID
- Objekttyp „Soldat“: Personen_ID
- Objekttyp „Ausrüstung“: Ausruestungs_ID

1.5 Modellierungsformen



Hinweis: Das Objekt Soldat enthält nicht alle Attribute. Es fehlen die Attribute Personalnummer, Plz, Ort, Straße und Hausnummer.

2 Beschreibung sachlogischer Zusammenhänge zwischen Objekttypen

Inhaltsangabe

Bisher wurden im Rahmen der Datenmodellierung die speicherwürdigen Objekte mit ihren relevanten Eigenschaften lediglich isoliert beschrieben. In der Praxis stehen die interessanten Objekte jedoch in vielfältiger Weise miteinander in Zusammenhang: Dienststellen befinden sich an Orten, Soldaten arbeiten in Dienststellen usw. Auch Objekte desselben Typs können miteinander in Zusammenhang stehen: Lehrer leiten Lehrer an, Schulen haben Partnerschaften mit anderen Schulen usw. Diese Zusammenhänge müssen ebenfalls im Datenmodell dargestellt werden, denn sie bringen wesentliche Aspekte der betrachteten Realität zum Ausdruck.

Die sachlogischen Zusammenhänge zwischen den Objekten werden in drei Gruppen von Beziehungstypen unterteilt:

- **Binäre Beziehungstypen:** Beschreiben den Zusammenhang zwischen jeweils zwei Objekten, die verschiedenen Objekttypen angehören.
- **Beziehungstypen n-ten Grades:** Sie Beschreiben den Zusammenhang zwischen mehr als zwei Objekten, die verschiedenen Objekttypen angehören.
- **Rekursiv-Beziehungstypen:** Objekte, die aus demselben Objekttyp stammen, stehen in Zusammenhang.

Im Folgenden werden nur die binären oder auch dualen Beziehungstypen erläutert. Auf Beziehungstypen höheren Grades wird im weiteren Verlauf nicht eingegangen, da diese in der Praxis kaum Relevanz besitzen. Die Behandlung der Rekursiven Beziehungstypen erfolgt im Kapitel 3.

Für den weiteren Verlauf sind die nachfolgenden zwei Definitionen zu unterscheiden:

- **Beziehung (engl. Relationship):** Kennzeichnet den konkreten Zusammenhang zwischen zwei realen Objekten. Beispiel: „Max Mustermann“ ist Lehrgangsteilnehmer im Lehrgang „Datenbank Administrator“
- **Beziehungstypen:** Beschreibt den typmäßigen Zusammenhang, der zwischen den Objekttypen besteht. Beispiel: Objekttypen sind „Soldat“, „Lehrgang“ mit dem Beziehungstyp „ist Lehrgangsteilnehmer in“

Während in der realen Welt der Zusammenhang zwischen zwei konkreten Objekten **beobachtet** wird, **beschreibt** man in der Modellwelt das verallgemeinerte Wechselspiel zwischen zwei Objekttypen. Im mathematischen Sinne ist ein Beziehungstyp zwischen den Objekten A und B die Menge der Beziehungen zwischen jeweils einem Objekt aus dem Objekttyp A und einem Objekt aus dem Objekttyp B. Die für den Beziehungstyp formulierten Angaben müssen somit für alle konkreten Beziehungen zwischen den betrachteten Objekttypen gültig sein.

2.1 Benennung, Optionalität und Kardinalität

Der sachlogische Zusammenhang zwischen den Objekten zweier Objekttypen, der durch einen Beziehungstyp beschrieben wird, besteht immer in **beiden Richtungen**: Soldaten stehen beispielsweise in einem Zusammenhang mit Dienststellen (sie arbeiten dort) und Dienststellen stehen im Zusammenhang mit Soldaten (sie beschäftigen sie). Jede der beiden Beziehungstyp-Richtungen wird durch 3 Angaben näher bestimmt.

2.1.1 Benennung

Betrachtet man im Beispiel aus Kapitel 1.1 den Zusammenhang zwischen Soldat und Ausrüstung, könnte man sich für folgende Dinge interessieren:

- Ein Soldat besitzt Ausrüstung.
- Ein Soldat empfängt seine Ausrüstung.
- Ein Soldat hat bestimmte Ausrüstungsgegenstände mitzuführen.

Welcher Zusammenhang gespeichert werden soll, wird durch die Benennung zum Ausdruck gebracht. Hier ist es „besitzt“.

2.1.2 Optionalität

Die Optionalität klärt die Frage, ob jedes Objekt des Objekttyps A mit mindestens einem Objekt des Objekttyp B in Beziehung stehen muss? Je nach Antwort unterscheidet man zwei Fälle:

- **Ja:** Die Beziehungstyp-Richtung wird als nichtoptional, also als **obligatorisch¹** bezeichnet.
- **Nein:** Die Beziehungstyp-Richtung wird als optional, also **kann vorhanden sein** bezeichnet.

¹obligare lat. = bindend, verpflichtend

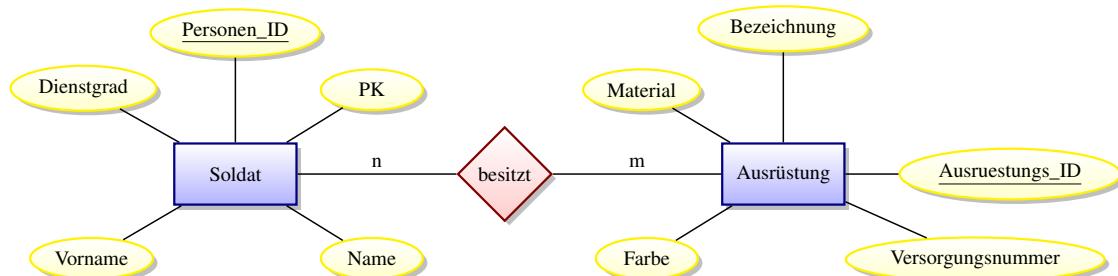
2.1.3 Kardinalität

Für die Angabe der Kardinalität einer Beziehungstyp-Richtung vom Objekttyp A zum Objekttyp B stellt man sich folgende Frage: „Kann ein Objekt des Objekttyps A mit mehreren Objekten des Objekttyps B in Beziehung stehen?“ Es können dabei zwei unterschiedliche Fälle auftreten:

- **Ja:** Der Beziehungstyp-Richtung wird die Kardinalität **N** zugeordnet. Dabei steht **N** für eine beliebige Zahl größer oder gleich 0.
- **Nein:** Die Beziehungstyp-Richtung wird die Kardinalität **1** zugeordnet, denn es gibt **höchstens ein** Objekt des Objekttyps B zu dem Objekt aus Objekttyp A.

2.1.4 Darstellung im Modell

Es gibt verschiedene Formen der Darstellung. Man kann jede Beziehungstyp-Richtung benennen, jedoch ist die Benennung meist nur für eine Richtung passend. Die Gegenrichtung müsste dann bei Bedarf umformuliert werden. Im Folgenden soll ein Beispiel die Zusammenhänge zwischen Benennung, Optionalität und Kardinalität zeigen. Gegeben seien die Objekttypen Soldat und Ausrüstung mit der angezeigten Beziehung.



Zu diesem Ausschnitt aus einem ER-Modell ergeben sich die drei folgenden Fragen:

1. Welcher Zusammenhang soll ausgedrückt werden (**Benennung**)?

Durch die Benennung der beiden Objekttypen mit „Soldat“ und „Ausrüstung“ zeigt sich der Zusammenhang, dass ein Soldat Ausrüstung besitzt.

2. Sind die beiden Objekttypen aneinander gebunden (**Optionalität**)?

In diesem Beispiel ist es so, dass ein Soldat Ausrüstung besitzen kann aber nicht muss, z.B. vor der Einkleidung ist man schon Soldat, obwohl noch jegliche Ausrüstungsgegenstände fehlen. Anders herum ist es möglich, dass Ausrüstungsgegenstände einem Soldaten zugeordnet wurden oder der Ausrüstungsgegenstand noch im Lager liegt. Antwort: Beide Richtungen sind optional.

3. Wie stehen die beiden Objekttypen in Zusammenhang (**Kardinalität**)?

- Fragerichtung vom Objekttyp „Soldat“ zum Objekttyp „Ausrüstung“: „Kann ein Soldat mehrere Ausrüstungsgegenstände besitzen?“, Antwort: Ja, daher N.
- Fragerichtung vom Objekttyp „Ausrüstung“ zu „Soldat“:
„Kann ein Ausrüstungsgegenstand von mehreren Soldaten besessen werden?“ Antwort: Ja, d. h. die Kardinalität lautet M.

Ein Ausrüstungsgegenstand ist durch eine Versorgungsnummer gekennzeichnet. Somit ist es möglich, unterschiedliche Ausrüstungsgegenstände mit der gleichen Versorgungsnummer an die Soldaten auszugeben. Es entsteht eine „N:M“ Kardinalität (vgl. ??).

Bei der Festlegung der Kardinalität ist außer dem zu beachten, über welchen Zeitraum hinweg die Angaben zu Beziehungen in der Datenbank aufgenommen werden sollen. Bei der Beziehung „Soldat arbeitet in Dienststelle“, ist die Kardinalität auf 1 zu setzen, wenn der Soldat immer nur in einer Dienststelle arbeiten soll. Will man aber die Zuordnungsverhältnisse über einen längeren Zeitraum speichern, so ist die Kardinalität auf N festzulegen, weil es dann vorkommen kann, dass ein Soldat mit mehreren Dienststellen in Verbindung gebracht werden muss, also eine Historie gespeichert wird.

2.2 Notationen für Kardinalitäten

Für die Kardinalität gibt es verschiedene Notationen, also einheitliche Schreibweisen. In dieser Unterlage wird die Chen-Notation kurz vorgestellt und die (Min,Max)-Notation eingehender behandelt. Sofern die Chen-Notation von Interesse ist, kann diese in vielen Fachbüchern leicht nachgelesen werden.

2.2.1 Die Chen-Notation

In der Chen-Notation gibt es im Wesentlichen drei verschiedene Beziehungstypen, dabei ist es unerheblich, ob die verwendeten Buchstaben groß oder klein geschrieben werden. Die Werte geben die maximale Anzahl von beteiligten Objekten an.

Mögliche Varianten (binäre Beziehungen):

- 1:1
- 1:n

- n:m
- n:m:k (ternäre Beziehungen)

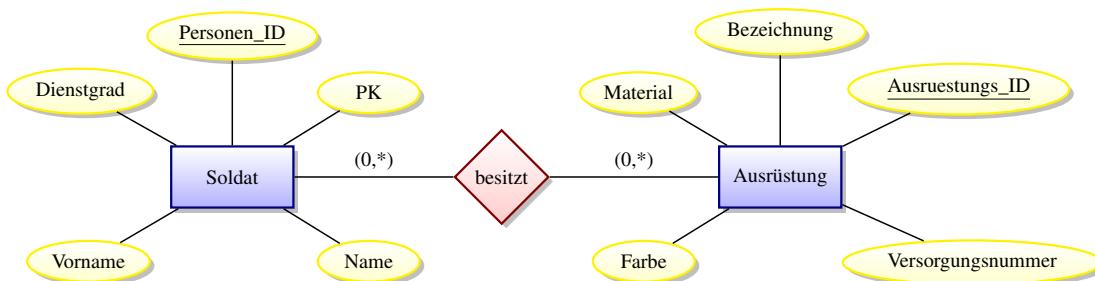
Welche Werte können angenommen werden bzw. wofür stehen die Buchstaben?

- n: beliebig viele (0,1,2...n)
- 1: höchstens ein (0 oder 1)
- m oder k: entsprichen der n-Definition

Die Angabe von genaueren Werten ist in der Chen-Notation nicht vorgesehen.

2.2.2 (Min,Max)-Notation

Die (Min,Max)-Notation ist im Wesentlichen eine Konkretisierung der Angaben in der Chen-Notation, denn es werden nicht nur die maximalen, sondern auch die minimalen Werte angegeben. Folgende Abbildung zeigt die oben verwendete Beziehung in der (Min,Max)-Notation.



Welche Werte können angenommen werden bzw. wofür stehen diese Werte? Bei der Min, Max-Notation gibt es eine Vielzahl von Möglichkeiten, diese hier aufzulisten, wäre schier unmöglich. Daher einige häufige Kombinationen.

Möglichkeit Bedeutung

- | | |
|-------|--|
| (1,1) | genau 1 \Rightarrow mindestens 1 und höchstens 1 |
| (1,*) | mindestens 1 \Rightarrow mindestens 1 und höchstens beliebig viele |
| (0,1) | höchstens 1 \Rightarrow mindestens 0 und höchstens 1 |
| (0,*) | kann haben \Rightarrow 0 oder beliebig viele |

- (2,100) mindestens 2 und höchstens 100
- (4,*) mindestens 4 und höchstens beliebig viele

Die beiden letzten Zeilen der obigen Auflistung sollen verdeutlichen, dass für die Min- und Max-Werte beliebige ganze Zahlen verwendet werden können. Hier ist jedoch zu beachten, dass die Umsetzung bestimmter Kombinationen in den Kardinalitäten in einem relationalen Datenbanksystem nicht mehr mit der referentiellen Integrität sichergestellt werden kann, sondern mit Elementen einer Programmiersprache auf Seiten der Anwendung oder der Datenbank. Später dazu mehr.

2.2.3 Schreib-/Leseweise der Kardinalitäten

Für die Syntax der Kardinalitäten gilt folgende Vorgehensweise.

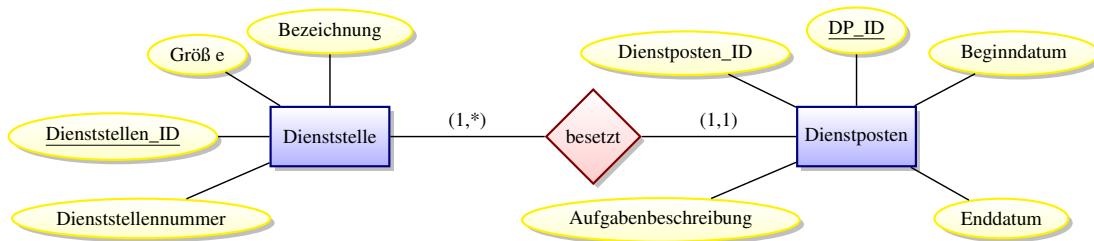
- Die (Min,Max)-Notation: Man betrachtet zunächst ein Objekt a auf der Seite des Objekttyps A und schreibt die Kardinalität auf die gleiche Seite des Beziehungstyps, an den Objekttyp A.
- (Chen)-Notation: Inverse Bezeichnung der Kardinalitäten wie bei der (Min,Max)-Notation. Die Kardinalität des Objekts a auf der Seite des Objekttyps A wird auf die andere Seite des Beziehungstyps, also Objekttyp B, geschrieben.
- Anschließend in gleicher Weise am Objekttyp B für die jeweilige Notation.

2.2.4 Referentielle Integrität

Die Referentielle Integrität stellt einen Satz aus zwei Regeln dar, der dazu dient, den korrekten Zusammenhang zwischen den Datensätzen zweier Tabellen zu regeln. Sie besagt:

1. Datensätze einer untergeordneten Entität dürfen nur auf existierende Datensätze ihrer übergeordneten Entität verweisen.
2. Datensätze aus einer übergeordneten Entität dürfen nur dann gelöscht werden, wenn es keine abhängigen Datensätze in einer untergeordneten Entität mehr gibt.

Hierzu ein Beispiel:



In diesem Beispiel stehen die beiden Entitäten Dienststelle und Dienstposten in Zusammenhang. Die Entität Dienststelle ist dabei der Entität Dienstposten übergeordnet. Wendet man die Regeln der Referentiellen Integrität an, bedeutet dies:

1. Es darf keinen Dienstposten geben, der zu einer nicht existenten Dienststelle gehört.
2. Es darf keine Dienststelle gelöscht werden, zu der es noch Dienstposten gibt.

2.3 Redundante Beziehungstypen

Lässt man den Vergleich von einem Objekttyp mit einer Dateninsel zu, kann man folgendes Bild aufbauen. Die Objekttypen werden als Dateninseln dargestellt und die Beziehungstypen bilden die Brücken zwischen diesen Inseln. Mit Hilfe der Beziehungen, die ja konkrete Ausprägungen der Beziehungstypen darstellen, kann man nun eine Brückenwanderung durchführen, indem man von den Eigenschaften eines Objektes zu den Eigenschaften des verknüpften Objektes gelangen kann.

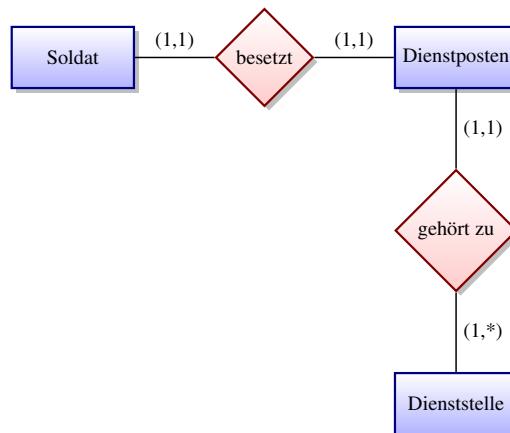
Nun können die Brücken aber so angelegt sein, dass es von Dateninsel A nach Dateninsel B zwei (oder mehr) verschiedene Wege gibt. Sind dann eine oder mehrere Brücken überflüssig? Bei der Datenmodellierung spricht man in solchen Fällen von **redundanten Beziehungstypen**. Das sind Beziehungstypen, die einen sachlogischen Zusammenhang zwischen zwei Objekttypen beschreiben, der bereits durch die Kombination anderer Beziehungstypen in gleicher Weise zum Ausdruck gebracht wird.

Der Begriff der Redundanz spielt bei der Informationsspeicherung eine große Rolle. Im praktischen Datenbankbetrieb wird zum Teil Redundanz erzeugt, um Suchprozesse innerhalb der Datenbank zu beschleunigen. In der Phase der Datenmodellierung sollte man Redundanzen vermeiden, denn diese führen u. a. zu folgenden Problemen:

- Mehrfache Eingabe derselben Informationen
- Unnötiger Speicherplatzbedarf

- Bei der Änderung der Informationen muss garantiert werden, dass alle Exemplare der redundant gespeicherten Information geändert werden, weil sonst sog. inkonsistente Daten vorliegen.

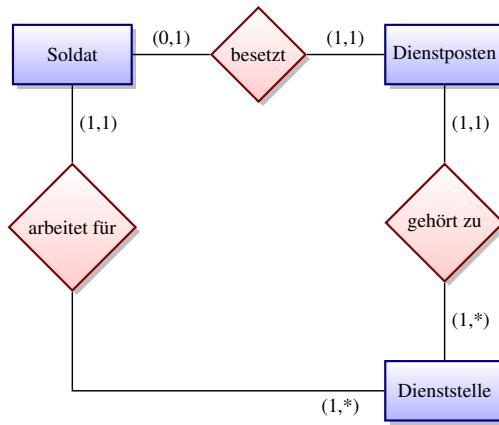
Der Verdacht auf einen redundanten Beziehungstyp ergibt sich i.d.R. bei zyklischen Beziehungstyp-Strukturen. Kann man nun aber allein aus strukturellen Merkmalen des Datenmodells die Redundanz ableiten? Wenn dies so wäre, könnten automatisierte Optimierungsprozesse diese Redundanz wieder entfernen. Zur Verdeutlichung soll das in der folgenden Abbildung gezeigte Beispiel untersucht werden.



Ein Soldat besetzt genau einen Dienstposten und ein Dienstposten kann zu gleichen Zeit auch immer nur von einem Soldaten besetzt werden. Ein Dienstposten gehört zu genau einer Dienststelle, wobei eine Dienststelle aus mindestens einem Dienstposten bestehen muss, um die Sinnhaftigkeit des Modells zu wahren.

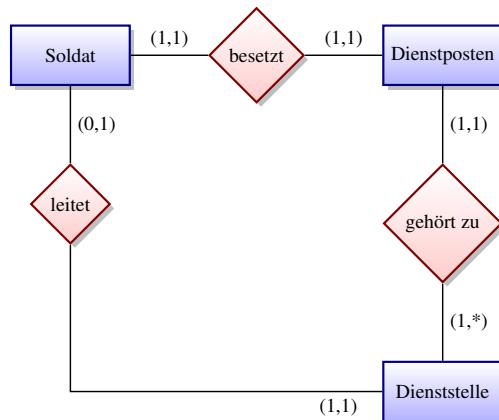
Nun kommt die „arbeitet für“-Beziehung hinzu, so dass auf Grund der entstehenden zyklischen Struktur zwei Wege vom Soldaten zur Dienststelle führen. Ist dieser Beziehungstyp nun redundant? Betrachten wir zwei verschiedene Interpretationen dieses Beziehungstyps.

Ein Soldat arbeitet für genau eine Dienststelle. Für eine Dienststelle arbeitet mindestens ein Soldat.



In diesem Falle wäre der Beziehungstyp „arbeitet für“ redundant, denn aus der Tatsache, dass ein Soldat einen Dienstposten besetzt und der Dienstposten zu einer Dienststelle gehört, folgt stets die Aussage, dass der Soldat für eine Dienststelle arbeitet.

Ein Soldat leitet höchstens eine Dienststelle und eine Dienststelle wird von genau einem Soldaten geleitet.



Der Beziehungstyp ist jetzt nicht redundant, weil aus der Tatsache, dass ein Soldat einen Dienstposten besetzt und der Dienstposten zu einer Dienststelle gehört, nicht in jedem Falle folgt, dass der Soldat die Dienststelle leitet.

Das Beispiel zeigt, dass sich die Frage, ob ein Beziehungstyp redundant ist, nicht auf Grund der Struktur des Datenmodells beantworten lässt, sondern dass sie nur durch eine inhaltliche Betrachtung der Zusammenhänge entschieden werden kann.

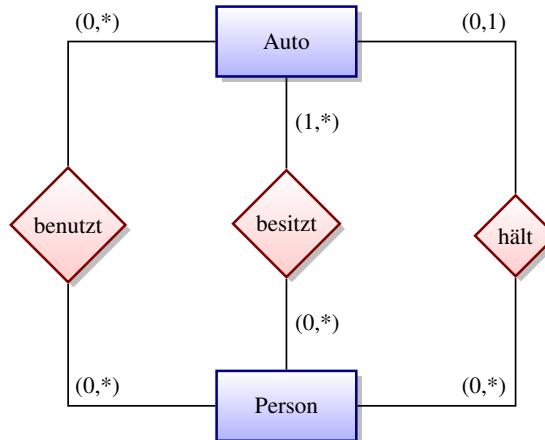
2.4 Parallelle Beziehungstypen

Häufig ist es bei der Sammlung von Informationen der Fall, dass unterschiedliche sachlogische Zusammenhänge zwischen zwei Objekttypen A und B zu berücksichtigen sind. Dies geschieht, in dem man mehrere Beziehungstypen zwischen A und B einfügt. Diese werden dann als **parallele Beziehungstypen** bezeichnet.

Sind nun Optionalität und Kardinalität der jeweiligen Beziehungstyp-Richtungen durch die beteiligten Objekttypen vorgegeben?

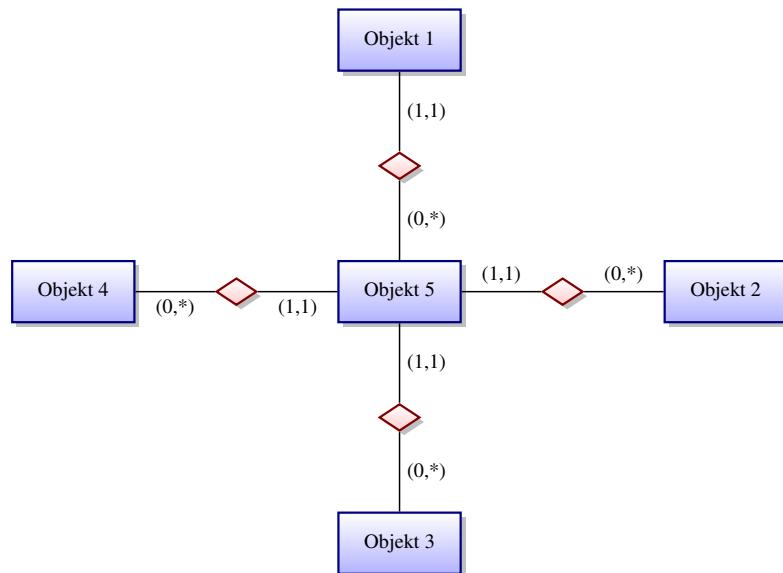
Nehmen wir an Sie wollen wie in der folgenden Abbildung dargestellt, für eine Personengruppe und eine definierte Menge von Autos drei Beziehungstypen modellieren.

Eine Person muss weder Eigentümer noch Halter noch Benutzer eines der betrachteten Autos sein, sie kann aber auch Eigentümer, Halter und Benutzer mehrerer Autos sein. Andererseits muss ein Auto mindestens einen, kann aber auch mehrere Eigentümer haben. Es kann keinen Halter haben, wenn es stillgelegt wurde, sonst aber höchstens einen. Es kann im betrachteten Zeitraum von keinem, aber auch von mehreren Personen benutzt werden. Man sieht, das die Optionalität und Kardinalität nicht allein durch die beteiligten Objekttypen festgelegt sind, sondern, dass sie durch die spezielle Semantik des jeweiligen sachlogischen Zusammenhangs bestimmt werden.



2.5 Mehrfachbeziehungen

Ein Objekttyp kann nicht nur mit einer, sondern mit beliebig vielen anderen Objekttypen in Beziehung stehen. Wenn man den Spezialfall „Parallele Beziehungstypen“ ausklammert, so lässt sich folgendes Beispiel aufzeichnen.

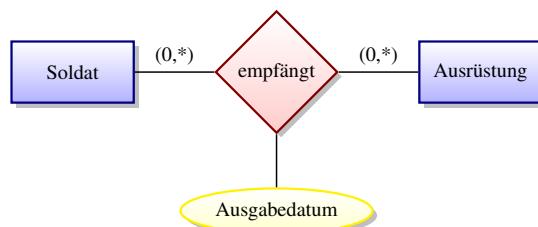


Der Objekttyp 5 steht hier mit vier anderen Objekttypen in Beziehung. Auch die übrigen Objekttypen können mit anderen Objekttypen in Beziehung stehen. Eine evtl. Problematik ergibt sich erst durch die Transformation der Beziehungstypen, da bei diesem Prozess weitere Fremdschlüssel, in die dann entstandenen Tabellen aufgenommen werden müssen. Die Transformation wird im Kapitel 4 ausführlich behandelt.

2.6 Eigenschaften von Beziehungstypen

Häufig besteht die Notwendigkeit, die konkrete Beziehung, die zwei Objekte des betrachteten Gegenstands-bereichs eingehen, genauer zu spezifizieren. Betrachten wir dazu folgenden Fall:

In der Bekleidungsstammkarte eines Soldaten werden Informationen darüber gespeichert, welcher Soldat welchen Ausrüstungsgegenstand empfangen hat. Nun soll das Datum der Ausgabe des Ausrüstungsgegenstandes gespeichert werden - in unserem Beispiel durch das Attribut „Ausgabedatum“. Diese Eigenschaft kann aber weder dem Objekttyp „Soldat“, noch dem Objekttyp „Ausrüstung“ zugeordnet werden. Es ist eine Eigenschaft des Beziehungstyps, der zwischen Soldat und Ausrüstung besteht. Folgende Abbildung stellt diese Situation dar.



2.7 Begriffe

An dieser Stelle soll eine Terminologie eingeführt werden, die beim Datenbank-Design üblich ist und in vielen Fällen eine kürzere Sprechweise ermöglicht:

- **Schlüssel:** Die minimale Kombination von Eigenschaften / Attributen durch die die Objekte eines Objekttyps eindeutig identifiziert werden können, wird als Schlüssel des Objekttyps bezeichnet. Ein Schlüssel kommt somit nicht doppelt vor. Der Eigenschaftswert des Schlüssels eines Objekttyps darf nicht leer sein.
- **Zusammengesetzter Schlüssel:** Ein Schlüssel, der sich aus mehreren Eigenschaften / Attributen zusammensetzt wird zusammengesetzter Schlüssel genannt. Häufig sind die verwendeten Eigenschaften Fremdschlüssel (siehe ??).
- **Teilschlüssel:** Ein Teilschlüssel entsteht dadurch, dass man aus einem zusammengesetzten Schlüssel wenigstens ein teilidentifizierendes Element (Attribut) entfernt.

Eine weitere und feinere Unterteilung erfolgt im Kapitel ?? (Transformation).

2.8 Übungen - Einfache ER-Modellierung

2.8.1 Übungsaufgabe Sportverein

Entwerfen Sie, basierend auf der folgenden Lage, ein ER-Modell inklusive der Beziehungen zwischen den Entitäten.

Ein Sportverein will zur besseren Verwaltung seiner eigenen Sportabteilungen, Trainer und Sportler eine Datenbank entwerfen. In der Datenbank soll ersichtlich werden, welcher Trainer welche Sportart trainiert und welcher Sportler in der jeweiligen Abteilung aktiv ist.

Im Folgenden werden die angesprochenen Datenbankinhalte spezifiziert:

- Zu jeder Sportabteilung muss eine eindeutige ID und deren vereinsinterne Bezeichnung gespeichert werden.
- Für jede Sportart ist eine ID und deren Bezeichnung wichtig. Eine Sportart wird in genau einer Abteilung durchgeführt, wobei in einer Abteilung mindestens eine Sportart durchgeführt wird.
- Für den jeweiligen Trainer ist der Vor- und Nachname, das Geburtsdatum und das Eintrittsdatum in den Verein zu speichern. Ein Trainer trainiert mindestens eine Sportart. Eine Sportart wird von höchstens einem Trainer trainiert, wobei es vorkommen kann, dass beim Ausscheiden eines Trainers aus dem Verein, eine Sportart kurzzeitig keinen Trainer hat.
- Jede Sportart wird von mindestens einem Sportler ausgeübt. Ein Sportler hingegen kann mehrere Sportarten ausüben, wobei es keine passiven Mitglieder/Sportler im Verein gibt. Bis auf die Trainernummer sind für den Sportler die selben Daten zu erheben, wie für die Trainer.
- Jede Sportabteilung hat mindestens einem Trainingsort (Adresse). Es kann sein, dass an einem Trainingsort mehrere Sportabteilungen trainieren. Bei der Adressspeicherung sind die Postleitzahl (PLZ), der Ort, die Straße und die Hausnummer relevant.
- Jeder Trainer und jeder Sportler wohnen bei genau einer Adresse. Es ist auch möglich, dass mehrere Trainer oder Sportler an der selben Adresse wohnen.

2.8.2 Übungsaufgabe IT-Helpdesk

Entwerfen Sie, basierend auf der folgenden Lage, ein ER-Modell inklusive der Beziehungen zwischen den Entitäten.

Für ein IT-Supportunternehmen soll eine Datenbank erschaffen werden, welche es ermöglicht, telefonische Supportanfragen von Kunden zu erfassen. Im Einzelnen müssen die nachfolgend beschriebenen Zusammenhänge in der Datenbank abgebildet werden.

Vorgaben

- Jeder Kunde, der den IT-Helpdesk anruft, muss mit Vorname, Nachname und Kundensnummer gespeichert werden.
- Die Datenbank muss es ermöglichen, zu einem Kunden, mindestens eine oder mehrere Adressen zu speichern. Eine Adresse besteht aus Straße, Hausnummer, Postleitzahl (PLZ) sowie Ort und muss mehreren Kunden zugeordnet werden können. Adressen zu denen keine Kunden mehr in der Datenbank existieren verbleiben noch für mindestens ein Jahr in der Datenbank, ehe sie gelöscht werden.
- Ein Kunde gibt beim IT-Helpdesk seine Kontaktdaten an. Diese Kontaktdaten bestehen meist aus Telefonnummer und E-Mail-Adresse. Die Telefonnummer muss nicht zwingend mit angegeben werden. Allerdings muss mind. eine Kontaktinformation gespeichert werden. Ein Kontaktdatensatz wird immer nur einem Kunden zugeordnet.
- Für das Management des IT-Supportunternehmen ist es wichtig zu wissen, welcher Mitarbeiter des Helpdesks mit welchem Kunden Kontakt hatte. Zu einem Mitarbeiter wird dessen Vorname, Nachname und seine Personalnummer gespeichert.
- Jedesmal wenn ein Kunde mit dem IT-Helpdesk einen Kontakt herstellt, muss der Mitarbeiter des Helpdesks die genaue Uhrzeit, das Datum, den Anlass und die Dauer der Dienstleistung notieren.

Zusatzaufgabe

Das IT-Supportunternehmen hat angefragt, ob es möglich ist, die Datenbank so zu verändern, dass mehrere Mitarbeiter an einem Kontakt arbeiten können. Jedesmal wenn ein Mitarbeiter an einem Kon-

takt arbeitet, müssen die Uhrzeit und das Datum des Telefonats, sowie dessen Dauer gespeichert werden.

Prüfen Sie, ob diese Möglichkeit gegeben ist und falls Ja, passen Sie das Modell entsprechend an!



2.8.3 Übungsaufgabe Unternehmensberatung

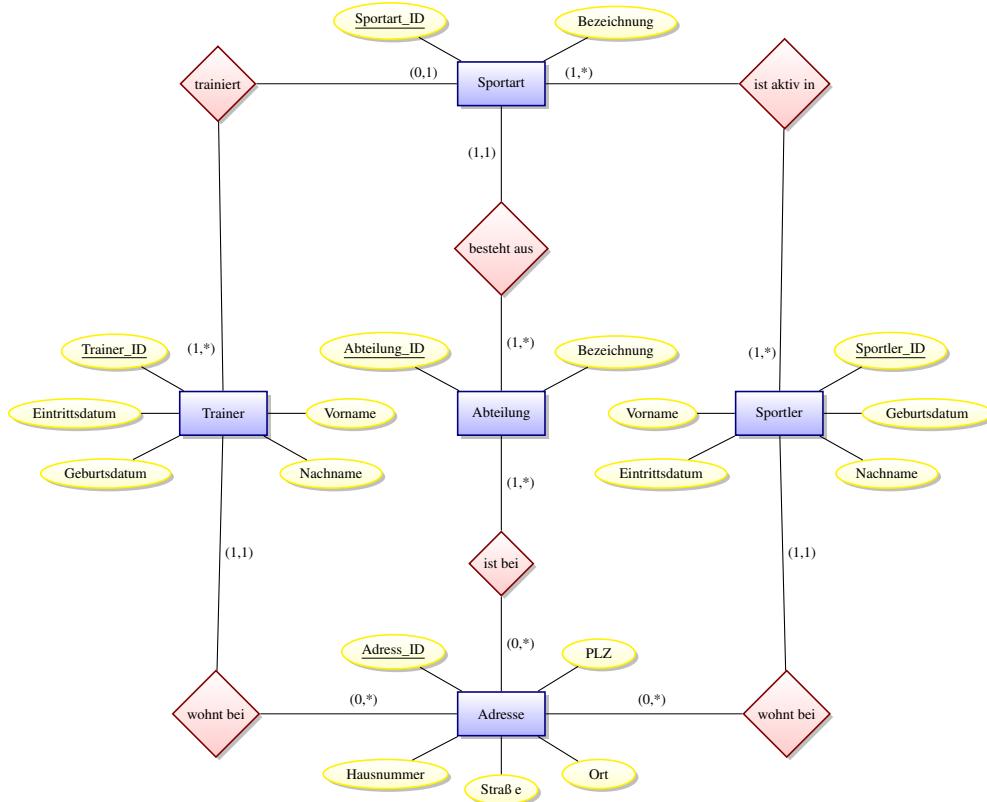
Entwerfen Sie, basierend auf der folgenden Lage, ein ER-Modell, inklusive der Beziehungen zwischen den Entitäten.

Ein Kunde tritt an die UBE Unternehmensberatung heran und gibt den Auftrag, seine Unternehmensstruktur als Datenbank abzubilden. Nach einer ersten Besprechung mit dem Kunden, stehen folgende Fakten fest:

- Die Datenbank muss so gestaltet sein, dass alle einzelnen Produktionsbetriebe des Kunden mit Betriebs_ID und Bezeichnung gespeichert werden können.
- Ein Produktionsbetrieb besteht aus mindestens einer Abteilung. Eine Abteilung gehört zu genau einem Betrieb. Zu jeder Abteilung muss deren Abteilungs_ID und die Bezeichnung gespeichert werden.
- In einer Abteilung arbeitet mindestens ein Mitarbeiter. Jeder Mitarbeiter arbeitet immer nur in einer Abteilung. Zu jedem Mitarbeiter muss dessen Mitarbeiter_ID, sein Name und sein Gehalt gespeichert werden.
- Es müssen alle Produkte gespeichert werden, die verkauft werden. Zu jedem Produkt ist die Produkt_ID und dessen Bezeichnung wichtig.
- Da verschiedene Betriebe auch Projekte durchführen, müssen diese mit Projekt_ID und Bezeichnung gespeichert werden.
- Produkte werden immer von mindestens einem Mitarbeiter verkauft, wobei nicht alle Mitarbeiter mit dem Verkauf von Produkten beschäftigt sind, da einige auch in Projekten arbeiten. Prinzipiell kann ein Mitarbeiter aber für mehrere Produkte zuständig sein. Es ist relevant, wie viele Stunden ein Mitarbeiter mit dem Verkauf von Produkten beschäftigt ist.
- Es müssen auch die Mitarbeiter berücksichtigt werden, die nicht am Verkauf von Produkten, sondern an einer Projektarbeit beteiligt sind. An einem Projekt arbeitet immer mindestens ein Mitarbeiter, wobei jeder Mitarbeiter an höchstens einem Projekt teilnehmen kann. Jedem Projekt muss genau ein Mitarbeiter als Projektleiter zugeteilt werden. Jeder Mitarbeiter kann immer nur höchstens ein Projekt leiten.

2.9 Lösungen - Einfache ER-Modellierung

2.9.1 Übungsaufgabe Sportverein

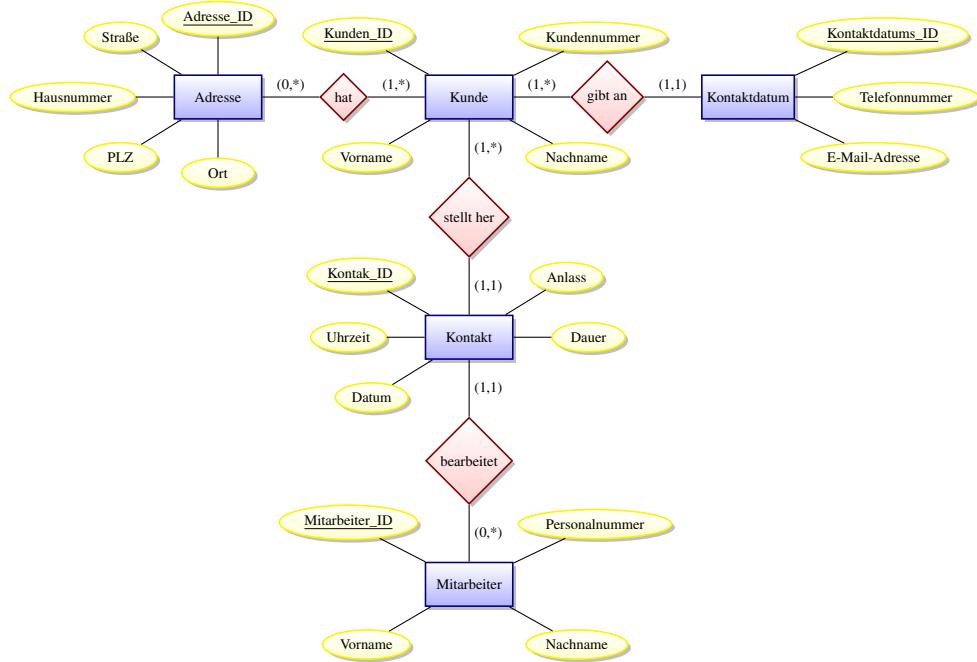


Transformation

Abteilung	(<u>Abteilung_ID</u> , Bezeichnung)
Sportart	(<u>Sportart_ID</u> , Bezeichnung, ↑Trainer_ID↑, ↑Abteilung_ID↑ [NN])
Trainer	(<u>Trainer_ID</u> , Vorname, Nachname, Geburtsdatum, Eintrittsdatum, ↑Adresse_ID↑ [NN])
Adresse	(<u>Adresse_ID</u> , PLZ, Ort, Strasse, Hausnummer)
Sportler	(<u>Sportler_ID</u> , Vorname, Nachname, Geburtsdatum, Eintrittsdatum, ↑Adresse_ID↑ [NN])
SportartSportler	(↑ <u>Sportart_ID</u> + <u>Sportler_ID</u> ↑)
AbteilungAdresse	(↑ <u>Abteilung_ID</u> + <u>Adresse_ID</u> ↑)

2.9.2 Übungsaufgabe IT-Helpdesk

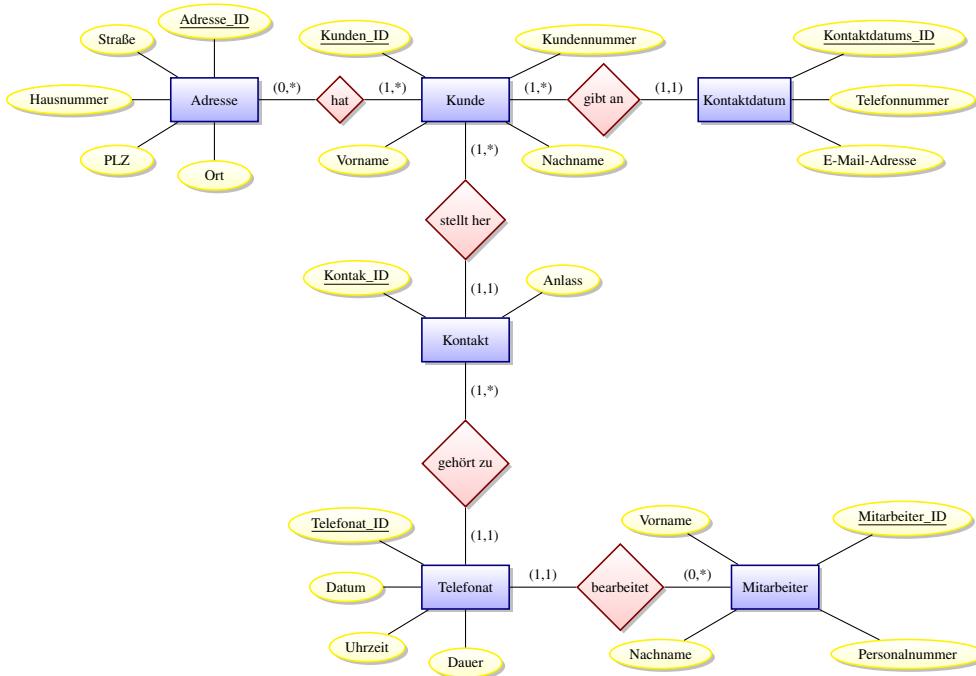
Vorgaben



Transformation

Kunde	(<u>Kunde_ID</u> , Kundennummer, Vorname, Nachname)
Adresse	(<u>Adresse_ID</u> , PLZ, Ort, Straße, Hausnummer)
KundeAdresse	(\uparrow <u>Kunde_ID</u> + <u>Adresse_ID</u> \uparrow)
Kontaktdatum	(<u>Kontaktdatums_ID</u> , E-Mail-Adresse, Telefonnummer, \uparrow <u>Kunde_ID</u> \uparrow [NN])
Kontakt	(<u>Kontak_ID</u> , \uparrow <u>Kunde_ID</u> \uparrow [NN], \uparrow <u>Mitarbeiter_ID</u> \uparrow [NN], Uhrzeit, Datum, Dauer, Anlass)
Mitarbeiter	(<u>Mitarbeiter_ID</u> , Personalnummer, Vorname, Nachname)

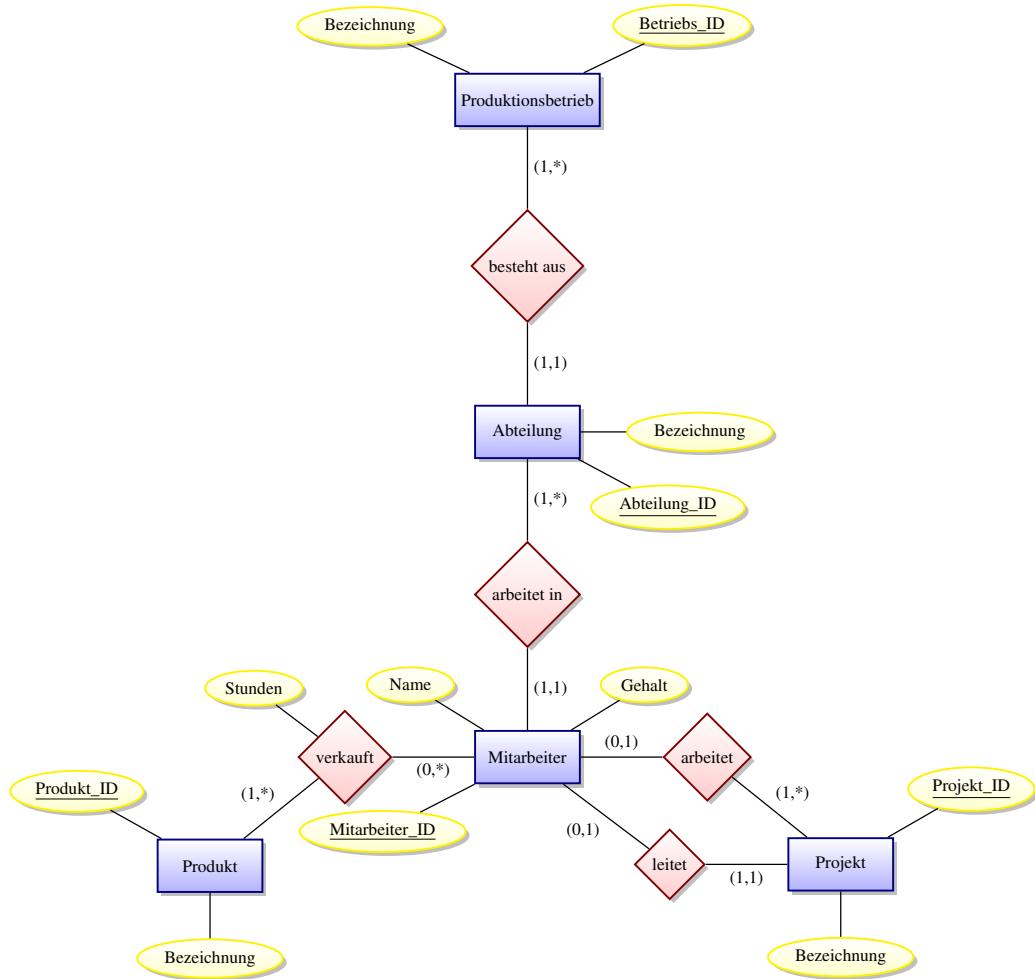
Zusatzaufgabe



Transformation

Kunde	(Kunde_ID, Kundennummer, Vorname, Nachname)
Adresse	(Adresse_ID, PLZ, Ort, Strasse, Hausnummer)
KundeAdresse	(↑Kunde_ID + Adresse_ID↑)
Kontaktdatum	(Kontaktdatums_ID, E-Mail-Adresse, Telefonnummer, ↑Kunde_ID↑ [NN])
Kontakt	(Kontak_ID, ↑Kunde_ID↑ [NN], Anlass)
Telefonat	(Telefonat_ID, ↑Kontakt_ID↑ [NN], ↑Mitarbeiter_ID↑ [NN], Datum, Uhrzeit, Dauer)
Mitarbeiter	(Mitarbeiter_ID, Personalnummer, Vorname, Nachname)

2.9.3 Übungsaufgabe Unternehmensberatung



Transformation

Produktionsbetrieb	(Betriebs_ID, Bezeichnung)
Abteilung	(Abteilungs_ID, Bezeichnung, ↑Betriebs_ID↑ [NN])
Mitarbeiter	(Mitarbeiter_ID, Name, Gehalt, ↑Abteilungs_ID↑ [NN], ↑Projekt_ID↑ [NN])
Projekt	(Projekt_ID, Bezeichnung, ↑Leiter_ID↑ [UN] [NN])
Produkt	(Produkt_ID, Bezeichnung)
ProduktMitarbeiter	(↑Produkt_ID + Mitarbeiter_ID↑, Stunden)

3 Erweiterte ER Modellierung

Inhaltsangabe

In diesem Abschnitt wird das ER-Modell um die drei Eigenschaften Rekursion, Spezialisierung und Generalisierung erweitert, so dass eine umfangreichere und genauere Beschreibung des betrachteten Gegenstandsbereichs möglich ist.

3.1 Rekursive Beziehungen

Häufig ist es wichtig und notwendig den sachlogischen Zusammenhang zwischen zwei Objekten festzuhalten, die beide demselben Objekttyp angehören. So ist es für ein Unternehmen nützlich zu wissen, welcher Mitarbeiter durch welche anderen Mitarbeiter - im Krankheits- oder Urlaubsfall - vertreten werden kann.

3.1.1 Definition eines rekursiven Beziehungstyps



Ein rekursiver Beziehungstyp beschreibt den sachlogischen Zusammenhang zwischen Objekten, die dem gleichen Objekttyp angehören.

Für einen rekursiven Beziehungstyp sind dieselben Angaben erforderlich, wie für einen binären. Somit ist für einen solchen Beziehungstyp folgendes festzulegen (siehe ??):

1. eine aussagekräftige Benennung
2. eine Angabe zur Optionalität
3. eine Angabe zur Kardinalität

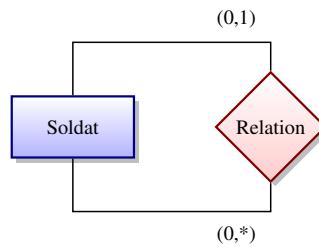
Bei der traditionellen Informationsspeicherung, mit Hilfe von Karteikärtchen, wird der rekursive Beziehungstyp durch Querverweise, innerhalb des Karteikastens, realisiert. Im betrachteten Vertretungsbeispiel würden auf der Karteikarte des Mitarbeiters die Personalnummern seiner möglichen Vertreter notiert werden.

3.1.2 Syntaxregeln für rekursive Beziehungstypen

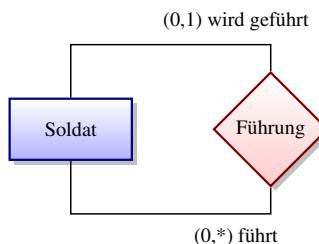
- Ein rekursiver Beziehungstyp zur Kennzeichnung eines sachlogischen Zusammenhangs zwischen den Objekten ein- und desselben Objekttyps A wird als eine Verbindungsline dargestellt, die den

Objekttyp A mit sich selbst verbindet. Diese Form der Verbindungsline, die aus A austritt und zu A zurückführt, ist der Anlass für die Bezeichnung „rekursiv“¹

- Die Benennung des Beziehungstyps steht in einer Raute, am Beziehungstyp (siehe ??)
- Optionalität und Kardinalität der jeweiligen Beziehungstyp-Richtung werden in gleicher Weise angegeben, wie beim binären Beziehungstyp.



Setzt man beim Führungsverhältnis der Soldaten voraus, dass ein Soldat von höchstens einem anderen Soldaten geführt wird und dass er selbst mehrere Soldaten führen kann, dann ergibt sich das folgende Datenmodell.



Bei rekursiven Beziehungen ist eine genauere Beschreibung der Beziehungstyp-Richtung notwendig. Im o. a. Beispiel wird der Sachverhalt „Führung“ durch „führt“ und „wird geführt“ näher erläutert.



Die Angaben sind folgendermaß en zu lesen:

- Ein Soldat „führt“ keinen oder mehrere andere Soldaten und
- ein Soldat „wird geführt“ von höchstens einem anderen Soldaten

Betrachtet man bei diesem Beziehungstyp Kardinalität und Optionalität, stellt sich nach einiger Überlegung die Frage, ob alle Kombinationen, welche bei den binären Beziehungstypen existieren, auch bei rekursiven wiederzufinden sind.

¹rekursiv = zurückführend

Die Besonderheit eines rekursiven Beziehungstyps gegenüber einem binären Beziehungstyp, der Objekttyp A und B miteinander verknüpft, besteht darin, dass die Objekttypen A und B identisch sind. Somit sind rekursive Beziehungstypen nur dann möglich, wenn die Objekttypen A und B, die Mengen von Objekten repräsentieren, die dieselbe Mächtigkeit besitzen können.



Die Mächtigkeit eines Objekttyps beschreibt die genaue Anzahl der in ihm zusammengefassten Objekte.

An dieser Stelle soll dieser Sachverhalt nicht näher erläutert, sondern nur das für die Modellierung relevante Ergebnis betrachtet werden.

Die Tabelle ?? „Mögliche Kombinationen von rekursiven Beziehungstypen“ zeigt, in der (Min,Max)-Notation, die möglichen Kombinationen von Kardinalität und Optionalität und trifft eine Aussage, über die Verwendungsmöglichkeit bei rekursiven Beziehungstypen.

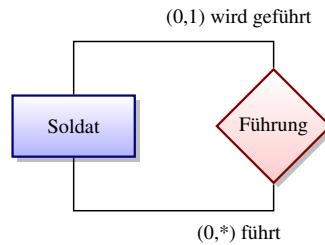
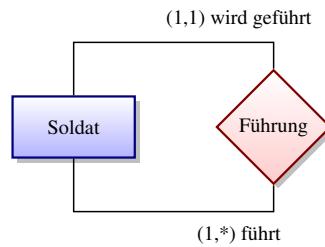
Tabelle 3.1: Mögliche Kombinationen von rekursiven Beziehungstypen

Beziehungstyp	Verwendung bei rekursiven Beziehungstypen
(1,1):(1,1)	möglich
(1,1):(0,1)	nicht möglich
(0,1):(0,1)	möglich
(1,1):(1,*)	nicht möglich
(0,1):(1,*)	nicht möglich
(1,1):(0,*)	möglich
(0,1):(0,*)	möglich
(1,*)(1,*)	möglich
(1,*)(0,*)	möglich
(0,*)(0,*)	möglich

Die sieben möglichen Kombinationen werden, im späteren Verlauf dieser Unterlage (Kapitel 4 Transformation), im Zusammenhang mit ihrer Repräsentation im physischen Datenbankmodell, der Reihe nach untersucht und durch Beispiele veranschaulicht.

3.2 Anwendung rekursiver Beziehungstypen

Hier soll nun gezeigt werden, wie Tabelle ?? hilft, in der Praxis Fehler zu vermeiden. Die nächste Abbildung zeigt zwei Entitäten (Version A und B), die das Verhältnis der Soldaten bzgl. Führung und geführt werden darstellen sollen.



Beide Versionen sollen bezüglich der nächst genannten Fragen untersucht werden:



- Was sagen die Versionen aus?
- Wird die Realität richtig abgebildet?
- Kann man sie realisieren (gemäß Tabelle ??)?

3.2.1 Version A

Was sagt die Version aus?

Ein Soldat führt mindestens einen oder mehrere andere Soldaten und ein Soldat wird von genau einem anderen Soldaten geführt.

In dieser Abbildung der Realität gibt es nur Vorgesetzte, da jeder Soldat immer mindestens einen anderen führt.

Wird die Realität richtig abgebildet?

Es gibt laut Version A nur solche Soldaten, die andere führen. Doch in der Realität ist dies sicherlich nicht der Fall, da z.B. ein Soldat in der Grundausbildung nicht immer einen anderen Soldaten führen wird. In

der anderen Fragerichtung muss man ebenfalls alle Soldaten betrachten. Die Aussage, dass ein Soldat von genau einem anderen geführt wird, betrachtet den Chef der Einheit nicht. Dieser sollte, in Bezug auf die eigene Einheit, keinen Vorgesetzten haben.

Es zeigt sich, dass die Realität mit sehr hoher Wahrscheinlichkeit nicht korrekt abgebildet wurde. Ausnahmen kann es natürlich jederzeit geben.

Kann man sie realisieren?

Tabelle ?? trifft, was die Realisierung in einem relationalem Datenbanksystem angeht, in Zeile 4 die klare Aussage, dass es nicht möglich ist. Man kann anhand der Tabelle schon einen Hinweis auf evtl. Fehler in der Modellierung finden.

3.2.2 Version B

Was sagt die Version aus?

Ein Soldat führt null oder mehrere Untergebene und ein Soldat wird von null oder höchstens einem anderen Soldaten geführt. Dabei kann es sich entweder um den Chef oder einen Untergebenen handeln.

Wird die Realität richtig abgebildet?

Version B stellt die Möglichkeit bereit, dass in der Tabelle Soldat sowohl der Chef als auch die Untergebenen aufgeführt werden können, was sich darin widerspiegelt, dass ein Soldat keinen oder höchstens einen Chef haben kann und ein Soldat keinen oder mehrere Soldaten führen kann. Dies entspricht, nach der allgemeinen Auffassung, der Realität.

Kann man sie realisieren?

In Tabelle ?? gibt in diesem Fall Zeile 7 die Auskunft, dass eine rekursive Beziehung mit diesen Kardinalitäten möglich ist. Eine Bestätigung, dass die Realität mit großer Wahrscheinlichkeit richtig modelliert wurde. Eine hundertprozentige Aussage über richtig oder falsch kann man in diesem Verfahren jedoch nicht treffen.

3.3 Spezialisierung

Die Spezialisierung ist eine Vorgehensweise, die Objekttypen als Mengen betrachtet. Es kann für die ER-Modellierung unter Umständen sinnvoll sein, eine Menge in mehrere Teilmengen zu zerlegen. Eine solche Zerlegung wird meist dann vorgenommen, wenn die Teilmengen eigene Attribute haben, die nur in den jeweiligen Teilmengen vorkommen. Auf diese Weise werden leere Felder in der Datenbank, sog. NULL-Werte vermieden, da möglichst immer alle Attribute mit Werten befüllt werden. Ein Beispiel hierzu:

Die Menge aller Personen an einer Universität enthält die beiden Teilmengen „Student“ und „WiMa“². Die Menge „Person“ wird als „Supertyp“, der übergeordnet Objekttyp, bezeichnet und die beiden Teilmengen „Student“ und „WiMa“ als „Subtyp“, die untergeordneten Objekttypen.

²WiMa = Wissenschaftlicher Mitarbeiter

Ein Student besitzt eine Matrikelnummer, mit der er an der Universität registriert wurde. Der WiMa besitzt diese nicht, bezieht aber ein Gehalt für seine wissenschaftlichen Studienarbeiten. Ein Attribut „Gehalt“ macht für den Studenten keinen Sinn, da er keines empfängt und der WiMa benötigt keine Matrikelnummer. Durch die Trennung von „Person“ in „Student“ und „WiMa“ werden unnötige NULL-Werte in der Datenbank vermieden.

Wie so vieles im Leben ist das Spezialisieren von Objekttypen jedoch nicht ganz so einfach, wie es im ersten Moment scheint. Bevor auf die insgesamt vier verschiedenen Variationen dieses Verfahrens eingegangen werden kann, werden noch einige Definitionen benötigt.

3.3.1 Definitionen

Disjunkt

Ein System von Subtypen und Supertypen heisst disjunkt, wenn die einzelnen Subtypen keine gemeinsamen Elemente haben, d. h. ein Datensatz kann nur in einem der Subtypen auftreten.

Vollständige Überdeckung

Ein System von Subtypen und Supertypen nennt man vollständig, wenn der Supertyp keine eigenen Elemente enthält, also jedes Element in einem der Subtypen enthalten ist.



Die beiden Begriffe „Disjunkt“ und „Vollständige Überdeckung“ werden als Konsistenzbedingungen bezeichnet. Konsistenzbedingungen ergeben sich entweder aus feststehenden Tatsachen oder sie müssen durch den Designer definiert werden!

3.3.2 Kombinationen

Die Unter- und Obermengenbeziehungen lassen sich mit diesen Begriffen in vier verschiedene Systeme einteilen:

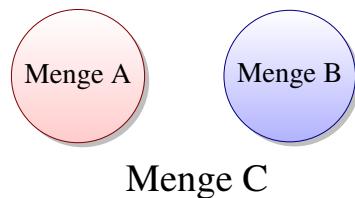
1. Vollständige Überdeckung und Überlappung nicht zugelassen (= disjunkt, kein gemeinsames Element)

2. Vollständige Überdeckung und Überlappung zugelassen (= nicht disjunkt, gemeinsame Elemente möglich)
3. Nicht vollständige Überdeckung und Überlappung nicht zugelassen (= disjunkt)
4. Nicht vollständige Überdeckung und Überlappung zugelassen (= nicht disjunkt)

Es handelt sich bei der Spezialisierung um den Sonderfall eines 1:1 Beziehungstyps mit besonderer Semantik. Es werden nun die oben angesprochenen Fälle erläutert und anhand von Beispielen, die immer zwei Untermengen angeben, veranschaulicht. In der Realität kann es natürlich weitere spezialisierte Objekttypen geben.

3.3.3 Vollständige Überdeckung und Disjunkt

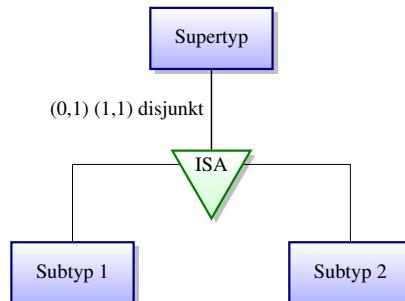
Wenn man die Objektmengen dieser 1. Beziehungsart grafisch darstellt, ergibt sich diese Abbildung:



Die Objektmenge C besteht hier vollständig aus den Objektmengen A und B, d. h. es gibt keine Elemente, die den beiden Mengen A und B nicht zuordenbar sind. Des Weiteren gibt es keine Schnittmenge zwischen A und B.

$$C := \{x \mid ((x \in A) \wedge (x \notin B)) \vee ((x \in B) \wedge (x \notin A))\}$$

Dieser Fall wird im ER-Modell folgendermaßen dargestellt.





Spezifische oder lokale Attribute hängen an den Untermengen, dies sind hier die Subtypen 1 und 2.

Der Supertyp (Person) umfasst alle Angestellten einer Universität. Der Subtyp 1 (Student) beinhaltet alle Studenten dieser Universität und der Subtyp 2 (WiMa) beinhaltet alle wissenschaftlichen Mitarbeiter (WiMa) einer Universität. Es existieren im Supertyp „Person“ nur Objekte, deren identifizierendes Attribut als Fremdschlüssel entweder im Subtyp „Student“ oder im Subtyp „WiMa“ vorkommt. Die drei folgenden Tabellen zeigen verschiedene Ausprägungen der Subtypen.

Tabelle 3.2: Person

Person_ID	Vorname	Nachname	Geburtsdatum
1	Heinz	Meier	01.02.1990
2	Michael	Schulz	02.05.1980
3	Frank	Bertling	04.10.1981
4	Hans	Fasshauer	07.09.1991

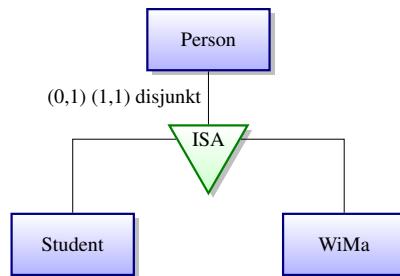
Tabelle 3.3: Student

Person_ID	Matrikelnummer
1	65420
4	66530

Tabelle 3.4: WiMa

Person_ID	Gehalt
2	3000
3	3150

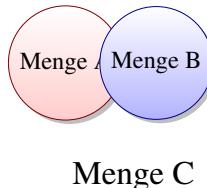
Zusammenfassend ergibt sich für den 1. Fall die folgende Abbildung:



Es müssen zusätzliche programmtechnische Maßnahmen getroffen werden, um sicherzustellen, dass jedes Objekt des Supertyps genau ein zugehöriges Objekt in Subtyp 1 oder Subtyp 2 besitzt.

3.3.4 Vollständige Überdeckung und nicht Disjunkt

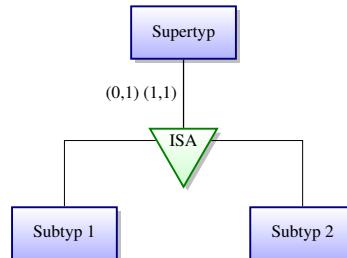
Wenn man die Objektmengen dieser 2. Beziehungsart grafisch darstellt, ergibt sich die folgende Abbildung:



Hier besteht die Objektmenge C ebenfalls vollständig aus den Objektmengen A und B, was wiederum heißt, dass es keine Elemente gibt, die den beiden Mengen A und B nicht zuordenbar sind.

Der Unterschied zu Fall 1 ist, dass es hier eine Schnittmenge zwischen A und B gibt.

Fall 2 wird im ER-Modell sehr ähnlich dargestellt, wie Fall 1, nur mit dem Unterschied, dass das Schlüsselwort „Disjunkt“ entfällt.



Nun soll das im vorigen Abschnitt eingeführte Beispiel herangezogen werden. Als Änderung soll ein WiMa an der selben Universität auch noch studieren können.

Tabelle 3.5: Person

Person_ID	Vorname	Nachname	Geburtsdatum
1	Heinz	Meier	01.02.1990
2	Michael	Schulz	02.05.1980
3	Frank	Bertling	04.10.1981
4	Hans	Fasshauer	07.09.1991
5	Tobias	Schreiber	20.07.1983

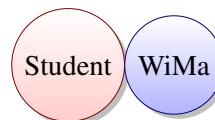
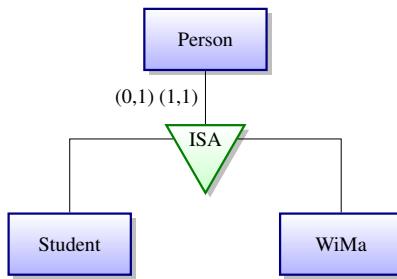
Tabelle 3.6: Student

Person_ID	Matrikelnummer
1	65420
4	66530
5	66460

Tabelle 3.7: WiMa

Person_ID	Gehalt
2	3000
3	3150
5	2900

Die Person mit der Nummer 5 ist ein studierender WiMa, welcher ein Aufbaustudium an der selben Universität absolviert und gehört sowohl dem Subtyp „Student“ als auch dem Subtyp „WiMa“ an. Als identifizierendes Attribut wird die Eigenschaft „Person_ID“ verwendet. Es ergibt sich für das Beispiel die folgende Abbildung:

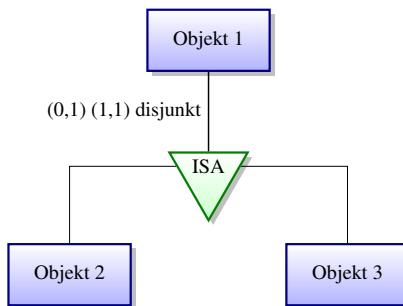
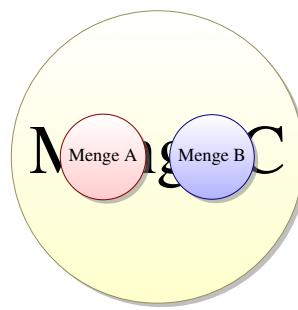


3.3.5 Nicht vollständige Überdeckung und Disjunkt

In Fall Nummer 3 besteht die Obermenge C aus den beiden Teilmengen A und B, jedoch ist es möglich, dass in der Menge C Elemente existieren, die nicht den Mengen A und B angehören. Da sich A und B disjunkt verhalten existiert keine Schnittmenge zwischen A und B.

$$C := \{x \mid (((x \in A) \wedge (x \notin B)) \vee ((x \in B) \wedge (x \notin A))) \vee ((x \notin A) \wedge (x \notin B))\}$$

Wenn man die Objektmengen dieser 3. Beziehungsart grafisch darstellt, ergibt sich folgende Abbildung:



Es wird wieder auf das Universitätsbeispiel zurückgegriffen. Diesmal existieren in dem Supertyp keine Objekte, deren identifizierendes Attribut sowohl im Subtyp „Student“ als auch im Subtyp „WiMa“ vorkommt. Dies wird in den folgenden Tabellen gezeigt.

Tabelle 3.8: Person

Person_ID	Vorname	Nachname	Geburtsdatum
1	Heinz	Meier	01.02.1990
2	Michael	Schulz	02.05.1980
3	Frank	Bertling	04.10.1981
4	Hans	Fasshauer	07.09.1991
5	Tobias	Schreiber	20.07.1983
6	Martin	Speier	21.08.1996

Tabelle 3.9: Student

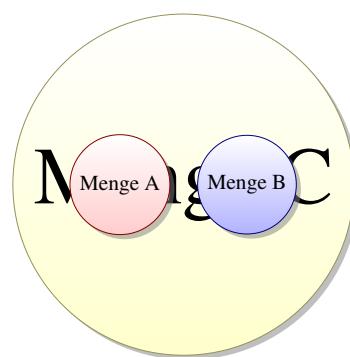
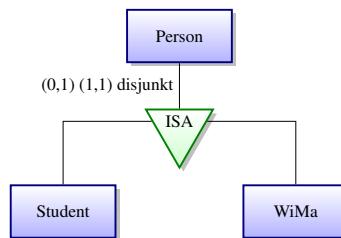
Person_ID	Matrikelnummer
1	65420
4	66530
5	66460

Tabelle 3.10: WiMa

Person_ID	Gehalt
2	3000
3	3150

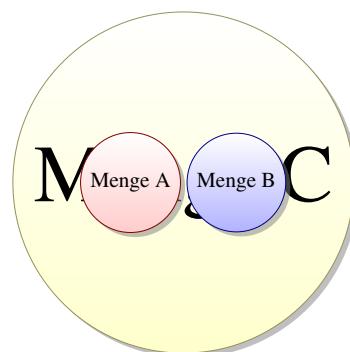
Die Person mit der Person_ID 6 kommt in den Subtypen nicht vor, da sie ein Praktikant ist und nur die Attribute des Supertyps in sich vereint.

Es ergibt sich im Beispiel für den 3. Fall die folgende Abbildung:

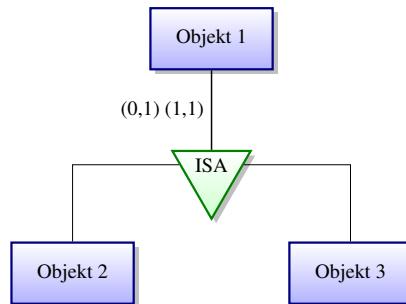


3.3.6 Nicht vollständige Überdeckung und nicht Disjunkt

Wenn man die Objektmengen dieser 4. Beziehungsart grafisch darstellt, ergibt sich folgende Abbildung:



In Fall Nummer 4 besteht die Obermenge C aus den beiden Teilmengen A und B, jedoch ist es möglich, dass in der Menge C Elemente existieren, die nicht den Mengen A und B angehören. Da dieser Fall nicht disjunkt ist, ist eine Schnittmenge zwischen A und B vorhanden.



Für den 4. Beziehungstyp wird erneut das Universitätsbeispiel herangezogen.

Tabelle 3.11: Person

Person_ID	Klasse	Vorname	Nachname	Geburtsdatum
1	S	Heinz	Meier	01.02.1990
2	W	Michael	Schulz	02.05.1980
3	W	Frank	Bertling	04.10.1981
4	S	Hans	Fasshauer	07.09.1991
5	W	Tobias	Schreiber	20.07.1983
6	M	Kai	Sperling	24.11.1980

Tabelle 3.12: Student

Person_ID	Matrikelnummer
1	65420
4	66530
5	66460

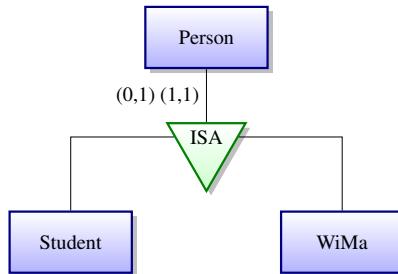
Tabelle 3.13: WiMa

Person_ID	Gehalt
2	3000
3	3150
5	2900

Hier wurde nun die Eigenschaft „Klasse“ eingeführt. Diese wird benötigt, um anzugeben, welches Objekt zu welcher Spezialisierung gehört. Diese Eigenschaft wird auch als „diskriminierende Eigenschaft“ bezeichnet.

Die Person mit der Person_ID 6 ist ein Mitarbeiter der Universität und gehört zu keinem der beiden Subtypen. Dieser Mitarbeiter vereint nur die Eigenschaften des Supertyps in sich. Als identifizierendes Attribut wird die Eigenschaft „Person_ID“ verwendet.

Es ergibt sich für den 4. Fall im Beispiel die folgende Abbildung:



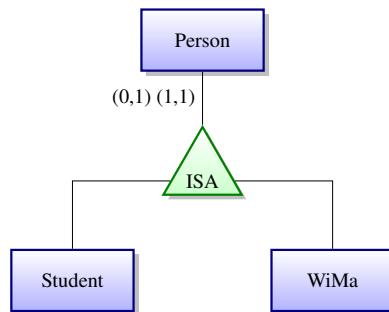
3.4 Generalisierung

Unter Generalisierung versteht man den Prozess der Gewinnung einer Obermenge aus mehreren ähnlichen Untermengen. Aus der gewonnenen Obermenge entsteht ein neuer Objekttyp, der durch diejenigen Eigenschaften beschrieben wird, die den ähnlichen Untermengen gemeinsam sind. Es handelt sich hierbei also um den Umkehrprozess zur Spezialisierung. Bei der Generalisierung handelt es sich um einen „Bottom-up-Ansatz“, zu dem die Spezialisierung den „Top-down-Ansatz“ bildet. In der Praxis findet man oft die Kombination aus beiden Ansätzen.

Gemeinsame Merkmale bzw. Eigenschaften von Objekttypen oder Subtypen werden identifiziert und zu einem einzigen Objekttyp (Obermenge) generalisiert.

3.4.1 Beispiel für die Generalisierung

Die Objekttypen „Student“ und „WiMa“ können zum Objekttyp „Person“ generalisiert werden. „Student“ und „WiMa“ sind jetzt Untermengen der generalisierten Obermenge „Person“. Bei der Generalisierung ist ebenfalls zu unterscheiden, ob bei den Mengen eine Überdeckung und/oder eine Überlappung möglich sein soll (vgl. Fallunterscheidung bei Spezialisierung). Im ER-Modell sieht dieser Sachverhalt folgendermaßen aus:



Das Ergebnis der Generalisierung unterscheidet sich zur Spezialisierung nicht, es handelt sich wie oben beschrieben nur um zwei verschiedene Ansätze der Modellierung.

Für die Generalisierung gilt ebenfalls, dass es mehr als zwei Subtypen geben kann. In der grafischen Darstellung werden diese weiteren Subtypen an das Dreieck angehängt. Im dargestellten Beispiel für die Generalisierung kann z.B. der Subtyp „Professor“ angehängt werden, um das Beispiel zu erweitern.

Würde ein Objekttyp „Praktikant“ in das Beispiel aufgenommen, ohne dass ein eigener Subtyp für ihn geschaffen wird, müsste er im Objekttyp „Person“ gespeichert werden, woraus folgt, dass das Beispiel keine vollständige Überdeckung mehr bietet. In der Mengendarstellung wäre der „Praktikant“ außerhalb von „Student“ und „WiMa“ im Rechteck von „Person“ zu finden.

3.5 Übungen - Erweiterte ER-Modellierung

3.5.1 Übungsaufgabe Schwimmbad

Entwerfen Sie, basierend auf der folgenden Lage, ein ER-Modell, inklusive der Beziehungen zwischen den Entitäten.

Für ein großes Freizeitbad soll eine Datenbank für die Buchhaltung geschaffen werden. Im Einzelnen müssen die nachfolgend beschriebenen Zusammenhänge in der Datenbank abgebildet werden.

Vorgaben

- Ein Freizeitbad besitzt 15 verschiedene Kartentypen, von denen deren Bezeichnung, der Kartenpreis und eine eindeutige ID zu speichern sind. Die unterschiedlichen Kartentypen sind nachfolgend aufgeschlüsselt:
 - Stundenkarte (2 und 4 Stunden) für Kinder, Erwachsene, Studenten
 - Tageskarte für Kinder, Erwachsene, Studenten
 - Monatskarte für Kinder, Erwachsene, Studenten
 - Saisonkarte für Kinder, Erwachsene, Studenten
- Für die Buchhaltung ist es wichtig, dass jede verkauft Eintrittskarte gespeichert wird, so dass am Jahresende eine korrekte Steuererklärung erstellt werden kann. Zu jeder Eintrittskarte wird eine Karten_ID, ein Barcode und das Verkaufsdatum gespeichert.
- Jede Eintrittskarte ist von genau einem Kartentyp, wobei es vorkommen kann, dass von einem Kartentyp keine Eintrittskarte verkauft wird.
- Der Verkauf der Eintrittskarten erfolgt durch die Bademeister. Zu jedem Bademeister ist sein Name (Vorname, Nachname), seine Wohnadresse, das Datum der Teilnahme am letzten Rettungsschwimmkurs und das Teilnahmedatum am letzten Erstehilfekurs, sowie eine eindeutige ID zu speichern.
- Wird eine Eintrittskarte verkauft, geschieht dies durch einen Bademeister. Es ist immer mindestens ein Bademeister mit dem Verkauf von Eintrittskarten beschäftigt.

- Die Bademeister können noch andere Aufgaben (Aufsicht, Reinigungsarbeiten, Wartungsarbeiten, etc.), die in Arbeitsplänen zusammengestellt werden, ausführen. Jeder Bademeister muss im System festhalten, wann er welche Arbeit auf seinem aktuellen Arbeitsplan erledigt hat. Dabei ist es möglich, dass eine Aufgabe von keinem Bademeister erledigt wird.
- Für jede Aufgabe ist deren Kurzbezeichnung, eine Beschreibung und die Arbeitsdauer in Stunden, sowie eine eindeutige ID zu speichern.

Zusatzaufgaben

1. Nach der Fertigstellung des Modells verlangt der Kunde, dass der Preis eines Kartentyps abhängig von der jeweiligen Badesaison sein muss. Für eine Saison wird deren Bezeichnung (z. B. Sommer 2012), das Anfangsdatum und das Enddatum gespeichert.

Modellieren Sie dieses Szenario und entscheiden Sie selbst, welche Kardinalitäten für die Beziehung/Beziehungen notwendig sind!

2. Unser Kunde, das Freizeitbad, benötigt eine weitere Änderung am bestehenden System. Bei der Ermittlung der Vorgaben wurde vergessen, dass alle Monats- und Saisonkarten Namensbezogen (Vorname, Nachname, Adresse) und mit einem Foto versehen, verkauft werden. Außerdem muss bei den Stundenkarten ein Zeitstempel gespeichert werden, so dass die Datenbank automatisch errechnen kann, wann die Person das Bad wieder verlassen, bzw. ob die Person bereits eine Nachzahlung leisten muss.
3. In einer erneuten Anfrage bittet die Leitung des Freizeitbades darum, dass eine weitere Änderung eingearbeitet wird. Wenn einem Bademeister gekündigt wird, wird dieser nicht aus dem System gelöscht, sondern nur sein Kündigungsdatum gespeichert.

Setzen Sie diese Änderung so um, dass keine NULL-Werte im System erzeugt werden.

3.5.2 Übungsaufgabe Kindergarten

Entwerfen Sie, basierend auf der folgenden Lage, ein ER-Modell, inklusive der Beziehungen zwischen den Entitäten.

Ein Kindergarten möchte alle anfallenden Daten in einer Datenbank speichern.

Er wird in Gruppen unterteilt. Zu jeder Gruppe ist deren Bezeichnung und der zugehörige Raum zu speichern. Des Weiteren steht in jedem Gruppenraum ein Telefon, dessen Telefonnummer ebenfalls erfasst werden muss.

In der Datenbank müssen Angaben über verschiedene Arten von Personen gespeichert werden. Zu jeder Person werden ihr Vorname, ihr Nachname und das Geschlecht gespeichert. Es gibt folgende Personenarten:

- Eltern: Jeder Elternteil (Vater und/oder Mutter) muss mit seiner Adresse erfasst werden. Es ist durchaus möglich, dass Elternteile getrennt leben und daher unterschiedliche Adressen haben. Kein Elternteil arbeitet im Kindergarten.
- Kinder: Zu jedem Kind muss die Gruppenzugehörigkeit und sein Geburtsdatum gespeichert werden. In einer Kindergartengruppe ist immer mindestens ein Kind, höchstens jedoch 20 Kinder. Ein Kind wird immer genau einer Gruppe zugeordnet.
- Weiterhin ist es wichtig zu wissen, welche Eltern (Vater und/oder Mutter) zu einem Kind gehören und bei welchem Elternteil das Kind (eines oder mehrere) lebt. Es muss möglich sein, Fälle abzubilden wie z. B.: Die Eltern sind verheiratet/leben zusammen und das Kind lebt im Haushalt; die Eltern leben getrennt und das Kind lebt bei der Mutter (oder beim Vater); das Kind hat nur noch einen Elternteil und lebt in dessen Haushalt. (Hinweis: In diesem Ausschnitt der Realität gibt es keine Waisenkinder!)
- Angestellte: Jeder Angestellte wird mit Sozialversicherungsnummer (SozVersNr) und Einstellungsdatum gespeichert.

Um den geforderten Realitätsausschnitt exakt abbilden zu können, müssen die Angestellten in zwei Gruppen unterteilt werden:

- Betreuer: Jeder Betreuer betreut genau eine Gruppe, wobei eine Gruppe immer von mindestens einem aber höchstens von zwei Betreuern beaufsichtigt wird. Zu jedem Betreuer ist dessen Gehalt zu speichern.

- Praktikanten: Bei einem Praktikanten ist von vornherein bekannt, bis zu welchem Datum (Praktikumsende) sein Praktikum geht. Dieses Datum ist wichtig und muss gespeichert werden. Jeder Praktikant wird genau einem Betreuer zugeordnet, wobei ein Betreuer sich um höchstens einen Praktikanten kümmert.

3.5.3 Übungsaufgabe Bank

Der Manager einer neu gegründeten Bank beauftragt Sie mit der Erstellung eines Datenmodells für die Verwaltung der Bankgeschäfte. In der vor kurzem erfolgten Besprechung wurden folgende Eckpunkte festgelegt:

Meine Bank hat mehrere Kunden aber mindestens Einen sonst würde meine Bank nicht existieren. Alle meine Kunden besitzen eine Kunden ID, einen Vornamen, einen Nachnamen, ein Geburtsdatum und eine Rechnungsadresse.

Kunden können eines oder mehrere Konten besitzen. Jedes Konto hat jedoch genau einen Besitzer. Ein Konto ist entweder ein Sparbuch, ein Depot oder ein Girokonto. Andere Arten von Konten werden bei uns nicht geführt und es gibt auch keine Mischformen dieser drei Kontoarten. Jedes Konto soll in unserer Datenbank mit einer IBAN (international bank account number) versehen werden. Für die Sparbücher und die Girokonten ist auch das aktuelle Guthaben zu speichern. Auf unsere Sparbücher gibt es einen Habenzinssatz, für die Girokonten wird ein Sollzinssatz geführt, der bei Überziehung des Kontos zum Tragen kommt. Wer ein Girokonto besitzt, für den wird jährlich eine Kontoführungsgebühr fällig. Bei einem Depot muss eine Eröffnungsgebühr gezahlt werden.

Unsere Kunden können einer anderen Person, die ebenfalls bei uns Kunde sein muss, höchstens eine Vollmacht über ein Konto geben. Ein Kunde kann mehrere Vollmachten über verschiedene Konten besitzen.

Ein wesentlicher Bestandteil der Datenbank ist unser Buchungssystem. Auf den Konten unserer Kunden erfolgen Buchungen (Einzahlungen, Auszahlungen, Überweisungen, usw. NICHT SPEZIALISIEREN!!!), die mit einem Betrag und einem Buchungsdatum gespeichert werden müssen. Jede Buchung ist immer genau einem Konto zuordenbar, während es für ein Konto mehrere Buchungen geben kann. Es muss auch der Fall berücksichtigt werden, dass z. B. auf einem neu eröffneten Konto noch keine Buchung vorgenommen wurde.

Damit unsere Kunden auch Geschäfte mit Kunden anderer Banken tätigen können, müssen von diesen fremden Personen der Vorname, der Nachname und eine IBAN gespeichert werden. Auch benötigen wir den Namen und den BIC (bank identity code) der fremden Bank (Hinweis: Hier bietet es sich an, die Kunden in Eigenkunden unserer Bank und Fremdkunden zu unterteilen!). Es kommt auch vor, dass einer unserer Kunden Kunde bei einer anderen Bank ist und Geld von einem seiner Konten auf ein Anderes, bei einer anderen Bank, transferieren möchte.

Die Eigenkunden können von mehreren Mitarbeitern unserer Bank betreut werden. Ein Mitarbeiter kann mehrere Kunden betreuen. Die Vorgesetzten der Filialleiter, die ebenfalls als Mitarbeiter geführt werden, haben keinen Kundenkontakt. Jeder Mitarbeiter hat genau einen Vorgesetzten, außer mir selbst. Ein vorge-

setzter Mitarbeiter hat aber mehrere Mitarbeiter, die er führt. Meine Mitarbeiter sollen in der Datenbank mit Vorname, Nachname und einer Mitarbeiter ID gespeichert werden.

Jeder Mitarbeiter der Bank arbeitet in einer Bankfiliale, außer den Vorgesetzten der Filialleiter. In einer Bankfiliale arbeiten mindestens ein aber maximal zehn Mitarbeiter. Die Bankfiliale soll mit ihrer Adresse in der Datenbank gespeichert werden.

3.5.4 Übungsaufgabe Autohändler

Entwerfen Sie, basierend auf der folgenden Lage, ein ER-Modell, inklusive der Beziehungen zwischen den Entitäten.

Der Eigentümer eines namhaften Autohauses der Region beauftragt Sie eine Datenbank zu entwerfen. Bei einem ersten Gespräch erfahren Sie, dass die Datenbank benötigt wird, um das Autohaus besser zu verwalten. Des Weiteren wird die Grobgliederung der Datenbank festgehalten.

- Die Datenbank soll alle Angestellten des Autohauses aufführen. Diese unterteilen sich in Verkäufer und Mechaniker. Verkäufer sind für die Betreuung der Kunden verantwortlich und führen Autoverkäufe durch. Außerdem nehmen die Verkäufer auch Aufträge für Reparaturen an. Mechaniker sind nur für die Durchführung der Reparaturen zuständig.
- In der Datenbank sollen Vor- und Nachname, Geburtsdatum und die Adresse des jeweiligen Mitarbeiters hinterlegt werden können. Jeder Verkäufer kann mehrere Kunden betreuen, einen Verkauf vornehmen oder eine Reparatur annehmen. Allerdings wird ein Kunde nur von genau einem Verkäufer betreut. Dies gilt auch im Bezug auf einen Autoverkauf und die Annahme einer Reparatur. Der Besitzer des Autohauses bittet Sie auch zu berücksichtigen, dass ein neu eingestellter Verkäufer noch nicht all diese Tätigkeiten durchführen kann.

Bei einem zweiten Treffen mit dem Autohausbesitzer werden die Details der Datenbank besprochen:

- Kunden des Autohauses sollen mit Vor- und Nachnamen sowie ihrer Adresse im System erfasst werden. Es ist dabei zu bedenken, dass Personen, die ihr Auto nur an das Autohaus verkaufen, auch als Kunden erfasst werden, selbst wenn diese dann kein anderes Auto im Autohaus, bei Ihrem Auftraggeber, kaufen.
- Ihr Auftraggeber bittet Sie weiterhin, in der Datenbank seine gesamten Autos, inklusive der Autos seiner Kunden zu berücksichtigen. Autos können schon einem Kunden gehören oder werden an einen Kunden verkauft. Manche Kunden besitzen kein Auto z. B. Neukunden, andere haben zwei oder mehr Autos.
- Oft werden Autos zur Reparatur gebracht. Reparaturaufträge werden genau einem Auto zugeordnet. Manche Autos, z. B. Neuwagen haben noch keine Reparaturen, andere dagegen schon mehrere.

- Die Fahrzeuge werden mit der Automarke, dem Modell, dem Marktpreis und der Fahrleistung erfasst. Es ist zu erwähnen, dass zu einem Kaufvertrag ein Datum und immer nur ein Auto gehören. Hier kann davon ausgegangen werden, dass jedes Auto höchstens einmal verkauft wird.
- Als Letztes soll festgehalten werden, dass ein Mechaniker eine oder mehrere Reparaturen durchführen kann. Zu jeder Reparatur müssen deren Datum, der Rechnungspreis für die geleisteten Arbeiten und die Ersatzteile, sowie die benötigten Arbeitsstunden gespeichert werden. Eine Reparatur wird auch nur von einem Mechaniker durchgeführt.

3.5.5 Übungsaufgabe Universität

Der Leiter der Abteilung für Verwaltungsangelegenheiten einer Universität beauftragt Sie mit der Erstellung eines Datenmodells, um die Verwaltungsstruktur effizienter zu gestalten. In einer Besprechung wurden folgende Eckpunkte festgelegt:

Meine Universität gliedert sich in Fakultäten. Diese umfassen immer mindestens ein Institut. Ein Institut kann niemals zu mehreren Fakultäten gleichzeitig gehören. Für beide müssen deren Bezeichnungen gespeichert werden.

In der Datenbank sollen drei unterschiedliche Personentypen eingetragen werden. Diese sind mit ihren Vornamen, Nachnamen, der Adresse und dem Geburtsdatum zu hinterlegen. Der erste Personentyp sind Professoren, welche genau ein Institut leiten. Umgekehrt kann ein Institut auch nur von genau einem Professor geleitet werden.

Professoren halten Vorlesungen. Jeder Professor hält mindestens eine Vorlesung. Der Fall, dass eine Vorlesung von mehreren Professoren gehalten wird, tritt nicht auf. Vorlesungen werden immer in Hörsälen gehalten. In einem Hörsaal können mehrere Vorlesungen gehalten werden, jedoch wird eine Vorlesung immer nur in genau einem Hörsaal abgehalten.

Eine weitere Aufgabe der Professoren ist es, Übungen bereitzustellen. Dies geschieht jedoch auf freiwilliger Basis, so dass nicht jeder Professor Übungen für seine Studenten zur Verfügung stellt. Eine Übung wird immer von genau einem Professor erstellt.

In bestimmten Zeitabständen wird ein Professor zum „Dekan“ gewählt. Der Dekan ist der Leiter einer Fakultät. Ein Professor kann nur höchstens eine Dekanstelle besetzen und eine Fakultät wird immer von genau einem Dekan geleitet.

Eine zweite Personengruppe in der Datenbank sind die Wissenschaftlichen Mitarbeiter (im Folgenden nur noch WiMa genannt). WiMas betreuen immer wieder Übungen, haben aber auch andere Aufgaben. Eine Übung kann nur von einem WiMa betreut werden. Zusätzlich zu den genannten Daten, die allgemein für einen Personentyp gespeichert werden, wird für WiMas und Professoren das jeweilige Gehalt in der Datenbank abgelegt.

Der dritte Personentyp sind Studenten. Diese können an Übungen und Vorlesungen teilnehmen. Damit eine Übung oder Vorlesung stattfindet, muss sich mindestens ein Student dafür eingeschrieben haben. Studenten erhalten, neben den angesprochenen Personenparametern, noch zusätzlich eine Matrikelnummer.

Die an der Universität gehaltenen Vorlesungen und Übungen sind mit einem Thema versehen. Die Übungen unterteilen sich in Rechen- bzw. Laborübungen und werden mit einer Aufgabennummer versehen. Rechenübungen finden in Hörsälen und Laborübungen in Laboren statt. Nicht jeder Hörsaal bzw. jedes Labor ist immer besetzt. Eine Übung findet allerdings immer nur in einem Raum statt. Für jeden Hörsaal und jedes Labor muss die Anzahl der Sitzplätze, die Raumnummer sowie die Gebäudenummer gespeichert werden.

3.5.6 Übungsaufgabe Diensthundeschule

Der Kommandeur der Schule für Diensthundewesen der Bundeswehr (SDstHundeBw) bittet Sie ein Datenmodell zu entwerfen, um eine bereits vorhandene Patientenanwendung zu ersetzen. Patienten im Sinne dieser Anwendung sind Diensthunde. In einer Besprechung wurden folgende Eckpunkte festgelegt:

Als Erstes müssen alle relevanten Dienststellen mit ihrer Bezeichnung, der Dienststellenummer, der Adresse und der Telefonnummer eines Ansprechpartners in der Datenbank hinterlegt werden. Jede Dienststelle untersteht höchstens einer anderen Dienststelle, eine Dienststelle hat keine oder mehrere Dienststellen unter sich.

Zu einem Diensthund sind sein Name, seine Fellfarbe, das Geschlecht und das Kaufdatum wichtig.

Zwischen den Dienststellen und den Diensthunden existieren zwei unterschiedliche Beziehungen. Es gibt Dienststellen, die Eigentümer der Diensthunde sind und andere Dienststellen, die Besitzer der Diensthunde sind. Dies kommt dadurch zu Stande, dass die Hunde nicht immer in der Dienststelle ihren Dienst verrichten, die der Eigentümer des Hundes ist. Jede hier erfasste Dienststelle besitzt bzw. ist Eigentümer von mindestens einem Hund. Ein Diensthund ist jedoch immer nur einer Dienststelle zugeordnet. Dies gilt für den Besitz eines Hundes und auch für das Eigentum an einem Hund.

Die an der Klinik der SDstHundeBw befindlichen Diensthunde haben alle genau einen Status (z. B. dienstfähig, eingeschränkt dienstfähig, usw.). Ein Status kann an mehrere Diensthunde vergeben werden. Des Weiteren bewohnt jeder Diensthund, während seines Aufenthaltes an der SDstHundeBw einen Zwinger. Für das Personal in der Tierklinik ist es wichtig zu wissen, welcher Diensthund wann und wie oft welchen Zwinger bewohnt hat (es soll eine Historie der Zwingeraufenthalte der Hunde möglich sein). Es muss mit eingeplant werden, dass nicht immer alle Zwinger von Hunden bewohnt werden. Ein Zwinger gehört zu genau einer Zwingerart. Beispielsweise gibt es normale Zwinger und Quarantäne Zwinger. Von jeder Zwingerart gibt es mindestens einen Zwinger in der Tierklinik. Zu einem Zwinger werden der Ort, an dem er sich befindet und die Zwingernummer gespeichert. Für die Zwingerart soll lediglich deren Bezeichnung angegeben werden.

Ein Diensthund durchläuft im Laufe seines Lebens verschiedene Untersuchungen in der Tierklinik. Zu jeder Untersuchung ist das Untersuchungsdatum wichtig. Bei jedem Untersuchungstermin wird immer nur genau ein Hund behandelt.

Durchgeführt werden die Untersuchungen von medizinischem Fachpersonal. Eine Untersuchung wird von genau einem Tierarzt durchgeführt, der während seiner Anwesenheit an der SDstHundeBw die verschiedensten Untersuchungen durchführen kann.

Zu jeder Untersuchung wird auch immer mindestens ein anderer Tierarzt oder eine Krankenschwester hinzugezogen. In der Datenbank soll für das medizinische Personal der Name und der Dienstgrad gespeichert werden.

Es gibt vier verschiedene Arten von Untersuchungen, welche unterschieden werden müssen:

- Die Ankaufuntersuchung: Jeder Hund wird vor seinem Ankauf durch die Bundeswehr gründlich untersucht.
- Die Behandlung: Ein Diensthund wird auch im Falle einer ganz normalen Erkrankung, während seiner Anwesenheit an der SDstHundeBw, in der Tierklinik behandelt.
- Die Nachuntersuchung: Nach seinem Ankauf durch die Bw wird ein Hund in seinen ersten zwei Dienstjahren mehrfach nachuntersucht.
- Das Ausmusterungsgutachten: Hat ein Diensthund ein gewisses Alter erreicht oder eine schwere Verletzung erlitten, wird er aus dem Dienst entlassen.

Für die Ankaufuntersuchung sind Angaben wie die Größe und das Gewicht des Hundes sowie sein Röntgenzahnalter wichtig. Zu einer Nachuntersuchung wird nur ein Befund in Textform gespeichert.

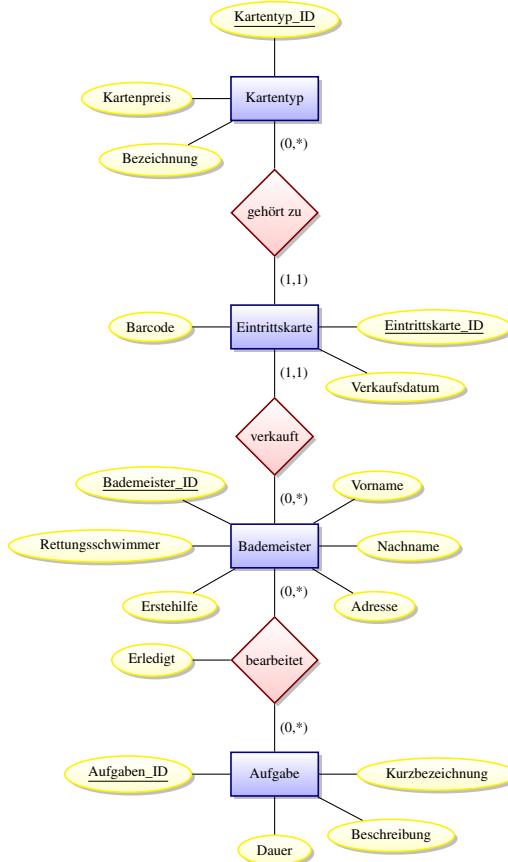
Wird eine „normale“ Behandlung an einem Diensthund durchgeführt, so besteht diese aus mindestens einer oder mehreren Behandlungspositionen (Operation, Medikamentengabe, usw.) die einzeln zu speichern sind. Dabei ist eine erfasste Behandlungsposition immer eindeutig einer Behandlung eines Diensthundes zuordenbar. Der behandelnde Arzt muss auch die Möglichkeit besitzen, zu einer Behandlungsposition eine kurze Notiz zu schreiben. Neben dieser Option ist zu jeder Behandlungsposition eine entsprechende Diagnose zu vermerken. Diese werden jedoch in einer separaten Liste verwaltet. So kann es auch vorkommen, dass dort Diagnosen aufgelistet sind, die bisher noch bei keinem Diensthund festgestellt wurden.

Für ein Ausmusterungsgutachten muss der Arzt einen kompletten Bericht im System hinterlegen können.

3.6 Lösungen - Erweiterte ER-Modellierung

3.6.1 Übungsaufgabe Schwimmbad

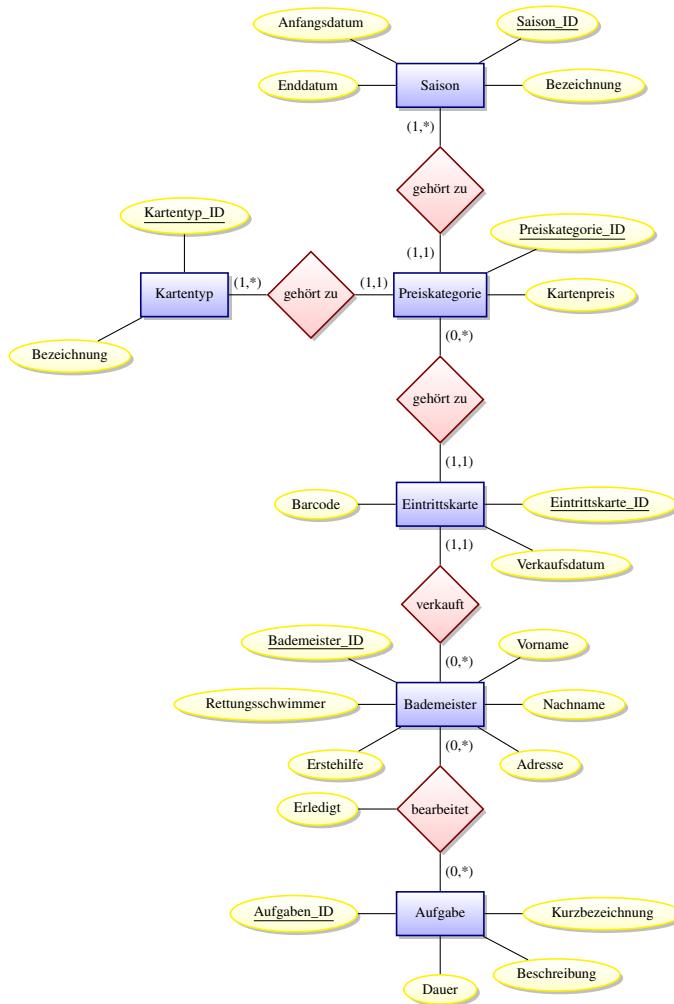
Vorgaben



Transformation

Kartentyp	(<u>Kartentyp_ID</u> , Kartenpreis, Bezeichnung)
Eintrittskarte	(<u>Eintrittskarte_ID</u> , Barcode, Verkaufsdatum, ↑ <u>Kartentyp_ID</u> ↑ [NN], ↑ <u>Bademeister_ID</u> ↑ [NN])
Bademeister	(<u>Bademeister_ID</u> , Vorname, Nachname, Adresse, Rettungsschwimmer, Erstehilfe)
Aufgaben	(<u>Aufgaben_ID</u> , Kurzbezeichnung, Beschreibung, Dauer)
Arbeitsplan	(↑ <u>Bademeister_ID</u> + <u>Aufgaben_ID</u> ↑, Erledigt)

Zusatzaufgabe 1



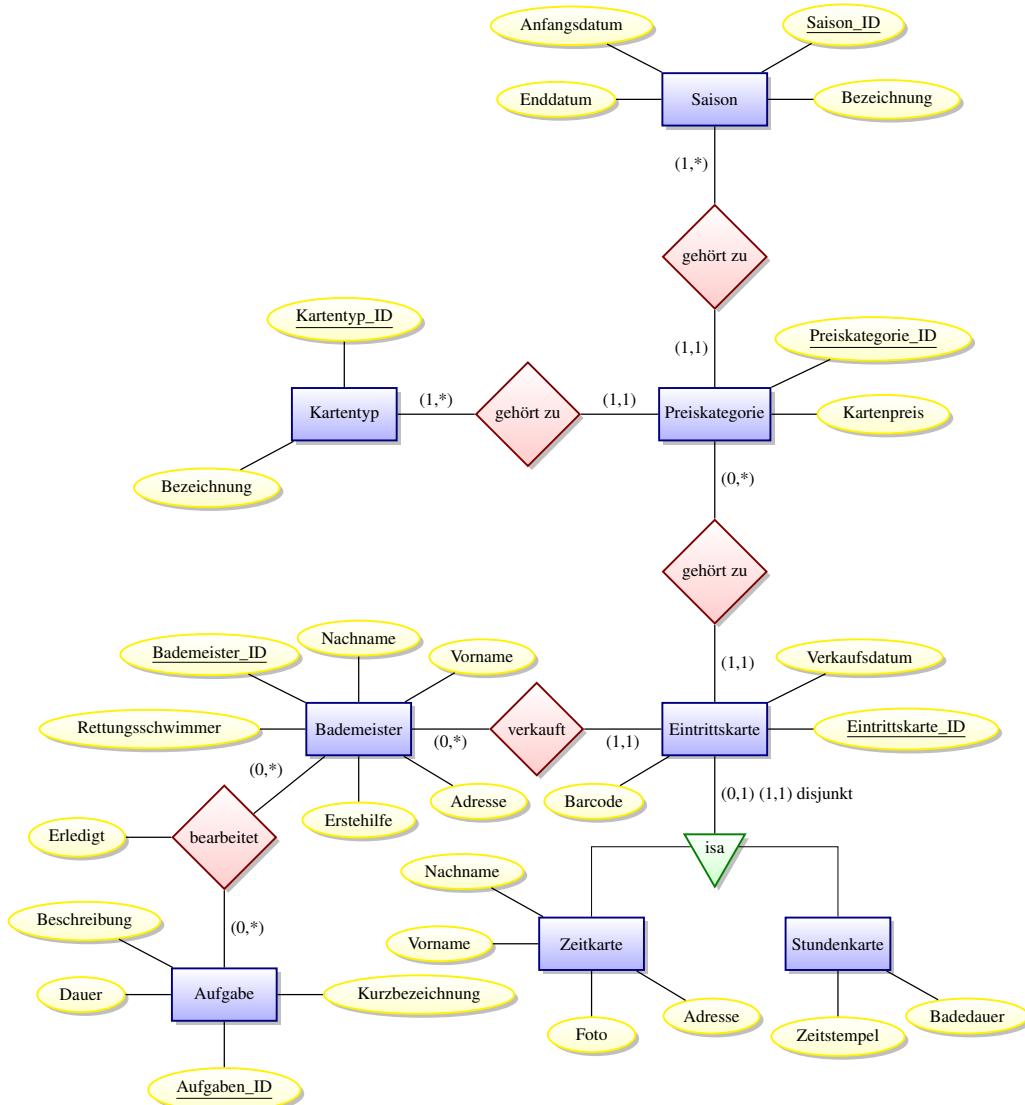
Transformation Zusatzaufgabe 1

In dieser Transformation werden nur die Änderungen zur ursprünglichen Aufgabe gezeigt!

Kartentyp
Preiskategorie
Eintrittskarte
Saison

(Kartentyp_ID)
(Preiskategorie_ID)
(Eintrittskarte_ID)
 \uparrow Bademeister_ID
(Saison_ID, ...)

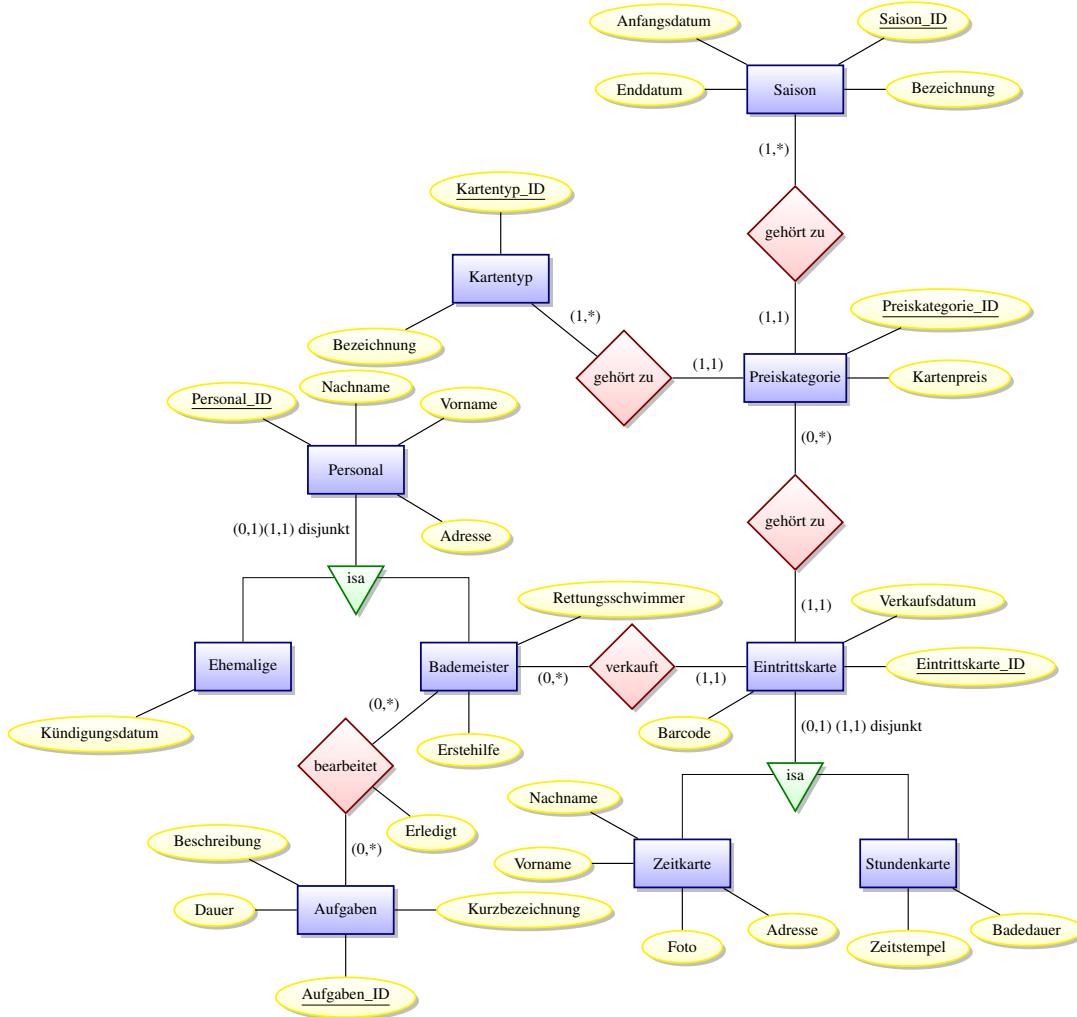
Zusatzaufgabe 2



Transformation Zusatzaufgabe 2

Eintrittskarte	(Eintrittskarte_ID, Barcode, Verkaufsdatum, ↑Preiskategorie_ID↑ [NN], ↑Bademeister_ID↑ [NN])
Zeitkarte	(↑Eintrittskarte_ID↑, Vorname, Nachname, Adresse, Foto)
Stundenkarte	(↑Eintrittskarte_ID↑, Zeitstempel)

Zusatzaufgabe 3



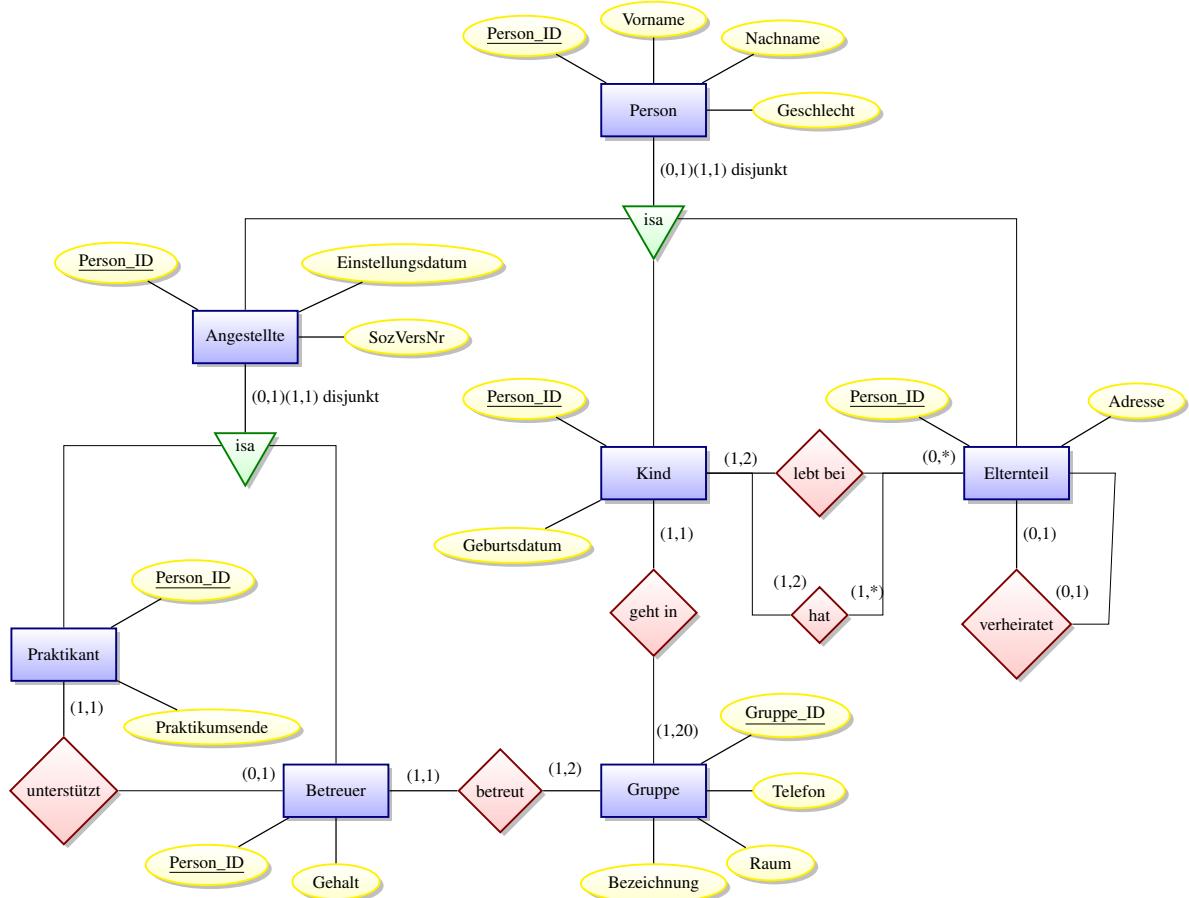
Aus Platzgründen wurde bei den beiden Entitäten „Ehemalige“ und „Bademeister“ das Attribut „Personal_ID“ weggelassen!

Transformation Zusatzaufgabe 3

Personal	(Personal_ID, Vorname, Nachname, Adresse)
Bademeister	(↑Personal_ID↑, Rettungsschwimmer, Erstehilfe)
Ehemalige	(↑Personal_ID↑, Kündigungsdatum)

3.6.2 Übungsaufgabe Kindergarten

Vorgaben

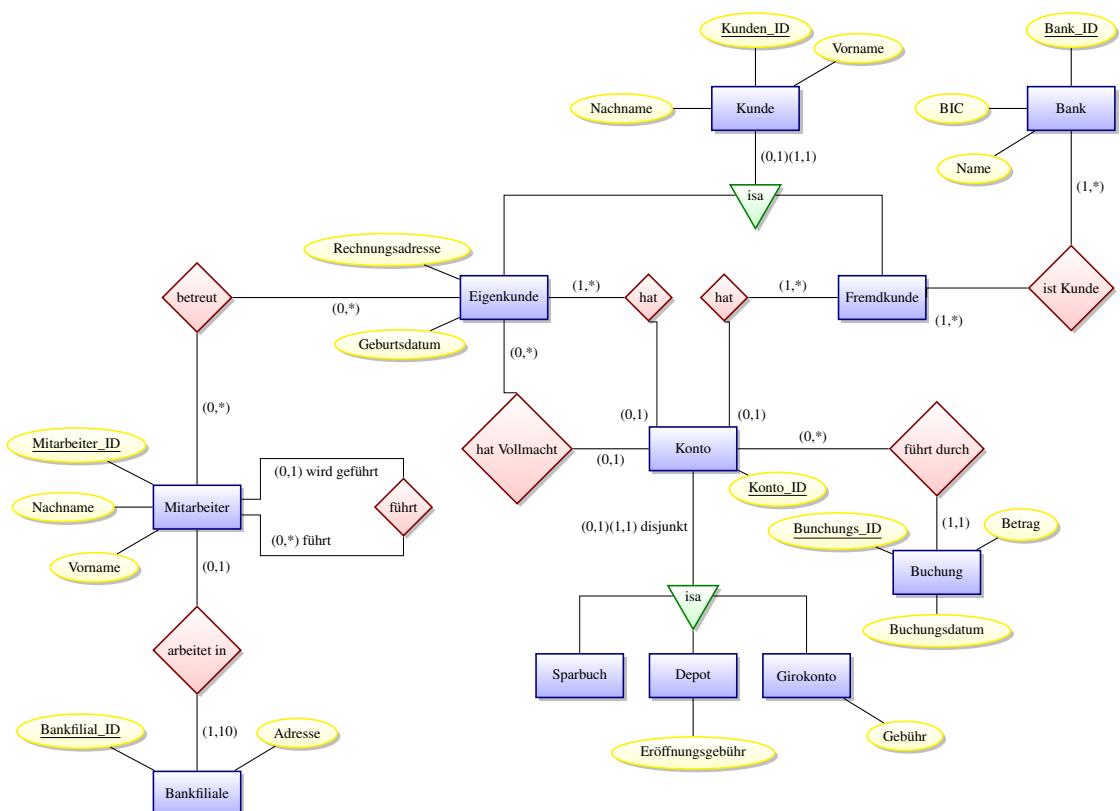


Transformation

Person	(<u>Person_ID</u> , Vorname, Nachname, Geschlecht)
Elternteil	(↑ <u>Person_ID</u> ↑, Adresse, ↑Ehepartner_ID↑ [UN])
Kind	(↑ <u>Person_ID</u> ↑, Geburtsdatum, ↑Gruppe_ID↑ [NN])
Angestellte	(↑ <u>Person_ID</u> ↑, Einstellungsdatum, SozVersNr)
Betreuer	(↑ <u>Person_ID</u> ↑, Gehalt, ↑Gruppe_ID↑ [NN])
Praktikant	(↑ <u>Person_ID</u> ↑, Praktikumsende, ↑Betreuer_ID↑ [UN] [NN])
Gruppe	(<u>Gruppe_ID</u> , Bezeichnung, Telefon, Raum)
Kinderwohnung	(↑ <u>Kind_ID</u> + Erwachsener_ID↑)
Familie	(↑ <u>Kind_ID</u> + Erwachsener_ID↑)

3.6.3 Übungsaufgabe Bank

Vorgaben



Aus Platzgründen sind nicht alle Attribute im Modell aufgezeigt.

Transformation Variante 1

Bank	(<u>Bank_ID</u> , BIC, Name)
Bankfiliale	(<u>Bankfiliale_ID</u> , Strasse, HsNr, PLZ, Ort)
BankFremdkunde	(↑ <u>Bank_ID</u> + <u>Kunden_ID</u> ↑)
Buchung	(<u>Buchung_ID</u> , Betrag, Buchungsdatum, ↑ <u>Konto_ID</u> ↑ [NN])
Depot	(↑ <u>Konto_ID</u> ↑, Eroeffnungsgebuehr)
Eigenkunde	(↑ <u>Kunden_ID</u> ↑, Geburtsdatum, Rechnungsadresse)
EigenkundeKonto	(↑ <u>Kunden_ID</u> + <u>Konto_ID</u> ↑)
Fremdkunde	(↑ <u>Kunden_ID</u> ↑)
FremdkundeKonto	(↑ <u>Kunden_ID</u> + <u>Konto_ID</u> ↑)
Girokonto	(↑ <u>Konto_ID</u> ↑, Guthaben, Sollzins, Kontofuehrungsgebuehr)
Konto	(<u>Konto_ID</u> , IBAN, ↑ <u>Bevollmaechtigter_ID</u> ↑)
Kunde	(<u>Kunden_ID</u> , Vorname, Nachname)
Mitarbeiter	(<u>Mitarbeiter_ID</u> , Vorname, Nachname, ↑ <u>Bankfiliale_ID</u> ↑, ↑ <u>Vorgesetzter_ID</u> ↑)
MitarbeiterEigenkunde	(↑ <u>Mitarbeiter_ID</u> + <u>Kunde_ID</u> ↑)
Sparbuch	(↑ <u>Konto_ID</u> ↑, Guthaben, Habenzins)

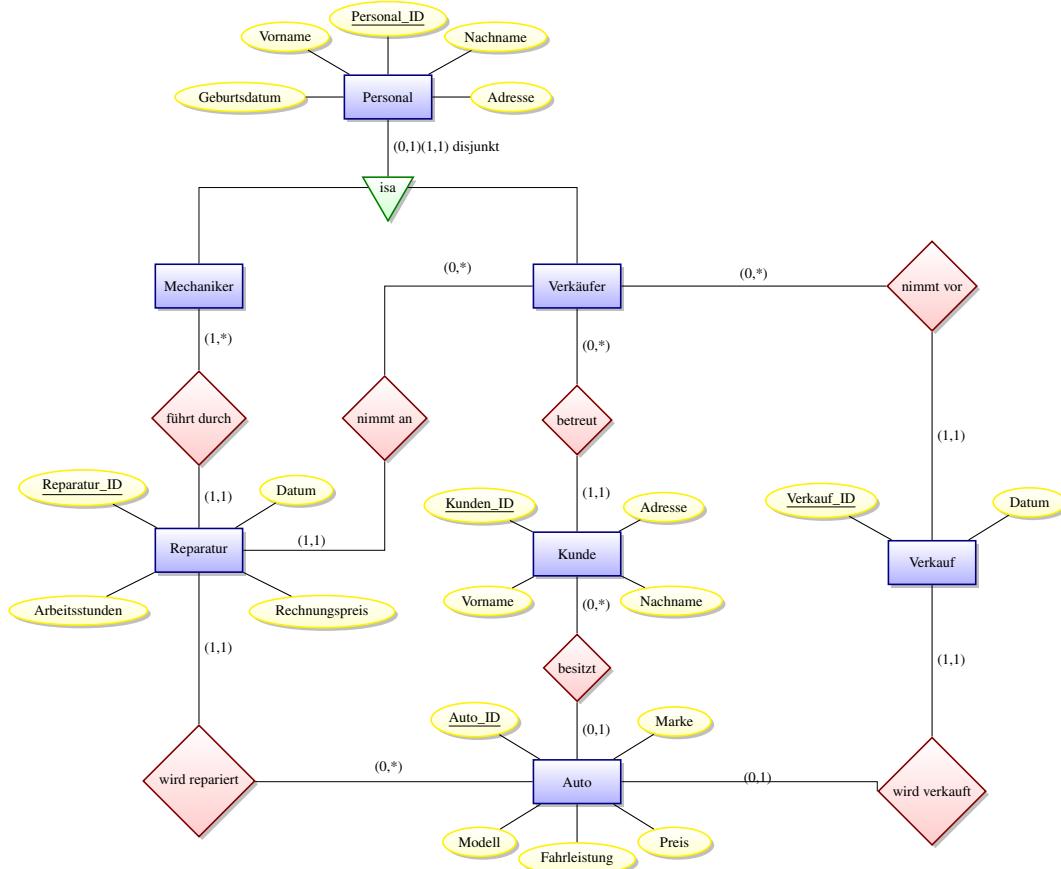
Transformation Variante 2

Anstatt zwei Hilfstabellen für die Beziehungen zwischen Konto und den beiden Kundenarten zu erstellen, können auch die Primärschlüssel von Eigenkunde und Fremdkunde als jeweils eigene Fremdschlüssel entsprechend der Transformationsregel T07 in die Relation Konto eingetragen werden. Dadurch fallen die Relationen EigenkundeKonto und FremdkundeKonto weg und es bleibt nur noch die Relation Konto mit zwei neuen Attributten:

Konto (Konto_ID, IBAN, ↑Bevollmaechtigter_ID↑, ↑Eigenkunde_ID↑, ↑Fremdkunde_ID↑)

3.6.4 Übungsaufgabe Autohändler

Vorgaben

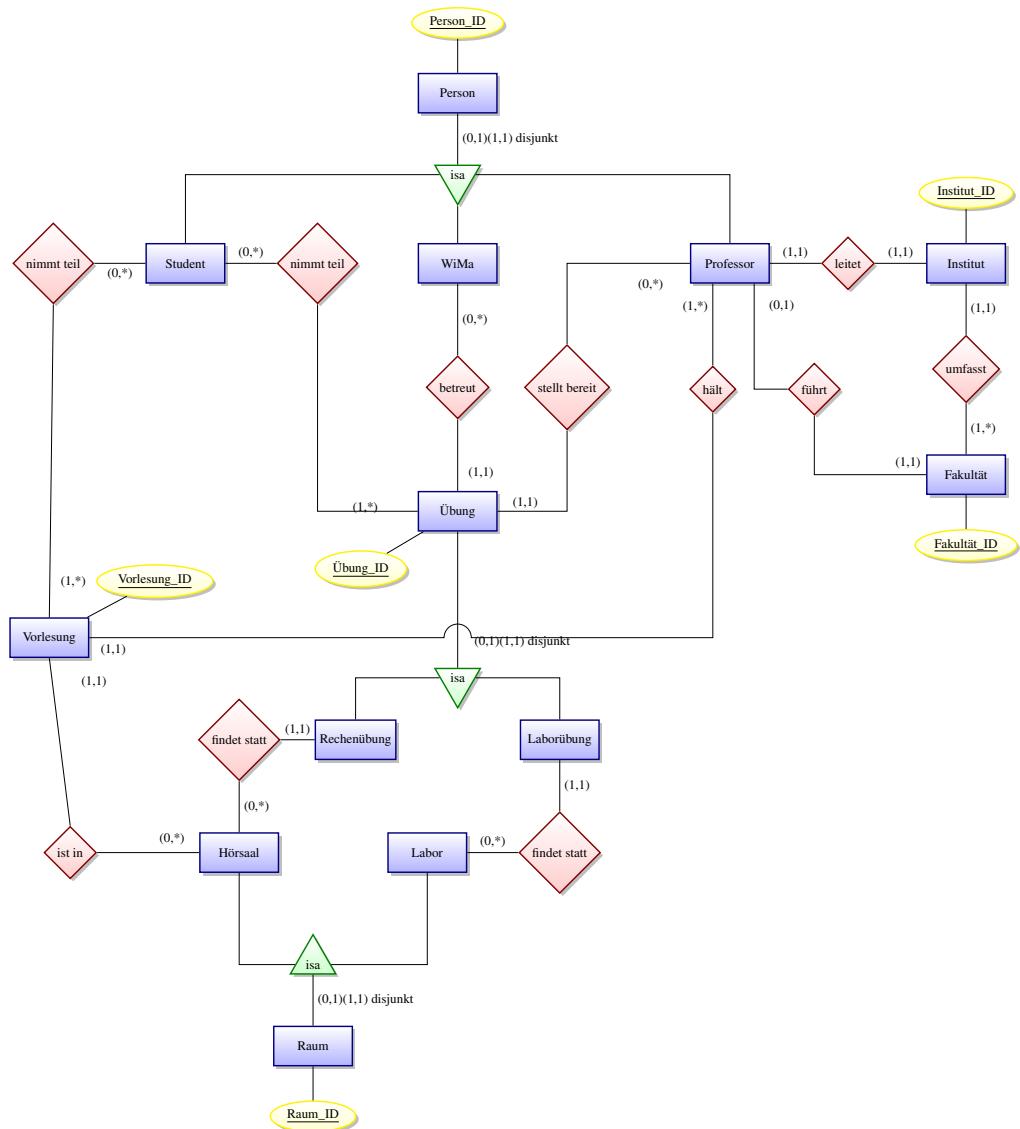


Transformation

Personal	(<u>Personal_ID</u> , Vorname, Nachname, Geburtsdatum, Adresse)
Mechaniker	(↑ <u>Personal_ID</u> ↑)
Verkäufer	(↑ <u>Personal_ID</u> ↑)
Kunde	(<u>Kunden_ID</u> , Vorname, Nachname, Adresse ↑ <u>Verkäufer_ID</u> ↑ [NN])
Auto	(<u>Auto_ID</u> , Marke, Modell, Preis, Fahrleistung, ↑ <u>Kunden_ID</u> ↑)
Verkauf	(<u>Verkauf_ID</u> , Datum, ↑ <u>Verkäufer_ID</u> ↑ [NN], ↑ <u>Auto_ID</u> ↑ [NN] [UN])
Reparatur	(<u>Reparatur_ID</u> , Arbeitsstunden, Datum, Rechnungspreis, ↑ <u>Verkäufer_ID</u> ↑ [NN], ↑ <u>Mechaniker_ID</u> ↑ [NN], ↑ <u>Auto_ID</u> ↑ [NN])

3.6.5 Übungsaufgabe Universität

Vorgaben



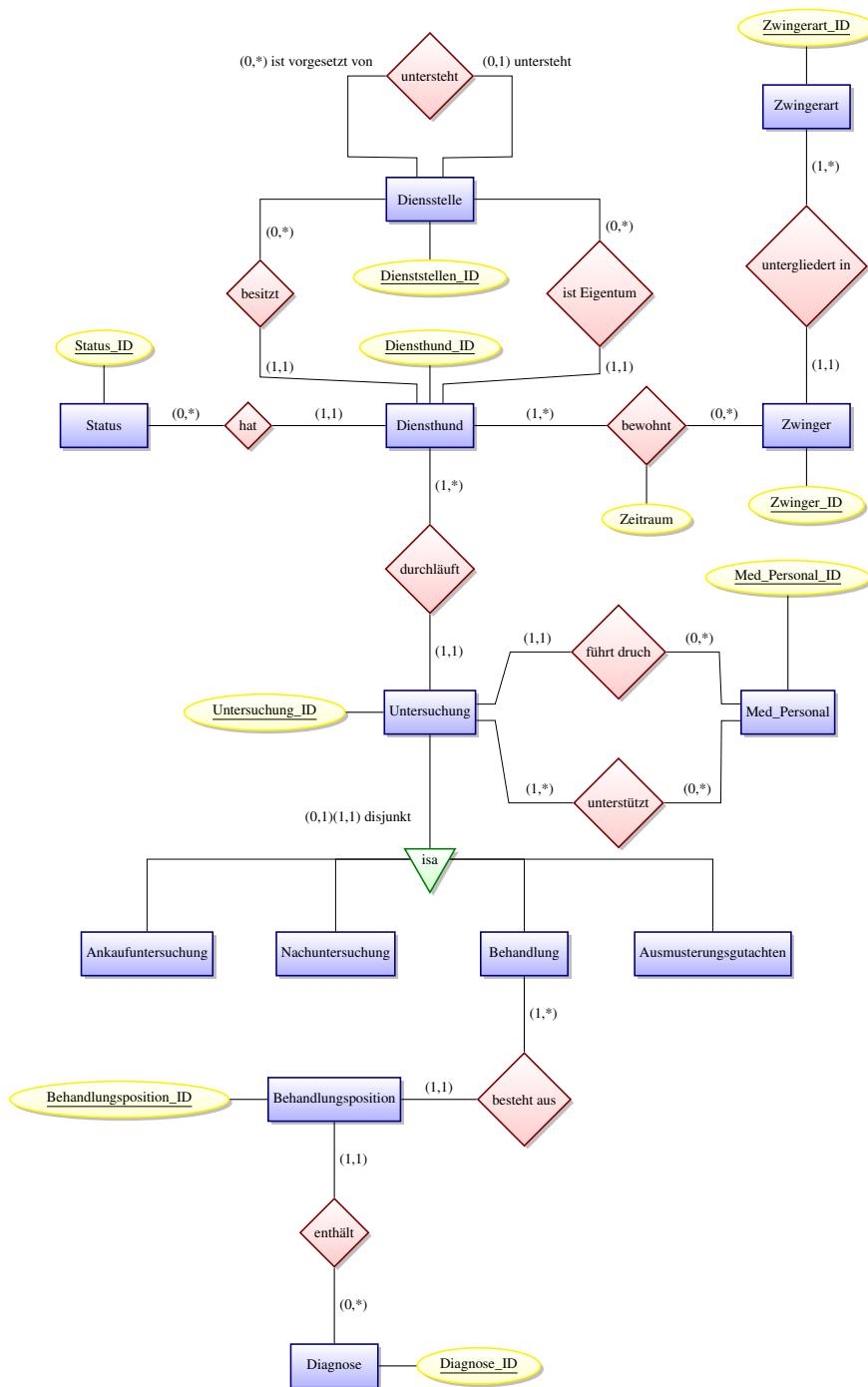
Aus Platzgründen sind nur die Primärschlüssel im Modell eingezeichnet.

Transformation

Person	(<u>Person_ID</u> , Vorname, Nachname, Adresse, Geburtsdatum)
Student	(↑ <u>Person_ID</u> ↑, Matrikelnummer)
WiMa	(↑ <u>Person_ID</u> ↑, Gehalt)
Professor	(↑ <u>Person_ID</u> ↑, Gehalt, ↑Institut_ID↑ [NN] [UN])
Institut	(<u>Institut_ID</u> , Bezeichnung, ↑Fakultaet_ID↑ [NN], ↑Professor_ID↑ [NN] [UN])
Fakultaet	(<u>Fakultaet_ID</u> , Bezeichnung, ↑Dekan_ID↑ [NN] [UN])
Raum	(<u>Raum_ID</u> , Sitzplaetze, Raumnummer, Gebaeudenummer)
Hoersaal	(↑ <u>Raum_ID</u> ↑)
Labor	(↑ <u>Raum_ID</u> ↑)
Uebung	(<u>Uebung_ID</u> , Thema, Aufgabennummer, ↑WiMa_ID↑ [NN], ↑Professor_ID↑ [NN])
Rechenuerbung	(<u>Uebung_ID</u> , ↑Raum_ID↑ [NN])
Laboruebung	(<u>Uebung_ID</u> , ↑Raum_ID↑ [NN])
StudentUebung	(↑ <u>Person_ID</u> + <u>Uebung_ID</u> ↑)
Vorlesung	(<u>Vorlesung_ID</u> , Thema, ↑Raum_ID↑ [NN], ↑Professor_ID↑ [NN])
StudentVorlesung	(↑ <u>Person_ID</u> + <u>Vorlesung_ID</u> ↑)

3.6.6 Übungsaufgabe Diensthundeschule

Vorgaben



Transformation

Dienststelle	(<u>Dienststellen_ID</u> , Bezeichnung, Dienststellennummer, Adresse, Telefonnr, ↑VorgesetzteDienststelle_ID↑)
Diensthund	(<u>Diensthund_ID</u> , Name, Fellfarbe, Geschlecht, Kaufdatum, ↑Besitzer_ID↑ [NN], ↑Eigentuemer_ID↑ [NN], ↑Status_ID↑ [NN])
Zwinger	(<u>Zwinger_ID</u> , Ort, Zwingernummer, ↑Zwingerart_ID↑ [NN])
Diensthundezwinger	(<u>DH_Zwinger_ID</u> , ↑Diensthund_ID↑ [NN], ↑Zwinger_ID↑ [NN], Zeitraum)
Zwingerart	(<u>Zwingerart_ID</u> , Bezeichnung)
Status	(<u>Status_ID</u> , Bezeichnung)
Untersuchung	(<u>Untersuchung_ID</u> , Datum, ↑Diensthund_ID↑ [NN], ↑MedPersonal_ID↑ [NN])
MedPersonal	(<u>Med_Personal_ID</u> , Name, Dienstgrad)
Untersuchungspersonal	(↑ <u>Untersuchung_ID</u> + <u>Med_Personal_ID</u> ↑)
Ankaufuntersuchung	(↑ <u>Untersuchung_ID</u> ↑, Groesse, Gewicht, Roentgenzahnalter)
Nachuntersuchung	(↑ <u>Untersuchung_ID</u> ↑, Befund)
Behandlung	(↑ <u>Untersuchung_ID</u> ↑)
Ausmusterungsgutachten	(↑ <u>Untersuchung_ID</u> ↑, Bericht)
Behandlungsposition	(<u>Behandlungsposition_ID</u> , Notiz, ↑ <u>Untersuchung_ID</u> ↑ [NN], ↑Diagnose_ID↑ [NN])
Diagnose	(<u>Diagnose_ID</u> , Diagnosetext)

4 Transformation

Inhaltsangabe

Bei der Beschreibung der Realität, mit Hilfe eines Entity-Relationship-Modells, bestand die Zielrichtung darin, Objekttypen und Beziehungstypen unabhängig vom später einzusetzenden Datenbankmanagementsystem und somit auch unabhängig von einem speziellen Datenbankmodell zu beschreiben. Nun muss der Versuch unternommen werden, das durch die Modellierung entstandene Datenmodell möglichst ohne semantische¹ Einbuß en mit den Strukturierungsmitteln der verfügbaren Datenbankmanagementsysteme wiederzugeben. Dieser Vorgang wird als Transformation bezeichnet. Es ist die Umsetzung des konzeptuellen Datenmodells, welches nur die möglichen Beziehungen zwischen den Objekttypen beschreibt, in das physische Datenmodell. Das physische Datenmodell berücksichtigt, wie die Beziehungen zwischen den Objekttypen auf Datenbankebene effektiv umgesetzt werden. Das physische Datenmodell wird in verschiedenen Quellen auch als relationales Modell bezeichnet.

Eine komplette Gleichsetzung ist je nach Interpretation nicht möglich, hier soll jedoch nicht weiter unterschieden werden.

Im Vorfeld sind diese beiden Fragen zu beantworten.

1. Welches Datenbankmodell soll der zu erstellenden Datenbank zugrunde liegen?
2. Welche Möglichkeiten bietet das zu verwendende Datenbankmanagementsystem für die Gestaltung der Datenstrukturen?

Hinsichtlich der ersten Frage hat heutzutage das relationale Datenbankmodell die meiste Relevanz. Was die zweite Fragestellung anbetrifft, so kommen hier nur die beiden Datenbankmanagementsysteme *ORACLE* und *Microsoft SQL Server* zum Einsatz. Bei beiden Systemen handelt es sich um relationale Datenbankmanagementsysteme (RDBMS), welche ihre Vor- und Nachteile besitzen, auf die im Unterricht kurz eingegangen werden soll.

Dieses Kapitel schafft einheitliche Begriffe und stellt Regeln für die Transformation eines ER-Modells in ein relationales Modell auf.

¹Semantik = Bedeutungslehre, bezeichnen, anzeigen

4.1 Grundlagen

4.1.1 Begriffsdefinitionen

Im vorangegangenen Kapitel wurde der Begriff „Schlüssel“ erläutert, der an dieser Stelle aber noch weiter differenziert werden muss.

Identifikationsschlüssel (ID-Schlüssel)

Jedes Objekt einer Objektmenge muss eindeutig identifizierbar sein. Dies kann durch eine Eigenschaft/Attribut oder eine Kombination von Eigenschaften/Attributen gewährleistet werden. Beispielsweise ist ein Soldat der Bundeswehr eindeutig durch die Personalnummer identifizierbar. Der Name einer Person kann **kein** Identifikationsschlüssel sein, weil es sehr wahrscheinlich ist, dass es mehrere Personen mit dem gleichen Namen gibt (z. B. mehrere Meier).

Der Identifikationsschlüssel muss folgende Kriterien erfüllen:

- Jedes Objekt muss eindeutig identifizierbar sein. Es dürfen nicht mehrere Objekte einen ID-Schlüssel mit dem gleichen Wert aufweisen.
- Jedem neuen Objekt muss augenblicklich ein Identifikationsschlüssel zugeteilt werden können, da sonst keine Speicherung des Objektes erfolgen darf.
- Der ID-Schlüsselwert eines Objektes darf sich während dessen Existenz nicht verändern.

Primärschlüssel

Der Primärschlüssel wird häufig mit dem Begriff „Identifikationsschlüssel“ gleichgesetzt. Diese beiden Begriffe sind aber nicht gleichbedeutend. Der Primärschlüssel (PK - primary key) wird direkt in die Speicherorganisation einbezogen und ist somit dem physischen Datenmodell zugeordnet. Der ID-Schlüssel hingegen ist dem konzeptionellen Datenmodell zugeordnet. Ansonsten gelten für die Eigenschaftswerte eines PK dieselben Bedingungen, wie beim ID-Schlüssel beschrieben. Jeder transformierte Objekttyp kann nur einen PK haben. Da jedes Objekt eindeutig über den PK identifiziert werden soll, ergibt sich automatisch die Bedingung, dass der Wert des PK einmalig (unikal) und nicht NULL (leer) sein muss. Daneben kann es aber auch weitere Eigenschaften mit eindeutigen Werten geben, die nicht zum PK gehören.

Fremdschlüssel

Ein Fremdschlüssel (FK = Foreign Key) ist ein Attribut, welches zur Verknüpfung zweier Tabellen dient. Ein Fremdschlüsselattribut wird in eine Tabelle eingefügt, welche sich als untergeordnete Tabelle auf eine andere bezieht. Das Fremdschlüsselattribut bezieht sich dabei immer auf den Primärschlüssel oder aber auf ein anderes, eindeutiges Attribut der übergeordneten Tabelle. Genauso wie der Primärschlüssel kann auch der Fremdschlüssel aus einer Kombination von Attributen bestehen. Im Gegensatz zum PK kann der FK NULL-Werte beinhalten.

NULL bzw. NOT NULL

Ein Eigenschaftswert kann in einer Datenbank verschiedene Einschränkungen haben, um die Konsistenz der Daten zu gewährleisten. Eine dieser Einschränkungen (engl. constraint) ist das NOT NULL constraint, welches im SQL Teil näher erläutert wird. Es wird damit festgelegt, ob der Eigenschaftswert beim Anlegen eines Datensatzes vorhanden sein muss (NOT NULL, Abk. [NN]) oder ob er leer sein darf (NULL, Standard). Damit kann in der Datenbank die Forderung nach der Eigenschaft „eingabepflichtig“ (Teil der Transformation) realisiert werden.

UNIQUE

Bei dem Begriff UNIQUE (Abk. [UN]) handelt es sich ebenfalls um ein constraint (Erläuterungen im SQL Teil). Mit diesem constraint kann die Forderung nach der „Unikalität“, die sich durch die Transformation ergibt, erfüllt werden. Das UNIQUE (einmalig) constraint sorgt dafür, dass die Eigenschaftswerte eines Objekttyps nicht doppelt vorkommen. Im RDBMS Oracle bildet hier der NULL-Wert eine Ausnahme, d.h. es können mehrere NULL Werte innerhalb einer Spalte vorkommen. In Microsoft SQL Server ist dies nicht der Fall.

4.1.2 Kurzschreibweise der Tabellen

Den Aufbau einer Tabelle kann man mit folgender Kurzschreibweise darstellen:

Tabellenname(ID-Schlüssel, Attribut₁, Attribut₂, Attribut₃, ..., Attribut_n)

Falls der PK aus zusammengesetzten Attributen besteht, werden alle erforderlichen Attribute unterstrichen:

Tabellenname(Teil-ID-Schlüssel₁, Teil-ID-Schlüssel₂, Attribut₁, Attribut₂,
Attribut₃, ..., Attribut_n)

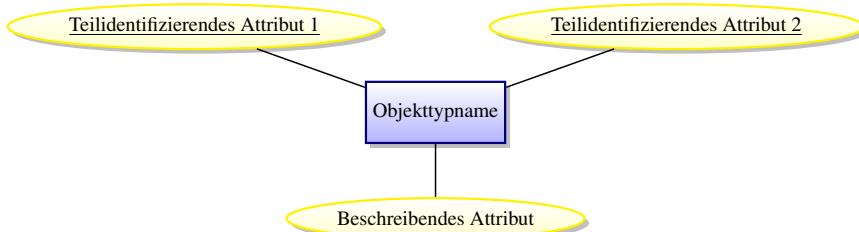
4.2 Transformation von Objekttypen

Die Objekttypen des ER-Modells stellen das Haupt-Ordnungsprinzip der Datenmodellierung dar. Sie beschreiben die Klassen-Struktur, in die die speicherrelevanten Objekte der abzubildenden Realität eingruppiert werden. Im physischen Datenbankmodell wird diese Klassen-Struktur durch einen Satz von Tabellen wiedergegeben.

Für jeden Objekttyp des ER-Modells wird eine Tabelle vereinbart. Dabei gilt die Transformationsregel T01.

Transformationsregel T01 (Objekttyp)

konzeptionelles Datenmodell	physisches Datenmodell
Objekttypname	\Rightarrow Tabellen-Bezeichnung
(Teil-)Identifizierende Eigenschaft	\Rightarrow Primärschlüssel-Attribut
Beschreibende Eigenschaft	\Rightarrow Spalten-Bezeichnung



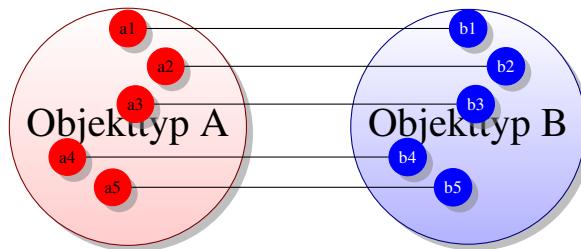
4.3 Transformation binärer Beziehungstypen

In vorangegangenen Abschnitten wurde festgestellt, dass sich Beziehungstypen im physischen Datenbankmodell nur dadurch repräsentieren lassen, dass der Primärschlüssel (PK) einer Tabelle „gedoppelt“ und als Fremdschlüssel an „anderer Stelle“ aufgenommen wird. Handelt es sich bei dieser „anderen Stelle“ um eine andere Tabelle, wird ein binärer Beziehungstyp dargestellt, der den sachlogischen Zusammenhang zwischen zwei Objekten aus verschiedenen Objekttypen beschreibt.

Liegt diese „andere Stelle“ dagegen in derselben Tabelle, aus der der PK stammt, wird ein rekursiver Beziehungstyp repräsentiert. Hier wird der sachlogische Zusammenhang zwischen zwei Objekten widergespiegelt, die demselben Objekttyp angehören. Nachfolgend werden die zehn möglichen binären und die sieben rekursiven Beziehungstypen in (Min,Max)-Notation erläutert.

4.3.1 Der (1,1):(1,1) Beziehungstyp

Beim (1,1):(1,1) Beziehungstyp ist jedes Objekt des Objekttyps A mit genau einem Objekt des Objekttyps B verbunden und umgekehrt. Je ein A-Objekt und ein B-Objekt gehen eine feste Paarung ein, wie Abbildung ?? zeigt.



(1,1):(1,1) Beziehungstypen sind bereits im ER-Modell kritisch zu betrachten, weil sie meist überflüssig sind. Ist nämlich jedes Objekt-A mit genau einem Objekt-B - und umgekehrt - verbunden, dann bildet sich eine derart feste Kopplung, dass sie als ein einziges, komplexes Objekt betrachtet werden können. Die Umsetzung erfolgt, in dem alle Attribute von B in die Tabelle A eingefügt werden. Die Kopplung wird dadurch erzwungen, dass der Schlüssel von B in der Tabelle A als eingabepflichtig (NOT NULL, NN) und als unikal (UNIQUE, UN) deklariert wird. Ein B-Objekt kann nun nicht losgelöst von „seinem“ Objekt A gespeichert werden.

Beispiel (1,1):(1,1) Beziehungstyp

Betrachten wir zunächst einen „überflüssigen“ (1,1):(1,1) Beziehungstyp: Mitarbeiter eines Unternehmens, von denen jeder genau einen Dienstausweis besitzt. Ein gegebener Dienstausweis ist natürlich für genau einen Mitarbeiter ausgestellt.

Da das Attribut „Ausweisnummer“ eingabepflichtig ist, muss jeder gespeicherte Mitarbeiter einen Ausweis haben. Andererseits ist das Attribut unikal, so dass eine Ausweisnummer nur einem Mitarbeiter zugeordnet sein kann.

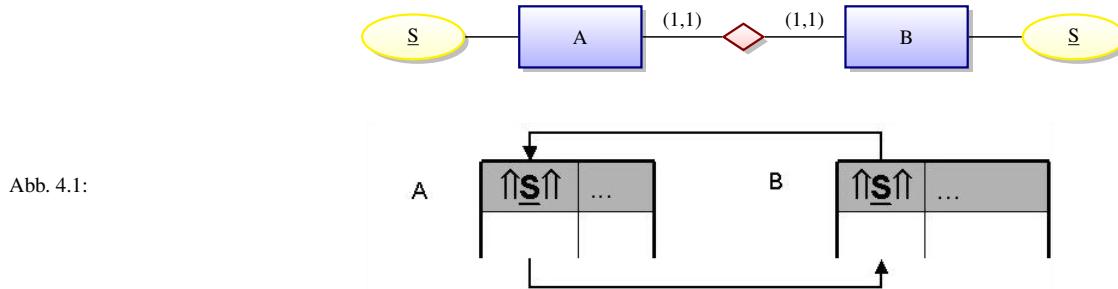
Es gibt aber auch Fälle, bei denen ein (1,1):(1,1) Beziehungstyp durchaus sinnvoll ist. Will man beispielsweise die Informationen über Objekte in öffentliche (z.B. Mitarbeiter-Offen(Personalnummer, Name, TelNr, Abteilung)) und vertrauliche Daten (z.B. MitarbeiterVS(Personalnummer, Gehalt, Konfession)) unterteilen, kann man zwei Objekttypen A und B mit demselben Schlüssel S ins Datenmodell aufnehmen. Dann wird sowohl der Schlüssel S von A in B als unikaler eingabepflichtiger Fremdschlüssel vereinbart und umgekehrt, der Schlüssel S von B wird in A als

unikaler eingabepflichtiger FK deklariert. Die folgende Tabelle zeigt die entsprechende Transformationsregel T02.

Transformationsregel T02 für sinnvolle (1,1):(1,1) Beziehungen

konzeptionelles Datenmodell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>S</u>	\Rightarrow	Tabelle A mit PK <u>S</u>
Objekttyp B mit Schlüssel <u>S</u>	\Rightarrow	Tabelle B mit PK <u>S</u>
(1,1):(1,1) Beziehungstyp	\Rightarrow	<u>S</u> wird sowohl in A als auch in B als unikaler [UN], eingabepflichtiger [NN] Fremdschlüssel vereinbart ($\uparrow\!S\!\uparrow$)
Alternative bei unterschiedlichen Schlüsseln:		
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
Objekttyp B mit Schlüssel <u>SB</u>	\Rightarrow	Tabelle B mit PK <u>SB</u>
(1,1):(1,1) Beziehungstyp	\Rightarrow	Es wird <u>SA</u> in B und <u>SB</u> in A als unikaler [UN], eingabepflichtiger [NN] Fremdschlüssel eingefügt ($\uparrow\!SA\!\uparrow$ und $\uparrow\!SB\!\uparrow$)

Ein Fremdschlüssel wird in die Verweispfeile eingeschlossen - $\uparrow\!xyz\!\uparrow$. Sollte der Fremdschlüssel seinerseits wieder Fremdschlüssel enthalten, werden für die inneren Fremdschlüssel die Verweispfeile weggelassen. Ist der Fremdschlüssel in der Tabelle für sich wiederum ein Primärschlüssel, so wird der FK unterstrichen und fett gedruckt - $\uparrow\!\underline{xyz}\!\uparrow$.



Bei diesem Beziehungstyp kann ein neues Objekt weder allein in A noch allein in B gespeichert werden. Das würde der Forderung nach Nichtoptionalität beider Beziehungstyprichtungen (siehe den jeweiligen Min-Wert) widersprechen. Deswegen muss vom Anwendungsprogramm im Rahmen einer Transaktion erreicht werden, dass zu einem neuen Schlüsselwert je eine Zeile in die Tabellen A und B eingetragen wird.

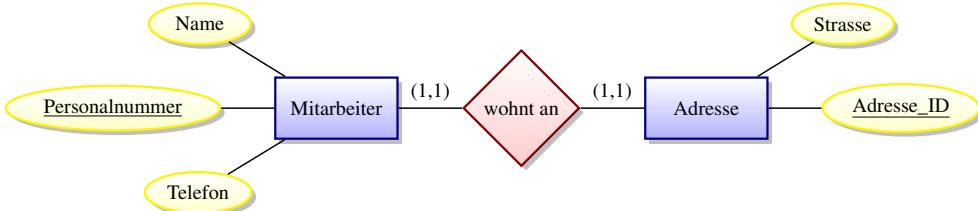


Eine Transaktion ist eine Folge von Operationen, bei der sichergestellt wird, dass entweder alle Operationen fehlerfrei beendet werden oder das keine der Operationen ausgeführt wird.

Beispiel - Mitarbeiteradresse

Werden in einer Datenbank Adressen für mehrere Objekttypen gespeichert, z. B. Mitarbeiter und Kunden, kann es sinnvoll sein, einen eigenen Objekttyp „Adresse“ zu erstellen.

Da die Fremdschlüssel jeweils auf die andere Tabelle verweisen, muss es nach den Regeln der referentiellen Integrität zu jedem FK genau einen Datensatz in der anderen Tabelle geben.

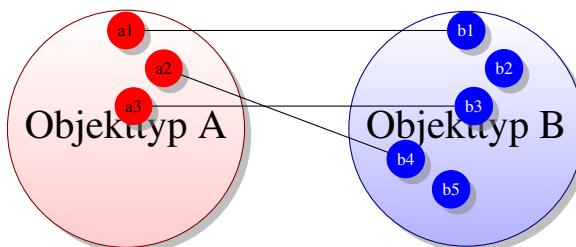


Mitarbeiter(Personalnummer, Name, Telefon, ↑Adresse_ID↑ [NN] [UN])

Adresse(Adresse_ID, Strasse, Hausnummer, PLZ, Ort, ↑Personalnummer↑ [NN] [UN])

4.3.2 Der (1,1):(0,1) Beziehungstyp

Beim (1,1):(0,1) Beziehungstyp ist jedes Objekt des Objekttyps A mit genau einem Objekt des Objekttyps B verbunden. Ein Objekt aus B kann aber nur mit höchstens einem Objekt aus A gekoppelt sein. Die Information(en) im A-Objekt können als fakultative² ergänzende Angaben zum B-Objekt interpretiert werden. Abbildung ?? verdeutlicht dies.



Die Art der Transformation richtet sich nun danach, wie hoch der Anteil jener B-Objekte ist, für die ergänzende Angaben gemacht werden, die also in Beziehung zu einem A-Objekt stehen. Oder anders gefragt, ob der Objekttyp B wesentlich mehr Objekte enthält als der Objekttyp A.

²wahlfreie, beliebige, optionale

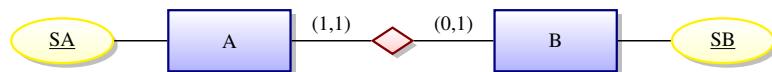
Fall 1: $\#A$ unwesentlich kleiner als $\#B$

Gibt es bei einem (1,1):(0,1) Beziehungstyp nur unwesentlich mehr B-Objekte als A-Objekte, können die Daten beider Objekttypen in einer Tabelle zusammengefasst werden. Die Eigenschaften von A werden dabei als nicht-eingabepflichtig deklariert.

Der Schlüssel SA von A wird in der Tabelle B jedoch als unikal deklariert. Ein A-Objekt, das damit nur zu einem einzigen B-Objekt gehören kann, kann nicht losgelöst von „seinem“ B-Objekt gespeichert werden. Bei den wenigen B-Objekten, die nicht mit einem A-Objekt verbunden sind, nehmen die A-Attribute den Wert NULL an.

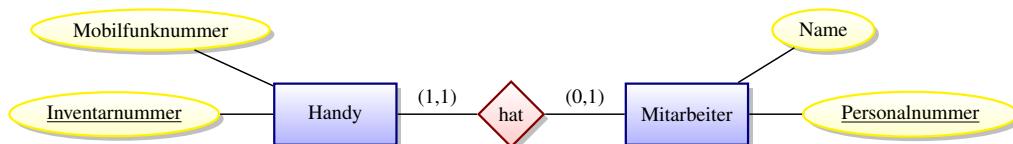
Transformationsregel T03 für den (1,1):(0,1) Beziehungstyp (selten realisierte Optionalität)

konzeptionelles Datenmodell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	⇒	Alle Eigenschaften von A werden als nicht-eingabepflichtige Attribute in B aufgenommen. SA wird außerdem als unikal [UN] deklariert
Objekttyp B mit Schlüssel <u>SB</u>	⇒	Tabelle B mit PK <u>SB</u>
(1,1):(0,1) Beziehungstyp	⇒	wird nicht gesondert dargestellt



Beispiel - Mitarbeiterhandy

Werden fast alle Mitarbeiter eines Unternehmens mit genau einem Handy ausgestattet, so sind die ergänzenden Angaben für das Handy bei nahezu allen Mitarbeitern erforderlich. Die entsprechende Transformation zeigt die folgende Abbildung.



Mitarbeiter(Personalnummer, Name, Handy-Inventarnummer [UN], Mobilfunknummer)

Die ursprüngliche Eigenschaftsbezeichnung „Inventarnummer“ wurde durch den Zusatz „Handy“ in ihrem neuen Kontext „sprechender“ gewählt. Die Handyspalten der Tabelle Mitarbeiter sind nicht-eingabepflichtig, d.h. sie nehmen für jene Mitarbeiter, die nicht mit einem Handy ausgestattet sind, den NULL-Wert an. Dadurch, dass das Attribut „Handy-Inventarnummer“ als unikal deklariert ist, kann ein und dasselbe Handy nicht mehreren Mitarbeitern zugeordnet werden.

Fall 2: #A ist wesentlich kleiner als #B

Betrachten wir nun den Fall, dass es wesentlich mehr B-Objekte als A-Objekte gibt. Würde man jetzt die beiden Objekttypen in einer Tabelle vereinen, hätten die Attribute der A-Objekte in vielen Zeilen den NULL-Wert. Um dies zu vermeiden, werden zwei Tabellen angelegt: eine B-Tabelle für alle B-Objekte und eine A-Tabelle für die seltenen A-Objekte, die durch einen eingabepflichtigen Fremdschlüssel jeweils auf „ihr“ B-Objekt verweisen. Die Kardinalität „1“ von (0,1) auf der B-Seite wird dadurch erzwungen, dass der Fremdschlüssel in A als unikal deklariert wird. Diese Umsetzung entspricht der Transformationsregel T04.

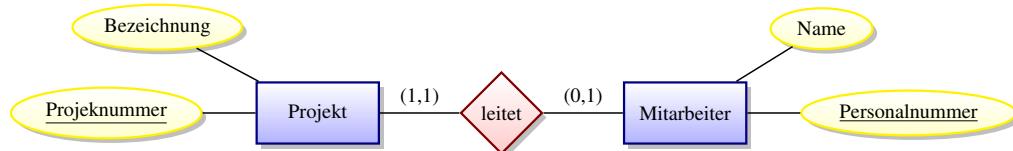
Transformationsregel T04 für den (1,1):(0,1) Beziehungstyp (oft realisierte Optionalität)

konzeptionelles Datenmodell	physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow Tabelle A mit PK <u>SA</u>
Objekttyp B mit Schlüssel <u>SB</u>	\Rightarrow Tabelle B mit PK <u>SB</u>
(1,1):(0,1) Beziehungstyp	\Rightarrow <u>SB</u> wird in A als eingabepflichtiger unikaler Fremdschlüssel aufgenommen ($\uparrow SB \uparrow [NN, UN]$)



Beispiel - Projektleiter

Soll beispielsweise festgehalten werden, welcher Mitarbeiter ein - und höchstens ein - Projekt leitet, wobei für jedes Projekt genau ein Mitarbeiter verantwortlich ist, so wird es viele Mitarbeiter ohne Projektverantwortung geben. Daher ist in diesem Fall die Transformationsregel T04 anzuwenden.



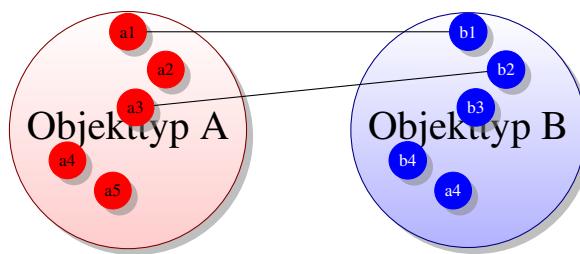
(A:) Projekt(Projektnummer, Bezeichnung, \uparrow Personalnummer \uparrow [NN, UN])

(B:) Mitarbeiter(Personalnummer, Name)

Da der Fremdschlüssel \uparrow Personalnummer \uparrow eingabepflichtig ist, muss jedes Projekt genau einen Projektleiter haben. Andererseits ist der Fremdschlüssel unikal, so dass ein Mitarbeiter nur für ein Projekt als Leiter ausgewiesen sein kann. Da es nicht möglich ist sicherzustellen, dass jede Personalnummer eines Mitarbeiters auch tatsächlich als Wert in der Fremdschlüsselspalte auftritt, kann es Mitarbeiter geben, die mit keinem Projekt als Leiter verbunden sind.

4.3.3 Der (0,1):(0,1) Beziehungstyp

Beim (0,1):(0,1) Beziehungstyp ist jedes Objekt des Typs A mit keinem oder einem Objekt des Typs B verbunden und umgekehrt. Es gibt also A-Objekte ohne B-Partner und B-Objekte ohne A-Partner. Jedes der Objekte kann aber höchstens einen Partner haben. Schematisch ist dies in der folgenden Abbildung dargestellt.

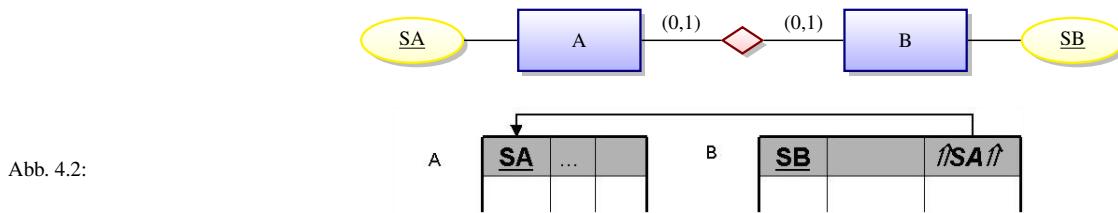


Wir nehmen für die folgenden Betrachtungen - ohne Beschränkung der Allgemeinheit - an, dass es höchstens so viele B-Objekte gibt wie A-Objekte ($\#A \geq \#B$, andernfalls müssen die Objekttypen einfach die Seiten tauschen). Da sowohl die A-Objekte als auch die B-Objekte unabhängig voneinander existieren können, müssen sie jeweils in einer eigenen Tabelle gespeichert werden.

Bei den B-Objekten wird ein Verweis auf „ihren“ A-Partner hinterlegt. Bei den „partnerlosen“ B-Objekten hat dieser Verweis dann den NULL-Wert. Deshalb wird der Primärschlüssel von A in der Tabelle B als nicht-eingabepflichtiger Fremdschlüssel aufgenommen. Fordert man für den Fremdschlüssel außerdem die Unikalität, kann auf ein A-Objekt nur von höchstens einem B-Objekt aus verwiesen werden. Die Regel T05 zeigt diesen Sachverhalt.

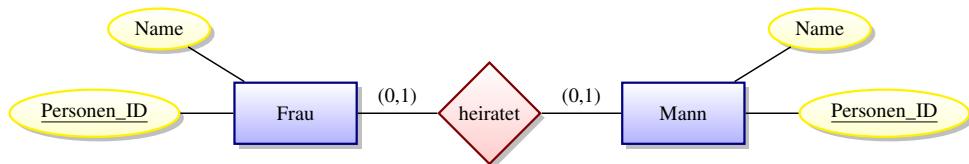
Transformationsregel T05 für den (0,1):(0,1) Beziehungstyp

konzeptionelles Datenmodell	physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow Tabelle A mit PK <u>SA</u>
Objekttyp B mit Schlüssel <u>SB</u>	\Rightarrow Tabelle B mit PK <u>SB</u>
(0,1):(0,1) Beziehungstyp	\Rightarrow <u>SA</u> wird in B als nicht-eingabepflichtiger unikaler Fremdschlüssel aufgenommen (\uparrow <u>SA</u> \uparrow [UN])



Beispiel - Ehe ohne Scheidung

In einem hier nicht näher genannten Land wird die monogame Ehe praktiziert, jedoch mit dem Unterschied zu Deutschland, dass eine Scheidung rechtlich nicht vorgesehen ist. Die Eheschließung zweier Personen ließe sich somit als binärer $(0,1):(0,1)$ -Beziehungstyp darstellen, da jede Frau höchstens einen Mann und jeder Mann höchstens eine Frau heiraten kann.



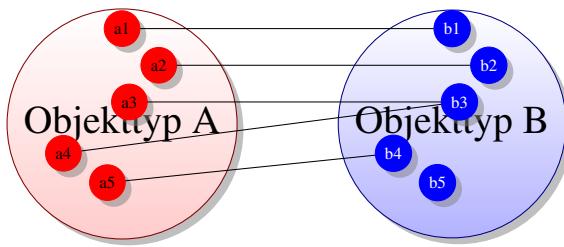
(A:) Frau(Personen_ID, Name)

(B:) Mann(Personen_ID, Name, \uparrow Ehegatten_ID \uparrow [UN])

Nicht verheiratete Männer haben, statt des Verweises auf die \uparrow Personen_ID \uparrow des Ehegatten, einen NULL-Wert. Wegen der Unikalität des Fremdschlüssels kann eine Frau höchstens einen Mann heiraten.

4.3.4 Der $(1,1):(0,*)$ Beziehungstyp

Beim $(1,1):(0,*)$ Beziehungstyp ist jedes Objekt B mit keinem, einem oder mehreren Objekten des Objekttyps A verbunden. Ein A-Objekt ist dagegen immer mit genau einem B-Objekt gekoppelt. Die folgende Abbildung zeigt dies schematisch.



Der Beziehungstyp wird im physischen Datenbankmodell durch je eine Tabelle für die A-Objekte und die B-Objekte repräsentiert, wobei der PK SB von B als eingabepflichtiger Fremdschlüssel in A aufgenommen wird. Jedes A-Objekt muss dann auf genau ein B-Objekt verweisen. Da der Fremdschlüssel aber nicht als unikal vereinbart ist, können mehrere A-Objekte mit demselben B-Objekt gekoppelt sein.

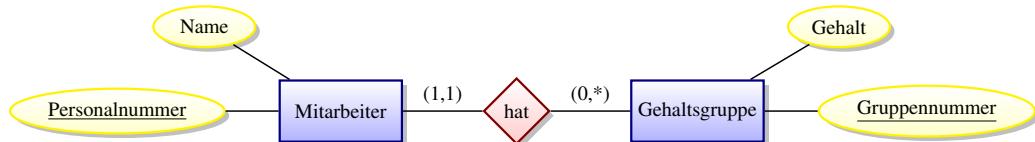
Transformationsregel T06 für den (1,1):(0,*) Beziehungstyp

konzeptionelles Datenmodell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	⇒	Tabelle A mit PK <u>SA</u>
Objekttyp B mit Schlüssel <u>SB</u>	⇒	Tabelle B mit PK <u>SB</u>
(1,1):(0,*) Beziehungstyp	⇒	<u>SB</u> wird in A als eingabepflichtiger nicht-unikaler Fremdschlüssel aufgenommen ($\uparrow SB \uparrow [NN]$)



Beispiel - Gehaltsgruppen

Beispielsweise kann eine Gehaltsgruppe noch keinem, erst einem Mitarbeiter oder bereits mehreren Mitarbeitern zugeordnet sein. Andererseits wird jeder Mitarbeiter in genau eine Gehaltsgruppe eingeordnet.



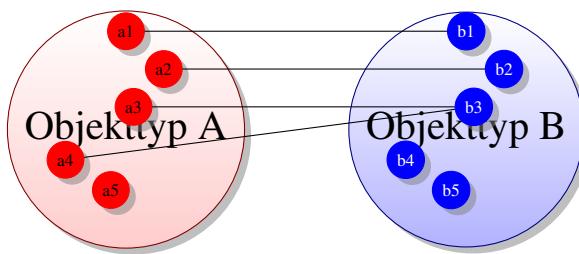
(B:) Gehaltsgruppe(Gruppennummer, Gehalt)

(A:) Mitarbeiter(Personalnummer, Name, \uparrow Gruppennummer \uparrow [NN])

Da der Fremdschlüssel \uparrow Gruppennummer \uparrow eingabepflichtig ist, muss jedem Mitarbeiter genau eine Gehaltsgruppe zugeordnet werden. Andererseits ist der Fremdschlüssel nichtunikal, so dass mehrere Mitarbeiter auf dieselbe Gehaltsgruppe verweisen können. Da nicht zu fordern ist, dass jeder Wert des Primärschlüssels Gruppennummer auch tatsächlich als Wert des Fremdschlüssels \uparrow Gruppennummer \uparrow auftreten muss, kann es Gehaltsgruppen geben, auf die noch nicht verwiesen wird.

4.3.5 Der (0,1):(0,*) Beziehungstyp

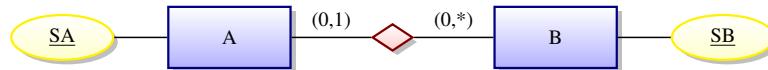
Beim (0,1):(0,*) Beziehungstyp kann ein Objekt des Typs B mit keinem, einem oder mehreren A-Objekten gekoppelt sein. Ein A-Objekt kann aber zu höchstens einem B-Objekt in Beziehung stehen.



Die Transformation dieses Beziehungstyps erfolgt, indem der Primärschlüssel von B als nicht-eingabepflichtiger und nicht-unikaler Fremdschlüssel in A aufgenommen wird.

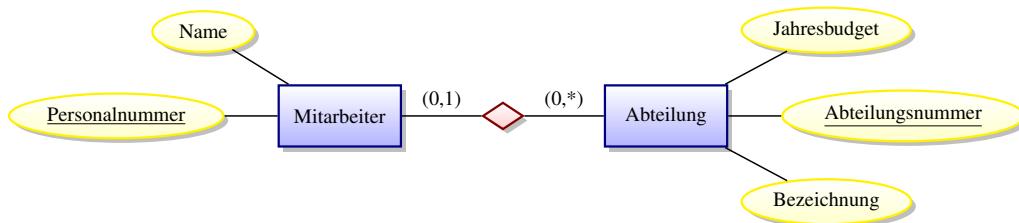
Transformationsregel T07 für den (0,1):(0,*) Beziehungstyp

konzeptionelles Datenmodell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
Objekttyp B mit Schlüssel <u>SB</u>	\Rightarrow	Tabelle B mit PK <u>SB</u>
(0,1):(0,*) Beziehungstyp	\Rightarrow	<u>SB</u> wird in A als nichteingabepflichtiger nicht-unikaler Fremdschlüssel aufgenommen ($\uparrow SB \uparrow$)



Beispiel - Abteilungsmitarbeiter

Betrachten wir eine Abteilung, die noch keinen, schon einen oder bereits mehrere Mitarbeiter hat. Die meisten Mitarbeiter sind in eine Abteilung eingeordnet, einige wenige - mit zentralen Aufgaben - gehören jedoch zu keiner Abteilung.



(B:) Abteilung(Abteilungsnummer, Jahresbudget, Bezeichnung)

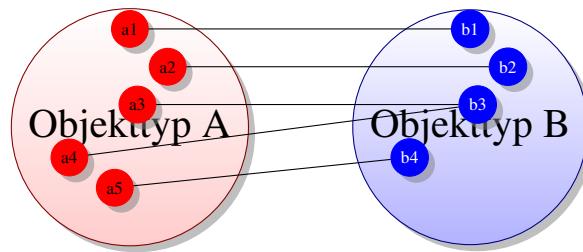
(A:) Mitarbeiter(Personalnummer, Name, \uparrow Abteilungsnummer \uparrow)

Bei den mit zentralen Aufgaben betrauten Mitarbeitern wird in der Fremdschlüssel Spalte \uparrow Abteilungsnummer \uparrow ein NULL-Wert stehen. Da der Fremdschlüssel nicht-unikal ist, können mehrere Mitarbeiter mit derselben Abteilung in Verbindung gebracht werden.

Da es ohnehin nicht möglich ist zufordern, dass jeder Wert eines Primärschlüssels auch in der Spalte des Fremdschlüssels \uparrow Abteilungsnummer \uparrow auftritt, kann es Abteilungen ohne Mitarbeiter geben.

4.3.6 Der (1,1):(1,*) Beziehungstyp

Der (1,1):(1,*) Beziehungstyp unterscheidet sich vom (1,1):(0,*) Beziehungstyp nur dadurch, dass jedes Objekt des Objekttyps B mit mindestens einem Objekt des Objekttyps A verbunden sein muss.



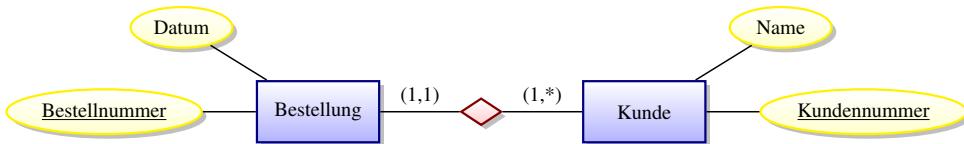
In der Literatur zum physischen Datenbankmodell (auch relationales Modell) wird der Beziehungstyp (1,1):(1,*) gewöhnlich als „meistverbreitetste“ Art von Beziehungstypen bezeichnet. Das ist aber falsch, denn dieser Beziehungstyp lässt sich im physikalischen Datenbankmodell gar nicht repräsentieren. Der Grund dafür liegt darin, dass es auf der Ebene der Tabellen-Typbeschreibungen keine Möglichkeit gibt, die Nichtoptionalität der Beziehungstyprichtung von B zu A zu repräsentieren.

Betrachten wir die Situation im Einzelnen. Die letzten Lösungen zur Transformation eines Beziehungstyps bestanden darin, den PK der B-Tabelle in die A-Tabelle als FK aufzunehmen. Soll nun jedes B-Objekt mit mindestens einem A-Objekt gekoppelt sein, so ist das gleichbedeutend mit der Forderung, dass jeder Wert, den der PK B annimmt, wenigstens einmal als Wert des FK in A auftauchen muss. Diese Forderung hat nichts mit der referentiellen Integrität zu tun. Diese fordert nur, dass jeder Wert des FK in A entweder der NULL-Wert sein oder aber als Wert des PK in B vorhanden sein muss. Die Umkehrrichtung, dass jeder Wert des PK auch als Wert des FK auftauchen muss, lässt sich im physischen Datenbankmodell nicht formulieren.

Die Transformation des (1,1):(1,*) Beziehungstyps in das physische Datenbankmodell erfolgt deshalb auf die gleiche Weise, wie die des (1,1):(0,*) Beziehungstyps, gemäß T06. Dabei muss allerdings in Kauf genommen werden, dass wichtige semantische Informationen, die im konzeptionellen Datenmodell repräsentiert werden, verloren gehen. Diese Informationen können nur im Rahmen der Anwendungsprogrammierung berücksichtigt werden.

Beispiel Kundenbestellung

Betrachten wir ein Unternehmen, in dem eine Geschäftsregel besagt, dass ein Kunde erst dann gespeichert wird, wenn er die erste Bestellung vornimmt. Im Laufe der Zeit können einem Kunden natürlich mehrere Bestellungen zugeordnet werden. Jede Bestellung kommt von genau einem Kunden. Ein Kunde, für den keine Bestellung mehr besteht (weil er alle seine Bestellungen storniert hat), wird wieder gelöscht. Das Datenmodell muss nach der Transformationsregel T06, entsprechend dem folgenden Bild, umgesetzt werden.



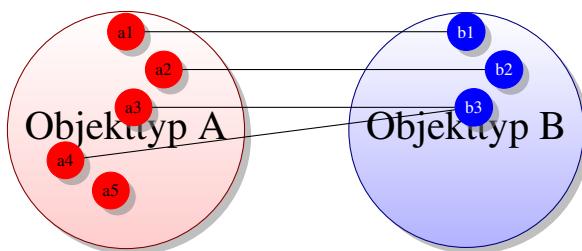
(B:) Kunde(Kundennummer, Name)

(A:) Bestellung(Bestellnummer, Datum, ↑Kundennummer↑ [NN])

Der FK ↑Kundennummer↑ ist eingabepflichtig: Jede Bestellung wird also genau einem Kunden zugeordnet. Der FK ist nicht-unikal: Mehrere Bestellungen können also auf denselben Kunden verweisen. Es wird aber nicht gesichert, dass jeder Kunde mit mindestens einer Bestellung verknüpft ist. Diese Geschäftsregel kann nicht beim Datenbankentwurf „festgeschrieben“ werden, sondern muss durch die Anwendungssoftware erzwungen werden.

4.3.7 Der (0,1):(1,*) Beziehungstyp

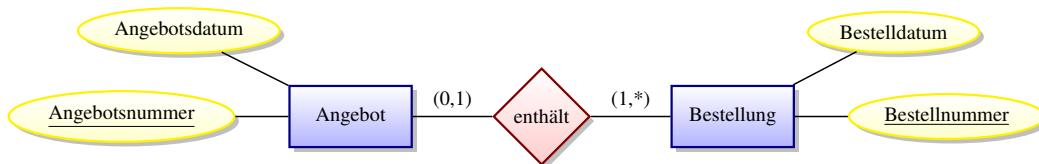
Der (0,1):(1,*) Beziehungstyp unterscheidet sich vom (1,1):(1,*) Beziehungstyp dadurch, dass nicht jedes Objekt des Objekttyps A mit einem Objekt des Objekttyps B gekoppelt sein muss. Es müssen aber wieder alle Objekte aus B mindestens eine Beziehung zu Objekten aus A haben.



Wie schon beim (1,1):(1,*) Beziehungstyp, so ist auch hier die Nichtoptionalität der Beziehungstyprichtung von B zu A im physischen Datenbankmodell nicht zu erzwingen. Die Transformation des (0,1):(1,*) Beziehungstyps kann nur wie beim (0,1):(0,*) Beziehungstyp erfolgen, also gemäß T07. Es muss auch hier der Verlust von semantischen Informationen in Kauf genommen werden.

Beispiel - Exklusivangebotsbestellung

Betrachten wir als Beispiel einen Versandhandel, der Exklusivangebote für seine Kunden erstellt. Jeder Kunde kann zu einem Angebot höchstens eine Bestellung abschicken, er muss es aber nicht tun (Kardinalität (0,1)). In einer Bestellung kann der Kunde sich aber nicht nur auf ein Angebot beziehen, sondern auch auf mehrere (Kardinalität (1,*)).



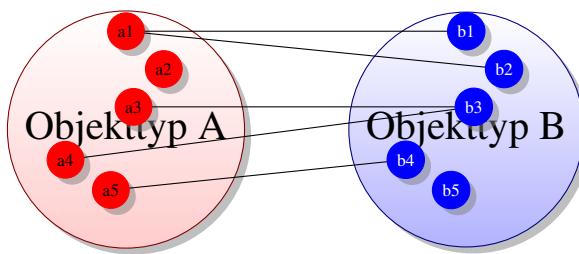
(B:) Bestellung(Bestellnummer, Bestelldatum)

(A:) Angebot(Angebotsnummer, Angebotsdatum, ↑Bestellnummer↑) Der FK ↑Bestellnummer↑ ist nicht-eingabepflichtig: Ein Angebot muss also nicht unbedingt eine Bestellung hervorrufen. Der FK ist nicht-unikal: Mehrere Angebote können in einer Bestellung genutzt werden. Es lässt sich aber nicht erzwingen, dass jede Bestellung sich auf mindestens ein Angebot bezieht. Die Datenbank würde also durchaus zulassen, dass ein Kunde eine Bestellung tätigt, ohne ein entsprechendes Angebot vorliegen zu haben. Will man diesen, in der Praxis nicht hinnehmbaren Fehler vermeiden, kann dies nur durch eine entsprechende Gestaltung der Anwendungssoftware erreicht werden.

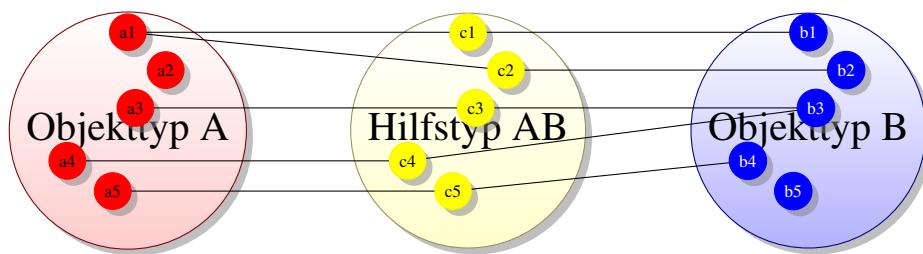
4.3.8 Der (0,*)(0,*) Beziehungstyp

Beziehungstypen, die in beiden Richtungen die Kardinalität „*“ aufweisen, lassen sich nicht direkt im physischen Datenbankmodell darstellen. Der Grund dafür liegt in der Tatsache, dass Attributwerte nur atomare Werte haben dürfen. Der Wert eines FK kann somit nur auf eine Zeile und nicht auf mehrere Zeilen verweisen.

Beim (0,*)(0,*) Beziehungstyp kann ein Objekt des Objekttyps A jedoch nicht nur mit keinem oder einem, sondern auch mit mehreren Objekten des Objekttyps B verbunden sein. Ebenso wie ein B-Objekt mit keinem, einem oder mehreren A-Objekten gekoppelt sein kann. Folgende Abbildung zeigt dafür ein Beispiel.



Die Repräsentation ist im physischen Datenbankmodell nur dadurch möglich, dass man einen neuen - rein technisch bedingten - Hilfsobjekttyp A/B einführt. In mancher Literatur wird auch von einem Koppel-Objekttyp gesprochen. A/B wird mit den Objekttypen A und B jeweils durch einen $(0,*):(1,1)$ Beziehungstyp verbunden.



Die beiden $(1,1):(0,*)$ Beziehungstypen werden gemäß T06 umgewandelt. Zusammenfassend gilt für den $(0,*):(0,*)$ Beziehungstyp die Transformationsregel T08.

Transformationsregel T08 für den $(0,*):(0,*)$ Beziehungstyp

ER-Modell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
Objekttyp B mit Schlüssel <u>SB</u>	\Rightarrow	Tabelle B mit PK <u>SB</u>
$(0,*):(0,*)$ Beziehungstyp	\Rightarrow	Hilfstabelle A/B. <u>SA</u> und <u>SB</u> werden als eingabepflichtige nicht-unikale Fremdschlüssel in A/B aufgenommen. Die Kombination der FK $\uparrow\!SA\!\uparrow$ und $\uparrow\!SB\!\uparrow$ als unikal vereinbart. Sie bildet den PK von A/B

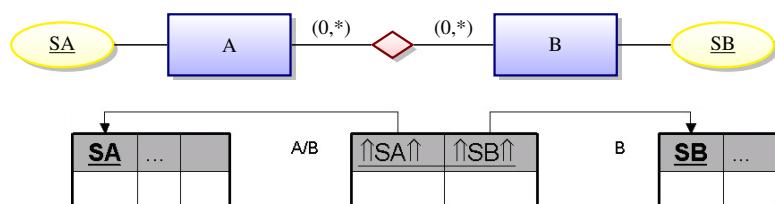
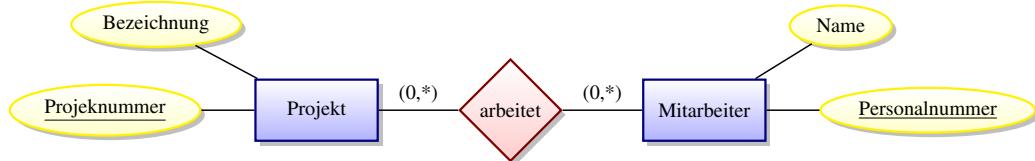


Abb. 4.3:

Beispiel - Projektmitarbeiter

Als Beispiel betrachten wir noch einmal die Mitarbeiter, die an keinem, einem oder mehreren Projekten beteiligt sein können, wobei ein Projekt (noch) von keinem, einem oder auch von mehreren Mitarbeitern bearbeitet wird.



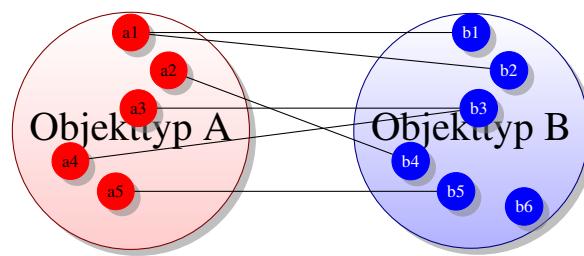
(B:) Mitarbeiter(Personalnummer, Name)

(A:) Projekt(Projektnummer, Bezeichnung)

(A/B:) MitarbeiterProjekt(\uparrow Personalnummer + Projektnummer \uparrow) Eine Zeile der Hilfstabelle „MitarbeiterProjekt“ verweist auf genau einen Mitarbeiter und auf genau ein Projekt. Da der FK \uparrow Personalnummer \uparrow für sich alleine nichtunikal ist, kann es mehrere Zeilen der Hilfstabelle geben, die auf denselben Mitarbeiter verweisen. Die Nichtunikalität des FK \uparrow Projektnummer \uparrow ermöglicht es, dass mehrere Zeilen der Hilfstabelle mit demselben Projekt verknüpft sind. Erst die Kombination \uparrow Personalnummer + Projektnummer \uparrow ist als unikal vereinbart und bildet den PK der Hilfstabelle.

4.3.9 Der (1,*):(0,*) Beziehungstyp

Beim (1,*):(0,*) Beziehungstyp muss ein Objekt des Objekttyps A mit mindestens einem oder mehreren Objekten des Objekttyps B verbunden sein, ein B-Objekt jedoch mit keinem, einem oder aber mehreren A-Objekten.



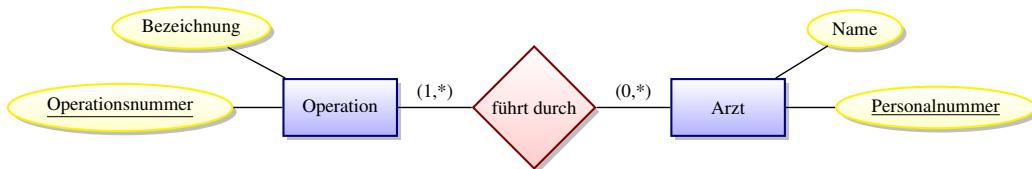
Analog zum (0, *):(0, *) Beziehungstyp muss der (1, *):(0, *) Beziehungstyp vor seiner Transformation in das physische Datenbankmodell durch die Einführung eines neuen Objekttyps A/B in einen (1,1):(0,*) Beziehungstyp und einen (1, *):(1,1) Beziehungstyp umgewandelt werden.



Es wurde bereits gezeigt, dass der $(1, *):(1, 1)$ Beziehungstyp im physischen Datenbankmodell lediglich als $(0, *):(1, 1)$ Beziehungstyp dargestellt werden kann. Die Transformation des $(1, *):(0, *)$ Beziehungstyps erfolgt damit nach der Transformationsregel T08.

Beispiel - ärzte mit Operation(en)

Als Beispiel betrachten wir ärzte, die Operationen durchführen. Eine Operation ohne ärzte gibt es nicht. Mitunter wird eine Operation aber von mehreren ärzten ausgeführt.



(B:) Arzt(Personalnummer, Name)

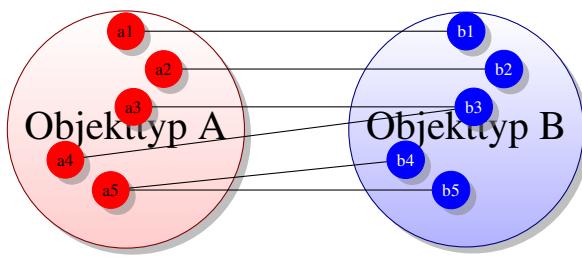
(A:) Operation(Operationsnummer, Bezeichnung)

(B/A:) ArztOperation(\uparrow Personalnummer + Operationsnummer \uparrow) Durch jede Zeile der Hilfstabelle „ArztOperation“ wird ein Arzt mit einer Operation in Verbindung gebracht. Keiner der beiden Fremdschlüsselelemente \uparrow Personalnummer \uparrow beziehungsweise \uparrow Operationsnummer \uparrow muss für sich genommen unikal sein. Damit können mehrere Zeilen der Hilfstabelle auf denselben Arzt verweisen. Ebenso können mehrere Zeilen dieselbe Operation betreffen.

Es kann aber durch die Tabellentypbeschreibungen nicht erzwungen werden, dass jede Operationsnummer auch tatsächlich als FK in der Hilfstabelle auftaucht. In der DB kann also die falsche Aussage gespeichert werden, dass einer Operation kein Arzt zugeordnet ist. Dieser Fehler kann wiederum nur software-technisch vermieden werden.

4.3.10 Der $(1, *):(1, *)$ Beziehungstyp

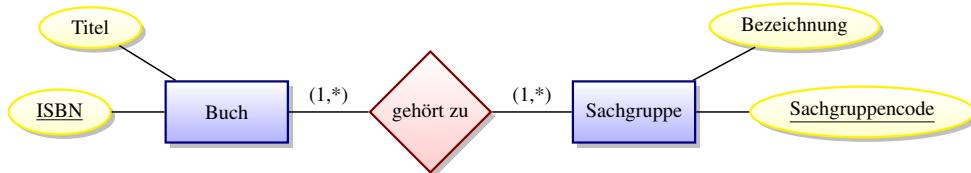
Beim $(1, *):(1, *)$ Beziehungstyp ist jedes Objekt des Objekttyps A mit einem oder mehreren Objekten des Objekttyps B verbunden, ebenso wie jedes B-Objekt mit einem oder mehreren A-Objekten gekoppelt ist.



Dieser Beziehungstyp muss vor der Transformation in einen $(1, *):(1, 1)$ und einen $(1, 1):(1, *)$ Beziehungstyp umgeformt werden. Beide neuen Beziehungstypen müssen wiederum nur unter Semantikverlust in $(0, *):(1, 1)$ bzw. $(1, 1):(0, *)$ Beziehungstypen überführt werden. Die Transformation des $(1, *):(1, *)$ Beziehungstyps erfolgt damit - so wie für den $(1, *):(0, *)$ Beziehungstyp - nach der Transformationsregel T08.

Beispiel - Büchersachgruppe

Als Beispiel betrachten wir eine Bibliothek, in der die Bücher den einzelnen Sachgruppen zugeordnet werden. Ein Buch wird meist nur für eine Sachgruppe, mitunter aber auch zu mehreren Sachgruppen zugeordnet. Eine Sachgruppe wird erst dann eingeführt, wenn sie wenigstens ein Buch enthält. Die Transformation sieht folgendermaßen aus.



(A:) Buch(ISBN, Titel)

(B:) Sachgruppe(Sachgruppencode, Bezeichnung)

(A/B:) BuchSachgruppe(↑ISBN + Sachgruppencode↑)

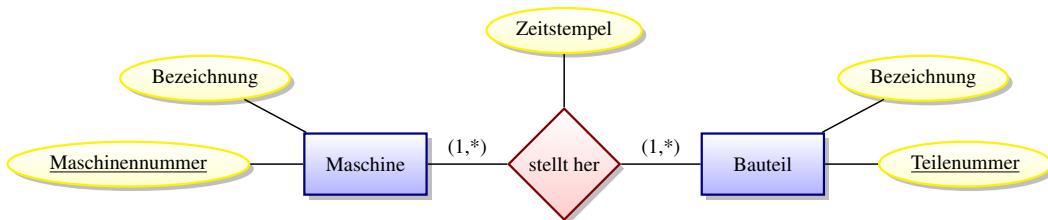
Eine Zeile der Hilfstabelle „Buch/Sachgruppe“ ordnet ein Buch einer Sachgruppe zu. Da weder der FK \uparrow ISBN \uparrow noch der FK \uparrow Sachgruppencode \uparrow für sich unikal sind, können mehrere Zeilen der Hilfstabelle auf dasselbe Buch oder auch auf dieselbe Sachgruppe verweisen. Die Tabellentypbeschreibungen sichern aber weder, dass einem Buch wenigstens eine Sachgruppe zugeordnet wird, noch können sie garantieren, dass es keine „leere“ Sachgruppe gibt. Diese Geschäftsregeln müssen von der Anwendungssoftware durchgesetzt werden.

4.3.11 Transformation von Beziehungsattributen

Bei der Transformation von Eigenschaften eines Beziehungstyps (Beziehungsattributen) kann folgende Verallgemeinerung angewandt werden:

- Das Beziehungsattribut „wandert“ in die Tabelle auf der 1-Seite des Beziehungstyps.
- Gibt es auf beiden Seiten einen * in der Kardinalität, dann „wandert/wandern“ das/die Beziehungsattribut(e) in das Hilfsobjekt.
- Gibt es keine *-Seite, dann muss die angewandte Transformationsregel betrachtet und geprüft werden, in welcher Tabelle das Beziehungsattribut am sinnvollsten ist. Normalerweise ist dies die Tabelle mit dem FK.

Transformation von Beziehungsattributen bei einer m:n-Beziehung

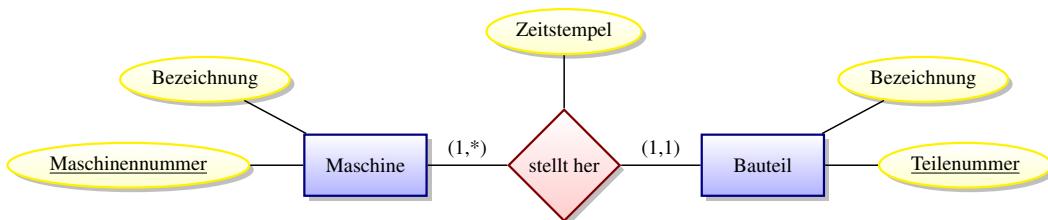


(A:) **Maschine**(Maschinennummer, Bezeichnung)

(B:) **Bauteil**(Teilenummer, Bezeichnung)

(A/B:) Herstellung(\uparrow Maschinennummer + Teilenummer \uparrow , Zeitstempel)

Transformation von Beziehungsattributen bei einer 1:n-Beziehung



(B): **Maschine**(Maschinennummer, Bezeichnung)

(A): Bauteil(Teilenummer, Bezeichnung, \uparrow Maschinensummer \uparrow , Zeitstempel)

Transformation von Beziehungsattributen bei einer 1:1-Beziehung

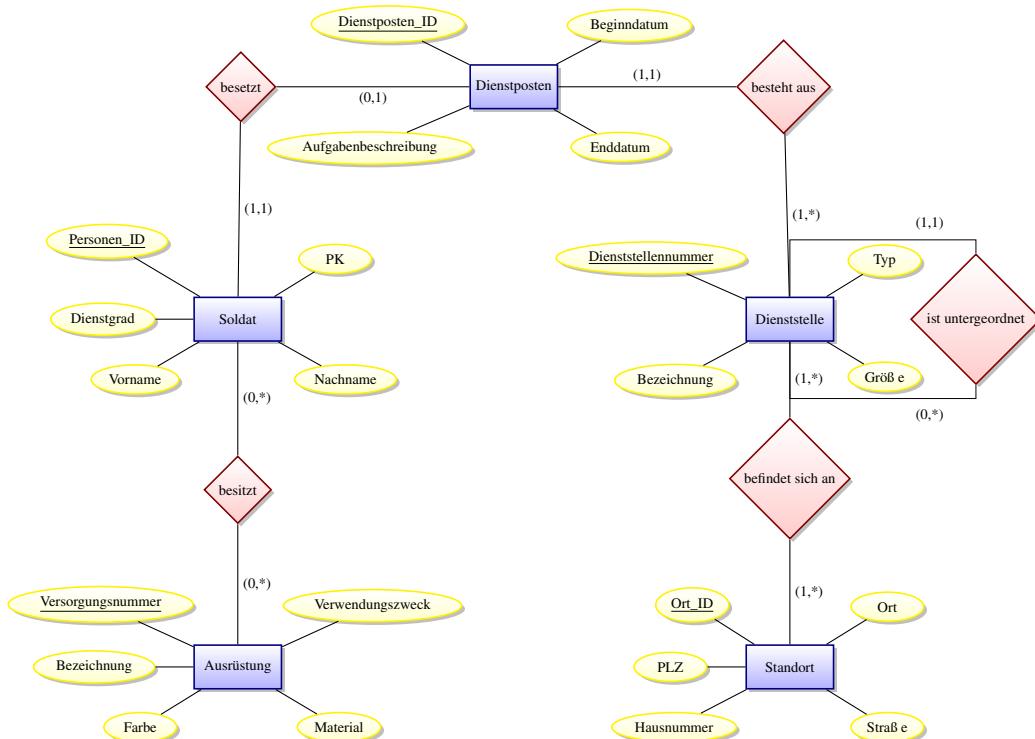
Da bei Beziehungen des Typs 1:1 nur extrem selten Beziehungsattribute vorkommen, muss in solchen Fällen eine Einzelfallbetrachtung durchgeführt werden. Daher wird an dieser Stelle auf ein Transformationsbeispiel verzichtet.

4.4 Transformation des Begleitbeispiels Bundeswehr

In diesem Abschnitt sollen die bis hier erlernten Dinge praktisch geübt werden. Aufgabe soll es sein, für das begleitende Beispiel „Bundeswehr“, ein ER-Modell zu konzipieren und dieses anschließend, mittels der in den vorigen Abschnitten kennengelernten Transformationsregeln, in ein relationales Modell umzuwandeln.

Nachfolgendes ER-Modell kann als Lösungsansatz für das Beispiel „Bundeswehr“ gewählt werden.

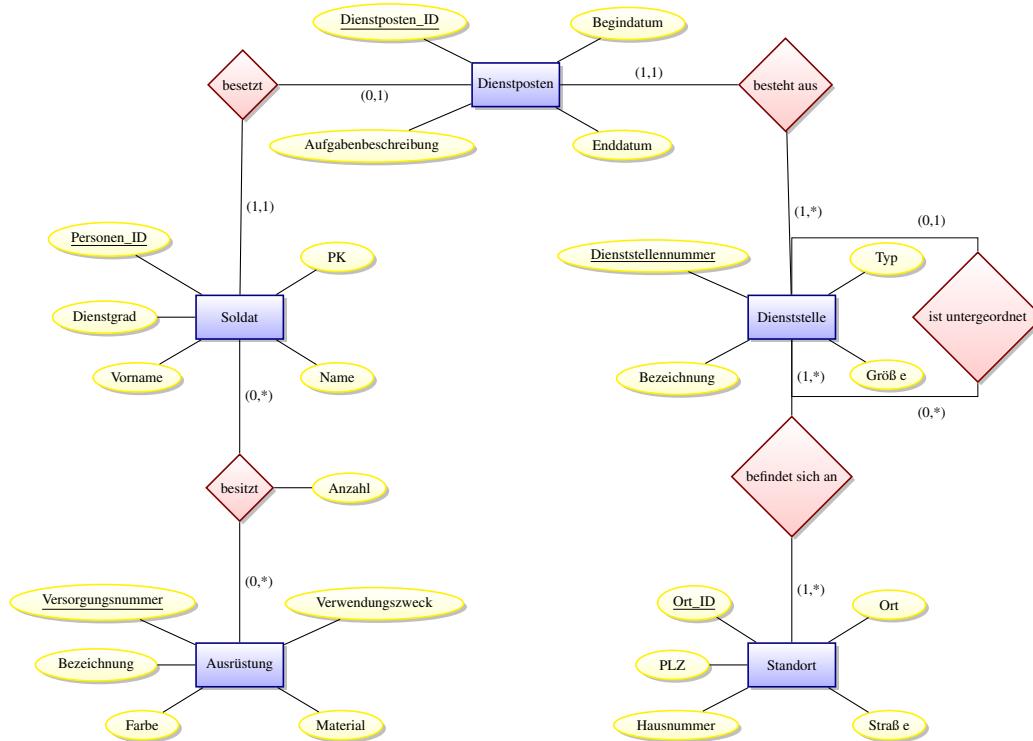
Dienstposten	(<u>Dienstposten_ID</u> , Beginndatum, Enddatum, Aufgabenbeschreibung)
Soldat	(<u>Personen_ID</u> , PK, Vorname, Nachname, Dienstgrad)
Ausrüstung	(<u>Versorgungsnummer</u> , Bezeichnung, Verwendungszweck, Material, Farbe)
Dienststelle	(<u>Dienststellenummer</u> , Bezeichnung, Typ, Größe)
Standort	(<u>Ort_ID</u> , PLZ, Ort, Straße, Hausnummer)
	Auflösen des Beziehungstyps (besetzt) mit T04
	Auflösen des Beziehungstyps (besteht aus) mit T06
	Auflösen der Beziehungstypen (besitzt, befindet sich an) mit T08
	Auflösen des Beziehungstyps (ist untergeordnet) mit T11
DienststelleStandort	(↑ <u>Dienststellenummer</u> + <u>Ort_ID</u> ↑)
SoldatAusrüstung	(↑ <u>Personen_ID</u> + <u>Versorgungsnummer</u> ↑)



Bei weiterer Betrachtung des ER-Modells sollte auffallen, dass die Anzahl an Ausrüstungsgegenständen eines Soldaten nicht berücksichtigt wurde. In der Tabelle „SoldatAusrüstung“ kann ein Attribut „Anzahl“ aufgenommen werden, welches die jeweilige Anzahl an Ausrüstungsgegenständen eines Soldaten enthält. Die neue Tabelle sieht dann folgendermaßen aus:

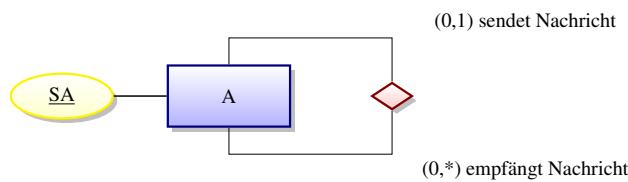
SoldatAusrüstung(Personen_ID + Versorgungsnummer, Anzahl)

Dieser Fall muss sich dann mit einem Beziehungsattribut (vgl. Abschnitt ??) im ER-Modell wiederfinden. Das angepasste ER-Modell sieht folgendermaßen aus:



4.5 Transformation rekursiver Beziehungstypen

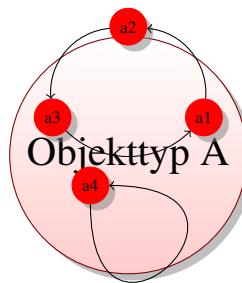
Da bei Rekursiv-Beziehungstypen die miteinander gekoppelten Objekte beide im selben Objekttyp liegen, können sie nicht - wie bei den binären Beziehungstypen - durch ihre Zugehörigkeit zum jeweiligen Objekttyp unterschieden werden. Bei Rekursiv-Beziehungstypen müssen die Objekte hinsichtlich ihrer Rolle unterschieden werden. Deshalb soll folgende Sprachregelung eingeführt werden. Wir sprechen bei einem Rekursiv-Beziehungstyp allgemein von einem „Sender“ (rechte Seite), der eine Nachricht an einen „Empfänger“ (linke Seite) sendet. Als Beispiel dafür ist in der folgenden Abbildung ein (0,1):(0,*) Rekursiv-Beziehungstyp dargestellt.



An dieser Stelle sei auf den Abschnitt ?? mit der Tabelle ?? verwiesen. In der Tabelle ?? sind die 7 möglichen Rekursiv-Beziehungstypen aufgelistet.

4.5.1 Der (1,1):(1,1) Rekursiv-Beziehungstyp

Beim (1,1):(1,1) Beziehungstyp muss jedes Objekt des Objekttyps genau eine Nachricht an ein Objekt desselben Objekttyps senden. Umgekehrt muss jedes Objekt genau eine Nachricht von einem anderen Objekt empfangen. Die folgende Abbildung zeigt die vorliegenden Verhältnisse.



Wie man sehen kann, können ausschließlich Objekt-Zyklen gebildet werden ($a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_1$). Im Minimalfall gibt es nur ein Objekt, welches dann einen Ein-Objekt-Zyklus bildet ($a_1 \rightarrow a_1$).

Bei der Repräsentation von Rekursiv-Beziehungstypen im physischen Datenbankmodell muss zunächst jedes Objekt des Objekttyps A - unabhängig von den anderen Objekten - in einer Tabellenzeile gespeichert werden. Die Beziehung zwischen zwei Objekten a_1 und a_2 kann nun dadurch ausgedrückt werden, dass in der Tabellenzeile von a_2 (beim Empfänger) auf das Objekt a_1 (auf den Sender) verwiesen wird. Der Verweis wird, wie schon bei den binären Beziehungstypen, durch Abspeichern des Identifikators von a_1 realisiert. Dieser Identifikator ist aber der Wert, den der Primärschlüssel von A im Falle des Objekts a_1 annimmt. Die Tabelle für den Objekttyp A muss somit den Schlüssel des Objekttyps A in doppelter Ausführung aufnehmen:

1. als Primärschlüssel, um eine Zeile der Tabelle eindeutig identifizieren zu können;
2. als Fremdschlüssel, um auf eine andere (oder auch auf dieselbe) Zeile der Tabelle, nämlich auf den Sender verweisen zu können.

Da die Spaltenbezeichnungen einer Tabelle voneinander verschieden sein müssen, ist es erforderlich, für das Attribut (bzw. die Attribute) des PK im Falle ihrer „Wiedergeburt“ als FK, andere Bezeichnungen zu vergeben.

Im physischen Datenbankmodell lässt sich die Forderung, dass jedes Objekt genau eine Nachricht empfangen muss, einfach dadurch sichern, dass der FK, der auf den Sender verweist, als *eingabepflichtig* deklariert wird, dass also jedes Objekt auf „sein“ Senderobjekt verweisen muss. Dass ein und dasselbe Objekt nicht von mehreren Empfängern als ihr Sender angegeben werden kann, wird durch die Unikalität des FK erreicht.

Transformationsregel T09 für den (1,1):(1,1) Rekursiv-Beziehungstyp

ER-Modell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
(1,1):(1,1) Rekursiv-Beziehungstyp	\Rightarrow	<u>SA</u> wird in A mit einer anderen Attributbezeichnung „gedoppelt“ und als eingabepflichtiger unikaler FK $\uparrow\!SA'\!\uparrow$ vereinbart, der auf den Sender verweist. ($\uparrow\!SA'\!\uparrow$ [NN, UN])

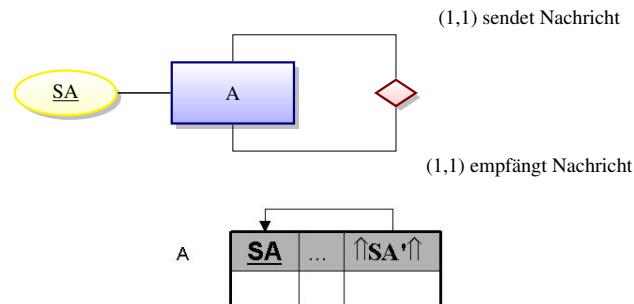


Abb. 4.4:

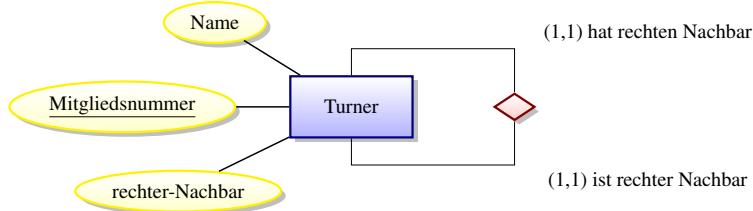
Beispiel – Kreis einer Turnergruppe

Als Beispiel betrachten wir eine Turnergruppe, die bei einem Sportfest einen Kreis bilden soll. Für jeden Turner wird festgelegt, wer sein rechter Nachbar ist.

Von jedem Turner wird obligatorisch auf seinen (einzigen) rechten Nachbarn verwiesen. Durch die Unikalität des FK wird außerdem gesichert, dass ein Turner nur für einen anderen Turner rechter Nachbar

sein kann. Da jedem Turner ein rechter Nachbar zugeordnet wird, muss auch jeder Turner rechter Nachbar genau eines anderen Turners sein. Damit ist die Kreisstruktur der Aufstellung der Turner erzwungen. Allerdings lässt sich durch die Tabellentypbeschreibung nicht ausschließen, dass ein Turner als sein eigener rechter Nachbar eingesetzt wird. (Ein-Objekt-Zyklus).

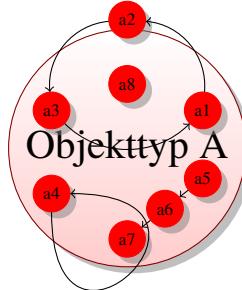
Schwierig wird bei diesem Beziehungstyp die Speicherung der Daten so vorzunehmen, dass keine Integritätsverletzung auftritt. Es sind entsprechende software-technische Maßnahmen zu treffen, die hier nicht weiter erläutert werden sollen.



Turner(Mitgliedsnummer, Name, ↗rechter-Nachbar↖ [NN, UN])

4.5.2 Der (0,1):(0,1) Rekursiv-Beziehungstyp

Beim (0,1):(0,1) Rekursiv-Beziehungstyp kann ein Objekt keine oder eine Nachricht an ein anderes Objekt (oder an sich selbst) senden. Andererseits kann ein Objekt keine oder eine Nachricht empfangen.



Beim (0,1):(0,1) Rekursiv-Beziehungstyp sind also - wie schon beim (1,1):(1,1) Rekursiv-Beziehungstyp - Objekt-Zyklen ($a_1 \rightarrow a_2 \rightarrow a_3 \rightarrow a_1$) möglich, die im Minimalfall Ein-Objekt-Zyklen ($a_4 \rightarrow a_4$) sind. Darüber hinaus kann es Objekt-Ketten ($a_5 \rightarrow a_6 \rightarrow a_7$) geben, die gegebenenfalls nur ein einziges Objekt enthalten (a_8). Ein solches Objekt ist dann weder Sender noch Empfänger.

Für die Umsetzung im physischen Modell, betrachten wir den Fall, dass die Objekte einen Verweis auf „ihren“ Sender benötigen. Dieser Sender kann der Empfänger selbst oder ein anderes A-Objekt sein. Da nicht jedes Objekt ein Empfänger ist, muss der FK, der auf den Sender verweist, als nicht-eingabepflichtig deklariert werden. Andererseits muss er unikal sein, damit ein a_1 Objekt nur von höchstens einem Objekt a_2 als „sein“ Sender ausgewiesen werden kann. Zusammengefasst ist dies die Transformationsregel T10.

Transformationsregel T10 für den (0,1):(0,1) Rekursiv-Beziehungstyp

ER-Modell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	⇒	Tabelle A mit PK <u>SA</u>
(0,1):(0,1) Rekursiv-Beziehungstyp	⇒	<u>SA</u> wird in A mit einer anderen Attributbezeichnung „gedoppelt“ und als nicht-eingabepflichtiger unikaler FK $\uparrow\!\!/\!\!\downarrow\text{SA'}$ vereinbart, der auf den Sender verweist. ($\uparrow\!\!/\!\!\downarrow\text{SA'}$ [UN])

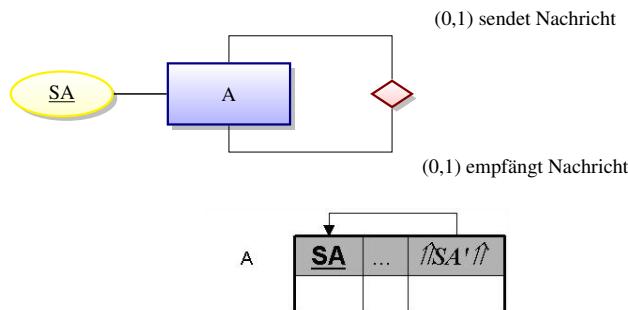
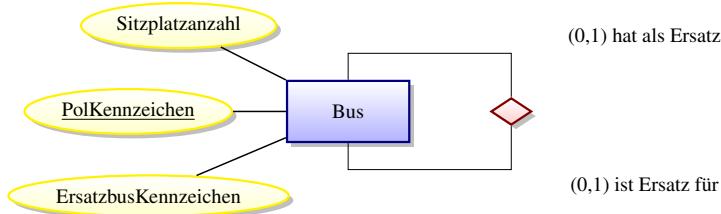


Abb. 4.5:

Beispiel - Ersatzbus

Nehmen wir als Beispiel an, dass in einem Busreiseunternehmen festgelegt wird, welcher Bus als Ersatz verwendet werden soll, wenn ein Bus defekt ist. Dabei soll ein Bus höchstens für einen anderen als Ersatz dienen. Für die wenigen neuen Busse mit geringer Ausfallwahrscheinlichkeit wird kein Ersatz vorgesehen und sie können auch nicht als Ersatz dienen, da sie ständig im Einsatz sind. Für andere Busse wird festgelegt, dass sie bei Schäden schnell repariert werden; sie sind dann Ersatz für sich selbst (Ein-Objekt-Zyklen). Typischerweise wird für jeden Bus ein anderer als Ersatz festgelegt und jeder Bus kann als Ersatz für einen anderen Bus dienen (Objekte in einem Mehr-Objekte-Zyklus oder „innere Objekte“ in einer Objekt-Kette). Es ist in seltenen Fällen aber auch nicht auszuschließen, dass ein Bus zwar nicht als Ersatz dienen kann, aber einen Ersatz braucht (Anfangsglied einer Kette). Ebenso kann ein Bus in Reserve gehalten werden, also nur als Ersatz dienen, ohne selbst einen Ersatz zu benötigen (Endglied einer Kette).



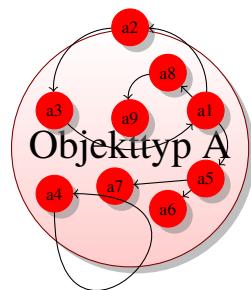
Bus(PolKennzeichen, Sitzplatzanzahl, $\uparrow\!\!/\!\!\downarrow\text{ErsatzbusKennzeichen}$ [UN])

Da der FK \uparrow ErsatzbusKennzeichen \uparrow als nicht-eingabepflichtig deklariert ist, kann ein Bus auf keinen oder auf einen anderen Bus (evtl. auf sich selbst) als Ersatzbus verweisen. Da der FK unikal ist, kann auf einen Ersatzbus nur von *einem* anderen Bus verwiesen werden. Andererseits ist nicht gefordert - und kann auch gar nicht gefordert werden -, dass jeder Wert des PK PolKennzeichen auch tatsächlich als FK auftritt. Somit kann es Busse geben, die nicht durch einen FK-Wert als Ersatzbusse ausgewiesen sind.

4.5.3 Der (1,1):(0,*) Rekursiv-Beziehungstyp

Beim (1,1):(0,*) Rekursiv-Beziehungstyp kann ein Objekt keine, eine oder mehrere Nachrichten senden, es muss aber stets genau eine Nachricht empfangen werden. Das entspricht den Bedingungen des (1,1):(1,1) Rekursiv-Beziehungstyps, zuzüglich der Möglichkeit eines Objekts, entweder gar nicht als Sender oder aber als Sender mehrerer Nachrichten aufzutreten.

Da jedes Objekt des Objekttyps A genau eine Nachricht empfangen muss, müssen auch #A Nachrichten gesendet werden. Für jedes Nicht-Sender-Objekt, muss somit ein anderes - gewissermaß en sein Vertreter - eine zusätzliche Nachricht senden.



Bei der Transformation, muss der Identifikator des Senders eingabepflichtig beim Empfänger hinterlegt werden. Wird der FK außer dem als nicht-unikal deklariert, kann ein Objekt von mehreren Empfänger-Objekten als „ihr“ Sender ausgewiesen werden. Folgend ist die Transformationsregel T11 dargestellt.

Transformationsregel T11 für den (1,1):(0,*) Rekursiv-Beziehungstyp

ER-Modell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
(1,1):(0,*) Rekursiv-Beziehungstyp	\Rightarrow	<u>SA</u> wird in A mit anderen Attributbezeichnungen „gedoppelt“ und als eingabepflichtiger nicht-unikaler FK $\uparrow\text{SA}'\uparrow$ vereinbart, der auf den Sender verweist. ($\uparrow\text{SA}'\uparrow$ [NN])

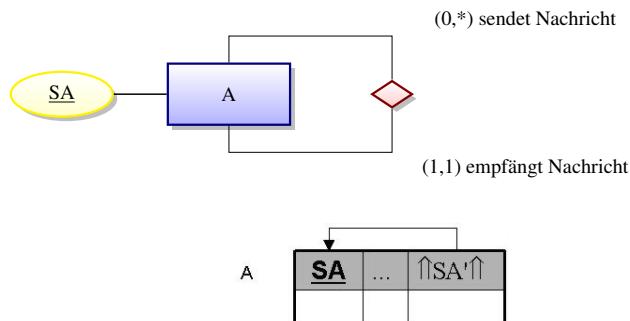
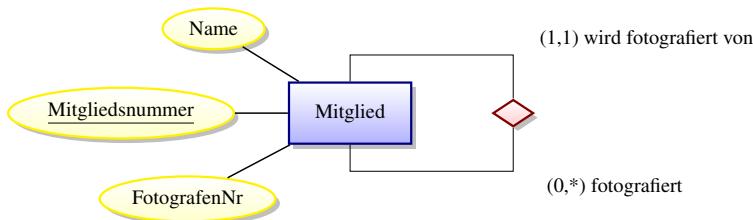


Abb. 4.6:

Beispiel - Fotos der Vereinsmitglieder

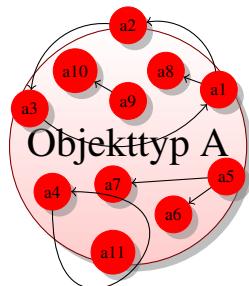
Nehmen wir als Beispiel an, dass für die Festzeitschrift eines Vereins von allen Mitgliedern ein Foto benötigt wird. Im Interesse der Kostendämpfung wird kein externer Fotograf hinzugezogen. Jedes Mitglied muss genau einmal fotografiert werden. Manche Mitglieder fotografieren gar nicht, andere müssen dafür umso mehr Fotos machen. Damit auch wirklich jeder Fotograf selbst auf einem Foto zu sehen ist, ist letztlich erforderlich, dass sich ein Fotograf entweder selbst fotografiert (Ein-Objekt-Zyklus) oder dass ihn jemand fotografiert, der selbst bereits fotografiert wurde (Mehr-Objekte-Zyklus). Die erforderliche Transformation ist in folgender Abbildung dargestellt.



Mitglied(Mitgliedsnummer, Name, $\uparrow\text{FotografenNr}\uparrow$ [NN])

4.5.4 Der (0,1):(0,*) Rekursiv-Beziehungstyp

Der (0,1):(0,*) Rekursiv-Beziehungstyp unterscheidet sich vom (1,1):(0,*) Rekursiv-Beziehungstyp nur durch die Aufhebung der Forderung, dass jedes Objekt eine Nachricht empfangen muss. Die folgende Abbildung zeigt dafür ein Beispiel.



Dieser Rekursiv-Beziehungstyp ist prädestiniert für die Repräsentation von Monohierarchien, wie sie in der Praxis in Form von Organigrammen oder Stücklisten auftreten. Als Sonderfall eines „Baumes“ ist eine Liste ($a_9 \rightarrow a_{10}$) möglich, die evtl. auch nur aus einem Element (a_{11}) bestehen kann.

Bei der Transformation dieses Rekursiv-Beziehungstyps in das physische Datenbankmodell wird der FK als nicht-eingabepflichtig und nicht-unikal deklariert. Die entsprechende Transformation ist in T12 zusammengefasst.

Transformationsregel T12 für den (0,1):(0,*) Rekursiv-Beziehungstyp

ER-Modell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
(0,1):(0,*) Rekursiv-Beziehungstyp	\Rightarrow	<u>SA</u> wird in A mit anderen Attributbezeichnungen „gedoppelt“ und als nicht-eingabepflichtiger nicht-unikaler FK $\uparrow SA \uparrow$ vereinbart, der auf den Sender verweist. ($\uparrow SA' \uparrow$)

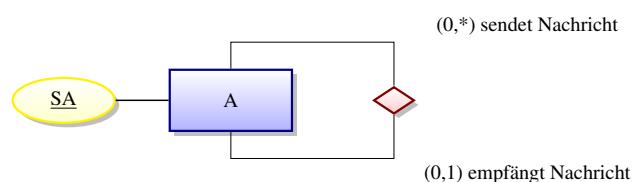
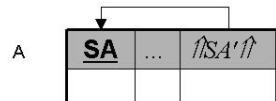
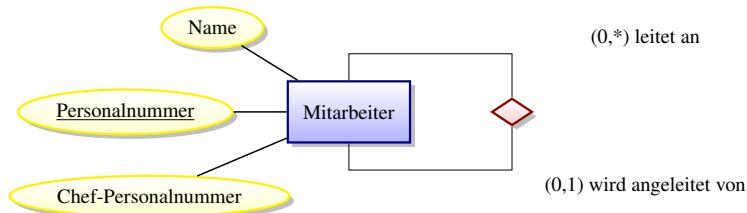


Abb. 4.7:



Beispiel - Unternehmenshierarchie

Betrachten wir als Beispiel die Leitungshierarchie im Unternehmen. Ein Mitarbeiter kann ohne Leistungsfunktion sein, er kann aber auch mehrere andere Mitarbeiter anleiten. Dies kann über mehrere Hierarchiestufen erfolgen. Andererseits haben die meisten Mitarbeiter genau einen Chef. Mitarbeiter der obersten Leitungsebene haben keinen Chef. Die Transformation ist in der folgenden Abbildung dargestellt.



Mitarbeiter(Personalnummer, Name, ↑Chef-Personalnummer)

Der FK ↑Chef-Personalnummer ist nicht-eingabepflichtig, somit kann es Mitarbeiter ohne Chef geben. Da der FK nicht-unikal ist, können mehrere Mitarbeiter auf denselben Chef verweisen. An diesem Beispiel sieht man, dass der (0,1):(0,*) Rekursiv-Beziehungstyp wesentlich mehr Objekt-Strukturen umfasst, als für die Repräsentation monohierarchischer Zusammenhänge benötigt werden:

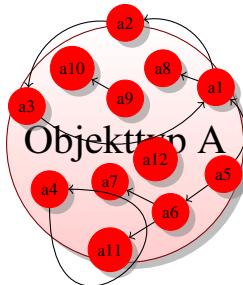
- *Mehr-Objekt-Zyklen*: Der Mitarbeiter mit der Personalnummer „0815“ kann Chef des Mitarbeiters „0816“ sein. Dieser kann den Mitarbeiter „0817“ anleiten, der wiederum Chef des Mitarbeiters „0815“ ist.
- *Ein-Objekt-Zyklen*: Ein Mitarbeiter kann als sein eigener Chef ausgewiesen werden, wenn nämlich in einem Datensatz die Werte von PK und FK identisch sind.
- *Mehr-Objekt-Ketten*: Ein Mitarbeiter kann einen einzigen Mitarbeiter anleiten, der wiederum Chef eines einzigen Mitarbeiter ist.
- *Ein-Objekt-Ketten*: Ein Mitarbeiter, der keinen Chef hat, leitet selbst auch niemanden an.

In der Praxis sind solche Situationen gewöhnlich verboten, sie können aber durch die Tabellentypbeschreibung nicht verhindert werden. Die Anwendungssoftware muss sichern, dass die beschriebenen Fälle nicht realisiert werden, sofern es nicht gewollt ist.

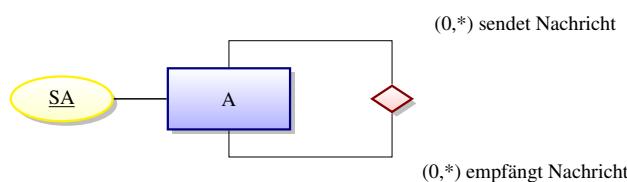
4.5.5 Der $(0,*)(0,*)$ Rekursiv-Beziehungstyp

Der $(0,*)(0,*)$ Rekursiv-Beziehungstyp stellt keinerlei einschränkende Bedingungen an die Struktur, nach der die Objekte eines Objekttyps A miteinander in Beziehung stehen. Jedes Objekt kann keine, eine oder mehrere Nachrichten senden oder empfangen. Die nachstehende Abbildung vermittelt davon einen Eindruck, ohne dass sie alle Möglichkeiten berücksichtigen kann.

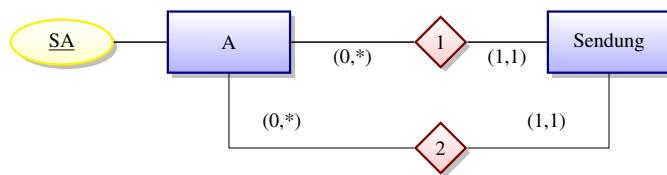
Es lassen sich beliebig strukturierte Netzwerke darstellen. Insbesondere können dies Polyhierarchien sein. In einer Polyhierarchie hat jedes Objekt keines, eines oder mehrere untergeordnete Objekte. Jedes Objekt kann keines, eines oder mehrere übergeordnete Objekte besitzen. Als Sonderfall sind natürlich auch Monohierarchien möglich, diese entarten aber mitunter zur Objektkette, die evtl. auch nur aus einem Element bestehen kann. Die Objektkette kann in sich geschlossen sein und wird dann zum Objekt-Zyklus, der im Minimalfall ein Ein-Objekt-Zyklus ist.



Der $(0,*)(0,*)$ Rekursiv-Beziehungstyp lässt sich in das physische Datenbankmodell nur nach vorheriger Umwandlung in einen neuen Objekttyp überführen. Diese Umwandlung erfolgt mit Hilfe eines Koppel-Objekttyps, der als „Sendung“ bezeichnet werden soll. Er repräsentiert die Sender-Empfänger-Beziehung zwischen den Objekten des Objekttyps A. Der Objekttyp A ist mit dem Objekttyp „Sendung“ durch zwei $(1,1):(0,*)$ Beziehungen zu verbinden.



wird umgewandelt in:

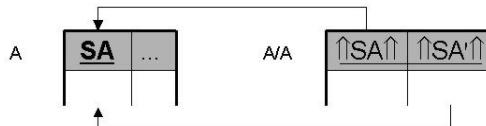


Stellt man den Objekttyp „Sendung“ im physischen Datenbankmodell als Koppel-Tabelle A/A dar, so enthält er den PK des Objekttyps A zweimal, als eingabepflichtige FK, die auf den Empfänger bzw. auf den Sender einer Nachricht verweisen. Beide FK sind für sich genommen nicht-unikal, so dass ein Objekt in mehreren Tabellenzeilen als Empfänger bzw. als Sender auftreten kann. Die Kombination dieser beiden FK stellt den PK der Koppel-Tabelle dar. Dieses Vorgehen ist in der Transformationsregel T13 dargestellt.

Transformationsregel T13 für den (0,*)(0,*) Rekursiv-Beziehungstyp

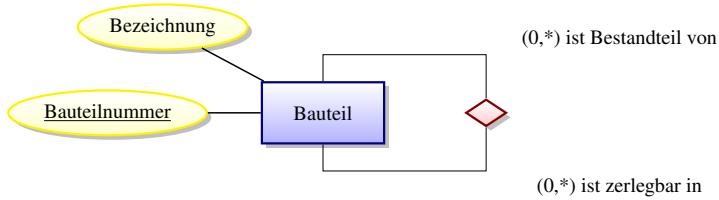
ER-Modell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
(0,*)(0,*) Rekursiv-Beziehungstyp	\Rightarrow	Koppel-Tabelle A/A, die <u>SA</u> in zwei Exemplaren, als eingabepflichtige, nicht-unikale Fremdschlüsselelemente enthält: als Empfänger-Fremdschlüssel $\uparrow\text{SA}\uparrow$ und als Sender-Fremdschlüssel $\uparrow\text{SA}'\uparrow$. Die Kombination beider Fremdschlüsselelemente bildet den Primärschlüssel von A/A.

Abb. 4.8:



Beispiel - komplexes Bauteil

Zur Veranschaulichung der Transformationsregel T13 betrachten wir eine Stückliste, die den Aufbau eines komplexen Bauteils - z.B. eines Autos - aus kleineren Bauteilen beschreibt. Ein Bauteil kann elementar sein, kann sich also nicht mehr in kleinere Bauteile zerlegen lassen (z.B. Schraube). Andere Bauteile lassen sich, wie z.B. der Motor, in kleinere Einzelteile zerlegen. Des Weiteren ist es möglich, dass ein gegebenes Bauteil nicht Bestandteil eines größeren Bauteils ist, wenn es nämlich beispielsweise das komplette Auto darstellt. Es kann aber auch Bestandteil mehrerer größerer Bauteile sein, wenn ein bestimmter Motor in mehrere Modelle eingebaut werden kann. Die erforderliche Transformation zeigt die folgende Abbildung.



Bauteil(Bauteilnummer, Bezeichnung)

BauteilBauteil(↑Grossteilnummer + Kleinteilnummer↑)

Jeder der beiden FK ist für sich nicht-unikal, so dass mehrere Bauteile jeweils als Groß teil bzw. Kleinteil in der Stückliste auftreten können. Es kann sein, dass eine bestimmte Bauteilnummer nicht als Wert des FK \uparrow Grossteilnummer \uparrow auftritt. Dann ist dieses Bauteil nicht weiter zerlegbar. Ist die Bauteilnummer nie Wert des FK \uparrow Kleinteilnummer \uparrow , dann gehört dies zu keinem größeren Bauteil.

Die Tabellentypbeschreibungen lassen allerdings wiederum einige Situationen zu, welche man im angegebenen Praxisfall eigentlich ausschließen möchte. Dazu gehören beispielsweise:

- *Mehr-Objekte-Zyklen*: Das Bauteil „1001“ hat als eines seiner Bestandteile das Bauteil „1002“, für das wiederum als eines seiner Bestandteile das Bauteil „1003“ angegeben ist. Für das Bauteil „1003“ ist aber angegeben, dass es das Bauteil „1001“ als Bestandteil enthält. Die Koppel-Tabelle „BauteilBauteil“ enthält dann folgende Zeilen.

Grossteilnummer	Kleinteilnummer
....
1001	1002
1002	1003
1003	1001
...	...

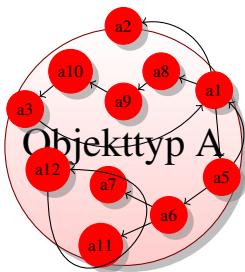
- *Ein-Objekt-Zyklen*: Ein Bauteil kann als sein eigenes Bestandteil ausgewiesen werden, wenn seine Bauteilnummer in einer Zeile der Koppel-Tabelle als Wert beider FK angegeben wird.
- *Mehr-Objekte-Ketten*: Ein Bauteil kann als Bestandteil nur ein einziges anderes Bauteil haben. Seine Nummer kommt dann in der Koppel-Tabelle „BauteilBauteil“ nur einmal als Wert des FK \uparrow Grossteilnummer \uparrow vor. Das ist eine unsinnige Konstruktion, denn ein Bauteil lässt sich entweder nicht zerlegen oder es ist mindestens in zwei Bestandteile zerlegbar.
- *Ein-Objekt-Ketten*: Die Typbeschreibungen lassen die Speicherung eines Bauteils zu, das weder zerlegbar, noch Bestandteil eines größeren Bauteils ist. Das wäre beispielsweise eine Schraube,

die in keinem weiteren größeren Bauteil Verwendung findet. Ihre Bauteilnummer taucht dann in der Koppel-Tabelle „BauteilBauteil“ an keiner Stelle auf, die Speicherung solcher Bauteile ist im betrachteten Zusammenhang jedoch sinnlos.

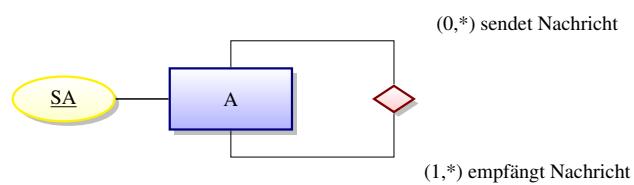
Die beschriebenen Situationen kann das Datenbankmanagementsystem nicht verhindern. Sie müssen durch entsprechende Anwendungsprogrammierung unterbunden werden.

4.5.6 Der (1,*):(0,*) Rekursiv-Beziehungstyp

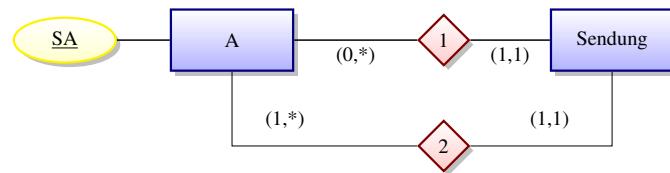
Beim (1,*):(0,*) Rekursiv-Beziehungstyp kann ein Objekt keine, eine oder mehrere Nachrichten senden, es muss aber stets mindestens eine Nachricht empfangen. Die folgende Abbildung vermittelt einen Eindruck von möglichen Beziehungsstrukturen, ohne alle Varianten abzudecken.



Der (1,*):(0,*) Rekursiv-Beziehungstyp lässt sich erst nach Umwandlung, mit Hilfe eines Koppel-Objekttyps „Sendung“, in das physische Modell überführen. Der Objekttyp A ist mit dem Objekttyp „Sendung“ durch einen (1,1):(0,*) und (1,1):(1,*) verbunden. Die folgende Abbildung zeigt das Schema der Umwandlung.



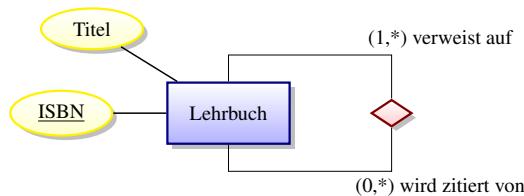
wird umgewandelt in:



Nun haben wir bei den binären Beziehungstypen gesehen, dass sich ein $(1,1):(1,*)$ Beziehungstyp im physischen Datenbankmodell nur als $(1,1):(0,*)$ Beziehungstyp repräsentieren lässt. Damit kann der $(1,*)(0,*)$ Rekursiv-Beziehungstyp nur gemäß der Transformationsregel T13 wie ein $(0,*)(0,*)$ Rekursiv-Beziehungstyp dargestellt werden.

Beispiel - Literaturverweis

Betrachten wir als Beispiel Literaturverweise in Lehrbüchern. Jedes Lehrbuch verweist auf mindestens ein Vorgänger-Lehrbuch. Auf ein gerade erst erschienenes Lehrbuch kann noch nicht verwiesen werden. Handelt es sich um ein interessantes Lehrbuch, so wird im Laufe der Zeit von vielen Nachfolge-Lehrbüchern zitiert.



Lehrbuch(ISBN, Titel)

LehrbuchLehrbuch(\uparrow VorgaengerISBN + NachfolgerISBN \uparrow)

Jeder der beiden FK ist für sich nicht-unikal, so dass ein gegebenes Lehrbuch mehrmals als Vorgänger bzw. als Nachfolger in Erscheinung treten kann. Ist eine Lehrbuch ISBN kein einziges Mal Wert des FK \uparrow VorgaengerISBN \uparrow , dann wurde dieses Lehrbuch noch nicht zitiert.

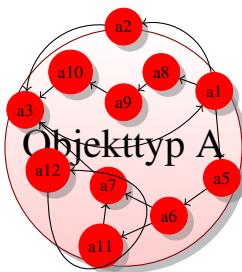
Die Tabellentypbeschreibungen lassen folgende Situationen zu, die nicht der Semantik entsprechen.

- *Vernachlässigung der Nichtoptionalität:* Die nicht-optionale Beziehungstyprichtung „Lehrbuch verweist auf Lehrbuch“ wird als optionale Beziehungstyprichtung repräsentiert. Es ist somit möglich, dass eine Lehrbuch ISBN nie als Wert des Fremdschlüssels \uparrow NachfolgerISBN \uparrow auftaucht. Dann ist dieses Lehrbuch nicht als Nachfolger eines anderen Lehrbuchs ausgewiesen, d.h. es verweist - entgegen der Praxisregel - auf kein Vorgängerlehrbuch.
- *Zyklen:* Ein Lehrbuch kann auf sich selbst als Vorgänger verweisen oder eine Objektkette kann sich schließen. Da Vorgänger-Verweise immer „in die Vergangenheit“ zeigen, sind Zyklen eigentlich verboten.

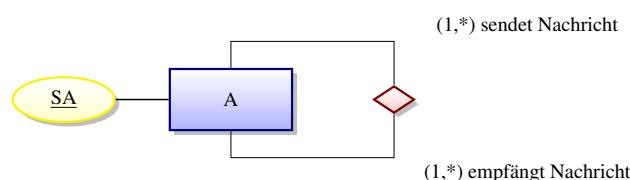
Die beschriebenen „pathologischen“ Situationen lassen sich nur durch die Anwendungsprogrammierung vermeiden.

4.5.7 Der $(1,*)(1,*)$: $(1,*)$ Rekursiv-Beziehungstyp

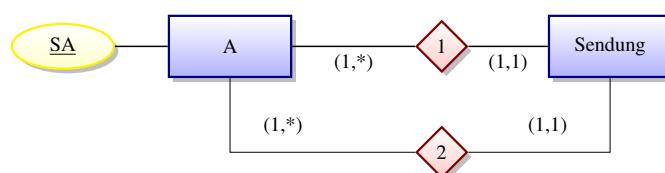
Der $(1,*)(1,*)$: $(1,*)$ Rekursiv-Beziehungstyp unterscheidet sich nur in einem Punkt, vom zuvor behandelten Rekursiv-Beziehungstyp $(1,*)(0,*)$. Ein Objekt muss mindestens eine Nachricht senden. Diese Forderung hat jedoch entscheidende Konsequenzen für die möglichen Beziehungsstrukturen, von denen in der folgenden Abbildung einige abgebildet sind.



Auch der $(1,*)(1,*)$: $(1,*)$ Rekursiv-Beziehungstyp lässt sich in das physische Modell, nur nach vorheriger Umwandlung in einen neuen Objekttyp, überführen. Der Objekttyp A ist mit dem Koppel-Objekttyp „Sendung“ durch zwei $(1,1):(1,*)$ Beziehungstypen verbunden.



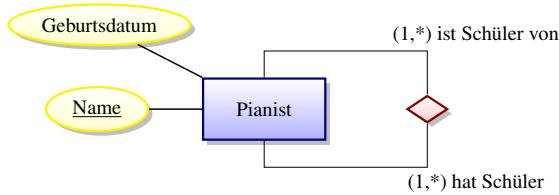
wird umgewandelt in:



Da sich ein $(1,1):(1,*)$ im physischen Modell nur als $(1,1):(0,*)$ Beziehungstyp repräsentieren lässt, muss der $(1,*)(1,*)$ Rekursiv-Beziehungstyp - unter Verlust von semantischen Informationen - gemäß der Transformationsregel T13 wie ein $(0,*)(0,*)$ Rekursiv-Beziehungstyp dargestellt werden.

Beispiel - Pianisten

Als Beispiel betrachten wir eine Kunsthochschule, die Informationen über Pianisten sammelt, die in der Ausbildung tätig sind. Wir nehmen der Einfachheit halber an, dass die Pianisten eindeutig durch ihren Namen unterschieden werden können. Jeder dieser Pianisten hat wenigstens einen anderen Pianisten als Schüler. Andererseits ist jeder Pianist Schüler eines oder mehrerer anderer Pianisten (Autodidakten werden nicht berücksichtigt).



Pianist(Name, Geburtsdatum)

PianistPianist(LehrerName + SchuelerName)

Jeder der beiden FK ist für sich nicht-unikal, so dass ein Pianist mehrmals als Lehrer bzw. Schüler in Erscheinung treten kann.

Die Tabellentypbeschreibungen ermöglichen Beziehungsstrukturen, die in der Praxis unzulässig sind, so beispielsweise:

- *Vernachlässigung der Nichtoptionalität:* Die beiden nicht-optionalen Beziehungstyprichtungen „Pianist hat als Schüler Pianist“ und „Pianist ist Schüler von Pianist“ werden als optionale Beziehungstyp-Richtungen repräsentiert. Es ist somit möglich, dass ein Pianist keine Schüler hat und dass ein Pianist nicht Schüler eines anderen Pianisten ist.
- *Zyklen:* Ein Pianist kann sein eigener Schüler sein (Ein-Objekt-Zyklus) oder eine Schüler-Kette kann zu ihrem Ausgangspunkt zurückkehren (Mehr-Objekte-Zyklus).

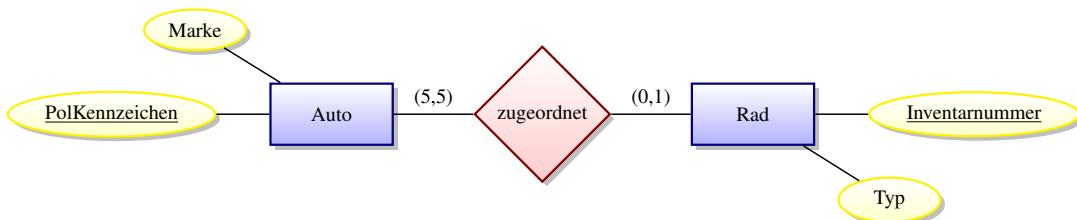
Diese unzulässigen Strukturen sind durch die Anwendungsprogrammierung zu verhindern.

4.6 Transformation von Kardinalitäts-Beschränkungen

Bei den Beziehungstypen gibt es im ER-Modell die Möglichkeit einschränkende Bedingungen für die Kardinalität anzugeben, wenn diese durch die „Geschäftsregeln“ der Realität vorgegeben sind. Nehmen wir

z.B. an, dass ein Unternehmen Informationen über die Dienstwagen und über die (Sommer- und Winter-) Räder speichern will. Jedes Auto ist mit genau 5 Rädern ausgerüstet (inklusive Ersatzrad). Einige Räder werden als Reserve im Lager aufbewahrt. Dies entspricht einem $(0,1):(1,*)$ Beziehungstyp, für den $(1,*)$ nicht beliebige Werte annehmen kann, sondern nur den Wert 5.

Wie wird diese Beschränkung in der Tabellentypbeschreibung berücksichtigt? Leider gar nicht! Es gibt im physischen Modell keine Möglichkeit die Kardinalität einzuschränken. Das gilt für den binären wie auch für den rekursiven oder ternären Beziehungstyp. Für die Repräsentation von Kardinalitätsbeschränkungen gilt also die negative Transformationsregel T14, welche im Folgenden dargestellt ist.



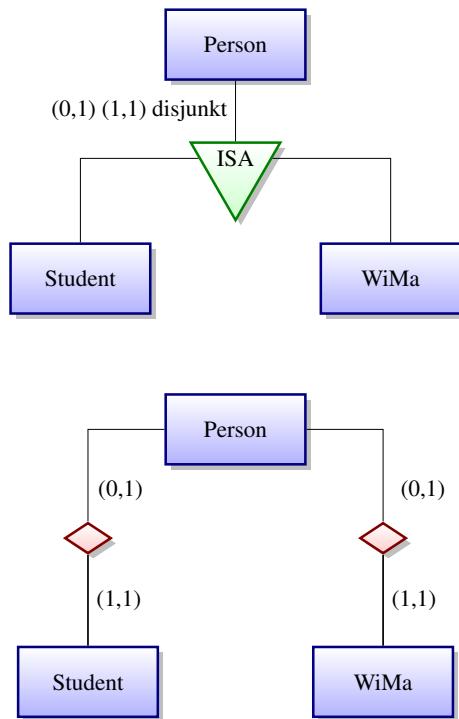
Transformationsregel T14 Kardinalitäts-Beschränkung

ER-Modell		physisches Datenmodell
Kardinalitäts-Beschränkung eines Beziehungstyps	\Rightarrow	Eine Beschränkung lässt sich nicht in der Typbeschreibung repräsentieren.

Auch wenn sich die Kardinalitäts-Beschränkungen nicht in das physische Modell transformieren lassen, sollte ihre Notierung im ER-Modell dennoch erfolgen, weil sie wichtige Geschäftsregeln darstellen. Diese müssen dann bei der Anwendungsprogrammierung berücksichtigt werden.

4.7 Transformation der Spezialisierung / Generalisierung

Die Transformation der Spezialisierung kann nicht in eigenständige Regeln gefasst werden, da der Einzelfall zu betrachten ist. Es sollen hier nur allgemeine Hinweise für die Überführung in das physische Modell gegeben werden, welche endgültig in Kombination mit den Transformationsregeln der binären Beziehungstypen erfolgt.



Die Transformation erfolgt mit T03 bzw. T04 und zusätzlichen programmtechnischen Maßnahmen, um die Disjunktion zu gewährleisten.

Teil II

SQL

5 Einstieg in SQL

Inhaltsangabe

5.1 Einführung

SQL¹ ist die auf dem Markt etablierte Manipulations- und Abfragesprache für relationale Datenbankmanagementsysteme. Es kam erstmals mit Oracle V2 1979 auf den Markt und wurde durch die beiden Institute ANSI (1986) und ISO (1987) standardisiert. 1989 wurden die Arbeitsergebnisse der beiden Gremien im „SQL-89“ bzw. „SQL-1“ Standard zusammengefasst. Eine grundlegende Überarbeitung erfolgte 1992 mit dem „SQL-2“ Standard, der auch als „SQL-92“ Standard bezeichnet wird, welcher im Wesentlichen auch heute noch Anwendung findet. Der aktuell gültige SQL-Standard ist der „SQL-2003“ Standard (SQL:2003 ISO/IEC 9075).

- **1989 - SQL-89-Standard, SQL-1-Standard**

- DML (Data Manipulation Language): SELECT, INSERT, UPDATE, DELETE
- DDL (Data Definition Language): Tabellen, Indizes, Sichten, GRANT/REVOKE
- Transaktionen: BEGIN, COMMIT, ROLLBACK
- Cursor

- **1992 - SQL-92-Standard, SQL-2-Standard**

- DDL (Data Definition Language): BLOB, VARCHAR, DATE, TIME, TIMESTAMP, BOOLEAN
- DML (Data Manipulation Language): INNER- und OUTER-Join, SET-Operatoren
- Transaktionen: SET TRANSACTION
- Cursor
- Constraints
- Systemtabellen

- **1999 - SQL-99-Standard, SQL-3-Standard**

- DML (Data Manipulation Language): Benutzerdefinierte Datentypen, Rollen
- DDL (Data Definition Language): Rekursive Abfragen

¹SQL = Structured Query Language

- * Intermediate Level: CASCADE DELETE
- * Full Level: CASCADE UPDATE
- Constraints: Trigger
- Call Level Interface: ODBC, JDBC, OLE DB
- Persistent Storage Modules: Stored Procedures

- **2003 - SQL-2003-Standard**

- DML (Data Manipulation Language): MERGE
- DDL (Data Definition Language): Identitätsattribute
- Schemata: Informations- und Definitions Schema
- SQL/XML: Einbettung von XML in die Datenbank
- Mediums: Zugriff auf externe Daten

Seit Ende der 80er-Jahr wird SQL, in Teilen, von nahezu allen relationalen Datenbankmanagementsystemen unterstützt, wobei jeder Hersteller seine eigenen Erweiterungen, zusätzlich zum Standard einfügt. Dies führt dazu, dass es eine Vielzahl von SQL-Dialekten gibt, welche sehr unterschiedlich sein können.

Die Syntax von SQL ist stark an die englische Sprache angelehnt und somit relativ einfach zu erlernen. Es stellt eine Reihe von Befehlen zur Verfügung, welche sich in vier Kategorien einteilen lassen:

- Abfragesprache (Query-Language)
- Datenmanipulationssprache (DML Data Manipulation Language)
- Datendefinitionssprache (DDL Data Definition Language)
- Datenkontrollsprache (DCL Data Control Language)

Durch die weitreichende Standardisierung von SQL, ist es möglich Anwendungsprogramme zu erstellen, welche eingeschränkt unabhängig vom verwendeten DBMS sind. Dies gilt zumindest für die Teile des SQL-Standards, die von allen Anbietern implementiert werden.

Im Gegensatz zu Programmiersprachen, wie z. B. Turbo Pascal, C++ oder Java, handelt es sich bei SQL um eine „Deklarative Programmiersprache“. Dies bedeutet, dass während in einer Sprache, wie C++, der *genaue Weg zur Lösung eines Problems* beschrieben wird, in SQL der Programmierer das *gewünschte Ergebnis so genau wie möglich* beschreiben muss. Den Weg dorthin findet SQL selbst, so dass sich der Programmierer nicht darum kümmern braucht, wie SQL zu seinem Ergebnis kommt.

Problematisch an diesem Ansatz kann sein, dass SQL immer ein Ergebnis liefert, solange die Syntax der Anweisung korrekt ist. Das gefundene Ergebnis muss jedoch nicht unbedingt das Gewollte sein und kann

sich, in sehr geringen Details, vom Richtigen unterscheiden, was eine Fehlersuche bzw. das Bemerkern eines Fehlers sehr schwierig gestaltet.

In diesem Skript werden Teile der beiden SQL-Dialekte, von Oracle 11g R2 und Microsoft SQL-Server 2008 R2 bzw. SQL Server 2012, anhand von praktischen Beispielen, näher erläutert.

5.2 Grundlegende Operationen in SQL

SQL basiert auf einer Relationalen Algebra. Im Sinne der Datenbanktheorie, ist dies eine formale Sprache, die es ermöglicht, Suchoperationen auf einem relationalen Schema durchzuführen. Mit ihrer Hilfe hat man die Möglichkeit, Relationen zu verknüpfen, zu reduzieren und komplexe Informationen zu ermitteln.

Die Relationale Algebra stellt verschiedene Operationen bereit, welche, mit Hilfe von SQL, umgesetzt werden können.

- Mengenoperationen
 - Vereinigung (UNION)
 - Schnittmenge (INTERSECT)
 - Differenz (MINUS)
- Projektion
- Selektion
- Kreuzprodukt (Kartesisches Produkt)
- Join

Für diese Unterrichtsunterlage hat die Relationale Algebra nur insofern eine Bedeutung, dass sie die theoretische Grundlage für die Sprache SQL darstellt.

5.2.1 Mengenoperationen

Die Vereinigung

Die Vereinigung ist eine Operation, bei der alle Zeilen zweier Relationen zu einer neuen Relation zusammengeführt werden. Dargestellt wird die Vereinigung in der Relationalen Algebra mit: $R_1 \cup R_2$ (R_1 vereinigt mit R_2).

Das folgende Beispiel veranschaulicht die Funktionsweise dieser Mengenoperation. Die beiden Relationen R_1 und R_2 werden miteinander vereinigt. Als Ergebnis entsteht eine neue Relation, die alle Zeilen der beiden zugrunde gelegten Relationen enthält, mit Ausnahme der Zeilen, welche redundant vorkommen. Dies betrifft hier die Zeile „1 2 3 4“, in der Relation R_2 . Eine gleiche Zeile existiert bereits in der Relation R_1 . Somit wird diese Zeile nur einmal im Ergebnis erscheinen.

Tabelle 5.1: R_1

A	B	C	D
1	2	3	4
5	6	7	8

Tabelle 5.2: R_2

A	B	C	D
0	9	8	7
6	5	4	3
1	2	3	4

Tabelle 5.3: $R_1 \cup R_2$

A	B	C	D
1	2	3	4
5	6	7	8
0	9	8	7
6	5	4	3



Redundante Zeilen werden bei der Vereinigung zweier Relationen R_1 und R_2 aus dem Ergebnis entfernt!

Die Schnittmenge

Die Schnittmenge enthält als Ergebnis nur die Zeilen, die in den beiden Relationen R_1 und R_2 identisch sind. Die Darstellung erfolgt in der Relationalen Algebra mit: $R_1 \cap R_2$ (R_1 geschnitten mit R_2).

Tabelle 5.4: R_1

A	B	C	D
1	2	3	4
5	6	7	8

Tabelle 5.5: R_2

A	B	C	D
0	9	8	7
6	5	4	3
1	2	3	4

Tabelle 5.6: $R_1 \cap R_2$

A	B	C	D
1	2	3	4

Da in diesem Beispiel nur die Zeile „1 2 3 4“ in beiden Relationen R_1 und R_2 vorhanden ist, wird nur diese in der Ergebnisrelation $R_1 \cap R_2$ abgebildet.

Die Differenz

Die Differenz zweier Relationen R_1 und R_2 entspricht den Zeilen, die nur in der linken der beiden Relationen vorkommen. D. h., die Differenz ist, wie in der Arithmetik auch, nicht kommutativ ($R_1 \setminus R_2 \neq R_2 \setminus R_1$).

Tabelle 5.7: R_1

A	B	C	D
1	2	3	4
5	6	7	8

Tabelle 5.8: R_2

A	B	C	D
0	9	8	7
6	5	4	3
1	2	3	4

Tabelle 5.9: $R_1 \setminus R_2$

A	B	C	D
5	6	7	8

Tabelle 5.10: $R_2 \setminus R_1$

A	B	C	D
0	9	8	7
6	5	4	3

5.2.2 Die Projektion

Die Projektion wählt Attribute aus einer Relation (Spalten aus einer Tabelle) aus, weshalb sie auch als *Attributbeschränkung* bezeichnet wird. Es ist dabei egal, ob alle Attribute oder nur eine Teilmenge der Attributmenge ausgewählt wird. In der Relationalen Algebra wird die Projektion mit $\pi_\beta(R_1)$ ausgedrückt, wobei β die Liste der gewählten Attribute bezeichnet. Hierzu ein paar Beispiele:

Tabelle 5.11: R_1

A	B	C	D
1	2	3	4
5	6	7	8

Tabelle 5.12: $\pi_{(A,B)}(R_1)$

A	B
1	2
5	6

Tabelle 5.13: $\pi_A(R_1)$

A
1
5

Tabelle 5.14: $\pi_*(R_1)$

A	B	C	D
1	2	3	4
5	6	7	8

5.2.3 Die Selektion

Die Selektion ermöglicht es ausgewählte Zeilen aus einer Relation in das Ergebnis aufzunehmen. Nicht erwünschte Zeilen werden einfach ausgeblendet. Dies geschieht mit einem „Selektionsausdruck“, einer einfachen Bedingung, die für jede einzelne Zeile geprüft wird. Mathematisch wird die Selektion als $\sigma_{\text{Ausdruck}}(R)$ dargestellt. σ (sigma) steht für Selektion.

Tabelle 5.15: R_1

A	B	C	D
1	2	3	4
5	6	7	8
9	0	1	2
3	4	5	6
7	8	9	0

Tabelle 5.17: $\sigma_{(B \geq 4)} \pi_{(A,B,C)}(R_1)$

Tabelle 5.16: $\sigma_{(A=3)} \pi_{(A,B)}(R_1)$

A	B
3	4

Tabelle 5.18: $\sigma_{(A>2 \wedge B \geq 4)} \pi_{*}(R_1)$

A	B	C
5	6	7
3	4	5
7	8	9

A	B	C	D
5	6	7	8
3	4	5	6
7	8	9	0

Es ist möglich, einen Selektionsausdruck zu definieren, der eine leere Menge \emptyset zum Ergebnis hat. Dies wäre z. B. bei folgendem Ausdruck der Fall: $\sigma_{(A=4)}(\pi_{*}(R_1))$. Da in der Spalte A in keiner Zeile der Wert 4 vorkommt, ist das Ergebnis eine leere Menge.

5.2.4 Kartesisches Produkt

Das Kartesische Produkt $R_1 \times R_2$ entspricht der Multiplikation aller Zeilen zweier Relationen R_1 und R_2 , sofern alle Attribute unterschiedlich sind. Daraus folgt, dass die Resultatrelation die Kombination aller Zeilen aus R_1 und R_2 umfasst.

Tabelle 5.21: $R_1 \times R_2$

Tabelle 5.19: R_1

A	B
1	2
5	6

Tabelle 5.20: R_2

C	D	E	F
0	9	8	7
6	5	4	3
1	2	3	4

A	B	C	D	E	F
1	2	0	9	8	7
1	2	6	5	4	3
1	2	1	2	3	4
5	6	0	9	8	7
5	6	6	5	4	3
5	6	1	2	3	4

5.2.5 Die Join-Operation im Allgemeinen

Das Wort Join bezeichnet zu deutsch einen Verbund. Im Sinne der Relationalen Algebra ist das der Verbund der beiden hintereinander ausgeführten Operationen „Kartesisches Produkt“ und „Selektion“.

Der Selektionsausdruck hat dabei die Form: $R_1 \theta R_2$, wobei θ (theta) einen geeigneten Vergleichsoperator ($=, \neq, >, <$ u. ä.) darstellt.

Die Relationale Algebra definiert den Join wie folgt:

$$R_1 \bowtie_{\text{Ausdruck}} R_2 := \{r_1 \cup r_2 \mid r_1 \in R_1 \wedge r_2 \in R_2 \wedge \text{Ausdruck}\}$$

Bedeutung:

- $R_1 \bowtie_{\text{Ausdruck}} R_2$: Verbund der beiden Relation R_1 und R_2
- $r_1 \cup r_2$: Vereinigung aller Zeilen aus R_1 und R_2 (Kreuzprodukt)
- $r_1 \in R_1$: r_1 ist eine Zeile aus der Relation R_1
- $r_2 \in R_2$: r_2 ist eine Zeile aus der Relation R_2
- \wedge : UND-Verknüpfung

Aus diesem Ausdruck lässt sich die folgende Formel ableiten:

$$R_1 \bowtie_{\text{Ausdruck}} R_2 := \sigma_{\text{Ausdruck}}(R_1 \times R_2)$$

5.2.6 Inner-Joins

Inner-Joins stellen die am häufigsten benötigte Form der Joins dar, so dass man hier von der „Standardform eines Joins“ sprechen könnte. Bei dieser Form des Joins wird das Kartesische Produkt zweier Relationen R_1 und R_2 gebildet und anschließend werden alle Zeilen eliminiert, die nicht der Join-Bedingung, auch Join-Prädikat genannt, genügen.

Es gibt zwei Unterarten des Inner-Join, den „Equi-Join“ und den „Non-Equi-Join“.

Equi-Join

Der Equi-Join ist eine spezielle Form des Inner-Join, der dadurch zustande kommt, dass im Join-Prädikat der Operator $=$ als Vergleichsoperator genutzt wird. Nur dann handelt es sich um einen Equi-Join.

Ein Beispiel:

$$\pi_{(R_1.A, R_1.B, R_2.C)}(R_1 \bowtie_{(R_1.A = R_2.A)} R_2 := \sigma_{(R_1.A = R_2.A)}(R_1 \times R_2))$$

Tabelle 5.22: R_1

A	B	C	D
1	2	3	4
5	6	7	8
9	0	1	2
3	4	5	6
7	8	9	0

Tabelle 5.23: R_2

A	B	C	D
9	8	7	6
4	5	2	3
1	0	9	8
6	4	5	7
0	1	2	3

Tabelle 5.24: $R_1 \bowtie R_2$

R1.A	R1.B	R2.C
1	2	9
9	0	7

In diesem Beispiel werden die Attributwerte der Spalten $R_1.A$ und $R_2.A$ miteinander verglichen und nur dort, wo eine Übereinstimmung gefunden wird, wird diese Zeile in die Ergebnisrelation aufgenommen.

Non-Equi-Join

Beim Non-Equi-Join handelt es sich um das Gegenteil zum Equi-Join. Sobald im Join-Prädikat nicht der $=$ -Operator genutzt wird, handelt es sich um einen Non-Equi-Join.

Ein Beispiel:

$$\pi_{R_2.*}(R_1 \bowtie_{(R_1.A > R_2.A \wedge R_1.B < R_2.B)} R_2)$$

Tabelle 5.25: R_1

A	B	C	D
1	2	3	4
5	6	7	8
9	0	1	2
3	4	5	6
7	8	9	0

Tabelle 5.26: R_2

A	B	C	D
9	8	7	6
5	4	3	2
1	0	9	8
7	6	5	4
3	2	1	0

Tabelle 5.27: $R_1 \bowtie R_2$

A	B	C	D
7	6	5	4
3	2	1	0

5.2.7 Outer-Joins

Outer-Joins unterscheiden sich dadurch von Inner-Joins, dass bei ihnen alle Zeilen, entweder der linken oder der rechten Join-Seite, in die Ergebnisrelation aufgenommen werden. Es gibt:

- Left-Outer-Join: Alle Zeilen der linken Join-Seite werden in die Ergebnisrelation aufgenommen.
- Right-Outer-Join: Alle Zeilen der rechten Join-Seite werden in die Ergebnisrelation aufgenommen.
- Outer-Join / Full-Outer-Join: Alle Zeilen beider Seiten werden in die Ergebnisrelation aufgenommen.

Nicht vorhandene Werte werden dabei durch sogenannte NULL-Werte aufgefüllt.

5.3 Auswahlabfragen mit SQL

5.3.1 Schema einer Auswahlabfrage

Die Struktur einer SQL-Abfrage ist sehr simpel und besteht aus maximal 6 Klauseln, die in der hier gezeigten Reihenfolge angegeben werden müssen.

```
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
```



Die kleinstmögliche SQL-Abfrage besteht aus den beiden Klausen **SELECT** und **FROM**.

Die **SELECT**-Klausel ist obligatorisch² und immer die erste in der Reihenfolge. Ihr folgt eine Liste von Attributen und Ausdrücken, sowie die **FROM**-Klausel. Letztere ist dafür zuständig, die „Datenquellen“ festzulegen, also die Tabellen, auf die sich die Abfrage bezieht.

Listing 5.1: Eine einfache Auswahlabfrage in Oracle

```
SELECT Vorname, Nachname
FROM Mitarbeiter;
```

In Beispiel ?? werden die Klauseln **SELECT** und **FROM** dazu benutzt, um die beiden Tabellenspalten VORNAME und NACHNAME, aus der Tabelle MITARBEITER abzufragen.

Das Ergebnis könnte sich so darstellen:



²obligatorisch = Zwingend notwendig

VORNAME	NACHNAME
Max	Winter
Sarah	Werner
Finn	Seifert
Sebastian	Schwarz
Tim	Sindermann
Peter	Müller
100 Zeilen ausgewählt	
Emily	Meier
Dirk	Peters
...	...
100 Zeilen ausgewählt	

Die Symbole, an der Seite der Ergebnistabellen, haben folgende Bedeutung:



So wird das Ergebnis in einer Oracle 11g R2 Datenbank dargestellt.



So wird das Ergebnis in einem Microsoft SQL Server 2008 R2 dargestellt.



Oracle und Microsoft SQL Server stellen das Ergebnis auf die gleiche Art und Weise dar.

Eine solche Abfrage lässt sich nun um beliebige Spalten erweitern.

Listing 5.2: Eine einfache Auswahlabfrage in Oracle

```
SELECT Vorname, Nachname, Geburtsdatum, SozVersNr
FROM Mitarbeiter;
```



VORNAME	NACHNAME	GEBURTSDATUM	SOZVERSNR
Max	Winter	31.08.88	D370941F-6CD-6C07977
Sarah	Werner	03.11.77	18FE2247-C53-7EAD1ED
Finn	Seifert	17.10.85	C973A840-B92-C5B97CD
Sebastian	Schwarz	27.06.92	E8F8587D-CBA-0F28A80
Tim	Sindermann	11.01.80	703838BD-E9C-BD118A6
100 Zeilen ausgewählt			

5.3.2 Der * als Joker

In verschiedenen Kartenspielen gibt es Karten, die als Joker dienen. Solche „Universalkarten“ können sich, in der richtigen Situation ausgespielt, als sehr nützlich erweisen. Auch in SQL gibt es einen Joker, den Stern *. Er ist immer dann dienlich, wenn man die Spaltenstruktur einer Tabelle nicht kennt oder einfach alle Spalten ausgeben möchte.

Listing 5.3: Der * als Joker

```
SELECT *
FROM   Bankfiliale;
```



BANKFILIALE_ID	STRASSE	HAUSNUMMER	PLZ	ORT
1	Poststraße	1	06449	Aschersleben
2	Markt	5	06449	Aschersleben
3	Goethestraße	4	39240	Calbe
4	Lessingstraße	1	06406	Bernburg
5	Schillerstraße	7	39240	Barby
6	Kirchstraße	8	39444	Hecklingen
7	Ringstraße	10	06420	Könnern
21 Zeilen ausgewählt				

Der Stern * dient, in der SELECT-Liste einer Auswahlabfrage, als Platzhalter, stellvertretend für alle Spalten einer Tabelle.

Sollen alle Spalten, bis auf eine einzige angezeigt werden, kann der Stern leider nicht weiterhelfen. Eine Formulierung wie „* -1“ ist nicht möglich. In einem solchen Fall müssen alle Spalten, bis auf die betreffende angegeben werden.

5.3.3 Arithmetische Ausdrücke

SQL ist, wie viele andere Programmiersprachen auch, in der Lage arithmetische Ausdrücke zu berechnen. Diese können nicht nur in der SELECT-Liste des SQL-Statements, sondern auch in verschiedenen anderen Klauseln, vorkommen. Die Syntax solcher Ausdrücke unterscheidet sich in Oracle und SQL Server nicht.

Tabelle 5.28: Arithmetische Operatoren in Oracle und SQL Server

(Operator)	(Bedeutung)
+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulo (Nur SQL-Server)
.	Dezimaltrennzeichen

Beispiel ?? zeigt ein einfaches Anwendungsbeispiel. Für die Entscheidung über die Gehaltserhöhung der Mitarbeiter, ist es notwendig, im Vorfeld eine Auswertung zu erstellen, die zeigt, wie sich die Erhöhung von 2,5 % auf die aktuellen Gehälter auswirkt.

Listing 5.4: Arithmetische Ausdrücke in SQL

```
SELECT Mitarbeiter_ID, Nachname, Gehalt, Gehalt * 1.025
FROM   Mitarbeiter;
```



MITARBEITER_ID	NACHNAME	GEHALT	GEHALT*1.025
1	Winter	88000	90200
2	Werner	50000	51250
3	Seifert	50000	51250
4	Schwarz	30000	30750
5	Sindermann	30000	30750
6	Müller	30000	30750

100 Zeilen ausgewählt



MITARBEITER_ID	NACHNAME	GEHALT	GEHALT*1.025
1	Winter	88000	90200
2	Werner	50000	51250
3	Seifert	50000	51250
4	Schwarz	30000	30750
5	Sindermann	30000	30750
6	Müller	30000	30750
100 Zeilen ausgewählt			



Oracle und SQL Server unterscheiden sich in der Anzeige von numerischen Werten. Oracle stellt Zahlen immer rechtsbündig dar. SQL Server zeigt dagegen alle Werte linksbündig an.

5.3.4 NULL Werte

Es kommt vor, dass nicht immer alle Attribute eines Datensatzes gefüllt sind, d. h. einige Attribute haben keinen Attributwert. Dies kann aus zwei Gründen der Fall sein:

- Der Wert eines Attributes ist zum Zeitpunkt der Eingabe des Datensatzes nicht bekannt (z. B. der Vorname einer Person ist nicht bekannt).
- Der Wert eines Attributs steht zum Zeitpunkt der Eingabe des Datensatzes noch nicht fest (z. B. das Sterbedatum einer Person bei der Erstellung der Geburtsurkunde).



Ein NULL Wert steht immer für einen unbekannten Wert und ist nicht mit der natürlichen Zahl 0 zu verwechseln.

NULL Werte haben insbesondere bei der Verwendung arithmetischer Ausdrücke eine große Bedeutung. Um dies zu demonstrieren, soll für alle Mitarbeiter deren Monatsgehalt berechnet werden. Dieses setzt sich aus dem Grundgehalt (Spalte GEHALT) und einer Provision (Spalten PROVISION) zusammen. Da nicht jeder Mitarbeiter eine Provision erhält, existieren NULL Werte in der Tabelle MITARBEITER.

Listing 5.5: NULL in arithmetischen Ausdrücken

```
SELECT Vorname, Nachname, Gehalt + Gehalt / 100 * Provision
FROM Mitarbeiter;
```

VORNAME	NACHNAME	GEHALT+GEHALT/100*PROVISION
Max	Winter	
Sarah	Werner	
...
Johannes	Lehmann	2400
Louis	Schmitz	2500
Martin	Schacke	1200

100 Zeilen ausgewählt



Einige Mitarbeiter erhalten keine Provision. Oracle und SQL Server unterscheiden sich bei der Anzeige der NULL Werte. Oracle zeigt für NULL Werte nichts an. SQL Server zeigt die Zeichenkette NULL an.

Und hier das Ergebnis für den MS SQL Server.



VORNAME	NACHNAME	(Kein Spaltenname)
Max	Winter	NULL
Sarah	Werner	NULL
Finn	Seifert	NULL
...
Johannes	Lehmann	2400
Louis	Schmitz	2500
Marie	Kipp	NULL
Amelie	Krüger	NULL
Martin	Schacke	1200
...
100 Zeilen ausgewählt		



Jeder arithmetische Ausdruck, in dem ein NULL Wert als Operand vorkommt, hat als Ergebnis den Wert NULL.

5.3.5 Verkettung von Zeichenketten

In manchen Fällen ist es notwendig, die Ausgabe einzelner Spalten miteinander zu verbinden. Dieser Vorgang wird als Konkatenation³ bezeichnet. Hierfür kennt Oracle den Operator || und SQL Server den Operator +.

Beispiel ?? wird nun dahingehend erweitert, dass die Gehaltserhöhung als Bericht angezeigt wird. Für jeden Mitarbeiter muss eine Zeile der Form „AAA hat ein Gehalt von XXX EUR und soll YYY EUR erhalten.“

³Verkettung von Zeichen oder Zeichenketten

Listing 5.6: Verkettung von Zeichenketten in Oracle

```
SELECT Nachname || ' hat ein Gehalt von ' || Gehalt ||
       ' EUR und soll ' || (Gehalt * 1.025) ||
       ' EUR erhalten.'
FROM   Mitarbeiter;
```



NACHNAME||'HATEINGEHALTVON'||GEHALT||'EURUNDSOL...'

Winter hat ein Gehalt von 88000 EUR und soll 90200 EUR erhalten.

...

Lehmann hat ein Gehalt von 2000 EUR und soll 2050 EUR erhalten.

Schmitz hat ein Gehalt von 2000 EUR und soll 2050 EUR erhalten.

Kipp hat ein Gehalt von 2000 EUR und soll 2050 EUR erhalten.

...

Listing 5.7: Verkettung von Zeichenketten in SQL Server

```
SELECT Nachname + ' hat ein Gehalt von '+ CAST(Gehalt AS VARCHAR(15)) +
       ' EUR und soll ' +
       CAST(Gehalt * 1.025 AS VARCHAR(15)) + ' erhalten.'
FROM   Mitarbeiter;
```



(Kein Spaltenname)

Winter hat ein Gehalt von 88000 EUR und soll 90200 EUR erhalten.

...

Lehmann hat ein Gehalt von 2000.00 EUR und soll ...

Schmitz hat ein Gehalt von 2000.00 EUR und soll ...

...

Schacke hat ein Gehalt von 1000.00 EUR und soll 1025 EUR erhalten.

...

100 Zeilen ausgewählt

An Beispiel ?? und Beispiel ?? sieht man sehr gut die unterschiedliche Reaktionsweise beider Datenbank Management Systeme. Oracle ist ohne weiteres in der Lage, Daten unterschiedlichen Typs (Zeichenketten, Zahl, Datums-werte, usw.) miteinander zu verknüpfen.

Bei Microsoft SQL Server ist dies nicht der Fall. Hier muss ein Vorgriff auf spätere Kapitel erfolgen. Der Zahlenwert der aus der Berechnung *gehalt * 1.025* resultiert, muss explizit in eine Zeichenkette umgewandelt werden, bevor die Verkettung funktioniert.

5.3.6 Spaltenaliasnamen

Bei der Anzeige des Ergebnisses einer Auswahlabfrage wird für jede Spalte eine Überschrift erzeugt. Oracle und Microsoft SQL Server sind sich dabei in sofern einig, als dass für die Spaltenüberschriften die Spaltenbezeichner der Tabellenspalten genutzt werden. Wird aber in der **SELECT**-Liste eine Konstante, eine Funktion oder ein anderer Ausdruck genutzt, scheiden sich die Geister. SQL Server zeigt in so einem Falle einfach gar keine Überschrift an, während Oracle einen Teil des Ausdrucks als Überschrift nutzt. Dieses Verhalten ist in den vorangegangenen Beispielen an einigen Stellen zu sehen.

Der SQL-Standard erlaubt es dieses Verhalten zu beeinflussen und manuell eine Spaltenüberschrift, einen sogenannten „Spaltenaliasnamen“, zu vergeben. Dies funktioniert im Falle von Oracle und SQL Server auf die gleiche Art und Weise, da sich hier beide an den Standard halten.

Listing 5.8: Vergabe eines Spaltenaliasnamen

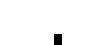
```
SELECT Vorname, Nachname, Gehalt + Gehalt / 100 * Provision AS "Neues Gehalt"
FROM Mitarbeiter;
```



VORNAME	NACHNAME	Neues Gehalt
...
Johannes	Lehmann	2400
Louis	Schmitz	2500
Marie	Kipp	
Amelie	Krüger	
Stefan	Beck	
Martin	Schacke	1200
...
100 Zeilen ausgewählt		

Der ANSI SQL-Standard sieht vier verschiedene Möglichkeiten zur Angabe eines Spaltenaliasnamen vor:

- [Ausdruck] **AS** "Spaltenaliasname"
- [Ausdruck] "Spaltenaliasname"
- [Ausdruck] Spaltenaliasname
- [Ausdruck] **AS** Spaltenaliasname



Oracle und MS SQL Server unterstützen alle vier Varianten. Die Angabe von Anführungszeichen ist nur dann notwendig, wenn im Spaltenaliasnamen Sonderzeichen oder Leerzeichen vorkommen.



Oracle wandelt einen Spaltenaliasnamen automatisch in Großbuchstaben um, es sei den, er wird in Anführungszeichen eingeschlossen.

5.4 Einige Konventionen zu SQL

Um die Lesbarkeit von SQL-Statements zu verbessern werden an dieser Stelle einige Konventionen genannt, die im praktischen Umgang mit SQL eingehalten werden sollten.

- Da SQL-Anweisungen mehrzeilig sein können erhält jede Klausel (SELECT, FROM, usw.) eine eigene Zeile.
- SQL ist nicht casesensitiv, d. h. Groß- und Kleinschreibung ist nicht relevant. Zur Verbesserung der Lesbarkeit sollten Schlüsselwörter groß geschrieben werden. Beispiele hierfür sind im gesamten Skript zu finden.
- Verwenden Sie Einrückungen zur Verbesserung der Lesbarkeit.

Listing 5.9: So nicht!

```
SELECT Nachname + ' hat ein Gehalt von ' +
CAST(Gehalt AS VARCHAR(15)) +
' EUR und soll ' +
CAST(Gehalt + Gehalt / 100 * Provision AS VARCHAR(15)) + ' erhalten.'
FROM Mitarbeiter;
```

Listing 5.10: Viel besser lesbar!

```
SELECT Nachname + ' hat ein Gehalt von ' + CAST(Gehalt AS VARCHAR(15)) +
' EUR und soll ' +
CAST(Gehalt + Gehalt / 100 * Provision AS VARCHAR(15)) + ' erhalten.'
FROM Mitarbeiter;
```

- Gemäß SQL-Standard muss jedes SQL-Statement mit einem Semikolon (;) abgeschlossen werden. Anwendungen wie der SQL*Developer, der JDeveloper oder das Management Studio verbergen diesen Sachverhalt jedoch.



In SQL*Plus muss jedes SQL-Statement mit ; oder / abgeschlossen werden!

6 Selektieren und Sortieren

Inhaltsangabe

6.1 Selektieren von Zeilen: Die WHERE-Klausel

Im vorangegangenen Kapitel wurde gezeigt, wie mit den beiden SQL-Klauseln `SELECT` und `FROM` der gesamte Inhalt einer Tabelle angezeigt werden kann. Zusätzlich zu diesen beiden Klauseln wird nun die optionale `WHERE`-Klausel eingeführt, die eine Selektion der Datensätze ermöglicht. Diese kann einen beliebig komplexen Ausdruck enthalten, der dann das „Auswahlkriterium“ für die Datensätze darstellt. Die Syntax der `WHERE`-Klausel lautet wie folgt:

Listing 6.1: Die WHERE-Klausel

```
WHERE <Ausdruck1> <Relationaler Operator> <Ausdruck2>
```



Der Begriff „Ausdruck“ steht in der Programmierung für ein auf einen Kontext bezogenes, auswertbares Gebilde. Bei *Ausdruck1* und *Ausdruck2* kann es sich beispielsweise um Spaltenbezeichner, Funktionsaufrufe, arithmetische Berechnungen, Konstanten usw. handeln.

Beispiel ?? zeigt insgesamt drei Ausdrücke:

- *<Ausdruck1>*
- *<Ausdruck2>*
- *<Ausdruck1> <Relationaler Operator> <Ausdruck2>*

Nicht nur *Ausdruck1* und *Ausdruck2* sind Ausdrücke, sondern auch die Verbindung beider, mittels eines Operators, wird als Ausdruck betrachtet.



Ein Operator ist ein mit einer Semantik belegtes Zeichen, dass eine genau definierte Operation darstellt. Operatoren werden meist in Gruppen eingeteilt, z. B. arithmetische Operatoren (+, -, *, /), relationale Operatoren, logische Operatoren, usw.

Tabelle ?? listet die in Oracle und MS SQL Server vorhandenen relationalen Operatoren auf.

6.1.1 Relationale Operatoren

Tabelle 6.1: Relationale Operatoren in Oracle und MS SQL Server

(Operator)	(Bedeutung)
=	Gleichheit
!=	Ungleichheit
<	Kleiner als
<=	Kleiner oder gleich
>	Größer als
>=	Größer oder gleich
LIKE	Ähnlichkeit zweier Zeichenketten
IN	Der linke Ausdruck befindet sich in einer Liste von Werten, die der rechte Ausdruck erzeugt.
IS NULL	Der linke Ausdruck liefert den Wert NULL zurück.
BETWEEN A AND B	Der Wert des linken Ausdrucks liegt zwischen den Wertgrenzen A und B. Die Wertgrenzen A und B werden in das Intervall mit einbezogen.

Numerische Werte vergleichen

Der Vergleich von numerischen Werten gestaltet sich sowohl in Oracle als auch im MS SQL Server sehr einfach.

Listing 6.2: Gleichheit zweier numerischer Werte

```
SELECT Vorname, Nachname
FROM Mitarbeiter
WHERE Mitarbeiter_ID = 5;
```

VORNAME	NACHNAME
Tim	Sindermann

1 Zeile ausgewählt



Listing 6.3: Wert A größer oder gleich Wert B

```
SELECT Vorname, Nachname, Gehalt
FROM Mitarbeiter
WHERE Mitarbeiter_ID >= 50;
```



VORNAME	NACHNAME	GEHALT
Emilia	Köhler	2500
Karolin	Klingner	2000
Chris	Roggatz	3000
Christian	Haas	2000
Jessica	Winkler	2000
Anna	Keller	2500
Johannes	Klingner	2500
Emma	Krüger	3500

51 Zeilen gewählt

Listing 6.4: Prüfen eines Intervalls

```
SELECT Mitarbeiter_ID, Vorname, Nachname
FROM Mitarbeiter
WHERE Mitarbeiter_ID BETWEEN 5 AND 9;
```



MITARBEITER_ID	VORNAME	NACHNAME
5	Tim	Sindermann
6	Peter	Müller
7	Emily	Meier
8	Dirk	Peters
9	Louis	Winter

5 Zeilen gewählt

Listing 6.5: Alle Zeilen aus einer Wertemenge anzeigen

```
SELECT Mitarbeiter_ID, Vorname, Nachname
FROM Mitarbeiter
WHERE Mitarbeiter_ID IN (5, 7, 9);
```



MITARBEITER_ID	VORNAME	NACHNAME
5	Tim	Sindermann
7	Emily	Meier
9	Louis	Winter

3 Zeilen gewählt

Zeichenketten vergleichen

Der Vergleich zweier Zeichenketten bringt, im Gegensatz zum Vergleich numerischer Werte, eine Schwierigkeit mit sich. Abhängig vom benutzten RDBMS¹ werden Zeichenkettenvergleiche casesensitiv oder incasesensitiv durchgeführt. In Oracle beispielsweise ist „Oracle“ ungleich „oracle“ oder „ORACLE“ ungleich „OrAcLe“. Der MS SQL Server hingegen verhält sich nicht casesensitiv. Für ihn sind alle vier Werte gleich.

Listing 6.6: Ein einfacher Zeichenkettenvergleich

```
SELECT Mitarbeiter_ID, Vorname, Nachname
FROM   Mitarbeiter
WHERE  Nachname = 'Scholz';
```

MITARBEITER_ID	VORNAME	NACHNAME
96	Johanna	Scholz

1 Zeile ausgewählt



Im nächsten Beispiel wird eine ähnliche `WHERE`-Klausel verwendet, wie in Beispiel ??, sie führt jedoch zu einem ganz anderen Ergebnis.



In SQL müssen Zeichenketten in Hochkommas ' eingeschlossen werden! Diese dürfen nicht mit den Akzent-Zeichen verwechselt werden!

Listing 6.7: Ein einfacher Zeichenkettenvergleich

```
SELECT Mitarbeiter_ID, Vorname, Nachname
FROM   Mitarbeiter
WHERE  Nachname = 'SCHOLZ';
```

Keine Zeilen ausgewählt!



Da die Oracle-Datenbank casesensitiv arbeitet, ist „SCHOLZ“ ungleich „Scholz“. Somit werden keine Datensätze gefunden. Der MS SQL Server hat hier keine Schwierigkeiten. Ihn stört die unterschiedliche Schreibweise der Zeichenketten nicht, weshalb er das gewünschte Ergebnis anzeigt.

¹RDBMS = Relationales Datenbank Management System



MITARBEITER_ID	VORNAME	NACHNAME
96	Johanna	Scholz

1 Zeile ausgewählt

Zeichenketten vergleichen mit LIKE

Ist es notwendig nach einem Zeichenmuster zu suchen, wie z. B. *Alle Mitarbeiter, deren Nachname mit „Sch“ beginnt*, so kann dies mit dem **LIKE**-Operator geschehen.

Listing 6.8: Zeichenkettensuche mit einem Suchmuster

```
SELECT Mitarbeiter_ID, Vorname, Nachname
FROM   Mitarbeiter
WHERE  Nachname LIKE 'Sch%';
```



MITARBEITER_ID	VORNAME	NACHNAME
4	Sebastian	Schwarz
11	Sophie	Schwarz
25	Elias	Schreiber
29	Louis	Schmitz
33	Martin	Schacke
36	Hans	Schumacher

10 Zeilen ausgewählt

Der **LIKE**-Operator nutzt zwei Wildcards, um Suchmuster für Zeichenketten zu erstellen.

Tabelle 6.2: Wildcards des LIKE-Operators

(Wildcard)	(Bedeutung)
%	(Prozentzeichen) Null, eines oder beliebig viele Zeichen
_	(Unterstrich) Genau ein Zeichen

Für Beispiel **??** bedeutet dies: *Die ersten drei Zeichen des Suchmusters sind S, c und h. Nach dem h können null, eines oder beliebig viele andere Zeichen stehen*. Im nächsten Beispiel wird die **_**-Wildcard benutzt, um alle Mitarbeiter zu suchen, deren Nachname an der dritten Stelle ein kleines g trägt.

Listing 6.9: Zeichenkettensuche mit einem etwas komplexeren Suchmuster

```
SELECT Mitarbeiter_ID, Vorname, Nachname
FROM   Mitarbeiter
WHERE  Nachname LIKE '_ _g%';
```

MITARBEITER_ID	VORNAME	NACHNAME
37	Louis	Wagner
52	Chris	Roggatz
83	Peter	Roggatz
88	Joachim	Wagner

4 Zeilen ausgewählt

Die ersten beiden Zeichen des Suchmusters sind `_` Unterstriche, d. h. an der ersten und zweiten Stelle der gesuchten Zeichenketten **muss** sich jeweils genau ein Zeichen befinden. Das dritte Zeichen ist mit dem `g` genau definiert. Anschließend können wieder null, eines oder beliebig viele andere Zeichen stehen.



Der LIKE-Operator verwendet die beiden Wildcards `%` und `_`. `%` steht für null, eines oder beliebig viele Zeichen. `_` steht für genau ein Zeichen.

Vergleiche mit NULL-Werten

Sowohl Oracle, als auch der MS SQL Server kennen beide den Operator `IS NULL`. Mit seiner Hilfe können Spalten auf NULL-Werte hin überprüft werden. Sollen z. B. alle Mitarbeiter, die keinen Vorgesetzten haben, angezeigt werden, wird ein Vergleich mit dem `IS NULL`-Operator angestellt.

Listing 6.10: Der IS NULL Operator

```
SELECT Mitarbeiter_ID, Vorname, Nachname, Vorgesetzter_ID
FROM   Mitarbeiter
WHERE  Vorgesetzter_ID IS NULL;
```

MITARBEITER_ID	VORNAME	NACHNAME	VORGESETZTER_ID
1	Max	Winter	
1 Zeile ausgewählt			

Da es in diesem Beispiel keinen wesentlichen Unterschied bei der Ergebnisanzeige zwischen Oracle und SQL Server gibt (SQL Server zeigt das Wort `NULL` für NULL-Werte und alle Spaltenwerte linksbündig an), wurde hier auf ein getrenntes Abdrucken der Ergebnisse verzichtet.

Das Gegenstück zum `IS NULL`-Operator, ist der `IS NOT NULL`-Operator.



Wird ein Ausdruck, mit Hilfe des Gleichheitsoperators (=), mit dem Wert NULL verglichen, ist das Ergebnis des Vergleichs immer NULL!

6.1.2 Logische Verknüpfung von Ausdrücken

In vielen Fällen ist es notwendig komplexe Ausdrücke zu formulieren, indem mehrere Ausdrücke miteinander verknüpft werden. Eine solche Verknüpfung geschieht unter Zuhilfenahme der logischen Operatoren *AND*, *OR* und *NOT*.

Logische Verknüpfungen mit AND

Der logische Operator *AND* verknüpft zwei Bedingungen miteinander und liefert ein wahres Ergebnis, sobald beide Ausdrücke ein wahres Ergebnis haben. Die Logikabelle Tabelle ?? zeigt die möglichen Ergebnisse einer AND-Verknüpfung.

Tabelle 6.3: Der logische Operator AND

Aussagen		(Wahr)	(Falsch)
Wahr	w	f	
Falsch	f	f	

In Beispiel ?? wird gezeigt, wie der *AND*-Operator dazu genutzt werden kann, um zwei Bedingungen miteinander zu verknüpfen. Es sollen alle Mitarbeiter angezeigt werden, deren Gehalt unter 1.500 EUR liegt und die in der Bankfiliale Nummer zwei arbeiten.

Listing 6.11: Der AND Operator

```
SELECT Vorname, Nachname, Gehalt, Bankfiliale_ID
FROM Mitarbeiter
WHERE Gehalt < 1500
AND Bankfiliale_ID = 2;
```



VORNAME	NACHNAME	GEHALT	BANKFILIALE_ID
Martin	Schacke	1000	2
2 Zeilen ausgewählt			
Oliver	Wolf	1000	2
2 Zeilen ausgewählt			

Logische Verknüpfungen mit OR

Der logische Operator *OR* liefert, im Unterschied zu *AND*, ein wahres Ergebnis, sobald mindestens einer der beiden Ausdrücke ein wahres Ergebnis hat.

Tabelle 6.4: Der logische Operator OR

	Aussagen	(Wahr)	(Falsch)
Wahr	w	w	
Falsch	w	f	

Wird in Beispiel ?? der Operator *AND* durch ein *OR* ersetzt, verändert sich die Ergebnismenge. Es werden jetzt alle Mitarbeiter angezeigt, die entweder ein Gehalt unter 1.500 EUR haben oder die in Bankfiliale Nummer zwei arbeiten.

Listing 6.12: Der OR Operator

```
SELECT Vorname, Nachname, Gehalt, Bankfiliale_ID
FROM Mitarbeiter
WHERE Gehalt < 1500
    OR Bankfiliale_ID = 2;
```



VORNAME	NACHNAME	GEHALT	BANKFILIALE_ID
Louis	Winter	12000	2
Stefan	Beck	1500	2
Martin	Schacke	1000	2
Max	Oswald	1500	2
Oliver	Wolf	1000	2
Hans	Schumacher	1000	3
Maja	Keller	1000	5
Elias	Sindermann	1000	8
Jonas	Meier	1000	12

9 Zeilen ausgewählt

Aussagen mit NOT umkehren

Die Bedeutung des Operators *NOT* ist sehr einfach zu umschreiben. Er kehrt ein Ergebnis um. Aus einem wahren Ergebnis wird ein falsches und umgekehrt. Dieser Effekt ist auch mit *IS NULL* und *IS NOT NULL* zu sehen. In Beispiel ?? werden alle Mitarbeiter angezeigt, deren Gehalt kleiner als 1.500 EUR ist und die nicht in der Bankfiliale Nummer zwei arbeiten.

Listing 6.13: Der NOT Operator

```
SELECT Vorname, Nachname, Gehalt, Bankfiliale_ID
FROM Mitarbeiter
WHERE Gehalt < 1500
    AND NOT Bankfiliale_ID = 2;
```

VORNAME	NACHNAME	GEHALT	BANKFILIALE_ID
Hans	Schumacher	1000	3
Ma ja	Keller	1000	5
Elias	Sindermann	1000	8
Jonas	Meier	1000	12

4 Zeilen ausgewählt



Die Klammern (und) haben Einfluss auf die Bedeutung von Ausdrücken. Werden mehrere logische Operatoren kombiniert, kann so die Lesbarkeit von Ausdrücken verbessert oder deren Bedeutung verändert werden.

6.2 Festlegen einer Sortierung

In allen vorangegangenen Beispielen war die Reihenfolge der Ausgabe der Datensätze unbestimmt. Sowohl Oracle als auch Microsoft SQL Server geben die Datensätze immer in der Reihenfolge aus, in der sie in der Quelltabelle vorliegen. Soll eine sortierte Ausgabe erfolgen, muss dies mit Hilfe der in Beispiel ?? gezeigten **ORDER BY**-Klausel geschehen. Dazu muss diese, mit Sortierbegriffen versehen, am Ende des SQL-Statements angegeben werden.

6.2.1 Die ORDER BY Klausel

Listing 6.14: Die ORDER BY Klausel

```
ORDER BY <Sortierbegriff 1> [asc|desc],
         <Sortierbegriff 2> [asc|desc],
         <Sortierbegriff n> [asc|desc] ...
```

Als Sortierbegriffe können Spaltenbezeichner, Spaltenaliasnamen, berechnete Ausdrücke und auch Spaltenpositionsangaben, bezogen auf die Reihenfolge der Spaltennamen in der SELECT-Liste, genutzt werden. Beispiel ?? und Beispiel ?? zeigen die Anwendung der **ORDER BY**-Klausel.



Werden mehrere Sortierbegriffe angegeben, wird die Sortierung von links nach rechts durchgeführt. Das bedeutet, dass zuerst nach dem äußerst linken Sortierbegriff sortiert wird und anschließend wird, innerhalb dieser Sortierung, jeder weitere Sortierbegriff angewandt. Die Sortierungen werden also ineinander geschachtelt.

Listing 6.15: Die ORDER BY Klausel mit Spaltenbezeichnern

```
SELECT Vorname, Nachname, Gehalt, Bankfiliale_ID
FROM Mitarbeiter
WHERE Gehalt <= 1500
ORDER BY Gehalt;
```



VORNAME	NACHNAME	GEHALT	BANKFILIALE_ID
Oliver	Wolf	1000	2
Hans	Schumacher	1000	3
Maja	Wolf	1000	5
Elias	Sindermann	1000	8
Jonas	Meier	1000	12
Martin	Schacke	1000	2
Max	Oswald	1500	2
Stefan	Beck	1500	2

18 Zeilen ausgewählt

Listing 6.16: Die ORDER BY Klausel mit Positionsangaben

```
SELECT Vorname, Nachname, Gehalt, Bankfiliale_ID
FROM Mitarbeiter
WHERE Gehalt <= 1500
ORDER BY 3, 2;
```



VORNAME	NACHNAME	GEHALT	BANKFILIALE_ID
Maja	Keller	1000	5
Jonas	Meier	1000	12
Martin	Schacke	1000	2
Hans	Schumacher	1000	3
Elias	Sindermann	1000	8
Oliver	Wolf	1000	2
Stefan	Beck	1500	2
Georg	Dühning	1500	20

18 Zeilen ausgewählt



Bei der Benutzung von Positionsangaben (siehe Beispiel ??) muss darauf geachtet werden, dass sich diese auf die Reihenfolge der Spaltenbezeichner in der SELECT-Liste beziehen. Wird die SELECT-Liste später verändert, müssen unter Umständen auch die Positionsangaben angepasst werden.

6.2.2 Auf- und absteigendes Sortieren

Zu jedem Suchbegriff können die beiden Kürzel **ASC** und **DESC** mit angegeben werden. **ASC**² bewirkt aufsteigende Sortierung (Standard) und **DESC**³ absteigende Sortierung. Beispiel ?? zeigt, wie sich das Ergebnis durch die absteigende Sortierung der Spalte GEHALT verändert.

Listing 6.17: Die ORDER BY Klausel mit absteigender Sortierung

```
SELECT      Vorname , Nachname , Gehalt , Bankfiliale_ID
FROM        Mitarbeiter
WHERE       Gehalt <= 1500
ORDER BY    Gehalt DESC , 2 ASC;
```

VORNAME	NACHNAME	GEHALT	BANKFILIALE_ID
Stefen	Beck	1500	2
Georg	Dühning	1500	20
Tom	Fischer	1500	17
Jannis	Friedrich	1500	14
Maximilian	Hahn	1500	13
Lena	Hermann	1500	4
Anne	Huber	1500	10

18 Zeilen ausgewählt



Eine in der **ORDER BY**-Klausel als Sortierbegriff genutzte Spalte, muss nicht in der SELECT-Liste der Abfrage vorhanden sein.



²engl. Ascending = aufsteigend

³engl. Descending = absteigend

6.3 Übungen - Selektieren und Sortieren

1. Erstellen Sie eine Abfrage, die die Konto_ID und das aktuelle Guthaben des Girokontos der Bankkunden anzeigt, die weniger als 1000 EUR Guthaben besitzen.



KONTO_ID	GUTHABEN
15	-10496,4
16	-54593,2
43	-42144,1
48	-140505,1
57	-1088,4
59	760,1
83	336,2
87	-9009,1
99	-69705,6

55 Zeilen ausgewählt

2. Erstellen Sie eine Abfrage, die die Mitarbeiter_ID und den Nachnamen der Mitarbeiter mit der Vorgesetzter_ID „2“ anzeigt.



MITARBEITER_ID	NACHNAME
4	Schwarz
5	Sindermann

2 Zeilen ausgewählt

3. Erstellen Sie eine Abfrage, die die Konto_ID und das aktuelle Guthaben des Girokontos der Bankkunden anzeigt, deren Guthaben nicht zwischen 1000 EUR und 1500 EUR liegt.



KONTO_ID	GUTHABEN
1	111316,9
2	96340,2
3	59633
5	98449
6	26130,7
7	23128,7
9	8857,6
10	68001,3

428 Zeilen ausgewählt

4. Lassen Sie sich die Kunden_ID und das Geburtsdatum aller Eigenkunden anzeigen, die zwischen dem 20. Februar 1980 und dem 02. März 1988 geboren sind. Zusätzlich soll die Abfrage nach dem Geburtsdatum in aufsteigender Reihenfolge sortiert werden.



KUNDEN_ID	GEBURTS DATUM
391	01.03.80
387	03.03.80
339	22.03.80
75	22.03.80
458	07.05.80
50	21.05.80

124 Zeilen ausgewählt

5. Zeigen Sie, in alphabetischer Reihenfolge, die Mitarbeiter_ID und den Nachnamen der Mitarbeiter an, die die Vorgesetzter_ID „5“ oder „6“ haben.



MITARBEITER_ID	NACHNAME
17	Becker
16	Berger
20	Große
13	Kaiser
18	Köhler
14	Lorenz
22	Rollert

10 Zeilen ausgewählt

6. Erstellen Sie eine Abfrage, die den Nachnamen und die Bankfiliale_ID der Mitarbeiter ausgibt, die die Vorgesetzten_ID „5“ oder „6“ haben und deren Bankfiliale_ID zwischen „10“ und „20“ ist. Die Spalten sollen mit „Mitarbeiter“ und „Bankfiliale“ benannt werden.



MITARBEITER	BANKFILIALE
Becker	10
Köhler	11
Weber	12
Große	13
Walther	14

6 Zeilen ausgewählt

7. Zeigen Sie die Mitarbeiter_ID und den Nachnamen des Mitarbeiters an, der keinen Vorgesetzten hat.



MITARBEITER_ID	NACHNAME
1	Winter

1 Zeile ausgewählt

8. Zeigen Sie die Kunden_ID und das Geburtsdatum derjenigen Eigenkunden an, die im Jahre 1980 geboren sind.



KUNDEN_ID	GEBURTSDATUM
387	03.03.80
538	22.01.80
161	17.08.80
254	12.09.80

14 Zeilen ausgewählt

9. Erstellen Sie eine Abfrage, die den Nachnamen, das Gehalt und die Provision für alle Mitarbeiter anzeigt, die eine Provision erhalten. Sortieren Sie die Ausgabe in absteigender Reihenfolge nach dem Gehalt.



NACHNAME	GEHALT	PROVISION
Hartmann	4000	30
Roth	3500	20
Walther	3500	20
Wagner	3500	20
Zimmermann	3500	30

33 Zeilen ausgewählt

10. Zeigen Sie die Nachnamen aller Mitarbeiter an, in deren Nachname an dritter Stelle ein „a“ vor kommt.



NACHNAME
Haas
Haas
Krause
Krause

4 Zeilen ausgewählt

11. Zeigen Sie die Nachnamen aller Mitarbeiter an, deren Nachname ein kleines „a“ und ein kleines „e“ enthält.



NACHNAME
Sindermann
Kaiser
Zimmermann
Walther
Neumann
Lehmann
24 Zeilen ausgewählt



6.4 Lösungen - Selektieren und sortieren

1. Erstellen Sie eine Abfrage, die die Konto_ID und das aktuelle Guthaben des Girokontos der Bankkunden anzeigt, die weniger als 1000 EUR Guthaben besitzen.



```
SELECT Konto_ID, Guthaben
FROM   Girokonto
WHERE  Guthaben < 1000;
```

2. Erstellen Sie eine Abfrage, die die Mitarbeiter_ID und den Nachnamen der Mitarbeiter mit der Vorgesetzter_ID „2“ anzeigt.



```
SELECT Mitarbeiter_ID, Nachname
FROM   Mitarbeiter
WHERE  Vorgesetzter_ID = 2;
```

3. Erstellen Sie eine Abfrage, die die Konto_ID und das aktuelle Guthaben des Girokontos der Bankkunden anzeigt, deren Guthaben nicht zwischen 1000 EUR und 1500 EUR liegt.



```
SELECT Konto_ID, Guthaben
FROM   Girokonto
WHERE  Guthaben NOT BETWEEN 1000 AND 1500;
```

4. Lassen Sie sich die Kunden_ID und das Geburtsdatum aller Eigenkunden anzeigen, die zwischen dem 20. Februar 1980 und dem 02. März 1988 geboren sind. Zusätzlich soll die Abfrage nach dem Geburtsdatum in aufsteigender Reihenfolge sortiert werden.



```
SELECT Kunden_ID, Geburtsdatum
FROM   Eigenkunde
WHERE  Geburtsdatum BETWEEN '20.02.1980' AND '02.03.1988'
ORDER BY Geburtsdatum;
```

5. Zeigen Sie, in alphabetischer Reihenfolge, die Mitarbeiter_ID und den Nachnamen der Mitarbeiter an, die die Vorgesetzter_ID „5“ oder „6“ haben.



```
SELECT Mitarbeiter_ID, Nachname
FROM Mitarbeiter
WHERE Vorgesetzter_ID IN (5, 6)
ORDER BY Nachname;
```

6. Erstellen Sie eine Abfrage, die den Nachnamen und die Bankfiliale_ID der Mitarbeiter ausgibt, die die Vorgesetzten_ID „5“ oder „6“ haben und deren Bankfiliale_ID zwischen „10“ und „20“ ist. Die Spalten sollen mit „Mitarbeiter“ und „Bankfiliale“ benannt werden.



```
SELECT Nachname AS "Mitarbeiter", Bankfiliale_ID AS "Bankfiliale"
FROM Mitarbeiter
WHERE Vorgesetzter_ID IN (5, 6)
AND Bankfiliale_ID BETWEEN 10 AND 20;
```

7. Zeigen Sie die Mitarbeiter_ID und den Nachnamen des Mitarbeiters an, der keinen Vorgesetzten hat.



```
SELECT Mitarbeiter_ID, Nachname
FROM Mitarbeiter
WHERE Vorgesetzter_ID IS NULL;
```

8. Zeigen Sie die Kunden_ID und das Geburtsdatum derjenigen Eigenkunden an, die im Jahre 1980 geboren sind.



```
SELECT Kunden_ID, Geburtsdatum
FROM Eigenkunde
WHERE Geburtsdatum BETWEEN '01.01.1980' AND '31.12.1980';
```

9. Erstellen Sie eine Abfrage, die den Nachnamen, das Gehalt und die Provision für alle Mitarbeiter anzeigt, die eine Provision erhalten. Sortieren Sie die Ausgabe in absteigender Reihenfolge nach dem Gehalt.



```
SELECT Nachname, Gehalt, Provision
FROM Mitarbeiter
WHERE Provision IS NOT NULL
ORDER BY Gehalt DESC;
```

10. Zeigen Sie die Nachnamen aller Mitarbeiter an, in deren Nachname an dritter Stelle ein „a“ vor kommt.



```
SELECT Nachname  
FROM Mitarbeiter  
WHERE Nachname LIKE '_ _a%';
```

11. Zeigen Sie die Nachnamen aller Mitarbeiter an, deren Nachname ein kleines „a“ und ein kleines „e“ enthält.

```
SELECT Nachname  
FROM Mitarbeiter  
WHERE Nachname LIKE '%a%'  
AND Nachname LIKE '%e%';
```



7 Funktionen

Inhaltsangabe

7.1 Der Begriff der Funktion



Eine Funktion ist eine Rechenvorschrift (ein kleines Programm), welches zu einer Menge von Eingabewerten eine Ergebnismenge (Ausgabewert) erzeugt. Die Eingabewerte werden als sogenannte Parameter/Argumente an die Funktion übergeben.

Bildlich kann man sich eine Funktion vorstellen, wie eine Maschine, an deren einem Ende etwas zugeführt wird und am Anderen entsteht ein Produkt (Ergebnis). SQL kennt zwei unterschiedliche Arten von Funktionen:

- **Single Row Functions / Skalare Funktionen:** Sie werden immer nur auf einen einzelnen Attributwert angewandt und müssen somit für jede Ergebnissezeile einmal ausgeführt werden.
- **Grouping Functions / Aggregatfunktionen:** Diese Art von Funktionen wird pro Abfrage nur einmal ausgeführt und bezieht sich immer auf eine Gruppe von Werten (siehe Abschnitt ??).

Die Gruppe der Single Row Functions (in SQL Server werden diese Funktionen als „Skalare Funktionen“ bezeichnet) wird wiederum in mehrere Arten unterteilt.

- **Zeichenkettenfunktionen:** Dieser Funktionstyp findet Anwendung auf Zeichenketten. Mit ihm kann in Zeichenketten gesucht, Teile aus Zeichenketten herausgeschnitten und noch vieles mehr gemacht werden.
- **Arithmetische Funktionen:** Die Klasse der arithmetischen Funktionen führt Rechenoperationen auf den Operanden durch (z. B. Runden, Radizieren, Logarithmieren, Potenzieren, Modulo, usw.).
- **Datumsfunktionen:** Datumsfunktionen stehen in Zusammenhang mit Datumswerten. Sie können z. B. den Wochentag zu einem Datum oder einfach nur das aktuelle Systemdatum anzeigen.
- **Sonstige Funktionen:** Alles was sich nicht in die oben stehenden drei Kategorien einteilen lässt, zählt zu den sonstigen Funktionen.



Mit Ausnahme der **FROM**-Klausel, können Single Row Functions in allen anderen Klauseln genutzt werden!

7.2 Zeichenkettenfunktionen

Die Kategorie der Zeichenkettenfunktionen stellt nützliche Werkzeuge zur Auswertung und Modifikation von Zeichenketten¹ zur Verfügung. Mit ihrer Hilfe kann man:

- die Schreibweise eines Strings (Groß- / Kleinschreibung) verändern,
- die Länge einer Zeichenkette ermitteln,
- Leerzeichen abschneiden,
- Teilzeichenketten (Substrings) ausschneiden

und noch vieles mehr. An dieser Stelle sollen einige Beispiele für Zeichenkettenfunktionen in Oracle und SQL Server gezeigt werden.



Das wesentliche Ziel dieses Abschnittes ist es, dem Teilnehmer die Anwendung von Funktionen im Allgemeinen näher zu bringen. Spezielle Kenntnisse über einzelne Funktionen stehen dabei im Hintergrund.

7.2.1 Groß- oder Kleinschreibung - UPPER, LOWER, INITCAP

In Abschnitt ?? wurde bereits auf die Problematik der Casesensitivität hingewiesen. Oracle ist standardmäßig casesensitiv, SQL Server nicht. Soll in Oracle nach einer bestimmten Zeichenkette, z. B. einem Nachnamen gesucht werden und die korrekte Schreibweise ist nicht bekannt, kann es vorkommen, dass das gewünschte Ergebnis nicht erreicht wird. Hierfür gibt es eine Lösung: Die Funktionen **UPPER**, **LOWER** und **INITCAP**.

Tabelle 7.1: Zeichenkettenfunktionen

Funktionsbezeichnung		Bedeutung
UPPER	UPPER	Wandelt die gesamte Zeichenkette in Großbuchstaben um.
LOWER	LOWER	Wandelt die gesamte Zeichenkette in Kleinbuchstaben um.
INITCAP	n. a.	Wandelt das erste Zeichen jedes Wortes in einen Großbuchstaben um.

¹Zeichenkette = engl. String



Die Funktion `INITCAP` existiert in MS SQL Server nicht!

Beispiel ?? zeigt die Anwendung der drei Funktionen **UPPER**, **LOWER** und **INITCAP** in Oracle.

Listing 7.1: UPPER, LOWER und INITCAP

```
SELECT UPPER(Vorname) AS GROSS, LOWER(Nachname) AS Klein,
       INITCAP(Vorname || ' ' || Nachname) AS NORMAL
  FROM Mitarbeiter;
```



GROSS	KLEIN	NORMAL
MAX	winter	Max Winter
SARAH	werner	Sarah Werner
FINN	seifert	Finn Seifert
SEBASTIAN	schwarz	Sebastian Schwarz

100 Zeilen ausgewählt



Die Anwendung der Funktionen **UPPER** und **LOWER** ist in Oracle und MS SQL Server identisch!

Eingangs wurde erwähnt, das Single Row Functions nicht nur in der **SELECT**-Klausel genutzt werden können, sondern, z. B. auch in der **WHERE**-Klausel. Dadurch kann das beschriebene Problem der Casesensitivität gelöst werden.

Listing 7.2: Das Problem der Casesensitivität

```
SELECT Mitarbeiter_ID, Vorname, Nachname
  FROM Mitarbeiter
 WHERE Nachname LIKE 'winter';
```



MITARBEITER_ID	VORNAME	NACHNAME

Keine Zeilen ausgewählt

Der Mitarbeiter „Winter“ wird von Oracle nicht gefunden, da er in der Datenbank mit einem großen W am Namensanfang gespeichert ist. Für Oracle sind „winter“ und „Winter“ zwei unterschiedliche Zeichenketten. Hier kann die **LOWER**-Funktion Abhilfe schaffen.

Listing 7.3: LOWER - Die Lösung des Problems

```
SELECT Mitarbeiter_ID, Vorname, Nachname
  FROM Mitarbeiter
 WHERE LOWER(Nachname) LIKE 'winter';
```



MITARBEITER_ID	VORNAME	NACHNAME
1	Max	Winter
9	Louis	Winter

2 Zeilen ausgewählt

Die **LOWER**-Funktion stellt in Beispiel ?? sicher, dass alle Werte der Spalte NACHNAME in Kleinbuchstaben ausgegeben werden. Somit ist der Vergleich mit „winter“ unproblematisch.

7.2.2 Zeichenketten bearbeiten

Eine weitere Anwendung von Zeichenkettenfunktionen besteht darin, Teile aus Zeichenketten herauszutrennen oder deren Länge festzustellen. Hierzu kennen Oracle und SQL Server unterschiedliche Funktionen, die in Tabelle ?? beschrieben werden. Sie stellt jedoch nur einen Ausschnitt aus der Menge der Zeichenkettenfunktionen dar.

Tabelle 7.2: Zeichenkettenfunktionen



Funktionsbezeichnung	Bedeutung	
SUBSTR	SUBSTRING	Schneidet einen Teil einer Zeichenkette aus und liefert ihn zurück.
LENGTH	LEN	Gibt die Länge einer Zeichenkette zurück.
INSTR	CHARINDEX	Diese Funktionen suchen nach Zeichenkette A in einer Zeichenkette B und liefern die Position von A zurück. Ist A nicht in B erhält man 0 als Ergebnis.

SUBSTR, LENGTH und INSTR in Oracle

Die Funktion **LENGTH** ermittelt, aus wie vielen Zeichen ein String besteht. Sie wird meist in Zusammenhang mit den beiden anderen Funktionen **SUBSTR** und **INSTR** genutzt. Beispiel ?? zeigt die Anwendung von **LENGTH** auf sehr einfache Art und Weise.

Listing 7.4: Die **LENGTH**-Funktion

```
SELECT LENGTH(IBAN), IBAN
FROM Konto
WHERE Konto_ID = 1281;
```



LENGTH (IBAN)	IBAN
22	DE23465387306148533897

1 Zeile ausgewählt

Alleine für sich, ist die Information „22“ auf den ersten Blick nutzlos, wenn man aber bedenkt, dass z. B. eine IBAN eine feste Länge von 22 Zeichen hat, kann man mit Hilfe von LENGTH verifizieren, ob es sich um eine IBAN mit gültiger Länge handelt.

SUBSTR ist dabei behilflich, einen Teil aus einer Zeichenkette auszuschneiden. Eine solche Vorgehensweise ist z. B. dann notwendig, wenn eine Information, in der Form „Winter, Max“, in einer Tabellenspalte abgelegt ist oder, wenn wie im Falle der IBAN, mehrere Informationen einfach verkettet wurden (Länderkennung, Prüfziffer, BLZ und KtoNr).



Das Ergebnis einer solchen Operation wird als „Teilzeichenkette“ oder „Substring“ bezeichnet, wobei „Substring“ die geläufigere Variante darstellt.

Beispiel ?? zeigt die Anwendung der Funktion SUBSTR, um die IBAN eines Kontos in ihre Bestandteile zu zerlegen. Hier kann für den dritten Wert der Funktion auch eine andere Funktion mit angegeben werden.

Listing 7.5: Die Anwendung der Funktion SUBSTR

```
SELECT SUBSTR(IBAN, 1, 2) AS Laenderkuerzel, SUBSTR(IBAN, 5, 8) AS BLZ,
       SUBSTR(IBAN, 13, LEN(IBAN)) AS KtoNr
  FROM Konto
 WHERE Konto_ID = 1281;
```



LAENDE	BLZ	KTONR
DE	46538730	6148533897

1 Zeile ausgewählt

Tabelle 7.3: Funktionsargumente von SUBSTR

Argument	Beispiel	Erläuterung
1	IBAN	Beliebige Zeichenkette (Literal, Spaltenbezeichner oder eine andere Funktion).
2	11	An welcher Stelle im ersten Argument soll das Ausschneiden begonnen werden?. Hier kann eine beliebige Integerzahl stehen, die kleiner ist, als die Gesamtlänge von Argument 1.

3	16	Mit dem dritten und letzten Argument wird angegeben, wie viele Zeichen ausgeschnitten werden sollen. <ul style="list-style-type: none">• $n > 1$: Es werden n Zeichen ausgeschnitten.• n nicht angegeben: Es werden alle Zeichen, bis zum Ende der Zeichenkette angezeigt.• $n < 1$: NULL-Wert als Ergebnis
---	----	---

Mit **INSTR** kann die Position eines Zeichens oder einer Zeichenkette in einer Zeichenkette bestimmt werden. Eine typische Aufgabenstellung könnte sein: „Bestimme ob das Länderkürzel DE in der IBAN vorkommt“. In SQL ausgedrückt sieht dies so aus:

Listing 7.6: Automatische Positionsbestimmung

```
SELECT INSTR(IBAN, 'DE') AS Position
FROM Konto
WHERE Konto_ID = 1281;
```



QUESTION

POSITION

1

1 Zeile ausgewählt

Beispiel ?? zeigt, wie mit Hilfe der **INSTR**-Funktion die Position eines einzelnen Zeichens bzw. mehrere Zeichen in einer Zeichenkette ermittelt werden kann.

Tabelle 7.4: Funktionsargumente von INSTR

Argument	Beispiel	Erläuterung
1	IBAN	Beliebige Zeichenkette (Literal, Spaltenbezeichner oder eine andere Funktion).
2	'DE'	Das zweite Argument gibt an, nach welchem Zeichen / welcher Zeichenkette gesucht werden soll.



Die Funktion **INSTR** kennt noch zwei weitere Parameter, welche hier nicht näher erläutert werden. Weitere Informationen zu dieser Funktion können aus der Online-Dokumentation entnommen werden.

Die folgenden Links verweisen auf die Online-Dokumentation der Oracle-Datenbank.



- [i77725]
- [i87066]
- [i77598]

SUBSTRING, LEN und CHARINDEX in MS SQL Server

Die Funktion **LEN** ermittelt, aus wie vielen Zeichen ein String besteht. Sie wird meist in Zusammenhang mit den beiden anderen Funktionen **SUBSTRING** und **CHARINDEX** genutzt. Beispiel ?? zeigt die Anwendung von **LEN** auf sehr einfache Art und Weise.

Listing 7.7: Die **LEN**-Funktion

```
SELECT LEN(IBAN), IBAN
FROM Konto
WHERE Konto_ID = 1281;
```



(Kein Spaltenname)	Nachname
22	DE23465387306148533897

1 Zeile ausgewählt

Alleine für sich, ist die Information „22“ auf den ersten Blick nutzlos, wenn man aber bedenkt, dass z. B. eine IBAN eine feste Länge von 22 Zeichen hat, kann man mit Hilfe von **LEN** verifizieren, ob es sich um eine IBAN mit gültiger Länge handelt.

SUBSTRING ist dabei behilflich, einen Teil aus einer Zeichenkette auszuschneiden. Eine solche Vorgehensweise ist z. B. dann notwendig, wenn eine Information, in der Form „Winter, Max“, in einer Tabellenspalte abgelegt ist oder, wenn wie im Falle der IBAN, mehrere Informationen einfach verkettet wurden (Länderkennung, Prüfziffer, BLZ und KtoNr).



Das Ergebnis einer solchen Operation wird als „Teilzeichenkette“ oder „Substring“ bezeichnet, wobei „Substring“ die geläufigere Variante darstellt.

Beispiel ?? zeigt die Anwendung der Funktion **SUBSTRING**, um die IBAN eines Kontos in ihre Bestandteile zu zerlegen.

Listing 7.8: Die Anwendung der Funktion **SUBSTRING**

```
SELECT SUBSTRING(IBAN, 1, 2) AS Laenderkuerzel, SUBSTRING(IBAN, 5, 8) AS BLZ,
       SUBSTRING(IBAN, 14) AS KtoNr
FROM Konto
WHERE Konto_ID = 1281;
```



LAENDE	BLZ	KTONR
DE	46538730	6148533897

1 Zeile ausgewählt

Tabelle 7.5: Funktionsargumente von SUBSTR

Argument	Beispiel	Erläuterung
1	IBAN	Beliebige Zeichenkette (Literal, Spaltenbezeichner oder eine andere Funktion).
2	11	An welcher Stelle im ersten Argument soll das Ausschneiden begonnen werden?. Hier kann eine beliebige Integerzahl stehen, die kleiner ist, als die Gesamtlänge von Argument 1.
3	16	Mit dem dritten und letzten Argument wird angegeben, wie viele Zeichen ausgeschnitten werden sollen. <ul style="list-style-type: none"> • $n > 1$: Es werden n Zeichen ausgeschnitten. • n nicht angegeben: Es werden alle Zeichen, bis zum Ende der Zeichenkette angezeigt. • $n < 1$: NULL-Wert als Ergebnis

Mit **CHARINDEX** kann die Position eines Zeichens oder einer Zeichenkette in einer Zeichenkette bestimmt werden. Eine typische Aufgabenstellung könnte sein: „Bestimme ob das Länderkürzel DE in der IBAN vorkommt“. In SQL ausgedrückt sieht dies so aus:

Listing 7.9: Automatische Positionsbestimmung

```
SELECT CHARINDEX('DE', IBAN) AS Position
FROM Konto
WHERE Konto_ID = 1281;
```



POSITION
1

1 Zeile ausgewählt

Beispiel ?? zeigt, wie mit Hilfe der **CHARINDEX**-Funktion die Position eines einzelnen Zeichens in einer Zeichenkette ermittelt werden kann.

Tabelle 7.6: Funktionsargumente von CHARINDEX

Argument	Beispiel	Erläuterung
1	'DE'	Das erste Argument gibt an, nach welchem Zeichen / welcher Zeichenkette gesucht werden soll.
2	Nachname + ', ' + Vorname	Das zweite Argument ist eine beliebige Zeichenkette. An dieser Stelle kann ein Literal, ein Spaltenbezeichner oder eine andere Funktion stehen.



Die Funktion `CHARINDEX` kennt noch einen weiteren Parameter, welcher hier nicht näher erläutert wird. Weitere Informationen zu dieser Funktion können aus der Online-Dokumentation, der MSDN, entnommen werden.

Die folgenden Links verweisen auf die Online-Dokumentation des MS SQL Server, im Microsoft Developer Network (MSDN).



- [\[ms190329\]](#)
- [\[ms187748\]](#)
- [\[ms186323\]](#)

7.3 Arithmetische Funktionen

Arithmetische Funktionen dienen dazu Berechnungen anzustellen. Dies kann beispielsweise sein:

- Runden,
- Radizieren (Wurzelziehen),
- Potenzieren,
- Logarithmieren

und vieles mehr. In dieser Unterrichtsunterlage werden nur einige Beispiele gezeigt.

7.3.1 FROM oder nicht FROM, das ist hier die Frage

In Beispiel ?? und Beispiel ?? wird die Berechnung der Quadratwurzel, der Zahl 4, in Oracle und MS SQL Server gezeigt. Die Anwendung der Funktion **SQRT** ist in beiden Systemen gleich. Trotzdem existiert ein gravierender Unterschied zwischen beiden Beispielen.

Listing 7.10: Berechnung der Quadartwurzel, der Zahl 4, in Oracle

```
SELECT SQRT(4) AS "Wurzel 4"  
FROM   dual;
```

Listing 7.11: Berechnung der Quadratwurzel, der Zahl 4, in MS SQL Server

```
SELECT SQRT(4) As "Wurzel 4";
```



Wurzel 4

2

1 Zeile ausgewählt

Eingangs wurde behauptet, dass jedes **SELECT**-Statement immer eine **FROM**-Klausel benötigen würde. Dies ist gemäß SQL-Standard auch richtig. Das DBMS Oracle setzt an dieser Stelle den Standard konsequent um, der MS SQL Server nicht.



Im DBMS Oracle muss jedes **SELECT**-Statement immer eine **FROM**-Klausel aufweisen, in MS SQL Server nicht.

Um den SQL-Standard einhalten zu können, gibt es in Oracle eine „Dummy-Tabelle“ namens DUAL. Diese enthält nur eine Spalte, mit einer Zeile, wie in Beispiel ?? zu sehen ist. Sie kommt immer dann zum Einsatz, wenn das Ergebnis einer Funktion, unabhängig von irgendwelchen Datensätzen, abgerufen werden muss.

Listing 7.12: Die Tabelle DUAL in Oracle

```
SELECT *
FROM   dual;
```



DUMMY

X

1 Zeile ausgewählt

Der MS SQL Server kommt ohne diese Tabelle aus, da bei ihm die **FROM**-Klausel einfach weggelassen werden darf.

7.3.2 Arithmetische Funktionen anwenden

Oracle kennt ca. 30 und MS SQL Server ca. 20 arithmetische Funktionen. Ziel dieser Unterrichtsunterlage ist es, einige wenige davon herauszugreifen und deren Anwendung zu erläutern. Hierbei handelt es sich um die folgenden Funktionen:

Tabelle 7.7: Arithmetische Funktionen



Funktionsbezeichnung		Bedeutung
CEIL	CEILING	Gibt immer die kleinste ganze Zahl aus, die größer oder gleich n ist (Ganzzahliges Aufrunden).
FLOOR	FLOOR	Gibt immer die größte ganze Zahl aus, die kleiner oder gleich n ist (Ganzzahliges Abrunden).
LOG	LOG 10	Berechnet den Logarithmus der Zahl x zur Basis n. MS-SQL nur zur Basis 10
MOD	%	Gibt den Rest einer ganzzahligen Division zurück.
POWER	POWER	Potenziert die Zahl x mit n.
ROUND	ROUND	Auf- bzw. Abrunden einer Zahl nach dem kaufmännischen Rundungsverfahren ($x < 0,5 = \text{Abrunden}$, $x \geq 0,5 = \text{Aufrunden}$).
TRUNC	n. a.	Schneidet die Nachkommastellen einer Zahl ab und gibt den ganzzahligen Anteil zurück.

In den beiden folgenden Beispielen wird die Anwendung von Rundungsfunktionen in Oracle und MS SQL Server gezeigt.

Listing 7.13: Rundungsfunktionen in Oracle

```
SELECT SQRT(3) AS "Wurzel 3", CEIL(SQRT(3)) AS "Aufrunden",
       FLOOR(SQRT(3)) AS "Abrunden",
       ROUND(SQRT(3), 2) AS "Kaufm. runden"
FROM   dual;
```

Wurzel 3	Aufrunden	Abrunden	Kaufm. runden
1,7320508	2	1	1,73

1 Zeile ausgewählt

Die Funktionen aus Beispiel ?? sind weitestgehend selbsterklärend. Die mit `SQRT(3)` erzeugte Zahl 1,7320508 wird durch `CEIL` aufgerundet auf 2, von `FLOOR` abgerundet auf 1 und mit Hilfe von `ROUND` kaufmännisch, auf zwei Nachkommastellen, aufgerundet.



Die Funktion `ROUND` rundet kaufmännisch. Das zweite Argument gibt an, auf welche Nachkommastelle gerundet werden soll. Durch die Angabe von 0 wird auf die nächste ganze Zahl gerundet.

Die Anwendung dieser Funktionen ist im MS SQL Server identisch, mit der Ausnahme das **CEIL** dort **CEILING** heißt.

Listing 7.14: Rundungsfunktionen in MS SQL

```
SELECT SQRT(3) AS "Wurzel 3", CEILING(SQRT(3)) AS "Aufrunden",
       FLOOR(SQRT(3)) AS "Abrunden",
       ROUND(SQRT(3), 2) AS "Kaufm. runden";
```



Wurzel 3	Aufrunden	Abrunden	Kaufm. runden
1,7320508	2	1	1,73

1 Zeile ausgewählt

In Beispiel ?? bzw. Beispiel ?? ist zu sehen, dass es möglich ist, Funktionen in einander zu verschachteln. Mit Hilfe des Ausdrucks `SQRT(3)` wird die Quadratwurzel der Zahl 3 errechnet (1,73205080756888). Dieser Wert soll anschließend auf 2 Nachkommastellen gerundet werden. Hierzu kann auf beiden Systemen die Funktion `ROUND` herangezogen werden.

Bei der Abarbeitung des Ausdrucks `ROUND(SQRT(3), 2)` halten sich Oracle und SQL Server an die Gesetze der Arithmetik, d. h. es wird zuerst der innerste Ausdruck (in diesem Falle `SQRT(3)`) aufgelöst und anschließend wird das Ergebnis dieses Ausdrucks durch die Funktion `ROUND` auf zwei Nachkommastellen gerundet.

Die zweite Gruppe der arithmetischen Funktionen, die hier vorgestellt werden sollen, bilden die so genannten „höhreren Rechenarten“ ab. Dies sind: Radizieren, Potenzieren, Logarithmieren. Zusätzlich ist hier noch die Modulo-Operation dargestellt, die den Rest einer ganzzahligen Division ausgibt.

Listing 7.15: Höhere Rechenarten in Oracle

```
SELECT MOD(9, 2) AS Modulo, POWER(10, 2) AS Power,
       LOG(10, 1000) AS Log, SQRT(16) AS Quadratwurzel,
       SQRT(27) / SQRT(3) AS "3. Wurzel von 27",
       LOG(5, 8) AS "Log 8 Basis 5"
FROM dual;
```



Modulo	Power	Log	Quadratwurzel	3. Wurzel von 27	Log 8 Basis 5
1	100	3	4	3	1,292029

1 Zeile ausgewählt

In MS SQL Server können die gleichen Berechnungen durchgeführt werden, jedoch mit dem Unterschied, dass:

- Die Modulo-Operation durch den %-Operator dargestellt wird und nicht durch eine Funktion.
- Es gibt in MS SQL Server keine Entsprechung zur LOG-Funktion.

In MS SQL Server gibt es nur die Funktion **LOG10** (Dekadischer Logarithmus zur Basis 10). Um nun die **LOG**-Funktion von Oracle nachzustellen muss hier $\log_5 8 = \log_{10} 8 / \log_{10} 5$ gerechnet werden.

Listing 7.16: Höhere Rechenarten in MS SQL Server

```
SELECT 9 % 2 AS Modulo, POWER(10, 2) AS Power,
       LOG10(1000) AS Log, SQRT(16) AS Quadratwurzel,
       SQRT(27) / SQRT(3) AS "3. Wurzel von 27",
       LOG10(8) / LOG10(5) AS "Log 8 Basis 5";
```

Modulo	Power	Log	Quadratwurzel	3. Wurzel von 27	Log 8 Basis 5
1	100	3	4	3	1,292029

1 Zeile ausgewählt



- [i97801]
- [i77449]
- [i84140]
- [i77996]
- [i78493]
- [i78633]
- [ms189818]
- [ms178531]
- [ms175121]
- [ms174276]
- [ms175003]



7.4 Datumsfunktionen

7.4.1 Datumswerte

Der Umgang mit Datumswerten in einer Datenbank ist meist nicht einfach. Jedes RDBMS speichert Datumsangaben anders und behandelt diese auch anders. Aus diesem Grund soll an dieser Stelle erst einmal ein Überblick darüber gegeben werden, wie Oracle und MS SQL Server mit Datumswerten umgehen.

Tabelle 7.8: Behandlung von Datumswerten




Eigenschaft		
Standarddatumsformat	DD-MON-YY (z. B. 12-NOV-08)	yyyy-mm-ddThh:mm:ss[.mmm] (z. B. 2004-05-23T14:25:10.487)
Speicherung	internes numerisches Format	internes numerisches Format
Wertebereich	Von 4713 vor Christus bis Dezember 9999.	Zwischen dem 1. Januar 1753 und dem 31. Dezember 9999.
Systemdatum anzeigen	SYSDATE / SYSTIMESTAMP	getdate()

7.4.2 Datumsarithmetik in Oracle

Unter dem Begriff „Datumsarithmetik“ versteht man das Rechnen mit Datumswerten. In Oracle gibt es zwei Möglichkeiten:

- Addition oder Subtraktion von Zahlen zu einem Datumswert
- Verwendung von Interval-Literalen

Führt man Datumsarithmetik durch Addition oder Subtraktion von Zahlen durch gelten folgende Regeln:

- Ganze Zahlen sind Tage, z. B. 1 ist ein Tag, 15 sind fünfzehn Tage
- Fraktale (Nachkommastellen) sind Stunden, Minuten und Sekunden, z. B. $\frac{1}{24} = 0,041666$ ist eine Stunde oder $\frac{1}{60} = 0,000694444$ ist eine Minute
- Es darf nur addiert oder subtrahiert werden, alle anderen Rechenoperationen sind verboten.

Einige Beispiele hierzu: So oder in Kommandozeile SQLPLUS !

Listing 7.17: Einfache Datumsarithmetik in Oracle

```
SELECT to_char (SYSDATE , 'DD.MM.yyyy hh24:mm:ss') AS "Datum/Uhrzeit", SYSDATE + 2 AS "2 Tage",
       to_char (SYSDATE - 3 / 24 , 'DD.MM.yyyy hh24:mm:ss') AS "3 Stunden"
  FROM dual;
```



Datum/Uhrzeit	2 Tage	3 Stunden
30.04.2013 14:36:24	02.05.2013 14:36:24	30.04.2013 11:36:24
1 Zeile ausgewählt		

Intervall-Literale (Oracle)

Intervall-Literale sind dazu da, um Zeiträume anzugeben. Diese können in Form von Jahren, Monaten, Tagen, Stunden, Minuten oder Sekunden ausgedrückt werden. Es gibt zwei grundsätzlich unterschiedliche Arten von Intervallen:

- YEAR TO MONTH
- DAY TO SECOND

Das YEAR TO MONTH Intervall kann aus bis zu zwei Feldern bestehen, wobei das erste die Jahre und das zweite die Monate angibt. Tabelle ?? zeigt hierzu einige Beispiele.

Tabelle 7.9: Das YEAR TO MONTH Intervall

Beispiel	Bedeutung
INTERVAL '10' YEAR	Ein Intervall von 10 Jahren (und 0 Monaten)
INTERVAL '101' YEAR(3)	Ein Intervall von 101 Jahren
INTERVAL '10' YEAR TO YEAR	Das gleiche wie INTERVAL '10' YEAR
INTERVAL '10-3' YEAR TO MONTH	Ein Intervall von 10 Jahren und 3 Monaten
INTERVAL '27' MONTH	Ein Intervall von 27 Monaten

Bei der Angabe von Intervall-Literalen gibt es wichtige Dinge zu beachten:

- Die Zeitangabe wird immer in Hochkommas gesetzt,
- zu jedem Intervall muss angegeben werden, ob es sich um Jahre (YEAR) oder Monate (MONTH) handelt,
- die Standardpräzision eines YEAR TO MONTH Intervalls ist immer 2-stellig. Bei drei- oder mehrstelligen Jahresangaben, muss die Präzision angegeben werden. In Beispiel ?? wird dieses Problem gezeigt.

Listing 7.18: Richtiger Umgang mit YEAR TO MONTH Intervallen

```
SELECT SYSDATE - INTERVAL '101' YEAR
FROM   dual;
```

```
ORA-01873: the leading precision of the interval is too small
01873. 00000 - "the leading precision of the interval is too small"
*Cause:    The leading precision of the interval is too small to store the
           specified interval.
*Action:   Increase the leading precision of the interval or specify an
           interval with a smaller leading precision.
```

```
SELECT SYSDATE - INTERVAL '101' YEAR(3)
FROM   dual;
```



SYSDATE-INTERVAL' 101 (3) ' YEAR

02.05.03

1 Zeile ausgewählt



Mit der Präzision wird angegeben, wie viele Stellen die Jahresangabe haben darf. Der Standardwert ist 2.

Beispiel ?? zeigt, dass die Angaben `INTERVAL '101' YEAR` mit dem Oracle-Fehler ORA-01873 scheitert, da die Standardpräzision nur 2-stellig ist und daher bei einer dreistelligen Jahresangabe die Präzision durch die Angaben `YEAR(3)` auf drei Stellen erhöht werden muss.

Bei der zweiten Art von Intervall-Literal, dem DAY TO SECOND Intervall verhält es sich mit der Syntax genauso, wie beim YEAR TO MONTH Intervall. Tabelle ?? zeigt Beispiele für solche Intervalle.

Tabelle 7.10: Das DAY TO SECOND Intervall

Beispiel	Bedeutung
<code>INTERVAL '8' DAY</code>	Ein Intervall von 8 Tagen
<code>INTERVAL '8 4' DAY TO HOUR</code>	Ein Intervall von 8 Tagen und 4 Stunden
<code>INTERVAL '8 4:25' DAY TO MINUTE</code>	Ein Intervall von 8 Tagen, 4 Stunden und 25 Minuten
<code>INTERVAL '8 4:25:10' DAY TO SECOND</code>	8 Tage, 4 Stunden, 25 Minuten und 10 Sekunden
<code>INTERVAL '120' DAY(3)</code>	120 Tage (Präzision 3!!!)
<code>INTERVAL '3' HOUR</code>	3 Stunden
<code>INTERVAL '3:18' HOUR TO MINUTE</code>	3 Stunden und 18 Minuten
<code>INTERVAL '3:18:10' HOUR TO SECOND</code>	3 Stunden, 18 Minuten und 10 Sekunden
<code>INTERVAL '210' HOUR</code>	210 Stunden
<code>INTERVAL '18' MINUTE</code>	18 Minuten
<code>INTERVAL '18:10' MINUTE TO SECOND</code>	18 Minuten und 10 Sekunden
<code>INTERVAL '120' MINUTE</code>	120 Minuten
<code>INTERVAL '10' SECOND</code>	10 Sekunden
<code>INTERVAL '180' SECOND</code>	180 Sekunden



Für das DAY TO SECOND Intervall gelten, bezüglich der Präzision, die gleichen Regeln, wie beim YEAR TO MONTH Intervall.

Manchmal ist es notwendig Zeitintervalle zu formulieren, die zu keinem der beiden Typen passen. Dies könnte z. B. 4 Jahre, 10 Monate, 8 Tage und 5 Stunden sein. Auch das ist möglich, wie Beispiel ?? zeigt.

Listing 7.19: Ein komplexe Zeitintervall

```
SELECT SYSDATE - INTERVAL '4-10' YEAR TO MONTH -
       INTERVAL '8 5' DAY TO HOUR AS Interval
FROM   dual;
```



INTERVAL

24.06.08
1 Zeile ausgewählt

7.4.3 Datumsarithmetik in MS SQL Server

Unter dem Begriff „Datumsarithmetik“ versteht man das Rechnen mit Datumswerten. In SQL Server stehen hierfür die Funktionen:

- DATEADD
- DATEDIFF
- DATEPART
- DATENAME

zur Verfügung. Sie verarbeiten Datumswerte und Zeitintervalle.



Die verschiedenen Intervalle, die für die genannten Funktionen zur Verfügung stehen sind: NANOSECOND, MICROSECOND, MILLISECOND, SECOND, MINUTE, HOUR, WEEKDAY, WEEK, DAY, DAYOFYEAR, MONTH, QUARTER, YEAR.

Die Funktion DATEADD - Datumswerte addieren

Beispiel ?? zeigt, die Anwendung der Funktion DATEADD

Listing 7.20: Die Funktion DATEADD in SQL Server

```
SELECT GETDATE(), DATEADD(DAY, 1, GETDATE()), DATEADD(HOUR, 1, GETDATE())
      DATEADD(MINUTE, 1, GETDATE());
```



(Kein Spaltenname)	(Kein Spaltenname)
2013-05-02 12:00:36.047	2013-05-03 12:00:36.047
(Kein Spaltenname)	(Kein Spaltenname)
2013-05-02 13:00:36.047	2013-05-02 12:01:36.050

Diese Funktion ermöglicht es, ein Zeitintervall zu einem Datum zu addieren.

Die Funktion DATEDIFF - Eine Differenz bilden

Um die Möglichkeit zu schaffen, die Differenz zwischen zwei Datumswerten zu bilden, wurde die Funktion **DATEDIFF** in MS SQL Server integriert.

Listing 7.21: Die Funktion **DATEDIFF** in SQL Server

```
SELECT GETDATE(),
       DATEDIFF(DAY, CONVERT(DATETIME2, '01.05.2010', 104), GETDATE()),
       DATEDIFF(YEAR, CONVERT(DATETIME2, '01.05.2010', 104), GETDATE());
```



(Kein Spaltenname)	(Kein Spaltenname)	(Kein Spaltenname)
2013-05-13 11:49:34.730	1108	3

Das Ergebnis dieser Funktion ist die Differenz zwischen dem Startdatum und dem Enddatum, in dem angegebenen Intervall.

Die Funktionen DATEPART/DATENAME - Teile eines Datums extrahieren

Mit Hilfe der Funktionen **DATEPART** und **DATENAME** können Teile eines Datums, als Zeichenkette extrahiert werden.

Listing 7.22: **DATEPART** und **DATENAME** in SQL Server

```
SELECT GETDATE(), DATEPART(YEAR, GETDATE()),
       DATEPART(MONTH, GETDATE()) AS "DATEPART",
       DATENAME(MONTH, GETDATE()) AS "DATENAME";
```



(Kein Spaltenname)	(Kein Spaltenname)
2013-05-13 11:49:34.730	2013
(DATEPART)	(DATENAME)
5	Mai

7.5 Sonstige Funktionen

Das NULL-Werte etwas besonderes darstellen wurde in Abschnitt ?? bereits erläutert. Welche Effekte durch diese Besonderheit auftreten, war in Beispiel ?? zu sehen. Dieser Abschnitt zeigt nun, wie man mit der Besonderheit der NULL-Werte umgeht.

Oracle und SQL Server kennen Funktionen, um dieser Problematik Herr zu werden. In Oracle ist es die **NVL**, in SQL Server die **ISNULL** (nicht zu verwechseln mit **IS NULL**) Funktion. Beide Funktionen ersetzen NULL-Werte durch einen nahezu freiwählbaren Ersatzwert. Beispiel ?? zeigt die Funktion **NVL** in Oracle.

Listing 7.23: Die Funktion **NVL**

```
SELECT Gehalt + (Gehalt / 100 * Provision) AS "Mit NULL",
       Gehalt + (Gehalt / 100 * NVL(Provision, 0)) AS "Ohne NULL"
  FROM Mitarbeiter;
```



Mit NULL	Ohne NULL
30000	30000
30000	
...	...
2400	2400
2500	2500
2000	
1500	
1200	1200
101 Zeilen ausgewählt	



In Beispiel ?? geschieht folgendes:

- Bei der Berechnung von $Gehalt + (Gehalt / 100 * Provision)$ kommt die Problematik mit den NULL-Werten zum Tragen und es wird, in einigen Zeilen, der Wert NULL angezeigt.
- Bei der Berechnung von $Gehalt + (Gehalt / 100 * NVL(Provision, 0))$ werden die in der Spalte PROVISION auftretenden NULL-Werte von NVL durch den Wert 0 ersetzt, so dass die Berechnung ein gültiges Ergebnis liefern kann.

Gleiches geschieht beim MS SQL Server durch die Funktion `ISNULL`, wie in Beispiel ?? zu sehen ist.

Listing 7.24: Die Funktion `ISNULL`

```
SELECT Gehalt + (Gehalt / 100 * Provision) AS "Mit NULL",
       Gehalt + (Gehalt / 100 * ISNULL(Provision, 0)) AS "Ohne NULL"
FROM   Mitarbeiter;
```



Mit NULL	Ohne NULL
	30000.000000
	30000.000000
...	...
2400.000000	2400.000000
2500.000000	2500.000000
101 Zeilen ausgewählt	

7.6 Datentypen

Datentypen helfen in der Programmierung konkrete Wertebereiche und darauf definierte Operationen festzulegen. Ist eine Tabellenspalte beispielsweise so erstellt worden, dass sie nur Datumswerte akzeptiert, können dort keine anderen Werte, wie z. B. Zahlen oder Zeichenketten, gespeichert werden. Auch die für Datumswerte definierten Operationen sind exakt begrenzt. Während Addition oder Subtraktion von Zahlen zu einem Datumswert erlaubt sind, ist die Addition zweier Datumswerte nicht möglich. Ebenso verhält es sich mit Zahlen und Zeichenketten. Auf Zahlen können die Rechenoperationen der Arithmetik (+, -, * und /) angewandt werden, auf Zeichenketten nicht.

Tabelle ?? liefert einen kleinen Ausschnitt aus der Menge der Datentypen, die Oracle und SQL Server kennen und erläutert deren Bedeutung.

Tabelle 7.11: Datentypen



Bedeutung

NUMBER	NUMERIC	Numerische Datentypen mit fester Genauigkeit und fester Anzahl von Dezimalstellen.
DATE	DATETIME2	Datums- und Zeitdatentypen zum Darstellen von Datum und Tageszeit.
TIMESTAMP	DATETIME2	Ein Datums- und Zeitdatentyp mit höherer Genauigkeit als DATE.
VARCHAR2	VARCHAR	Nicht-Unicode-Zeichendaten variabler Länge.
CHAR	CHAR	Nicht-Unicode-Zeichendaten fester Länge

7.6.1 Numerische Datentypen

Aufbau

Die beiden Datentypen NUMBER (Oracle) und NUMERIC (SQL Server) sind dazu da, um positive oder negative numerische Werte, mit oder ohne Nachkommastellen, aufzunehmen. Die Wertebereiche, die beide Datentypen aufnehmen können, unterscheiden sich.

Tabelle 7.12: Datentypen




		Wertebereich
NUMBER		$\pm 1.0 * 10^{-130}$ bis $\pm 1.0 * 10^{126} - 1$
	NUMERIC	$-1.0 * 10^{-38}$ bis $1.0 * 10^{38} - 1$



Zahlen die größer oder kleiner als die angegebenen Wertebereiche sind, können nicht aufgenommen werden.

Präzision und Skalarität

Unter der Präzision versteht man die Angabe, bei wie vielen Stellen insgesamt noch ein rundungsfehlerfreies Ergebnis angezeigt werden kann. Die maximale Präzision beider Typen beschränkt sich auf Zahlen, die kleiner oder gleich 10^{38} sind. Daraus folgt, dass solange sich eine Zahl in diesem Wertebereich befindet, sie frei von Rundungsfehlern ist. Ist sie größer, können Rundungsfehler auftreten. Beim Microsoft SQL Server stellt sich diese Problematik nicht, da bei NUMERIC der Wertebereich auf 10^{38} beschränkt ist.

Um die benötigte Präzision einstellen zu können, ist es auf beiden Systemen möglich, die maximale Anzahl Stellen, die eine Tabellenspalte aufnehmen können soll, auf einen Wert zwischen 1 und 38 einzuschränken. Ist eine Spalte, z. B. auf neun Stellen begrenzt, ist die größte Zahl, die in diese Spalte eingefügt werden kann, die 999.999.999.

Mit Hilfe der Skalarität kann manuell festgelegt werden, wie viele der durch die Präzision angegebenen Stellen rechts vom Komma verwendet werden. Wird eine Spalte mit einer Präzision von neun und einer Skalarität von zwei definiert, kann sie maximal sieben Stellen links und zwei Stellen rechts vom Komma enthalten. Die größte Zahl, die in eine solche Spalte eingefügt werden kann, ist somit die 9.999.999,99.

7.6.2 Zeichendatentypen

Typen fester Länge

Bei den Zeichendatentypen wird nach Typen fester Länge und Typen variabler Länge unterschieden. Der Datentyp zur Aufnahme von Zeichenketten fester Länge, heißt in beiden Systemen CHAR. Datentypen

fester Länge haben ihren Namen daher, dass bei der Definition einer Tabellenspalte, mit einem solchen Typ, die Länge der Spalte fest angegeben werden muss.



Der Speicherplatzverbrauch einer solchen Spalte richtet sich nicht nach ihrem Inhalt (Anzahl der enthaltenen Zeichen), sondern nach der vorgegebenen Größe. Wird eine Spalte mit einer Größe von 20 definiert, beträgt ihr Speicherplatzverbrauch 20, 40 oder 80 Byte^a. Dies trifft auch dann zu, wenn die Spalte nur ein einziges Zeichen enthält.

^aJe nach dem, welcher Zeichensatz verwendet wird, werden pro Zeichen 1, 2 oder 4 Byte Speicherplatz benötigt.

Typen variabler Länge

Die Zeichendatentypen variabler Länge unterscheiden sich in Oracle und SQL Server nur in ihrem Namen. SQL Server verwendet die SQL-92-Standard gemäß Bezeichnung VARCHAR, während Oracle die Bezeichnung VARCHAR2, als Synonym für VARCHAR verwendet. Die Nutzung von VARCHAR ist aber auch in Oracle zulässig.



Im Unterschied zu den Typen fester Länge, ergibt sich der Speicherplatzverbrauch bei Typen variabler Länge nicht durch eine fest definierte Größe, sondern anhand ihres Inhalts. Es kann eine maximale Größe angegeben werden. Die Spalte kann dann maximal so viele Zeichen aufnehmen, wie angegeben.

7.6.3 Datums- und Zeittypen

Zur Speicherung von Datums- und Zeitwerten kennt Oracle die beiden Datentypen DATE und TIMESTAMP. Microsoft SQL Server verwendet DATETIME2.



In MS SQL Server gibt es auch einen Datentyp TIMESTAMP. Dieser ist jedoch, abweichend vom SQL-2003-Standard, nicht zur Speicherung von Datums- und Zeitwerten vorgesehen und seit SQL Server 2008 als veraltet eingestuft.

Oracle - DATE und TIMESTAMP

Der Datentyp DATE speichert seine Datumswerte in einem internen, numerischen Format. Er berücksichtigt dabei: Jahrzehnt, Jahr, Monat, Tag, Stunde, Minute, Sekunde. Der Typ TIMESTAMP ist eine Erweiterung. Er kann Datums- und Zeitangaben auf bis zu 9 Stellen (Nanosekunde) genau, im Sekundenbereich speichern.

SQL Server - DATETIME2

Werte des DATETIME2-Datentyps werden von der Microsoft SQL Server 2008 R2 Database Engine (Datenbankmodul) intern, als zwei 4 Byte lange, ganze Zahlen, gespeichert. Die ersten 4 Byte enthalten die Anzahl von Tagen, vor oder nach dem Referenzdatum, dem 1. Januar 1900. Die anderen 4 Byte speichern die Tageszeit, die als Anzahl von Millisekunden seit Mitternacht dargestellt wird.

7.7 Konvertierung von Datentypen

Unter der Konvertierung von Datentypen versteht man das Umwandeln eines Wertes, eines bestimmten Typs, in einen anderen Typ. Beispielsweise kann eine Zeichenkette „1234“ in die gleichlautende Zahl „1234“ umgewandelt werden. Intern wird nur der Datentyp von VARCHAR/VARCHAR2 auf NUMERIC/NUMBER geändert. Dies ist z. B. notwendig, um arithmetische Operationen durchzuführen.

7.7.1 Implizite Datentypkonvertierung



Unter der impliziten Konvertierung versteht man, dass automatische Umwandeln eines Datentyp durch das DBMS in einen Anderen.

In Beispiel ?? wird die implizite Konvertierung der Gehälter in Zeichenketten gezeigt.

Listing 7.25: Implizite Konvertierung von NUMBER zu VARCHAR

```
SELECT 'Das Gehalt von ' || Nachname || ' betraegt: ' || gehalt
      AS "Implizite Konvertierung"
   FROM Mitarbeiter;
```

Implizite Konvertierung

Das Gehalt von Winter betraegt: 88000

Das Gehalt von Werner betraegt: 50000

Das Gehalt von Seifert betraegt: 50000

Das Gehalt von Schwarz betraegt: 30000

101 Zeilen ausgewählt

Damit Oracle eine Ausgabe wie „Das Gehalt von Winter betraegt: 88000“ darstellen kann, muss ein einheitlicher Datentyp geschaffen werden. Hierzu wird ein Ausdruck in zwei Teile zerlegt, einen linken und einen rechten. Der rechte Teil wird dann, sofern möglich, in den Datentyp des Linken konvertiert. Für Beispiel ?? bedeutet dies konkret:

- Linke Seite: 'Das Gehalt von ' || Nachname || 'betraegt: '
- Rechte Seite: Gehalt
- Datentyp der linken Seite: VARCHAR2
- Datentyp der rechten Seite: NUMBER
- Daraus folgt: Konvertiere NUMBER nach VARCHAR2

Nicht immer ist das Konvertieren eines Datentyps in einen anderen möglich. Für die implizite Konvertierung gibt es einige Einschränkungen.

- Der Wert selbst muss in den neuen Typ konvertierbar sein. Beispielsweise kann die Zeichenkette ABCDE nicht in eine Zahl oder ein Datum umgewandelt werden, da sie kein sinnvolles Datum darstellt.
- Der Datentyp selbst muss in den neuen Datentyp konvertierbar sein. Zum Beispiel kann Oracle einen Wert des Datentyps NUMBER nicht direkt in einen Wert des Typs DATE konvertieren, da hierzu ein Referenzdatum benötigt wird.

Die beiden folgenden Tabellen zeigen, welche impliziten Typkonvertierung in Oracle und SQL Server möglich sind. Dabei wird immer zugrunde gelegt, dass der betreffende Wert auch konvertierbar ist.

Tabelle 7.13: Implizite Datentypkonvertierung in Oracle



	NUMBER	VARCHAR2	CHAR	DATE	TIMESTAMP
NUMBER	-	X	X	-	-
VARCHAR2	X	-	X	X	X
CHAR	X	X	-	X	X
DATE	-	X	X	-	-
TIMESTAMP	X	X	X	-	-

Tabelle 7.14: Implizite Datentypkonvertierung in Microsoft SQL Server



	NUMERIC	VARCHAR	CHAR	DATETIME2
NUMERIC	-	X	X	X
VARCHAR	X	-	X	X
CHAR	X	X	-	X
DATETIME	-	X	X	-



Der SQL Server besitzt intern eine Rangfolge seiner Datentypen. Dadurch wird bei der impliziten Konvertierung der Datentyp mit der niedrigeren Rangfolge in den Datentyp mit der höheren Rangfolge umgewandelt.



- [autoId8]
- [ms191530]
- [ms190309]

7.7.2 Explizite Datentypkonvertierung

Explizite Datentypkonvertierung in Oracle



Bei der expliziten Datentypkonvertierung werden Werte, mit Hilfe von Funktionen, von einem Datentyp in einen anderen konvertiert.

Oracle kennt für das explizite Konvertieren von Daten verschiedene Funktionen. Diese sind:

- TO_CHAR
- TO_DATE
- TO_TIMESTAMP

- TO_NUMBER

Es existieren noch weitere Funktionen, die an dieser Stelle jedoch ungenannt bleiben. Die folgende Tabelle zeigt eine Übersicht, welche Datentypen, mit Hilfe der expliziten Datentypkonvertierung umgewandelt werden können.

Tabelle 7.15: Explizite Datentypkonvertierung in Oracle



	NUMBER	VARCHAR2	CHAR	DATE	TIMESTAMP
NUMBER	-	TO_CHAR	TO_CHAR		-
VARCHAR2	TO_NUMBER	-	-	TO_DATE	TO_TIMESTAMP
CHAR	TO_NUMBER	-	-	TO_DATE	TO_TIMESTAMP
DATE	-	TO_CHAR	TO_CHAR	-	TO_TIMESTAMP
TIMESTAMP	-	TO_CHAR	TO_CHAR	-	-

Beispiel ?? zeigt die Umwandlung des Systemdatums in eine Zeichenkette.

Listing 7.26: Explizite Datentypkonvertierung in Oracle

```
SELECT TO_CHAR(SYSDATE)
FROM   dual;
```



TO_CHAR (SYSDATE)
02.05.13
1 Zeile ausgewählt

Tatsächlich geschieht in Beispiel ?? nichts sichtbares, dennoch hat eine Umwandlung stattgefunden. Um diese sichtbar zu machen, kann während der Konvertierung eine Formatierung des Ergebniswertes durchgeführt werden.

Listing 7.27: Konvertierung und Formatierung

```
SELECT TO_CHAR(SYSDATE, 'DD.MM.YYYY')
FROM   dual;
```



TO_CHAR (SYSDATE, ' DD.MM.YYYY')
02.05.2013
1 Zeile ausgewählt

Das zweite Argument der **TO_CHAR**-Funktion, 'DD.MM.YYYY', wird als „Formatmodell“ bezeichnet. Mit Hilfe dieser Buchstabenkombination wird das Ausgabeformat für die Zeichenfolge festgelegt. Die Bedeutung der Buchstaben ist:

- **DD**: Tag 2-stellig
- **MM**: Monat 2-stellig
- **YYYY**: Jahr 4-stellig

Beispiel ?? zeigt ein weiteres, individuelles Formatmodell für einen Datumswert.

Listing 7.28: Ein anderes Formatmodell

```
SELECT TO_CHAR(SYSDATE, 'DD. MON YYYY')
FROM   dual;
```

TO_CHAR(SYSDATE, 'DD.MONYYYY')
02. MAI 2013
1 Zeile ausgewählt



Die Buchstabenkombination **MON** sorgt dafür, dass der Monat als Wort angezeigt wird. Ob „Mai“ oder „May“ angezeigt wird, ist abhängig von den Ländereinstellungen der Datenbank.



Ein Formatmodell ist eine Zeichenfolge, die das Format eines Datums oder eines numerischen Wertes beschreibt. Ein Formatmodell ändert nicht die interne Darstellung eines Wertes in der Datenbank, sondern formatiert lediglich die Ausgabe. Es setzt sich aus mehreren Formatelementen zusammen, z. B. DD oder MM oder YYYY.

Welche Formatmodelle erstellt werden können, hängt davon ab, welche Formatelemente die einzelnen Funktionen kennen. Die folgenden Literaturhinweise führen zu den Tabellen in der Oracle Online-Dokumentation, die alle existierenden Formatelemente enthält.

- [i34570]
- [i34924]





Für die Funktion `TO_CHAR` existiert keine eigenständige Zusammenstellung von Formatelementen, da sie sowohl Datums- als auch Zahlenwerte in Text konvertiert und dafür die Formatelemente von `TO_NUMBER` und `TO_DATE` nutzt.

Explizite Datentypkonvertierung in Microsoft SQL Server



Bei der expliziten Datentypkonvertierung werden Werte, mit Hilfe von Funktionen, von einem Datentyp in einen anderen konvertiert.

MS SQL Server kennt die Funktion `CONVERT` zur Konvertierung von Datentypen². Die Syntax dieser Funktion lautet:

Listing 7.29: Die Syntax der CONVERT-Funktion in MS SQL Server

```
CONVERT( <Datentyp>[(Laenge)] , <Ausdruck>[ , Style])
```

Tabelle 7.16: Funktionsargumente von CONVERT

Argument	Beispiel	Erläuterung
<Datentyp>	DATETIME	Dies ist die Angabe des Datentyps, in den <Ausdruck> konvertiert werden soll. Für jeden Datentyp kann optional eine Länge mit angegeben werden.
<Ausdruck>	'16.01.2009'	<Ausdruck> ist eine Zeichenkette, Zahl, ein Datums- Zeitwert oder eine Funktion.
[Style]	104	Optional kann bei der CONVERT-Funktion ein Formatmodell angegeben werden.

Ein Formatmodell legt fest, wie ein Eingabewert interpretiert oder ein Ausgabewert formatiert werden soll. Alle Formatmodelle sind durch Zahlen kodiert.

Die Bedeutung der Formatmodelle soll konkret anhand von Beispiel ?? erläutert werden.

Listing 7.30: CONVERT mit Formatmodell

```
SELECT CONVERT(DATETIME2 , '02.05.2013' , 104)
```



²Es gibt zusätzlich die Funktion `CAST`. Jedoch wird seitens Microsoft empfohlen, `CONVERT` zu nutzen, obgleich die Verwendung von `CAST` dem ISO-Standard entsprechen würde.

(Kein Spaltenname)

2013-05-02 00:00:00.0000000

Beispiel ?? zeigt die Konvertierung der Zeichenkette „02.05.2013“ in ein Datum. Damit dies korrekt ablaufen kann, muss die Datenbank wissen, wie die einzelnen Teile der Zeichenkette zu verstehen sind. Die Formatmodellnummer 104 besagt, dass der angegebene Ausdruck im Format „DD.MM.YYYY“ zu interpretieren ist.

Was passiert, wenn ein zum Ausdruck inkompatisches Formatmodell angegeben wird, zeigt Beispiel ???. Das Formatmodell „4“ liest den Ausdruck „02.05.2013“ als „DD.MM.YY“ (2-stellige Jahreszahl). Da der Ausdruck aber mit 4-stelliger Jahreszahl angegeben ist, führt dieses Beispiel zu einer Fehlermeldung.

Listing 7.31: CONVERT mit falschem Formatmodell

```
SELECT CONVERT(DATETIME2, '02.05.2013', 4)

Meldung 241, Ebene 16, Status 1, Zeile 1
Fehler beim Konvertieren einer Zeichenfolge in einen datetime-Wert.
```

In Beispiel ?? diente **CONVERT** dazu, um die Zeichenfolge „02.05.2013“ korrekt zu interpretieren (Eingabeformat). Die gleiche Funktion kann aber auch das Ausgabeformat eines Ausdrucks bestimmen. Beispiel ?? zeigt, wie das aktuelle Systemdatum in eine Zeichenkette konvertiert wird. Dabei wird die 2-stellige Jahresangabe in eine 4-stellige umformatiert.

Listing 7.32: Formatieren den Ausgabe mit **CONVERT**

```
SELECT GETDATE(), CONVERT(VARCHAR, GETDATE(), 104)
```

(Kein Spaltenname)

(Kein Spaltenname)

2013-05-02 17:18:24.763 02.05.2013



Welche Formatmodelle SQL Server kennt ist unter dem folgenden Literaturhinweis nachlesbar.

- [ms191530]



7.8 Übungen - Funktionen

- Lassen Sie das aktuelle Datum auf dem Bildschirm ausgeben und benennen Sie die Spalte mit „Datum“.



Datum

12.05.13
1 Zeile ausgewählt

- Lassen Sie das aktuelle Datum mit Uhrzeit auf dem Bildschirm ausgeben und benennen Sie die Spalte mit „Datum/Uhrzeit“.



Datum/Uhrzeit

12.05.13 10:58:45,439419 +02:00
1 Zeile ausgewählt

- Schreiben Sie eine Abfrage, welche die Mitarbeiternummer, den Nachnamen, das Gehalt und ein um 3,5 % erhöhtes Gehalt für jeden Mitarbeiter anzeigt. Das erhöhte Gehalt soll als ganze Zahl und mit dem Spaltenalias „Neues Gehalt“ ausgegeben werden!



MITARBEITER_ID	NACHNAME	GEHALT	Neues Gehalt
1	Winter	88000	91080
2	Werner	50000	51750
3	Seifert	50000	51750
4	Schwarz	30000	31050
100 Zeilen ausgewählt			

- Verändern Sie die Abfrage, aus der vorangegangenen Aufgabe so, dass eine zusätzliche Spalte hinzugefügt wird, die die Differenz zwischen dem alten und dem erhöhten Gehalt anzeigt. Benennen Sie die Spalte mit „Gehaltserhoehung“.



MITARBEITER_ID	NACHNAME	GEHALT	Neues Gehalt	Gehaltserhöhung
	1 Winter	88000	91080	3080
	2 Werner	50000	51750	1750
3 Seifert	50000	51750	1750	
4 Schwarz	30000	31050	1050	

100 Zeilen ausgewählt

5. Zeigen Sie die Nachnamen und die Länge der Nachnamen aller Mitarbeiter an, deren Nachname mit einem der Buchstaben „J“, „M“ oder „S“ beginnt. Die Spalten sollen, wie in der Lösung zu sehen ist, beschriftet sein. Die Nachnamen müssen in Großbuchstaben ausgegeben werden. Sortieren Sie die Abfrage in absteigender Reihenfolge nach den Nachnamen!



Nachname	Laenge
SINDERMANN	10
SINDERMANN	10
SIMON	5
SIMON	5
SIMON	5
SEIFERT	7
SEIFERT	7
SCHWARZ	7
SCHWARZ	7

23 Zeilen ausgewählt

6. Zeigen Sie für jeden Mitarbeiter den Nachnamen an, sein Geburtsdatum und seit wie vielen Monaten dieser bereits 18 Jahre alt ist (gerundet auf zwei Stellen, nach dem Komma). Benennen Sie die Spalte mit den Monaten: „Alter in Monaten“. Sortieren Sie die Abfrage in aufsteigender Reihenfolge nach der Spalte „Alter in Monaten“.



Zur Lösung dieser Aufgabe mit Oracle soll die Funktion **MONTHS_BETWEEN** herangezogen werden, deren Syntax der Oracle Onlinedokumentation entnommen werden kann.



NACHNAME	GEBURTS DATUM	Alter in Monaten
Krüger	31.05.93	23,4
Walther	07.01.93	28,18
Lehmann	07.11.92	30,18
Keller	04.11.92	30,27
Schwarz	27.06.92	34,53
Weber	10.06.92	35,08
Peters	13.05.92	35,98
Köhler	05.05.92	36,24
Lorenz	13.12.91	40,98

100 Zeilen ausgewählt

7. Ermitteln Sie Vorname, Nachname und Geburtsdatum der Mitarbeiter, die mindestens 1 Jahr und 4 Monate nach dem „07.05.1978“ geboren sind. Sortieren Sie die Abfrage in absteigender Reihenfolge nach dem Geburtsdatum.



VORNAME	NACHNAME	GEBURTSDATUM
Emma	Krüger	31.05.93
Lina	Walther	07.01.93
Johannes	Lehmann	07.11.92

81 Zeilen ausgewählt

8. Zeigen Sie für jeden Mitarbeiter, der zum Zeitpunkt der Ausführung dieser Abfrage mindestens 35 Jahre alt ist, dessen Mitarbeiter_ID, das Geburtsdatum und den Wochentag seiner Geburt an. Beschriften Sie die Spalten, wie in der Lösung vorgegeben. Ordnen Sie die Abfrage in aufsteigender Reihenfolge nach dem Wochentag, beginnend beim ersten Tag der Woche!



MITARBEITER_ID	GEBURTSDATUM	Wochentag
42	31.01.77	MONTAG
90	14.12.76	DIENSTAG
36	14.02.78	DIENSTAG
2	03.11.77	DONNERSTAG
51	19.02.76	DONNERSTAG

11 Zeilen ausgewählt

9. Schreiben Sie eine Abfrage, die für alle Mitarbeiter deren Nachnamen und die Bankfiliale_ID anzeigt. Wenn ein Mitarbeiter in keiner Bankfiliale tätig ist, soll „Keine Bankfiliale“ angezeigt werden.



NACHNAME	BANKFILIALE
Möller	Keine Bankfiliale
Winter	Keine Bankfiliale
Meier	Keine Bankfiliale
Sindermann	Keine Bankfiliale
Schwarz	Keine Bankfiliale
Werner	Keine Bankfiliale
Krüger	1
Peters	1
Kipp	1

100 Zeilen ausgewählt

7.9 Lösungen - Funktionen

- Lassen Sie das aktuelle Datum auf dem Bildschirm ausgeben und benennen Sie die Spalte mit „Datum“.



```
SELECT SYSDATE AS "Datum"
FROM   dual;
```



```
SELECT GETDATE() AS "Datum";
```

- Lassen Sie das aktuelle Datum mit Uhrzeit auf dem Bildschirm ausgeben und benennen Sie die Spalte mit „Datum/Uhrzeit“.



```
SELECT SYSTIMESTAMP AS "Datum/Uhrzeit"
FROM   dual;
```



```
SELECT GETDATE() AS "Datum/Uhrzeit";
```

- Schreiben Sie eine Abfrage, welche die Mitarbeiternummer, den Nachnamen, das Gehalt und ein um 3,5 % erhöhtes Gehalt für jeden Mitarbeiter anzeigt. Das erhöhte Gehalt soll als ganze Zahl und mit dem Spaltenalias „Neues Gehalt“ ausgegeben werden!



```
SELECT Mitarbeiter_ID , Nachname , Gehalt ,
       ROUND(Gehalt * 1.035 , 0) AS "Neues Gehalt"
  FROM   Mitarbeiter;
```



```
SELECT Mitarbeiter_ID , Nachname , Gehalt ,
       CEILING(ROUND(Gehalt * 1.035 , 0)) AS "Neues Gehalt"
  FROM   Mitarbeiter;
```

- Verändern Sie die Abfrage, aus der vorangegangenen Aufgabe so, dass eine zusätzliche Spalte hinzugefügt wird, die die Differenz zwischen dem alten und dem erhöhten Gehalt anzeigt. Benennen Sie die Spalte mit „Gehaltserhöhung“.

```
SELECT Mitarbeiter_ID, Nachname, Gehalt,
       ROUND(Gehalt * 1.035, 0) AS "Neues Gehalt",
       ROUND(Gehalt * 1.035, 0) - Gehalt AS "Gehaltserhoehung"
FROM   Mitarbeiter;
```



```
SELECT Mitarbeiter_ID, Nachname, Gehalt,
       CEILING(ROUND(Gehalt * 1.035, 0)) AS "Neues Gehalt",
       CEILING(ROUND(Gehalt * 1.035, 0)) - Gehalt AS "Gehaltserhoehung"
FROM   Mitarbeiter;
```

5. Zeigen Sie die Nachnamen und die Länge der Nachnamen aller Mitarbeiter an, deren Nachname mit einem der Buchstaben „J“, „M“ oder „S“ beginnt. Die Spalten sollen, wie in der Lösung zu sehen ist, beschriftet sein. Die Nachnamen müssen in Großbuchstaben ausgegeben werden. Sortieren Sie die Abfrage in absteigender Reihenfolge nach den Nachnamen!



```
SELECT  UPPER(Nachname) AS "Nachname",
        LENGTH(Nachname) AS "Laenge"
FROM    Mitarbeiter
WHERE   (UPPER(Nachname) LIKE 'J%')
        OR      UPPER(Nachname) LIKE 'M%'
        OR      UPPER(Nachname) LIKE 'S%')
ORDER BY Nachname DESC;
```



```
SELECT  UPPER(Nachname) AS "Nachname",
        LEN(Nachname) AS "Laenge"
FROM    Mitarbeiter
WHERE   (Nachname LIKE 'J%')
        OR      Nachname LIKE 'M%'
        OR      Nachname LIKE 'S%')
ORDER BY Nachname DESC;
```

6. Zeigen Sie für jeden Mitarbeiter den Nachnamen an, sein Geburtsdatum und seit wie vielen Monaten dieser bereits 18 Jahre alt ist (gerundet auf zwei Stellen, nach dem Komma). Benennen Sie die Spalte mit den Monaten: „Alter in Monaten“. Sortieren Sie die Abfrage in aufsteigender Reihenfolge nach der Spalte „Alter in Monaten“.



```
SELECT  Nachname, Geburtsdatum,
        ROUND(MONTHS_BETWEEN(
            SYSDATE, Geburtsdatum +
            INTERVAL '18' YEAR), 2) AS "Alter in Monaten"
FROM    Mitarbeiter
ORDER BY 3;
```



```
SELECT Nachname, Geburtsdatum,
       ROUND(DATEDIFF(MONTH, DATEADD(YEAR, 18, Geburtsdatum),
                      GETDATE()), 2) AS "Alter in Monaten"
FROM Mitarbeiter
ORDER BY 3;
```

7. Ermitteln Sie Vorname, Nachname und Geburtsdatum der Mitarbeiter, die mindestens 1 Jahr und 4 Monate nach dem „07.05.1978“ geboren sind. Sortieren Sie die Abfrage in absteigender Reihenfolge nach dem Geburtsdatum.



```
SELECT Vorname, Nachname, Geburtsdatum
FROM Mitarbeiter
WHERE Geburtsdatum >
      TO_DATE('07.05.1978', 'DD.MM.YYYY') +
      INTERVAL '1-4' YEAR TO MONTH
ORDER BY 3 DESC;
```



```
SELECT Vorname, Nachname, Geburtsdatum
FROM Mitarbeiter
WHERE Geburtsdatum > DATEADD(MONTH, 4, DATEADD(YEAR, 1, '07.05.1978'))
ORDER BY 3 DESC;
```

8. Zeigen Sie für jeden Mitarbeiter, der zum Zeitpunkt der Ausführung dieser Abfrage mindestens 35 Jahre alt ist, dessen Mitarbeiter_ID, das Geburtsdatum und den Wochentag seiner Geburt an. Beschriften Sie die Spalten, wie in der Lösung vorgegeben. Ordnen Sie die Abfrage in aufsteigender Reihenfolge nach dem Wochentag, beginnend beim ersten Tag der Woche!



```
SELECT Mitarbeiter_ID, Geburtsdatum,
       TO_CHAR(Geburtsdatum, 'DAY') AS "Wochentag"
FROM Mitarbeiter
WHERE SYSDATE > Geburtsdatum + INTERVAL '35' YEAR
ORDER BY TO_CHAR(Geburtsdatum, 'D');
```



```
SELECT Mitarbeiter_ID, Geburtsdatum,
       DATENAME(WEEKDAY, Geburtsdatum) AS "Wochentag"
FROM Mitarbeiter
WHERE GETDATE() > DATEADD(YEAR, 35, Geburtsdatum)
ORDER BY DATEPART(WEEKDAY, Geburtsdatum);
```

9. Schreiben Sie eine Abfrage, die für alle Mitarbeiter deren Nachnamen und die Bankfiliale_ID anzeigt. Wenn ein Mitarbeiter in keiner Bankfiliale tätig ist, soll „Keine Bankfiliale“ angezeigt werden.



```
SELECT    Nachname , NVL(          TO_CHAR(Bankfiliale_ID) , 'Keine Bankfiliale') AS Bankfiliale
FROM      Mitarbeiter
ORDER BY  Bankfiliale_ID;
```



```
SELECT Nachname ,
       ISNULL(CONVERT(VARCHAR , Bankfiliale_ID) ,
               'Keine Bankfiliale') AS Bankfiliale
FROM   Mitarbeiter
ORDER BY Bankfiliale_ID;
```

7.10 Konvertierungsfunktion für Römische Zahlen

1. Deiese beiden Funktionen wandeln Arabische Zahlen in Römische und umgekehrt um.



Listing 7.33: Die Fehlermeldung in SQL Server

```
IF OBJECT_ID('dbo.ToRomanNumerals') is NOT NULL
    drop function dbo.ToRomanNumerals

go
CREATE FUNCTION dbo.ToRomanNumerals (@Number INT)
/*
summary:      >
This is a simple routine for converting a decimal integer into a roman numeral.
Author: Phil Factor
Revision: 1.2
date: 3rd Feb 2014
Why: converted to run on SQL Server 2008-12
example:
    - code: Select dbo.ToRomanNumerals(187)
    - code: Select dbo.ToRomanNumerals(2011)
returns:      >
The Mediaeval-style 'roman' numeral as a string.
*/
RETURNS NVARCHAR(100)
AS
BEGIN
    IF @Number<0
        BEGIN
            RETURN 'De romanorum non numero negative'
        end
    IF @Number> 200000
        BEGIN
            RETURN 'O Juppiter, magnus numerus'
        end
    DECLARE @RomanNumeral AS NVARCHAR(100)
    DECLARE @RomanSystem TABLE (symbol NVARCHAR(20)
                                COLLATE SQL_Latin1_General_Cp437_BIN ,
                                DecimalValue INT PRIMARY key)
    INSERT INTO @RomanSystem (symbol, DecimalValue)
    VALUES ('I', 1),
           ('IV', 4),
           ('V', 5),
           ('IX', 9),
           ('X', 10),
           ('XL', 40),
           ('L', 50),
           ('XC', 90),
           ('C', 100).
```

```

( 'CD' , 400) ,
( 'D' , 500) ,
( 'CM' , 900) ,
( 'M' , 1000) ,
( N'|      ' , 5000) ,
( N'cc|      ' , 10000) ,
( N'|      ' , 50000) ,
( N'ccc|      ' , 100000) ,
( N'ccc|      ' , 150000)

WHILE @Number > 0
    SELECT @RomanNumeral = COALESCE(@RomanNumeral , '') + symbol ,
           @Number = @Number - DecimalValue
    FROM   @RomanSystem
    WHERE  DecimalValue = (SELECT MAX(DecimalValue)
                           FROM   @RomanSystem
                           WHERE  DecimalValue <= @number)
RETURN COALESCE(@RomanNumeral , 'nulla')
END
go

/* and we do our unit tests. */
if NOT dbo.ToRomanNumerals(87) = 'LXXXVII'
    RAISERROR ('failed first test',16,1)
if NOT dbo.ToRomanNumerals(99) = 'XCIX'
    RAISERROR ('failed second test',16,1)
if NOT dbo.ToRomanNumerals(0) = 'nulla'
    RAISERROR ('failed third test',16,1)
if NOT dbo.ToRomanNumerals(300000) = 'O Juppiter, magnus numerus'
    RAISERROR ('failed fourth test',16,1)
if NOT dbo.ToRomanNumerals(2725) = 'MMDCXXV'
    RAISERROR ('failed fifth test',16,1)
if NOT dbo.ToRomanNumerals(949) = 'CMXLI'
    RAISERROR ('failed Sixth test',16,1)
if NOT dbo.ToRomanNumerals(154321) = N'ccc| MMMMCCCCXXI
    RAISERROR ('failed Seventh test',16,1)
GO

IF OBJECT_ID('dbo.FromRomanNumerals') is NOT NULL
    drop function dbo.FromRomanNumerals
go
CREATE FUNCTION dbo.FromRomanNumerals (@RomanNumeral NVarchar(100))
/*
summary:    >
This is a simple routine for converting roman numeral into an integer
Author: Phil Factor
Revision: 1.2
date: 2nd Feb 2014
Why: converted to run on SQL Server 2008-12

```

```

example:
- code: Select dbo.FromRomanNumerals('CXVII')
- code: Select dbo.FromRomanNumerals('')

returns:    >
The Integer.
*/
RETURNS int
AS
BEGIN
DECLARE @RomanSystem TABLE (symbol NVARCHAR(20)
                            COLLATE SQL_Latin1_General_Cp437_BIN ,
                            DecimalValue INT PRIMARY key)

DECLARE @Numeral INT
DECLARE @Rowcount int
DECLARE @InString int
SELECT @InString=LEN(@RomanNumeral),@rowcount=100
IF @RomanNumeral='nulla' return 0
INSERT INTO @RomanSystem (symbol , DecimalValue)
VALUES ('I', 1),
       ('IV', 4),
       ('V', 5),
       ('IX', 9),
       ('X', 10),
       ('XL', 40),
       ('L', 50),
       ('XC', 90),
       ('C', 100),
       ('CD', 400),
       ('D', 500),
       ('CM', 900),
       ('M', 1000),
       (N'I', 5000),
       (N'CC', 10000),
       (N'C', 50000),
       (N'CCC', 100000),
       (N'CCCI', 150000)

WHILE @instring>0 AND @RowCount>0
BEGIN
SELECT TOP 1 @Numeral=COALESCE(@Numeral,0)+ DecimalValue ,
          @InString=@Instring -LEN(symbol) FROM
@RomanSystem
Where RIGHT(@RomanNumeral,@InString) LIKE symbol+'%'
      COLLATE SQL_Latin1_General_CP850_Bin
AND @Instring -LEN(symbol)>=0
ORDER BY DecimalValue DESC
SELECT @Rowcount=@@Rowcount
end
RETURN CASE WHEN @RowCount=0 THEN NULL ELSE @Numeral END
END

```

```

go
/* and we do our unit tests. */
if NOT dbo.FromRomanNumerals ('LXXXVII')=87
    RAISERROR ('failed first test',16,1)
if NOT dbo.FromRomanNumerals('XCIX') = 99
    RAISERROR ('failed second test',16,1)
if NOT dbo.FromRomanNumerals('nulla') = 0
    RAISERROR ('failed third test',16,1)
if NOT dbo.FromRomanNumerals('MMDCXXV')= 2725
    RAISERROR ('failed fourth test',16,1)
if NOT dbo.FromRomanNumerals('CMXLIX') = 949
    RAISERROR ('failed fifth test',16,1)

DECLARE @Start DATETIME
SELECT @Start=GETDATE()
DECLARE @ii INT
SELECT @ii=1
WHILE @ii<200000
BEGIN
    IF dbo.FromRomanNumerals (dbo.ToRomanNumerals(@ii)) <> @ii
        BEGIN
            RAISERROR ('failed iteration test at %d test',16,1,@ii)
            SELECT dbo.ToRomanNumerals(@ii)
            SELECT dbo.FromRomanNumerals(dbo.ToRomanNumerals(@ii))
            BREAK
        end
    SELECT @ii=@ii+1
end
SELECT 'That took '
    + CONVERT(VARCHAR(10), DATEDIFF(ms ,@start ,GETDATE())))
+ ' Ms'

```

8 Erweiterte Datenselektion

Inhaltsangabe

Werden zwei Relationen R₁ und R₂ in einer Abfrage miteinander verknüpft, entsteht ein kartesisches Kreuzprodukt. Die Anzahl der Zeilen in diesem Produkt entspricht $R_1 * R_2$. Es bildet die Grundlage für eine Join-Operation, bei der aus einem Kreuzprodukt, mit Hilfe eines Selektionsausdruckes, gezielt die nicht benötigten Zeilen eliminiert werden.

8.1 Der Inner Join

Beim Inner Join werden, im Ergebnis der Abfrage, nur die Zeilen angezeigt, die der Join-Bedingung genügen.

8.1.1 Die ON-Klausel

Die **ON**-Klausel stellt die flexibelste und am Häufigsten genutzte Möglichkeit dar, um zwei Tabellen, in einer Join-Operation, miteinander zu verknüpfen. Dafür werden zwei Spaltenbezeichner und ein Operator benötigt. Beispiel ?? zeigt einen Inner Join zwischen den beiden Tabellen KUNDE und EIGENKUNDE.

Listing 8.1: Ein Join zwischen Kunde und Eigenkunde

```
SELECT Vorname , Nachname , PLZ , Ort
FROM Kunde INNER JOIN Eigenkunde
      ON (Kunde.Kunden_ID = Eigenkunde.Kunden_ID);
```



VORNAME	NACHNAME	PLZ	ORT
Sophie	Junge	39435	Bördeau
Hanna	Beck	39439	Güsten
Noah	Bunzel	39435	Egeln
Sebastian	Peters	39240	Staßfurt
Leni	Braun	06425	Alsleben
Jannis	Schreiber	06406	Bernburg
Noah	Rollert	39435	Wolmirsleben
Amelie	Becker	06425	Plötzkau
Christian	Keller	06449	Giersleben

400 Zeilen ausgewählt

Im Vergleich zu allen Beispielen, die in den vorangegangenen Kapiteln zu sehen waren, ändert sich in Beispiel ?? nur die **FROM**-Klausel. Hier werden zwei Tabellen, KUNDE und EIGENKUNDE, getrennt durch

die beiden Schlüsselworte **INNER JOIN** angegeben. Diese Syntax stammt aus dem SQL-99-Standard und ist selbsterklärend.

In der **ON**-Klausel werden die beiden Spalten angegeben, mit deren Hilfe die Verknüpfung zwischen den Tabellen hergestellt wird. Wichtig für diese beiden Spalten ist, dass sie beide miteinander vergleichbare Werte enthalten. Eine Namensgleichheit beider Spalten ist jedoch nicht notwendig.



Da beide Spalten den Bezeichner **KUNDEN_ID** haben, ist es notwendig die Spaltenbezeichner voll zu qualifizieren. Ein voll qualifizierter Spaltenbezeichner wird immer in der Form **TABELLENBEZEICHNER.SPALtenbezeichner** angegeben.

Es wird empfohlen Spaltenbezeichner immer zu qualifizieren, da dies der Datenbank das Auffinden der Spalten erleichtert und somit die Performance des SQL-Statements steigt. Ohne die Qualifizierung der Spaltenbezeichner in der **ON**-Klausel antworten sowohl Oracle, als auch der MS SQL Server mit einer Fehlermeldung.

Listing 8.2: Eine fehlerhafte ON-Klausel in Oracle

```
SELECT Vorname , Nachname , PLZ , Ort
FROM Kunde INNER JOIN Eigenkunde
      ON (Kunden_ID = Kunden_ID);

Fehler bei Befehlszeile:3 Spalte:24
Fehlerbericht:
SQL-Fehler: ORA-00918: column ambiguously defined
00918. 00000 - "column ambiguously defined"
*Cause:
*Action:
```

Listing 8.3: Eine fehlerhafte ON-Klausel in MS SQL Server

```
SELECT Vorname , Nachname , PLZ , Ort
FROM Kunde INNER JOIN Eigenkunde
      ON (Kunden_ID = Kunden_ID);

Meldung 209, Ebene 16, Status 1, Zeile 3
Mehrdeutiger Spaltenname 'Kunden_ID'.
Meldung 209, Ebene 16, Status 1, Zeile 3
Mehrdeutiger Spaltenname 'Kunden_ID'.
```

8.1.2 Tabellenaliasnamen

Genau wie bei Spaltenbezeichnern existiert auch für Tabellenbezeichner die Möglichkeit, Aliasnamen festzulegen. Der Vorteil solcher Tabellenaliasnamen liegt darin, dass die Länge eines SQL-Statements, durch die Vergabe von sehr kurzen Aliasnamen, stark reduziert werden kann. Beispiel ?? produziert das gleiche Ergebnis, wie Beispiel ??, nutzt jedoch Aliasnamen für die beiden Tabellen.

Listing 8.4: Die Benutzung von Tabellenaliasnamen

```
SELECT Vorname, Nachname, PLZ, Ort
FROM Kunde k INNER JOIN Eigenkunde ek
ON (k.Kunden_ID = ek.Kunden_ID);
```



VORNAME	NACHNAME	PLZ	ORT
Sophie	Junge	39435	Bördeause
Hanna	Beck	39439	Güsten
Noah	Bunzel	39435	Egeln
Sebastian	Peters	39240	Staßfurt

400 Zeilen ausgewählt



Tabellenaliasnamen gelten nur innerhalb eines Statements und beeinflussen die Struktur der Datenbank nicht. Wird ein Tabellenaliasname vergeben, so muss er im gesamten SQL-Statement genutzt werden!

Die bereits bekannten Klauseln `WHERE` und `ORDER BY` können auch in einer Join-Abfrage genutzt werden. In Beispiel ?? wird das Ergebnis auf die Kunden mit Wohnort „Egeln“ reduziert und eine aufsteigende Sortierung nach dem Feld `NACHNAME` eingerichtet.

Listing 8.5: Join mit einschränkender WHERE-Klausel und Sortierung

```
SELECT Vorname, Nachname, PLZ, Ort
FROM Kunde k INNER JOIN Eigenkunde ek
ON (k.Kunden_ID = ek.Kunden_ID)
WHERE Ort LIKE 'Egeln'
ORDER BY Nachname;
```



VORNAME	NACHNAME	PLZ	ORT
Alina	Braun	39435	Egeln
Noah	Bunzel	39435	Egeln
Hanna	Bunzel	39435	Egeln
Paul	Koch	39435	Egeln

18 Zeilen ausgewählt

8.1.3 Die USING-Klausel (Nur Oracle)

Die **USING**-Klausel stellt eine weitere Möglichkeit dar, eine Join-Operation durchzuführen. Sie ist eine Kurzschreibweise für **ON** R1.Spalte = R2.Spalte.

Listing 8.6: Die USING-Klausel

```
SELECT      Vorname, Nachname, PLZ, Ort
FROM        Kunde k INNER JOIN Eigenkunde ek
            USING(Kunden_ID)
WHERE       Ort LIKE 'Egeln'
ORDER BY    Nachname;
```



VORNAME	NACHNAME	PLZ	ORT
Alina	Braun	39435	Egeln
Noah	Bunzel	39435	Egeln
Hanna	Bunzel	39435	Egeln
Paul	Koch	39435	Egeln
Karolin	Lange	39435	Egeln
Marie	Lehmann	39435	Egeln

18 Zeilen ausgewählt

Die Nutzung der **USING**-Klausel unterliegt auch einigen Einschränkungen.

- Die in der **USING**-Klausel genutzte Spalte darf nicht qualifiziert werden.
- Die in der **USING**-Klausel genutzte Spalte muss in den beiden, an der Join-Operation teilnehmenden Tabellen den gleichen Namen tragen.

Listing 8.7: Fehlerhafte Nutzung der USING-Klausel in Oracle

```
SELECT      Vorname, Nachname, PLZ, Ort
FROM        Kunde k INNER JOIN Eigenkunde ek USING(Kunden_ID)
WHERE       k.Kunden_ID = 200
ORDER BY    Nachname;
```

Fehler bei Befehlszeile:4 Spalte:10
Fehlerbericht:
SQL-Fehler: ORA-00904: "D"."KUNDEN_ID": invalid identifier
00904. 00000 - "%s: invalid identifier"
*Cause:
*Action:

8.1.4 Der Natural-Join (Nur Oracle)

Die Natural-Join-Syntax stellt die dritte Variante zur Realisierung von Inner Joins dar.

Listing 8.8: Die Natural-Join-Syntax

```
SELECT Vorname, Nachname, PLZ, Ort
FROM Kunde k NATURAL JOIN Eigenkunde ek
WHERE Ort LIKE 'Egeln'
ORDER BY Nachname;
```



VORNAME	NACHNAME	PLZ	ORT
Alina	Braun	39435	Egeln
Noah	Bunzel	39435	Egeln
Hanna	Bunzel	39435	Egeln

18 Zeilen ausgewählt

Beispiel ?? zeigt, das bei dieser Syntax sowohl die `ON`-Klausel, als auch die `USING`-Klausel überflüssig sind. Dies röhrt daher, dass Oracle automatisch die Spalten in den beiden Tabellen sucht, die den gleichen Namen und den gleichen Datentyp aufweisen. Es werden dabei so vielen Spalten einbezogen wie möglich.



Da Oracle immer alle Spalten mit gleichem Namen und gleichem Datentyp in den Natural-Join einbezieht, sollte diese Syntax mit bedacht genutzt werden!

8.1.5 Die Theta-Style Syntax



Sowohl in Oracle, als auch in MS SQL Server kann die Theta-Style-Syntax nur noch zur Realisierung von Inner Joins genutzt werden. Bis auf wenige Ausnahmen ist daher die ANSI-Style-Syntax, mit dem Schlüsselwort `INNER JOIN`, vorzuziehen!

Die Theta-Style-Syntax stellt die Urvariante der Join-Syntax dar, die auch schon vor dem SQL-99-Standard existierte. Bei dieser Form der Syntax wird in der `FROM`-Klausel nur eine kommasseparierte Liste von Tabellen angegeben, während die Verknüpfungsbedingung in der `WHERE`-Klausel formuliert wird.

Listing 8.9: Ein Inner Join mit Theta-Style-Syntax

```
SELECT Vorname, Nachname, PLZ, Ort
FROM Kunde k, Eigenkunde ek
WHERE k.Kunden_ID = ek.Kunden_ID
AND Ort LIKE 'Egeln'
```

```
ORDER BY Nachname;
```



VORNAME	NACHNAME	PLZ	ORT
Alina	Braun	39435	Egeln
Noah	Bunzel	39435	Egeln
Hanna	Bunzel	39435	Egeln

8.1.6 Mehr als zwei Tabellen verknüpfen

Bei komplexeren Abfragen ist es oft notwendig, auf die Daten von mehr als nur zwei Tabellen zurückzugreifen. Dies kann mit allen bisher gezeigten Syntax-Varianten geschehen.



Da der MS SQL Server sowohl die `USING`-Klausel, als auch die `NATURAL JOIN`-Klausel nicht kennt, kann dieser nur die ANSI-Style-Syntax und den Theta-Style nutzen!

Listing 8.10: Vier Tabellen, verbunden durch Inner Joins

```
SELECT      Vorname ,  Nachname ,  IBAN
FROM        Kunde k  INNER JOIN Eigenkunde ek
          ON (k.Kunden_ID = ek.Kunden_ID)
          INNER JOIN EigenkundeKonto ekk
          ON (ek.Kunden_ID = ekk.Kunden_ID)
          INNER JOIN Konto ko
          ON (ekk.Konto_ID = ko.Konto_ID)
WHERE       Ort LIKE 'Egeln'
ORDER BY    Nachname ,  IBAN;
```



VORNAME	NACHNAME	IBAN
Alina	Braun	DE2327682878309669110
Alina	Braun	DE23582034208834002588
Hanna	Bunzel	DE23343859500956216053
Noah	Bunzel	DE23419162344850780394
Noah	Bunzel	DE23506210719641227144

46 Zeilen ausgewählt

46 Zeilen ausgewählt



Für die Ausführung des SQL-Statements ist die Reihenfolge, in der die Tabellen miteinander verbunden werden, nicht wichtig.

Das gleiche Ergebnis lässt sich auch mit der Theta-Style-Syntax erzielen, wie Beispiel ?? zeigt.

Listing 8.11: Ein komplexer Join in der Theta-Style-Syntax

```
SELECT Vorname, Nachname, IBAN
FROM Kunde k, Eigenkunde ek, EigenkundeKonto ekk, Konto ko
WHERE k.Kunden_ID = ek.Kunden_ID
AND ek.Kunden_ID = ekk.Kunden_ID
AND ekk.Konto_ID = ko.Konto_ID
AND Ort LIKE 'Egeln'
ORDER BY Nachname, IBAN;
```



VORNAME	NACHNAME	IBAN
Alina	Braun	DE2327682878309669110
Alina	Braun	DE23582034208834002588
Hanna	Bunzel	DE23343859500956216053
Noah	Bunzel	DE23419162344850780394
Noah	Bunzel	DE23506210719641227144
Hanna	Bunzel	DE23916870475976982996
Paul	Koch	DE23337659559291799957
Paul	Koch	DE23747825550493162192
Karolin	Lange	DE2338135354878273969
Karolin	Lange	DE23657965268917709598
Marie	Lehmann	DE23311656553298147754
46 Zeilen ausgewählt		

8.2 Outer Joins

Während bei den Inner Joins nur solche Zeilen im Ergebnis angezeigt werden, die der Join-Bedingung genügen, ist dieses Verhalten bei den Outer-Joins anders, da auch Datensätze sichtbar werden, die der Join-Bedingung nicht entsprechen.



Wie bereits erwähnt, können Outer-Joins nicht mehr, mit Hilfe der Theta-Style-Syntax, dargestellt werden!

8.2.1 Left- und Right-Outer-Join

Bei Left- bzw. Right-Outer-Joins wird eine der beiden teilnehmenden Tabellen vollständig angezeigt. Die Schlüsselworte **LEFT** und **RIGHT** geben dabei an, welche der beiden Seiten komplett angezeigt werden soll.

Der Left-Outer-Join

Beim Left-Outer-Join wird die Tabelle, die auf der linken Seite der Join-Klausel steht vollständig angezeigt. Von der Tabelle auf der rechten Seite werden nur solche Datensätze angezeigt, die der Join-Bedingung genügen.

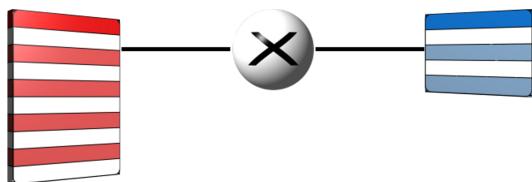


Abb. 8.1:
Left-Outer-Join

Beispiel ?? zeigt einen Left-Outer-Join zwischen den beiden Tabellen **MITARBEITER** und **BANKFILIALE**. Die Auswirkungen des Left-Outer-Joins zeigen sich im Ergebnis nur in den letzten sieben Zeilen. Dort werden Mitarbeiter angezeigt, die in keiner Bankfiliale arbeiten und somit nicht der Join-Bedingung genügen.

Listing 8.12: Ein Left-Outer-Join in Oracle

```
SELECT Vorname, Nachname, b.PLZ, b.Ort
FROM Mitarbeiter m LEFT OUTER JOIN Bankfiliale b
      ON (m.Bankfiliale_ID = b.Bankfiliale_ID);
```

VORNAME	NACHNAME	B.PLZ	B.ORT
Amelie	Krüger	06449	Aschersleben
Marie	Kipp	06449	Aschersleben
...
Emily	Meier		
Peter	Müller		
Tim	Sindermann		
Sebastian	Schwarz		
Finn	Seifert		
Sarah	Werner		
100 Zeilen ausgewählt			
Max Winter			
100 Zeilen ausgewählt			





Da in Beispiel ?? keine Sortierung vorgegeben wurde zeigt Oracle die Zeilen mit den NULL-Werten, in den Spalten PLZ und ORT, automatisch ganz zuletzt an! Dieses Verhalten kann mit dem **NULLS FIRST**-Schlüsselwort, in der **ORDER BY**-Klausel geändert werden.

Listing 8.13: NULL-Werte nach oben sortieren, NULLS FIRST

```
SELECT Vorname, Nachname, b.PLZ, b.Ort
FROM Mitarbeiter m LEFT OUTER JOIN Bankfiliale b
  ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
ORDER BY PLZ NULLS FIRST;
```

VORNAME	NACHNAME	B.PLZ	B.ORT
Emily	Meier		
Peter	Möller		
Tim	Sindermann		
Sebastian	Schwarz		
Max	Winter		
Sarah	Werner		
Finn	Seifert		
Sophie	Schwarz	06406	Bernburg
100 Zeilen ausgewählt			

Der MS SQL Server unterstützt die gleiche Syntax wie Oracle, kennt jedoch das **NULLS FIRST**-Schlüsselwort nicht, da er NULL-Werte bei Angabe einer **ORDER BY**-Klausel automatisch oben anzeigt.

Listing 8.14: Der Left-Outer-Join im MS SQL Server

```
SELECT      Vorname , Nachname , b.PLZ , b.Ort
FROM        Mitarbeiter m LEFT OUTER JOIN Bankfiliale b
           ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
ORDER BY    PLZ;
```

VORNAME	NACHNAME	PLZ	ORT
Emily	Meier	NULL	NULL
Peter	Möller	NULL	NULL
Tim	Sindermann	NULL	NULL
Sebastian	Schwarz	NULL	NULL
Max	Winter	NULL	NULL
Sarah	Werner	NULL	NULL
Finn	Seifert	NULL	NULL
Sophie	Schwarz	06406	Bernburg
100 Zeilen ausgewählt			



Der Right-Outer-Join

Der Right-Outer-Join ist das Komplement zum Left-Outer-Join. Er zeigt alle Datensätze der Tabelle an, die sich auf der rechten Seite befindet. Aus der Tabelle auf der linken Join-Seite werden wiederum nur jene Zeilen angezeigt, die der Join-Bedingung genügen.

Listing 8.15: Ein Right-Outer-Join in Oracle

```
SELECT      Vorname , Nachname , b.PLZ , b.Ort
```

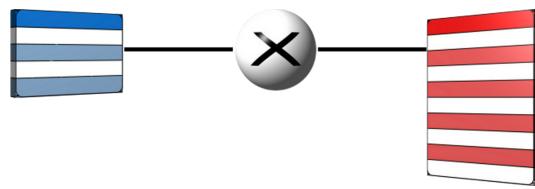


Abb. 8.2:
Der
Right-Outer-Join

```
FROM      Mitarbeiter m RIGHT OUTER JOIN Bankfiliale b
          ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
ORDER BY Nachname NULLS FIRST, PLZ;
```



VORNAME	NACHNAME	B.PLZ	B.ORT
		06425	Alsleben
Finn	Bauer	06425	Plötzkau
Leonie	Bauer	39444	Hecklingen

94 Zeilen ausgewählt

Der erste Datensatz aus Beispiel ?? zeigt, dass es eine Bankfiliale gibt, in der noch keine Mitarbeiter arbeiten. Das gleiche Beispiel lässt sich auch in MS SQL Server abarbeiten.

Listing 8.16: Der gleiche Right-Outer-Join in MS SQL Server

```
SELECT      Vorname, Nachname, b.PLZ, b.Ort
FROM        Mitarbeiter m RIGHT OUTER JOIN Bankfiliale b
          ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
ORDER BY    PLZ, Nachname;
```



VORNAME	NACHNAME	PLZ	ORT
NULL	NULL	06425	Alsleben
Finn	Bauer	06425	Plötzkau
Leonie	Bauer	39444	Hecklingen

94 Zeilen ausgewählt

8.2.2 Der Full Outer Join

Der Full-Outer-Join stellt die logische Ergänzung zu Left-Outer-Join und Right-Outer-Join dar. Er verküpft zwei Tabellen miteinander und zeigt auf beiden Seiten jeweils alle Tabellenzeilen an. Er ist in beiden DBMS, Oracle und MS SQL Server bekannt und syntaktisch gleich.

Listing 8.17: Ein Full-Outer-Join in Oracle

```
SELECT      Vorname, Nachname, b.PLZ, b.Ort
```

```
FROM      Mitarbeiter m FULL OUTER JOIN Bankfiliale b
          ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
ORDER BY PLZ NULLS FIRST, Nachname NULLS FIRST;
```



VORNAME	NACHNAME	B.PLZ	B.ORT
Emily	Meier		
Peter	Möller		
Sebastian	Schwarz		
Finn	Seifert		
...
Anne	Zimmermann	06406	Bernburg
Franz	Berger	06408	Ilberstedt
...
		06425	Alsleben
Finn	Bauer	06425	Plötzkau

101 Zeilen ausgewählt

8.3 Spezielle Joins

8.3.1 Der Self-Join

Ein Self-Join ist eine besondere Form des Inner Join. Er kommt immer dann zum Einsatz wenn der Primary Key einer Tabelle auf einen Foreign Key in der gleichen Tabelle zeigt, also bei rekursiven Beziehungstypen. Ein solcher rekursiver Beziehungstyp existiert in der Tabelle MITARBEITER. Er stellt das Vorgesetztenverhältnis zwischen den Mitarbeitern dar.

Wenn als Ergebnis einer Abfrage zu jedem Mitarbeiter sein Vorgesetzter angezeigt werden soll, so geht dies nur mittels Self-Join. Die folgende Tabelle zeigt das Ergebnis eines solchen Self-Joins.



M#	MVORNAME	MNACHNAME	V#	VVORNAME	VNACHNAME
2	Sarah	Werner	1	Max	Winter
3	Finn	Seifert	1	Max	Winter
4	Sebastian	Schwarz	2	Sarah	Werner
5	Tim	Sindermann	2	Sarah	Werner
6	Peter	Möller	3	Finn	Seifert
7	Emily	Meier	3	Finn	Seifert
8	Dirk	Peters	4	Sebastian	Schwarz
9	Louis	Winter	4	Sebastian	Schwarz



Die Quelltabelle aufspalten

Grundsätzlich ist die Aufgabe einer Join-Operation zwei Tabellen zu einer Ergebnisrelation zu verknüpfen. Im besonderen Falle eines rekursiven Beziehungstyps existiert jedoch nur eine Tabelle. Wie kann der Join stattfinden? Die Antwort auf diese Frage liegt in der Nutzung von Tabellenaliasnamen.

Durch die Vergabe von Tabellenaliasnamen kann mehrfach auf ein und die selbe Tabelle, innerhalb eines SQL-Statements, zugegriffen werden!

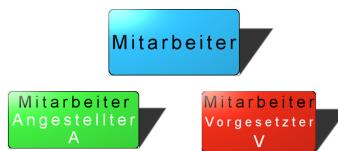


Abb. 8.3:
Gespaltene
Persönlichkeit -
Eine Tabelle,
zwei Aliase

Abbildung ?? zeigt, dass für die Tabelle MITARBEITER zwei Tabellenaliasnamen vergeben werden, nämlich „A“ für Angestellter und „V“ für Vorgesetzter. In SQL ausgedrückt bedeutet dies:

Listing 8.18: Eine Tabelle - zwei Aliasnamen

```

SELECT m.*
FROM   Mitarbeiter m INNER JOIN Mitarbeiter v
...
  
```

Die richtige Join-Bedingung finden

Die eigentliche Leistung, bei der Erstellung eines Self-Join, liegt darin, die korrekte Join-Bedingung zu finden. Fest steht, dass die beiden Spalten MITARBEITER_ID und VORGESETZTER_ID am Join beteiligt sein werden, aber es gibt insgesamt vier verschiedene Möglichkeiten, diese beiden Spalten zu kombinieren:

- **ON** (m.Mitarbeiter_ID = v.Mitarbeiter_ID)
- **ON** (m.Mitarbeiter_ID = v.Vorgesetzter_ID)
- **ON** (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
- **ON** (m.Vorgesetzter_ID = v.Vorgesetzter_ID)

Nun gilt es herauszufinden, welche die richtige Variante ist. Dies geht am Einfachsten, in dem man sich Beispieldaten schafft.



MIT_M#	MIT_NACHNAME	MIT_V#	VOR_M#	VOR_NACHNAME	VOR_V#
3	Seifert	1	1	Winter	
2	Werner	1	1	Winter	
5	Sindermann	2	2	Werner	1
4	Schwarz	2	2	Werner	1
7	Meier	3	3	Seifert	1
6	Möller	3	3	Seifert	1
12	Weber	4	4	Schwarz	2
11	Schwarz	4	4	Schwarz	2

Betrachtet man nun diese vier Join-Bedingungen, im Zusammenhang mit den Beispieldaten, lassen sich zwei davon direkt ausschließen.

- **ON** (*m.Mitarbeiter_ID* = *v.Mitarbeiter_ID*)
- **ON** (*m.Vorgesetzter_ID* = *v.Vorgesetzter_ID*)

Die Bedingung **ON** (*m.Mitarbeiter_ID* = *v.Mitarbeiter_ID*) verknüpft den Mitarbeiter aus der „Tabelle A“ mit dem gleichen Mitarbeiter aus der „Tabelle V“. Das bedeutet, dass alle Mitarbeiter mit sich selbst verknüpft werden, aber nicht mit Ihrem Vorgesetzten.

Die zweite Bedingung **ON** (*m.Vorgesetzter_ID* = *v.Vorgesetzter_ID*) erzeugt „logisches Chaos“. Der Mitarbeiter Seifert liefert hierzu ein gutes Beispiel:



MIT_M#	MIT_NACHNAME	MIT_V#	VOR_M#	VOR_NACHNAME	VOR_V#
3	Seifert	1	1	Werner	1
3	Seifert	1	1	Seifert	1

Es zeigt sich, dass der Mitarbeiter Seifert mit sich selbst und mit seinem Kollegen Werner verknüpft wird. Beide haben eines gemeinsam: Sie haben den gleichen Vorgesetzten. Somit verbleiben nur noch zwei Bedingungen:

- **ON** (*m.Mitarbeiter_ID* = *v.Vorgesetzter_ID*)
- **ON** (*m.Vorgesetzter_ID* = *v.Mitarbeiter_ID*)

Die verbliebenen Bedingungen liefern beide ein sinnvolles Ergebnis. Verwendet man die erste **ON** (*m.Mitarbeiter_ID* = *v.Vorgesetzter_ID*), so ergibt sich folgendes Ergebnis:

MIT_M#	MIT_NACHNAME	MIT_V#	VOR_M#	VOR_NACHNAME	VOR_V#
3	Seifert	1	6	Möller	3
3	Seifert	1	7	Meier	3

Bei beiden Mitarbeitern, Möller und Meier, steht in der Spalte VORGESETZTER_ID der Wert 3. Daraus folgt, beide haben den Mitarbeiter Nummer drei als Vorgesetzten. Mitarbeiter Nummer drei ist Seifert. Mit Hilfe dieser Join-Bedingung werden zu jedem Vorgesetzten die Untergebenen angezeigt. Gesucht ist aber etwas anderes:

Zu jedem Angestellten soll der Vorgesetzte angezeigt werden. Die aktuelle Join-Bedingung zeigt die Informationen also nur aus der falschen Sichtweise an.

Was bleibt, ist nur noch die Bedingung **ON** (`m.Vorgesetzter_ID = v.Mitarbeiter_ID`). Diese zeigt das korrekte, gewünschte Ergebnis an.

MIT_M#	MIT_NACHNAME	MIT_V#	VOR_M#	VOR_NACHNAME	VOR_V#
3	Seifert	1	1	Winter	

Das komplette SQL-Statement zu dieser Problemstellung lautet:

Listing 8.19: Ein Self-Join

```
SELECT m.Mitarbeiter_ID AS MIT_M#, m.Vorname AS MIT_Vorname,
       m.Nachname AS MIT_Nachname, m.Vorgesetzter_ID AS MIT_V#,
       v.Mitarbeiter_ID AS VOR_M#, v.Vorname AS VOR_Vorname,
       v.Nachname AS VOR_Nachname, v.Vorgesetzter_ID AS VOR_V#
FROM Mitarbeiter m INNER JOIN Mitarbeiter v
ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID);
```

8.3.2 Non-Equi-Joins

Kurzgesagt ist ein Non-Equi-Join ein Join, der nicht den Gleichheitsoperator (=) verwendet, sondern einen beliebigen anderen. Meist ist dies dann der **BETWEEN**-Operator. Da diese Art von Join in der Praxis jedoch äußerst selten ist, soll an dieser Stelle nicht weiter darauf eingegangen werden.

8.4 Mengenoperationen

In den vorangegangenen Abschnitten wurde gezeigt, wie zwei Tabellen durch eine Join-Operation miteinander verknüpft werden können. Dies bedingt immer, dass in beiden Tabellen eine Spalte vorhanden ist, die als Join-Attribut genutzt werden kann. Zusätzlich dazu, gibt es noch eine weitere Methode Datensätze unterschiedlicher Tabellen miteinander zu verknüpfen, die *SET-Operatoren*.

SET-Operatoren ermöglichen es, die Operationen der Mengenlehre in einer Datenbank durchzuführen. Tabelle ?? zeigt die Operationen und die dazu gehörenden Operatoren:

Tabelle 8.1: Die SET-Operatoren

Mengenoperation	SET-Operator	Erläuterung
Vereinigung	UNION	Zeigt die Vereinigungsmenge der beiden beteiligten Tabellen an. Duplikatzeilen werden vor der Anzeige eliminiert.
Vollständige Vereinigung	UNION ALL	Zeigt die Vereinigungsmenge der beiden beteiligten Tabellen an. Duplikatzeilen werden vor der Anzeige nicht eliminiert.
Differenz (Oracle)	MINUS	Zeigt nur die Datensätze an, die in der linken der beiden Tabellen vorkommen und keine Entsprechung in der rechten Tabelle haben.
Differenz (MS SQL Server)	EXCEPT	Zeigt nur die Datensätze an, die in der linken der beiden Tabellen vorkommen und keine Entsprechung in der rechten Tabelle haben.
Durchschnitt	INTERSECT	Zeigt nur die Schnittmenge beider Tabellen an.

8.4.1 Voraussetzungen zur Nutzung der SET-Operatoren

Um Mengenoperationen, auf zwei Relationen R und S, anwenden zu können, müssen beide miteinander kompatibel sein. Diese Form der Kompatibilität wird *Typenkompatibilität* oder auch *Vereinigungsverträglichkeit* genannt. Damit zwei Tabellen zueinander Typenkompatibel sind, müssen folgende Bedingungen gegeben sein:

- R und S müssen die gleiche Anzahl Attribute aufweisen.
- Der Wertebereich/Datentyp der Attribute von R und S muss identisch sein.

Das bedeutet zum einen, dass nur solche Abfragen mit Hilfe von SET-Operatoren kombiniert werden können, die die gleiche Anzahl Spalten in der **SELECT**-Klausel haben. Zum anderen müssen die verknüpften Spalten den gleichen Datentyp aufweisen.

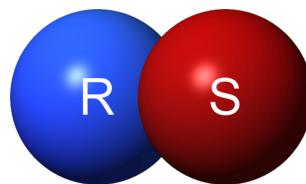
8.4.2 Die SET-Operatoren

UNION und UNION ALL

Der **UNION ALL**-Operator verbindet die Ergebnisse zweier **SELECT**-Statements (Vereinigungsmenge). Sollte es Datensätze geben, die in beiden Abfragen ausgewählt werden (redundante Zeilen), werden diese angezeigt.

Der **UNION**-Operator verbindet, genau wie der **UNION ALL**-Operator, die Ergebnisse zweier SQL-Statements. Der Unterschied zwischen beiden liegt darin, dass der **UNION**-Operator redundante Zeilen ausschließt.

Abb. 8.4:
Vereinigungs-
menge mit
UNION ALL



In einem einfachen Beispiel zum **UNION ALL**-Operator sollen alle Orte angezeigt werden, in denen Kunden oder Mitarbeiter leben. Um diese Aufgabe zu lösen, müssen zwei Abfragen ausgeführt werden.

Listing 8.20: Orte, an denen Kunden leben

```
SELECT Ort
FROM Eigenkunde;
```

Listing 8.21: Orte, an denen Mitarbeiter leben

```
SELECT Ort
FROM Mitarbeiter;
```

Zur Lösung der Aufgabe, müssen die Ergebnisse beider Abfragen kombiniert werden. Dies wird im ersten Anlauf durch den **UNION ALL**-Operator erledigt.

Listing 8.22: Orte, an denen Kunden oder Mitarbeiter leben

```
SELECT Ort
FROM Mitarbeiter
UNION ALL
SELECT Ort
FROM Eigenkunde;
```

ORT

Aschersleben
 Bördehue
 Borne
 Schönebeck
 Alsleben
 Hamburg
 Borne
 Egeln
 Schönebeck

500 Zeilen ausgewählt

An einigen Orten, wie z. B. Aschersleben, Borne, Egeln und Schönebeck, ist zu erkennen, dass der **UNION ALL**-Operator keine redundanten Zeilen ausblendet. Soll das Ergebnis reduziert werden, so dass jeder Ort genau einmal angezeigt wird, kommt der **UNION**-Operator zum Einsatz.

Listing 8.23: Orte, an denen Kunden oder Mitarbeiter leben (reduziert)

```
SELECT Ort
FROM Mitarbeiter
UNION
SELECT Ort
FROM Eigenkunde;
```

**ORT**

Alsleben
 Aschersleben
 Barby
 Berlin
 Bernburg
 Borne
 Bördehue
 Calbe

30 Zeilen ausgewählt

Durch die Anwendung des **UNION**-Operators, statt des **UNION ALL**-Operators verkürzt sich das Ergebnis von 500 Zeilen auf 30.



Der **UNION ALL**-Operator sollte nur dann zum Einsatz kommen, wenn dies zwingend notwendig ist!

In einem weiteren Beispiel soll gezeigt werden, wie Datensätze aus unterschiedlichen Tabellen im Ergebnis gekennzeichnet werden können. In einer Abfrage sollen alle Mitarbeiter und alle Kunden mit den Attributen VORNAME, NACHNAME, PLZ und ORT angezeigt werden. Für die Kunden muss in einer extra Spalte der Buchstabe „K“ und für alle Mitarbeiter der Buchstabe „M“ angezeigt werden.

Listing 8.24: Spalten mit konstanten Werten und UNION

```
SELECT 'M' AS Personentyp, Vorname, Nachname, PLZ, Ort
FROM Mitarbeiter
UNION
SELECT 'K', Vorname, Nachname, PLZ, Ort
FROM Kunde k INNER JOIN Eigenkunde ek
ON (k.Kunden_ID = ek.Kunden_ID);
```



PERSONENTYP	VORNAME	NACHNAME	PLZ	ORT
K	Alexander	Huber	22043	Hamburg
K	Alexander	Lorenz	06408	Ilberstedt
K	Alina	Baumann	07545	Gera
K	Alina	Braun	39435	Egeln
...
M	Alexander	Weber	06449	Aschersleben
M	Amelie	Krüger	03042	Cottbus
M	Anna	Keller	39104	Magdeburg
M	Anna	Schneider	06449	Giersleben

500 Zeilen ausgewählt

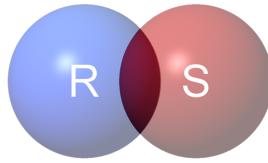
In der ersten Abfrage wird eine Spalte, mit Aliasnamen PERSONENTYP eingefügt. Sie bezieht ihren Wert nicht aus einer Tabelle, sondern sie enthält einfach nur den Buchstaben „K“ für Kunde. Die gleiche Spalte muss nun auch in der zweiten Abfrage eingeführt werden, da beide Abfragen, wie bereits erwähnt, die gleiche Anzahl Spalten, mit den gleichen Datentypen haben müssen. In der zweiten Abfrage kann jedoch der Aliasname entfallen, da dieser nur in der ersten Abfrage registriert/genutzt wird.

INTERSECT

Mit Hilfe des **INTERSECT**-Operators kann der Durchschnitt zweier Ergebnisse angezeigt werden. Das bedeutet, es werden nur die Zeilen angezeigt, die in beiden Relationen, R und S, gleichermaßen vorkommen.

Um die Wirkungsweise dieses Operators zu demonstrieren, wird Beispiel ?? abgewandelt. Der **UNION**-Operator wird durch den **INTERSECT**-Operator ausgetauscht.

Abb. 8.5:
Schnittmenge mit
INTERSECT



Listing 8.25: Orte, an denen sowohl Kunden als auch Mitarbeiter leben

```
SELECT Ort
FROM   Mitarbeiter
INTERSECT
SELECT Ort
FROM   Eigenkunde;
```

**ORT**

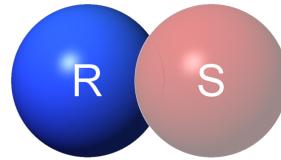
Alsleben
Aschersleben
Bernburg
Borne
Bördehue
25 Zeilen ausgewählt

Das Ergebnis dieser Abfrage liefert nur noch die Orte, an denen sowohl Kunden als auch Mitarbeiter leben.

MINUS / EXCEPT

Dieser Operator zeigt den Inhalt der linken Relation, ohne den Inhalt der Rechten an. Korrekt ausgedrückt bedeutet dies: $t1 MINUS t2 = t1 \setminus t2$. Für SQL Server muss anstatt **MINUS** der Operator **EXCEPT** genutzt werden.

Abb. 8.6:
Der MINUS /
EXCEPT
Operator



Für das kommende Beispiel werden die beiden Tabellen **MITARBEITER** und **EIGENKUNDE** vertauscht.
Der **INTERSECT**-Operator wird gegen den **MINUS**-Operator ausgetauscht.

Listing 8.26: Orte, an denen nur Kunden, aber keine Mitarbeiter leben

```
SELECT Ort
FROM Eigenkunde
MINUS
SELECT Ort
FROM Mitarbeiter;
```

**ORT**

Barby
Berlin
Leipzig
Staßfurt
Wolmirsleben
5 Zeilen ausgewählt

Es gibt 30 verschiedene Orte, an denen Kunden leben und 25 verschiedene Orte, an denen Mitarbeiter leben. In 5 Orten leben nur Kunden, aber keine Mitarbeiter. Der **MINUS**-Operation bzw. der **EXCEPT**-Operator ist dabei behilflich, diese Orte herauszufiltern.

8.5 Übungen - Erweiterte Datenselektion

1. Schreiben Sie eine Abfrage, die für jeden Mitarbeiter den Vornamen, den Nachnamen, die Bankfiliale_ID und den Ort anzeigt, an dem sich seine Filiale befindet.



VORNAME	NACHNAME	BANKFILIALE_ID	ORT
Marie	Kipp	1	Aschersleben
Louis	Schmitz	1	Aschersleben
Johannes	Lehmann	1	Aschersleben
Dirk	Peters	1	Aschersleben
Amelie	Krüger	1	Aschersleben
Martin	Schacke	2	Aschersleben

93 Zeilen ausgewählt

2. Schreiben Sie eine Abfrage, welche die Mitarbeiternummer, den Nachnamen, das Gehalt und ein um 3,5 % erhöhtes Gehalt für alle Mitarbeiter anzeigt, die in einer Filiale in „Aschersleben“ arbeiten. Das erhöhte Gehalt soll als ganze Zahl und mit dem Spaltenalias „Neues Gehalt“ ausgegeben werden.



MITARBEITER_ID	NACHNAME	GEHALT	Neues Gehalt
8	Peters	12000	12420
9	Winter	12000	12420
28	Lehmann	2000	2070
29	Schmitz	2000	2070
30	Kipp	2000	2070
31	Krüger	2500	2588
32	Beck	1500	1553
33	Schacke	1000	1035
34	Oswald	1500	1553
35	Wolf	1000	1035

10 Zeilen ausgewählt

3. Erstellen Sie eine Abfrage, die zu jedem Eigenkunden, der ein Depot besitzt, seinen Vor- und Nachnamen, die Strasse mit der Hausnummer, sowie PLZ und Ort anzeigt.



VORNAME	NACHNAME	STRASSE	PLZ	ORT
Sophie	Junge	Plutoweg 3	39435	Bördeau
Hanna	Beck	Beimsstraße 9	39439	Güsten
Sebastian	Peters	Steinigstraße 3	39240	Staßfurt
Tina	Berger	Bundschuhstraße 1	04177	Leipzig

239 Zeilen ausgewählt

4. Schreiben Sie eine Abfrage, die für alle Eigenkunden deren Vor- und Nachnamen anzeigt, sowie den Vor- und den Nachnamen ihres persönlichen Finanzberaters (Tabelle EIGENKUNDEMitarbeiter). Sortieren Sie die Abfrage nach den Nachnamen der Finanzberater.



Vorname Kunde	Nachname Kunde	Vorname Berater	Nachname Berater
Amelie	Fuchs	Leonie	Bauer
Sarah	Becker	Leonie	Bauer
Pia	Zimmermann	Leonie	Bauer
Hanna	Schreiber	Leonie	Bauer
Frank	Zimmermann	Leonie	Bauer
Chris	Wagner	Leonie	Bauer
Petra	Berger	Leonie	Bauer
Maximilian	Junge	Leonie	Bauer

384 Zeilen ausgewählt

5. Schreiben Sie eine Abfrage, die für alle Eigenkunden, die keinen Berater haben (die nicht in der Tabelle EIGENKUNDEMitarbeiter enthalten sind), den Vor- und den Nachnamen anzeigt.



VORNAME	NACHNAME
Sebastian	Schröder
Udo	Schumacher
Mia	Huber
Simon	Witte
Max	Bunzel
Finn	Fischer
Lara	Meierhöfer
Jannis	Meier

16 Zeilen ausgewählt

6. Schreiben Sie eine Abfrage, die zu jedem Mitarbeiter (Vorname, Nachname) den Vor- und den Nachnamen seines Vorgesetzten anzeigt.



VORNAME_M	NACHNAME_M	VORNAME_V	NACHNAME_V
Finn	Seifert	Max	Winter
Sarah	Werner	Max	Winter
Tim	Sindermann	Sarah	Werner
Sebastian	Schwarz	Sarah	Werner
Emily	Meier	Finn	Seifert
Peter	Möller	Finn	Seifert

99 Zeilen ausgewählt

7. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass alle Mitarbeiter, einschließlich des Mitarbeiters „Winter“, der keinen Vorgesetzten hat, angezeigt werden. Sortieren Sie das Ergebnis aufsteigend nach der Vorgesetzten_ID. Der Mitarbeiter „Winter“ soll ganz oben auf der Liste stehen.



VORNAME	NACHNAME	VORNAME	NACHNAME
Max	Winter		
Finn	Seifert	Max	Winter
Sarah	Werner	Max	Winter
Tim	Sindermann	Sarah	Werner
Sebastian	Schwarz	Sarah	Werner
Emily	Meier	Finn	Seifert

100 Zeilen ausgewählt

8. Erstellen Sie eine Abfrage, die ermittelt, ob es Mitarbeiter gibt, die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten).



VORNAME	NACHNAME
Finn	Bauer
Stefan	Beck
Lina	Becker
Emma	Berger
Udo	Bosse
Georg	Dühning
Tom	Fischer

60 Zeilen ausgewählt

9. Schreiben Sie eine Abfrage, die für alle Mitarbeiter, die höchstens 3 Jahre älter, aber keinesfalls jünger sind als ihr Vorgesetzter, den Vornamen, den Nachnamen, das Geburtsdatum und das Geburtsdatum des Vorgesetzten anzeigt.



VORNAME	NACHNAME	GEBURTS DATUM	Geburtstag Chef
Finn	Seifert	17.10.85	31.08.88
Jessica	Weber	10.06.92	27.06.92
Dirk	Peters	16.09.91	27.06.92
Chris	Lang	08.10.86	30.01.89
Marie	Kipp	27.09.90	16.09.91

20 Zeilen ausgewählt

10. Schreiben Sie eine Abfrage, die für alle Mitarbeiter, die am gleichen Ort arbeiten, an dem sie auch wohnen, deren Vorname, Nachname den Wohnort und den Arbeitsort anzeigt. Beschriften Sie die Spalten, wie es in der Lösung zu sehen ist. Sortieren Sie die Abfragen in absteigender Reihenfolge nach dem Wohnort.



VORNAME	NACHNAME	Wohnort	Arbeitsort
Emily	Günther	Plötzkau	Plötzkau
Jannis	Friedrich	Güsten	Güsten
Tim	Zimmermann	Egeln	Egeln

3 Zeilen ausgewählt

11. Erstellen Sie eine Abfrage, die ermittelt, ob es Mitarbeiter gibt (Vorname und Nachname), die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten) und Filialleiter.



VORNAME	NACHNAME
Amelie	Krüger
Anna	Schneider
Chris	Simon
Christian	Haas
Elias	Sindermann
Emilia	Köhler
Emma	Krüger

40 Zeilen ausgewählt

12. Erstellen Sie eine Abfrage, die alle Eigenkunden anzeigt, die nur Girokonten aber keine anderen Konten besitzen.



VORNAME	NACHNAME
Amelie	Becker
Amelie	Richter
Chris	Walther
Emilia	Keller
Georg	Keller
Johanna	Schäfer

21 Zeilen ausgewählt

13. Erstellen Sie mit Hilfe einer Abfrage eine Liste, die den Vor- und den Nachnamen aller Kunden enthält, die sowohl ein Sparbuch, als auch ein Depot besitzen. Ob die Kunden ein Girokonto haben oder nicht ist irrelevant.



VORNAME	NACHNAME
Alexander	Lorenz
Alina	Baumann
Alina	Huber
Alina	Peters
Alina	Schumacher
Alina	Schütz
Amelie	Fuchs
Amelie	Günther

176 Zeilen ausgewählt

14. Schreiben Sie eine Abfrage, die eine Liste aller Eigenkunden ausgibt, die ein Girokonto und ein Sparbuch besitzen, aber kein Depot.



VORNAME	NACHNAME
Alina	Braun
Andy	Klingner
Anna	Schubert
Anna	Sindermann
Anna	Wagner
Bea	Witte
Ben	Lehmann
Chris	Beck
Chris	Weber

134 Zeilen ausgewählt

8.6 Lösungen - Erweiterte Datenselektion

1. Schreiben Sie eine Abfrage, die für jeden Mitarbeiter den Vornamen, den Nachnamen, die Bankfiliale_ID und den Ort anzeigt, an dem sich seine Filiale befindet.

```

SELECT m.Vorname, m.Nachname, m.Bankfiliale_ID, b.Ort
FROM Mitarbeiter m INNER JOIN Bankfiliale b
ON (b.Bankfiliale_ID = m.Bankfiliale_ID);
```

2. Schreiben Sie eine Abfrage, welche die Mitarbeiternummer, den Nachnamen, das Gehalt und ein um 3,5 % erhöhtes Gehalt für alle Mitarbeiter anzeigt, die in einer Filiale in „Aschersleben“ arbeiten. Das erhöhte Gehalt soll als ganze Zahl und mit dem Spaltenalias „Neues Gehalt“ ausgegeben werden.

```

SELECT m.Mitarbeiter_ID, m.Nachname, m.Gehalt,
ROUND(m.Gehalt * 1.035, 0) AS "Neues Gehalt"
FROM Mitarbeiter m INNER JOIN Bankfiliale b
ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
WHERE LOWER(b.Ort) LIKE 'aschersleben';
```

3. Erstellen Sie eine Abfrage, die zu jedem Eigenkunden, der ein Depot besitzt, seinen Vor- und Nachnamen, die Strasse mit der Hausnummer, sowie PLZ und Ort anzeigt.

```

SELECT k.Vorname, k.Nachname, ek.Strasse || ' ' || ek.Hausnummer AS Strasse,
ek.PLZ, ek.Ort
FROM Kunde k INNER JOIN Eigenkunde ek
ON (k.Kunden_ID = ek.Kunden_ID)
INNER JOIN EigenkundeKonto ekk
ON (ek.Kunden_ID = ekk.Kunden_ID)
INNER JOIN Depot d
ON (ek.Konto_ID = d.Konto_ID);
```

```

SELECT k.Vorname, k.Nachname, ek.Strasse + ' ' + ek.Hausnummer AS Strasse,
ek.PLZ, ek.Ort
FROM Kunde k INNER JOIN Eigenkunde ek
ON (k.Kunden_ID = ek.Kunden_ID)
INNER JOIN EigenkundeKonto ekk
ON (ek.Kunden_ID = ekk.Kunden_ID)
```

```
INNER JOIN Depot d  
ON (ekk.Konto_ID = d.Konto_ID);
```

4. Schreiben Sie eine Abfrage, die für alle Eigenkunden deren Vor- und Nachnamen anzeigt, sowie den Vor- und den Nachnamen ihres persönlichen Finanzberaters (Tabelle EIGENKUNDEMitarbeiter). Sortieren Sie die Abfrage nach den Nachnamen der Finanzberater.

```

SELECT k.Vorname AS "Vorname Kunde", k.Nachname AS "Nachname Kunde",
       m.Vorname AS "Vorname Berater", m.Nachname AS "Nachname Berater"
  FROM Kunde k INNER JOIN Eigenkunde ek
    ON (k.Kunden_ID = ek.Kunden_ID)
   INNER JOIN EigenkundeMitarbeiter ekm
    ON (ek.Kunden_ID = ekm.Kunden_ID)
   INNER JOIN Mitarbeiter m
    ON (ekm.Mitarbeiter_ID = m.Mitarbeiter_ID)
 ORDER BY m.Nachname;
```

5. Schreiben Sie eine Abfrage, die für alle Eigenkunden, die keinen Berater haben (die nicht in der Tabelle EIGENKUNDEMitarbeiter enthalten sind), den Vor- und den Nachnamen anzeigt.

```

SELECT k.Vorname, k.Nachname
  FROM Kunde k INNER JOIN Eigenkunde ek
    ON (k.Kunden_ID = ek.Kunden_ID)
   LEFT OUTER JOIN EigenkundeMitarbeiter ekm
    ON (ek.Kunden_ID = ekm.Kunden_ID)
 WHERE ekm.Kunden_ID IS NULL;
```

6. Schreiben Sie eine Abfrage, die zu jedem Mitarbeiter (Vorname, Nachname) den Vor- und den Nachnamen seines Vorgesetzten anzeigt.

```

SELECT m.Vorname AS Vorname_M, m.Nachname AS Nachname_M,
       v.Vorname AS Vorname_V, v.Nachname AS Nachname_V
  FROM Mitarbeiter m INNER JOIN Mitarbeiter v
    ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID);
```

7. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass alle Mitarbeiter, einschließlich des Mitarbeiters „Winter“, der keinen Vorgesetzten hat, angezeigt werden. Sortieren Sie das Ergebnis aufsteigend nach der Vorgesetzten_ID. Der Mitarbeiter „Winter“ soll ganz oben auf der Liste stehen.

```

SELECT m.Vorname AS Vorname_M, m.Nachname AS Nachname_M,
       v.Vorname AS Vorname_V, v.Nachname AS Nachname_V
  FROM Mitarbeiter m LEFT OUTER JOIN Mitarbeiter v
```

```
    ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
ORDER BY m.Vorgesetzter_ID NULLS FIRST;
```



```
SELECT      m.Vorname AS Vorname_M, m.Nachname AS Nachname_M,
            v.Vorname AS Vorname_V, v.Nachname AS Nachname_V
FROM        Mitarbeiter m LEFT OUTER JOIN Mitarbeiter v
            ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
ORDER BY    m.Vorgesetzter_ID;
```

8. Erstellen Sie eine Abfrage, die ermittelt, ob es Mitarbeiter gibt, die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten).



```
SELECT m.Vorname, m.Nachname
FROM   Mitarbeiter m LEFT OUTER JOIN EigenkundeMitarbeiter ekm
       ON (m.Mitarbeiter_ID = ekm.Mitarbeiter_ID)
WHERE  ekm.Mitarbeiter_ID IS NULL
       AND m.Bankfiliale_ID IS NOT NULL;
```

9. Schreiben Sie eine Abfrage, die für alle Mitarbeiter, die höchstens 3 Jahre älter, aber keinesfalls jünger sind als ihr Vorgesetzter, den Vornamen, den Nachnamen, das Geburtsdatum und das Geburtsdatum des Vorgesetzten anzeigt.



```
SELECT m.Vorname, m.Nachname, m.Geburtsdatum,
       v.Geburtsdatum AS "Geburtstag Chef"
FROM   Mitarbeiter m INNER JOIN Mitarbeiter v
       ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
WHERE  m.Geburtsdatum BETWEEN v.Geburtsdatum - INTERVAL '3' YEAR AND
       v.Geburtsdatum;
```



```
SELECT m.Vorname, m.Nachname, m.Geburtsdatum,
       v.Geburtsdatum AS "Geburtstag Chef"
FROM   Mitarbeiter m INNER JOIN Mitarbeiter v
       ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
WHERE  m.Geburtsdatum BETWEEN DATEADD(YEAR, -3, v.Geburtsdatum) AND
       v.Geburtsdatum;
```

10. Schreiben Sie eine Abfrage, die für alle Mitarbeiter, die am gleichen Ort arbeiten, an dem sie auch wohnen, deren Vorname, Nachname den Wohnort und den Arbeitsort anzeigt. Beschriften Sie die

Spalten, wie es in der Lösung zu sehen ist. Sortieren Sie die Abfragen in absteigender Reihenfolge nach dem Wohnort.



```
SELECT      m.Vorname, m.Nachname, m.Ort AS "Wohnort", b.Ort AS "Arbeitsort"
FROM        Mitarbeiter m INNER JOIN Bankfiliale b
            ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
WHERE       m.Ort = b.Ort
ORDER BY    m.Ort DESC;
```

11. Erstellen Sie eine Abfrage, die ermittelt, ob es Mitarbeiter gibt (Vorname und Nachname), die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten) und Filialleiter.



```

SELECT m.Vorname , m.Nachname
FROM   Mitarbeiter m LEFT OUTER JOIN EigenkundeMitarbeiter ekm
       ON (m.Mitarbeiter_ID = ekm.Mitarbeiter_ID)
WHERE  ekm.Mitarbeiter_ID IS NULL
MINUS
SELECT DISTINCT v.Vorname , v.Nachname
FROM   Mitarbeiter m INNER JOIN Mitarbeiter v
       ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID);

```



```

SELECT m.Vorname , m.Nachname
FROM   Mitarbeiter m LEFT OUTER JOIN EigenkundeMitarbeiter ekm
       ON (m.Mitarbeiter_ID = ekm.Mitarbeiter_ID)
WHERE  ekm.Mitarbeiter_ID IS NULL
EXCEPT
SELECT DISTINCT v.Vorname , v.Nachname
FROM   Mitarbeiter m INNER JOIN Mitarbeiter v
       ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID);

```

12. Erstellen Sie eine Abfrage, die alle Eigenkunden anzeigt, die nur Girokonten aber keine anderen Konten besitzen.



```

SELECT k.Vorname , k.Nachname
FROM   Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
       INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
       INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
MINUS
SELECT k.Vorname , k.Nachname
FROM   Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
       INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
       INNER JOIN Sparbuch s ON (ekk.Konto_ID = s.Konto_ID)
MINUS
SELECT k.Vorname , k.Nachname
FROM   Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
       INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
       INNER JOIN Depot d ON (ekk.Konto_ID = d.Konto_ID);

```



```

SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
EXCEPT
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Sparbuch s ON (ekk.Konto_ID = s.Konto_ID)
EXCEPT
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Depot d ON (ekk.Konto_ID = d.Konto_ID);

```

13. Erstellen Sie mit Hilfe einer Abfrage eine Liste, die den Vor- und den Nachnamen aller Kunden enthält, die sowohl ein Sparbuch, als auch ein Depot besitzen. Ob die Kunden ein Girokonto haben oder nicht ist irrelevant.



```

SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Sparbuch s ON (ekk.Konto_ID = s.Konto_ID)
INTERSECT
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Depot d ON (ekk.Konto_ID = d.Konto_ID);

```

14. Schreiben Sie eine Abfrage, die eine Liste aller Eigenkunden ausgibt, die ein Girokonto und ein Sparbuch besitzen, aber kein Depot.



```

SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
INTERSECT
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Sparbuch s ON (ekk.Konto_ID = s.Konto_ID)
MINUS
SELECT k.Vorname, k.Nachname

```

```
FROM  Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
      INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
      INNER JOIN Depot d ON (eck.Konto_ID = d.Konto_ID);
```

```
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
INTERSECT
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Sparbuch s ON (ekk.Konto_ID = s.Konto_ID)
EXCEPT
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Depot d ON (ekk.Konto_ID = d.Konto_ID);
```


9 Gruppenfunktionen

Inhaltsangabe

In vielen Fällen ist es notwendig, die aus einer Abfrage resultierenden Datensätze nicht einzeln anzuziegen, sondern sie nach bestimmten Kriterien zusammenzufassen. Dieser Vorgang wird als „gruppieren“ bezeichnet und mittels der **GROUP BY**-Klausel umgesetzt. Sie wird zwischen die beiden Klauseln **WHERE** und **ORDER BY** eingefügt.

9.1 Die GROUP BY-Klausel

In einem ersten Beispiel werden aus der Tabelle **MITARBEITER** die IDs aller Vorgesetzten angezeigt, so dass eine „Liste der Vorgesetzten“ entsteht.

Listing 9.1: Die „Liste der Vorgesetzten“

```
SELECT    Vorgesetzter_ID
FROM      Mitarbeiter
GROUP BY  Vorgesetzter_ID
ORDER BY  1;
```

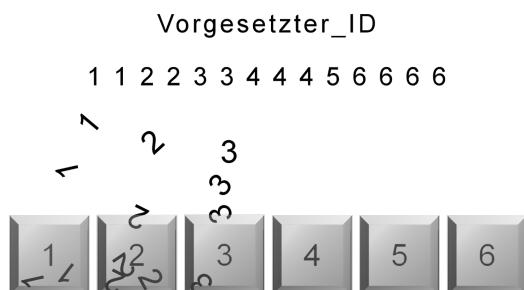


VORGESETZTER_ID

VORGESETZTER_ID
1
2
3
...
27
NULL

28 Zeilen ausgewählt

Abb. 9.1:
Gruppieren von
Datensätzen



Mit Hilfe der **GROUP BY**-Klausel werden die einzelnen IDs in Gruppen eingeteilt. Anschließend wird für jede Gruppe die ID genau einmal angezeigt.

Statt einer einfachen Gruppierung, wie sie in Beispiel ?? erzeugt wurde, kann auch eine mehrfache Gruppierung erzeugt werden. Diese sind aber meist nur in Verbindung mit Aggregatfunktionen, die im folgenden Abschnitt behandelt werden, sinnvoll.

9.2 Die Aggregatfunktionen

Im Gegensatz zu den Single Row Functions, die sich immer nur auf eine Zeile auswirken und deshalb pro Zeile einmal ausgeführt werden müssen, beziehen sich Aggregatfunktionen immer auf eine Gruppierung. Dies können alle Werte einer Spalte oder mehrere getrennte Bereiche sein. Sinn und Zweck dieser Funktionen ist es, den Anwender dabei zu unterstützen, vordefinierte Berechnungen durchzuführen. Tabelle ?? zeigt einen Überblick, über die wichtigsten Aggregatfunktionen.

Tabelle 9.1: Aggregatfunktionen

Aggregatfunktion	Bedeutung	Wertebereich
AVG	Berechnet für den übergebenen Bereich den Durchschnitt aller Werte. NULL-Werte werden bei der Berechnung nicht berücksichtigt.	Numerisch
COUNT	Zählt die zur Gruppierung gehörenden Datensätze. Als Funktionsargument kann ein beliebiger Ausdruck übergeben werden. Wird ein Spaltenbezeichner verwendet, zählt die Funktion die Anzahl der Werte in dieser Spalte. NULL-Werte werden von dieser Funktion nicht berücksichtigt.	Universell
MAX	Liefert den größten Wert eines Bereiches zurück.	Universell
MIN	Liefert den kleinsten Wert eines Bereiches zurück.	Universell
SUM	Berechnet die Summe, für den übergebenen Bereich. NULL-Werte werden bei der Berechnung nicht berücksichtigt.	Numerisch

Beispiel ?? zeigt die Anwendung der Summen-Funktion **SUM**, um die Gehälter aller Mitarbeiter pro Filiale zu ermitteln.

Listing 9.2: Fehler in der Gruppierung

```
SELECT Bankfiliale_ID , SUM(Gehalt)
FROM Mitarbeiter;
```

Auch wenn auf den ersten Blick an dieser Abfrage nichts falsches zu bemerken ist, antwortet das DBMS mit einer Fehlermeldung, was daran liegt, dass die Funktion **SUM** automatisch eine Gruppierung bildet, in die die Spalte **BANKFILIALE_ID** nicht mit einbezogen wird.

Listing 9.3: Die Fehlermeldung in Oracle

```
Fehler beim Start in Zeile 1 in Befehl:
SELECT Bankfiliale_ID , SUM(Gehalt)
```

```
FROM Mitarbeiter
Fehler bei Befehlszeile:1 Spalte:7
Fehlerbericht:
SQL-Fehler: ORA-00937: keine Gruppenfunktion fuer Einzelgruppe
00937. 00000 - "not a single-group group function"
```

Listing 9.4: Die Fehlermeldung in SQL Server

```
Meldung 8120, Ebene 16, Status 1, Zeile 1
Die 'Bank.dbo.Bankfiliale_ID'-Spalte
ist in der Auswahlliste ungültig, da sie nicht in einer
Aggregatfunktion und nicht in der GROUP BY-Klausel enthalten ist.
```



Sobald eine Gruppenfunktion zum Einsatz kommt, müssen alle in der **SELECT**-Klausel gelisteten Attribute gruppiert werden. Dies kann durch die Anwendung weiterer Gruppenfunktionen oder durch die **GROUP BY**-Klausel geschehen.

Um diesen Fehler zu beheben, muss das Statement aus Beispiel ?? um eine **GROUP BY**-Klausel erweitert werden.

Listing 9.5: Der korrekte Einsatz der **SUM**-Funktion

```
SELECT      Bankfiliale_ID,  SUM(Gehalt)
FROM        Mitarbeiter
GROUP  BY    Bankfiliale_ID;
```



BANKFILIALE_ID	SUM(GEHALT)
	308000
1	20500
6	21000
11	23000
13	20000
2	17000
14	19500
20	19500

21 Zeilen ausgewählt

9.2.1 Die Funktion COUNT

Wie in Tabelle ?? beschrieben, wird **COUNT** zum Zählen von Werten genutzt. In Beispiel ?? wird ermittelt, wie viele Mitarbeiter in der Tabelle MITARBEITER gespeichert sind.

Listing 9.6: Das Zählen von Datensätzen

```
SELECT COUNT(*) AS "Mitarbeiter"
FROM   Mitarbeiter;
```



MITARBEITER

100

1 Zeile ausgewählt



Der Stern * kann in der COUNT-Funktion als „Joker“ genutzt werden. COUNT(*) zählt alle Zeilen einer Tabelle und hat somit den gleichen Effekt, wie das Zählen der Einträge der Primärschlüssel Spalte einer Tabelle.

Ein weiteres Beispiel zeigt, dass die COUNT-Funktion NULL-Werte nicht berücksichtigt.

Listing 9.7: NULL-Werte werden nicht gezählt!

```
SELECT COUNT(Vorgesetzter_ID)
FROM Mitarbeiter;
```



COUNT (VORGESETZTER_ID)

99

1 Zeile ausgewählt

Da der Mitarbeiter Winter (MITARBEITER_ID = 1) keinen Vorgesetzten hat, ist bei ihm ein NULL-Wert in der Spalte VORGESETZTER_ID, was dazu führt, dass COUNT nur 99 Werte zählt.

9.2.2 Die Funktion SUM

Mit Hilfe von SUM kann die Summe aller Werte eines Bereichs gebildet werden. Ein Beispiel zu dieser Funktion ist in Beispiel ?? zu sehen. Im Gegensatz zur COUNT-Funktion, spielen NULL-Werte keine Rolle, in Bezug auf die SUM-Funktion. Ein NULL-Wert wird durch die SUM-Funktion einfach ignoriert und verfälscht das Ergebnis dadurch nicht.

9.2.3 Die Funktion AVG

Die Abkürzung „AVG“ steht für das englische Wort „average“ = Durchschnitt (arithmetisches Mittel). Die Funktion AVG berechnet den Durchschnitt der Werte einer Gruppierung.

In Beispiel ?? wird die **AVG**-Funktion genutzt, um das Durchschnittsgehalt für jede Filiale zu berechnen.

Listing 9.8: Die AVG-Funktion

```
SELECT      Bankfiliale_ID ,  AVG(Gehalt)
FROM        Mitarbeiter
GROUP  BY   Bankfiliale_ID ;
```



BANKFILIALE_ID	Avg (Gehalt)
	44000
1	4100
6	4200
11	4600
13	5000
2	3400
14	4875
20	4875
4	4100

21 Zeilen ausgewählt

Das nächste Beispiel erläutert den Zusammenhang zwischen **AVG** und NULL-Werten. Im Versuch soll die durchschnittliche Provision, die ein Mitarbeiter erhält, berechnet werden. Wichtig für diesen Versuch ist, dass nur ein Teil der Mitarbeiter eine Provision erhält.

Listing 9.9: AVG und NULL-Werte in Oracle

```
SELECT SUM(Provision),
       COUNT(Provision) AS Anzahl, ROUND(AVG(Provision), 2) AS "AVG",
       COUNT(NVL(Provision, 0)) AS "Anzahl NVL",
       AVG(NVL(Provision, 0)) AS "AVG NVL"
FROM Mitarbeiter;
```



ANZAHL	AVG	ANZAHL NVL	AVG NVL
33	22,58	100	7,45

1 Zeile ausgewählt

Listing 9.10: AVG und NULL-Werte im MS SQL Server

```
SELECT COUNT(Provision) AS Anzahl, ROUND(AVG(Provision), 2) AS "AVG",
       COUNT(ISNULL(Provision, 0)) AS "Anzahl ISNULL",
       AVG(ISNULL(Provision, 0)) AS "AVG ISNULL"
FROM Mitarbeiter;
```



ANZAHL	AVG	ANZAHL NVL	AVG NVL
33	22,58	100	7,45

1 Zeile ausgewählt

1 Zeile ausgewählt

Je nach dem, ob NULL-Werte durch die `NVL`-Funktion/`ISNULL`-Funktion bereinigt werden oder nicht, wird ein unterschiedliches Ergebnis, durch die `AVG`-Funktion, errechnet.

9.2.4 Die Funktionen MIN und MAX

Die Funktionen `MIN` und `MAX` ermitteln den größten bzw. kleinsten Wert aus einer Menge und können auf nahezu jeden Datentyp angewendet werden. Beispiel ?? zeigt die Anwendung von `MAX` auf Spalten verschiedener Datentypen.

Listing 9.11: Anwendung von MAX auf verschiedene Datentypen

```
SELECT MAX(Nachname), MAX(Geburtsdatum), MAX(PLZ), MAX(Provision)
FROM Mitarbeiter;
```



MAX (NACHNAME)	MAX (GEBURTSDATUM)	MAX (PLZ)	MAX (PROVISION)
Zimmermann	31.05.93	80995	30
1 Zeile ausgewählt			

Zu beachten ist, dass die angezeigten Daten in keiner Beziehung zueinander stehen. Es handelt sich um die Maximalwerte aus den einzelnen Spalten. Der Angestellte Zimmermann hat keinesfalls das Geburtsdatum 31.05.1993 und er bekommt auch keine Provision.

Bezüglich NULL-Werte gibt es, sowohl in Oracle, als auch in MS SQL Server, keine Probleme mit `MIN` oder `MAX`, wie das folgende Beispiel ?? zeigt.

Listing 9.12: Die MAX-Funktion und NULL-Werte (Oracle)

```
SELECT MAX(Provision), MAX(NVL(Provision,0))
FROM Mitarbeiter;
```



MAX (PROVISION)	MAX (NVL (PROVISION, 0))
30	30
1 Zeile ausgewählt	

Das gleiche Beispiel kann in MS SQL Server, mit Hilfe der `ISNULL`-Funktion reproduziert werden.

Alle Eigenschaften, die für die `MAX`-Funktion gelten, gelten uneingeschränkt auch für die `MIN`-Funktion.

9.2.5 Gruppierungen mit mehreren Ebenen

Wie bereits angekündigt, ist es möglich, mit Hilfe der **GROUP BY**-Klausel, eine Abfrage mehrfach zu gruppieren. Eine Mehrfachgruppierung ist immer dann notwendig, wenn innerhalb einer Gruppe weitere Gruppen gebildet werden müssen. Beispiel ?? listet alle Kunden auf, die nach dem 01.01.1995 geboren wurden, gruppiert nach Ort und Strasse.

Listing 9.13: Eine Gruppierung mit mehreren Ebenen

```
SELECT Ort, Strasse, COUNT(*) AS Anzahl
FROM Kunde k INNER JOIN Eigenkunde ek
  ON (k.Kunden_ID = ek.Kunden_ID)
WHERE Geburtsdatum > '01.01.1995'
GROUP BY Ort, Strasse
ORDER BY Ort;
```



ORT	STRASSE	ANZAHL
Aschersleben	Am Markt	1
Bördehue	Plutoweg	1
Bördehue	Okerstraße	1
...
Hecklingen	Turmstraße	1
Hecklingen	Pestalozzistraße	1
Hecklingen	Seestraße	1
...
Staßfurt	Wielandstraße	2
21 Zeilen ausgewählt		

9.3 Gruppierte Abfragen filtern

9.3.1 Die WHERE-Klausel

Das die **WHERE**-Klausel auch auf gruppierte Abfragen angewandt werden kann, ist bereits in Beispiel ?? zu sehen. Wesentlich dabei ist, dass sie vor dem Gruppieren abgearbeitet wird, d. h. es wird die Menge der Zeilen eingeschränkt, die noch gruppiert werden muss. Hierzu ein Beispiel: Mit Hilfe einer Abfrage soll ermittelt werden, wer der jüngste Kunde ist.

Listing 9.14: Wer ist der jüngste Kunde

```
SELECT MAX(Geburtsdatum)
FROM Eigenkunde;
```

MAX (GEBURTSDATUM)

07.04.97

1 Zeile ausgewählt

Im Folgenden wird Beispiel ?? durch eine WHERE-Klausel eingeschränkt.

Listing 9.15: Der jüngste Kunde aus Alsleben

```
SELECT MAX(Geburtsdatum)
FROM Eigenkunde
WHERE Ort LIKE 'Alsleben';
```



MAX (GEBURTSDATUM)

21.05.93

1 Zeile ausgewählt

Die angefügte WHERE-Klausel sorgt dafür, dass die Gruppierung, die durch die MAX-Funktion entsteht, nur auf die Kunden angewandt wird, die in Alsleben wohnen.



Die WHERE-Klausel wird immer vor dem Gruppieren abgearbeitet!

9.3.2 Die HAVING-Klausel

Gerade eben wurde gezeigt, dass mit Hilfe der WHERE-Klausel eine Selektion vor der Gruppierung der Datensätze erreicht werden kann. Was aber ist, wenn eine Auswahl auf gruppierten Datensätzen erfolgen soll? Im folgenden Versuch soll für jede Bankfiliale das niedrigste Gehalt aufgelistet werden, aber nur dann, wenn es größer als 1.500 EUR ist.

Listing 9.16: Ein Versuch... mit Oracle

```
SELECT Bankfiliale_ID, MIN(Gehalt)
FROM Mitarbeiter
WHERE MIN(Gehalt) > 1500
GROUP BY Bankfiliale_ID;

ORA-00934: group function is not allowed here
00934. 00000 - "group function is not allowed here"
```

Listing 9.17: Der gleiche Versuch... mit MS SQL Server

```
SELECT Bankfiliale_ID, MIN(Gehalt)
FROM Mitarbeiter
WHERE MIN(Gehalt) > 1500
GROUP BY Bankfiliale_ID;

Meldung 147, Ebene 15, Status 1, Zeile 3
An aggregate may not appear in the WHERE clause unless it is in a subquery contained in a
Verweise.
```

Beispiel ?? und Beispiel ?? zeigen, dass die Verarbeitung einer Aggregatfunktion in der **WHERE**-Klausel nicht möglich ist. Dies liegt daran, dass, wie bereits erwähnt, die **WHERE**-Klausel schon vor der Gruppierungsphase abgearbeitet wird.

Um das gewünschte Ziel erreichen zu können, muss eine neue Klausel, die **HAVING**-Klausel, eingeführt werden. Sie ermöglicht es, Selektionen auf gruppierten Zeilen durchzuführen. Beispiel ?? muss korrekt lauten:

Listing 9.18: Die HAVING-Klausel

```
SELECT      Bankfiliale_ID ,  MIN(Gehalt)
FROM        Mitarbeiter
GROUP  BY   Bankfiliale_ID
HAVING     MIN(Gehalt) > 1500;
```



BANKFILIALE_ID	MIN(GEHALT)
	30000
1	2000
6	2000
11	2000
7	2000
18	2500
15	2000
16	3000
19	2000

9 Zeilen ausgewählt

Die **HAVING**-Klausel eliminiert, nach dem Gruppieren, alle Datensätze, auf die die Bedingung zutrifft: **MIN(Gehalt) <= 1500**.



Die **HAVING**-Klausel wird auf gruppierte Zeilen angewandt und kann deshalb nur in Verbindung mit der **GROUP BY**-Klausel stehen.

9.4 Die Abarbeitungsreihenfolge des SELECT-Statements

Nachdem nun in den vorangegangenen Kapiteln alle standardisierten Klauseln des **SELECT**-Kommandos behandelt wurden, stellt sich noch immer die Frage: „In welcher Reihenfolge werden die Klauseln des **SELECT**-Kommandos abgearbeitet?“. Die korrekte Antwort lautet:

1. **FROM**
2. **WHERE**
3. **GROUP BY**
4. **HAVING**
5. **SELECT**
6. **ORDER BY**

Zuerst wird mit Hilfe der **FROM**-Klausel ermittelt, auf welche Tabellen sich das **SELECT**-Statement bezieht. Im zweiten Schritt filtert die **WHERE**-Klausel alle Zeilen aus den Quelltabellen, die für das Statement nicht mehr relevant sind. Die beiden Klauseln **GROUP BY** und **HAVING** sorgen für Gruppierungen und das Filtern von gruppierten Zeilen. Zu guter Letzt werden die **SELECT**- und die **ORDER BY**-Klausel abgearbeitet, so dass das Ergebnis auf dem Bildschirm ausgegeben werden kann.

9.5 Übungen - Gruppenfunktionen

1. Schreiben Sie eine Abfrage, die das höchste und das niedrigste Gehalt, das Durchschnittsgehalt und die Summe aller Gehälter ausgibt. Beschriften Sie die Spalten, wie es in der Lösung zu sehen ist.



Maximum	Minimum	Mittelwert	Summe
88000	1000	7255	725500

1 Zeile ausgewählt

2. Verändern Sie die Abfrage aus der vorangegangenen Abfrage so, dass die Informationen für jede einzelne Bankfiliale angezeigt werden. Sortieren sie das Ergebnis nach den IDs der Bankfilialen.



BANKFILIALE_ID	Maximum	Minimum	Mittelwert	Summe
1	12000	2000	4100	20500
2	12000	1000	3400	17000
3	12000	1000	3900	19500
4	12000	1500	4100	20500
5	12000	1000	4200	21000
6	12000	2000	4200	21000

20 Zeilen ausgewählt

3. Schreiben Sie eine Abfrage, die die Anzahl der Mitarbeiter pro Bankfiliale ausgibt. Beschriften Sie die Spalten so, wie es in der Lösung zu sehen ist und sortieren Sie das Ergebnis nach den IDs der Filialen.



BANKFILIALE_ID	Anzahl
1	5
2	5
3	5
4	5
5	5
6	5

20 Zeilen ausgewählt

4. Schreiben Sie eine Abfrage, die für jeden Ort einzeln, die Anzahl der Eigenkunden zählt, die vor dem „01.01.1990“ 18 Jahre alt waren.



ORT	ANZAHL
Nienburg	5
Calbe	3
Hecklingen	3
Dresden	1
Berlin	2
Schönebeck	1
Leipzig	1

15 Zeilen ausgewählt

5. Erstellen Sie eine Abfrage, die für alle bankeigenen Kunden die Buchungen auf deren Girokonten zählt. Interessant sind nur Buchungen mit einem Betrag >10.000 EUR. Sortieren Sie die Abfrage nach der Spalte KONTO_ID.



KONTO_ID	COUNT (*)
1	8
2	11
3	10
5	7
6	8
7	9
9	8

367 Zeilen ausgewählt

6. Schreiben Sie eine Abfrage, die alle Mitarbeiter anzeigt, deren Gehalt um mehr als 4.000 EUR niedriger ist, als das Durchschnittsgehalt aller Mitarbeiter.



VORNAME	NACHNAME	GEHALT
Louis	Wagner	1500
Lukas	Weiβ	2000
Maja	Keller	1000
Karolin	Klingner	2000
Elias	Sindermann	1000

59 Zeilen ausgewählt

59 Zeilen ausgewählt

7. Schreiben Sie eine Abfrage, die alle Mitarbeiter anzeigt, die höchstens zwei Jahre älter sind, als der jüngste Mitarbeiter in deren Bankfiliale!



VORNAME	NACHNAME	GEBURTSDATUM	Juengster Mitarbeiter
Johannes	Lehmann	1992-11-07	1992-11-07
Dirk	Peters	1991-09-16	1992-11-07
Stefan	Beck	1983-12-21	1984-11-16
Martin	Schacke	1984-11-16	1984-11-16
Lukas	Weiβ	1989-03-23	1989-03-23
Alexander	Weber	1987-11-05	1989-03-23
Anne	Zimmermann	1991-01-28	1991-01-28

32 Zeilen ausgewählt

8. Schreiben Sie eine Abfrage, die zu jedem Filialleiter, das Gehalt seines am schlechtesten bezahlten Mitarbeiters anzeigt. Sortieren Sie die Abfrage nach den Bankfilial-IDs der Filialleiter.



VORNAME	NACHNAME	GEHALT	Kleindestes Gehalt
Dirk	Peters	12000	2000
Louis	Winter	12000	1000
Alexander	Weber	12000	1000
Sophie	Schwarz	12000	1500
Jessica	Weber	12000	1000

20 Zeilen ausgewählt

9.6 Lösungen - Gruppenfunktionen

1. Schreiben Sie eine Abfrage, die das höchste und das niedrigste Gehalt, das Durchschnittsgehalt und die Summe aller Gehälter ausgibt. Beschriften Sie die Spalten, wie es in der Lösung zu sehen ist.



```
SELECT MAX(Gehalt) AS "Maximum", MIN(Gehalt) AS "Minimum",
       AVG(Gehalt) AS "Mittelwert", SUM(Gehalt) AS "Summe"
  FROM Mitarbeiter;
```

2. Verändern Sie die Abfrage aus der vorangegangenen Abfrage so, dass die Informationen für jede einzelne Bankfiliale angezeigt werden. Sortieren sie das Ergebnis nach den IDs der Bankfilialen.



```
SELECT Bankfiliale_ID, MAX(Gehalt) AS "Maximum", MIN(Gehalt) AS "Minimum",
       AVG(Gehalt) AS "Mittelwert", SUM(Gehalt) AS "Summe"
  FROM Mitarbeiter
 WHERE Bankfiliale_ID IS NOT NULL
 GROUP BY Bankfiliale_ID
 ORDER BY Bankfiliale_ID;
```

3. Schreiben Sie eine Abfrage, die die Anzahl der Mitarbeiter pro Bankfiliale ausgibt. Beschriften Sie die Spalten so, wie es in der Lösung zu sehen ist und sortieren Sie das Ergebnis nach den IDs der Filialen.



```
SELECT Bankfiliale_ID, COUNT(*) AS "Anzahl"
  FROM Mitarbeiter
 GROUP BY Bankfiliale_ID
 ORDER BY Bankfiliale_ID;
```

4. Schreiben Sie eine Abfrage, die für jeden Ort einzeln, die Anzahl der Eigenkunden zählt, die vor dem „01.01.1990“ 18 Jahre alt waren.



```
SELECT Ort, COUNT(*) AS "Anzahl"
  FROM Eigenkunde ek
 WHERE Geburtsdatum + INTERVAL '18' YEAR <
          TO_DATE('01.01.1990', 'DD.MM.YYYY')
 GROUP BY Ort;
```



```
SELECT Ort, COUNT(*) AS "Anzahl"
FROM Eigenkunde ek
WHERE DATEADD(YEAR, 18, Geburtsdatum) <
      CONVERT(DATETIME2, '01.01.1990', 104)
GROUP BY Ort;
```

5. Erstellen Sie eine Abfrage, die für alle bankeigenen Kunden die Buchungen auf deren Girokonten zählt. Interessant sind nur Buchungen mit einem Betrag >10.000 EUR. Sortieren Sie die Abfrage nach der Spalte KONTO_ID.



```
SELECT ekk.Konto_ID, COUNT(*)
FROM EigenkundeKonto ekk INNER JOIN Girokonto g
    ON (ekk.Konto_ID = g.Konto_ID)
    INNER JOIN Buchung b ON (g.Konto_ID = b.Konto_ID)
WHERE b.Betrag > 10000
GROUP BY ekk.Konto_ID
ORDER BY 1;
```

6. Schreiben Sie eine Abfrage, die alle Mitarbeiter anzeigt, deren Gehalt um mehr als 4.000 EUR niedriger ist, als das Durchschnittsgehalt aller Mitarbeiter.



```
SELECT m.Vorname, m.Nachname, m.Gehalt
FROM Mitarbeiter m, Mitarbeiter v
GROUP BY m.Mitarbeiter_ID, m.Vorname, m.Nachname, m.Gehalt
HAVING (m.Gehalt + 4000) < AVG(v.Gehalt);
```

7. Schreiben Sie eine Abfrage, die alle Mitarbeiter anzeigt, die höchstens zwei Jahre älter sind, als der jüngste Mitarbeiter in deren Bankfiliale!



```
SELECT m.Vorname, m.Nachname, m.Geburtsdatum,
       MAX(a.Geburtsdatum) AS "JUENGSTER MITARBEITER"
FROM Mitarbeiter m INNER JOIN Mitarbeiter a
    ON (m.Bankfiliale_ID = a.Bankfiliale_ID)
GROUP BY m.Mitarbeiter_ID, m.Vorname, m.Nachname, m.Geburtsdatum
HAVING m.Geburtsdatum BETWEEN MAX(a.Geburtsdatum) - INTERVAL '2' YEAR AND
                           MAX(a.Geburtsdatum);
```



```
SELECT      m.Vorname ,  m.Nachname ,  m.Geburtsdatum ,
            MAX(a.Geburtsdatum) AS "JUENGSTER MITARBEITER"
FROM        Mitarbeiter m INNER JOIN Mitarbeiter a
            ON (m.Bankfiliale_ID = a.Bankfiliale_ID)
GROUP BY    m.Mitarbeiter_ID ,  m.Vorname ,  m.Nachname ,  m.Geburtsdatum
HAVING      m.Geburtsdatum BETWEEN DATEADD(YEAR, -2, MAX(a.Geburtsdatum)) AND
            MAX(a.Geburtsdatum);
```

8. Schreiben Sie eine Abfrage, die zu jedem Filialleiter, das Gehalt seines am schlechtesten bezahlten Mitarbeiters anzeigt. Sortieren Sie die Abfrage nach den Bankfilial-IDs der Filialleiter.

```
SELECT      v.Vorname, v.Nachname, v.Gehalt,  
           MIN(m.Gehalt) AS "Kleindestes Gehalt"  
  FROM      Mitarbeiter m INNER JOIN Mitarbeiter v  
            ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)  
 WHERE     v.Bankfiliale_ID IS NOT NULL  
 GROUP BY  v.Mitarbeiter_ID, v.Vorname, v.Nachname, v.Gehalt, v.Bankfiliale_ID  
 ORDER BY  v.Bankfiliale_ID;
```



10 Unterabfragen (Subqueries)

Inhaltsangabe

10.1 Grundsätzliches zu Unterabfragen

10.1.1 Was sind Unterabfragen?

Unterabfragen sind Abfragen, die in eine andere Abfrage, die Hauptabfrage oder „Mainquery“, eingebettet werden. Dies kann an mehreren Stellen geschehen.

- **SELECT**-Klausel
- **FROM**-Klausel (Inlineview)
- **WHERE**-Klausel
- **HAVING**-Klausel

Abb. 10.1:
Unterabfragen



Für Unterabfragen gibt es die unterschiedlichsten Bezeichnungen.

- Subquery
- Inner query
- Nested query

10.1.2 Wann sind Unterabfragen notwendig?

Mit Hilfe von SQL können zwei verschiedene Arten von Problemstellungen gelöst werden:

- Einschrittige Problemstellungen
- Mehrschrittige Problemstellungen

Unter einer einschrittigen Problemstellung versteht man die Art von Fragestellung, die mit einer einzigen Abfrage (einem einzigen Arbeitsschritt) gelöst werden kann, so wie dies in den vorangegangenen Kapiteln der Fall war.

Mehrschrittige Problemstellungen erfordern, wie der Name es sagt, mehrere Abfragen, die aufeinander aufbauen (die eine Abfrage benötigt das Ergebnis der anderen), um zu einer Lösung zu kommen. Eine solche Problemstellung könnte z. B. so lauten: „Wie hoch ist das Gehalt des Vorgesetzten der Mitarbeiterin *Lena Große*?“

Diese Frage lässt sich in zwei Fragen teilen:

1. Wer ist der Vorgesetzte von Lena Große?
2. Wie hoch ist dessen Gehalt?

Die Antworten zu beiden Fragen lassen sich sehr einfach als SQL-Statements formulieren.

Listing 10.1: Wer ist der Vorgesetzte von Lena Grosse

```
SELECT Vorgesetzter_ID
FROM Mitarbeiter
WHERE Vorname LIKE 'Lena' AND Nachname LIKE 'Grosse';
```



VORGESETZTER_ID
6

1 Zeile ausgewählt

Listing 10.2: Wie hoch ist dessen Gehalt

```
SELECT Gehalt
FROM Mitarbeiter
WHERE Mitarbeiter_ID = 6;
```



GEHALT
30000

1 Zeile ausgewählt

Mit Hilfe der beiden Abfragen wurde die Antwort ermittelt: „Der Vorgesetzte von Lena Große hat ein Gehalt von 30.000 EUR.“ Durch das Kombinieren beider Queries, lässt sich diese Aufgabe viel eleganter lösen. Beispiel ?? zeigt einen möglichen Lösungsansatz.

Listing 10.3: Wie hoch ist das Gehalt des Vorgesetzten der Mitarbeiterin *Lena Große*?

```
SELECT Gehalt
FROM Mitarbeiter
WHERE Mitarbeiter_ID = (SELECT Vorgesetzter_ID
                        FROM Mitarbeiter
                        WHERE Vorname LIKE 'Lena'
                        AND Nachname LIKE 'Grosse');
```



GEHALT

30000

1 Zeile ausgewählt



Das DBMS arbeitet bei einer solchen Auswahlabfrage immer zuerst die Unterabfrage(n) ab!

10.1.3 Regeln für Unterabfragen

Für die Anwendung von Unterabfragen gelten die folgenden Grundsätze:

- Unterabfragen stehen immer in Klammern!
- Es können alle ihnen bisher bekannten Operatoren eingesetzt werden!
- Unterabfragen **sollten immer ohne ORDER BY-Klausel** erstellt werden!

Die Aussage, dass Unterabfragen immer ohne **ORDER BY** verwendet werden sollten, röhrt daher, dass falls eine Sortierung in der Hauptabfrage stattfindet, zuerst in der Unterabfrage sortiert wird und anschließend nochmals in der Hauptabfrage. Dies führt zu unnötiger Sortierarbeit, die die Datenbank belastet.



In MS SQL Server darf eine Unterabfrage kein **ORDER BY** enthalten. Das DBMS antwort sonst mit einer Fehlermeldung (Meldung 1033, Ebene 15).

10.1.4 Arten von Unterabfragen

Grundsätzlich gibt es vier unterschiedliche Arten von Unterabfragen:

- Skalare Unterabfragen: Eine solche Abfrage liefert exakt einen Wert zurück.
- Einspaltige Unterabfragen: Dieser Abfragetyp liefert mehrere Werte aus einer Spalte zurück.
- Mehrspaltige Unterabfragen: Mit einer solchen Abfrage werden Werte mehrerer Spalten zurückgeliefert.
- Korrelierte Unterabfragen: Ihre Ausführung ist von der Hauptabfrage abhängig.



10.2 Skalare Unterabfragen (Scalar Subqueries)

Skalar:

Größe aus der Mathematik, die durch die Angabe eines einzelnen Wertes genau definiert werden kann.

Beispiele für Skalare sind: Gehalt, Provision, ...

Skalare Unterabfragen zeichnen sich dadurch aus, dass sie genau einen einzigen Wert zurückliefern. Dies wird mit Hilfe einer entsprechenden **WHERE**-Klausel innerhalb der Unterabfrage erreicht. Ergibt die Abfrage kein Ergebnis, wird NULL zurückgeliefert. Ein erstes Beispiel für diese Art von Unterabfrage war in Beispiel ?? zu sehen. Es wird nur das GEHALT eines einzigen Angestellten angezeigt.

10.2.1 Wo können skalare Unterabfragen stehen?

Skalare Unterabfragen können in allen in Abschnitt ?? erwähnten Klauseln stehen.

Skalare Unterabfragen in der SELECT-Klausel

Skalare Unterabfragen sind die einzigen, die in der **SELECT**-Klausel eines SQL-Statements stehen dürfen. Sie können beispielsweise dazu dienen, um einen Outer-Join zu vermeiden, meist ist jedoch die Join-Variante sehr viel performanter. Aus diesem Grund sollten skalare Unterabfragen in der **SELECT**-Klausel absolut vermieden werden.



Skalare Unterabfragen in der **SELECT**-Klausel sollten unter allen Umständen vermieden werden!

Skalare Unterabfragen in der WHERE-Klausel

Die **WHERE**-Klausel ist der Ort, an dem skalare Unterabfragen am häufigsten anzutreffen sind. Sie dienen zur Berechnung von Werten, mit deren Hilfe das Resultat der Hauptabfrage eingeschränkt wird (siehe Beispiel ??).

Skalare Unterabfragen in der Having-Klausel

Hier gelten die gleichen Grundsätze, wie in der **WHERE**-Klausel. Der einzige Unterschied ist, das hier ein Aggregat mit dem Resultat einer skalaren Unterabfrage verglichen werden kann.

10.2.2 Fehlerquellen in skalaren Unterabfragen

Die häufigste Fehlerquelle, im Umgang mit skalaren Unterabfragen, ist eine falsche **WHERE**-Klausel. Schränkt sie das Ergebnis der Unterabfrage nicht genügend ein, wird mehr als ein Datensatz/Wert zurückgeliefert und die Datenbank antwortet mit einer Fehlermeldung.

Listing 10.4: Mehr als eine Zeile: Fehlermeldung in Oracle

```
ORA-01427: Unterabfrage fuer eine Zeile liefert mehr als eine Zeile
```

Listing 10.5: Mehr als eine Zeile: Fehlermeldung in SQL Server

```
Meldung 512, Ebene 16, Status 1, Zeile 1
Die Unterabfrage hat mehr als einen Wert zurueckgegeben. Das ist nicht
zulaessig, wenn die Unterabfrage auf =, !=, <, <=, > oder >= folgt oder
als Ausdruck verwendet wird.
```

Hier ein Beispiel zu diesen Fehlermeldungen: Es soll das Geburtsdatum des Vorgesetzten der Mitarbeiterin „Große“ ermittelt werden.

Listing 10.6: Eine Single Row Unterabfrage mit Problemen!

```
SELECT Geburtsdatum
FROM Mitarbeiter
WHERE Mitarbeiter_ID = (SELECT Vorgesetzter_ID
                         FROM Mitarbeiter
                         WHERE LOWER(Nachname) LIKE 'grosse');
```

Das Problem bei dieser Abfrage ist, dass die Tabelle MITARBEITER zwei Angestellte mit dem Namen „Große“ enthält. Das bedeutet, die Unterabfrage liefert mehr als einen Wert zurück, so dass der Vergleich mit einem Single Row Operator scheitert.



Als Single Row Operatoren werden relationale Operatoren bezeichnet, die einen Wert auf ihrer linken Seite mit genau einem Wert auf ihrer rechten Seite vergleichen können. Hierzu zählen: = >= <= < > != LIKE

10.3 Einspaltige Unterabfragen

Diese Kategorie der Unterabfragen unterscheidet sich von den skalaren dahingehend, dass sie eine einspaltige Liste von mehreren Werten (Vektor) zurückliefern und das sie nicht in der **SELECT**-Klausel eines SQL-Statements vorkommen dürfen.

10.3.1 Einspaltige Unterabfragen in WHERE- und HAVING-Klausel

IN (bekannt aus Abschnitt ??) ist der einzige Operator, der auf seiner rechten Seite nicht nur einen einzelnen Wert, sondern eine ganze Wertemenge verarbeiten kann. Dies kann eine konstante Menge sein, so wie dies bisher der Fall war, aber es kann auch eine, durch eine Query dynamisch generierte Menge sein. Beispiel ?? zeigt den Einsatz des **IN**-Operators. Es muss eine Liste aller Kunden ermittelt werden, die vor dem „01.01.1980“ ein Konto bei der Bank eröffnet haben.

Listing 10.7: **IN** mit Unterabfrage

```
SELECT Vorname, Nachname
FROM Kunde
WHERE Kunden_ID IN (SELECT Kunden_ID
                      FROM EigenkundeKonto
                      WHERE Eroeffnungsdatum < TO_DATE('01.01.1980'));
```

Listing 10.8: Die Fehlermeldung in SQL Server

```
SELECT Vorname, Nachname
FROM Kunde
WHERE Kunden_ID IN (SELECT Kunden_ID
                      FROM EigenkundeKonto
                      WHERE Eroeffnungsdatum < CONVERT(DATETIME2,'01.01.1980',104));
```

VORNAME	NACHNAME
Jan	Wei�
Petra	Berger
Karolin	Lange
Tom	Hartmann
28 Zeilen ausgewhlt	

Auf die gleiche Art und Weise, wie in Beispiel ?? gezeigt, knnen einspaltige Unterabfragen auch in einer **HAVING**-Klausel eingesetzt werden, was jedoch nur sehr selten vorkommt.

10.3.2 Existenzprüfungen

Der EXISTS-Operator

Der Name *Existenzprüfung* sagt ohne Umschweife aus, worum es geht. Mit Hilfe des Operators **EXISTS** kann die Existenz bestimmter Daten geprüft werden. Beispiel ?? zeigt auf worum es sich hierbei handelt. Es soll eine Liste der Bankfilialen ermittelt werden, in denen Mitarbeiter eingesetzt sind.

Listing 10.9: Der **EXISTS**-Operator

```
SELECT Strasse, Hausnummer, PLZ, Ort
FROM   Bankfiliale b
WHERE  EXISTS (SELECT 1
                FROM   Mitarbeiter m
                WHERE  b.Bankfiliale_ID = m.Bankfiliale_ID);
```



STRASSE	HAUSNUMMER	PLZ	ORT
Poststraße	1	06449	Aschersleben
Markt	5	06449	Aschersleben
Goethestraße	4	39240	Calbe
Lessingstraße	1	06406	Bernburg
Schillerstraße	7	39240	Barby

20 Zeilen ausgewählt

Das Ergebnis dieser Auswahlabfrage sind alle Bankfilialen, in denen Mitarbeiter arbeiten. Es verbleibt eine Filiale ohne Mitarbeiter.

Was geschieht in dieser Abfrage nun in welcher Reihenfolge?

1. Die **FROM**-Klausel der Hauptabfrage wird ausgewehrtet und die erforderlichen Daten werden ermittelt.
2. Die **FROM**-Klausel der Unterabfrage wird ausgewehrtet und die erforderlichen Daten werden ermittelt.
3. Die **WHERE**-Klausel der Unterabfrage wird ausgeführt. Der Join zwischen **BANKFILIALE** und **MITARBEITER** wird gebildet.
4. Die **WHERE**-Klausel der Hauptabfrage wird ausgeführt.
5. Die **SELECT**-Klausel der Hauptabfrage liefert die benötigten Daten.

Das Besondere an dieser Form der Abfrage ist die **WHERE**-Klausel der Unterabfrage. Dort wird die Tabelle **BANKFILIALE** (Hauptabfrage) mit der Tabelle **MITARBEITER** (Unterabfrage) verknüpft. Die Unterabfrage kann somit auf die Datensätze der Hauptabfrage zugreifen.



Werden die Tabellen einer Unterabfrage mit einer Tabelle der Hauptabfrage verknüpft, spricht man von einer „korrelierten Unterabfrage“.

Für die Ausführung des gesamten Statements bedeutet dies, dass die Unterabfrage nicht nur einmal, sondern mehrfach ausgeführt werden muss. Genauer gesagt wird die Unterabfrage für jede Zeile der Hauptabfrage einmal ausgeführt. Bezogen auf Beispiel ?? bedeutet dies, dass die Unterabfrage 21 mal ausgeführt wird, da die Tabelle Bankfiliale 21 Datensätze hat. Die Mehrfachausführung der Unterabfrage ist notwendig, da für jede Bankfiliale einzeln geprüft werden muss, ob es dort Mitarbeiter gibt oder nicht.

Eine weitere Besonderheit dieser Art von Abfrage ist die **SELECT**-Klausel der Unterabfrage. Dort stehen keine Spaltenbezeichner und auch kein *. Statt dessen wird hier ein Literal, eine 1 (eins) verwendet. Der Hintergrund hierfür ist, dass die **SELECT**-Klausel der Unterabfrage für die Ausführung des gesamten Statements keine Bedeutung hat. Es wird nur geprüft, ob für jeden Datensatz der Hauptabfrage ein Datensatz in der Unterabfrage existiert. Das bedeutet, dass sobald die Unterabfrage eine Zeile zurückliefert die Bedingung erfüllt ist und der Datensatz der Hauptabfrage angezeigt wird.

Der NOT EXISTS-Operator

Der **NOT EXISTS**-Operator stellt das Pendant zum **EXISTS**-Operator dar. Müssen beispielsweise alle Filialen ermittelt werden, in denen keine Mitarbeiter arbeiten kommt **NOT EXISTS** zum Einsatz.

Listing 10.10: Der **NOT EXISTS**-Operator

```
SELECT Strasse , Hausnummer , PLZ , Ort
FROM   Bankfiliale b
WHERE  NOT EXISTS (SELECT 1
                    FROM   Mitarbeiter m
                    WHERE  b.Bankfiliale_ID = m.Bankfiliale_ID);
```



STRASSE	HAUSNUMMER	PLZ	ORT
Kurze Gasse	47	06425	Alsleben

1 Zeile ausgewählt

10.4 Inlineviews / Derived Tables

In Abschnitt ?? wurde bereits erwähnt, dass eine Unterabfrage auch in der **FROM**-Klausel eines SQL-Statements stehen kann.



Eine Unterabfrage in der `FROM`-Klausel wird in Oracle als „Inlineview“ und in MS SQL Server als „Derived Table“ bezeichnet.

Beispiel ?? zeigt ein SQL-Statement, welches eine Inlineview nutzt.

Listing 10.11: Eine Inlineview

```
SELECT Vorname, Nachname, MinGehalt
  FROM (SELECT Bankfiliale_ID, MIN(Gehalt) MinGehalt
          FROM Mitarbeiter
         GROUP BY Bankfiliale_ID) m1
    INNER JOIN Mitarbeiter m
      ON (m1.Bankfiliale_ID = m.Bankfiliale_ID)
 WHERE m.Gehalt = m1.MinGehalt;
```



VORNAME	NACHNAME	MINGEHALT
Johannes	Lehmann	2000
Louis	Schmitz	2000
Marie	Kipp	2000
Martin	Schacke	1000
Oliver	Wolf	1000
Hans	Schumacher	1000
Lena	Herrmann	1500
29 Zeilen ausgewählt		

In Beispiel ?? wird die Inlineview dazu benutzt, um das kleinste Gehalt je Abteilung zu berechnen. Mit Hilfe des Joins wird sie mit der Tabelle MITARBEITER verknüpft, so dass die Attribute VORNAME und NACHNAME angezeigt werden können, ohne in Konflikt mit der `GROUP BY`-Klausel zu kommen.



Inlineviews bieten eine gute Möglichkeit, um gruppierte und ungruppierte Informationen in einer Abfrage gemeinsam anzeigen zu können.

10.5 Top N Analysen

Die Top N Analyse ist ein Verfahren, bei dem Datensätze in ein Ranking eingeordnet werden. Hiermit werden Fragestellungen geklärt wie z. B.:

- Die 3 reichsten Kunden anzeigen
- Die 5 Mitarbeiter mit den höchsten Gehältern auflisten
- Die beiden größten Schuldner der Bank ermitteln

Beide Datenbankmanagementsysteme beherrschen diese Technik, gehen dabei aber unterschiedliche Wege.

10.5.1 Die Top N Analyse in Oracle

Die Top N Analyse funktioniert in Oracle mit Hilfe einer sortierten Inlineview und einer Pseudospalte Namens ROWNUM.

Die Pseudospalte Rownum

Mit der Bezeichnung „Pseudospalte“ ist gemeint, dass die ROWNUM keine tatsächlich vorhandene Spalte ist, obwohl sie in jeder Abfrage verwendet werden kann. Sie bietet die Möglichkeit, die Ergebnissezeilen einer Abfrage fortlaufend zu nummerieren (1, 2, 3, ..., N). Zu beachten ist dabei, dass eine Zeile in einer Oracle-Datenbank keine feste Nummerierung hat. Diese wird erst im Ergebnis einer Abfrage zugeordnet.

Listing 10.12: Ein einfaches Beispiel für die Rownum

```
SELECT Rownum, Vorname, Nachname
FROM Mitarbeiter
WHERE Ort LIKE 'Aschersleben';
```



ROWNUM	VORNAME	NACHNAME
1	Max	Winter
2	Alexander	Weber
3	Leni	Dühning

3 Zeilen ausgewählt



Eine Tabellenzeile hat keine feste Nummerierung. Die Rownum wird während der Abarbeitung einer Abfrage zugewiesen.

Eine weitere, entscheidende Tatsache ist, dass die ROWNUM erst nach der Abarbeitung der WHERE-Klausel zugeordnet wird, aber noch bevor Gruppierungen oder Sortierungen ausgeführt werden. Aus diesem Grund, wird die Abfrage in Beispiel ?? ein falsches Ergebnis liefern, da die Sortierung hätte zuerst stattfinden müssen. Hier werden höchstwahrscheinlich nicht die beiden größten Guthaben, sondern zwei beliebige Guthaben angezeigt. Welche Zeilen gelistet werden hängt davon ab, welche die Abfrage zuerst ermittelt.

Listing 10.13: Falsche Anwendung der Rownum-Pseudospalte

```
SELECT      Konto_ID ,  Guthaben
FROM        Girokonto
WHERE       Rownum < 3
ORDER BY    Guthaben DESC;
```



KONTO_ID	GUTHABEN
1	111316,9
2	96340,2

2 Zeilen ausgewählt



Die Rownum wird erst nach Abarbeitung der WHERE-Klausel, aber noch vor allen Gruppierungen und Sortierungen hinzugefügt.

Ein dritter „Stolperstein“, in Zusammenhang mit der ROWNUM ist, dass die ROWNUM erst inkrementiert wird, wenn sie zugewiesen wurde. Das soll heißen, dass die WHERE-Klausel in Beispiel ?? ebenfalls fehlschlägt, da nach allen Rownums größer eins gefragt wird, ohne das Rownum eins jemals zugewiesen worden wäre (ohne 1 keine 2).

Listing 10.14: Erneut eine falsche Anwendung der Rownum

```
SELECT      Konto_ID ,  Guthaben
FROM        Girokonto
WHERE       Rownum > 3
ORDER BY    Guthaben DESC;
```



KONTO_ID	GUTHABEN
0 Zeilen ausgewählt	
0 Zeilen ausgewählt	

Die Lösung für diese Probleme besteht nun darin,

1. dass niemals einer der beiden Operatoren > oder >= in Zusammenhang mit der ROWNUM verwendet werden sollte und
2. dass die Abfrage aus Beispiel ?? in eine Inlineview geschachtelt wird.

Durchführung der Top N Analyse

Die korrekte Form der Top N Analyse sieht in Oracle wie folgt aus:

Listing 10.15: Eine korrekt funktionierende Top N Analyse in Oracle

```
SELECT *
FROM  (SELECT Konto_ID , Guthaben
       FROM Girokonto
      ORDER BY Guthaben DESC)
WHERE Rownum < 3;
```

KONTO_ID	GUTHABEN
362	147670,3
198	147264

2 Zeilen ausgewählt

Im Gegensatz zu Beispiel ?? werden hier wirklich die beiden größten Gehälter angezeigt. Warum dies so ist, kann durch die Abarbeitungsreihenfolge der Abfrage aus Beispiel ?? erklärt werden.

1. `FROM`-Klausel der Inlineview
2. `SELECT`- und `ORDER BY`-Klausel der Inlineview
3. `FROM`-Klausel der Hauptabfrage
4. Zuweisung der ROWNUM
5. Ausführung der `WHERE`-Klausel der Hauptabfrage
6. `SELECT`-Klausel der Hauptabfrage

In Beispiel ?? wird also zuerst nummeriert und dann selektiert.

10.5.2 Die Top N Analyse in MS SQL Server

In Microsoft SQL Server existiert eigens der Operator `TOP` zur Durchführung von Top N Analysen. Er wird in der `SELECT`-Klausel eingesetzt und legt fest, wie viele Zeilen angezeigt werden.

Listing 10.16: Top N Analyse in MS SQL Server

```
SELECT TOP (2) Konto_ID, Guthaben
FROM Girokonto
ORDER BY Guthaben DESC;
```

KONTO_ID	GUTHABEN
362	147670,3
198	147264

2 Zeilen ausgewählt
2 Zeilen ausgewählt



Durch die Angabe von **TOP** (2) werden nur die ersten zwei Zeilen der Ergebnismenge angezeigt.

10.6 Pivot-Tabellen

Mit MS SQL Server 2005 bzw. Oracle 11g R1 wurden der **PIVOT** und der **UNPIVOT**-Operator eingeführt. Diese ermöglichen die einfache Erstellung von Pivottabellen.



In einer Pivottabelle werden Daten, die im Zeilenformat vorliegen, im Spaltenformat angezeigt oder umgekehrt. Das „Drehen“ der Daten wird als „Pivoting“ bezeichnet, woraus sich der Name für diese Tabellen ableitet.

- **PIVOT:** Dreht Daten die zeilenweise vorliegen so, dass eine spaltenweise Darstellung möglich ist.
- **UNPIVOT:** Dreht Daten die spaltenweise vorliegen so, dass eine zeilenweise Darstellung möglich ist.

10.6.1 Der PIVOT-Operator (Oracle)

Die Möglichkeiten, die der **PIVOT**-Operator bietet, werden anhand des folgenden Beispiels verdeutlicht. Für die Filialen 1 bis 3 sollen die jeweils kleinsten Gehälter angezeigt werden.

Listing 10.17: Die niedrigsten Gehälter in den Filialen 1 bis 3

```
SELECT      Bankfiliale_ID ,  MIN(Gehalt)
FROM        Mitarbeiter
WHERE       Bankfiliale_ID  IN  (1, 2, 3)
GROUP  BY   Bankfiliale_ID;
```



BANKFILIALE_ID	MIN(GEHALT)
1	2000
2	1000
3	1000

3 Zeilen ausgewählt

In Beispiel ?? werden die gewünschten Zahlen ermittelt. Die Darstellung der Gehälter erfolgt zeilenweise. Sollen die gleichen Zahlen spaltenweise dargestellt werden, wird der **PIVOT**-Operator benötigt. Beispiel ?? zeigt dessen Einsatz.

Listing 10.18: Das Ergebnis als Pivottabelle

```
SELECT *
FROM   (SELECT Gehalt, Bankfiliale_ID
        FROM   Mitarbeiter)
PIVOT  (MIN(Gehalt) AS Gehalt FOR Bankfiliale_ID IN (1, 2, 3));
```



1_GEHALT	2_GEHALT	3_GEHALT
2000	1000	1000

1 Zeile ausgewählt

Die Syntax des PIVOT-Operators

Da das SQL-Statement aus Beispiel ?? auf den ersten Blick sehr komplex wirkt, ist es notwendig, es an dieser Stelle im Detail zu betrachten.

Für die Ausführung des Pivotings wird in Beispiel ?? eine Inlineview verwendet.

Listing 10.19: Die Inlineview

```
(SELECT Gehalt, Bankfiliale_ID
  FROM   Mitarbeiter)
```

Diese Inlineview legt fest, welche Spalten im Endergebnis der Abfrage zu sehen sein werden. Sie kann beliebig komplex sein. Der **PIVOT**-Operator verarbeitet im zweiten Schritt die Spalten dieser View weiter.

Listing 10.20: Der PIVOT-Operator

```
PIVOT  (MIN(Gehalt) AS Gehalt FOR Bankfiliale_ID IN (1, 2, 3));
```

Die Bedeutung dieses Operators ist:

- Gruppiere nach der Spalte BANKFILIALE_ID.
- Zeige **MIN(Gehalt)** für Bankfiliale_ID = 1.
- Zeige **MIN(Gehalt)** für Bankfiliale_ID = 2.
- Zeige **MIN(Gehalt)** für Bankfiliale_ID = 3.
- Benutzte den Alias „Gehalt“ für den Ausdruck **MIN(Gehalt)**.

Spaltenalias in der FOR-Klausel

Die Spaltenbezeichnungen im Ergebnis von Beispiel ?? werden gebildet, in dem der Name der aggregierten Spalte (hier GEHALT) mit den Werten der **FOR**-Klausel kombiniert werden. Dadurch entstehen die Namen 1_GEHALT, 2_GEHALT und 3_GEHALT. Auch an dieser Stelle sind Aliasnamen möglich.

Listing 10.21: Die **FOR**-Klausel mit Aliasnamen

```
SELECT *
FROM  (SELECT Gehalt, Bankfiliale_ID
       FROM Mitarbeiter)
PIVOT (MIN(Gehalt) AS Gehalt FOR Bankfiliale_ID
       IN (1 AS "Filiale 1", 2 AS "Filiale 2", 3 AS "Filiale 3"));
```



Filiale 1_GEHALT	Filiale 2_GEHALT	Filiale 3_GEHALT
2000	1000	1000

1 Zeile ausgewählt

Zusätzliche Spalten zum Pivoting

In einer Pivot-Abfrage können noch weitere Spalten enthalten sein, die nicht aggregiert oder in der **FOR**-Klausel genutzt werden. Diese Spalten werden als zusätzliche Gruppierungsmerkmale genutzt.



Oracle führt eine implizite Gruppierung der Ergebnismenge durch. Diese basiert auf allen nicht gruppierten Spalten, inklusive der Spalten, die in der **FOR**-Klausel genutzt werden.

In Beispiel ?? wird im ersten Schritt nach dem Geburtsjahr, von 1987 bis 1989 gruppiert. Da diese Spalte in der **FOR**-Klausel verwendet wird, wird diese Information in Spaltenform dargestellt.

Die Spalte Ort hingegen, wird in Zeilenform angezeigt, da sie nicht in der **FOR**-Klausel angegeben wurde.



Ob eine Information in Spalten- oder Zeilenform dargestellt wird, hängt davon ab, ob die betreffende Spalte in der **FOR**-Klausel gelistet wurde oder nicht.

Listing 10.22: Zusätzliche Gruppierungen in einer Pivot-Abfrage

```
SELECT *
FROM  (SELECT Gehalt, TO_CHAR(Geburtsdatum, 'YYYY') AS Geburtsdatum, Ort
       FROM Mitarbeiter)
PIVOT (MIN(Gehalt) AS Gehalt FOR Geburtsdatum IN ('1987', '1988', '1989'));
```



ORT	'1987'_GEHALT	'1988'_GEHALT	'1989'_GEHALT
Calbe			
Plötzkau	2500		
Nienburg			
Bernburg			
Dresden			
Hecklingen		3000	
Borne		30000	
Schönebeck			
Giersleben			
Gera		3500	
München			
Egeln			
Güsten			
Seeland		2500	
Ilberstedt			
Bördehue			
Hamburg	12000		
Alsleben			
Schwerin			
Dessau	2500		
Könnern			
Cottbus			
Potsdam	3500	2000	2000
Aschersleben	12000	88000	2000
Magdeburg			3000

25 Zeilen ausgewählt

Die vorangegangenen Beispiele stellen nur einen Einstieg in das Thema „Pivottabellen“ dar. Tatsächlich ist der **PIVOT**-Operator noch weitaus mächtiger.

10.6.2 Der PIVOT-Operator (MS SQL Server)

Die Möglichkeiten, welche der **PIVOT**-Operator bietet, werden anhand des folgenden Beispiels verdeutlicht. Für die Bankfilialen 1 bis 3 sollen die jeweils kleinsten Gehälter angezeigt werden.

Listing 10.23: Die niedrigsten Gehälter in den Filialen 1 bis 3

```
SELECT  Bankfiliale_ID ,  MIN(Gehalt)
FROM    Mitarbeiter
WHERE   Bankfiliale_ID IN (1, 2, 3)
GROUP  BY Bankfiliale_ID;
```



Bankfiliale_ID (Kein Spaltenname)	
1	2000
2	1000
3	1000

3 Zeilen ausgewählt

In Beispiel ?? werden die gewünschten Zahlen ermittelt. Die Darstellung der Gehälter erfolgt zeilenweise. Sollen die gleichen Zahlen spaltenweise dargestellt werden, wird der **PIVOT**-Operator benötigt. Beispiel ?? zeigt dessen Einsatz.

Listing 10.24: Das Ergebnis als Pivottabelle

```
SELECT *
FROM   (SELECT Gehalt, Bankfiliale_ID
        FROM   Mitarbeiter) AS Sourcetable
PIVOT  (MIN(Gehalt)
        FOR Bankfiliale_ID IN ([1], [2], [3])
       ) AS Pivottable;
```



1	2	3
2000	1000	1000

1 Zeile ausgewählt

Die Syntax des PIVOT-Operators

Da das SQL-Statement aus Beispiel ?? auf den ersten Blick sehr komplex wirkt, ist es notwendig, es an dieser Stelle im Detail zu betrachten.

Für die Ausführung des Pivotings wird in Beispiel ?? eine Inlineview verwendet.

Listing 10.25: Die Inlineview

```
(SELECT Gehalt, Bankfiliale_ID
  FROM   Mitarbeiter) AS Sourcetable
```

Diese Inlineview legt fest, welche Spalten im Endergebnis der Abfrage zu sehen sein werden. Sie kann beliebig komplex sein. Der **PIVOT**-Operator verarbeitet im zweiten Schritt die Spalten dieser View weiter.

Listing 10.26: Der **PIVOT**-Operator

```
PIVOT  (MIN(Gehalt) FOR Bankfiliale_ID IN ([1], [2], [3])) AS Pivottable;
```

Die Bedeutung dieses Operators ist:

- Gruppieren nach der Spalte BANKFILIALE_ID.
- Zeige **MIN**(Gehalt) für Bankfiliale_ID = 1.
- Zeige **MIN**(Gehalt) für Bankfiliale_ID = 2.
- Zeige **MIN**(Gehalt) für Bankfiliale_ID = 3.

Für eine Pivotabfrage gelten in MS SQL Server folgende Syntaxregeln:

- Für die Quell-View muss zwingend ein Aliasname vergeben werden. In Beispiel ?? ist dies „Source-table“
- Für die Pivottabelle muss zwingend ein Aliasname vergeben werden. In Beispiel ?? ist dies „Pivotable“
- Es dürfen in der Pivottabelle keine Aliasnamen vergeben werden.
- Die Werte in der **FOR**-Klausel müssen in eckigen Klammern stehen.

Zusätzliche Spalten zum Pivoting

In einer Pivot-Abfrage können noch weitere Spalten enthalten sein, die nicht aggregiert oder in der **FOR**-Klausel genutzt werden. Diese Spalten werden als zusätzliche Gruppierungsmerkmale genutzt.



MS SQL Server führt eine implizite Gruppierung der Ergebnismenge durch. Diese basiert auf allen nicht gruppierten Spalten, inklusive der Spalten, die in der **FOR**-Klausel genutzt werden.

In Beispiel ?? wird im ersten Schritt nach dem Geburtsjahr, von 1987 bis 1989 gruppiert. Da diese Spalte in der **FOR**-Klausel verwendet wird, wird diese Information in Spaltenform dargestellt.

Die Spalte Ort hingegen, wird in Zeilenform angezeigt, da sie nicht in der **FOR**-Klausel angegeben wurde.



Ob eine Information in Spalten- oder Zeilenform dargestellt wird, hängt davon ab, ob die betreffende Spalte in der **FOR**-Klausel gelistet wurde oder nicht.

Listing 10.27: Zusätzliche Gruppierungen in einer Pivot-Abfrage

```

SELECT *
FROM   (SELECT Gehalt, DATEPART(YEAR, Geburtsdatum) AS Geburtsdatum, Ort
       FROM   Mitarbeiter) AS Sourcetable
PIVOT  (MIN(Gehalt)
       FOR Geburtsdatum IN ([1987], [1988], [1989])) AS Pivottable;

```



Ort	1987	1988	1989
Calbe			
Plötzkau	2500		
Nienburg			
Bernburg			
Dresden			
Hecklingen		3000	
Borne		30000	
Schönebeck			
Giersleben			
Gera		3500	
München			
Egeln			
Güsten			
Seeland		2500	
Ilberstedt			
Bördehue			
Hamburg	12000		
Alsleben			
Schwerin			
Dessau	2500		
Könnern			
Cottbus			
Potsdam	3500	2000	2000
Aschersleben	12000	88000	2000
Magdeburg		3000	

25 Zeilen ausgewählt

Die vorangegangenen Beispiele stellen nur einen Einstieg in das Thema „Pivottabellen“ dar. Tatsächlich ist der **PIVOT**-Operator noch weitaus mächtiger.

10.7 Übungen - Unterabfragen

1. Schreiben Sie eine Abfrage, die für alle Eigenkunden, die keinen Berater haben (die nicht in der Tabelle EIGENKUNDEMitarbeiter enthalten sind), den Vor- und den Nachnamen anzeigt.

- Lösen Sie die Aufgabe mit Hilfe des **EXISTS**-Operators!
- Lösen Sie die Aufgabe mit Hilfe des **IN**-Operators!



VORNAME	NACHNAME
Sebastian	Schröder
Udo	Schumacher
Mia	Huber
Simon	Witte
Max	Bunzel
Finn	Fischer
Lara	Meierhöfer
Jannis	Meier

16 Zeilen ausgewählt

2. Erstellen Sie eine Abfrage, die ermittelt, ob es Mitarbeiter gibt (Vorname und Nachname), die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten) und Filialleiter.

- Lösen Sie die Aufgabe mit Hilfe des **EXISTS**-Operators!
- Lösen Sie die Aufgabe mit Hilfe des **IN**-Operators!



VORNAME	NACHNAME
Amelie	Krüger
Anna	Schneider
Chris	Simon
Christian	Haas
Elias	Sindermann
Emilia	Köhler
Emma	Krüger

40 Zeilen ausgewählt
40 Zeilen ausgewählt

3. Schreiben Sie eine Abfrage, die den häufigsten Vornamen der Bankmitarbeiter anzeigt und wie oft dieser in der Tabelle MITARBEITER vorkommt.



VORNAME	ANZAHL
Chris	5

1 Zeile ausgewählt

4. Schreiben Sie eine Abfrage, welche die drei Eigenkunden mit den niedrigsten Guthaben auf den Girokonten anzeigt.



VORNAME	NACHNAME	GUTHABEN
Franz	Walther	-140505,1
Jan	Simon	-98218,6
Philipp	Hartmann	-69705,6

3 Zeilen ausgewählt

5. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass die drei Eigenkunden mit dem niedrigsten Guthaben (Girokonto + Sparbuch) angezeigt werden. Es müssen auch diejenigen Kunden angezeigt werden, die nur ein Girokonto oder nur ein Sparbuch haben!



VORNAME	NACHNAME	SUM (GUTHABEN)
Franz	Walther	-139154,4
Jan	Simon	-98218,6
Philipp	Hartmann	-69065,9

3 Zeilen ausgewählt

6. Schreiben Sie eine Abfrage, die alle Eigenkunden anzeigt, welche im Jahr 1985 keine Buchungen verursacht haben.



VORNAME	NACHNAME
Sarah	Bauer
Sofia	Bauer
Tom	Bauer
Alina	Baumann

285 Zeilen ausgewählt

285 Zeilen ausgewählt

7. Schreiben Sie eine Abfrage, die für jede Bankfiliale den Mitarbeiter mit dem höchsten Gehalt ausgibt.



BANKFILIALE	VORNAME	NACHNAME	GEHALT
Poststraße 1 06449 Aschersleben	Dirk	Peters	12000
Kirchstraße 8 39444 Hecklingen	Leonie	Kaiser	12000
Schmiedestraße 3 39240 Staßfurt	Finn	Köhler	12000
Am Dom 11 06449 Giersleben	Lena	Große	12000
20 Zeilen ausgewählt			

8. Schreiben Sie eine Abfrage, die für jeden Wohnort (EIGENKUNDE.ORT) den Kunden anzeigt, der im Jahr 1987 das höchste Einkommen hatte (Das Einkommen ist die Summe aller Beträge eines Kunden, in der Tabelle BUCHUNG). Sortieren Sie die Abfrage nach den Wohnorten.



ORT	VORNAME	NACHNAME	BETRAG
Alsleben	Peter	Koch	57855,4
Aschersleben	Lara	Dühning	2395,7
Barby	Chris	Beck	-6817,8
30 Zeilen ausgewählt			

9. Erstellen Sie eine Abfrage, die die Umsätze der Bank (SUM(Buchung.Betrag)) für die Jahre 1985 bis einschließlich 1989 als Pivottabelle anzeigt.



'1985'	'1986'	'1987'	'1988'	'1989'
559132,5	539497,2	-2036841,3	1081361	1027003,1
1 Zeile ausgewählt				

10. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass die Beträge innerhalb der einzelnen Jahre nach Quartalen aufgeteilt werden.



QUARTAL	'1985'	'1986'	'1987'	'1988'	'1989'
1	32204,8	985,2	2981,1	176852	9777,1
3	-11792,8	-71935,3	191697,3	282848	681185,9
2	151841,1	53654,8	-2174503,9	430097,2	223402,7
4	386879,4	556792,5	-57015,8	191563,8	112637,4

4 Zeilen ausgewählt

11. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass eine Summenzeile, unterhalb der Pivottabelle angezeigt wird.



QUARTAL	'1985'	'1986'	'1987'	'1988'	'1989'
1	32204,8	985,2	2981,1	176852	9777,1
2	151841,1	53654,8	-2174503,9	430097,2	223402,7
3	-11792,8	-71935,3	191697,3	282848	681185,9
4	386879,4	556792,5	-57015,8	191563,8	112637,4
Summe	559132,5	539497,2	-2036841,3	1081361	1027003,1

5 Zeilen ausgewählt

10.8 Lösungen - Unterabfragen

1. Schreiben Sie eine Abfrage, die für alle Eigenkunden, die keinen Berater haben (die nicht in der Tabelle EIGENKUNDEMitarbeiter enthalten sind), den Vor- und den Nachnamen anzeigt.

- Lösen Sie die Aufgabe mit Hilfe des `EXISTS`-Operators!

- Lösen Sie die Aufgabe mit Hilfe des `IN`-Operators!

```

SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
WHERE NOT EXISTS (SELECT 1
                   FROM EigenkundeMitarbeiter ekm
                   WHERE ekm.Kunden_ID = ek.Kunden_ID);

SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
WHERE ek.Kunden_ID NOT IN (SELECT Kunden_ID
                            FROM EigenkundeMitarbeiter);

```



2. Erstellen Sie eine Abfrage, die ermittelt, ob es Mitarbeiter gibt (Vorname und Nachname), die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten) und Filialleiter.

- Lösen Sie die Aufgabe mit Hilfe des `EXISTS`-Operators!

- Lösen Sie die Aufgabe mit Hilfe des `IN`-Operators!

```

SELECT m.Vorname, m.Nachname
FROM Mitarbeiter m
INNER JOIN Mitarbeiter m1 ON (m.Vorgesetzter_ID = m1.Mitarbeiter_ID)
INNER JOIN Bankfiliale b ON (m1.Bankfiliale_ID = b.Bankfiliale_ID)
WHERE NOT EXISTS (SELECT 1
                   FROM EigenkundeMitarbeiter ekm
                   WHERE m.Mitarbeiter_ID = ekm.Mitarbeiter_ID)
ORDER BY m.Vorname;

SELECT m.Vorname, m.Nachname
FROM Mitarbeiter m
INNER JOIN Mitarbeiter m1 ON (m.Vorgesetzter_ID = m1.Mitarbeiter_ID)
INNER JOIN Bankfiliale b ON (m1.Bankfiliale_ID = b.Bankfiliale_ID)

```



```
WHERE m.Mitarbeiter_ID NOT IN (SELECT Mitarbeiter_ID  
                                FROM EigenkundeMitarbeiter ekm)  
ORDER BY m.Vorname;
```

3. Schreiben Sie eine Abfrage, die den häufigsten Vornamen der Bankmitarbeiter anzeigt und wie oft dieser in der Tabelle MITARBEITER vorkommt.

```
SELECT Vorname, Anzahl
FROM   (SELECT Vorname, COUNT(Vorname) AS Anzahl
        FROM Mitarbeiter
        GROUP BY Vorname
        ORDER BY COUNT(Vorname) DESC
       )
WHERE rownum = 1;
```



```
SELECT TOP 1 Vorname, COUNT(Vorname) AS Anzahl
FROM Mitarbeiter
GROUP BY Vorname
ORDER BY COUNT(Vorname) DESC;
```



4. Schreiben Sie eine Abfrage, welche die drei Eigenkunden mit den niedrigsten Guthaben auf den Girokonten anzeigt.

```
SELECT *
FROM   (SELECT Vorname, Nachname, Guthaben
        FROM Kunde k INNER JOIN EigenkundeKonto ekk
                      ON (k.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
        ORDER BY Guthaben)
WHERE rownum <= 3;
```



```
SELECT TOP 3 Vorname, Nachname, Guthaben
FROM Kunde k INNER JOIN EigenkundeKonto ekk
                  ON (k.Kunden_ID = ekk.Kunden_ID)
                  INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
                  ORDER BY Guthaben;
```



5. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass die drei Eigenkunden mit dem niedrigsten Guthaben (Girokonto + Sparbuch) angezeigt werden. Es müssen auch diejenigen Kunden angezeigt werden, die nur ein Girokonto oder nur ein Sparbuch haben!



```

SELECT *
FROM  (SELECT k.Vorname, k.Nachname, SUM(Guthaben)
       FROM  (SELECT Kunden_ID, Guthaben
              FROM EigenkundeKonto ekk INNER JOIN Girokonto g
                ON (ekk.Konto_ID = g.Konto_ID)
            UNION
            SELECT Kunden_ID, Guthaben
              FROM EigenkundeKonto ekk INNER JOIN Sparbuch s
                ON (ekk.Konto_ID = s.Konto_ID)) gut
            INNER JOIN Kunde k
              ON (k.Kunden_ID = gut.Kunden_ID)
        GROUP BY k.Kunden_ID, k.Vorname, k.Nachname
        ORDER BY 3)
WHERE rownum <= 3;

```



```

SELECT TOP 3 k.Vorname, k.Nachname, SUM(Guthaben)
FROM  (SELECT Kunden_ID, Guthaben
       FROM EigenkundeKonto ekk INNER JOIN Girokonto g
         ON (ekk.Konto_ID = g.Konto_ID)
      UNION
      SELECT Kunden_ID, Guthaben
        FROM EigenkundeKonto ekk INNER JOIN Sparbuch s
          ON (ekk.Konto_ID = s.Konto_ID)) gut
        INNER JOIN Kunde k ON (k.Kunden_ID = gut.Kunden_ID)
    GROUP BY k.Kunden_ID, k.Vorname, k.Nachname
    ORDER BY 3;

```

6. Schreiben Sie eine Abfrage, die alle Eigenkunden anzeigt, welche im Jahr 1985 keine Buchungen verursacht haben.



```

SELECT Vorname, Nachname
FROM Kunde k INNER JOIN EigenkundeKonto ekk
  ON (k.Kunden_ID = ekk.Kunden_ID)
WHERE NOT EXISTS (SELECT 1
                  FROM Buchung b INNER JOIN EigenkundeKonto ekk1
                    ON (b.Konto_ID = ekk1.Konto_ID)
                  WHERE ekk1.Kunden_ID = ekk.Kunden_ID
                    AND Buchungsdatum BETWEEN
                      TO_DATE('01.01.1985') AND
                      TO_DATE('31.12.1985'))
    GROUP BY k.Kunden_ID, Vorname, Nachname
    ORDER BY Nachname, Vorname;

```



```

SELECT DISTINCT Vorname, Nachname
FROM Kunde k INNER JOIN EigenkundeKonto ekk
    ON (k.Kunden_ID = ekk.Kunden_ID)
WHERE k.Kunden_ID NOT IN (SELECT ek.kunden_id
                           FROM Buchung b1 INNER JOIN EigenkundeKonto ek
                           ON (b1.Konto_ID = ek.Konto_ID)
                           WHERE b1.Buchungsdatum BETWEEN
                                 CONVERT(DATETIME2, '01.01.1985', 104) AND
                                 CONVERT(DATETIME2, '31.12.1985', 104))
ORDER BY Nachname, Vorname;

```

7. Schreiben Sie eine Abfrage, die für jede Bankfiliale den Mitarbeiter mit dem höchsten Gehalt ausgibt.



```

SELECT b.Strasse || ' ' || b.Hausnummer || ' ' || b.PLZ || ' ' ||
       b.Ort AS Bankfiliale,
       m.Vorname, m.Nachname, m.Gehalt
FROM Mitarbeiter m INNER JOIN Bankfiliale b
    ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
WHERE Gehalt = (SELECT MAX(Gehalt)
                FROM Mitarbeiter m1
                WHERE m.Bankfiliale_ID = m1.Bankfiliale_ID);

```



```

SELECT b.Strasse + ' ' + b.Hausnummer + ' ' + b.PLZ + ' ' +
       b.Ort AS Bankfiliale,
       m.Vorname, m.Nachname, m.Gehalt
FROM Mitarbeiter m INNER JOIN Bankfiliale b
    ON (m.Bankfiliale_ID = b.Bankfiliale_ID)
WHERE Gehalt = (SELECT MAX(Gehalt)
                FROM Mitarbeiter m1
                WHERE m.Bankfiliale_ID = m1.Bankfiliale_ID);

```

8. Schreiben Sie eine Abfrage, die für jeden Wohnort (EIGENKUNDE.ORT) den Kunden anzeigt, der im Jahr 1987 das höchste Einkommen hatte (Das Einkommen ist die Summe aller Beträge eines Kunden, in der Tabelle BUCHUNG). Sortieren Sie die Abfrage nach den Wohnorten.



```

SELECT b1.Ort, b1.Vorname, b1.Nachname, b1.betrag
FROM   (SELECT ek.Kunden_ID, ek.Ort, k.Vorname, k.Nachname,
               SUM(Betrag) AS betrag
        FROM EigenkundeKonto ekk INNER JOIN Buchung b
          ON (ekk.Konto_ID = b.Konto_ID)
        INNER JOIN Eigenkunde ek
          ON (ekk.Kunden_ID = ek.Kunden_ID)
        INNER JOIN Kunde k
          ON (k.Kunden_ID = ek.Kunden_ID)
      WHERE Buchungsdatum BETWEEN
            TO_DATE('01.01.1987', 'DD.MM.YYYY') AND
            TO_DATE('31.12.1987', 'DD.MM.YYYY')
      GROUP BY ek.Kunden_ID, ek.Ort, k.Vorname, k.Nachname) b1
WHERE betrag = (SELECT MAX(betrag)
                  FROM   (SELECT ek.Kunden_ID, ek.Ort, SUM(Betrag) AS betrag
                        FROM EigenkundeKonto ekk INNER JOIN Buchung b
                          ON (ekk.Konto_ID = b.Konto_ID)
                        INNER JOIN Eigenkunde ek
                          ON (ekk.Kunden_ID = ek.Kunden_ID)
                      WHERE Buchungsdatum BETWEEN
                            TO_DATE('01.01.1987', 'DD.MM.YYYY') AND
                            TO_DATE('31.12.1987', 'DD.MM.YYYY')
                      GROUP BY ek.Kunden_ID, ek.Ort) b2
                  WHERE b1.Ort = b2.Ort)
ORDER BY b1.Ort;

```



```

SELECT b1.Ort, b1.Vorname, b1.Nachname, b1.betrag
FROM   (SELECT ek.Kunden_ID, ek.Ort, k.Vorname, k.Nachname,
               SUM(Betrag) AS betrag
        FROM EigenkundeKonto ekk INNER JOIN Buchung b
          ON (ekk.Konto_ID = b.Konto_ID)
        INNER JOIN Eigenkunde ek
          ON (ekk.Kunden_ID = ek.Kunden_ID)
        INNER JOIN Kunde k
          ON (k.Kunden_ID = ek.Kunden_ID)
        WHERE Buchungsdatum BETWEEN
              CONVERT(DATETIME2, '01.01.1987', 104) AND
              CONVERT(DATETIME2, '31.12.1987', 104)
        GROUP BY ek.Kunden_ID, ek.Ort, k.Vorname, k.Nachname) b1
WHERE betrag = (SELECT MAX(betrag)
                 FROM   (SELECT ek.Kunden_ID, ek.Ort, SUM(Betrag) AS betrag
                         FROM EigenkundeKonto ekk INNER JOIN Buchung b
                           ON (ekk.Konto_ID = b.Konto_ID)
                         INNER JOIN Eigenkunde ek
                           ON (ekk.Kunden_ID = ek.Kunden_ID)
                         WHERE Buchungsdatum BETWEEN
                               CONVERT(DATETIME2, '01.01.1987', 104) AND
                               CONVERT(DATETIME2, '31.12.1987', 104)
                         GROUP BY ek.Kunden_ID, ek.Ort) b2
                 WHERE b1.Ort = b2.Ort)
ORDER BY b1.Ort;

```

9. Erstellen Sie eine Abfrage, die die Umsätze der Bank (SUM(Buchung.Betrag)) für die Jahre 1985 bis einschließlich 1989 als Pivottafelle anzeigt.



```

SELECT *
FROM   (SELECT TO_CHAR(Buchungsdatum, 'YYYY') AS Datum, Betrag
        FROM Buchung)
PIVOT (SUM(Betrag) FOR Datum IN ('1985', '1986', '1987', '1988', '1989'));

```



```

SELECT *
FROM   (SELECT DATEPART(YEAR, Buchungsdatum) AS Datum, Betrag
        FROM Buchung) AS Sourcetable
PIVOT (SUM(Betrag)
      FOR Datum IN ([1985], [1986], [1987], [1988], [1989])) AS Pivottable;

```

10. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass die Beträge innerhalb der einzelnen Jahre nach Quartalen aufgeteilt werden.



```
SELECT *
FROM   (SELECT TO_CHAR(Buchungsdatum, 'Q') AS Quartal,
              TO_CHAR(Buchungsdatum, 'YYYY') AS Datum, Betrag
        FROM Buchung)
PIVOT (SUM(Betrag) FOR Datum IN ('1985', '1986', '1987', '1988', '1989'));
```



```
SELECT *
FROM   (SELECT DATENAME(QUARTER, Buchungsdatum) AS Quartal,
              DATENAME(YEAR, Buchungsdatum) AS Datum, Betrag
        FROM Buchung) AS Sourcetable
PIVOT (SUM(Betrag)
       FOR Datum IN ([1985], [1986], [1987], [1988], [1989])) AS Pivottable;
```

11. Verändern Sie die Abfrage aus der vorangegangenen Aufgabe so, dass eine Summenzeile, unterhalb der Pivottabelle angezeigt wird.



```
SELECT *
FROM   (SELECT TO_CHAR(Buchungsdatum, 'Q') AS Quartal,
              TO_CHAR(Buchungsdatum, 'YYYY') AS Datum, Betrag
        FROM Buchung)
PIVOT (SUM(Betrag) FOR Datum IN ('1985', '1986', '1987', '1988', '1989'))
UNION ALL
SELECT *
FROM   (SELECT 'Summe' AS Quartal, TO_CHAR(Buchungsdatum, 'YYYY') AS Datum,
              Betrag
        FROM Buchung)
PIVOT (SUM(Betrag) FOR Datum IN ('1985', '1986', '1987', '1988', '1989'));
```



```
SELECT *
FROM   (SELECT DATENAME(QUARTER, Buchungsdatum) AS Quartal,
              DATENAME(YEAR, Buchungsdatum) AS Datum, Betrag
        FROM Buchung) AS Sourcetable
PIVOT (SUM(Betrag) FOR Datum IN ([1985], [1986], [1987], [1988], [1989]))
UNION ALL
SELECT *
FROM   (SELECT 'Summe' AS Quartal, DATENAME(YEAR, Buchungsdatum) AS Datum,
              Betrag
        FROM Buchung) AS Sourcetable
PIVOT (SUM(Betrag) FOR Datum IN ([1985], [1986], [1987], [1988], [1989])) AS Pivottable;
```

11 Data Manipulation Language (DML)

Inhaltsangabe



Im Gegensatz zu Oracle SQL wird bei MS SQL keine implizite Transaktion gestartet bei DML Statements. Dies muss bei MS SQL explizit mit SET IMPLICIT TRANSACTION eingeschaltet werden.

In den vergangenen Kapiteln wurde bisher nur der Teil von SQL beschrieben, der als sog. „Query language“ bezeichnet wird. Hier wird jetzt gezeigt, wie vorhandene Daten manipuliert werden können. Der dafür zuständige Teil von SQL heißt: „Data Manipulation Language“ oder kurz „DML“.

Gemäß SQL-Standard besteht DML aus drei Befehlen:

- **INSERT**: Daten einfügen.
- **UPDATE**: Daten ändern.
- **DELETE**: Daten löschen.

11.1 Die DML-Anweisungen

11.1.1 Datensätze einfügen - Die INSERT-Anweisung

Mit Hilfe der **INSERT**-Anweisung werden neue Datensätze an eine Tabelle angefügt. Die Syntax für ein einfaches **INSERT** lautet:

Listing 11.1: Die INSERT Anweisungen

```
INSERT INTO <Tabelle> (<Spalte 1>, <Spalte 2>, ..., <Spalte n>)
VALUES (<Wert 1>, <Wert 2>, ..., <Wert n>);
```

Tabelle 11.1: Die INSERT-Anweisung

Ausdruck	Bedeutung
INSERT INTO <Tabelle>	An dieser Stelle steht der Name der Tabelle oder View, in die der Datensatz eingefügt werden soll.
<Spalte 1>, <Spalte 2>, ...	Dies ist die Spaltenliste. Hier können alle Spalten angegeben werden, in die Daten eingefügt werden. Die Spaltenliste ist optional.
VALUES <Wert 1>, ...	Dies ist die Werteliste. Hier werden alle Werte aufgeführt, die in <Tabelle> eingefügt werden sollen. Statt einem festen Wert, kann an jeder Stelle auch ein Ausdruck stehen, der einen Wert erzeugt (z. B. eine Funktion).

Ausdruck Bedeutung

Beispiel ?? demonstriert die einfachste Form eines **INSERT**-Statements: Es wird eine neue Zeile in die Tabelle BANKFILIALE eingefügt.

Listing 11.2: Ein einfaches INSERT

```
INSERT INTO Bankfiliale (Bankfiliale_ID, Strasse, Hausnummer, PLZ, Ort)
VALUES (22, 'Rosenweg', '14a', '06425', 'Ploetzkau');
```

In obigem Beispiel wird der Wert „22“ in die Spalte BANKFILIALE_ID, der Wert „Rosenweg“ in die Spalte STRASSE eingefügt. Die restlichen drei Werte werden in die Spalten HAUSNUMMER, PLZ und ORT geschrieben. Die Spaltenliste der **INSERT**-Anweisung muss die einzelnen Spalten keineswegs in der Reihenfolge enthalten, wie sie in der Tabelle enthalten sind.

Listing 11.3: Ein einfaches INSERT

```
INSERT INTO Bankfiliale (Strasse, Hausnummer, PLZ, Ort, Bankfiliale_ID)
VALUES ('Rosenweg', '14a', '06425', 'Ploetzkau', 22);
```



In der Spaltenliste müssen die Spalten nicht in der Reihenfolge aufgeführt werden, wie sie in der Tabelle vorkommen. Die Reihenfolge in der Spaltenliste ist beliebig!

Wie in Tabelle ?? bereits beschrieben, ist die Spaltenliste hinter den **INSERT INTO**-Schlüsselwörtern optional. Daraus folgt, dass sich Beispiel ?? auch so schreiben lässt:

Listing 11.4: Ein einfaches INSERT ohne Spaltenliste

```
INSERT INTO Bankfiliale
VALUES (22, 'Rosenweg', '14a', '06425', 'Ploetzkau');
```

Das **INSERT**-Statement in Beispiel ?? wird vom DBMS so interpretiert, dass der erste Wert in die erste Spalte, der zweite Wert in die zweite Spalte, der dritte Wert in die dritte Spalte, usw. eingefügt wird.

Die **INSERT**-Anweisung und NULL-Werte

Soll mit einer **INSERT**-Anweisung ein NULL-Wert in eine Tabellenspalte eingefügt werden, geschieht dies mit Hilfe des Schlüsselwortes **NULL**. In Beispiel ?? wird eine neue Zeile in die Tabelle BANK eingefügt. Während des Einfügevorgangs ist der Wert für die Spalte RATING noch nicht bekannt. Die Zeile soll nun ohne diesen Wert eingefügt werden.

Listing 11.5: Ein einfaches **INSERT** mit **NULL**-Werten

```
INSERT INTO Bank
VALUES (21, 'KRDCU21SES', 'Lokki Bank of Cyprus', 'Steuerparadies', '42',
        '01067', 'Berlin', NULL);
```



Mit Hilfe des Schlüsselwortes **NULL** kann ein **NULL**-Wert in eine Tabellenspalte eingefügt werden.

Standardwerte

Standardwerte werden meist dann genutzt, wenn in eine Spalte häufig der gleiche Wert eingefügt werden muss. Sie müssen bei der Erstellung einer Tabelle mit definiert werden. Ein Beispiel hierfür könnte die Spalte BUCHUNGSDATUM der Tabelle BUCHUNG sein. Wird eine neue Buchung erfasst, muss immer das aktuelle Tagesdatum eingetragen werden. Diese kann durch die Funktion **SYSDATE** (Oracle) bzw. **GETDATE** (SQL Server) erzeugt werden.

Beispiel ?? zeigt, wie in die Spalte BUCHUNGSDATUM das aktuelle Datum, als Standardwert eingefügt wird.

Listing 11.6: Einfügen eines Standardwertes

```
INSERT INTO Buchung (Buchungs_ID, Betrag, Buchungsdatum, Konto_ID,
                     Transaktions_ID)
VALUES (500000, 300.20, DEFAULT, 1, 666666);
```



Soll in eine Tabellenspalte deren Standardwert eingefügt werden, muss das Schlüsselwort **DEFAULT** benutzt werden.

Die INSERT-Anweisung und Unterabfragen

Die **INSERT**-Anweisung ist in der Lage eine Unterabfrage zu verwenden, um den Inhalt einer Tabelle in eine andere Tabelle einzufügen. Dies kann z. B. das Kopieren eines Datensatzes in eine Tabelle gleicher Struktur sein oder das Abfragen einzelner Attribute, um diese für die Berechnung neuer Werte zu nutzen. Die Syntax für die **INSERT**-Anweisung mit Unterabfrage lautet:

Listing 11.7: Die **INSERT**-Anweisung mit Unterabfrage

```
INSERT INTO <Tabelle> (<Spalte 1>, <Spalte 2>, ..., <Spalte n>)
<Unterabfrage>;
```

Das **INSERT**-Statement kann eine beliebig komplexe Unterabfrage, wie in Abschnitt ?? beschrieben, verwenden. Beispiel ?? zeigt, wie ein Datensatz aus der Tabelle MITARBEITER in die strukturgleiche Tabelle AUSGESCHIEDEN kopiert wird.

Listing 11.8: Die **INSERT**-Anweisung mit Unterabfrage

```
-- Erstellen der Tabelle in Oracle
CREATE TABLE Ausgeschieden
AS
SELECT *
FROM   Mitarbeiter
WHERE  1 = 2;
```

```
-- Erstellen der Tabelle in SQL Server
SELECT *
INTO Ausgeschieden
FROM Mitarbeiter
WHERE 1 = 2;

INSERT INTO Ausgeschieden
SELECT *
FROM Mitarbeiter
WHERE Mitarbeiter_ID = 70;
```

Der Datensatz des Mitarbeiters Nummer 70 wird in die Tabelle AUSGESCHIEDEN kopiert.

11.1.2 Datensätze ändern - Die UPDATE-Anweisung

Die **UPDATE**-Anweisung repräsentiert den Teil von DML der es ermöglicht, bestehende Datensätze zu verändern. Die Syntax von **UPDATE** lautet:

Listing 11.9: Die Syntax des **UPDATE**-Kommandos

```
UPDATE <Tabelle>
SET    <Spalte 1> = <Wert>,
       <Spalte 2> = <Wert>,
       ...
       <Spalte n> = <Wert>
[WHERE <Where-Klausel>];
```

Tabelle 11.2: Die UPDATE-Anweisung

Ausdruck	Bedeutung
UPDATE <Tabelle>	An dieser Stelle steht der Name der Tabelle oder View, in der ein Datensatz verändert werden soll.
SET <Spalte 1> = <Wert>	Die SET-Anweisung gibt die Spalten an, deren aktueller Wert durch den neuen Wert <Wert> ersetzt werden soll. Hier können mehrere „Spalte = Wert“-Paare, durch Komma getrennt, stehen.
WHERE <Where-Klausel>	Optionale WHERE-Klausel, die den Umfang der Datensätze, die geändert werden sollen, einschränkt.

Eine genauso einfache, wie auch gefährliche Form der **UPDATE**-Anweisung, ist in Beispiel ?? zu sehen.

Listing 11.10: Ein gefährliches **UPDATE**

```
UPDATE Mitarbeiter
SET    Gehalt = Gehalt * 1.035;
```

Die Gefahr bei dieser **UPDATE**-Anweisung besteht darin, dass die Angabe einer einschränkenden **WHERE**-Klausel fehlt. Das DBMS wird in diesem Falle alle Datensätze der Tabelle **MITARBEITER** verändern und nicht nur eine bestimmte Gruppe.

Soll nur das Gehalt des Mitarbeiters *Max Winter* geändert werden, muss das **UPDATE**-Statement um eine **WHERE**-Klausel erweitert werden:

Listing 11.11: Ein korrektes **UPDATE**

```
UPDATE Mitarbeiter
SET    Gehalt = Gehalt * 1.035
WHERE  Mitarbeiter_ID = 1;
```

Wie in Tabelle ?? zu sehen ist, können auch mehrere Spalten eines Datensatzes gleichzeitig geändert werden. In Beispiel ?? wird die Mitarbeiterin „*Lena Hermann*“ (**Mitarbeiter_ID** 40) von Filiale 4 nach Filiale 8 versetzt und gleichzeitig wird ihre Provision von 20 % auf 30 % erhöht.

Listing 11.12: Ein korrektes **UPDATE** mehrerer Spalten

```
UPDATE Mitarbeiter
SET    Bankfiliale_ID = 8,
        Provision = 30
WHERE  Mitarbeiter_ID = 40;
```

Wo Licht ist, da ist aber immer auch Schatten. Wenn bei einem Mitarbeiter die Provision erhöht wird, muss sie bei einem anderen gekürzt oder gestrichen werden. Der Mitarbeiter „*Lukas Weiß*“ hat im vergangenen Geschäftsjahr ein sehr schlechtes Ergebnis erzielt, weshalb ihm die Provision gestrichen wird. Dies geschieht, indem die Spalte **PROVISION** mit einem **NULL**-Wert gefüllt wird.

Listing 11.13: Da geht sie hin, die Provision

```
UPDATE Mitarbeiter
SET    Provision = NULL
WHERE  Mitarbeiter_ID = 38;
```

Nicht nur **NULL**-Werte, auch Standardwerte können innerhalb eines **UPDATE**-Statements genutzt werden.

Listing 11.14: Ein **UPDATE** mit Standardwert

```
UPDATE Mitarbeiter
SET    Gehalt = DEFAULT
WHERE  Mitarbeiter_ID = 82;
```

Beispiel ?? geht davon aus, dass für die Spalte **GEHALT** ein Standardwert von „1500“ festgelegt worden ist.

UPDATE mit Unterabfrage

Wie bei der `INSERT`-Anweisung, kann auch bei der `UPDATE`-Anweisung eine Unterabfrage genutzt werden. Diese kann an zwei Stellen stehen: In der `SET`-Klausel und in der `WHERE`-Klausel. Hierzu einige Beispiele.

Das Gehalt des Mitarbeiters „Jannis Friedrich“ soll geändert werden. Das neue Gehalt muss 20 % des Gehalts seines unmittelbaren Vorgesetzten betragen.

Listing 11.15: `UPDATE` mit Unterabfrage

```
UPDATE Mitarbeiter
SET Gehalt = (SELECT v.Gehalt
               FROM Mitarbeiter m INNER JOIN Mitarbeiter v
                     ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
               WHERE m.Mitarbeiter_ID = 79) * 0.20
WHERE Mitarbeiter_ID = 79;
```

Mit Hilfe der folgenden `SELECT`-Anweisung kann die Korrektheit des `UPDATE`-Statements aus Beispiel ?? nachgewiesen werden.

Listing 11.16: Der Beweis

```
SELECT Mitarbeiter_ID, Vorname, Nachname, Gehalt
FROM Mitarbeiter
WHERE Mitarbeiter_ID IN (79, 21);
```

In Beispiel ?? wird die `Mitarbeiter_ID` 79 an zwei Stellen angegeben. Durch eine Veränderung des `UPDATE`-Statements kann dies auf eine Angabe reduziert werden.

Listing 11.17: `UPDATE` mit korrelierter Unterabfrage in Oracle

```
UPDATE Mitarbeiter m1
SET Gehalt = (SELECT v.Gehalt
               FROM Mitarbeiter m INNER JOIN Mitarbeiter v
                     ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
               WHERE m.Mitarbeiter_ID = m1.Mitarbeiter_ID)
WHERE m1.Mitarbeiter_ID = 79;
```

In der `UPDATE`-Klausel wird ein Alias für die Tabelle `MITARBEITER` festgelegt. Diesen Alias benutzt die Unterabfrage, um auf die Werte des äußeren Statements, des `UPDATE`-Statements, zuzugreifen. Dadurch genügt es, wenn die `Mitarbeiter_ID` nur einmal gesetzt wird. Im MS SQL Server muss der Alias für die Tabelle `MITARBEITER` über eine `FROM`-Klausel definiert werden, so dass sich Beispiel ?? wie folgt ändert:

Listing 11.18: **UPDATE** mit korrelierter Unterabfrage im MS SQL Server

```
UPDATE m1
SET Gehalt = (SELECT v.Gehalt
               FROM Mitarbeiter m INNER JOIN Mitarbeiter v
                 ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
               WHERE m.Mitarbeiter_ID = m1.Mitarbeiter_ID)
FROM Mitarbeiter m1
WHERE m1.Mitarbeiter_ID = 79;
```

„Was des einen Freud ist, ist des andern Leid“. Dieser Grundsatz trifft auch bei der Gehaltserhöhung für Herrn Friedrich zu. Da er nun 400 EUR mehr Gehalt bekommt, müssen bei den anderen Angestellten dementsprechende Einsparungen vorgenommen werden. Für alle Mitarbeiter der Filiale 14, mit Ausnahme von Herrn Friedrich, muss das Gehalt um 2 % gekürzt werden.

Listing 11.19: Gehaltskürzung für eine ganze Filiale

```
UPDATE Mitarbeiter
SET Gehalt = Gehalt * 0.98
WHERE Mitarbeiter_ID IN (SELECT Mitarbeiter_ID
                           FROM Mitarbeiter
                           WHERE Bankfiliale_ID = 14
                           AND Mitarbeiter_ID <> 79);
```

11.1.3 Datensätze löschen - Die **DELETE**-Anweisung

Die dritte und letzte der DML-Anweisungen, ist die **DELETE**-Anweisung. Sie ermöglicht es, Datensätze zu löschen. Die Syntax der **DELETE**-Anweisung lautet wie folgt:

Listing 11.20: Die **DELETE**-Anweisung

```
DELETE FROM <Tabelle>
WHERE <Where-Klausel>;
```

Tabelle 11.3: Die **DELETE**-Anweisung

Ausdruck	Bedeutung
DELETE <Tabelle>	An dieser Stelle steht der Name der Tabelle oder View, aus der Datensätze gelöscht werden sollen.
WHERE <Where-Klausel>	Optionale WHERE-Klausel, die den Umfang der Datensätze begrenzt, die gelöscht werden sollen.

Ähnlich wie bei der **UPDATE**-Anweisung, gibt es auch bei der **DELETE**-Anweisung eine kleine Falle. Beispiel ?? zeigt, wie man mit einer sehr einfachen **DELETE**-Anweisung in große Schwierigkeiten geraten kann.

Listing 11.21: Eine tödliche **DELETE**-Anweisung

```
DELETE FROM Buchung;
```

Die Auswirkungen der **DELETE**-Anweisung aus Beispiel ?? sind einfach und kurz erklärt. Es werden alle Datensätze aus der Tabelle BUCHUNG gelöscht. Des Rätsels Lösung ist die gleiche wie beim **UPDATE**-Statement: Es fehlt die einschränkende **WHERE**-Klausel. Um beispielsweise nur eine einzelne Buchung zu löschen ist folgende Modifikation notwendig:

Listing 11.22: Schon viel besser!!!

```
DELETE FROM Buchung  
WHERE Transaktions_ID = 345;
```

DELETE mit Unterabfrage

Auch in der **DELETE**-Anweisung kann eine Unterabfrage genutzt werden. Hierzu ein einfaches Beispiel: Da die Bankfiliale, in der Poststraße, in Aschersleben aufgelöst wird, müssen leider auch die dort beschäftigten Mitarbeiter wieder dem Arbeitsmarkt zur Verfügung gestellt werden.

Listing 11.23: **DELETE** mit Unterabfrage

```
DELETE FROM Mitarbeiter  
WHERE Bankfiliale_ID = (SELECT Bankfiliale_ID  
                      FROM Bankfiliale  
                      WHERE LOWER(Strasse) LIKE 'poststrasse',  
                           AND PLZ = '06449');
```

11.2 Das Transaktionskonzept - COMMIT und ROLLBACK

Die Datenbankmanagementsysteme Oracle und SQL Server sind beides transaktionsbasierte DBMS. Das bedeutet, dass alle DML-Anweisungen innerhalb einer Transaktion ablaufen. Die Frage die sich dabei stellt ist: „Was ist eine Transaktion?“ Der Begriff Transaktion ist dem spätlateinischen „transagere = Überführen, Übertragen“ entliehen und den meisten Leuten aus dem Finanzbereich bekannt. Man denke einfach an die Überweisung eines Betrags von Konto A auf Konto B. Der vereinfachte Ablauf einer solchen Finanztransaktion könnte wie folgt aussehen:

1. Kontoinhaber A füllt einen Überweisungsträger aus. Damit beginnt die Transaktion.
2. Die Bank des Kontoinhabers A zieht den Überweisungsbetrag von seinem Konto ab und übermittelt die Informationen bezüglich der Überweisung an Bank B.

3. Bank B schreibt den Betrag auf dem Konto von Kontoinhaber B gut.
4. Der Vorgang wird in einem Journal protokolliert. Damit ist die Überweisung abgeschlossen.

Warum aber der Begriff der Transaktion? Die Antwort auf diese Frage hängt eng mit der Antwort auf eine andere Frage zusammen: „Was wäre wenn, nach der Abbuchung von Konto A der Vorgang unterbrochen würde?“ In so einem Falle ist das gewohnte Verhalten, dass alle bisher gemachten Schritte wieder rückgängig gemacht werden, d. h. der abgebuchte Betrag muss wieder auf das Konto von A zurückgebucht werden. Würde dies nicht geschehen, wäre das Geld von A verschwunden.

Das Rückgängigmachen aller bisher gemachten Aktionen ist aber nur dann möglich, wenn

- genau bekannt ist, welche Aktionen zusammengehören und
- in welcher Reihenfolge sie stattgefunden haben.

Deshalb werden alle Aktionen in einer größeren Einheit, der Transaktion, zusammengefaßt. Es muss also im Ernstfall nur ermittelt werden, zu welcher Transaktion die letzte Aktion gehörte um alle Vorgängeraktionen ermitteln zu können.



Definition Transaktion: Eine Transaktion ist eine logische Arbeitseinheit, die einen oder mehrere Arbeitsschritte enthält. Transaktionen sind in sich geschlossene Einheiten. Die Ergebnisse aller Arbeitsschritte einer Transaktion können entweder übernommen oder rückgängig gemacht werden.

Dieses Konzept lässt sich auch auf Datenbanken übertragen. Werden mehrere zusammengehörende SQL-Anweisungen ausgeführt, muss auch gewährleistet werden, dass entweder alle erfolgreich beendet werden oder aber alle rückgängig gemacht werden.

11.2.1 Beginn und Ende einer Transaktion

Wann beginnt eine Transaktion?

In Oracle startet eine Transaktion:

- Implizit bei jedem ersten DML-Statement.
- Explizit durch die Anweisung `SET TRANSACTION`.

In MS SQL Server startet eine Transaktion:

- Wenn der implizite Transaktionsmodus aktiviert wurde, bei jedem ersten DML-Statement.
- Explizit durch die Anweisung `BEGIN TRANSACTION`



Das Standardverhalten in SQL Server ist, dass jedes einzelne DML-Statement als eigene Transaktion abgehandelt wird. Zur Aktivierung des impliziten Transaktionsmodus muss die SQL-Anweisung `SET IMPLICIT_TRANSACTIONS ON` abgesetzt werden.

Wann endet eine Transaktion?

Eine Transaktion kann an zwei verschiedenen Punkten enden:

- Sie wird erfolgreich abgeschlossen.
- Sie wird manuell rückgängig gemacht.

11.2.2 Eine Transaktion erfolgreich abschließen

Das COMMIT-Kommando

Wenn alle Statements einer Transaktion erfolgreich verlaufen sind, muss die Transaktion beendet werden, um die gemachten Änderungen dauerhaft in der Datenbank zu speichern. Dies geschieht in Oracle mit Hilfe des Kommandos `COMMIT`. Wird eine Transaktion nicht mit `COMMIT` abgeschlossen, werden automatisch alle unbestätigten Änderungen rückgängig gemacht. Beispiel ?? ff. zeigen dieses Verhalten.

Listing 11.24: Eine Transaktion wird abgebrochen

```
SELECT Bank_ID , BIC , Name
FROM   Bank
WHERE  Bank_ID >= 18;
```



BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank

3 Zeilen ausgewählt

```
INSERT INTO Bank
VALUES      (21, 'NOSDEL21SES', 'Lokki Bank of Cyprus',
              'Strasse der Europaeischen Union', '3', '00000', 'Pleitingen',
              'D--');

SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;
```



BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank
21	NOSDEL21SES	Lokki Bank of Cyprus

4 Zeilen ausgewählt

```
-- An dieser Stelle findet ein Absturz der Client-Anwendung statt
-- und die Anwendung wird neu gestartet.
```

```
SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;
```



BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank

3 Zeilen ausgewählt

Weil vor dem Absturz der Client-Anwendung die Transaktion nicht mit **COMMIT** abgeschlossen wurde, ist die gemachte Änderung wieder verschwunden. Das gleiche Szenario nun noch einmal, aber mit **COMMIT** am Ende.

```
INSERT INTO Bank
VALUES      (21, 'NOSDEL21SES', 'Lokki Bank of Cyprus',
              'Strasse der Europaeischen Union', '3', '00000', 'Pleitingen',
              'D--');

SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;

COMMIT;
```



BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank
21	NOSDEL21SES	Lokki Bank of Cyprus

4 Zeilen ausgewählt

```
-- An dieser Stelle wird die Client-Anwendung beendet und
-- neu gestartet.
```

```
SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;
```

BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank
21	NOSDEL21SES	Lokki Bank of Cyprus

4 Zeilen ausgewählt



Die **COMMIT**-Anweisung persistiert^a die Aktionen einer Transaktion in der Datenbank. Ohne **COMMIT** werden alle Änderungen wieder zurückgerollt.

^apersistent = dauerhaft

COMMIT in Microsoft SQL Server

In Microsoft SQL Server muss dem **COMMIT**-Kommando noch das Schlüsselwort **TRANSACTION** (oder **TRAN**) hinzugefügt werden. Dies beendet sowohl implizite als auch explizite Transaktionen.

Listing 11.25: Eine implizite Transaktion committen

```
SET IMPLICIT_TRANSACTIONS ON
INSERT INTO Bank
VALUES      (21, 'NOSDEL21SES', 'Lokki Bank of Cyprus',
             'Strasse der Europaeischen Union', '3', '00000', 'Pleitingen',
             'D--');

SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;

COMMIT TRAN;
```

Listing 11.26: Eine explizite Transaktion committen

```
BEGIN TRANSACTION
INSERT INTO Bank
VALUES      (21, 'NOSDEL21SES', 'Lokki Bank of Cyprus',
              'Strasse der Europaeischen Union', '3', '00000', 'Pleitingen',
              'D--');

SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;
COMMIT TRAN;
```

11.2.3 Eine Transaktion rückgängig machen

Das ROLLBACK-Kommando

Das Kommando `ROLLBACK` stellt das Gegenstück zu `COMMIT` dar. Sollen die Aktionen einer Transaktion nicht dauerhaft in der Datenbank gespeichert werden, können sie mit `ROLLBACK` zurückgerollt (rückgängig gemacht) werden.

Listing 11.27: Eine Transaktion wird abgebrochen

```
SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;
```



BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIOODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank
21	NOSDEL21SES	Lokki Bank of Cyprus

4 Zeilen ausgewählt

```
DELETE FROM Bank
WHERE Bank_ID = 21;

SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;
```



BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIOODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank

3 Zeilen ausgewählt

```
ROLLBACK;

SELECT Bank_ID, BIC, Name
FROM   Bank
WHERE  Bank_ID >= 18;
```



BANK_ID	BIC	NAME
18	BVXYDE21SES	Bank der Landwirte
19	BGIOODE21SES	Austrailian Bank Association
20	DFGHDE21SES	South Africa Bank
21	NOSDEL21SES	Lokki Bank of Cyprus

4 Zeilen ausgewählt



Die Anweisung **ROLLBACK** rollt alle Aktionen einer Transaktion zurück und beendet sie.

ROLLBACK in Microsoft SQL Server

Genau wie das **COMMIT**-Kommando, muss auch das **ROLLBACK**-Kommando um das Schlüsselwort **TRANSACTION** ergänzt werden.

12 Data Definition Language

Inhaltsangabe

Die Data definition language ist der Teil von SQL, der es ermöglicht, Objekte in der Datenbank zu erstellen und zu verwalten. DDL besteht im wesentlichen aus den vier Befehlen:

- **CREATE:** Erstellen von Objekten
- **ALTER:** Ändern von Objekten
- **DROP:** Objekte löschen
- **TRUNCATE:** Leeren von Tabellen.

Der Begriff des „Objekts“ bezieht sich, je nach DBMS, auf die Unterschiedlichsten Dinge:

- Tabellen
- Views
- Indizes
- Sequenzen
- PL/SQL oder T-SQL Prozeduren und Funktionen

... und vieles mehr. Welche Möglichkeiten dem Anwender bei der Erstellung eines Objekts geboten werden, ist stark abhängig vom jeweiligen DBMS.

12.1 Tabellen erstellen und verwalten

12.1.1 Namenskonventionen und Einschränkungen

Bevor näher auf die Namenskonventionen für Objekte eingegangen wird, müssen an dieser Stelle zuerst einige Fachbegriffe geklärt werden.

- **Bezeichner:** Namen für Objekte (Tabellen, Spalten, Views, usw.) heißen im Fachjargon Bezeichner.
- **Umschlossene Bezeichner:** Sind Bezeichner, die in Anführungszeichen " eingeschlossen sind

- **Reservierte Wörter:** Begriffe die in SQL eine bestimmte Bedeutung haben, z. B. `SELECT`, `WHERE`, usw.
- **Namensraum:** Logische Einteilung für Objektnamen. Bezeichner müssen innerhalb eines Namensraumes eindeutig sein.

Tabelle ?? listet die wichtigsten Einschränkungen auf, die für Bezeichner in beiden DBMS gelten.

Tabelle 12.1: Einschränkungen für Bezeichner



Bezeichnerlänge	30	128
Reservierte Wörter	Bezeichner können keine reservierten Wörter sein, es sei denn, sie sind in Anführungszeichen " eingeschlossen.	Bezeichner können keine reservierten Wörter sein, es sei denn, sie sind in Anführungszeichen " eingeschlossen.
Namensgebung	Wenn Bezeichner nicht in Anführungszeichen (" einschlossen sind, müssen diese mit einem Buchstaben beginnen. Für umschlossene Bezeichner gilt dies nicht.	Wenn Bezeichner nicht in Anführungszeichen (" oder ([] einschlossen sind, müssen diese mit einem Buchstaben, _, @ oder # beginnen. Für umschlossene Bezeichner gilt dies nicht.
Gültige Zeichen	Nicht umschlossene Bezeichner können nur aus den Buchstaben a-z und A-Z, den Ziffern 0-9, sowie _, \$ und # bestehen. Für umschlossene Bezeichner gilt, dass dort alle Zeichen, auch Leerzeichen vorkommen können.	Nicht umschlossene Bezeichner können nur aus den Buchstaben a-z und A-Z, den Ziffern 0-9, sowie @, \$, _ und # bestehen. Für umschlossene Bezeichner gilt, dass dort alle Zeichen, auch Leerzeichen vorkommen können.
Namensgleichheit	Zwei Datenbankobjekte im gleichen Namensraum müssen unterschiedliche Namen haben.	Bezeichner müssen innerhalb eines Schemas eindeutig sein.
Casesensitivität	Nicht umschlossene Bezeichner sind nicht Casesensitiv. Bezeichner die mit (" oder ([] umschlossen sind, sind Casesensitiv.	Bezeichner die mit (" oder ([] umschlossen sind, sind nicht Casesensitiv.

Damit umschlossene Bezeichner in SQL Server 2008 R2 genutzt werden können, muss die Option `QUOTED_IDENTIFIER` den Wert `ON` haben. Dieser kann nötigenfalls mit `SET QUOTED_IDENTIFIER ON` gesetzt werden.

Die folgenden Internetliteraturhinweise liefern weitere Informationen.



- [i27561]
- [ms187879]

12.1.2 CREATE TABLE - Tabellen erstellen

Sowohl in Oracle als auch in SQL Server werden Tabellen mit Hilfe des Kommandos `CREATE TABLE` erstellt. Die grundlegende, SQL-Standardkonforme Syntax für `CREATE TABLE` lautet:

Listing 12.1: Die Syntax der CREATE TABLE-Anweisung

```
CREATE TABLE <tabellen_name> (
    <spaltenbezeichner 1> <datentyp>,
    <spaltenbezeichner 2> <datentyp>,
    ...,
    <spaltenbezeichner n> <datentyp>
);
```

Tabelle 12.2: Die CREATE TABLE-Anweisung

Ausdruck	Bedeutung
<code>CREATE TABLE <Tabellenname></code>	Diese Klausel leitet das Erstellen der Tabelle ein. Für den Tabellennamen gelten die in Tabelle ?? angegebenen Beschränkungen.
<code><Spaltenbezeichner> <Datentyp></code>	Jede Tabellenspalte wird durch einen Bezeichner/Name und einen Datentyp repräsentiert. Mit Hilfe des Namens kann die Spalte später angesprochen werden und der Datentyp legt den Wertebereich der Spalte fest. Je nach DBMS gelten auch hier unterschiedliche Einschränkungen.

Beispiel ?? zeigt ein einfaches `CREATE TABLE`-Statement.

Listing 12.2: Eine einfache CREATE TABLE-Anweisung in Oracle

```
CREATE TABLE Aktie (
    Aktie_ID NUMBER,
    Name      VARCHAR2(25),
    WKN       NUMBER,
    ISIN      VARCHAR2(12)
);
```

Listing 12.3: Das gleiche in MS SQL Server

```
CREATE TABLE Aktie (
    Aktie_ID NUMERIC,
    Name      VARCHAR(25),
    WKN       NUMERIC,
    ISIN      VARCHAR(12)
);
```

Es wird eine Tabelle namens AKTIE, mit den Spalten AKTIE_ID, NAME, WKN und ISIN angelegt.

Zur besseren Umsetzung der Beispiele in den folgenden Abschnitten, werden nun einige Datensätze in die Tabelle AKTIE eingefügt.

Listing 12.4: Beispieldatensätze

```
INSERT INTO Aktie
VALUES (1, 'Henker Co KG', 1236547, 'DE0006800002');

INSERT INTO Aktie
VALUES (2, 'AD and D', 43116589, 'DE0002300023');

COMMIT;
```

12.1.3 CREATE TABLE AS... (CTAS)

Die Abkürzung „CTAS“ steht für `CREATE TABLE AS` und meint ein `CREATE TABLE` mit Unterabfrage. Mit Hilfe von CTAS können bestehende Tabellen teilweise oder ganz kopiert werden. Beispiel ?? zeigt, wie in Oracle eine vollständige Kopie der Tabelle AKTIE angefertigt wird.

Listing 12.5: Oracle - CREATE TABLE AS (CTAS)

```
CREATE TABLE Aktie_Kopie
AS
SELECT *
FROM Aktie;
```

Es wird eine Tabelle namens AKTIE_KOPIE erstellt. Diese erhält die komplette Struktur und den gesamten Inhalt der Tabelle AKTIE.

In Microsoft SQL Server kennt das `CREATE TABLE`-Statement keine Möglichkeit, eine Unterabfrage zu nutzen. Hier muss stattdessen das `SELECT INTO`-Statement genutzt werden.

Listing 12.6: MS SQL Server - SELECT INTO

```
SELECT *
INTO Aktie_Kopie
FROM Aktie;
```

Die Auswirkungen bleiben die gleichen, wie unter Oracle mit CTAS.



Microsoft SQL Server kennt das `CREATE TABLE AS`-Statement nicht. Es muss stattdessen das `SELECT INTO`-Statement genutzt werden. Die Auswirkungen von `CREATE TABLE AS` und `SELECT INTO` sind gleich.

12.1.4 ALTER TABLE - Tabellen verändern

Mit Hilfe der **ALTER TABLE**-Anweisung können bestehende Tabellendefinition verändert werden. Dies betrifft z. B.:

- Das Hinzufügen neuer Spalten zu einer Tabelle.
- Das Löschen von Spalten.
- Das Umbenennen von Spalten.
- Das Ändern des Datentyps einer Spalte.
- Ändern der Größe einer Spalte.
- Das Hinzufügen, ändern und löschen eines Standardwerts.
- Das Hinzufügen und Löschen von Constraints (siehe Abschnitt ??)

Eine neue Spalte an eine Tabelle anfügen

In beiden DBMS gibt es, zum Hinzufügen einer Spalte zu einer Tabelle, die ADD-Klausel des **ALTER TABLE**-Kommandos. In Beispiel ??, wird der Tabelle AKTIE eine neue Spalte namens HERKUNFT hinzugefügt.

Listing 12.7: Oracle - Tabellenspalte hinzufügen

```
ALTER TABLE Aktie
ADD Herkunft VARCHAR2(25);
```

In SQL Server unterscheidet sich dieses Statement nur durch den Datentyp.

Listing 12.8: MS SQL Server - Tabellenspalte hinzufügen

```
ALTER TABLE Aktie
ADD Herkunft VARCHAR(25);
```



Wird eine neue Spalte an eine Tabelle angefügt, haben alle Zellen dieser Spalte den Wert NULL, es sei den, es wird ein Standardwert für diese Spalte definiert. In diesem Falle füllt Oracle die Spalte mit dem Standardwert auf. SQL Server tut dies nicht.

Beispiel ?? und Beispiel ?? zeigen, wie sich Oracle und MS SQL Server verhalten, wenn eine neue Spalte, mit einem Standardwert, hinzugefügt wird. Die Spalte HERKUNFT wird mit dem Standardwert „Deutschland“ an die Tabelle AKTIE angefügt. In Oracle werden dann automatisch alle bereits vorhandenen Zeilen mit dem neuen Standardwert aufgefüllt. In SQL Server wird dies nicht der Fall sein.

Listing 12.9: Tabellenspalte mit Standardwert hinzufügen in Oracle

```
ALTER TABLE Aktie
ADD Herkunft VARCHAR2(25) DEFAULT 'Deutschland';

SELECT *
FROM Aktie;
```



AKTIE_ID	NAME	WKN	ISIN	HERKUNFT
1	Henker Co KG	1236547	DE0006800002	Deutschland
2	AD and D	43116589	DE0002300023	Deutschland

2 Zeilen ausgewählt

Wird das gleiche Experiment in MS SQL Server durchgeführt, zeigt sich das die Spalte HERKUNFT nicht automatisch aufgefüllt wird.

Listing 12.10: Tabellenspalte mit Standardwert hinzufügen in SQL Server

```
ALTER TABLE Aktie
ADD DEFAULT 'Deutschland' for Herkunft;

SELECT *
FROM Aktie;
```



AKTIE_ID	NAME	WKN	ISIN	HERKUNFT
1	Henker Co KG	1236547	DE0006800002	NULL
2	AD and D	43116589	DE0002300023	NULL

2 Zeilen ausgewählt

Spalten vergrößern und verkleinern

Es besteht die Möglichkeit, die Definition einer Spalte nachträglich zu verändern. Dabei können verschiedene Dinge, wie z. B. der Spaltendatentyp oder der Standardwert einer Spalte geändert werden. Um eine solche Änderung durchzuführen, kennt das `ALTER TABLE`-Kommando unter Oracle die `MODIFY`-Klausel und unter SQL Server die `ALTER COLUMN`-Klausel. Hierzu einige Beispiele.

In Beispiel ?? wird die Breite der Spalte HERKUNFT in der Tabelle AKTIE verändert.

Listing 12.11: Anpassen der Spaltenlänge in Oracle

```
ALTER TABLE Aktie
MODIFY Herkunft VARCHAR2(30);
```

Eine Vergrößerung stellt prinzipiell niemals ein Problem dar. Schwieriger wird es hingegen, wenn eine Spalte verkleinert werden muss. In Oracle geht das nur dann, wenn die Inhalte der Spalte kleiner sind als die neue Spaltengröße. Andernfalls antwortet Oracle mit der in Beispiel ?? sichtbaren Fehlermeldung:

Listing 12.12: Fehlermeldung beim verkleinern einer Spalte in Oracle

```
MODIFY Herkunft VARCHAR2(15)
*
FEHLER in Zeile 2:
ORA-01441: Spaltenlaenge kann nicht vermindert werden, weil ein Wert zu gross
ist
```



Eine Tabellenspalte kann in Oracle nur auf die Größe des größten darin enthaltenden Werts verkleinert werden. In SQL Server kann eine Tabellenspalte auch mit Inhalt verkleinert werden.

Bei SQL Server muss lediglich die `MODIFY`-Klausel durch die `ALTER COLUMN`-Klausel ersetzt werden.

Listing 12.13: Anpassen der Spaltenlänge in SQL Server

```
ALTER TABLE Aktie
ALTER COLUMN Herkunft VARCHAR(30);
```

Ändern des Datentyps

Mit Hilfe der `MODIFY`-Klausel kann nicht nur die Größe einer Spalte verändert werden, sondern auch der Datentyp. In Beispiel ?? wird der Datentyp der Spalte WKN von `NUMBER` auf `VARCHAR2`, bzw. von `NUMERIC` auf `VARCHAR` geändert.

Listing 12.14: Ändern des Datentyps

```
ALTER TABLE Aktie
MODIFY WKN VARCHAR2(10);
```

In SQL Server sieht das Ändern des Datentyps einer Spalte sehr ähnlich aus.

Listing 12.15: Ändern des Datentyps

```
ALTER TABLE Aktie
ALTER COLUMN WKN VARCHAR(10);
```



Eine Tabellenspalte muss in Oracle leer sein, damit ihr Datentyp verändert werden kann. In SQL Server kann der Datentyp einer Spalte auch mit Inhalt verändert werden.

Einen Defaultvalue hinzufügen

Eine weitere Aktion die mit `MODIFY` bzw. `ALTER COLUMN` möglich ist, ist das Hinzufügen, ändern oder entfernen eines Standardwertes bei einer Tabellenspalte. In Beispiel ?? wird in Oracle der Standardwert der Spalte HERKUNFT von „Deutschland“ auf „USA“ geändert.

Listing 12.16: Einen Standardwert ändern

```
ALTER TABLE Aktie
MODIFY Herkunft DEFAULT 'USA';
```

Mit der gleichen Anweisung kann der Standardwert eine Spalte, unter Oracle, nicht nur geändert sondern auch hinzugefügt werden. Das Löschen des Standardwertes geschieht, indem NULL als Standardwert zugewiesen wird.

Listing 12.17: Standardwert hinzufügen

```
ALTER TABLE Aktie
MODIFY Herkunft DEFAULT NULL;
```

Bei SQL Server ist das Löschen eines Standardwerts anders als bei Oracle. In SQL Server wird ein Standardwert als sogenanntes Constraint¹ gehandhabt. Deshalb wird diese Aktion zu einem späteren Zeitpunkt in Abschnitt ?? behandelt.



Wird in Oracle mit Hilfe `ADD`-Klausel eine Spalte mit Standardwert hinzugefügt, werden alle NULL-Werte in der gleichen Spalte mit dem Standardwert aufgefüllt. Wird die Spalte dagegen mit der `MODIFY`-Klausel, nachträglich mit einem Default-Wert ausgestattet, bleiben alle NULL-Werte erhalten!

Tabellenspalten umbenennen

Es ist möglich, bestehende Spalten umzubenennen. Dafür wird in Oracle die `RENAME COLUMN`-Klausel des `ALTER TABLE`-Kommandos verwendet. In Microsoft SQL Server gibt es hierfür eine gespeicherte Hilfsprozedur, welche das Umbenennen übernimmt. Der neue Spaltenname muss innerhalb der Tabelle eindeutig sein und es dürfen keine anderen Operationen zusammen mit dem Umbenennen geschehen.

Listing 12.18: Tabellenspalte umbenennen in Oracle

```
ALTER TABLE Aktie
RENAME COLUMN Name TO Bezeichnung;
```

¹constraint engl. = Einschränkung

Listing 12.19: Tabellenspalte umbenennen in SQL Server

```
EXEC sp_rename 'Aktie.Name', 'Bezeichnung', 'COLUMN'
```



Für das Umbenennen von Objekten ist in SQL Server die gespeicherte Hilfsprozedur `sp_rename` zuständig.

Zu beachten ist, dass das Umbenennen einer Spalte Auswirkungen auf abhängige Objekte wie z. B. Views oder Trigger haben kann und deshalb mit größter Vorsicht durchzuführen ist.

Tabellenspalten löschen

Tabellenspalten, die nicht mehr benötigt werden, können jeder Zeit gelöscht werden. Auf diese einfache Art und Weise kann Speicherplatz zu weiteren Nutzung freigegeben werden. Allgemein gilt als Einschränkung beim Löschen einer Tabellenspalte:

- Die letzte Spalte in einer Tabelle kann nicht gelöscht werden. Es muss dann die gesamte Tabelle gelöscht werden.

Für Oracle gilt zusätzlich:

- Ein normaler Nutzer kann keine Spalten aus einer Tabelle löschen, die dem Nutzer sys gehört.

Beispiel ?? zeigt das Löschen einer Tabellenspalte in Oracle und SQL Server.

Listing 12.20: Tabellenspalte löschen

```
ALTER TABLE Aktie
DROP COLUMN WKN;
```

12.1.5 DROP TABLE - Tabellen löschen

Eine nicht mehr benötigte Tabelle, wird in Oracle und SQL Server mit dem `DROP TABLE`-Kommando gelöscht.

- Alle verknüpften Indizes und Trigger werden mitgelöscht.
- Alle abhängigen Views bleiben bestehen und werden ungültig.

Das folgende Beispiel löscht die Tabelle AKTIE.

Listing 12.21: Eine Tabelle löschen

```
DROP TABLE Aktie;
```

12.1.6 TRUNCATE TABLE - Tabellen leeren

Eine Tabelle kann mit `TRUNCATE TABLE` geleert werden. Die Tabelle selbst bleibt dabei erhalten. Um eine Tabelle zu leeren, gibt es drei Möglichkeiten:

- Das DML-Statement `DELETE`
- Das Löschen der Tabelle mit `DROP TABLE` und neu erstellen mit `CREATE TABLE`
- Das DDL-Statement `TRUNCATE`

Den Tabelleninhalt mit `DELETE` löschen

Es können alle Zeilen einer Tabelle mit dem DML-Kommando `DELETE` gelöscht werden.

Listing 12.22: Zeilen mit `DELETE` löschen

```
DELETE FROM Aktie;
```

Bei einer großen Tabelle werden hierfür sehr viele Systemressourcen benötigt (CPU, RAM, usw.). Des Weiteren kann es passieren, dass beim Löschen von Zeilen, Trigger ausgelöst werden.



Der Speicherplatz, der durch die Tabelle vor dem Löschen belegt wurde, bleibt bei der Verwendung von `DELETE` belegt.

Einziger Vorteil ist, dass mit der `DELETE`-Klausel die Zeilen ausgewählt werden können, die gelöscht werden sollen.

Die Tabelle löschen und neu erstellen

Eine Tabelle kann gelöscht und mit `CREATE TABLE` neu erstellt werden. Dabei gehen alle mit dieser Tabelle verbundenen Indizes, Integritäts Constraints und Trigger verloren und alle von der Tabelle abhängigen Objekte werden ungültig.

Eine Tabelle mit TRUNCATE leeren

Um alle Zeilen einer Tabelle zu löschen kann das `TRUNCATE`-Statement verwendet werden.

Listing 12.23: Zeilen mit TRUNCATE abschneiden

```
TRUNCATE TABLE Aktie;
```



In Oracle produziert das `TRUNCATE`-Statement, wie alle DDL-Statements, automatisch ein `COMMIT`, d. h. es kann nicht rückgängig gemacht werden. In SQL Server ist das Zurückrollen eines `TRUNCATE`-Statements möglich.



Der Speicherplatz, der durch die Tabelle vor dem Löschen belegt wurde, wird bei der Verwendung von `TRUNCATE`, freigegeben.

12.2 Views erstellen verwalten

12.2.1 Was sind Views?

Bei der täglichen Arbeit mit einer Datenbank treten häufig immer wiederkehrende `SELECT`-Statements auf. Dies kann z. B. deshalb sein, weil ein Nutzer immer wieder die gleiche Sicht (gleiche Spalten, gleiche Filterbedingung) auf die Daten einer Tabelle benötigt.



Eine View ist eine genau definierte Sicht auf eine bestimmte Datenmenge.

12.2.2 Views erstellen

Views werden mit dem `CREATE VIEW`-Kommando erstellt. Die Syntax für `CREATE VIEW` sieht wie folgt aus:

Listing 12.24: Die Syntax von CREATE VIEW

```
CREATE VIEW <View_name>
(<Spalten_alias 1, Spalten_alias 2, ..., Spalten_alias n>
AS
<Auswahlabfrage>;
```

Tabelle 12.3: Die CREATE VIEW-Anweisung

Ausdruck	Bedeutung
<code>CREATE VIEW <View_name></code>	Diese Klausel leitet das Erstellen der View ein. Für den Viewname gelten die in Tabelle ?? angegebenen Beschränkungen.

Ausdruck	Bedeutung
<Spalten_alias>	Für jeden Spaltenbezeichner, der in der Auswahlabfrage genutzt wird, kann an dieser Stelle ein Aliasname festgelegt werden.
<Auswahlabfrage>	Dies ist das SELECT-Statement.

Ein einfaches Beispiel für das Erstellen einer View ist in Beispiel ?? zu sehen.

Listing 12.25: Eine einfache View

```
CREATE VIEW v_Kunde
AS
SELECT Vorname, Nachname
FROM Kunde;
```

Was an dieser Stelle passiert ist, dass das DBMS das **SELECT**-Statement verarbeitet und unter dem Namen **v_KUNDEN** abspeichert. Anschließend kann, wie in Beispiel ??, mit SQL auf die View zugegriffen werden.

Listing 12.26: Zugriff auf eine View

```
SELECT Vorname, Nachname
FROM v_Kunde;
```



VORNAME	NACHNAME
Niklas	Schneider
Mia	Keller
Lilli	Beck
Emilia	Keller
Finn	Junge
Marie	Vogel
Rudi	Roggatz
Leni	Koch
Chris	Zimmermann
Justin	Gabriel
Sebastian	Schröder
561 Zeilen ausgewählt	

Auch wenn in der **SELECT**-Klausel der Auswahlabfrage der * verwendet wird, wird im Hintergrund folgendes Statement, als View gespeichert:

Listing 12.27: Was tatsächlich gespeichert wird

```
-- So wird die View erstellt
CREATE VIEW v_Kunde
AS
```

```
SELECT *
FROM Kunde;

-- Das wird gespeichert
SELECT Kunden_ID, Vorname, Nachname
FROM Kunde;
```

Diese Tatsache ist nicht ganz unwichtig, wie folgendes Szenario beweist:

Listing 12.28: Eine Szenario mit Tücke

```
CREATE VIEW v_Aktie
AS
SELECT *
FROM Aktie;

ALTER TABLE Aktie
ADD Herkunft VARCHAR2(30);

SELECT *
FROM v_Aktie;
```



AKTIE_ID	NAME	WKN	ISIN
1	Henker Co KG	1236547	DE0006800002
2	AD and D	43116589	DE0002300023

2 Zeilen ausgewählt

Da bei der Erstellung der View v_AKTIE, der * in die einzelnen Spalten der Tabelle AKTIE aufgelöst wurde, ist die neu hinzugefügte Spalte HERKUNFT, in der View v_AKTIE noch nicht zu sehen. Hierzu müsste die Viewdefinition geändert bzw. die View neu erstellt werden.



Wird in der Auswahlabfrage einer View das * Symbol verwendet, wird dieses interpretiert. D. h. es wird ersetzt durch die tatsächliche Spaltenliste der Quelltabelle. Änderungen an der Struktur der Tabelle werden somit von der View nicht erkannt.

Wie in Tabelle ?? bereits erklärt, kann bei der Erstellung einer View auch eine Liste mit Spaltenaliasnamen angegeben werden. Dies ist in den folgenden Fällen immer notwendig:

- Wenn in der View ein berechneter Ausdruck vorhanden ist
- Wenn in der View mehrere Tabellen mit einem Join verbunden sind und Spalten mit gleichem Namen ausgegeben werden müssen.

Beispiel ?? zeigt eine View mit Spaltenaliasliste.

Listing 12.29: Eine einfache View mit Spaltenaliasliste

```

CREATE VIEW v_Kunde
(Vorname , Nachname , Lebensalter)
AS
SELECT k.Vorname , k.Nachname ,
       ROUND(MONTHS_BETWEEN(SYSDATE , Geburtsdatum) / 12 , 0)
FROM   Kunde k INNER JOIN Eigenkunde ek
      ON (k.Kunden_ID = ek.Kunden_ID);

SELECT *
FROM   v_Kunde;

```



VORNAME	NACHNAME	LEBENSALTER
Mia	Keller	41
Emilia	Keller	23
Finn	Junge	37
Marie	Vogel	42
Rudi	Roggatz	26
Leni	Koch	38
Chris	Zimmermann	23
Sebastian	Schröder	24
Justin	Zimmermann	34
Petra	Krause	34
Clara	Rollert	23
Gustav	Witte	23
400 Zeilen ausgewählt		

Wie gut zu erkennen ist, ersetzen die Spaltenaliase die tatsächlichen Spaltennamen in v_KUNDE. Die gleiche Auswirkung wäre auch mit dem folgenden Statement zu erreichen:

Listing 12.30: Eine einfache View mit Spaltenaliasen

```

CREATE VIEW v_Kunde
AS
SELECT k.Vorname , k.Nachname ,
       ROUND(MONTHS_BETWEEN(SYSDATE , Geburtsdatum) / 12 , 0) AS Lebensalter
FROM   Kunde k INNER JOIN Eigenkunde ek
      ON (k.Kunden_ID = ek.Kunden_ID);

```



Wird eine Spaltenaliasliste genutzt, muss diese genauso viele Aliasnamen umfassen, wie die SELECT-Liste der Auswahlabfrage Spaltennamen zurückgibt.

Hierzu ein kleines Beispiel. Im folgenden `CREATE VIEW`-Statement werden zu wenige Spaltenalias angegeben. Oracle und auch SQL Server antworten prompt mit einer Fehlermeldung.

Listing 12.31: Eine einfache View mit fehlerhafter Spaltenaliasliste in Oracle

```
CREATE VIEW v_Kunde
(Vorname, Nachname)
AS
SELECT k.Vorname, k.Nachname,
       ROUND(MONTHS_BETWEEN(SYSDATE, Geburtsdatum) / 12, 0)
FROM   Kunde k INNER JOIN Eigenkunde ek
      ON (k.Kunden_ID = ek.Kunden_ID);

(Vorname, Nachname)
*
FEHLER in Zeile 2:
ORA-01730: invalid number of column names specified
```

SQL Server antwortet wie folgt:

Listing 12.32: Eine einfache View mit fehlerhafter Spaltenaliasliste in SQL Server

```
CREATE VIEW v_Kunde
(Vorname, Nachname)
AS
SELECT k.Vorname, k.Nachname, DATEDIFF(YEAR, getDate(), Geburtsdatum)
FROM   Kunde k INNER JOIN Eigenkunde ek
      ON (k.Kunden_ID = ek.Kunden_ID);

Meldung 8158, Ebene 16, Status 1, Prozedur v_Kunde, Zeile 4
'v_Kunde' besitzt mehr Spalten, als in der Spaltenliste angegeben sind.
```

Bereits weiter oben in diesem Abschnitt wurde erläutert, dass eine View, in der ein berechneter Ausdruck vorkommt, zwingend mit Spaltenaliasen versehen werden muss. Beispiel ?? beweist dies:

Listing 12.33: Eine View mit einer berechneten Spalte in Oracle

```
CREATE VIEW v_Kunde
(Vorname, Nachname)
AS
SELECT k.Vorname, k.Nachname,
       ROUND(MONTHS_BETWEEN(SYSDATE, Geburtsdatum) / 12, 0)
FROM   Kunde k INNER JOIN Eigenkunde ek
      ON (k.Kunden_ID = ek.Kunden_ID);

SELECT k.Vorname, k.Nachname,
       ROUND(MONTHS_BETWEEN(SYSDATE, Geburtsdatum) / 12, 0)
*
FEHLER in Zeile 3:
ORA-00998: Dieser Ausdruck braucht einen Spalten-Alias
```

Auch SQL Server hat hiermit Probleme:

Listing 12.34: Eine View mit einer berechneten Spalte in SQL Server

```
CREATE VIEW v_Kunde (Vorname, Nachname)
AS
SELECT k.Vorname, k.Nachname, DATEDIFF(YEAR, getDate(), Geburtsdatum)
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID);

Meldung 4511, Ebene 16, Status 1, Prozedur v_Kunde, Zeile 3
Fehler beim Ausführen von CREATE VIEW oder CREATE FUNCTION, da
für die 3-Spalte kein Spaltenname angegeben wurde.
```

Dieses Problem kann durch eine Spaltenaliasliste oder durch die direkte Vergabe eines Spaltenalias gelöst werden.



Bei SQL Server gibt es noch die Einschränkung, dass die Auswahlabfrage einer View keine `ORDER BY`-Klausel enthalten darf.

12.2.3 Views und DML

Views können auch für die Ausführung von DML-Statements verwendet werden. Dabei gibt es jedoch einige Einschränkungen und Regeln die zu beachten sind.

Tabelle 12.4: Regeln für DML-Operationen auf Views

Einschränkung	INSERT	UPDATE	DELETE
Es dürfen keine Aggregatfunktionen (COUNT, SUM, MAX, MIN, AVG) in der View genutzt werden.	X	X	X
Die View darf keine GROUP BY-Klausel enthalten.	X	X	X
Die View darf das DISTINCT-Schlüsselwort nicht benutzen.	X	X	X
Die View darf keine berechneten Ausdrücke aufweisen.	X	X	
Die View darf keine Pseudospalten enthalten	X		X
Die View darf keinen Join enthalten.	X	X	X
Alle mit NOT NULL markierten Spalten der Basistabelle müssen im INSERT-Statement berücksichtigt werden.	X	X	
Der Einfüge- bzw. Änderungsvorgang muss, falls eine CHECK-Option in der View vorhanden ist (siehe Abschnitt ??), den Vorgaben der WHERE-Klausel der Abfrage genügen.	X	X	

Die View darf keine READ ONLY-Option (siehe Abschnitt ??) enthalten.	X	X			
--	---	---	--	--	--

Die Einschränkung WITH CHECK OPTION

Bei der Erstellung einer View kann eine zusätzliche Einschränkung mit angegeben werden, die **CHECK OPTION**. Diese schränkt den Nutzer dahingehend ein, dass nur noch solche Datensätze geändert werden können, die auch in der View zu sehen sind.

Listing 12.35: Ein Experiment mit den CHECK OPTION

```
CREATE VIEW v_Mitarbeiter
AS
SELECT *
FROM Mitarbeiter
WHERE Bankfiliale_ID = 5;

INSERT INTO v_Mitarbeiter
VALUES (666, 'Florian', 'Weidinger', 12, 8,
        TO_DATE('01.03.1988', 'DD.MM.YYYY'),
        '38B546C1-CDF-36A7B97', 1500, 'Abendrot Gase',
        '13', '39444', 'Hecklingen', 20);

1 row inserted

ROLLBACK;
```

Obwohl die **WHERE**-Klausel der View V_MITARBEITER die Anzeige auf die Bankfiliale mit der ID fünf einschränkt, kann trotzdem ein Datensatz in die Bankfiliale Nummer acht eingefügt werden.

Um die DML-Möglichkeiten der View V_MITARBEITER einzuschränken, wird im nächsten Beispiel die **CHECK**-Option angewendet.

Listing 12.36: Ein Experiment mit der CHECK OPTION in Oracle

```
CREATE VIEW v_Mitarbeiter
AS
SELECT *
FROM Mitarbeiter
WHERE Bankfiliale_ID = 5
WITH CHECK OPTION;

INSERT INTO v_Mitarbeiter
VALUES (666, 'Florian', 'Weidinger', 12, 8,
        TO_DATE('01.03.1988', 'DD.MM.YYYY'),
        '38B546C1-CDF-36A7B97', 1500, 'Abendrot Gase',
        '13', '39444', 'Hecklingen', 20);
```

```

INSERT INTO v_Mitarbeiter
*
FEHLER in Zeile 1:
ORA-01402: Verletzung der where-Klausel einer View with check option

```

Da jetzt die **CHECK**-Option genutzt wurde, reagiert das DBMS mit einer Fehlermeldung auf DML-Statements, die sich auf **v_Mitarbeiter** beziehen und nicht der **WHERE**-Klausel der View entsprechen.

Listing 12.37: Ein Experiment mit der CHECK OPTION in SQL Server

```

CREATE VIEW v_Mitarbeiter
AS
SELECT *
FROM Mitarbeiter
WHERE Bankfiliale_ID = 5
WITH CHECK OPTION;

INSERT INTO v_Mitarbeiter
VALUES (666, 'Florian', 'Weidinger', 12, 8,
        CONVERT(DATETIME2, '01.03.1988', 104),
        '38B546C1-CDF-36A7B97', 1500, 'Abendrot Gase',
        '13', '39444', 'Hecklingen', 20);

Meldung 550, Ebene 16, Status 1, Zeile 1
Fehler beim Einfügen oder Aktualisieren, da die Zielsicht WITH CHECK OPTION
angibt oder sich auf eine Sicht erstreckt, die WITH CHECK OPTION angibt,
und mindestens eine Ergebnissezeile nicht der CHECK OPTION-Einschränkung
entsprach.

```

Die Einschränkung WITH READ ONLY - Oracle

Die **READ ONLY**-Option für Views ermöglicht es, einem Nutzer den Schreibzugriff auf eine View zu verbieten. Die View kann somit nur noch lesend genutzt werden.

Listing 12.38: Eine View mit mit READ ONLY Option erstellen

```

CREATE VIEW v_Mitarbeiter
AS
SELECT *
FROM Mitarbeiter
WITH READ ONLY;

```

Versucht ein Nutzer trotzdem mit einem DML-Statement auf die View zuzugreifen, wird er mit einer Fehlermeldung abgewiesen.

Listing 12.39: Daten in eine READ ONLY View einfügen schlägt fehl

```

INSERT INTO v_Mitarbeiter
VALUES (666, 'Florian', 'Weidinger', 12, 8,
        TO_DATE('01.03.1988', 'DD.MM.YYYY'),
        '38B546C1-CDF-36A7B97', 1500, 'Abendrot Gase',
        '13', '39444', 'Hecklingen', 20);

INSERT INTO v_Mitarbeiter
*
FEHLER in Zeile 1:
ORA-42399: cannot perform a DML operation on a read-only view

```

Um diese Option wieder von der View zu nehmen, muss die View neu erstellt werden (siehe Abschnitt ??)

12.2.4 Views ändern

Müssen an einer View Veränderungen vorgenommen werden, bedeutet dies immer, dass die View neu erstellt werden muss. Oracle und SQL Server kennen hierzu unterschiedliche Wege:

- In Oracle wird die `CREATE VIEW`-Klausel erweitert: `CREATE OR REPLACE VIEW`.
- SQL Server benutzt hierfür die `ALTER VIEW`-Anweisung.

Die beiden Beispiele Beispiel ?? und Beispiel ?? zeigen, wie in Oracle und SQL Server eine Viewdefinition geändert werden kann.

Listing 12.40: Eine View ändern in Oracle

```

-- Zuerst wird die View erstellt
CREATE VIEW v_Mitarbeiter
AS
  SELECT *
  FROM   Mitarbeiter
WITH READ ONLY;

-- Dann wird sie geändert
CREATE OR REPLACE VIEW v_Mitarbeiter
AS
  SELECT *
  FROM   Mitarbeiter;

```

Und nun SQL Server.

Listing 12.41: Eine View ändern in SQL Server

```
-- Zuerst wird die View erstellt
CREATE VIEW v_Mitarbeiter
AS
    SELECT *
    FROM    Mitarbeiter;

-- Dann wird sie geändert
ALTER VIEW v_Mitarbeiter
AS
    SELECT *
    FROM    Mitarbeiter
    WHERE   Bankfiliale_ID = 5;
```

12.2.5 Views löschen

Zum Löschen von Views gibt es das Kommando `DROP VIEW`.

Listing 12.42: Eine View löschen

```
DROP VIEW viw_countries;
```

12.3 Übungen - Erstellen von Views

1. Erstellen Sie die View v_ARBEITSORT. Diese muss für jeden Mitarbeiter den Vornamen, den Nachnamen, die Bankfiliale_ID und den Ort anzeigen, an dem sich die Filiale befindet.



VORNAME	NACHNAME	BANKFILIALE_ID	ORT
Marie	Kipp	1	Aschersleben
Louis	Schmitz	1	Aschersleben
Johannes	Lehmann	1	Aschersleben
Dirk	Peters	1	Aschersleben
Amelie	Krüger	1	Aschersleben
93 Zeilen ausgewählt			

2. Erstellen Sie die View v_DEPOTBESITZER, die zu jedem Eigenkunden, der ein Depot besitzt, seinen Vor- und Nachnamen, die Strasse mit der Hausnummer, sowie PLZ und Ort anzeigt.



VORNAME	NACHNAME	STRASSE	PLZ	ORT
Sophie	Junge	Plutoweg 3	39435	Bördeaeue
Hanna	Beck	Beimsstraße 9	39439	Güsten
Sebastian	Peters	Steinigstraße 3	39240	Staßfurt
Tina	Berger	Bundschuhstraße 1	04177	Leipzig
239 Zeilen ausgewählt				

3. Erstellen Sie die View v_FINANZBERATER, die für alle Eigenkunden deren Vor- und Nachnamen anzeigt, sowie den Vor- und den Nachnamen ihres persönlichen Finanzberaters (Tabelle EIGENKUNDEMitarbeiter).



Vorname	Kunde	Nachname	Kunde	Vorname	Berater	Nachname	Berater
Mia		Keller		Lena		Herrmann	
Emilia		Keller		Louis		Wagner	
Finn		Junge		Leni		Friedrich	
Marie		Vogel		Finn		Wolf	
Rudi		Roggatz		Frank		Meierhöfer	
Leni		Koch		Frank		Hartmann	
Chris		Zimmermann		Clara		Walther	
Justin		Zimmermann		Leni		Friedrich	
Petra	Krause	Chris	Hartmann				
Clara	Rollert	Franz	Berger				

400 Zeilen ausgewählt

4. Erstellen Sie die View v_UNTERSTELLUNGSVERHAELTNIS, die zu jedem Mitarbeiter (Vorname, Nachname) den Vor- und den Nachnamen seines Vorgesetzten anzeigt. Wichtig ist, dass alle Mitarbeiter, auch Herr Max Winter, der keinen Vorgesetzten hat, angezeigt werden.



VORNAME	NACHNAME	VORNAME	NACHNAME
Finn	Seifert	Max	Winter
Sarah	Werner	Max	Winter
Tim	Sindermann	Sarah	Werner
Sebastian	Schwarz	Sarah	Werner
Emily	Meier	Finn	Seifert
Peter	Möller	Finn	Seifert

100 Zeilen ausgewählt

5. Erstellen Sie die View v_INNENDIENSTMitarbeiter, die ermittelt, ob es Mitarbeiter gibt (Vorname und Nachname), die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten) und Filialleiter.



VORNAME	NACHNAME
Amelie	Krüger
Anna	Schneider
Chris	Simon
Christian	Haas
Elias	Sindermann

40 Zeilen ausgewählt

6. Erstellen Sie die View v_GIROKONTOINHABER, die alle Eigenkunden anzeigt, die nur ein Girokonto besitzen.



VORNAME	NACHNAME
Amelie	Becker
Amelie	Richter
Chris	Walther
Emilia	Keller
Georg	Keller
Johanna	Schäfer
Justin	Zimmermann

21 Zeilen ausgewählt

12.4 Lösungen - Erstellen von Views

1. Erstellen Sie die View v_ARBEITSORT. Diese muss für jeden Mitarbeiter den Vornamen, den Nachnamen, die Bankfiliale_ID und den Ort anzeigen, an dem sich die Filiale befindet.

```

CREATE VIEW v_Arbeitsort
AS
SELECT m.Vorname, m.Nachname, m.Bankfiliale_ID, b.Ort
FROM Mitarbeiter m INNER JOIN Bankfiliale b
ON (b.Bankfiliale_ID = m.Bankfiliale_ID);
```

2. Erstellen Sie die View v_DEPOTBESITZER, die zu jedem Eigenkunden, der ein Depot besitzt, seinen Vor- und Nachnamen, die Strasse mit der Hausnummer, sowie PLZ und Ort anzeigt.

```

CREATE VIEW v_Depotbesitzer
AS
SELECT k.Vorname, k.Nachname, ek.Strasse || ' ' || 
ek.Hausnummer AS Strasse,
ek.PLZ, ek.Ort
FROM Kunde k INNER JOIN Eigenkunde ek
ON (k.Kunden_ID = ek.Kunden_ID)
INNER JOIN EigenkundeKonto ekk
ON (ek.Kunden_ID = ekk.Kunden_ID)
INNER JOIN Depot d
ON (ekk.Konto_ID = d.Konto_ID);
```

```

CREATE VIEW v_Depotbesitzer
AS
SELECT k.Vorname, k.Nachname, ek.Strasse + ' ' + ek.Hausnummer AS Strasse,
ek.PLZ, ek.Ort
FROM Kunde k INNER JOIN Eigenkunde ek
ON (k.Kunden_ID = ek.Kunden_ID)
INNER JOIN EigenkundeKonto ekk
ON (ek.Kunden_ID = ekk.Kunden_ID)
INNER JOIN Depot d
ON (ekk.Konto_ID = d.Konto_ID);
```

3. Erstellen Sie die View v_FINANZBERATER, die für alle Eigenkunden deren Vor- und Nachnamen anzeigt, sowie den Vor- und den Nachnamen ihres persönlichen Finanzberaters (Tabelle EIGENKUNDEMitarbeiter).



```
CREATE VIEW v_Finanzberater
("Vorname Kunde", "Nachname Kunde", "Vorname Berater", "Nachname Berater")
AS
  SELECT      k.Vorname , k.Nachname , m.Vorname , m.Nachname
  FROM        Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
              LEFT OUTER JOIN EigenkundeMitarbeiter ekm
                ON (ek.Kunden_ID = ekm.Kunden_ID)
              LEFT OUTER JOIN Mitarbeiter m
                ON (ekm.Mitarbeiter_ID = m.Mitarbeiter_ID);
```

4. Erstellen Sie die View v_UNTERSTELLUNGSVERHAELTNIS, die zu jedem Mitarbeiter (Vorname, Nachname) den Vor- und den Nachnamen seines Vorgesetzten anzeigt. Wichtig ist, dass alle Mitarbeiter, auch Herr Max Winter, der keinen Vorgesetzten hat, angezeigt werden.



```
CREATE VIEW v_Unterstellungsverhaeltnis
AS
  SELECT      m.Vorname , m.Nachname , v.Vorname , v.Nachname
  FROM        Mitarbeiter m LEFT OUTER JOIN Mitarbeiter v
                ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID)
```

5. Erstellen Sie die View v_INNENDIENSTMitarbeiter, die ermittelt, ob es Mitarbeiter gibt (Vorname und Nachname), die keine Kundenberatung durchführen. Ausgenommen sind leitende Mitarbeiter (Mitarbeiter die in keiner Bankfiliale arbeiten) und Filialleiter.



```
CREATE VIEW v_Innendienstmitarbeiter
AS
  SELECT      m.Vorname , m.Nachname
  FROM        Mitarbeiter m LEFT OUTER JOIN EigenkundeMitarbeiter ekm
                ON (m.Mitarbeiter_ID = ekm.Mitarbeiter_ID)
  WHERE      ekm.Mitarbeiter_ID IS NULL
  MINUS
  SELECT      DISTINCT v.Vorname , v.Nachname
  FROM        Mitarbeiter m INNER JOIN Mitarbeiter v
                ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID);
```



```

CREATE VIEW v_Innendienstmitarbeiter
AS
SELECT m.Vorname, m.Nachname
FROM Mitarbeiter m LEFT OUTER JOIN EigenkundeMitarbeiter ekm
    ON (m.Mitarbeiter_ID = ekm.Mitarbeiter_ID)
WHERE ekm.Mitarbeiter_ID IS NULL
EXCEPT
SELECT DISTINCT v.Vorname, v.Nachname
FROM Mitarbeiter m INNER JOIN Mitarbeiter v
    ON (m.Vorgesetzter_ID = v.Mitarbeiter_ID);

```

6. Erstellen Sie die View v_GIROKONTOINHABER, die alle Eigenkunden anzeigt, die nur ein Girokonto besitzen.



```

CREATE VIEW v_Girokontoinhaber
AS
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
MINUS
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Sparbuch s ON (ekk.Konto_ID = s.Konto_ID)
MINUS
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Depot d ON (ekk.Konto_ID = d.Konto_ID);

```



```

CREATE VIEW v_Girokontoinhaber
AS
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Girokonto g ON (ekk.Konto_ID = g.Konto_ID)
EXCEPT
SELECT k.Vorname, k.Nachname
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
    INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
        INNER JOIN Sparbuch s ON (ekk.Konto_ID = s.Konto_ID)
EXCEPT
SELECT k.Vorname, k.Nachname

```

```
FROM Kunde k INNER JOIN Eigenkunde ek ON (k.Kunden_ID = ek.Kunden_ID)
INNER JOIN EigenkundeKonto ekk ON (ek.Kunden_ID = ekk.Kunden_ID)
INNER JOIN Depot d ON (ekk.Konto_ID = d.Konto_ID);
```

13 Constraints

Inhaltsangabe

13.1 Was sind Constraints

Der englische Begriff „Constraint“ bedeutet übersetzt soviel wie: „Einschränkung“ oder „Zwang“. Constraints werden in Datenbankmanagementsystemen verwendet, um genau definierte Richtlinien für die Erfassung und die Verwaltung der Daten zu schaffen. Sie sorgen z. B. dafür, dass manche Spalten immer zwingend einen Wert ungleich NULL haben müssen oder das sie nur eindeutige Werte aufnehmen können. Es ist auch möglich einen genauen Wertebereich für eine Spalte zu definieren oder Werte aus Spalten anderer Tabellen zu referenzieren. Oracle und MS SQL Server kennen fünf Constraints für das relationale Datenmodell:

- **CHECK:** Definiert einen exakten Wertebereich für eine Spalte.
- **NOT NULL:** Definiert eine Spalte so, dass sie zwingend immer einen Wert ungleich NULL enthalten muss.
- **UNIQUE:** Legt fest, dass die Werte einer Spalte oder einer Kombination von Spalten eindeutig sein müssen.
- **PRIMARY KEY:** Hat die Aufgabe, ein eindeutiges Identifikationsmerkmal für jede Zeile einer Tabelle darzustellen. Er ist eine Kombination aus dem **NOT NULL**- und dem **UNIQUE**-Constraint und kann sich ebenfalls auf eine Kombination von Spalten beziehen.
- **FOREIGN KEY:** Referenziert eine Spalte einer anderen Tabelle, die mit einem **UNIQUE**- oder **PRIMARY KEY**-Constraint versehen sein muss und stellt somit die referentielle Integrität (siehe Abschnitt ??) der Datenbank sicher.

Zusätzlich zu diesen fünf kennt Oracle noch das „REF“-Constraint, das jedoch nur im Rahmen der objektorientierten Anteile von Oracle Bedeutung hat und hier keine weitere Erwähnung findet. SQL Server kennt zusätzlich noch ein weiteres Constraint: das **DEFAULT**-Constraint.

13.2 Die Constraints

Constraints können mit Hilfe der beiden Kommandos **CREATE TABLE** und **ALTER TABLE** angelegt werden. Sie werden durch einen Bezeichner und ihren Typ repräsentiert. Die Bezeichner von Constraints unterliegen ebenfalls den in Tabelle ?? beschriebenen Regeln.



Wird für ein Constraint kein Name festgelegt, legt Oracle automatisch einen Namen nach dem Schema „SYS_Cn“ fest, wobei n eine sechstellige Zufallszahl darstellt z. B. SYS_C168349. SQL Server verwendet ein Namensschema mit dem Aufbau „typ_tabelle_spalte_n“ wobei n eine eindeutige hexadezimal Nummer darstellt, z. B. PK_mitarbeiter_mitarbeiter_id_4B561A78.

In einem `CREATE TABLE`-Kommando können Constraints als „Inline Constraint“ und als „Out Of Line Constraint“ angelegt werden.

Listing 13.1: Constraints erstellen

```
CREATE TABLE <Tabellenname>(
    <Spalte 1> <Datentyp> CONSTRAINT <Inline Constraint Name> <Constraint Typ>,
    <Spalte 2> <Datentyp> CONSTRAINT <Inline Constraint Name> <Constraint Typ>,
    ...
    <Spalte n> <Datentyp> CONSTRAINT <Inline Constraint Name> <Constraint Typ>,
    CONSTRAINT <Out Of Line Constraint Name> <Constraint Typ> <Spalte 1, Spalte n>
    CONSTRAINT <Out Of Line Constraint Name> <Constraint Typ> <Spalte>
);
```



Wird ein Constraint direkt mit der Definition einer Spalte angelegt, wird es als Inline Constraint bezeichnet und bezieht sich auf die Spalte mit der es definiert wurde. Wird ein Constraint im Anschluss an die Spaltendefinitionen angelegt, wird es als Out Of Line Constraint bezeichnet und kann sich auf mehrere Spalten beziehen.

13.2.1 Das CHECK-Constraint

Das `CHECK`-Constraint hat die Aufgabe einen genauen Wertebereich für eine Spalte festzulegen. Beispielsweise wäre ein `CHECK`-Constraint auf der Spalte GEHALT der Tabelle MITARBEITER sinnvoll, das definiert, dass Gehälter niemals negativ und niemals über 90000 EUR sein können.

In Beispiel ?? wird gezeigt, wie die oben genannte Einschränkung für die GEHALT-Spalte der Tabelle MITARBEITER als Out Of Line Constraint angelegt wird.

Listing 13.2: Ein `CHECK`-Constraint als Out Of Line Constraint

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT gehalt_ck CHECK (Gehalt > 0 AND Gehalt <= 90000);
```

Um ein `CHECK`-Constraint als Inline Constraint anzulegen, muss es direkt bei der Tabellenerstellung mit angelegt werden. Beispiel ?? zeigt das gleiche Constraint noch einmal, aber als Inline Constraint.

Listing 13.3: Ein **CHECK**-Constraint als Inline Constraint

```
CREATE TABLE Mitarbeiter (
...
    Gehalt      NUMBER(12,2)
    CONSTRAINT gehalt_ck (Gehalt > 0 AND Gehalt <= 90000),
...
);
```



In welchem Format ein **CHECK**-Constraint angelegt wird, ob als Inline oder als Out Of Line Constraint, spielt keine Rolle. Beide Formen sind möglich. Der Unterschied besteht darin, dass sich ein Inline Constraint nur auf die Spalte beziehen kann, mit deren Definition es angelegt wurde. Ein Out Of Line Constraint kann sich auf alle Spalten der Tabelle beziehen, mit der zusammen es angelegt wurde.

Um die Auswirkungen des obigen Merksatzes zu zeigen, wird das GEHALT_CK-Constraint ein wenig modifiziert. Es muss jetzt auch die Spalte PROVISION mit einbezogen werden. Das Gesamtgehalt eines Mitarbeiters darf 90.000 EUR nicht überschreiten, die Provision mit eingerechnet.

Listing 13.4: Ein komplexes **CHECK**-Constraint

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT gehalt_ck CHECK (Gehalt > 0
                                  AND (Gehalt + (Gehalt * Provision / 100)) <= 90000);
```

13.2.2 Das NOT NULL-Constraint

Das **NOT NULL**-Constraint ist dafür zuständig sicherzustellen, dass beim Einfügen oder Ändern einer Tabellenzeile bestimmte Spalten immer einen Wert haben müssen.



Das **NOT NULL**-Constraint stellt eine Ausnahme zu allen anderen Constraints dar, denn es kann nur als Inline Constraint angelegt werden.

Beispiel ?? zeigt, wie ein **NOT NULL**-Constraint angelegt wird.

Listing 13.5: Ein **NOT NULL**-Constraint anlegen in Oracle

```
ALTER TABLE Mitarbeiter
MODIFY Gehalt CONSTRAINT gehalt_nn NOT NULL;
```

Um ein solches Constraint wieder rückgängig zu machen, kann die folgende Kurzform verwendet werden:

Listing 13.6: Das Gegenteil von `NOT NULL`

```
ALTER TABLE Mitarbeiter
MODIFY Gehalt NULL;
```

In den meisten DBMS wird ein `NOT NULL`-Constraint intern als `CHECK`-Constraint umgesetzt, weshalb Beispiel ?? und Beispiel ?? gleichbedeutend sind.

Listing 13.7: Die alternative Form eines `NOT NULL`-Constraints in Oracle

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT gehalt_nn CHECK (Gehalt IS NOT NULL);
```

In beiden Fällen wird intern ein `CHECK`-Constraint, nach dem in Beispiel ?? gezeigten Schema, angelegt. Auch in SQL Server ist dies der Fall. Im Gegensatz zu Oracle, muss bei SQL Server immer der Datentyp der Spalte mit angegeben werden, wenn eine Spalte ein `NOT NULL`-Constraint erhält.

Listing 13.8: Ein `NOT NULL` Constraint anlegen in SQL Server

```
ALTER TABLE Mitarbeiter
ALTER COLUMN Gehalt NUMERIC(12,2) NOT NULL;
```

Listing 13.9: Die alternative Form eines `NOT NULL` Constraints in SQL Server

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT gehalt_nn CHECK Gehalt IS NOT NULL;
```



Um in SQL Server eine Spalte mit einem `NOT NULL`-Constraint zu belegen, muss der Datentyp der Spalte mit angegeben werden, auch wenn dieser sich nicht ändern soll!

13.2.3 Das `UNIQUE`-Constraint

Das `UNIQUE`-Constraint hat die Aufgabe, dafür Sorge zu tragen, dass alle Werte, die in eine Tabellenspalte eingetragen werden, eindeutig sind.



In Oracle sind `NONE`-Werte eindeutig. Das heißt, in einer mit einem `UNIQUE`-Constraint belegten Spalte können beliebig viele `NONE`-Werte vorkommen. In SQL Server sind `NONE`-Werte nicht eindeutig. Somit kann in SQL Server nur ein `NONE`-Wert pro Tabellenspalte vorkommen, wenn die Spalte mit einem `UNIQUE`-Constraint belegt ist.

Beispiel ?? zeigt, wie in Oracle und SQL Server ein **UNIQUE**-Constraint auf die Spalte SOZVERSNR der Tabelle MITARBEITER gelegt wird.

Listing 13.10: Ein UNIQUE-Constraint anlegen

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT sozversnr_uk UNIQUE (SozVersNr);
```

Wie bereits beim **CHECK**-Constraint gezeigt, kann auch ein **UNIQUE**-Constraint als Inline Constraint erstellt werden. Beispiel ?? zeigt diesen Vorgang. Die Syntax ist in Oracle und SQL Server gleich.

Listing 13.11: Ein **UNIQUE**-Constraint als Inline Constraint anlegen

```
CREATE TABLE Mitarbeiter (
...
    SozVersNr      VARCHAR2(20)
    CONSTRAINT sozversnr_uk UNIQUE ,
...
);
```

Oftmals genügt es nicht, wenn der Wert einer Spalte eindeutig ist. Es kann auch sein, dass die Kombination mehrerer Werte aus mehreren Spalten eindeutig sein muss. In so einem Fall kann ein **UNIQUE**-Constraint auch auf eine Kombination mehrerer Spalten gelegt werden, wie Beispiel ?? zeigt.

Listing 13.12: Ein kombiniertes **UNIQUE**-Constraint anlegen

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT mitarbeiter_uk UNIQUE (Vorname , Nachname , SozVersNr);
```

13.2.4 Das **PRIMARY KEY**-Constraint

Das **PRIMARY KEY**-Constraint hat eine ganz besondere Aufgabe. Es ist dafür zuständig, ein Attribut oder eine Gruppe von Attributen einer Tabelle als eindeutig zu kennzeichnen, um so ein Identifikationsmerkmal für jede Tabellenzeile einer Tabelle zu schaffen.

Die Nutzung von Primärschlüsseln ist notwendig, da es eine wesentliche Leistung eines relationalen Datenbankmanagementsystems ist, die Datenkonsistenz zu gewährleisten und hierzu gehört auch das Vermeiden von redundanten Datensätzen.



Der Unterschied zwischen einem **UNIQUE**-Constraint und einem **PRIMARY KEY**-Constraint ist, dass ein **PRIMARY KEY**-Constraint keine NULL-Werte zulässt. Ein **PRIMARY KEY**-Constraint ist eine Mischung aus einem **NOT NULL**- und einem **UNIQUE**-Constraint.

Da eine relationale Datenbank nicht ohne **PRIMARY KEY**-Constraints auskommt, werden diese meist schon bei der Erstellung einer Tabelle angelegt.

Listing 13.13: Ein PRIMARY KEY-Constraint als Inline Constraint anlegen

```
CREATE TABLE Mitarbeiter (
    Mitarbeiter_ID      NUMBER CONSTRAINT mitarbeiter_pk PRIMARY KEY,
    ...
);
```

Genau wie bei einem **UNIQUE**-Constraint, kann es notwendig sein, einen Primärschlüssel nicht nur auf eine Spalte, sondern auf eine Gruppe von Spalten zu legen. Dies ist meist in schwachen Entitäten der Fall, da hier die Kombination zweier Primärschlüsse aus den beiden äußeren Entitäten als Primärschlüssel genutzt wird.

Listing 13.14: Ein PRIMARY KEY-Constraint als Out Of Line Constraint auf mehrere Spalten anlegen

```
CREATE TABLE Mitarbeiter (
    Mitarbeiter_ID      NUMBER ,
    Vorname              VARCHAR2(30) ,
    Nachname             VARCHAR2(35) ,
    ...
    CONSTRAINT mitarbeiter_pk
        PRIMARY KEY (Mitarbeiter_ID , Vorname , Nachname)
    ...
);
```

13.2.5 Das FOREIGN KEY-Constraint

In einem RDBMS steht üblicherweise keine Entität „einzelne im Raum“. Sie steht immer in Zusammenhang mit anderen Entitäten. Diese Zusammenhänge werden durch Foreign Key-Constraints dargestellt und überwacht.



Der Zusammenhang, in dem die Entitäten eines RDBMS stehen, wird als „Referentielle Integrität“ bezeichnet.

Ein Beispiel hierfür stellen die beiden Tabellen **MITARBEITER** und **BANKFILIALE** bereit. Sie stehen, durch die Spalte **BANKFILIALE_ID**, die in beiden Relationen vorkommt, in Zusammenhang zu einander. Dieser Zusammenhang besteht darin, dass jeder Mitarbeiter genau einer Bankfiliale zugeordnet ist. Das heißt, in die Spalte **BANKFILIALE_ID** der Tabelle **MITARBEITER** werden die Primärschlüsselwerte der Tabelle **BANKFILIALE** eingetragen, um so den Zusammenhang herzustellen. Beispiel ?? zeigt, wie ein Fremdschlüsselconstraint angelegt wird.

Listing 13.15: Ein Foreign Key-Constraint als Out Of Line Constraint anlegen

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT mitarbeiter_filiale_fk
FOREIGN KEY (Bankfiliale_ID)
REFERENCES Bankfiliale(Bankfiliale_ID);
```

Die Definition eines Fremdschlüssels als Out Of Line Constraint hat zwei Teile:

- Die `FOREIGN KEY`-Klausel: Sie legt fest, welche Spalte die referenzierende Spalte ist.
- die `REFERENCES`-Klausel: Sie legt fest, welche Spalte referenziert wird. Bei dieser Spalte muss es sich um eine Primärschlüssel- oder `UNIQUE`-Spalte handeln.



Wird ein Fremdschlüssel als Inline Constraint bei der Erstellung der Tabelle miterstellt, entfällt die `FOREIGN KEY`-Klausel.



Es gibt zwei Situationen, die in einer relationalen Datenbank keines Falls auftreten dürfen:

- Ein referenzierter Primärschlüsselwert wird gelöscht. Beispiel: Eine Bankfiliale, in der sich noch Mitarbeiter befinden, wird aus der Tabelle BANKFILIALE gelöscht. Dies würde Datensätze in der Tabelle MITARBEITER zurücklassen, die sich auf eine Filiale beziehen, die gar nicht mehr existiert.
- In eine Fremdschlüsselspalte wird ein Wert eingetragen, der in der referenzierten Primärschlüsselspalte nicht vorkommt. Beispiel: Ein Mitarbeiter wird in die Bankfiliale mit der ID 300 aufgenommen, welche gar nicht existiert. Auch hier würde sich ein Angestellter auf eine Abteilung beziehen, welche es nicht gibt.

In beiden Fällen wäre die Referentielle Integrität der Datenbank verletzt, was zu Informationsverlust bzw. fehlerhafter Information führt. Dies zu vermeiden ist die Aufgabe des Foreign Key-Constraints.

Listing 13.16: Ein Foreign Key-Constraint als Inline Constraint anlegen

```
CREATE TABLE Mitarbeiter (
...
    Bankfiliale_ID      NUMBER
    CONSTRAINT mitarbeiter_filiale_fk
        REFERENCES Bankfiliale(Bankfiliale_ID)
...
);
```

Der SQL-Standard kennt zwei Erweiterungen zum `FOREIGN KEY`-Constraint. Dies sind die Klauseln `ON DELETE CASCADE` und `ON DELETE SET NULL`.

- **`ON DELETE CASCADE`**: Wird ein referenzierter Wert gelöscht, werden automatisch alle referenzierenden Werte mitgelöscht. Beispiel: Wird eine Filiale aus der Tabelle `BANKFILIALE` gelöscht, werden automatisch auch alle Mitarbeiter gelöscht, welche sich in dieser Filiale befinden.
- **`ON DELETE SET NULL`**: Wird ein referenzierter Wert gelöscht, werden automatisch alle referenzierenden Werte auf `NULL` gesetzt. Beispiel: Wird eine Filiale aus der Tabelle `BANKFILIALE` gelöscht, wird die `BANKFILIALE_ID` eines jeden Angestellten automatisch auf `NULL` gesetzt.

Beide Zusätze können sehr nützlich sein, bergen jedoch auch große Risiken in sich. Wird die `ON DELETE CASCADE`-Klausel zu unvorsichtig angewandt, kann es passieren, dass Daten gelöscht werden, die gar nicht gelöscht werden dürfen.

Die `ON DELETE SET NULL` ist nicht so radikal, wie `ON DELETE CASCADE`, aber auch sie ist nicht ganz ungefährlich. Wird ein referenzierter Wert gelöscht, werden alle referenzierenden Werte kaskadierend auf `NULL` gesetzt. Das hat zur Folge, dass plötzlich Datensätze bestehen, die keinen Bezug mehr zu anderen Datensätzen haben.



Sowohl bei der `ON DELETE CASCADE`- als auch bei der `ON DELETE SET NULL`-Klausel muss mit äußerster Vorsicht gearbeitet werden.

Beispiel ?? und Beispiel ?? zeigen, wie diese Klauseln angewandt werden.

Listing 13.17: Ein Foreign Key-Constraint mit ON DELETE CASCADE-Klausel

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT mitarbeiter_filiale_fk FOREIGN KEY (Bankfiliale_ID)
    REFERENCES Bankfiliale(Bankfiliale_ID)
    ON DELETE CASCADE;
```

Listing 13.18: Ein Foreign Key-Constraint mit ON DELETE SET NULL-Klausel

```
ALTER TABLE Mitarbeiter
ADD CONSTRAINT mitarbeiter_filiale_fk FOREIGN KEY (Bankfiliale_ID)
    REFERENCES Bankfiliale(Bankfiliale_ID)
    ON DELETE SET NULL;
```

13.2.6 Das SQL Server DEFAULT-Constraint

In Microsoft SQL Server werden Standardwerte als Constraints an eine Spalte angefügt. Das Anfügen eines Default-Constraints an eine Spalte während der Tabellenerstellung funktioniert genauso wie in Oracle.

Listing 13.19: Erstellen einer Tabelle mit einem Standardwert

```
CREATE TABLE
Aktie ( Aktie_ID  NUMERIC ,
Name      VARCHAR(25) ,
Herkunft   VARCHAR(25) DEFAULT 'USA' ,
WKN       NUMERIC ,
ISIN      VARCHAR(12)
);
```

Der Unterschied zwischen Oracle und MS SQL Server zeigt sich aber, wenn ein Default-Constraint nachträglich hinzugefügt werden soll.

Listing 13.20: Tabellenspalte mit Standardwert hinzufügen in SQL Server

```
ALTER TABLE Aktie
ADD CONSTRAINT herkunft_dv
DEFAULT 'USA'
FOR Herkunft;
```

Anders als in Oracle muss für den SQL Server die `ADD CONSTRAINT`-Klausel benutzt werden.

13.3 Constraints umbenennen und löschen

13.3.1 Constraints umbenennen

Sowohl in Oracle als auch in SQL Server ist es möglich, ein Constraint umzubenennen.

Listing 13.21: Ein Constraint umbenennen in Oracle

```
ALTER TABLE Mitarbeiter
RENAME CONSTRAINT gehalt_ck TO gehalt_provision_ck;
```

Listing 13.22: Ein Constraint umbenennen in SQL Server

```
EXEC sp_rename 'gehalt_ck', 'gehalt_provision_ck', 'OBJECT';
```

13.3.2 Constraints löschen

Soll ein bereits bestehendes Constraint wieder entfernt werden, muss in Oracle und SQL Server die `DROP CONSTRAINT`-Klausel des `ALTER TABLE`-Kommandos genutzt werden.

Listing 13.23: Ein Constraint löschen

```
ALTER TABLE Mitarbeiter  
DROP CONSTRAINT mitarbeiter_filiale_fk;
```

Dies lässt sich auf alle fünf Constraintarten anwenden.

Enthält eine zu lösche Tabelle Primärschlüssel- oder UniqueConstraints, welche durch Fremdschlüssel anderer Tabellen referenziert werden, muss in Oracle zusätzlich die Klausel `CASCADE CONSTRAINTS` verwendet werden. Dadurch werden die Fremdschlüssel der anderer Objekte entfernt. In SQL Server müssen zuerst die referenzierenden Foreign Key Constraints gelöscht werden, ehe die Tabelle gelöscht werden kann.

Listing 13.24: Eine Tabelle mit Fremdschlüsselbeziehungen löschen

```
DROP TABLE Mitarbeiter CASCADE CONSTRAINTS;
```

13.3.3 Standardwerte in SQL Server löschen

Was Standardwerte sind, ist bereits aus dem vorhergehenden Kapitel bekannt. Wie sie in Oracle und in SQL Server angelegt werden ist ebenfalls bekannt. Was bisher noch nicht gezeigt wurde, ist, wie sie in SQL Server wieder gelöscht werden. Da in SQL Server ein Standardwert wie ein Constraint behandelt wird, muss auch die `DROP CONSTRAINT`-Klausel des `ALTER TABLE`-Statements verwendet werden, um einen Standardwert zu löschen.

Listing 13.25: Einen Standardwert in SQL Server löschen

```
ALTER TABLE Aktie  
DROP CONSTRAINT herkunft_dv;
```

Teil III

Datenbankadministration

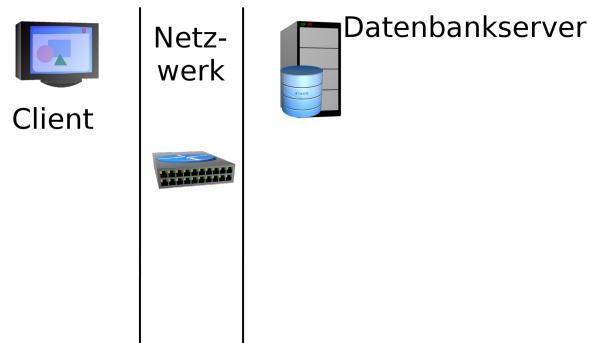
14 Einführung in die SQL Server Datenbankarchitektur

Inhaltsangabe

14.1 SQL Server und die Client-Server-Architektur

SQL Server Datenbanken sind dazu geschaffen, um große Datenmengen zu verwalten und diese in einer Multi-User Umgebung einer großen Anzahl von Benutzern zur Verfügung zu stellen.

Abb. 14.1:
Der SQL Server
in einer
Client-Server
Umgebung



14.1.1 Der Client

Auf der Client-Seite kann eine beliebige Software zur Anwendung kommen, die in der Lage ist, sich mit dem SQL Server zu verbinden. Ein Beispiel hierfür ist das „SQL Server Management Studio“, kurz SSMS.

Connection

Eine Connection ist eine physikalische Verbindung zwischen einem Client und dem Datenbankserver, die mittels eines Netzwerk- oder IPC-Protokolls aufgebaut wird. Der SQL Server kennt vier verschiedene Protokolle, welche die Herstellung einer Connection ermöglichen:

- **Shared Memory:** Das Shared Memory Protokoll ermöglicht die Verbindung zwischen Clientsoftware und Datenbankserver, sofern sich beide auf dem gleichen Rechner befinden. Es ist das einzige Protokoll, dass nur lokale Verbindungen zulässt.
- **TCP/IP:** Dies ist das mit Sicherheit weltweit meist genutzte Netzwerkprotokoll, das auch im SQL Server Umfeld am Gängigsten ist.
- **Named Pipes:** Named Pipes sind ein Konstrukt, welches aus der Unix-Welt stammt. Microsoft hat eine eigene, abgewandelte Implementierung, mit deren Hilfe eine Peer-to-Peer Kommunikation zwischen einem Client und einem Server hergestellt werden kann.

Mittels Named Pipes kann auf sehr einfachem Wege eine Verbindung hergestellt werden, jedoch sind die Möglichkeiten einer solchen Verbindung auch sicher zu gestalten nur sehr gering, weshalb fast immer TCP/IP den Vorzug bekommt.

- **VIA:** Das „Virtual Interface Adapter“ Netzwerkprotokoll ist ein mit TCP/IP vergleichbares Protokoll. Der Unterschied liegt darin, dass VIA für Hochgeschwindigkeitsverbindungen zwischen zwei Partnern ausgelegt ist. Die hohe Performance dieses Protokolls kann nur durch spezielle Hardware erreicht werden, was den Einsatz von VIA sehr kostspielig macht.



Das VIA-Protokoll gilt als deprecated und wird zukünftig aus dem SQL Server entfernt werden. Die aktuellen SQL Server Versionen 2008, 2008 R2, 2012 und 2014 unterstützen VIA aber noch.

Session

Sobald eine Connection zwischen dem Client und dem Datenbankserver besteht, kann sich der Benutzer am Server authentifizieren, um eine Session aufzubauen. Während eine Connection eine physikalische Verbindung darstellt, ist eine Session eine Kommunikationsverbindung, die einem authentifizierten Nutzer gewährt wird.



Eine Session stellt einen Zugang zur Datenbank dar.

14.1.2 Der Datenbankserver

Server

Als Datenbankserver wird der Hostrechner bezeichnet, auf dem die SQL Server Software installiert wird. Die Verwaltung dieses Teils der Architektur geschieht nicht durch den SQL Server, sondern durch das Betriebssystem. Der SQL Server nutzt lediglich die zur Verfügung gestellten Ressourcen.

Instanz

Bei einer Instanz handelt es sich um eine Sammlung aus Arbeitsspeicherstrukturen und Prozessen, die unter dem SQL-Server Windowsdienst zusammengefasst sind. Sie sollen ein schnelles und effizientes Arbeiten mit den Daten ermöglichen.

Auf einem Datenbankserver können mehrere Instanzen betrieben werden. Jede Instanz hat Ihre eigene Softwareinstallation, da bestimmte Teile der SQL Server Software instanzabhängig sind. Es werden zwei Instanzarten unterschieden:

- **Default Instance:** Die Standardinstanz ist immer die erste Instanz auf einem SQL Server. Sie wird durch den Rechnernamen und die Zeichenfolge MSSQLSERVER identifiziert, z. B. FEA11WV0010A\MSSQL
- **Named Instance:** Wenn mehr als eine Instanz auf einem Server betrieben werden muss, dann handelt es sich immer um sogenannte „benannte Instanzen“. Eine benannte Instanz wird durch den Rechnernamen, plus einen bei der Installation zuwählenden Namen identifiziert. Beispiele könnten sein: FEA11WV0010A\CRM oder FEA11WV0010A\ERP



Benutzt ein Client zur Verbindung mit dem Server nur den Rechnernamen, wird er automatisch mit der Standardinstanz verbunden!

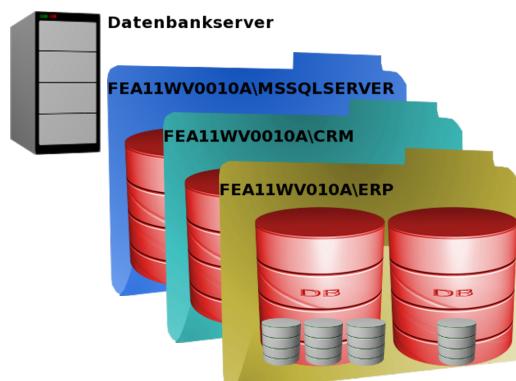
Datenbank

Datenbanken stellen die dritte Ebene in der Architektur des SQL Servers dar. Sie bestehen aus einer oder mehreren Datendateien und dienen als Datenspeicher. Sie enthalten sowohl Meta- als auch Nutzdaten. Jede Datenbank kann eine Größe von ca. 524 TB erreichen und wird von genau einer Instanz verwaltet. Zusätzlich zu den Datendateien besitzt eine Datenbank noch ein sogenanntes „Transaktionsprotokoll“.



Zwischen Datenbanken und Instanzen besteht bei SQL Server eine 1:n Beziehung, d. h. es können mehrere Datenbanken zu einer Instanz gehören.

Abb. 14.2:
Server - Instanzen
- Datenbanken



14.2 Speicherstrukturen der SQL Server Instanz

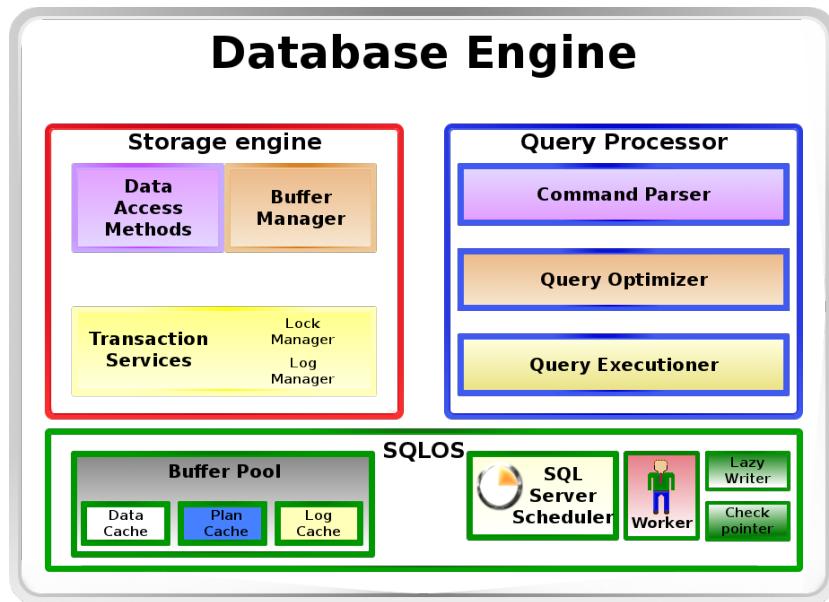


Abb. 14.3:
Die SQL Server
Database Engine

Die „Database Engine“ ist das Herzstück der SQL Server Architektur. Sie besteht aus einer Reihe von Caches und Prozessen, mit deren Hilfe die komplette Verarbeitung von Clientanfragen geschieht. Ihre vier Hauptkomponenten sind:

- Die Zugriffsprotokolle (in der Grafik nicht dargestellt),
- die „Storage Engine“,
- der „Query Processor“, der auch als „Relational Engine“ bezeichnet wird,
- und das SQL Server Operating System, kurz „SQLOS“ oder auch „SOS“.

Jede Anfrage eines Clients wird während ihrer Abarbeitung durch alle Ebenen gereicht. Da die Database Engine grundlegender Bestandteil des SQL Servers ist, ist es wichtig, deren Funktionsweise zu kennen und zu verstehen. Dieses Wissen ist hilfreich, um die Auswirkungen von Arbeitsspeicher und Datenträgern auf die Datenbankperformance einschätzen zu können. Des Weiteren helfen diese Kenntnisse bei der Erstellung von Backup und Recovery Strategien.

Das SQL Server Operating System, kurz SQLOS, bildet die Schnittstelle zwischen dem SQL Server und dem Betriebssystem. Diese Schicht wurde mit dem SQL Server 2005 eingeführt, um die immer komplexeren Anforderungen des SQL Servers an das Betriebssystem erfüllen zu können. Die wichtigsten Aufgaben des SQLOS sind:

- Memory management: Anforderung von Arbeitsspeicher für die Komponenten der Database Engine.
- Scheduling: Verwaltung von Threads¹ und Fibres²
- Deadlock Detection: Auflösen von Situationen, in denen sich zwei Prozesse gegenseitig blockieren.
- Asynchroner I/O: Schreiben von Daten, ohne auf eine Bestätigung zu warten.

14.2.1 Zugriffsprotokolle

Die Protokollebene ist die äußerste Schicht der SQL Server Database Engine. Ihre Hauptaufgabe ist es, den Clients zu ermöglichen, Connections zum Datenbankserver aufzubauen. In ihr sind die vier bereits im Vorfeld erwähnten Zugriffsprotokolle gekapselt:

- Shared Memory
- Named Pipes
- TCP/IP
- VIA

Ein weiterer Arbeitsschritt der auf dieser Ebene verrichtet wird, ist das Entgegennehmen von Clientrequests. Diese werden in ein Microsoft spezifisches Datenformat, mit Namen „Tabular data stream“ (TDS) umgewandelt. Dabei handelt es sich um ein Application Layer Protocol, welches zum Datentransfer zwischen einem Datenbankserver und einer Clientanwendung dient. Es hat verschiedene Fähigkeiten, wie z. B. die Authentifizierung von Clients, das Ausführen von SQL Batches und Stored Procedures oder das Ausliefern von Result Sets.



- [dd304523]

¹Thread (Informatik): Ein Thread ist ein Teil eines Prozesses

²Fibre (Informatik): Teil eines Prozesses, der speziell auf kooperatives Multitasking ausgerichtet ist

14.2.2 Der Buffer Pool und seine Bestandteile

Der Buffer Pool ist die größte, im Arbeitsspeicher befindliche, Komponente des SQL Servers. Er dient als Reservoir für die verschiedenen Caches des SQL Servers, wie z. B. den Data Cache oder den Plan Cache.

Der Data Cache

Der Data Cache dient als Raum für die Verarbeitung von Daten. Direkt nach dem Hochfahren einer SQL Server Instanz ist er mit einem frisch formatierten Dateisystem vergleichbar. Er besteht aus vielen 8 KB grossen Speicherseiten, den Buffers, die dazu bestimmt sind, Information aus den Datendateien aufzunehmen.



Als „Buffer“ wird eine im Buffer Pool befindliche Speicherseite (Page) bezeichnet.

Mit zunehmender Arbeitslast füllt sich der Data Cache mit Buffers, die von den Clients angefordert wurden. Es werden zwei Buffer-Arten unterschieden:

- **clean buffers:** Ein Buffer wird als „clean“ bezeichnet, wenn er entweder leer oder sein Inhalt synchron mit einer Speicherseite auf dem Datenträger ist.
- **dirty buffer:** Ein Buffer gilt als „dirty“, wenn sein Inhalt nicht synchron mit einem Buffer auf dem Datenträger ist, d. h. wenn er geänderten Inhalt aufweist.

Alle als clean markierten Buffer werden in einer Liste, der „Free Buffer List“ verwaltet. Diese Liste dient zum Auffinden freier Blöcke, so dass Informationen möglichst einfach und schnell im Data Cache abgelegt werden können.

Modifikationen an Tabellen oder Indizes finden grundsätzlich immer im Data Cache und nie auf dem Datenträger statt. Sollen die Daten einer Tabelle erstmalig modifiziert oder gelesen werden, müssen diese zuerst in den Data Cache geschrieben werden. Existiert wiederholter Bedarf für die gleichen Tabellenzeilen, entfällt der Zugriff auf den Datenträger, da sich die benötigten Inhalte bereits im Data Cache befinden. Auf diese Weise können Datenträgerzugriffe gespart und die I/O Performance verbessert werden.

Der Data Cache wird mittels eines LRU-K³ Algorithmus verwaltet, der dafür sorgt, dass häufig gefragte Speicherseiten im Data Cache verbleiben und nur selten benötigte Seiten aus dem Data Cache entfernt werden, um Platz für Neue zu schaffen. Eine nähere Beschreibung dieses Algorithmus übersteigt jedoch den Horizont dieser Unterrichtsunterlage.

³LRU = Least Recently Used

Der Plan Cache

Der Plan Cache enthält Informationen über alle ausgeführten SQL-Statements und Stored Procedures. Setzt ein Nutzer ein SQL-Statement ab, werden Informationen darüber in Form eines Ausführungsplanes im Plan Cache abgelegt. Wird exakt das gleiche Statement von einem anderen Nutzer erneut ausgeführt, können die vorhanden Informationen im Plan Cache wiederverwendet werden, was die Ausführungsgeschwindigkeit wesentlich erhöht.

Auch hier kommt ein LRU Algorithmus zur Verwaltung der Pläne zum Einsatz.

Der Log Cache

Der Log Cache nimmt Protokolleinträge zu allen in der Datenbank durchgeföhrten Änderungen auf, bevor diese auf den Datenträger geschrieben werden. Seine Struktur ist anders, als die des Data Caches oder die des Plan Caches. Er besteht aus einer Liste, die auf einem 32-Bit System bis zu 32 Einträge und auf einem 64-Bit System bis zu 128 Einträge umfassen kann. Jeder Eintrag verweist auf einem sogenannten „Log Block“. In den Log Blöcken werden dann die Änderungen protokolliert.

Ein Log Block kann eine Größe von 512 B bis 60 KB aufweisen. Größe und Anzahl der Log Blöcke sind abhängig von der Arbeitslast, die die SQL Server Instanz bewältigen muss. Sobald der Inhalt eines Log Blocks vom Log Manager auf den Datenträger geschrieben wurde, kann der Block wiederverwendet werden. Sollte die aktuelle Anzahl an Log Blöcken nicht ausreichen, kann der SQL Server neue Blöcke allokierten.

14.2.3 Die Prozessarchitektur des SQL Servers

Die Storage Engine und das SQLOS stellen eine Reihe von Hintergrundprozessen zur Verfügung, die die gesamte Arbeit im DBMS verrichten. Die meisten von Ihnen greifen auf verschiedene Buffers und Caches zu, die wiederum vom SQL Server Operating System verwaltet werden.

Data Access Methods

Die Data Access Methods sind mit dem Abarbeiten der Requests betraut. Sie koordinieren Buffer- und Transaction Manager. Wenn Zeilen aus der Datenbank gelesen werden müssen, initiieren sie den Lese- oder Schreibvorgang und alle dafür notwendigen Schritte. Sie führen diese Schritte aber nicht selbst aus.

An Ihrer statt muss der Buffer Manager, die Informationen auf dem Datenträger lesen und im Buffer Pool bereitstellen.

Buffer Manager

Der Buffer Manager hat die Aufgabe, Speicherseiten (Buffers) zu verwalten. Dies sind 8 KB große Einheiten, die in allen Speicherstrukturen (Buffers / Caches) des SQL Server genutzt werden. Immer dann, wenn die Access Methods einen Buffer benötigen, muss der Buffer Manager diesen im Buffer Pool zur Verfügung stellen. Dazu prüft er, ob die Seite bereits im Buffer Pool vorhanden ist oder nicht.

Transaction Manager

Transaktionen sind ein wichtiges Konzept innerhalb einer relationalen Datenbank. Sie ermöglichen es, Operationen die einen logischen Zusammenhang haben, zu gruppieren. Der Sinn der dahinter steckt ist, dass entweder alle Operationen einer Gruppe/Transaktion erfolgreich ausgeführt werden oder gar keine. Damit soll Dateninkonsistenz verhindert werden. Dieser Transaktionsmechanismus funktioniert innerhalb einer SQL Server Instanz, datenbankübergreifend.

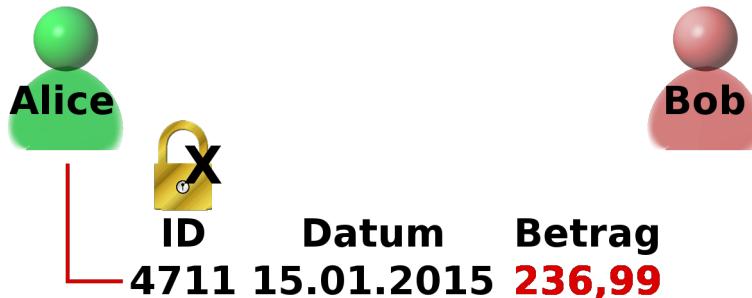
Zur Umsetzung eines Transaktionskonzeptes besitzt der Transaction Manager zwei wichtige Komponenten: Den „Lock Manager“ und den „Log Manager“.

Lock Manager

Das Setzen von Sperren ist in einer Mutli-User-Umgebung eine unerlässliche Tätigkeit. Mit Hilfe von Sperren wird gewährleistet, dass trotz konkurrierender Zugriffe die Daten konsistent bleiben. Die Aufgabe des Lock Managers ist es, in Zusammenarbeit mit dem Transaction Manager, Sperren im System zu setzen und zu verwalten. Die folgende Abbildung verdeutlicht, wie wichtig die Arbeit des Lock Managers ist

Die Benutzerin Alice soll eine Änderung an einer Tabellenzeile durchführen. Um dies ungestört tun zu können, muss die betreffende Zeile exklusiv für Alice gesperrt werden. Dadurch wird verhindert, dass Bob gleichzeitig mit Alice an der selben Zeile arbeitet. Nur so kann die Konsistenz der Tabellenzeile gewährleistet werden.

Abb. 14.4:
Eine exklusive
Schreibsperrre



Log Manager

Eine grundlegende Anforderung an moderne Datenbank Management Systeme ist der effektive Schutz vor Datenverlust. Der einfachste Fall, bei dem es zu Datenverlust kommen kann, ist der Absturz des Datenbankservers. Da durch ein solches Ereignis der gesamte Inhalt des Arbeitsspeichers verloren geht, verschwinden auch die im Buffer Pool gespeicherten Buffer mit neuem/geändertem Inhalt. Deshalb wird ein Mechanismus benötigt, der vor einem solchen Verlust schützt. Dieser Mechanismus wird in SQL Server als „Transaction Logging“ bezeichnet.

Alle Änderungen die im Data Cache durchgeführt werden, müssen zuvor durch den Buffer Manager in dem eigens dafür eingerichteten „Log Cache“ protokolliert werden. Dadurch wird gewährleistet, dass jede Änderung auf jeden Fall protokolliert wird, wodurch das Risiko für Datenverlust deutlich sinkt.

Um den Inhalt des Log Buffers auf den Datenträger zu schreiben, hat man dem SQL Server einen eigenen Hintergrundprozess spendiert, den „Log Manager“. Seine Aufgabe ist es, den Inhalt des Log Caches im Write-Ahead-Logging Verfahren auf den Datenträger zu sichern. Dabei muss er immer zuerst die Protokolleinträge schreiben, bevor die Access Methods Änderungen auf den Datenträger übertragen. Sollten nun geänderte Buffer verloren gehen, können die Änderungen aus dem Transaktionsprotokoll rekonstruiert werden.

Jede Änderung, die ein Nutzer an einer Tabellenzeile vornimmt, wird zuerst im Log Cache, niedergeschrieben, bevor sie vollzogen wird. Da dieses Protokoll sehr wichtig ist, wird es nahezu ohne Unterbrechung vom Log Manager auf dem Datenträger gesichert.



Der Lazy Writer

Der Lazy Writer hat die Aufgabe

- im Bufferpool nach Dirty-Buffers zu suchen,

- deren Inhalt in die Datendateien zurückzuschreiben,
- den Block zu leeren und
- ihn erneut auf die Free Buffer List zu setzen.

Er tut dies, um zu gewährleisten, dass immer eine ausreichende Menge freier Buffer im Data Cache zur Verfügung steht. Da der Data Cache mittels eines LRU Algorithmus verwaltet wird schreibt der Lazy Writer immer alte, wenig benutzte Buffer auf den Datenträger.

Ist die Anzahl der Clean Buffers im Data Cache sehr hoch, hat der Lazy Writer auch nur wenig zu tun. Sollte es jedoch so sein, dass der Lazy Writer nahezu permanent schreiben muss, so ist dies ein Anzeichen für eine zu geringe Speichermenge im Data Cache.

Im ganz extremen Situationen kann es sogar soweit kommen, dass der Lazy Writer Hilfe von Worker Threads (später in diesem Kapitel erläutert) benötigt. Diese führen dann eine interne Routine namens „HelpLazyWriter“ aus um ihn bei seiner Arbeit zu unterstützen.

Checkpoints und der Checkpointer



Per Definition ist ein Checkpoint: „Ein Ereignis, das durch andere Ereignisse ausgelöst wird!“.

Auslöser für Checkpoints sind:

- Das SQL-Kommando **CHECKPOINT**
- Ein zu 70 % gefülltes Transaktions Log (nur im Simple Recovery Model)
- Die geschätzte Zeit für ein Recovery ist größer als die angegebene Recovery Zeitspanne.
- Es wird ein Backup oder ein Snapshot der Datenbank angefertigt
- Der SQL Server wird heruntergefahren

Bei jedem Checkpoint werden geänderte Speicherseiten, aus dem Data Cache auf den Datenträger geschrieben. Dadurch gewährleistet die Datenbank zum einen Datenkonsistenz und zum andern hält sie die Zeitspanne gering, die nach einem Absturz des Servers für ein Recovery benötigt würde, da möglichst viele

Speicherseiten mit bestätigten Änderungen, in regelmäßigen Intervallen auf den Datenträger geschrieben werden.

Ausgeführt werden Checkpoints durch einen Hintergrundprozesse namens „Checkpointer“. Das ist der Prozess, der beim Auslösen eines Checkpoints den Data Cache nach geänderten Seiten durchsucht und die gefundenen Seiten auf den Datenträger überträgt. Der Unterschied zum Lazy Writer ist, dass er nur den Inhalt von Dirty Buffer auf den Datenträger überträgt, ohne dabei die Free Buffer List anzupassen.



Der Checkpointer kümmert sich nur darum, dass die Zeitspanne für ein Instance Recovery möglichst gering gehalten wird und nicht um freien Speicher im Data Cache.

Der Scheduler

Die Zuteilung von Prozessorressourcen an Threads ist ein unabdingbarer Bestandteil des Multitaskings. Jedes multitaskingfähige Betriebssystem besitzt deshalb einen Scheduler, der im Zeitscheibenverfahren Prozessorzeit an die Threads verteilt. Das Ziel des Schedulers ist es, die verfügbare Prozessorkapazität möglichst gleichmäßig auf alle Threads zu verteilen. Dabei wird zwischen zwei verschiedenen Scheduling-Verfahren unterschieden:

- Präemptiv⁴: Der Scheduler behält die Kontrolle über alle Ressourcen. Er teilt sie zu und entzieht sie auch wieder. Ein einzelner Thread kann nicht das gesamte System blockieren oder zum Absturz bringen.
- Kooperativ: Der Scheduler teilt Ressourcen zu und wartet darauf, dass der Thread sie wieder zurück gibt. Die Threads bestimmen selbst, wann sie ihre Ressourcen abgeben.

Der Windowscheduler arbeitet nach dem präemptiven Modell weshalb er nicht in der Lage ist, dem SQL Server optimale Performance zu gewährleisten, da er nicht speziell auf die Anforderung des SQL Servers zugeschnitten ist. Mit der Einführung des SQL Server Operating System Schedulers, kurz SOS, wurde dem SQL Server die Möglichkeit eröffnet, auch große Datenbanken performant zu verwalten. Dies ist möglich, da der SOS eine Mischung aus präemptivem und kooperativem Modell benutzt, was eine bessere Zusammenarbeit der SQL Server Workerthreads ermöglicht. Konkret bedeutet dies:

Wie beim präemptiven Multitasking üblich, wird eine Obergrenze für die nutzbare Prozessorzeit vom SOS festgelegt (4 Millisekunden). Threads die mehr als die ihnen zur Verfügung stehenden Ressourcen benötigen, werden nicht gestoppt. Diese haben die Möglichkeit sich in eine Warteliste einzutragen und

⁴Präemptiv (Duden): einer sich bereits abzeichnenden Entwicklung zuvorkommend, vorsorglich, vorbeugend

neue Prozessorzeit zu erhalten, sobald dies möglich ist. Threads die ihre Ressourcen nicht mehr benötigen, geben diese unmittelbar zurück.



Worker werden nur angehalten, wenn sie ihre Zeitscheibe von 4 Millisekunden überschreiten und sich nicht in einer Operation befinden, die keinesfalls unterbrochen werden darf.

SQL Server Workerthreads

Workerthreads sind die „ausführenden Organe“ des SQL Servers. Sie werden bei Bedarf vom Scheduler erstellt und verbrauchen im Minimum 0,5 M (32-Bit System) bzw. 2 M (64-Bit System) Arbeitsspeicher. Jeder Worker ist an die CPU gebunden, von der er erstellt wurde. Ein Wechsel zu einer anderen CPU ist nicht möglich. Es kann jedoch sein, dass es so erscheint, als würde es zu einem Wechsel kommen, wenn ein Worker zerstört und neu erstellt wird.

Die Verteilung der Arbeit auf die Workerthreads geschieht ebenfalls durch den Scheduler. Ein Arbeitspaket für einen Worker wird als „Task“ bezeichnet.



Jeder Worker arbeitet einen Task immer vollständig ab, bevor er neue Arbeit annimmt.

Worker werden durch den Scheduler eliminiert, wenn:

- sie mindestens 15 Minuten im Leerlauf (idle) waren oder
- wenn der SQL Server zu wenig Arbeitsspeicher zur Verfügung hat.

Die Workerthreads greifen auf die SQL Server Access Methods zu, um Daten zu verarbeiten und sie dann den Clients zur Verfügung zu stellen. Sie sind somit das Bindeglied zwischen dem Client und der Database Engine.

Tasks

Ein Task ist ein Arbeitspaket, dass einem Worker von seinem Scheduler zugewiesen wird. Sie entstehen dadurch, dass Clients Anforderungen (sogenannte „Requests“) an den SQL Server senden. Unter einem Request versteht der SQL Server einen einzelnen Batch⁵.

14.3 Query Processor (Relational Engine)

Der SQL Server Query Processor, der auch als Relational Engine bezeichnet wird, beinhaltet Werkzeuge, die dafür Zuständig sind zu entscheiden, was beim Ausführen einer Abfrage getan werden muss und wie

⁵Batch = Eine Reihe von SQL-Anweisungen, die mit dem GO-Befehl abgeschlossen werden

dies am performantesten geschehen kann. Sie steht in enger Verbindung mit der Storage Engine, da sie diese benötigt, um Zugriff auf Datenträger und Arbeitsspeicher zu erhalten.

14.3.1 Command Parser

Die Hauptaufgabe des Command Parsers ist es, T-SQL-Statements auf eine korrekte Syntax hin zu überprüfen und sie in ihre Bestandteile zu zerlegen. Diese Tätigkeit wird „parsen“ genannt.

Nach dem Parsen wird ein zweiter Arbeitsschritt vollzogen, das „Binding“. Beim Binding werden alle Objektbezeichner (Tabellennamen, Viewnamen, usw.) durch deren Unique Identifier ersetzt.

Parsen und Binding geschehen, um das Statement in eine neue Form zu bringen, die für die Verarbeitung durch die nächste Komponente, den Query Optimizer, besser geeignet ist. Diese neue Form wird als „Query Tree“ bezeichnet.

14.3.2 Query Optimizer

Dies ist der wohl komplexeste Bestandteil der Database Engine. Er nimmt den Query Tree vom Command Parser entgegen und bereitet ihn für die Ausführung vor. Sein Hauptaugenmerk liegt dabei auf den DML-Kommandos **SELECT**, **INSERT**, **UPDATE** und **DELETE**, da diese auf unterschiedliche Art und Weise abgearbeitet werden können. Andere Statements, wie beispielsweise DDL-Kommandos können nicht optimiert werden und werden daher auch nicht durch den Optimizer betrachtet.

Das Arbeitsergebnis des Query Optimizers ist der „Ausführungsplan“. Er kann als eine Art Arbeitsplan oder Kochrezept verstanden werden. In ihm sind alle Einzelschritte enthalten, die für die Ausführung des SQL-Statements notwendig sind. Der Weg zu einem performanten Ausführungsplan ist aufwendig. Der Query Tree muss in kleine Stücke zerlegt werden (normalisieren), Statistiken und Indizes müssen geprüft werden.

Zusätzlich zu all diesen Aufgaben muss der Optimizer auch darauf achten, dass die Erstellung eines optimalen Ausführungsplanes nicht deutlich länger dauert, als dessen Ausführung. Unter Umständen kann es sinnvoll sein, einen weniger optimalen Plan heranzuziehen, um so Zeit bei der Planerstellung zusparen und letztendlich das gewünschte Ergebnis in einer kürzeren Zeit zu erzielen, als dies nach der Erstellung des optimalsten Planes möglich wäre.

Abb. 14.5:
Berechnung von
Ausführungsplä-
nen



14.3.3 Query Executor

Der Query Executor ist der Dritte im Bunde und hat die Aufgabe einen Ausführungsplan zu verarbeiten, also das SQL-Statement auszuführen.

- [dn205319]



14.4 Datendateien - Aufbau und Verwaltung

Jede SQL Server Datenbank besteht aus wenigstens einer Datendatei, die mindestens 5 MB groß sein muss und maximal 16 TB groß sein darf. Es können bis zu 32.767 Datendateien pro Datenbank genutzt werden. Der Dateiname der ersten Datendatei einer jeden Datenbank trägt immer die Dateiendung *.mdf, jede weitere Datendatei benutzt statt dessen *.ndf.



Die Dateiendung *.mdf wird in der Literatur unterschiedlich interpretiert. Teilweise als „Master Data File“ aber manchmal auch als „Meta Data File“!

Datendateien werden in sogenannten „Dateigruppen“ gegliedert. Dies sind logische Verwaltungseinheiten, die aus einer oder mehreren Datendateien bestehen. Sie dienen im Wesentlichen zur Vereinfachung administrativer Tätigkeiten, wie z. B. Backup und Recovery verschiedener Teile der Datenbank. Statt die Namen von mehreren Datendateien angeben zu müssen, genügt es, nur den Namen der Dateigruppe zu benutzen.



Der Name der ersten Dateigruppe einer SQL Server Datenbank lautet „Primary“. Diese Dateigruppe muss immer existieren und kann nicht umbenannt werden. Es können bis zu 32.767 Dateigruppen pro Datenbank angelegt werden.

14.4.1 Aufbau einer Datendatei

Datendateien bestehen aus „Speicherseiten“, engl. „Pages“, welche die kleinsten Speichereinheiten sind, die von SQL Server verwaltet werden. Es handelt sich um 8 KB große Portionen, die zu unterschiedlichen Zwecken genutzt werden. Insgesamt gibt es neun verschiedene Arten von Speicherseiten:

- Data Pages
- Index Pages
- LOB Pages
- Global Allocation Map Pages
- Shared Global Allocation Map Pages
- Page Free Space Pages
- Index Allocation Map Pages
- Bulk Changed Map Pages
- Differential Changed Map Pages

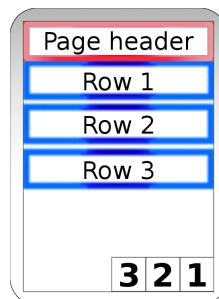


Abb. 14.6:
Aufbau einer
Data Page

Data Pages

Data Pages sind der häufigste Typ Speicherseite in einer Datenbank. Sie nehmen alle Nutzdaten, in Form von seriell gespeicherten Zeilen auf und teilen sich in drei Bereiche:

- Page Header (96 Bytes)
- Row Space (ca. 8.060 Bytes)
- Row Offset Table (36 Bytes +)

Der Page Header enthält Verwaltungsinformationen zur Speicherseite (Seitennummer, Seitentyp, freier Speicher, Extent ID, usw.). Er umfasst eine Länge von 96 Bytes. Am unteren Ende der Data Page befindet sich die Row Offset Table. Sie hat eine Mindestgröße von 36 Bytes und enthält für jede

Zeile das Offset. Es gibt an, wie viele Bytes die Tabellenzeile vom Page Header entfernt liegt. Beispiel:

- Zeile Nummer 1 hat eine Länge von 100 Bytes und beginnt 1 Byte hinter dem Page Header: Das Offset ist 1.
- Zeile Nummer 2 hat eine Länge von 120 Bytes und beginnt 1 Byte hinter Zeile Nummer 1. Das Offset ist (Offset Zeile 1 + Länge Zeile 1) + 1 = (1 Byte + 100 Byte) + 1 Byte = 102.
- Zeile Nummer 3 beginnt 1 Byte nach Zeile Nummer 2. Das Offset ist: (Offset Zeile 2 + Länge Zeile 2) + 1 = 102 Byte + 120 Byte + 1 Byte = 223.



Für jedes Offset wird 1 Byte benötigt, um es zu speichern. Reichen die 36 Bytes der Row Offset Table nicht aus, kann sie wachsen.

Tabellenzeilen

Für Tabellenzeilen werden innerhalb einer Speicherseite 8.060 Bytes reserviert (8.192 - 96 Bytes - 36 Bytes = 8.060 Bytes). Diesen Platz müssen sich die Zeilen mit der Row Offset Table teilen, da diese wächst, falls mehr als 36 Zeilen in einer Seite zu speichern sind.

Tabellenzeilen können sich nicht über mehrere Speicherseiten erstrecken. Seit SQL Server 2005 ist es jedoch möglich, dass einzelne Spalten in einen „Row Overflow Data“ Bereich verschoben werden. Es gelten dabei folgende Regeln:

- Die maximale Länge einer Tabellenzeile ist 8.060 Bytes.
- übersteigt die Länge einer Zeile 8.060 Bytes, können Tabellenspalten mit variabler Länge (Datentyp: Varchar, NVarchar, Varbinary und SQL_Variant) in den Row Overflow Data Bereich verschoben werden.
- Wird eine Spalte verschoben, verbleibt ein 24 Byte großer Zeiger in der Originalspeicherseite, der auf den Row Overflow Data Bereich zeigt.
- Es wird immer mit der breitesten Spalte variabler Länge begonnen.
- Verringert sich die Länge einer Zeile unter die 8.060 Byte Grenze, werden die Spalten aus dem Row Overflow Data Bereich zurück in die Data Page verschoben.

Extents

Extents sind logische Verwaltungseinheiten, die zur effizienten Verwaltung von Speicherseiten dienen. Jedes Extent besteht aus acht zusammenhängenden Speicherseiten, hat also eine Größe von 64 KB. Wenn die Datenbank neuen Speicherplatz allokiert (anfordern) muss, wird immer gleich ein ganzes Extent angefordert. Damit wird vermieden, dass die Datenbank Speicherseite für Speicherseite anfordern muss, was einen ungleich höheren Aufwand bedeuten würde.

Es gibt zwei unterschiedliche Arten von Extents:

- Mixed Extents
- Uniform Extents

Der Unterschied zwischen beiden Varianten besteht in deren Nutzung. In einem Uniform Extent gehören alle Pages zu einem und demselben Objekt. In einem Mixed Extent ist dies nicht der Fall.



Mixed Extents

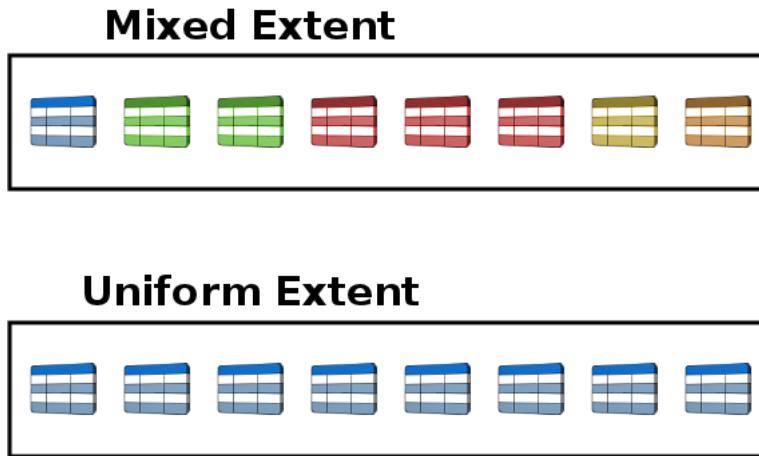
Der Name „Mixed Extents“ röhrt daher, dass sich mehrere Objekte die Speicherseiten eines solchen Extents teilen. Diese Art von Extent ist für kleine Objekte gedacht. Wenn ein Objekt neu erstellt wird, besteht es zuerst aus nur einer einzigen Page. Diese Page wird in einem Mixed Extent angefordert. Wächst das Objekt, werden weitere Speicherseiten aus Mixed Extents angefordert, bis das Objekt eine Größe von acht Data Pages erreicht hat.

Ab einer Größe von acht Pages, werden nur noch Uniform Extents für das betreffende Objekt angefordert. Das bedeutet, dass die ersten acht Pages eines Objekts sich in Mixed Extents befinden, während die neunte Speicherseite und alle Folgenden in Uniform Extents allokiert werden. Durch die Nutzung von Mixed Extents für kleine Objekte wird eine effiziente Außennutzung des vorhandenen Speicherplatzes erreicht.

Uniform Extents

In einem Uniform Extent gehören alle Speicherseiten zu einem und demselben Objekt. Da alle Pages in einem solchen Extent unmittelbar aufeinander folgen, wird erreicht, dass die Fragmentierung großer Objekt geringer wird. Außerdem werden immer gleich acht Seiten für das Objekt angefordert, was die Anzahl der Speicheranforderungsorgänge deutlich reduziert.

Abb. 14.7:
Mixed und
Uniform Extents



14.4.2 Speicherverwaltung in den Datendateien

Damit das Auffinden freier Speicherseiten performant gelingen kann, muss SQL Server verschiedene Informationen besitzen. Dies sind:

- Welchen Status hat ein Extent (Mixed oder Uniform)?
- In welchem Extent gibt es noch freie Seiten?

Diese Verwaltungsinformationen werden in drei verschiedenen Seitentypen abgelegt:

- GAM Pages: Global Allocation Maps
- SGAM Pages: Shared Global Allocation Maps
- PFS Pages: Page Free Space Pages

Die Global Allocation Map (GAM)

In einer GAM Page wird die Information gespeichert, welche Extents noch freie Seiten haben und welche bereits vollständig belegt sind. Jede GAM hat eine Größe von 64 KB und ist in der Lage, den Zustand von 64.000 Extents zu verwalten. Dies entspricht einer Datenmenge von ca. 4 GB.

Die Informationen in einer GAM liegen in Form ein Bitmap vor. Für jedes zuverwaltende Extent wird genau ein Bit benutzt. Hat das Bit den Wert 0, gilt das Extent als Belegt. Hat es den Wert 1, ist das verwaltete Extent frei.

Eine GAM verwaltet immer die 64.000 direkt auf sie selbst folgenden Extents. Das erste Bit in der GAM steht für das erste Extent hinter der GAM, das zweite Bit für das zweite Extent hinter der GAM und das Dritte für das dritte Extent, usw.

Ist eine Datenbank so groß geworden, dass sie mehr als 64.000 Extents umfasst, so wird einfach eine weitere GAM in einer Datendatei angelegt. Diese verwaltet dann die nächsten 64.000 Extents (Extent 64.001 bis Extent 128.000).

Die Shared Global Allocation Map (SGAM)

Der Aufbau einer SGAM ist mit dem einer GAM identisch. Auch die SGAM hat eine Größe von 64 KB, enthält eine Bitmap und verwaltet 64.000 Extents, jedoch haben die Bits einer SGAM eine andere Bedeutung, als die der GAM.

Die SGAM speichert Information über die Nutzung eines Extents, als Mixed Extent oder als Uniform Extent. Die Bits der SGAM haben folgende Bedeutung:

- 0: Das Extent ist entweder ein belegtes Mixed Extent oder es handelt sich nicht um ein Mixed Extent.
- 1: Das Extent ist ein Mixed Extent mit freien Speicherseiten.

Für die SGAM gilt, genau wie für die GAM, dass in einer Datenbank mit mehr als 64.000 Extents eine weitere SGAM angelegt wird.

Zusammenhang zwischen GAM und SGAM

Damit die Informationen der Global Allocation Map und der Shared Global Allocation Map einen Sinn ergeben, müssen beide Strukturen immer im Zusammenhang gesehen werden. Die folgende Tabelle verdeutlicht, was die unterschiedlichen Bit-Kombinationen der GAM und der SGAM aussagen.

Tabelle 14.1: Bedeutung von GAM und SGAM

Bedeutung	GAM	SGAM
Freies, unbenutztes Extent	1	0
Uniform oder Full Mixed Extent	0	0
Mixed Extent mit freien Seiten	0	1

Extentallokation

Uniform Extents werden allokiert, indem die GAM nach einem freien Extent durchsucht wird (GAM-Bit = 1). Diese Bit wird anschließend auf 0 gekippt.

Um ein Mixed Extent mit freien Seiten zufinden, muss die SGAM nach einem Bit mit dem Wert 1 gescanned werden. Dieses Bit bleibt unverändert, solang sich noch freie Seiten in dem Extent befinden.

Soll ein neues Mixed Extent allokiert werden, muss zuerst die GAM nach einem freien Extent (GAM-Bit = 1) abgesucht werden. Diese Bit wird anschließend auf den Wert 0 gekippt. Zuletzt muss noch das korrespondierende Bit der SGAM auf den Wert 1 gesetzt werden.

Aufbau der PFS

Sobald aber ein Extent mit freien Pages gefunden wurde, stellt sich die Frage: „Wie viel Speicherplatz ist in den einzelnen Speicherseiten noch frei?“. Hier sind es die „Page Free Space Pages“ (kurz PFS) die Abhilfe schaffen.

Sie speichern Informationen über den ungefähren Füllstand der Data Pages. Jede PFS kann dabei bis zu 8.000 einzelne Data Pages verwalten. Umfasst eine Datenbank mehr als 8.000 Data Pages, werden in regelmäßigen Abständen weitere PFS-Pages angelegt.



Eine einzelne PFS verwaltet ein Datenvolumen von bis zu 64 Gigabyte!

Im Gegensatz zur Global Allocation Map oder zur Shared Global Allocation Map, arbeitet eine Page Free Space Page nicht mit einer Bitmap, sondern mit einer Bytemap. Für jede einzelne Speicherseite werden hier 7 Bit zur Darstellung des Füllgrades und zur Speicherung diverser anderer Informationen genutzt.

- Bit 0 bis 2: Der Füllgrad, unterteilt in fünf Gruppen.
 - 0x00: Frei
 - 0x01: Belegung zwischen 1 % und 50 %
 - 0x02: Belegung zwischen 51 % und 80 %

- 0x03: Belegung zwischen 81 % und 95 %
- 0x04: Belegung zwischen 96 % und 100 %
- Bit 3 = 0x08: Es befinden sich Ghost-Records⁶ auf der Seite.
- Bit 4 = 0x10: Bei der Seite handelt es sich um eine Index Allocation Map.
- Bit 5 = 0x20: Die Seite liegt in einem Mixed Extent
- Bit 6 = 0x40: Die Seite ist belegt

⁶Ghost-Record = Ein als gelöscht markierter Datensatz innerhalb einer Index-Leaf-Page

Eine zu 30 % belegte Data Page, die sich in einem Mixed Extent befindet, hätte den Wert 0x61 (0x40 + 0x20 + 0x01). Eine zu 90 % belegte Page in einem Uniform Extent würde hingegen mit dem Wert 0x43 (0x40 + 0x03) geführt werden. So kann durch Kombination der einzelnen Bits jeder beliebige Füllstand abgebildet werden.

Nachdem nun mit Hilfe der PFS eine Data Page mit ausreichend Speicherplatz identifiziert wurde, kann durch einen Blick in den Page Header der exakte Füllstand herausgefunden werden. Durch die Nutzung der PFS wird vermieden, dass SQL Server, Page für Page nach dem exakten Füllstand absuchen muss, bis eine Seite mit genügend freiem Speicher gefunden wurde.

Aufbau einer Datendatei

GAM, SGAM und PFS-Pages werden immer nach dem gleichen Muster in einer Datendatei abgelegt:

- Seite Nummer 1: Der Datendateiheader
- Seite Nummer 2: GAM
- Seite Nummer 3: SGAM
- Seite Nummer 4: PFS
- Ab Seite Nummer 5: Data Pages und andere Seiten

14.5 Das Transaktionsprotokoll

Eine der Anforderungen, die an ein relationales Datenbank Management System gestellt werden ist, dass Änderungen an einer Datenbank niemals verloren gehen dürfen. In nahe zu allen DBMS wird diese Forderung mit Hilfe eines Änderungsprotokolls umgesetzt.

Das Transaktionsprotokoll ist in Form einer eigenen Datei mit der Endung *.ldf implementiert. Es ist möglich, dass Transaktionsprotokoll auf zwei oder mehr Dateien auszudehnen, was jedoch nur geschehen sollte, wenn dies die Situation unbedingt erfordert. Die Einträge des Protokoll folgen nicht dem Format der Speicherseiten, sondern besitzen ein eigenständiges Aussehen.



Jede Datenbank hat ihr eigenes Transaktionsprotokoll. Es wird direkt bei der Datenbankerstellung mit seiner Datenbank untrennbar verknüpft!

14.5.1 Features des Transaktionsprotokolls

Mit Hilfe des Transaktionsprotokolls kann eine MS SQL Server Datenbank die folgenden Features zur Verfügung stellen:

- **Zurückrollen einzelner Transaktionen:** Unter zur Hilfenahme der im Protokoll gespeicherten Informationen können die Änderungen einer Transaktion vollständig rückgängig gemacht werden.
- **Recovery offener Transaktionen nach einem Instanzabsturz:** Beim Starten einer SQL Server Instanz müssen alle offenen Transaktionen (Transaktionen die nicht durch ein `COMMIT` bestätigt wurden) rückgängig gemacht werden, um die Konsistenz des Datenbestandes zu gewährleisten.
- **Point-In-Time Recovery:** Unter bestimmten Umständen kann es erforderlich sein, eine Datenbank auf einen genau definierten Zeitpunkt zurückzusetzen. Die Inhalt des Protokolls sind ein wesentlicher Bestandteil eines solchen Prozesses.
- **Transaktionsreplikation:** Die Transaktionen einer Datenbank können in eine andere Datenbank übertragen und ausgeführt werden.
- **Stand-By Server:** Mittels des Transaktionsprotokolls kann eine Art „Aktiv-Passiv-Cluster“ eingerichtet werden. Während der eine Datenbankserver arbeitet, wird ein zweiter ständig mit den Änderungen des ersten versorgt. Falls der erste Datenbankserver aus, kann der zweite nach einer sehr kurzen Recovery-Phase die Arbeit übernehmen.

14.5.2 Aufbau des Transaktions Logs

Der Log file header

Jede Transaktionsprotokolldatei hat einen 8 KB großen Headerblock, der ganz am Anfang der Datei liegt. Dort werden verschiedene Metadaten abgelegt, wie z. B. die aktuelle Logfile size oder auch die Angaben zum automatischen Wachstum.

Standardgröße des Logfiles

Sofern beim Anlegen einer Datenbank keine Angabe zum Transaktionsprotokoll gemacht wird, benutzt der SQL Server zwei Werte, um die Größe der Log Datei zu ermitteln:

- 0,5 MB
- 25 % der Gesamtgröße aller Datendateien der Datenbank

Je nachdem, welcher der beiden Werte größer ist, wird dieser Benutzt. Beispiel: Eine fiktive Datenbank hat eine Gesamtgröße von 1 GB. 25 % von 1 GB sind 256 MB. 256 MB sind größer als 0,5 MB. Das Logfile würde mit 256 MB angelegt werden.



- [PSRSSIVsnadlfs]

Die Log Sequence Number (LSN)

Aus logischer Sicht ist das Transaktionsprotokoll wie eine fortlaufende Liste aufgebaut. Die Einträge werden immer am Ende des Log Files angehängt und jeder wird durch einen eindeutigen Schlüssel, die „Log Sequence Number“, identifiziert.

Die Log Sequence Number, kurz LSN, ist eine Zahl des Typs **NUMERIC(25,0)**, die in fortlaufender und aufsteigener Reihenfolge an die Einträge des Transaktionsprotokolls vergeben wird. Jede LSN markiert einen Log-Eintrag eindeutig.

Die Darstellung der LSN erfolgt manchmal dezimal und manchmal auch hexadezimal. Wird die LSN als Hexadezimalzahl dargestellt, kann man erkennen, dass sie aus drei Bestandteilen zusammengesetzt ist:

Beispiel LSN: 00000015:000001a3:02ef

- 00000015: Virtual Log File Sequence Number
- 000001a3: Log Block Offset
- 02ef: Log Block Slot Number

Die gleiche LSN sieht dezimal dargestellt so aus: 21000000041900751. Die Dezimaldarstellung wird durch eine einfache Umrechnung der drei hexadezimalen Bestandteile in Dezimalzahlen erreicht: $00000015_{(16)} = 210000000_{(10)}$, $000001a3_{(16)} = 41900_{(10)}$ und $02ef_{(16)} = 751_{(10)}$.

Transaktionsprotokolleinträge - Log Records

Im Transaktionsprotokoll sind Einträge zu allen an der Datenbank durchgeführten Änderungen enthalten. Außerdem werden Beginn und Ende einer jeden Transaktion festgehalten. Die in einem Eintrag enthaltenen Daten sind:

- Eine Beschreibung der Änderung
- IDs der geänderten Speicherseiten
- Hinzugefügte, geänderte oder gelöschte Datenwerte
- ID, Beginn und Ende der Transaktion
- Die Log Sequence Number

Jeder Eintrag (Log Record) im Transaktionsprotokoll trägt eine eindeutige LSN, die seine genaue Position im Protokoll definiert. Alle Log Records, welche zu einer Transaktion gehören, werden als Sequenz gespeichert und tragen alle die gleiche TransaktionsID. Somit sind die Log Records einer Transaktion geordnet (durch die aufsteigende LSN) und gruppiert (durch die TransaktionsID).

Hier ein fiktiver Ausschnitt aus einem Transaktions Log.



Tabelle 14.2: Ausschnitt aus einem Transaktionsprotokoll

Operation	page id	transaction id	current lsn
LOP_BEGIN_XACT	0001:00000018	0000:00003f60	00000017:00001560:0001
LOP_INSERT_ROWS	0001:00000018	0000:00003f60	00000017:00001b40:0001
LOP_INSERT_ROWS	0001:00000067	0000:00003f60	00000017:00001b40:0002
LOP_INSERT_ROWS	0001:00000062	0000:00003f60	00000017:000022f1:0001
LOP_COMMIT_XACT	0001:00000062	0000:00003f60	00000017:000022f1:0003
LOP_BEGIN_XACT	0001:00000112	0000:00003f61	00000014:000000f3:0001
LOP MODIFY_ROW	0001:00000112	0000:00003f61	00000014:000000f3:0002
LOP_DELETE_ROWS	0001:00000112	0000:00003f61	00000014:000001a2:0001
LOP MODIFY_ROW	0001:00000113	0000:00003f61	00000014:000001a2:0004
LOP_COMMIT_XACT	0001:00000113	0000:00003f61	00000014:000001cc:0001

Das Transaktions Log zeigt Einträge aus zwei Transaktionen, deren Beginn und Ende rot bzw. blau gekennzeichnet sind. In der Spalte „transaction id“ ist die ID der jeweiligen Transaktion zu sehen. Diese ist bei allen Log Records, welche zu einer Transaktion gehören, gleich. Somit sind die Einträge gruppiert.

Betrachtet man die Spalte „current lsn“ fällt auf, dass sich die LSN innerhalb einer Transaktion aufsteigend verhält. Dadurch sind die Einträge in einer festen Reihenfolge geordnet.



Jede Transaktion besteht aus mindestens drei Log Records. Einem LOP_BEGIN_XACT-Eintrag, der den Beginn der Transaktion markiert, einem oder mehreren Einträgen, welche die Aktionen innerhalb der Transaktion darstellen (INSERT, UPDATE, DELETE, etc.) und einem LOP_COMMIT_XACT- oder einem LOP_ABORT_XACT-Eintrag der die Transaktion beendet.

Die Idee dabei ist folgende: Sollten Änderungen, z. B. wegen eines Serverabsturzes, verloren gehen, können diese unter Zuhilfenahme des Transaktionsprotokolls nachvollzogen und der Verlust damit kompensiert werden.



- [ms180892]

Virtual Log Files (VLFs)

Obwohl das Transaktionsprotokoll in der Regel aus nur einer Datei besteht, ist es intern in mehrere Bereiche, die sogenannten „Virtual Log Files“ (VLFs) gegliedert.

Abb. 14.8:
Eine Log-Datei,
unterteilt in acht
virtuelle



In Abbildung ?? ist eine Log Datei mit acht virtual log files zu sehen.

14.5.3 Virtual Log Files im Detail

Der VLF header

Genau wie eine Log Datei hat auch jedes VLF einen eigenen Headerblock mit Metadaten.

Größe und Anzahl der VLFs

Die Database Engine ist aus Performancegründen darum bemüht, die Anzahl der VLFs immer möglichst gering zu halten. Anzahl und Größe der VLFs lassen sich nach einem einfachen Schema bestimmen:

Tabelle 14.3: Größe und Anzahl von Virtual Log Files

Transaktionsprotokollgröße	Anzahl der VLFs	Größe der VLFs
$\leq 1 \text{ MB}$	Log size / 256 KB	256 KB
$> 1\text{MB} \text{ und } \leq 64 \text{ MB}$	4	Growth Junksize / 4
$> 64 \text{ MB} \text{ und } \leq 1 \text{ GB}$	8	Growth Junksize / 8
$> 1 \text{ GB}$	16	Log size / 16

Die Angaben zu diesem Berechnungsschema stammen aus [[isbn9780735658561](#)].

Würde einer neuangelegten Datenbank eine Log Datei mit einer Größe von 4 GB und einer „Growth Junksize“ von 1 GB hinzugefügt entstünden folgende VLFs:

Initial: 16 VLFs á 256 MB, da 4 GB Growth Junksize

Pro Wachstumsvorgang: 8 VLFs á 64 MB, da 1 GB Growth Junksize

Würde diese Log Datei auf 100 GB anwachsen, bestünde sie aus $16 + (192 * 8) = 1552$ (192 Wachstumsvorgänge) VLFs. Diese Zahl steht im krassen Missverhältnis zu der Aussage, dass sich der SQL Server immer darum bemüht möglichst wenige VLFs zu produzieren. Aus diesem Grund hat Microsoft mit dem SQL Server 2014 eine Neuerung bei der Berechnung der Anzahl der VLFs eingeführt:

Es wird folgende Bedingung geprüft: Ist die Growth Junksize kleiner als 1/8 der aktuellen Log Filegröße?

- **Ja:** Füge nur ein neues VLF in der Growth Junksize hinzu.
- **Nein:** Benutze das herkömmliche Berechnungsverfahren.

Das bedeutet, dass sich das obige Beispiel mit der Log Datei von 4 GB und der Growth Junksize von 1 GB wie folgt verändert:

Initial: 16 VLFs á 256 MB, da 4 GB Growth Junksize

Tabelle 14.4: Wachstumsvorgäng des Logfiles

Transaktionsprotokollgröße	1/8 der Logfile size	Growth Junksize	Anzahl VLFs
4096 MB	512 MB	1024 MB	8
5120 MB	640 MB	1024 MB	8
6144 MB	768 MB	1024 MB	8
7168 MB	896 MB	1024 MB	8
8192 MB	1024 MB	1024 MB	8
9216 MB	1152 MB	1024 MB	1

Wie aus Tabelle ?? entnommen werden kann, wächst das Logfile bei den ersten fünf Vorgängen um je 8 VLFs an, da die Growth Junksize größer ist, als 1/8 der Logfile size. Ab den sechsten Vorgäng ändert sich dies, da 1/8 von 9216 MB = 1152 MB und somit größer ist, als 1024 MB.



Wächst eine Log Datei, so geschieht dies immer durch Anfügen ganzer VLFs. Auch beim Schrumpfen einer Log Datei geht dies immer nur bis zur Grenze eines belegten VLF.



- [ms179355]
- [KLTTLIctVcaISS]

Geschicktes Anlegen einer Logdatei

Anzahl und Größe des VLFs haben einen sehr hohen Einfluss auf die Performance der gesamten Datenbank.

- Hat eine Logdatei zu viele VLFs, kann dies zu einer Fragmentierung der Logdatei führen, wodurch Lese- und Schreibperformance reduziert werden.
- Sind es zu wenige VLFs, kann es im Extremfall passieren, dass kein neues, leeres VLFs für einen Wechsel bereisteht und das die Logdatei wachsen muss.
- Sind die VLFs einer Datenbank zu klein, kommt es zu häufigen Wechseln und damit zu einem erhöhten Verwaltungsaufwand.
- Sind die VLFs zu groß, kann es zu Problemen bei den Transaktionslog-Backups kommen, da die Wechsel von VLF zu VLF zu selten geschehen.

Aus diesen Gründen sollte das Transaktionsprotokoll eine ausgewogene Anzahl VLFs mit einer sinnvollen Größe haben. Um dieses Ziel zu erreichen, muss der Administrator vor dem Anlegen der Datenbank berechnen, ob er die Logdatei komplett am Stück oder in mehreren Abschnitten erstellt. Hierzu nun einige Beispiele.

Es soll eine Datenbank, mit einer 2 GB großen Logdatei, erstellt werden.

Würde diese Logdatei am Stück erstellt werden, bestünde sie aus 16 VLFs, á 128 MB. Größe und Menge der VLFs sind bei einer so kleinen Logdatei absolut in Ordnung.

Nun soll eine sehr viel größere Datenbank, mit einer Logdatei von 40 GB, erstellt werden.

Die Erstellung der Logdatei an einem Stück würde 16 VLFs, mit einer Größe von je 2,5 GB, nach sich ziehen. Während eine Anzahl von 16 durchaus in Ordnung geht, ist eine Größe von 2,5 GB pro VLF etwas hoch. Hier könnte es angebracht sein, die Logdatei in fünf Schritten zu je 8 GB aufzubauen. Damit steigt zwar die Anzahl der VLFs sprunghaft an ($5 * 16 = 80$), dafür sinkt die Größe pro VLF auf 512 MB.

Eine andere mögliche Vorgehensweise könnte auch sein, die Logdatei in zwei Schritten, zu je 20 GB zu erstellen. Die Folge wären 32 VLFs, á 1,25 GB. Welche der beiden Vorgehensweisen nun die bessere ist, bleibt immer eine Einzelfallentscheidung.

Log Blocks

Jedes VLF besteht aus einer Anzahl von sogenannten „Log Blocks“. Diese haben, genau wie die darin enthaltenen Log Records, eine variable Größe. Ein Log Block ist im Minimum 512 B und im Maximum 60 KB groß. Wie groß ein Log Block wird, hängt vom Transaktionsverhalten der Datenbank ab: Je größer die Transaktionen in einer Datenbank sind (hier ist die Anzahl der Log Records und deren Größe gemeint), desto größer werden auch die Log Blocks.

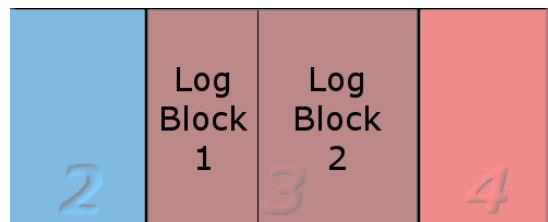


Abb. 14.9:
VLF mit zwei
Log Blocks

Abbildung ?? zeigt einen Ausschnitt aus dem Transaktionsprotokoll. Darin ist zusehen, dass das Virtual Log File nummer 3 zwei unterschiedlich große Log Blocks enthält.

Sobald ein Log Block seine Maximalgröße von 60 KB erreicht hat, wird der Log Manager diesen aus dem Log Buffer auf den Datenträger schreiben. Der Log Block im Buffer kann anschließend wiederverwendet werden.



Der SQL Server Log Manager schreibt niemals einzelne Log Records, sondern immer ganze Log Blocks auf den Datenträger.

- [MirDimSSTLP1LSaWALA]



Aktive und inaktive VLFs

Abbildung ?? zeigt eine Log Datei, welche aus acht virtuellen Log Dateien besteht. Die blau gekennzeichneten VLFs 1, 2, 7 und 8 sind unbenutzt bzw. inaktiv. Die rot gekennzeichneten VLFs 3, 4, 5 und 6 sind aktiv.

Ein VLF gilt als aktiv, wenn es mindestens einen Active Log Record enthält. Ein Log Record hat den Status active, wenn eine der folgenden Bediengungen auf ihn zutrifft:

- Ein Log Record bezieht sich auf einen offene Transaktion
- Eine von einer Änderung betroffene Data Page befindet sich noch im Data Cache (wurde noch nicht auf den Datenträger zurückübertragen).
- Ein Log Record wird noch für ein Backup benötigt.
- Ein Log Record wird von einer SQL Server Technologie, wie z. B. der Replikation benötigt.

Log Rotation

Das Transaktionsprotokoll ist intern als sogenannter „Ringpuffer“ aufgebaut, d. h. sobald ein VLF zu 100 % gefüllt ist, wird automatisch auf das nächste verfügbare (inaktive) VLF gewechselt. Dies gilt auch, falls das Ende des Transaktionsprotokolls erreicht wurde. Der SQL Server beginnt dann einfach wieder am Anfang des Transaktionsprotokolls nach einem inaktiven VLF zu suchen.

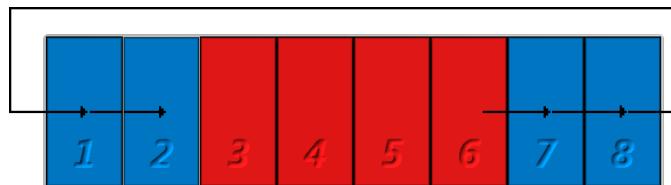


Abb. 14.10:
Rotation im
Transaktionspro-
tokoll

Sollte jedoch der Fall eintreten, dass kein VLF mit dem Status inactive mehr zur Verfügung steht, muss das Transaktionsprotokoll wachsen. Die Protokolldatei wird nach den bereits genannten Regeln vergrößert, so dass weitere VLFs zur Verfügung stehen.

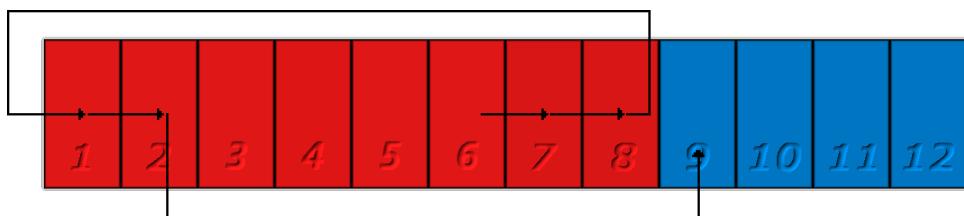


Abb. 14.11:
Die Protokolldatei
wächst an

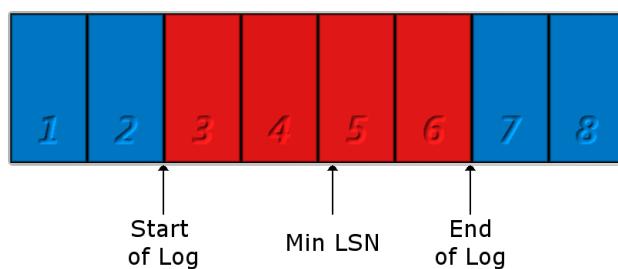
Abschneiden des Log-Protokolls (Log Truncation)

Damit ein Transaktionsprotokoll nicht ins Unendliche anwächst, muss dort in gewissen Zeitabständen Speicherplatz freigegeben werden. Dieser Mechanismus wird als „Log Truncation“ bezeichnet. Es werden dabei nur solche Einträge überschrieben, die keine direkte Relevanz mehr haben, d. h. für die keine offenen Änderungen mehr im Data Cache existieren. So wird Platz für neue Einträge geschaffen.

Abbildung ?? zeigt ein Transaktionsprotokoll, mit den folgenden Angaben:

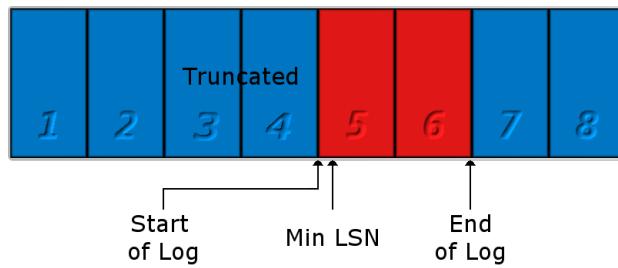
- **Start of Log:** An dieser Stelle beginnt das logische Logfile. Die VLFs 3, 4, 5 und 6 sind aktiv.
- **End of Log:** Dieser Punkt markiert das Ende des logischen Logfiles.
- **Min LSN:** An diesem Punkt innerhalb des logischen Logfiles steht der älteste Log Record, der zu einer offenen Transaktion gehört (der älteste aktive Eintrag). Das bedeutet, dass alle Einträge links der Min LSN beim nächsten Checkpoint aus dem Logfile entfernt werden können, während sich rechts der Min LSN noch weitere aktive Log Einträge befinden können. Dieser Punkt ist wichtig für das Recovery einer SQL Server Datenbank.

Abb. 14.12:
Ein Protokoll vor
dem Abschneiden



Beim Eintreten des nächsten Checkpoints werden nun alle Einträge zwischen den Punkten „Start of Log“ und „Min LSN“ aus dem logischen Logfiles abgeschnitten. Somit sind die beiden VLFs 3 und 4 wieder frei und werden als „inaktiv“ markiert.

Abb. 14.13:
Ein Protokoll
nach dem
Abschneiden



Log Truncation kann auf zwei unterschiedliche Arten geschehen:

- Durch automatisches, periodisches Abschneiden des Protokolls
- Durch ein sogenanntes „Transaktionsprotokoll-Backup“

Beim automatischen, periodischen Löschen werden nicht mehr relevante Einträge einfach gelöscht. Diese sind im Anschluss daran nicht mehr nachvollziehbar.

Mit einem Transaktionsprotokoll-Backup werden alle Einträge im Log gesichert und anschließend aus dem Protokoll entfernt. Im Falle dessen, dass der Administration ein Recovery der Datenbank durchführen muss, sind alle Einträge aus den Backups wiederherstellbar.



Der Mechanismus der Log Truncation sorgt nicht dafür, dass die Transaktionsprotokoll-Datei physikalisch kleiner wird. Es wird nur Speicherplatz innerhalb der Datei zur erneuten Benutzung freigegeben. Um die Protokoll-Datei zu schrumpfen, muss der Administration manuell eingreifen.

Es kann vorkommen, dass das Arbeitsvolumen einer Instanz so hoch ist, dass noch keine Bereiche existieren, die wiederverwendet werden könnten, weil alle Informationen noch gebraucht werden. In einem solchen Fall versucht der SQL Server das Log File zu vergrößern. Falls dies aber nicht gelingt, wird der Systemfehler 9002 ausgegeben.



- [ms190925]

14.5.4 Write-Ahead-Logging

Wie bereits erwähnt wurde, müssen alle Änderungen, die im Data Cache durchgeführt werden, zuerst im Log Cache protokolliert werden. Um dies zu gewährleisten, kommt ein Verfahren zum Einsatz, welches als „Write-Ahead-Logging“ bezeichnet wird. Dieser Mechanismus wird intern umgesetzt, in dem der Buffer Manager im Header eines jeden Buffers die LSN der letzten Änderung verzeichnet. Des Weiteren ist die LSN des letzten Log-Eintrags im Log Buffer für alle Hintergrundprozesse zugänglich. Somit kann gewährleistet werden, dass der Lazy Writer nur solche Buffer auf den Datenträger schreibt, deren Änderungen bereits gelogged wurden.

Hier ein Beispiel dazu:

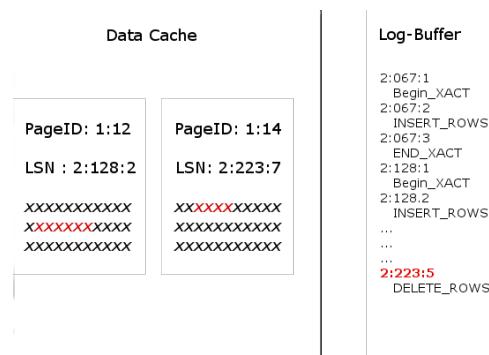


Abb. 14.14:
Write-Ahead-
Logging und die
LSN

Der Buffer mit der PageID 1:12 wurde zuletzt geändert, als die LSN 2:128:2 gültig war. Da die aktuelle LSN im Log-Buffer die 2:223:5 ist, kann dieser Block auf den Datenträger geschrieben werden (es gilt: 2:128:2 < 2:223:5).

Der rechte Buffer (PageID 1:14) trägt die LSN 2:223:7. Da diese LSN größer ist, als 2:223:5, bedeutet dies, dass seine Änderung noch nicht protokolliert wurde. Dieser Buffer darf also noch nicht auf den Datenträger geschrieben werden.

Damit dieses System Schutz vor Datenverlust gewährleisten kann, muss eine wichtige Regel eingehalten werden: Sobald der User ein **COMMIT**-Statement absetzt und somit seine Transaktion beendet, muss ein synchroner Schreibvorgang des Log-Buffers auf den Datenträger erfolgen. Synchron bedeutet in diesem Fall, dass der User gezwungen wird zu warten, bis der Schreibvorgang beendet ist. Erst dann darf er weiter arbeiten. So wird sichergestellt, dass alle Log-Einträge, die zu einer Transaktion gehören, nach einem **COMMIT**, in jedem Fall im Transaktionsprotokoll verfügbar sind. Sollten nun die Änderungen im Data Cache verloren gehen, können diese aus den Log-Einträgen im Transaktionsprotokoll rekonstruiert werden.

15 Installation des SQL Server 2014

Inhaltsangabe

Mit dem Betrieb eines Windows Dienstes ergeben sich in der Praxis verschiedene administrative Probleme. Jeder Dienst benötigt ein Dienstkonto, welches in manchen Fällen weitreichende Berechtigungen haben muss. Um den Server sicher zu gestalten, muss sich der Administrator unter anderem darum kümmern, dass das Konto ein sicheres Passwort besitzt, welches in regelmäßigen Zeitabständen gewechselt werden sollte.

In der Vergangenheit wurde dieses Problem häufig dadurch gelöst, dass für alle Dienste eines Servers die lokalen Konten LOKALER DIENST, NETZWERKDIENST oder LOKALES SYSTEM genutzt wurden. Ein anderer Lösungsansatz bestand darin, normale Domänenkonten mit den passenden Berechtigungen auszustatten, um so eine Verwaltbarkeit der Dienstkontakte in der Domäne zu erreichen.

Mit Windows Server 2008 R2 / Windows 7 führte Microsoft zwei neue Kontoarten ein, die zur Entlastung der Administratoren und zur Verbesserung der Sicherheit dienen sollten. Es handelt sich hierbei um die „Managed Service Accounts“ (MSA) und die „Virtual Accounts“.

15.1 Managed Service Accounts

Managed Service Accounts sind genau das, was ihr Name verspricht: Automatisch verwaltete Domänenaccounts, speziell für den Betrieb von Windows Diensten.

15.1.1 Fähigkeiten

Um die gesetzten Ziele für Verwaltbarkeit und Sicherheit erreichen zu können, sind dieser Kontoart eine ganze Reihe neuer Fähigkeiten verliehen worden. Im Einzelnen sind dies:

- Automatische Passwortverwaltung
- Vereinfachte Verwaltung von SPNs
- Direkte Verknüpfung mit genau einem Computerkonto
- Statische Bindung an genau einen Zielserver
- Delegation der SPN-Verwaltung an andere Administratoren

Gerade die ersten beiden Punkte stellen eine wesentliche Arbeitserleichterung für den Administrator dar. An seiner statt übernimmt das Active Directory die Kennwortverwaltung für alle MSAs. Es gelten dabei folgende Regeln:

- Einem MSA wird ein stark verschlüsseltes Passwort mit einer Länge von 120 Zeichen zugewiesen. Diese Passwörter bestehen aus Buchstaben, Ziffern und Sonderzeichen.
- Das Kennwort wird automatisch alle 30 Tage geändert. Dieses Intervall kann durch den Administrator beeinflusst werden.

Der Administrator könnte zwar eine manuelle Passwortänderung veranlassen, jedoch ist ein Eingreifen seinerseits, im Normalfall, nicht erforderlich. Durch die Verknüpfung eines MSA mit einem Computerkonto wird erreicht, dass das MSA als Zwidder aus Benutzer- und Computerkonto erscheint. Dadurch vereinte es die Vorteile beider Kontoarten.

15.1.2 Nachteile

Nach einem kurzen Blick hinter die Kulissen fallen aber sofort zwei ganz bedeutende Nachteile auf:

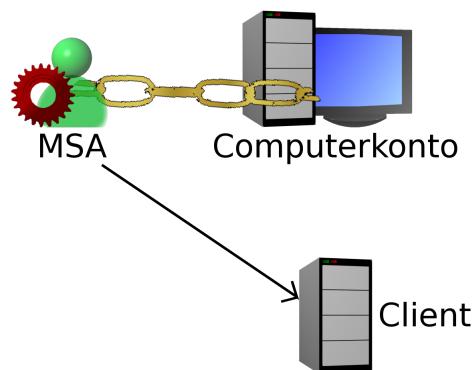
- Statische Bindung an genau einen Zielserver, was eine Nutzung auf einem geclusterten System unmöglich macht und
- Dienste wie Exchange und SQL Server werden nicht unterstützt.



Ein Nachteil, der sich aus der statischen Bindung zwischen MSA und Zielserver ergibt ist der, dass MSAs auf geclusterten Systemen nicht eingesetzt werden können, da dort ein Dienst wechselweise auf mehreren Clusterknoten betrieben werden kann.

Abb. 15.1:

MSA -
Computerkonto -
Client



Mit Windows Server 2012 kommt jedoch die Erlösung für alle Admins. Die erweiterte Fassung der Managed Service Accounts, die „Group Managed Service Accounts“, kurz „gMSA“. Diese bieten

nun alle Vorteile der MSAs, inklusive der Tatsache, dass die gerade genannten Nachteile überwunden sind.



Um im Folgenden MSAs und gMSAs besser unterscheiden zu können, werden MSAs als „standalone Managed Service Accounts“, kurz sMSA, bezeichnet!

15.2 Group Managed Service Accounts

15.2.1 Grundvoraussetzungen

Um gMSAs nutzen zu können, müssen Clients mit Windows XP oder höher betrieben werden, und es muss sich mindestens ein Domänencontroller mit Windows Server 2012 in der Domänengesamtstruktur befinden. Außerdem wird ein Windows Server 2012 oder Windows 8 Rechner mit dem „Active Directory-Modul für Windows Powershell“ benötigt.



Wird Windows 7 als Clientbetriebssystem eingesetzt, sollte das Hotfix KB2494158 „Managed service account authentication fails after its password is changed“ installiert werden!

Sollte die Domäne noch auf dem Windows Server 2003 / 2008 Level betrieben werden, muss der Administrator einige zusätzliche Konfigurationsschritte ausführen.

1. Ausführen von adprep /forestprep auf Forest-Ebene.
2. Ausführen von adprep /domainprep in jeder Domäne, welche mit MSAs umgehen können muss.
3. Ausrollen eines Domaincontrollers mit Windows Server 2012.



In dieser Unterrichtsunterlage wird davon ausgegangen, dass als Betriebssystem „Windows Server 2012 R2“ zum Einsatz kommt!

Um die Verwaltung von gMSAs zu ermöglichen, müssen drei Voraussetzungen auf dem betreffenden Admin-PC bzw. in der Domäne getroffen werden:

- Installation des .NET Framework 3.5.1
- Installation des „Active Directory-Modul für Windows Powershell“
- Erstellung eines „KDS¹ Root Key“ in der Domäne

Installation des .NET Framework 3.5.1

Installieren Sie das .NET Framework 3.5.1 und das Active Directory-Modul für Windows Powershell wie folgt:

1. Legen Sie die Windows Server 2012 - DVD ins Laufwerk ein!
2. Starten Sie den Server Manager!
3. Klicken Sie im Menü „Verwalten“ auf „Rollen und Features hinzufügen“! Es öffnet sich der „Assistent zum Hinzufügen von Rollen und Features“.
4. Wählen Sie im Fenster „Installationstyp auswählen“ die Option „Rollenbasierte oder featurebasierte Installation“!
5. Im Dialogfenster „Zielserver auswählen“ wählen Sie den betreffenden Server aus!
6. Überspringen Sie den Dialog „Serverrollen auswählen“!
7. Öffnen Sie im Dialogfenster „Features auswählen“ die Option „.NET Framework 3.5-Funktionen“ und wählen Sie das „.NET Framework 3.5“ aus!
8. Öffnen Sie die Option „Remoteserver-Verwaltungstools“, „Rollenverwaltungstools“, „AD DS- und AD LDS-Tools“ und wählen Sie „Active Directory-Modul für Windows Powershell“. Klicken Sie auf „Weiter“!
9. Sie werden nun aufgefordert, einen alternativen Quellspfad für das .NET Framework anzugeben, da sich das .NET-Installationspaket noch nicht auf Ihrem Rechner befindet. Klicken Sie unten links auf den Link „Alternativen Quellpfad angeben“.

¹KDS = Key Distribution Service (neu in Windows Server 2012)

Abb. 15.2:
Assistent zum Hinzufügen von Rollen und Features - 1

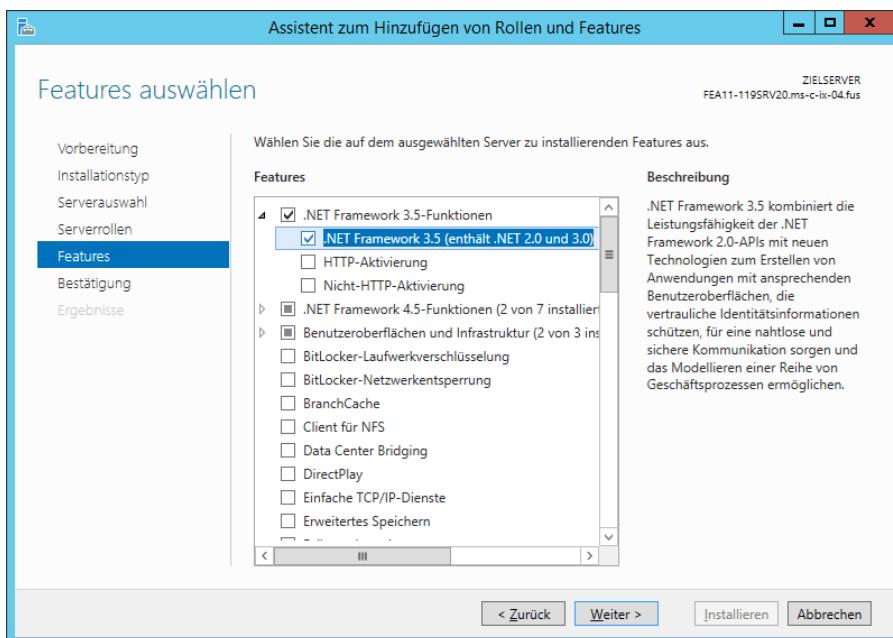


Abb. 15.3:
Assistent zum Hinzufügen von Rollen und Features - 2

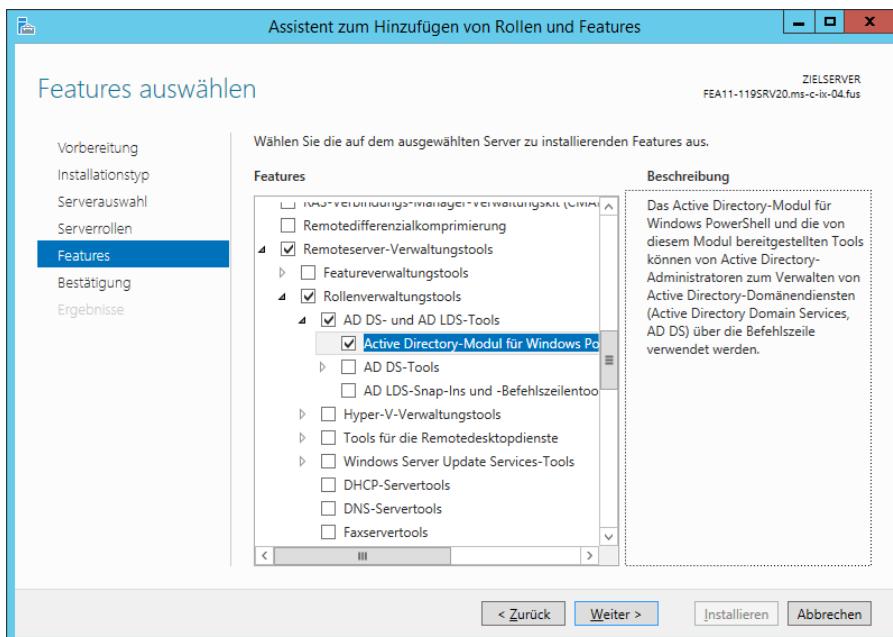
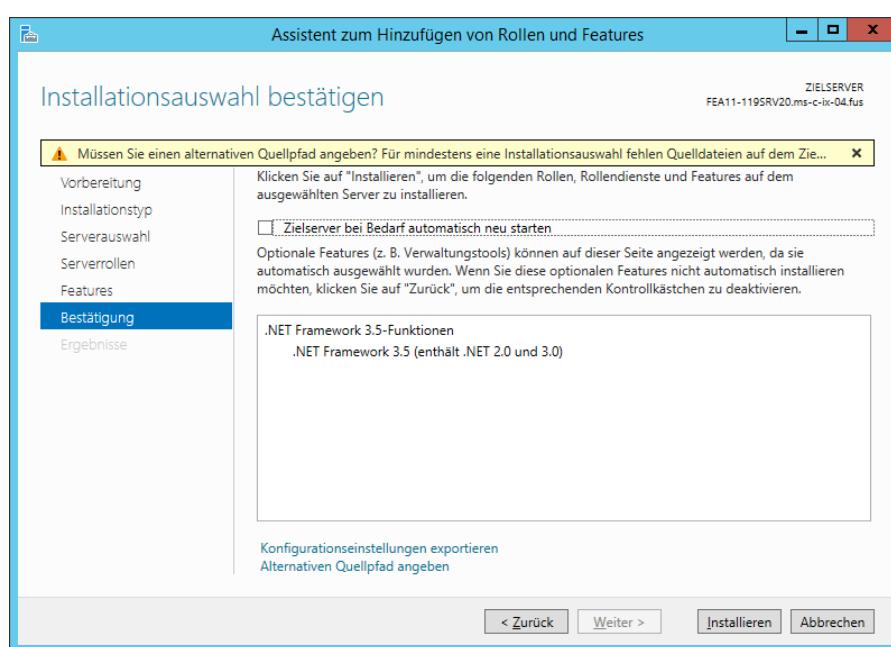


Abb. 15.4:
Assistent zum
Hinzufügen von
Rollen und
Features - 3



10. Geben Sie folgenden Pfad im soeben erscheinenden Dialogfenster an:

<DVD-Laufwerk>\sources\sxs

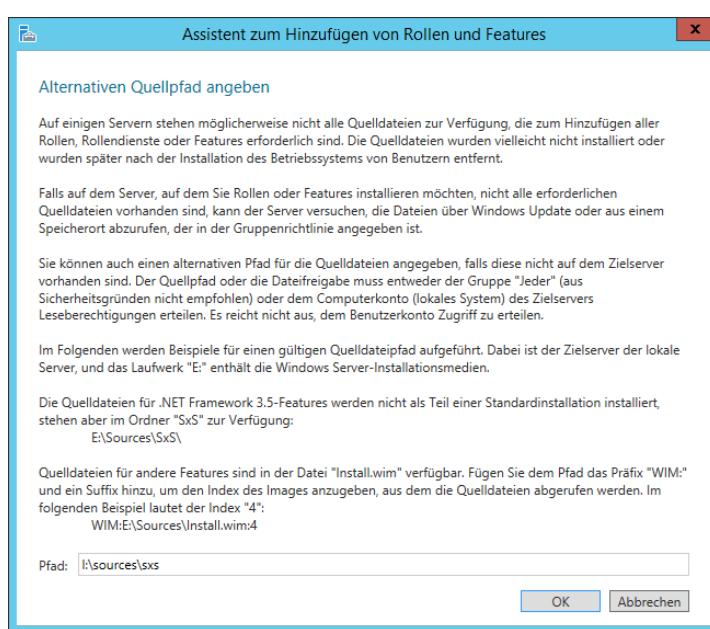


Abb. 15.5:
Assistent zum
Hinzufügen von
Rollen und
Features - 4

11. Klicken Sie auf „OK“!

12. Klicken Sie auf „Installieren“ um den Installationsvorgang zu starten

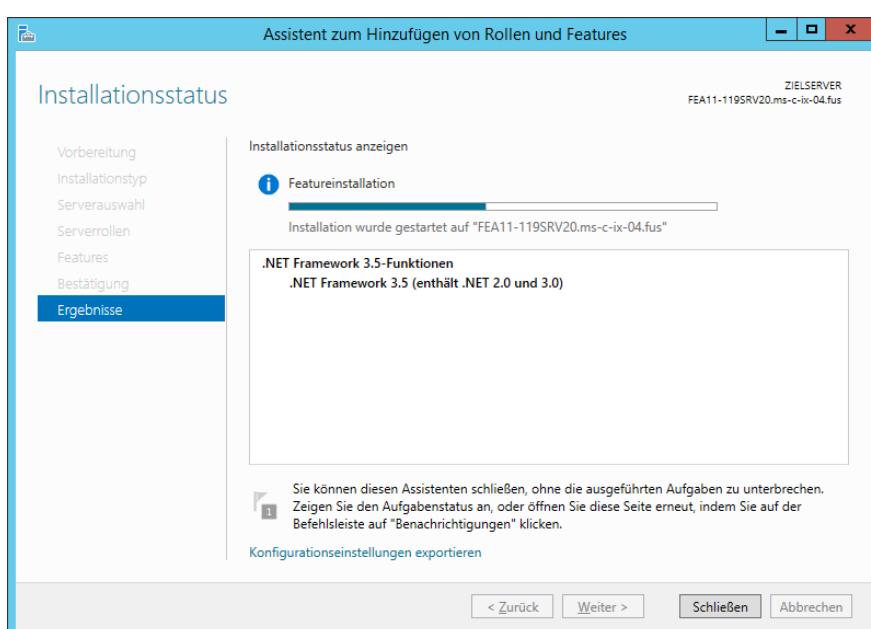


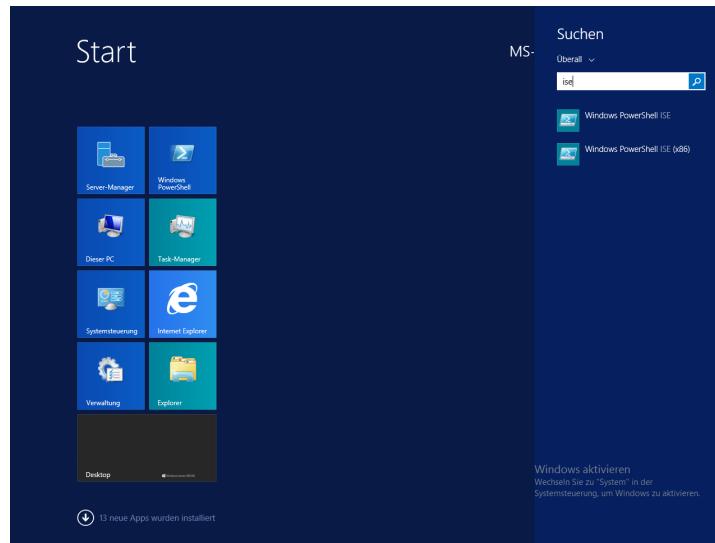
Abb. 15.6:
Assistent zum
Hinzufügen von
Rollen und
Features - 5

Bereitstellung des KDS Root Key

Die Bereitstellung des KDS Root Key erfolgt nun mittels der Powershell. Voraussetzung für die folgende Prozedur ist die Mitgliedschaft in der Gruppe der Domänen-Admins bzw. Enterprise-Admins.

1. Starten Sie die Powershell ISE, indem Sie auf den Windows-Button klicken und als Suchbegriff „ISE“ eingeben! Es erscheint am rechten Bildschirmrand eine Auflistung der Suchergebnisse.

Abb. 15.7:
Starten der
Powershell ISE



2. Erstellen Sie den KDS Root Key.

Listing 15.1: Erstellen eines KDS Root Keys

```
Add-KDSRootKey -EffectiveImmediately
```



Wurde der KDS Root key auf diese Art und Weise erzeugt, muss erst eine Sicherheitsfrist von 10 Stunden verstreichen, bevor ein gMSA genutzt werden kann.

Diese Frist existiert, damit in einer Umgebung mit mehreren Domänen-Controllern sichergestellt wird, dass wirklich alle DCs den KDS Root Key repliziert bekommen, bevor ein erster gMSA genutzt wird. Andernfalls könnte es passieren, dass ein Domänen-Controller die Passwortanfrage eines gMSA nicht beantworten und somit ein Dienst nicht starten kann.

Um diese Frist in einer Domäne mit nur einem DC zu umgehen, muss das `Add-KDSRootKey`-Commandlet mit einer zusätzlichen Angabe ausgeführt werden:

Listing 15.2: Erstellen eines KDS Root Keys

```
Add-KDSRootKey -EffectiveTime ((get-date).addhours(-10))
```



- [jj128430]

15.2.2 Kurzer Ausflug: AGDLP

„AGDLP“ - Account, Global Group, Domain local Group, Permission - bezeichnet ein Verfahren für die rollenbasierte Zuteilung von Berechtigungen (RBAC²). Seine Aufgabe ist es, für ein strukturiertes Rollen-Rechte-Konzept in einer Active Directory Domäne oder sogar in einem Forest zu sorgen.

Hinter der Abkürzung AGDLP steckt im Wesentlichen die Idee, dass Berechtigungen nicht einem einzelnen Objekt, z. B. einem Nutzer zugeteilt werden, sondern einer Gruppe, deren Mitglieder dann die Berechtigungen erben. In einer Active Directory Domäne existieren für diesen Zweck zwei unterschiedliche Gruppenarten:

- **Globale Gruppen:** Mit Hilfe dieser Gruppen werden Rollen oder Tätigkeiten abgebildet (z. B. DB-Admin, AccountManager usw.). In diesen Gruppen dürfen Nutzer und andere globale Gruppen der gleichen Domäne Mitglied sein und sie können von Ressourcen anderer Domänen genutzt werden.
- **Domänen-lokale Gruppen:** Diese Gruppenart ist es, die die Berechtigungen zugewiesen bekommt. Es können universelle, globale und lokale Gruppen Mitglied sein. Auch Nutzerkonten sind als Mitglieder erlaubt. Domänen-lokale Gruppen können nur innerhalb der eigenen Domäne genutzt werden.

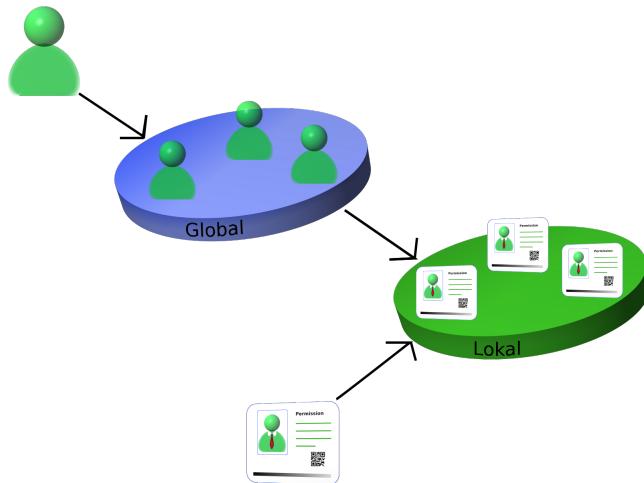
Gemäß der Empfehlung von Microsoft sollte AGDLP wie folgt umgesetzt werden:

- Zuerst wird eine globale Gruppe erstellt.
- Alle betreffenden Accounts werden Mitglieder der globalen Gruppe.
- Es wird eine domänen-lokale Gruppe erstellt. Diese erhält alle vorgesehenen Berechtigungen.
- Die globale Gruppe wird Mitglied der domänen-lokalen Gruppe.



- [AGDLP]

²RBAC = Role-Based Access Controls

Abb. 15.8:
AGDLP

15.2.3 Einen Managed Service Account erstellen



Um einen MSA erstellen zu können muss der Benutzer

- der Domänen-Administrator sein oder
- ein Mitglied der Gruppe der Domänen-Admins sein oder
- ein Mitglied der Gruppe der Konten-Operatoren sein oder
- er muss die „Create/DeleteMS-ManagedServiceAccount“ Berechtigung haben.

1. Starten Sie die Powershell ISE

2. Nachdem die ISE gestartet hat, importieren Sie zuerst das Active Directory-Modul mit dem Kommando:

Listing 15.3: Importieren des Active Directory-Moduls für Windows Powershell

```
Import-Module ActiveDirectory
```

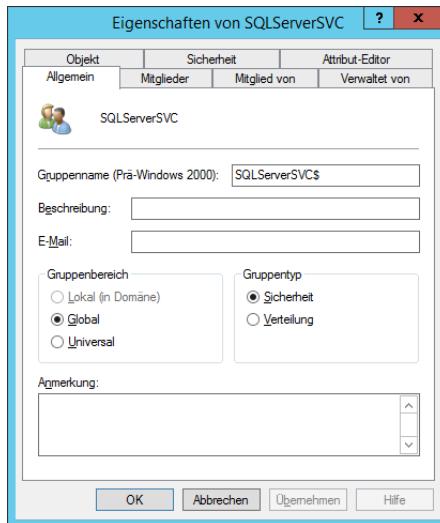
3. Erstellen Sie eine neue Sicherheitsgruppe im Active Directory mit Hilfe des Commandlets `New-ADGroup`.
Dieser Schritt kann auch mittels der MMC durchgeführt werden.

Listing 15.4: Eine globale Sicherheitsgruppe erstellen

```
New-ADGroup -Name SQLServerSVC -GroupScope Global '  
-GroupCategory Security -SamAccountName SQLServerSVC$ '  
-Path "OU=Groups, OU=MS-C-IX-04-20, DC=MS-C-IX-04, DC=FUS"
```

4. Fügen Sie Ihr Computerkonto der Gruppe SQLSERVERSVC als Mitglied hinzu. Dieser Schritt kann mittels Powershell oder mit Hilfe der Microsoft Management Konsole geschehen.

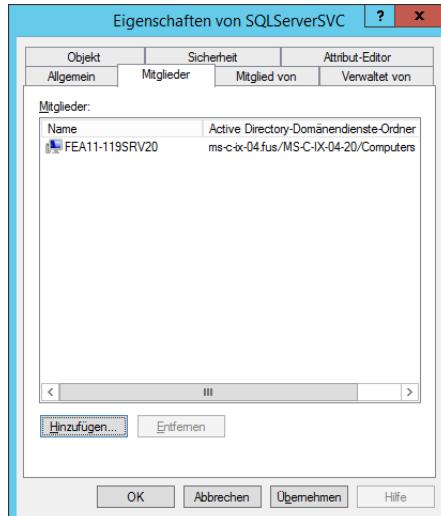
Abb. 15.9:
Die Gruppe
„SQLServerSVC“



Listing 15.5: Einer Gruppe ein Mitglied hinzufügen

```
Add-ADGroupMember `n
-Identity `n
"CN=SQLServerSVC,OU=Groups,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS" `n
-Members `n
"CN=FEA11-119SRV20,OU=Computers,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS"
```

Abb. 15.10:
Gruppen-
mitglieder von
SQLServerSVC



5. Benutzen Sie das Commandlet [New-ADServiceAccount](#) um einen neuen MSA zu erstellen.

Listing 15.6: Erstellen des MSA

```
New-ADServiceAccount -Name MSSQLServer20 -SamAccountName MSSQLServer20$ `n
-DNSHostname FEA11-119SRVAD.MS-C-IX-04.FUS `n
-Path "OU=MSA,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS" `n
-PrincipalsAllowedToRetrieveManagedPassword `n
"CN=SQLServerSVC,OU=Groups,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS" `n
-Enabled $true
```

Das Commandlet `New-ADServiceAccount` nimmt aktuell ca. 20 verschiedene Parameter entgegen. In Beispiel ?? wurden nur die wichtigsten benutzt. Welche Bedeutung sie haben wird nachfolgend erläutert:

- `-Name`: Setzt das NAME-Attribute des Accounts im Active Directory
- `-SamAccountName`: Gibt den Security Account Managed Name des gMSA an. Dieser darf nicht länger als 20 Zeichen sein, um die Kompatibilität zu älteren Systemen zu wahren. Wird am Ende des SamAccountName kein \$ angegeben, hängt Windows automatisch eines an.
- `-DNSHostname`: Gibt den Namen eines DNS-Servers an. Dieser Parameter wird benötigt, um den Domänen-Controller auffinden zu können.
- `-Path`: Nimmt einen X.500 konformen Pfad entgegen, mit dessen Hilfe angegeben wird, wo im AD der gMSA erstellt werden soll.
- `-PrincipalsAllowedToRetrieveManagedPassword`: Setzt das „msDS-GroupMSAMembership“-Attribut und gibt damit an, welche Computer den gMSA nutzen können.
- `-Enabled`: Gibt an, ob der gMSA aktiviert oder deaktiviert ist. Es sind nur die beiden Werte `$true` und `$false` zulässig.

In Beispiel ?? wird für den Parameter `-PrincipalsAllowedToRetrieveManagedPassword` die Gruppe SQL-SERVERSVC benutzt. Alle Computerkonten, die dieser Gruppe als Mitglieder zugewiesen werden haben dadurch Zugriff auf den gMSA MSSQLSERVER20. Sollte aus irgend einem Grund nicht mehr bekannt sein, welche Benutzer/Gruppen berechtigt sind, dass gMSA zu nutzen (`-PrincipalsAllowedToRetrieveManagedPassword`), kann dies mittels des Commandlets `Get-ADServiceAccount` nachgeschlagen werden.

Listing 15.7: Einer Gruppe ein Mitglied hinzufügen

```
Get-ADServiceAccount '  
-Identity "CN=MSSQLServer20,OU=MSA,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS" '  
-properties principalsallowedtoretrievemanagedpassword
```



- [hh852236]
- [ee617258]
- [askpfeplat20121217]

15.2.4 (De-)Installation eines gMSA auf einem Zielserver

Installation eines gMSA auf dem Zielserver

Die Installation eines gMSA auf dem Zielrechner ist kein zwingend notwendiger Schritt. Warum sollte sie dennoch erfolgen? Durch die Installation wird der betroffene Server ermächtigt, die periodische Passwortänderung für den gMSA durchzuführen. Ohne Installation müsste dieser entscheidende Punkt manuell durchgeführt werden.



Für die Installation des MSA auf einem Clientcomputer muss der Benutzer ein Mitglied der Gruppe der lokalen Administratoren sein.

Installiert wird der gMSA mit dem Commandlet [Install-ADServiceAccount](#).

1. Booten Sie Ihren Windows Server neu!
2. Starten Sie die Powershell ISE im Administratormodus (Als Administrator ausführen)!
3. Führen Sie das genannte Commandlet aus!

Listing 15.8: Einen MSA auf dem Zielserver installieren

```
Install-ADServiceAccount '  
-Identity "CN=MSSQLServer20 ,OU=MSA ,OU=MS-C-IX-04-20 ,DC=MS-C-IX-04 ,DC=FUS"
```

4. Testen Sie, ob die Installation erfolgreich war!

Listing 15.9: Prüfen des Installationserfolges

```
Test-ADServiceAccount '  
-Identity "CN=MSSQLServer20 ,OU=MSA ,OU=MS-C-IX-04-20 ,DC=MS-C-IX-04 ,DC=FUS"
```

Sollte die Installation erfolgreich gewesen sein, liefert das Commandlet [Test-AdServiceAccount](#) den Wert `$true` zurück. Andernfalls wird eine detaillierte Fehlermeldung ausgegeben.



Damit dieser Vorgang erfolgreich sein kann, muss die ISE zwingend im Administratormodus ausgeführt werden!



- [ee617223]

Deinstallation eines gMSA vom Zielserver

Deinstalliert wird ein gMSA mit dem `Uninstall-ADServiceAccount` Commandlet.



Auch hier muss die ISE zwingend im Administratormodus (Als Administrator ausführen) gestartet werden!

1. Starten Sie die Powershell ISE im Administratormodus (Als Administrator ausführen)!
2. Führen Sie das genannte Commandlet aus!

Listing 15.10: Einen MSA vom Zielserver deinstallieren

```
Uninstall-ADServiceAccount '  
-Identity "CN=MSSQLServer20,OU=MSA,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS"
```

3. Testen Sie, ob die Deinstallation erfolgreich war!

Listing 15.11: Prüfen des Deinstallationserfolges

```
Test-ADServiceAccount '  
-Identity "CN=MSSQLServer20,OU=MSA,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS"
```



- [ee617202]

Zurücksetzen des Passworts

Das Passwort eines gMSA kann mit Hilfe des Commandlets `Reset-ADServiceAccountPassword` geschehen.

Listing 15.12: Das Passwort eines MSA manuell zurücksetzen

```
Reset-ADServiceAccountPassword '  
"CN=MSSQLServer20,OU=MSA,OU=MS-C-IX-04-20,DC=MS-C-IX-04,DC=FUS"
```



- [dd391923]
- [jj128431]
- [sqlosteam20140219]
- [ee617202]

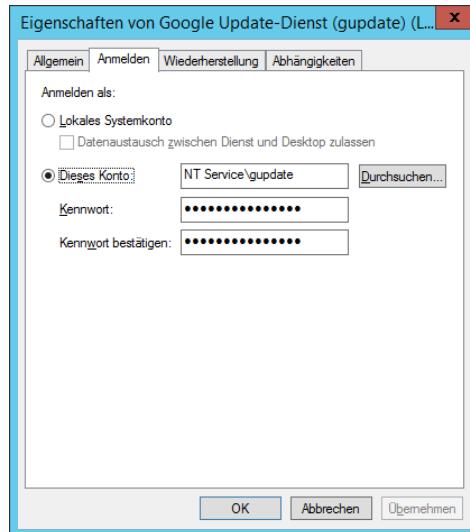
15.3 Virtual Accounts

Virtual Accounts sind das lokale Gegenstück zu den MSAs. Sie existieren nur auf dem lokalen Rechner und erfordern wenig/keinen administrativen Aufwand. Für diese Kontoart gelten die folgenden Punkte:

- Der Name eines virtuellen Kontos beginnt immer mit „NT Service“
- Es existiert, ohne dass es vorher angelegt werden muss.
- Die Kennwortverwaltung läuft automatisch ab, wie bei den MSAs
- Es erscheint nicht in der MMC „Lokale Benutzer und Gruppen“
- Es kann in jeder ACL mit aufgeführt werden
- Benötigt es Zugriff auf externe Ressourcen im Netzwerk, geschieht dies mit den Berechtigungen des Computerkontos des eigenen Servers.

Virtual Accounts sind somit ideal für einfache Dienste, die hauptsächlich Zugriff auf lokale Ressourcen benötigen. Erstellen kann man sie, indem man in den Eigenschaften eines Dienstes einfach einen neuen Benutzernamen, z. B. NT SERVICE\SQLSERVER einträgt. Der Benutzer muss nicht vorher angelegt werden.

Abb. 15.11:
Ein virtueller
Account für
Google Update



- [vafduw78as2008r2]
- [msavvaiws2008r2]

15.4 Installation des SQL Server 2014

15.4.1 Systemvoraussetzungen

Eine vollständige Liste aller Hardware- und Softwareanforderungen für SQL Server 2014 kann dem Microsoft Artikel [[ms143506](#)] entnommen werden. An dieser Stelle erfolgt lediglich eine kurze Auflistung der notwendigsten Informationen.

Softwareanforderungen

Die in der folgenden Auflistung gezeigten Softwareprodukte müssen zwingend vor der Installation des SQL Server 2014 auf dem System vorhanden sein, andernfalls kann der Installationsvorgang nicht begonnen werden:

- **.NET Framework 3.5.1:** Wird der SQL Server auf einem Computer mit Windows Vista SP2 oder Windows Server 2008 SP2 installiert, muss das .NET Framework 3.5.1 aus dem Internet heruntergeladen und installiert werden. Bei Windows Server 2008 R2 SP1 ist das .NET Framework bereits vorhanden und muss nur noch aktiviert werden.
- **.NET Framework 4.0:** Die Version 4.0 des .NET Frameworks wird automatisch während des SQL Server-Setup Vorganges installiert. Ein Eingreifen des Administrators ist nicht notwendig.
- **Windows Powershell:** Die Windows Powershell 2.0 wird von der Database Engine und verschiedenen anderen Komponenten benötigt, weshalb auch sie im Vorfeld aktiviert werden muss.

Hardwareanforderungen

Die folgende Liste zeigt die Hardwareanforderungen, die der Server, auf dem Microsoft SQL Server 2014 installiert wird, erfüllen sollte. Es werden immer die Mindestanforderungen sowie die optimalen Bedingungen angegeben.

- **CPU:** Die Mindesttaktrate des Prozessors beträgt 1.4 GHz. Microsoft empfiehlt aber eine Taktrate von 2.0 GHz oder höher, da die performante Ausführung von SQL-Anweisungen sehr viel Leistung beansprucht.

- **Prozessortyp:** Idealerweise hat der SQL Server einen oder mehrere Prozessoren der Typen AMD Opteron, AMD Athlon 64, Intel Xeon mit Intel EM64T Unterstützung oder Intel Pentium IV mit EM64T Unterstützung zur Verfügung.
- **Speicher:** Es sollte ein Minimum von 1 GB vorhanden sein, empfohlen werden jedoch 4 GB oder mehr



Zu Beachten ist, dass sich alle hier angegebenen Hardwareanforderungen auf 64-Bit Systeme beziehen.
Bei der Installation auf einem 32-Bit System können die Hardwareanforderungen stark abweichen!

Anforderungen an den Datenträger

Während der Installation legt der SQL Server 2014 ca. 6 GB an temporären Daten an. Des Weiteren benötigen die einzelnen SQL Server Komponenten Speicherplatz, der in der folgenden Liste aufgeführt ist:

- Database Engine, Replikation, Volltextsuche und Data Quality Services: **811 MB**
- Analysis Services: **345 MB**
- Reporting Services und Berichts- Manager: **304 MB**
- Integration Services: **591 MB**
- Master Data Services: **243 MB**
- Clientkomponenten ohne SQL Server Onlinedokumentation: **1823 MB**
- SQL Server Onlinedokumentationstool: **375 KB**



Microsoft empfiehlt, dass die Installation des SQL Server niemals auf einem Domänen-Controller erfolgen sollte, da dies eine ganze Reihe von Einschränkungen mit sich bringt!



- [ms143506]

15.4.2 SQL Server-Instanzen

Unter einer SQL Server-Instanz wird einen Windows Dienst verstanden, der die Fähigkeiten des SQL Server zur Verfügung stellt. Es gibt zwei unterschiedliche Arten von SQL Server-Instanzen.

Standardinstanz

In den meisten Situationen wird nur eine Instanz des SQL Servers auf einem Computer benötigt. Diese Instanz wird als Standardinstanz bezeichnet und ist an ihrem festgelegten Namen MSSQLSERVER zu erkennen.

Benannte Instanz

Sobald mehrere Instanzen des SQL Servers auf einem Rechner gefordert sind, müssen diese alle einen eigenen Instanznamen aufweisen. Deshalb werden diese Zusätzlichen Instanzen als benannten Instanzen bezeichnet. Für die Namensvergabe bei einer benannten Instanz ist folgendes zu beachten:

- Der Instanzname ist nicht case-sensitiv (Groß-/Kleinschreibung wird nicht berücksichtigt)
- Es dürfen keine reservierten Worte in einem Instanznamen vorkommen, wie zum Beispiel DEFAULT oder MSSQLSERVER. Dies würde zu einem Fehler führen.
- Die maximale Namenslänge beträgt 16 Zeichen.
- Instanznamen müssen mit einem Buchstaben (a - z oder A - Z) bzw. einem Unterstrich beginnen.
- Ab dem zweiten Zeichen dürfen auch die Ziffern 0 - 9, das Dollarzeichen (\$) und der Unterstrich verwendet werden.
- Grundsätzlich unzulässige Zeichen sind : Leerzeichen, Backslash \, Komma , , Semikolon ; , Doppelpunkt : , Hochkomma ' , kaufmännisches Und-Zeichen &, Hashtag # und das At-Zeichen @.

15.4.3 Die Verzeichnisse des SQL Servers

Bei der Installation eines Microsoft SQL Server 2014 werden eine ganze Reihe verschiedener Verzeichnisse angelegt. Im Wesentlichen hängt dies damit zusammen, dass jede SQL Server-Instanz eigene

Komponenten mitbringt, die von den Komponenten anderer Instanzen getrennt gespeichert werden müssen.

Instanzspezifische Komponenten

Um die instanzspezifischen Komponenten in unterschiedliche Verzeichnisse zu trennen, werden unterschiedliche Instanz-IDs für alle Komponenten erzeugt. Zum Beispiel hat die Database Engine einer Standardinstanz eines SQL Server 2014 standardmäßig die Instanz-ID `MSSQL12.MSSQLSERVER`. Dieser Wert setzt sich wie folgt zusammen:

- **MSSQL**: Fixe Zeichenfolge
- **12**: Die Versionsnummer des SQL Server 2014. Diese wird ggf. von einem Unterstrich und einer Nebenversionsnummer gefolgt.
- **MSSQLSERVER**: Der Instanzname, der getrennt durch einen Punkt, an die anderen Angaben angehängt wird.

Andere Beispiele für Instanz-IDs sind `MSAS12.MSSQLSERVER` für eine Analysis Services Instanz oder `MSSQL12.CRM` für eine benannte Instanz mit dem Namen CRM.

Die Instanz-ID wird dann im Installationspfad verwendet, um die einzelnen Komponenten zu trennen, z. B. wird die Database Engine einer SQL Server 2014 Instanz in das Verzeichnis C:\Program Files\Microsoft SQL Server\MSSQL12.MSSQLSERVER installiert.



- Das Stammverzeichnis C:\Program Files\Microsoft SQL Server darf nicht auf einem Wechseldatenträger, einem komprimierten Dateisystem oder einem Verzeichnis mit Systemdateien liegen. Des Weiteren ist die Ablage in einem freigegebenen Verzeichnis eines Failoverclusters ebenfalls nicht erlaubt.
- Seit SQL Server 2012 darf das Stammverzeichnis auf einer Windows Freigabe liegen.
- Während einer Installation kann eine benutzerdefinierte Instanz-ID angegeben werden. Diese sollte jedoch keine Sonderzeichen oder reservierte Wörter enthalten.

Eine vollständige Liste alle Verzeichnisse, die während der Installation eines Microsoft SQL Server angelegt werden, kann im den folgenden Technet Artikel abgerufen werden.



- [ms143547]

Gemeinsam genutzte SQL Server Komponenten

Unter gemeinsam genutzten Komponenten versteht man Teile der SQL Server Software, die von allen auf einem Server installierten Instanzen genutzt werden können und dabei nur einmal vorhanden sein müssen. Dazu zählen beispielsweise die Verwaltungstools, die Master Data Services oder die SQL Server Integration Services. Installiert werden diese Komponenten in das Verzeichnis C:\Program Files\Microsoft SQL Server\120.

Der Laufwerksbuchstabe kann in diesem Pfad kann variieren.

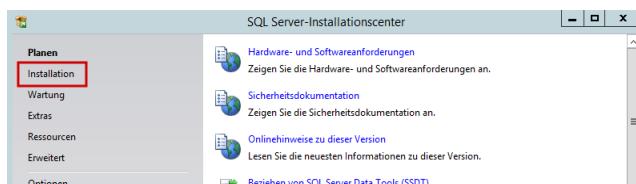


- [ms143547]

15.4.4 SQL Server installieren

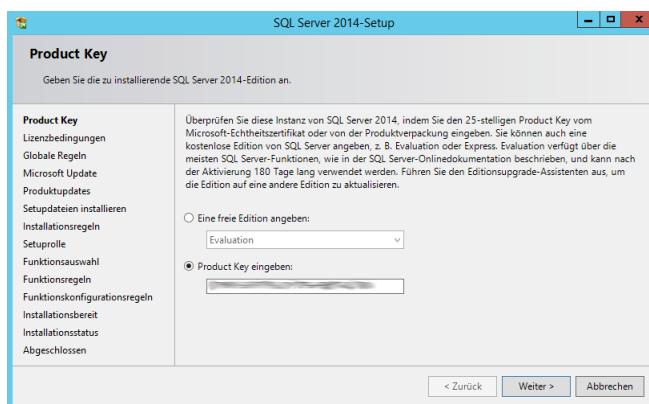
1. Nach dem einlegen der DVD erscheint das Aktionsfenster. Wählen Sie hier die Aktion „Setup.exe ausführen“.
2. Der Setupvorgang wird nun gestartet.
3. Es öffnet sich der „SQL Server-Installationscenter“. Wählen Sie links außen die Option „Installation“ und dann im rechten Fenster „Neue eigenständige SQL Server-Installation oder Hinzufügen von Funktionen zu einer vorhandenen Installation“

Abb. 15.12:
Das SQL-Server-Installationscenter



4. Der erste Schritt bei der Installation des SQL Server 2014 ist die Abfrage eines Product Keys, um die Echtheit der Softwarelizenz zu verifizieren.

Abb. 15.13:
Eingabe des Product Keys



5. Nach dem ein gültiger Product Key eingegeben wurde, muss der Benutzer die Zurkenntnisnahme und die Einhalt der Lizenzbedingungen bestätigen, bevor das Setup vorgesetzt werden kann. Einer Übertragung der Funktionsverwendungsdaten muss nicht zugestimmt werden.
6. Im dritten Schritt des SQL Server 2014-Setup wird eine Überprüfung der Globalen Regeln durchgeführt. Hierbei handelt es sich um Voraussetzungen, die gegeben sein müssen, damit die Installation des SQL Server 2014 erfolgreich sein kann.
7. Eine Überprüfung von Produktupdates ist fest in den Installationsablauf integriert. Dieser Vorgang kann bei Bedarf übersprungen werden, z. B. wenn keine direkte Internetverbindung besteht.

Abb. 15.14:
Die Lizenzbedingungen

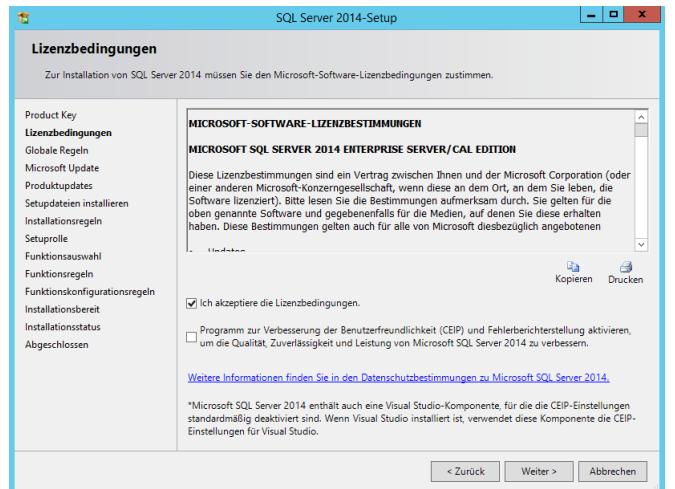


Abb. 15.15:
Überprüfung der Globalen Regeln

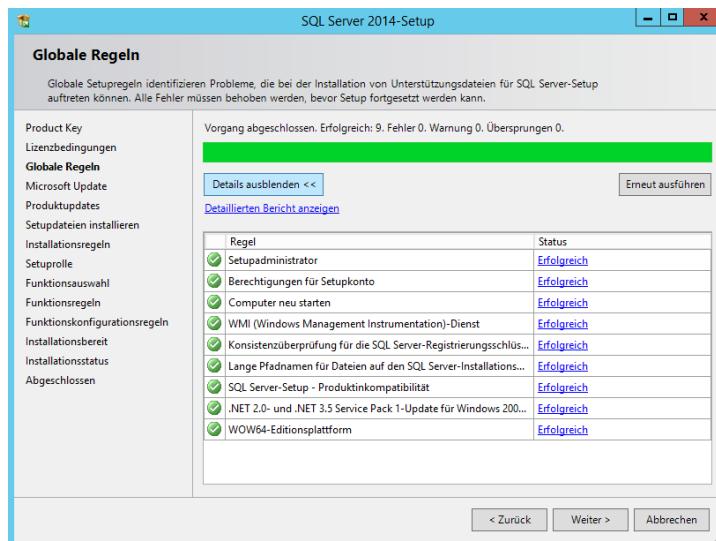
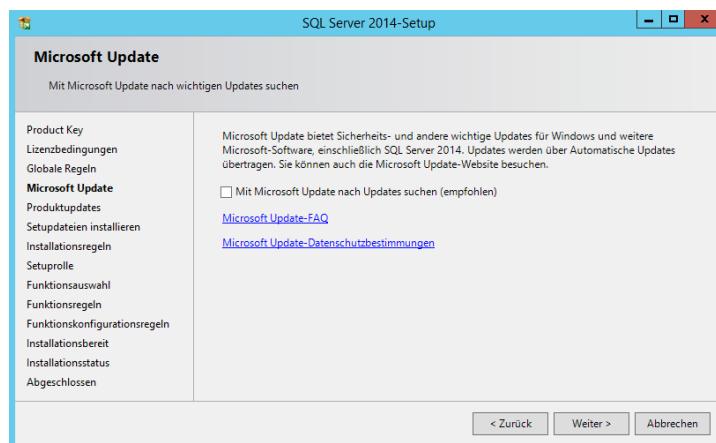
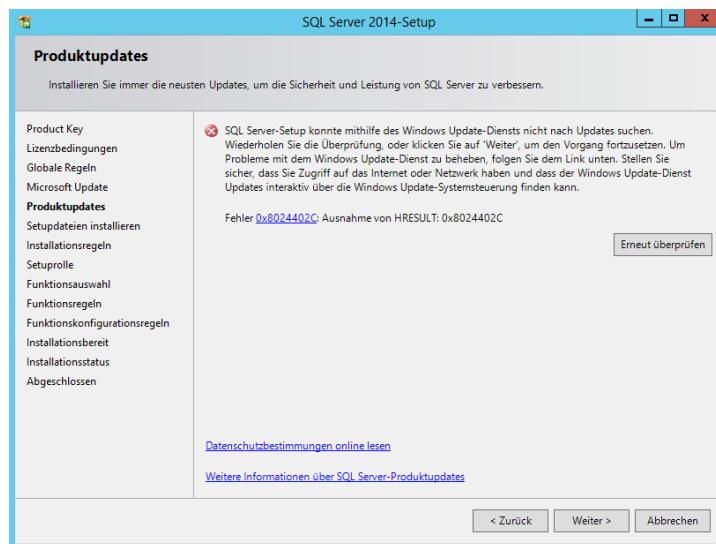


Abb. 15.16:
Überprüfung der Produktupdates



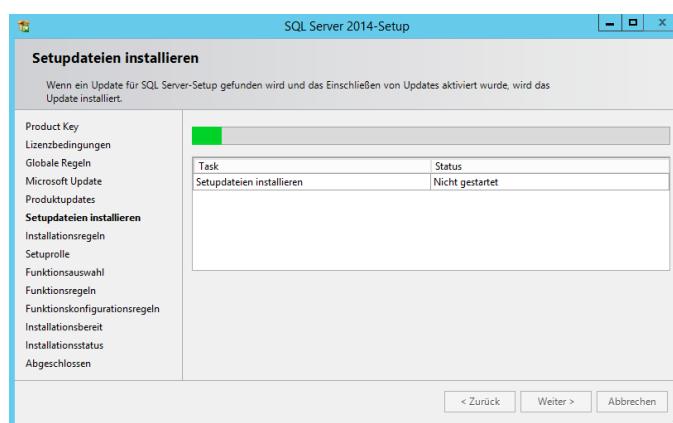
8. Jetzt werden die heruntergeladenen Produktupdates angezeigt. Sollte keine Internetverbindung zur Verfügung stehen, wird das Setup die Fehlermeldung „0x8024402C“ anzeigen.

Abb. 15.17:
Installation der
Produktupdates



9. SQL Server 2014-Setup installiert nun die SQL Server Setupdateien. Im Zuge dieses Installations-schrittes werden auch heruntergeladene Updates mitinstalliert.

Abb. 15.18:
Installation der
Setupdateien



10. In diesem Schritt werden weitere Setupregeln geprüft, um die Installierbarkeit des SQL Servers sicher zustellen.

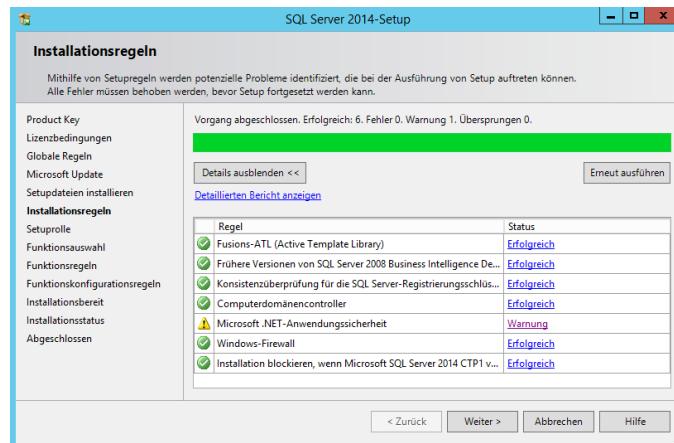


Abb. 15.19:
Überprüfung der
Setupregeln

11. Bei der Auswahl der SQL Server Setuprolle gibt es drei Optionen:

- **SQL Server-Funktionsinstallation**

Mit dieser Option wird eine Standalone-Installation des SQL Server mit vollem Funktionsumfang gestartet, unabhängig von Microsoft SharePoint.

- **SQL Server PowerPivot für SharePoint**

Installiert PowerPivot für Microsoft SharePoint.

- **Alle Funktionen mit Standardwerten**

Führt eine SQL Server-Funktionsinstallation mit Standardwerten durch.

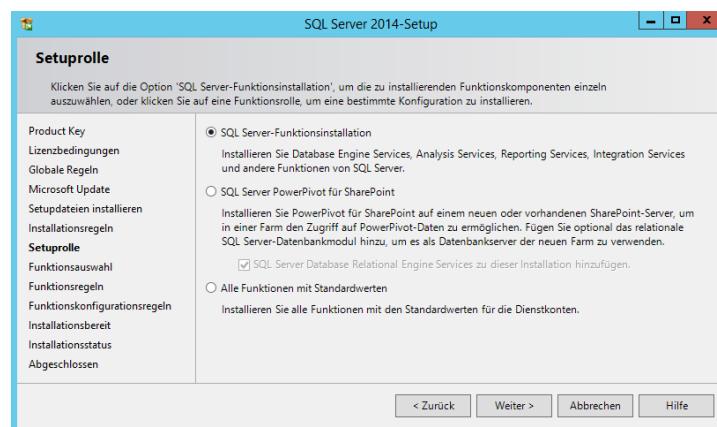


Abb. 15.20:
Auswahl der
Setuprolle

- Bei der SQL Server-Funktionsauswahl können die zu installierenden Komponenten des SQL Server, sowie die Installationsverzeichnisse gewählt werden.

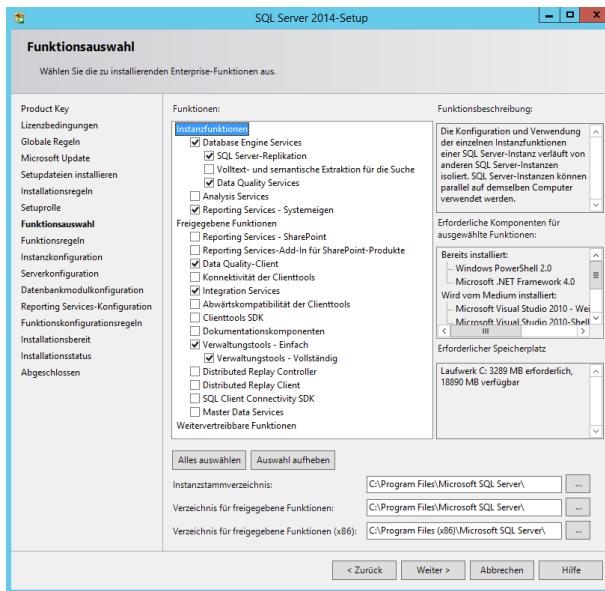


Abb. 15.21:
Die SQL Server-Funktionsauswahl

- Die Prüfung der Installationsregeln ist vergleichbar mit der Überprüfung der Setupunterstützungsregeln. Es werden Probleme aufgespürt, die zu einer Blockade des Installationsvorganges führen würden.

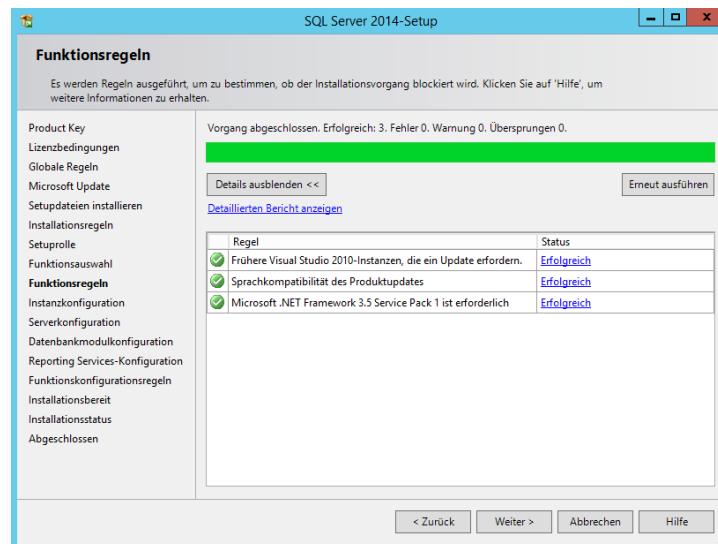


Abb. 15.22:
Prüfung der
Funktionsregeln

- Bei der Instanzkonfiguration werden verschiedene Dinge erledigt. Zum einen wird der zu installierende Instanztyp, standard oder benannt, ausgewählt und zum anderen wird die Instanz-ID festgelegt. Die Instanz-ID gibt, zusammen mit dem Instanzstammverzeichnis, das Verzeichnis an, in welches die Softwareinstallation erfolgen soll. Bei einer Standardinstanz sind Instanzname und Instanz-ID meist gleich, nämlich MSSQLSERVER. Bei einer benannten Instanz weichen beide häufig von einander ab.

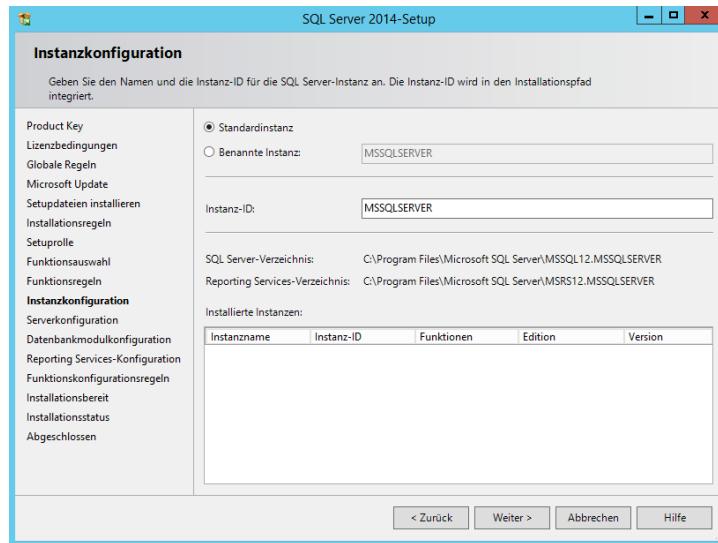


Abb. 15.23:
Instanz-
konfiguration

15. Während der „Serverkonfiguration“ müssen zwei Schritte ausgeführt werden. Zuerst sollten die Dienstkonten für die verschiedenen SQL-Server-Dienste ausgewählt werden. Hierzu sollten schon im Vorfeld gMSAs erstellt werden. Der einzige Dienst, der standardmäßig deaktiviert ist und für den auch an dieser Stelle kein Dienstkontakt ausgewählt werden kann, ist der „SQL Server-Browser“.

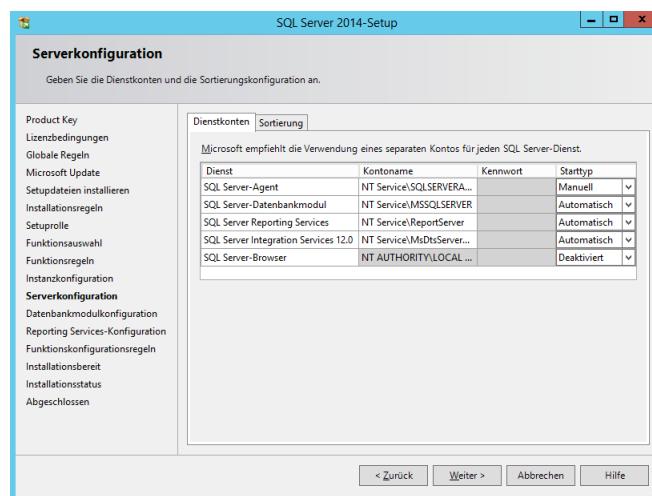


Abb. 15.24:
Auswahl der
Dienstkonten

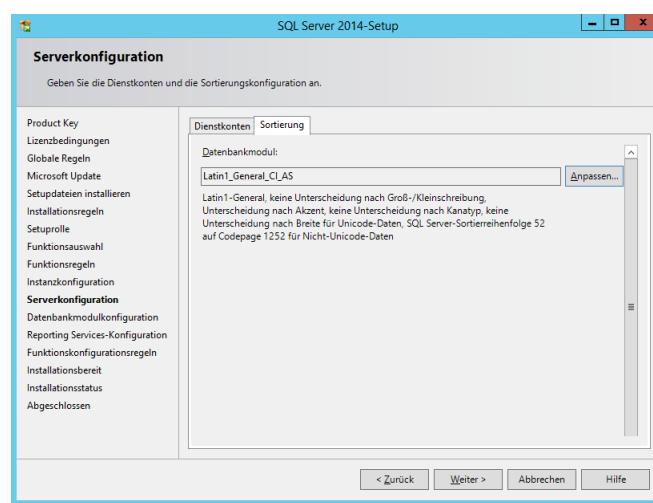
16. Auf der Registerkarte „Sortierung“ kann eine Sortierreihenfolge, engl. „collation order“ ausgewählt werden. Diese legt verschiedene Dinge fest, wie z. B.:

- Welcher Zeichensatz wird genutzt (hier Latin1)?
- In welcher Reihenfolge stehen die Buchstaben im Alphabet?
- Wo werden die Umlaute ä, ö und ü bzw. das sz ß einsortiert?
- Wo werden Ziffern in die Sortierreihenfolge eingereiht?

- Was geschieht mit diakritischen Zeichen (á, ò, ü)?
- Ist Groß-/Kleinschreibung für die Sortierung relevant?

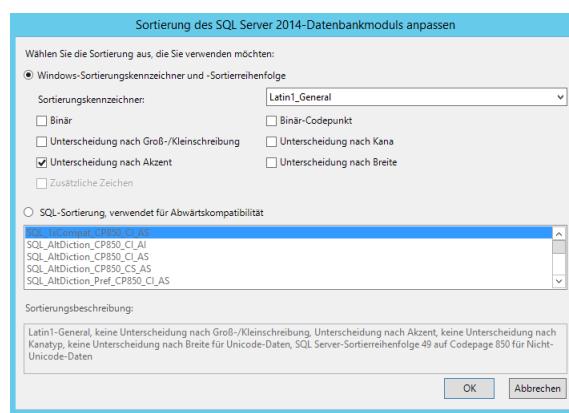
Der Name der Standardsortierreihenfolge „Latin1_General_CI_AS“ enthält die beiden Buchstabenkombinationen „CI“ und „AS“. CI steht für case-insensitive, was bedeutet, dass Groß-/Kleinschreibung irrelevant ist. AS heißt akzent-sensitiv und meint, dass diakritische Zeichen und Umlaute von normalen Buchstaben unterschieden werden (ü ist nicht gleich u und ö ist ungleich o).

Abb. 15.25:
Auswahl der
Sortier-
reihenfolge



Mit einem Klick auf die Schaltfläche „Anpassen“ kann eine eigene Sortierreihenfolge zusammengestellt werden.

Abb. 15.26:
Erstellen einer
eigenen
Sortierreihenfolge



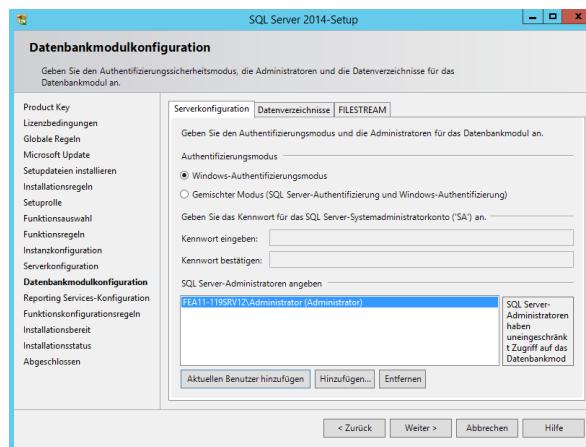
17. Auf der Registerkarte „Serverkonfiguration“ stehen zwei mögliche Authentifizierungsmodi zur Verfügung: Die Windows-Authentifizierung und der Gemische Modus. Gemäß Empfehlung von Microsoft sollte nur noch die Windows-Authentifizierung genutzt werden, da die gesamte Nutzerwaltung so in das Active Directory übertragen wird, ohne das der SQL Server noch darin eingebunden wäre. Über

die Schaltflächen „Aktuellen Benutzer hinzufügen“ bzw. „Hinzufügen“ können Windows-Benutzer hinzugefügt werden, die dann als SQL Server-Administratoren berechtigt werden.



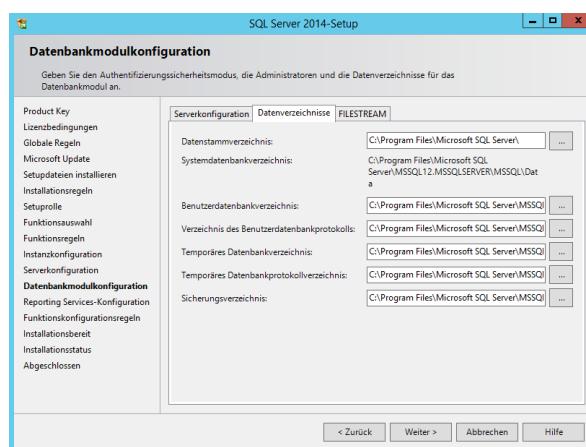
Es muss mindestens ein Windows-Account ausgewählt werden! Dieser muss nicht zwingend lokaler Administrator oder Domänen-Admin sein.

Abb. 15.27:
Datenbankmodul-
konfiguration -
Serverkonfigurati-
on



18. Auf der Registerkarte „Datenverzeichnisse“ können die Standardverzeichnisse für die verschiedenen Dateiarten des SQL Servers gewählt werden.

Abb. 15.28:
Datenbankmodul-
konfiguration
-Datenverzeich-
nisse



19. Die Registerkarte „FILESTREAM“ bietet die Möglichkeit, das Feature FileStream des SQL Server 2014 zu aktivieren.

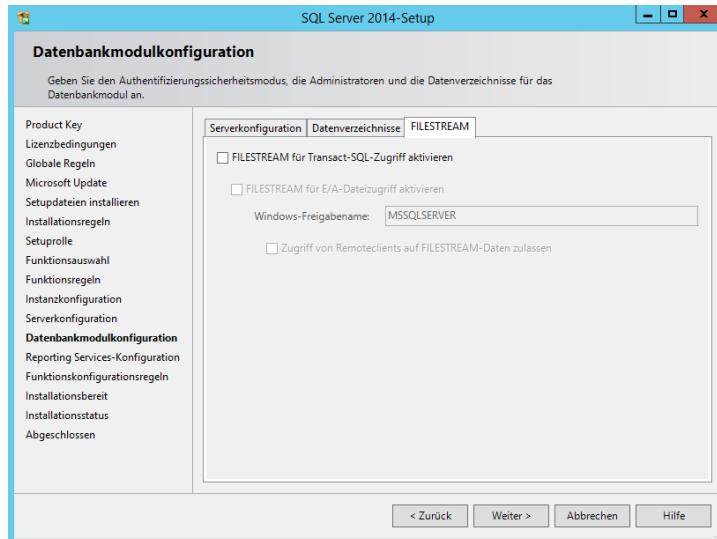


Abb. 15.29:
Datenbankmodulkonfiguration - FileStream

20. Da bei der Funktionsauswahl die SQL Server Reporting Services mit ausgewählt wurden, muss nun hier entschieden werden, ob diese nur installiert oder auch konfiguriert werden sollen.

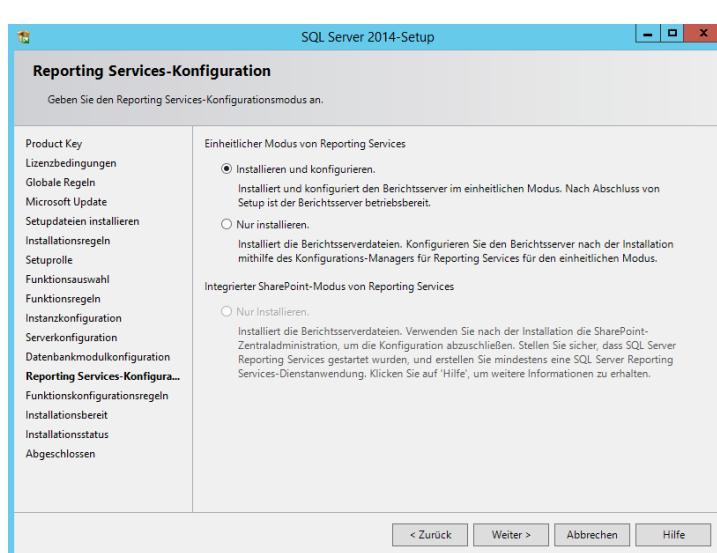


Abb. 15.30:
Installation / Konfiguration der Reporting-services

21. Die Prüfung der Funktionskonfigurationsregeln für die Installation ist vergleichbar mit der Überprüfung der Setupsregeln. Es werden Probleme aufgespürt, die zu einer Blockade des Installationsvorganges führen würden.
22. In diesem vorletzten Schritt zeigt der Setup-Assistent eine Zusammenfassung aller Installationsoperationen. Mit einem Klick auf „Installieren“ startet die Installation der SQL Server-Software.

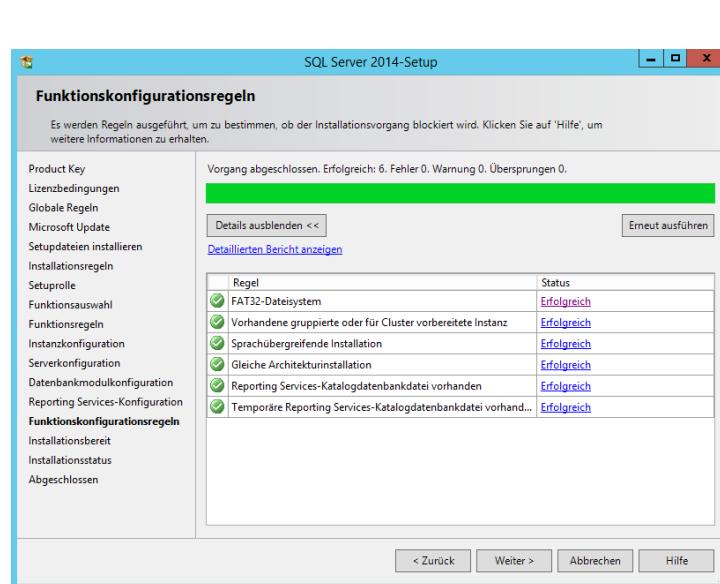


Abb. 15.31:
Prüfung der
Funktions-
konfigurations-
regeln für die
Installation

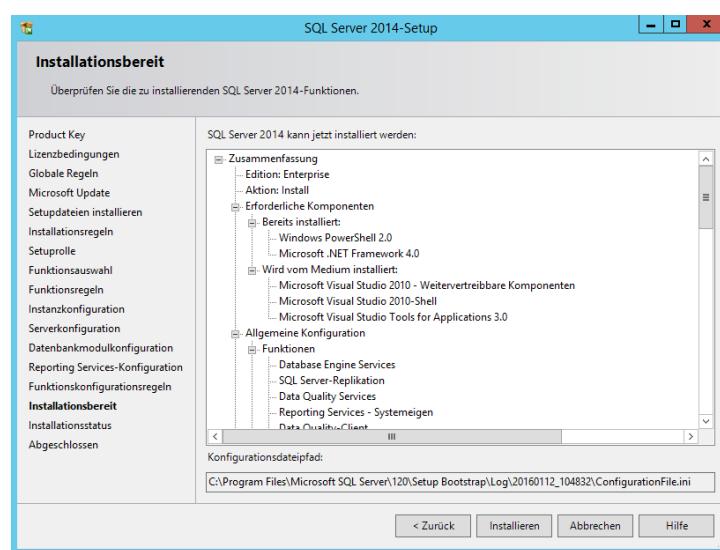


Abb. 15.32:
Zusammen-
fassung der
Installations-
optionen

23. Die Installation der SQL Server-Software läuft.

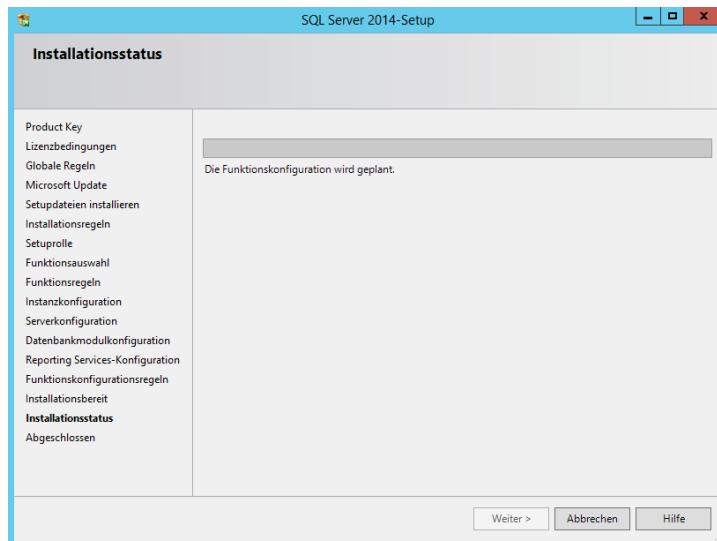


Abb. 15.33:
Installation der
SQL
ServerSoftware

24. Am Ende der Installation zeigt das Setup noch eine Zusammenfassung der ausgeführten Tätigkeiten an.

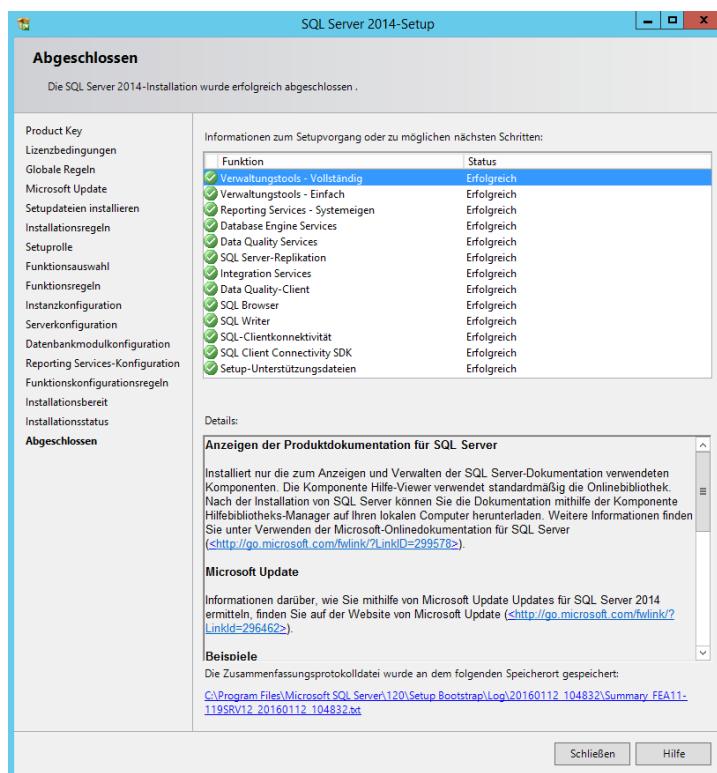


Abb. 15.34:
Installations-
zusammen-
fassung

15.5 Installationsnachbereitung

Nach erfolgreicher Installation des SQL Servers sollte bzw. muss der Administrator noch einige Einstellungen am Server vornehmen.

15.5.1 Der SQL Server Configuration Manager

Der SQL Server Configuration Manager erlaubt es dem Administrator Änderungen an den SQL Server-Diensten und den SQL Server-Netzwerkeinstellungen vorzunehmen.



Microsoft empfiehlt, alle Einstellungen an den SQL Server-Diensten nur mittels des SQL Server Configuration Manager vorzunehmen!

Den Configuration Manager starten

Gestartet wird der Configuration Manager aus dem Windows Startmenü.

Abb. 15.35:
Starten des
Configuration
Manager



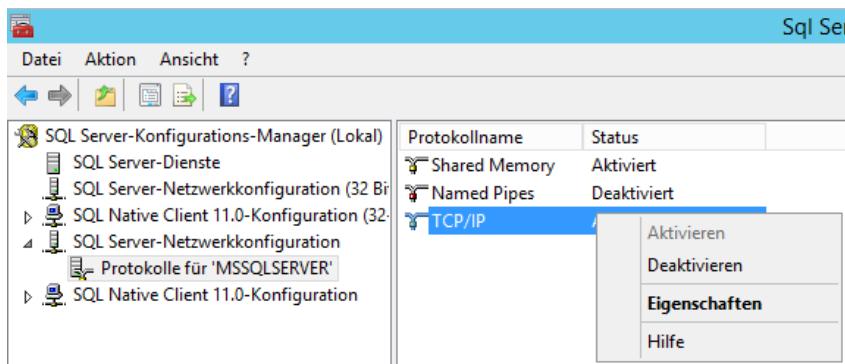
Netzwerkkonfiguration - TCP/IP

Der SQL Server kennt drei Netzwerkprotokolle. Dies sind: „Shared Memory“, „Named Pipes“ und „TCP/IP“. Für jedes dieser Protokolle können die Einstellungen im SQL Server Configuration Manager vorgenommen werden. An dieser Stelle wird aber nur auf die Konfigurationsmöglichkeiten für das Protokoll TCP/IP eingegangen, da die anderen beiden eine eher untergeordnete Rolle spielen.

Nach einem Klick auf „Eigenschaften“ zeigt sich der Dialog aus Abbildung ??, mit seinen beiden Registerkarten „Protokoll“ und „IP-Adressen“. Auf der Registerkarte „Protokoll“ können die folgenden Optionen geändert werden:

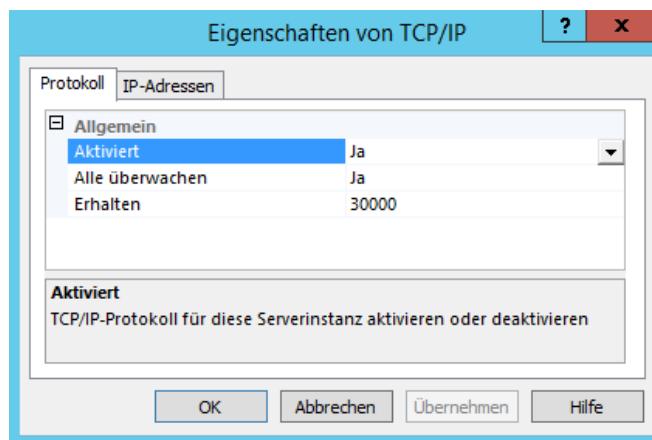
- **Aktiviert:** [Ja | Nein] Wird hier der Wert „Nein“ eingetragen, steht das TCP/IP-Protokoll für den SQL Server nicht mehr zur Verfügung. Mögliche Werte sind Ja und Nein

Abb. 15.36:
Die TCP/IP-
Eigenschaften
1



- **Alle überwachen:** [Ja | Nein] Diese Option steht standardmäßig auf Ja, was bedeutet, dass der SQL Server auf allen im zur Verfügungstehenden IP-Adressen lauscht. Wird der Wert auf Nein geändert, müssen im Folgenden alle IP-Adressen einzeln konfiguriert werden.
- **Erhalten:** [number] Gibt das Verbindungserhaltungsintervall in Millisekunden wieder. Ein Wert von 30.000 bedeutet, dass eine Verbindung für mindestens 30 Sekunden offen gehalten wird, auch wenn sie inaktiv ist. Bei einem Wert von 0 würde eine inaktive Verbindung sofort geschlossen werden.

Abb. 15.37:
Die TCP/IP-
Eigenschaften
2



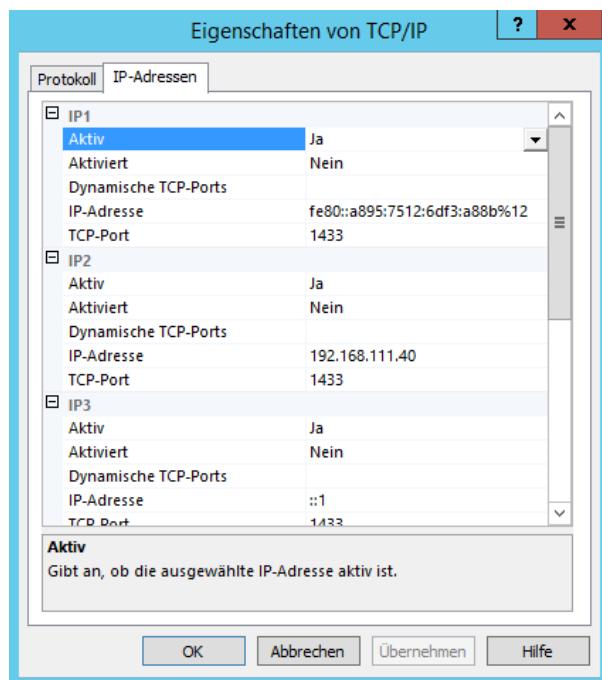
Wechselt man auf die Registerkarte „IP-Adressen“ erhält man dort die Möglichkeit, alle IP-Adressen des Servers einzeln zu konfigurieren.

Hier können folgende Einstellungen pro IP-Adresse geändert werden:

- **Aktiv:** [Ja | Nein] Gibt an, ob die IP-Adresse im Betriebssystem aktiviert oder deaktiviert ist.
- **Aktiviert:** [Ja | Nein] Gibt an, ob der SQL Server auf dieser IP-Adresse Verbindungsanforderungen entgegen nimmt oder nicht. Diese Option wird nur dann wirksam, wenn auf der Registerkarte „Protokoll“ der Parameter „Alle überwachen“ den Wert Ja hat, andernfalls wird sie ignoriert.

- **Dynamische TCP-Ports:** [NULL | 0] Durch das Eintragen des Wertes 0 wird beim nächsten Start des SQL Server-Dienstes ein freier TCP-Port dynamisch vergeben. Um einen Port statisch zu vergeben, muss das Feld leer gelassen werden.
- **IP-Adresse:** [w.x.y.z] Hier wird die IP-Adresse eingetragen. Dies kann sowohl eine IPv4- als auch eine IPv6-Adresse sein.
- **TCP-Port:** [0 - 65535] Hier wird der TCP-Port zur IP-Adresse gewählt. Es kann auch eine Liste von Ports, getrennt durch Koma-Zeichen, angegeben werden, wenn auf der Registerkarte „Protokoll“ der Parameter „Alle überwachen“ den Wert Ja hat.

Abb. 15.38:
Die Registerkarte
IP-Adressen der
TCP/IP-Eigen-
schaften



Netzwerkkonfiguration - Dynamische Ports

Der SQL Server kann dynamische Ports benutzen. Das heißt, dass bei jedem Neustart des SQL Server-Dienstes automatisch ein freier TCP-Port gewählt wird. Alle Clients müssen dann diesen Port zur Verbindung mit dem SQL Server benutzen. Um dies realisieren zu können, wird der SQL Server Browser Dienst zur Verfügung gestellt.



Dynamische Ports sind nicht ganz so dynamisch, wie es auf den ersten Blick scheint. SQL Server versucht nämlich den bereits benutzten Port zu reservieren, es sei denn, dass dieser schon anders gebunden wurde. Zudem kann die Nutzung dynamischer Ports, in Verbindung mit einer Firewall zu Verbindungsproblemen bei den Clients führen.

Der SQL Server Browser-Dienst lauscht auf dem Port 1434 und veröffentlicht für die Clients die Verbindungsinformationen zu den SQL Server-Instanzen. Ein Client muss lediglich ein UDP-Paket an den Port 1434 schicken, um alle notwendigen Informationen zu erhalten.



Microsoft empfiehlt im Technet Artikel „Configure a Windows Firewall for Database Engine Access“ [ms175043], den SQL Server Browser-Dienst möglichst nicht zu benutzen, da er sehr viele Informationen preisgibt.

Netzwerkkonfiguration - Netzwerkaliase

Ein Netzwerkalias ist ein Name für eine Verbindungszeichenfolge zu einer SQL Server-Instanz. Der Aliasname ist vom Benutzer frei wählbar. Für die zugehörige Verbindungszeichenfolge gibt es drei unterschiedliche Varianten:

- `tcp:<servername>[<instancename>],<port>`
- `tcp:<IPAddress>[<instancename>],<port>`
- `local`



In einer Verbindungszeichenfolge werden die beiden Adressstypen IPv4 und IPv6 akzeptiert.

Netzwerkaliasnamen kommen immer dann zum Einsatz, wenn:

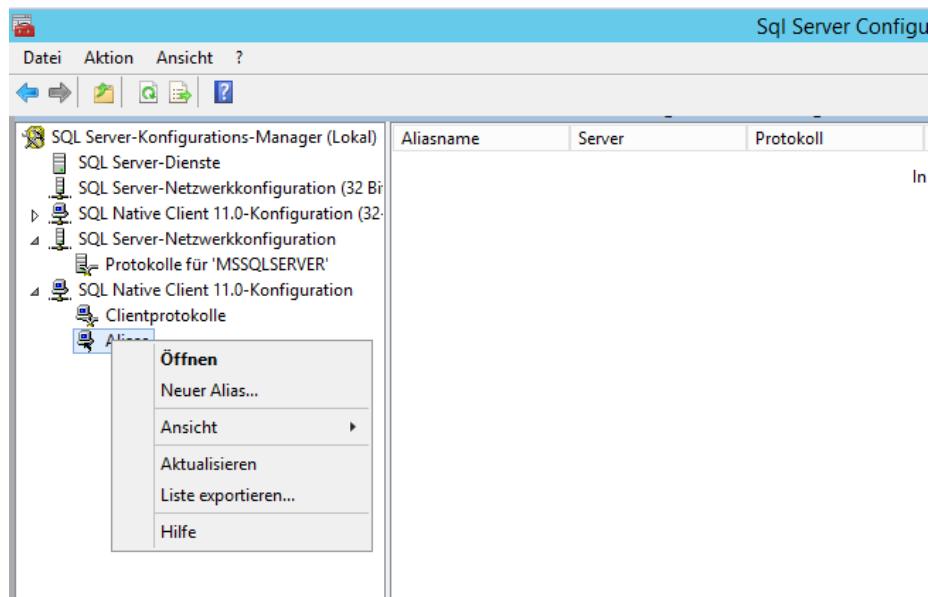
- Ein anderer Port, als der Standardport 1433 genutzt werden soll,
- oder wenn eine Instanz auf einen anderen Serverumgezogen werden muss.

Sie lösen das Problem, dass in jeder Clientsoftware eine Verbindungszeichenfolge angegeben werden muss, die auf einen ganz bestimmten Server verweist. Wenn z. B. die benannte Instanz CRM bisher auf dem

Server SRV01 war und nun auf SRV02 umgezogen werden muss, lautete die alte Verbindungszeichenfolge SRV01\CRM. Wurde kein Alias verwendet, müsste in jeder Clientsoftware die neue Verbindungszeichenfolge SRV02\CRM eingerichtet werden. Ist die Zeichenfolge fest in der Anwendung codiert, ist eine solche Umstellung nur mittels des Anwendungsentwicklers möglich, der die Software erstellt hat. Ist die Software konfigurierbar, muss auf jedem Computer, auf dem die Software benutzt wird, die Konfiguration geändert werden, was einen sehr hohen Arbeitsaufwand bedeuten kann.

Mit Hilfe eines Netzwerkaliases wird das Problem deutlich vereinfacht. Statt an jeder Software eine Änderung zu vollziehen, wird lediglich die Verbindungszeichenfolge des Alias geändert und die Software selbst bleibt unverändert.

Abb. 15.39:
Anlegen eines
Netzwerk-
aliasnamens



15.5.2 Instant File Initialization

Erstellt der Microsoft SQL Server eine neue Datei, wird diese immer zuerst mit Nullen auf ihre volle Größe aufgefüllt, bevor sie benutzbar ist. Da dieser Vorgang sehr kostenintensiv ist, sollte er umgangen werden.

Instant File Initialization in der lokalen Sicherheitsrichtlinie

In der lokalen Sicherheitsrichtlinie kann dem SQL Server-Dienstkonto unter:

- Lokale Richtlinien

- Zuweisen von Benutzerrechten

das Recht: „Performe Volume Maintenance Tasks“ bzw. „Durchführen von Volumenwartungsaufgaben“ zugewiesen werden.

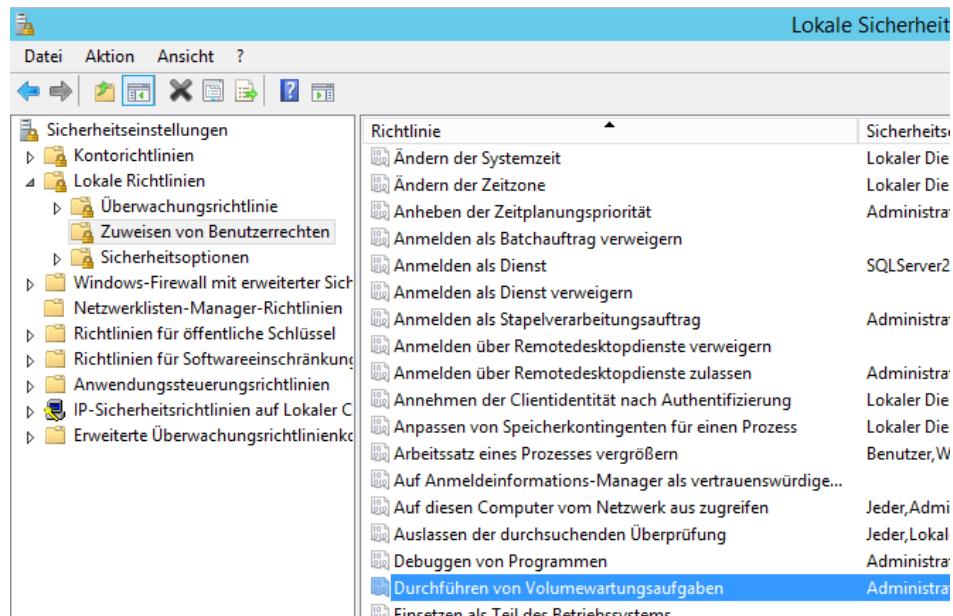


Abb. 15.40:
Die lokale Sicherheitsrichtlinie

Instant File Initialization als Gruppenrichtlinie

Befindet sich der SQL Server in einer Windows-Domäne, was fast immer der Fall ist, dann sollten Berechtigungen, wie das Durchführen von Volumenwartungsaufgaben als Gruppenrichtlinie im Active Directory hinterlegt werden. Gehen Sie dazu wie folgt vor:

1. Starten Sie den Gruppenrichtlinieneditor



Abb. 15.41:
Starten des
Gruppenrichtlinie-
editors

2. Klicken Sie im linken Teilfenster, mit der rechten Maustaste, auf „Gruppenrichtlinienverwaltung“. Wählen Sie im Kontextmenü „Gesamtstruktur hinzufügen“.
3. Geben Sie im Dialogfenster „Gesamtstruktur hinzufügen“ den Namen der Domäne ein. Dies ist hier: MS-C-IX-04.FUS.

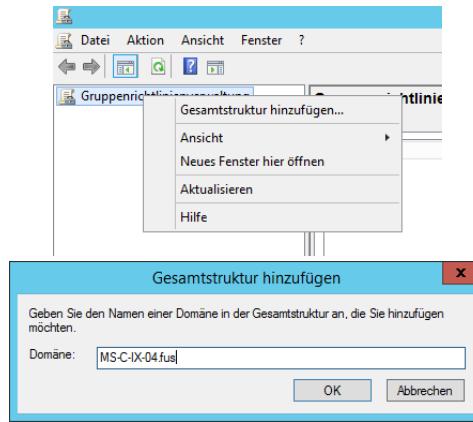
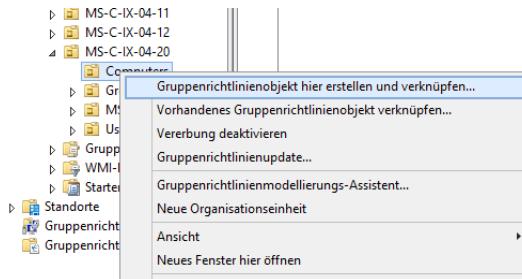


Abb. 15.42:
Hinzufügen einer
Gesamtstruktur 1

Abb. 15.43:
Hinzufügen einer
Gesamtstruktur 2

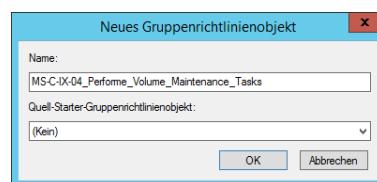
4. Klicken Sie im linken Teilfenster auf „Gesamtstruktur: ms-c-ix-04.fus“, „Domänen“, „ms-c-ix-04.fus“, Ihre OU, z. B. „MS-C-IX-04-20“ und dann auf die OU „Computers“.
5. Klicken Sie mit der rechten Maustaste auf die OU „Computers“ und wählen Sie aus dem Kontextmenü den Punkt „Gruppenrichtlinienobjekt hier erstellen und verknüpfen“.

Abb. 15.44:
Hinzufügen einer
Gruppenrichtlinie



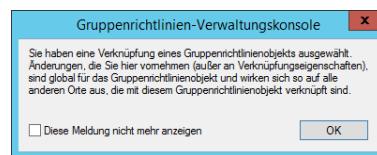
6. Geben Sie der Gruppenrichtlinie im Dialogfenster „Neues Gruppenrichtlinienobjekt“ einen Namen, z. B. „MS-C-IX-04-20_Perfome_Volume_Maintenance_Tasks“

Abb. 15.45:
Hinzufügen einer
Gruppenrichtlinie



7. Es erscheint eine Warnmeldung. Klicken Sie auf OK!

Abb. 15.46:
Warnung



8. Klicken Sie mit der rechten Maustaste auf die Gruppenrichtlinie und wählen Sie aus dem Kontextmenü den Punkt „Bearbeiten“

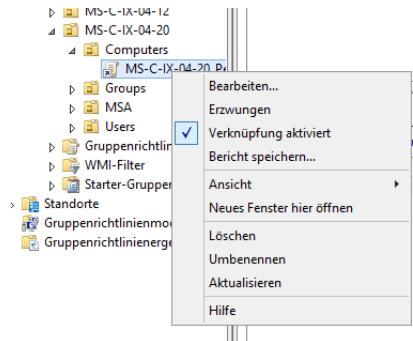


Abb. 15.47:
Die
Gruppenrichtlinie
bearbeiten

9. Im Gruppenrichtlinienverwaltungs-Editor wählen Sie im linken Teilfenster: „Computerkonfiguration“, „Richtlinien“, „Windows-Einstellungen“, „Sicherheitseinstellungen“, „Lokale Richtlinien“, „Zuweisen von Benutzerrechten“

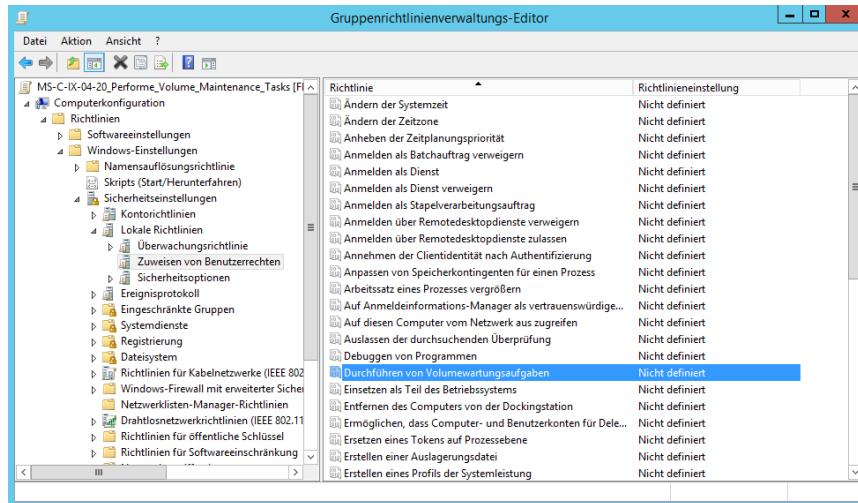
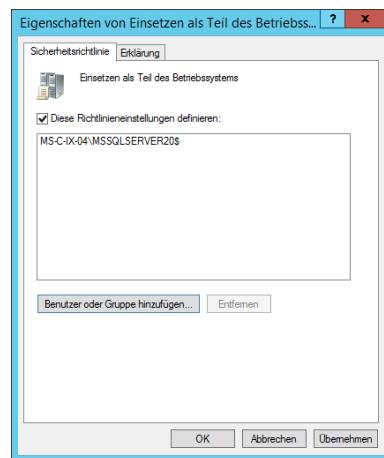


Abb. 15.48:
Die
Gruppenrichtlinie
bearbeiten

10. Wählen Sie die Richtlinie „Durchführen von Volumewartungsaufgaben“ mit einem Doppelklick aus.

11. Konfigurieren Sie die Richtlinie, in dem Sie Ihr gMSA-Konto, welches von Ihrem SQL Server-Dienst genutzt wird in die Richtlinie aufnehmen.

Abb. 15.49:
Die Gruppenrichtlinie bearbeiten



12. Schließen Sie die Gruppenrichtlinienverwaltung

15.5.3 Lock Pages in Memory (Sperren von Seiten im Speicher)

Um ein möglichst performantes Arbeiten mit den Nutzdaten zu ermöglichen, verfügt der SQL Server über eine ganze Reihe von Buffers und Caches, die im Arbeitsspeicher des Servers liegen. Um die Performance des SQL Servers nicht zu schädigen ist es wichtig, dass diese Buffers und Caches keinesfalls auf den Datenträger ausgelagert werden (pagefile.sys). Da Microsoft Windows aber standardmäßig die Inhalte des Arbeitsspeichers bei Bedarf auslagert, könnte genau dieser unerwünschte Fall eintreten.

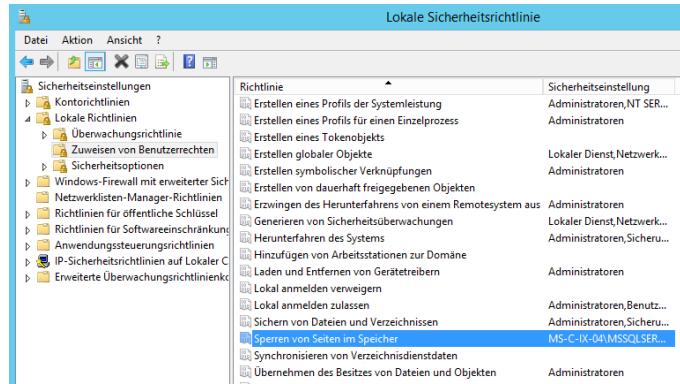
Lock Pages in Memory in der lokalen Sicherheitsrichtlinie

Um dies zu verhindern kann dem SQL Server-Dienstkonto in der lokalen Sicherheitsrichtlinie unter:

- Lokale Richtlinien
- Zuweisen von Benutzerrechten

das Recht: „Sperren von Seiten im Speicher“ bzw. „Lock pages in memory“ zugewiesen werden. Diese Richtlinie erlaubt es dem SQL Server, dass Auslagern seiner Speicherseiten durch das Betriebssystem zu verhindern.

Abb. 15.50:
Die lokale Sicherheitsrichtlinie



Lock Pages in Memory als Gruppenrichtlinie

Die Richtlinie „Lock Pages in Memory“ wird auf die gleiche Art und Weise als Gruppenrichtlinie angelegt, wie es unter ?? beschrieben wurde.



- [ms190730]

16 Verwalten einer SQL Server Instanz

Inhaltsangabe

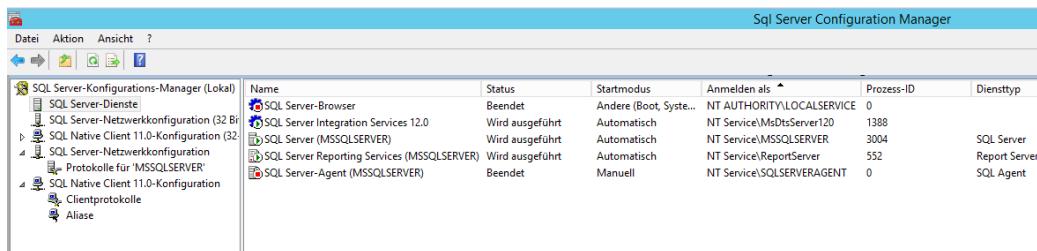
16.1 Die Windows Dienste des SQL Servers

Der Microsoft SQL Server ist ein Produkt, dass aus einer Vielzahl verschiedener Windowsdienste besteht. Zur Kontrolle und Konfiguration dieser Dienste wird der „SQL Server Configuration Manager“ mitgeliefert.



Die Dienste des SQL Servers sollten immer nur mit Hilfe des SQL Server Configuration Managers verwaltet werden!

Abb. 16.1:
Die
Windowsdienste
des SQL Server
2014



Sql Server Configuration Manager						
	Name	Status	Startmodus	Anmelden als	Prozess-ID	Diensttyp
SQL Server-Konfigurations-Manager (Lokal)	SQL Server-Browser	Beendet	Anderer (Boot, Syste...)	NT AUTHORITY\LOCALSERVICE	0	
SQL Server-Dienste	SQL Server Integration Services 12.0	Wird ausgeführt	Automatisch	NT Service\MsDtsServer120	1388	
SQL Native Client 11.0-Konfiguration (32 Bit)	SQL Server (MSSQLSERVER)	Wird ausgeführt	Automatisch	NT Service\MSSQLSERVER	3004	SQL Server
SQL Server-Netzwerkkonfiguration	SQL Server Reporting Services (MSSQLSERVER)	Wird ausgeführt	Automatisch	NT Service\ReportServer	552	Report Server
Protokolle für 'MSSQLSERVER'	SQL Server-Agent (MSSQLSERVER)	Beendet	Manuell	NT Service\SQLSERVERAGENT	0	SQL Agent
SQL Native Client 11.0-Konfiguration						
Clientprotokolle						
Aliase						

16.1.1 SQL Server

Der „SQL Server“-Dienst ist der wichtigste von allen. Von ihm wird die sogenannte „Database Engine“, der Datenbankdienst, zur Verfügung gestellt. Mit seiner Hilfe werden alle Dateiarten verwaltet, die eine SQL Server Datenbank besitzt, er führt Transakt-SQL-Statements aus, überwacht die Einhaltung der Referentiellen Integrität und bewahrt somit die Konsistenz der Daten.

Namensgebung

Wenn es nur eine SQL Server Instanz auf einem Server gibt, trägt dieser Dienst den Namen SQL SERVER (MSSQLSERVER). Im Falle einer benannten Instanz wird der Instanzname mit angegeben: SQL SERVER (MSSQLSERVER\$CRM).

Startparameter

Hinter dem SQL Server-Dienst verbirgt sich die ausführbare Datei `sqlserv.exe`. Diese befindet sich, wie die meisten .EXE Dateien im Ordner BINN. Zu diesem Programm gibt es eine Reihe von Startparametern.



Die folgende Tabelle zeigt nur die wichtigsten Parameter. Eine vollständige Auflistung kann der MSDN entnommen werden.

- [ms162819]

-d	Gibt den vollständigen Dateipfad zur Master-Datenbank an. Es dürfen sich keine Leerzeichen zwischen dem Parameter und dem Dateipfad befinden. Sollte dieser Parameter nicht angegeben werden, benutzt SQL Server die Angaben aus der Windows-Registry.
-l	Gibt den vollständigen Dateipfad zur Log-Datei der Master-Datenbank an. Es darf sich kein Leerzeichen zwischen dem Parameter und dem Dateipfad befinden.
-e	Dieser Parameter gibt den vollständigen Dateipfad für das Fehlerprotokoll der SQL Server-Instanz an. Sollte dieser Parameter nicht angegeben worden sein, so wird für das Fehlerprotokoll der Standardinstanz der Dateiname C:\Programme\Microsoft SQL Server\MSSQL\Log\Errorlog benutzt. Bei einer benannten Instanz wird das Protokoll im Ordner C:\Programme\Microsoft SQL Server\MSSQL\$Instanzname\Log\Errorlog gespeichert.
-f	Ermöglicht den Start des SQL Servers mit Minimalkonfiguration. Dieser Parameter sollte nur im Fehlerfall zum Einsatz kommen, falls der SQL Server aufgrund einer fehlerhaften Einstellung nicht mehr startet.
-m	Startet den SQL Server im Einzelbenutzermodus. Verwenden Sie diesen Parameter, wenn es Probleme mit einer der Systemdatenbanken gibt, z. B. die Master-Datenbank muss wiederhergestellt werden.
-T	Aktiviert die Ablaufverfolgung mittels Trace-Flags. Trace-Flags verändern das Verhalten des SQL Servers und dienen zu sehr unterschiedlichen Zwecken, meist aber zur Gewinnung zusätzlicher Informationen.

Startmodus und Recovery-Startmodus auswählen

für den SQL Server-Dienst werden drei verschiedene Startmodi bereitgestellt.

- **Automatisch:** Der Dienst wird mit dem Betriebssystem gestartet
- **Manuell:** Der Dienst muss manuell, durch den Administrator, gestartet werden
- **Deaktiviert:** Der Dienst wird nicht gestartet. Ein manuelles Starten ist ebenfalls nicht möglich.

Nach der Installation der SQL Server-Software ist der SQL Server-Dienst so konfiguriert, dass er automatisch mit dem Betriebssystem startet. Soll der Dienst beispielsweise deaktiviert oder aus irgendeinem Grund nicht automatisch gestartet werden, kann man den Startmodus im SQL Server Configuration Manager verändern.

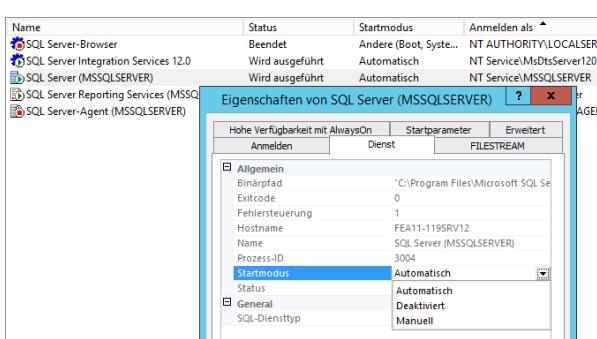


Abb. 16.2:
Den Startmodus
des SQL
Server-Dienstes
verändern

Sollte der SQL Server-Dienst einmal ausfallen/abstürzen, kann über die Windows Dienstekonsole eine Notfallaktion geplant werden. Auf der Registerkarte „Wiederherstellung“ können für jeden Dienst Aktionen angegeben werden, die dann ausgeführt werden, wenn der Dienst das erste mal bzw. das zweite mal oder ein wiederholtes mal abgestürzt ist.

Es empfiehlt sich, die Option ERSTER FEHLER auf den Wert „Dienst neu starten“ einzustellen, so dass der SQL Server Dienst neugestartet wird. Zusätzlich dazu sollten die beiden Optionen FEHLERZÄHLER NACH N TAGEN ZURÜCKSETZEN und DIENST NACH N MINUTE(N) NEU STARTEN auf den Wert 1 eingestellt werden.

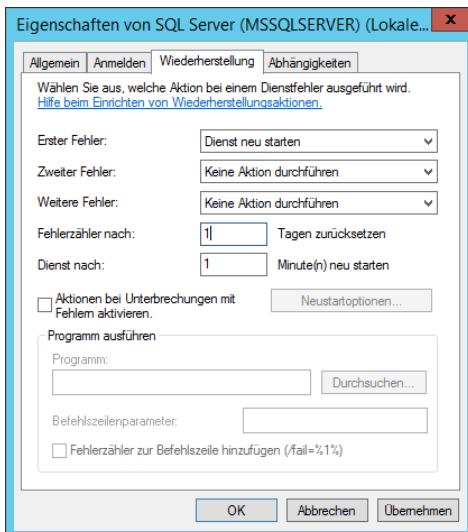


Abb. 16.3:
Eine Wieder-
herstellungsaktion
planen

16.1.2 SQL Server-Agent

Der SQL Server Agent wird durch die Datei **SQLAGENT.EXE** bereitgestellt. Er ist ein Dienst, der dem Administrator die Automatisierung von Aufgaben innerhalb der Datenbank erlaubt. Er kann z. B. Backups zeitgesteuert ablaufen lassen, auf Warnungen des SQL Servers reagieren, Nachrichten an den Admin versenden und ähnliches.

Dieser Dienst ist immer 1:1 mit einer SQL Server-Instanz verbunden, die ihm mittels des Initialisierungsparameters `-i Instanzname` mitgeteilt wird. Im Falle einer Standardinstanz trägt er den Namen SQL SERVER-AGENT. Bei einer benannten Instanz ist der Instanzname mit angehängt: SQL SERVER-AGENT\$CRM.

16.1.3 SQL Server-Browser

Der SQL Server-Browser hat die Aufgabe, nach eingehenden Ressourcenanforderungen von Datenbank-Clients zu lauschen. Er stellt den Clients Informationen zu allen installierten SQL Server- und SQL Server Analysis Services-Instanzen zur Verfügung. Außerdem ermöglicht er die Verbindung zu einer benannten SQL Server-Instanz oder den Verbindungsaufbau mittels der DAC (Dedicated Administrator Connection).

Dieser Dienst wurde mit dem SQL Server 2005, als Ersatz für das „SQL Server Resolution Protocol“ eingeführt. Er arbeitet auf Port 1434 und wird durch die .EXE-Datei `sqlbrowser.exe` zur Verfügung gestellt.

16.1.4 SQL Server Integration Services 12.0

Die SQL Server Integration Services stellen eine komplexe Plattform zur Transformation und Integration von Daten dar. Mit Ihnen können Daten kopiert, per FTP heruntergeladen oder per Mail versandt werden. Die Integration Services sind in der Lage, Daten aus einer großen Zahl von Quellen (XML, relationale Datenbanken, Textdateien, Excel-Tabellen, usw.) zu lesen und zu verarbeiten.

Der Integration Services-Dienst wird durch die ausführbare Datei `MsDtsSrvr.exe` bereitgestellt. Diese benötigt keine Startparameter.



Da dieser Dienst optional ist, sollte er nur bei Bedarf installiert werden!

16.1.5 Der SQL Server VSS Writer

Der SQL Server VSS Writer ermöglicht es Backups von Datendateien mittels des Windows Volume Shadow Copy Service zu erzeugen. Wenn er aktiviert ist, kann jede Backupsoftware, die den VSS-Dienst nutzt, Online-Backups der Datendateien machen. Ist der Dienst deaktiviert, können Online-Backups nur

mit Hilfe des SQL Servers selbst durchgeführt werden, da der SQL Server-Dienst eine Sperre auf seine Dateien legt.

Sofern Backups nur mit Hilfe des SQL Server BACKUP-Kommandos gemacht werden, kann dieser Dienst deaktiviert werden.



Der SQL Server VSS Writer muss immer mit dem lokalen System-Konto betrieben werden! Er ist der einzige SQL Server-Dienst, der nur über die Windows Dienste Konsole konfiguriert werden kann. Er ist nicht im SQL Server Configuration Manager abgebildet.



- [ms175536]

16.2 Das SQL Server Management Studio

Das SQL Server Management Studio (SSMS) ist das zentrale Werkzeug zur Verwaltung des SQL Server 2014. Es wurde mit der Version 2005 des SQL Server eingeführt und ersetzte damals eine Vielzahl von eigenständigen Tools, wie z. B. den Enterprise Manager, den Query Analyzer oder den Analysis-Manager.



Abb. 16.4:
Das SQL Server
2014
Management
Studio

16.2.1 Login mit dem Management Studio

Servertypes und Servername

Nach dem Starten des SSMS erscheint zuerst der Dialog „Verbindung mit Server herstellen“. Dieser dient dazu, sich bei einem der verschiedenen Servertypen zu authentifizieren. Dabei stehen zur Auswahl:

- Datenbankmodul
- Analysis Services
- Reporting Services
- Integration Services

Abb. 16.5:
Die
verschiedenen
Servertypen im
Login-Dialog



Unter „Servername“ muss der DNS-Name des Datenbankservers eingetragen werden, an dem die Anmeldung erfolgen soll. Wenn das Ziel eine benannte Instanz eines SQL Servers ist, muss der Instanzname zusätzlich zum Rechnernamen mit angegeben werden, z. B. FEA11-119SRV12\$CRM.

Authentifizierung

für die Authentifizierung stehen dem Benutzer zwei verschiedene Verfahren offen:

- **Windows-Authentifizierung:** Diese Form der Authentifizierung wird auch als „integrierte Sicherheit“ bezeichnet und stellt den Standardauthentifizierungsmechanismus dar. Es werden bestimmte Windows Benutzer oder Gruppen im SQL Server als vertrauenswürdig eingetragen, so dass sich diese, dann ohne die Angabe eines Kennwortes, am SQL Server anmelden dürfen.

- **SQL Server-Authentifizierung:** Bei diesem Verfahren wird die komplette Authentifizierung der Benutzer durch den SQL Server selbst durchgeführt. Die Sicherheit dieses Verfahrens ist längst nicht so hoch, wie bei der Windows-Authentifizierung.

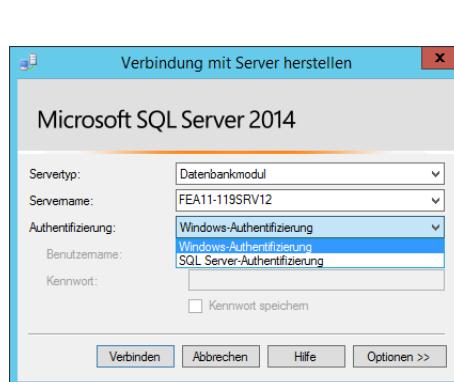


Abb. 16.6:
Die
verschiedenen
Authen-
tifizierungstypen
im Login-Dialog

Weitere Informationen zu diesen beiden Authentifizierungsverfahren finden Sie im Kapitel „Sicherheit im SQL Server“.



16.2.2 Der Aufbau des SSMS - Objekt-Explorer

Nach einer erfolgreichen Authentifizierung erscheint dann das SQL Server Management Studio in seiner Grundeinstellung.

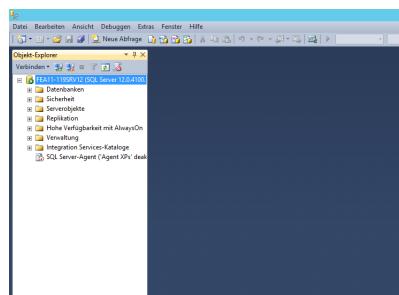


Abb. 16.7:
Das SQL Server
Management
Studio

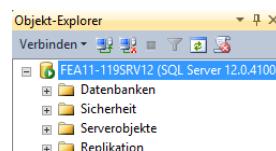
Zu Beginn ist nur ein einziger Bereich sichtbar, der „Objekt-Explorer“. Mit ihm ist es möglich:

- Server zu verwalten
- Objekte zu erstellen und zu verändern
- Log-Dateien auszuwerten

und noch vieles Anderes.

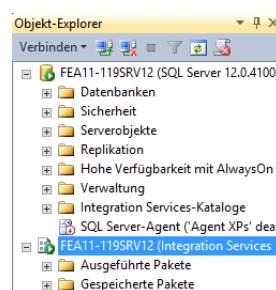
Die Daten werden im Objekt-Explorer immer hierarchisch aufgebaut. Zu jeder Instanz, werden die vorhandenen Objekte als Baum angezeigt. Da während des Logins eine Verbindung zu einem Datenbankmodul hergestellt wurde, wird im Objekt-Explorer die betreffende Datenbankinstanz angezeigt.

Abb. 16.8:
Die
Datenbankinstanz
FEA11-
119SRV12



Abhängig davon, mit welcher Art von Dienst man verbunden ist, variiert der Inhalt des Objekt-Explorers.

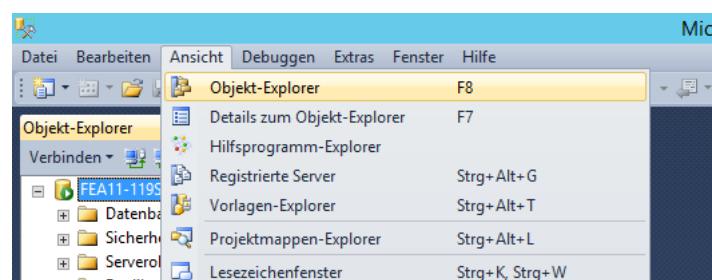
Abb. 16.9:
Zwei
unterschiedliche
Instanzen im
Objekt-Explorer



Wenn der Objekt-Explorer einmal nicht da ist

Sollte der Objekt-Explorer aus irgendeinem Grund nicht zu sehen sein, kann er über das Menü ANSICHT - OBJEKT-EXPLORER oder durch Drücken der Taste F8 angezeigt werden.

Abb. 16.10:
Das Menü
Ansicht



Verbindung mit einer Instanz herstellen

Wie in Abbildung ?? zu sehen ist, kann der Objekt-Explorer Verbindungen zu mehreren Instanzen aufbauen. Um sich mit einer Instanz zu verbinden, bzw. um eine zusätzliche Verbindung herzustellen, existiert der „Verbinden“-Button, oben links im Objekt-Explorer.

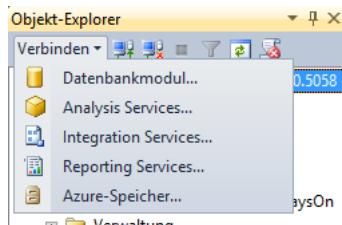


Abb. 16.11:
Die Schaltfläche
„Verbinden“

Nach der Auswahl des Server-Typs erscheint der Login-Dialog, um den Benutzer zu authentifizieren.

Verbindungen trennen

Wenn die Verbindung zu einem Server Typ getrennt werden soll, geschieht dies über den „Trennen“-Button im Objekt-Explorer. Der betroffene Server Typ wird aus dem Objekt-Explorer entfernt.



Abb. 16.12:
Die Schaltfläche
„Trennen“

16.2.3 Der Aufbau des SSMS - Der Abfrageeditor

Eines der wichtigsten Tools im SSMS ist der Abfrageeditor. Er dient zur Eingabe und Ausführung jeglicher T-SQL Statements. Aufgerufen wird er durch einen Klick auf die Schaltfläche „Neue Abfrage“ in der Symbolleiste.

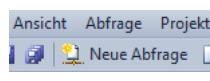
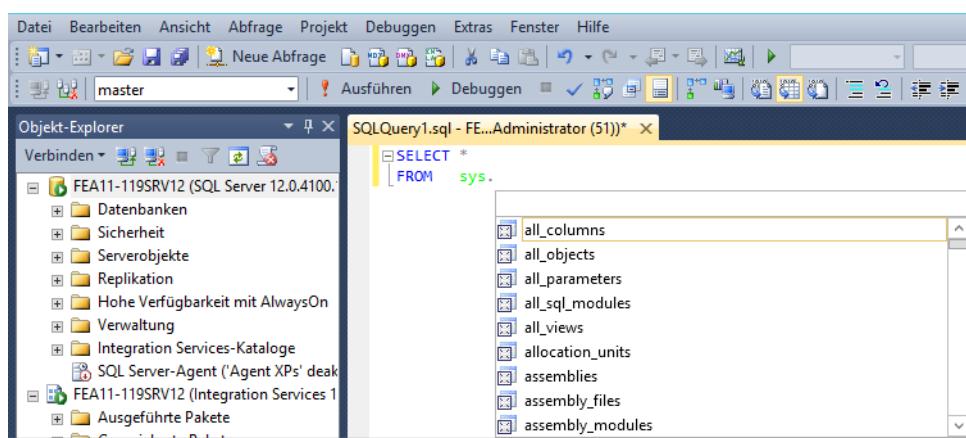


Abb. 16.13:
Die Schaltfläche
„Neue Abfrage“

Der Abfrageeditor bietet verschiedene Funktionen an, die dem Anwender das Arbeiten sehr erleichtern können. Dies sind beispielsweise IntelliSense oder Syntax Highlighting.

Wird eine Abfrage durch das Drücken der Taste F5 oder durch das Anklicken der Schaltfläche AUSFÜHREN ausgeführt, so werden die Ergebnisse unten, im Fenster „Ergebnisse“, angezeigt.



The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. On the left, the Object Explorer displays the database structure of the 'master' database, including 'Datenbanken', 'Sicherheit', 'Serverobjekte', 'Replikation', 'Hohe Verfügbarkeit mit AlwaysOn', 'Verwaltung', 'Integration Services-Kataloge', and 'SQL Server-Agent ('Agent XPs' deaktiviert)'. Below these are 'FEA11-119SRV12 (Integration Services 1)' and 'Ausgeführte Pakete'. The central pane shows a query window titled 'SQLQuery1.sql - FE...Administrator (51)*' containing the following code:

```
SELECT *
FROM sys.
```

IntelliSense is visible as a dropdown menu listing various system objects from the 'sys.' schema, such as 'all_columns', 'all_objects', 'all_parameters', etc. The 'sys.' prefix is highlighted in green, indicating it is a schema name. The rest of the code is highlighted in yellow, indicating it is a SQL keyword or identifier.

Abb. 16.14:
IntelliSense und
Syntax
highlighting



Wenn der Abfrageeditor mehrere SQL-Statements enthält, führt das Drücken der Taste F5 dazu, dass alle Statements ausgeführt werden. Soll nur ein einzelnes Statement ausgeführt werden, muss dieses vorher markiert werden.

name	object_id	principal_id	schema
sp_MSalreadyhavegeneration	-1073624922	NULL	4
sp_MSwritemergeperfcounter	-1072815163	NULL	4
TABLE_PRIVILEGES	.1072372588	NULL	3
sp_replsetsyncstatus	.1071944761	NULL	4
sp_replshowcmds	.1070913306	NULL	4
sp_publishtdb	.1070573756	NULL	4
sp_addqueued_artinfo	.1068897509	NULL	4

name
sp_MSalreadyhavegeneration
sp_MSwritemergeperfcounter
TABLE_PRIVILEGES
sp_replsetsyncstatus
sp_replshowcmds
sp_publishtdb
sp_addqueued_artinfo

Abb. 16.15:
Die
Ergebnissansicht

Wie in Abbildung ?? zu sehen ist, können die Ergebnisse einer Abfrage in unterschiedliche Art und Weise angezeigt werden. Es existieren drei Möglichkeiten:

- Anzeige im Gitternetz
- Anzeige als Text
- Abspeichern in einer Textdatei

Gesteuert wird die Anzeigeart durch die drei Schaltflächen in der mitte der Symbolleiste. Vorausgewählt ist die Anzeige im Gitternetz.



Abb. 16.16:
IntelliSense und
Syntax
highlighting

16.2.4 Aufbau des SSMS - Der Projektmappen-Explorer

Den Projektmappen-Explorer einblenden

Der Projektmappen-Explorer ist ein zusätzliches Tool, welches nicht standardmäßig eingeblendet ist. Um ihn anzuzeigen kann der Shortcut STRG + ALT + L oder das Menü ANSICHT genutzt werden.



Die Funktion des Projektmappen-Explorers wird, laut Aussage von Microsoft ab dem SQL Server 2014, nicht mehr unterstützt werden!

Aufbau des Projektmappen-Explorers

Der Projektmappen-Explorer untergliedert sich in Projektmappen. Eine Projektmappe stellt einen Kontainer für Projekte dar. Projekte können wiederum SQL-Skripte, Abfragen, Datenbankverbindungen oder Textdateien enthalten.

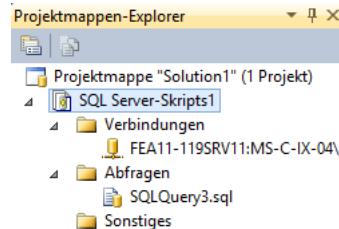


Abb. 16.17:
Der Aufbau des
Projektmappen-
Explorers

Einsatzzweck des Projektmappen-Explorers

Die Verwendung des Projektmappen-Explorers bringt folgende Vorteile mit sich:

- Zusammengehörende Elemente können in einem gemeinsamen Kontainer gespeichert werden (z. B. SQL-Skripte und Datenbankverbindungen)
- Die enthaltenen Dateien werden automatisch in einer sinnvollen Verzeichnisstruktur abgelegt.
- Projektmappen und Projekte können auch in anderen Microsoftprodukten (z. B. Microsoft Visual Studio) verwaltet und bearbeitet werden.
- Projektmappen können in ein Quellcodeverwaltungssystem eingefügt werden (z. B. Microsoft Source Safe)

16.3 Die SQL Server Systemdatenbanken

16.3.1 Die MASTER-Datenbank

Die Datenbank **MASTER** ist die wichtigste, innerhalb des SQL Servers. Sie ist der zentrale Speicherort für instanzweite Metadaten, wie z. B. Anmeldeinformationen oder Verbindungsserver. Seit SQL Server 2008 werden dort nur noch Metadaten und keine Objekte mehr gespeichert. Systemviews beispielsweise, werden in der Resource-Datenbank **MSSQLSYSTEMRESOURCE** abgelegt.



Ohne seine MASTER-Datenbank kann der SQL Server nicht gestartet werden.

Physikalische Eingeschafoten

Die MASTER-Datenbank besteht aus zwei Dateien: Der Datendatei `master.mdf` (5 MB), die sich in der primären Dateigruppe befindet und der Log-Datei `master.ldf` (2 MB). für beide Dateien ist ein automatisches Wachstum konfiguriert.

Einschränkungen bei der Administration

für die MASTER-Datenbank gelten verschiedene administrative Einschränkungen. Eine vollständige Liste kann dem Microsoft Technet entnommen werden.

- Es können keine Dateien oder Dateigruppen hinzugefügt werden
- Der Datenbankbesitzer kann nicht geändert werden (Besitzer ist der Nutzer SA)
- Die Datenbank kann nicht gelöscht werden
- Die Datenbank kann nicht umbenannt werden
- Die primäre Dateigruppe kann nur mit einem speziellen Verfahren wiederhergestellt werden.

Empfehlungen

Da die MASTER-Datenbank eine enorm wichtige Rolle spielt, empfiehlt Microsoft, nach den folgenden Operationen eine Sicherheitskopie dieser Datenbank anzulegen.

- Hinzufügen, Ändern oder Löschen von Datenbanken
- Hinzufügen, Ändern oder Löschen von Anmeldungen (Logins)
- Ändern von Server- oder Datenbankkonfigurationswerten



- [ms187837]

16.3.2 Die MSDB-Datenbank

Die MSDB-Datenbank spielt im Vergleich zur MASTER-Datenbank nur eine untergeordnete Rolle. Sie sammelt Informationen über Datenbank-Backups, die sogenannten „Sicherungsverläufe“.

Das SQL Server Management Studio verwendet diese Informationen dann, um im Falle dessen, dass ein Recovery notwendig wird, dem Nutzer einen Vorschlag für den Recovery-Vorgang unterbreiten zu können.

Physikalische Eingeschafeten

Die MSDB-Datenbank besteht aus zwei Dateien: Der Datendatei MSDBData.mdf (17 MB), in der sich die primäre Dateigruppe befindet und der Log-Datei MSDBLog.ldf (20 MB). für beide Dateien ist ein automatisches Wachstum konfiguriert.

Einschränkungen bei der Administration

für die MSDB-Datenbank gelten eine Reihe von Einschränkungen bei der Administration. Eine vollständige Liste dieser Einschränkungen kann dem Microsoft Technet entnommen werden. Hier sollen nur die wichtigsten Einschränkungen aufgelistet werden.

- Die Datenbank kann nicht gelöscht werden
- Die Datenbank kann nicht umbenannt werden



- [ms187112]

16.3.3 Die MODEL-Datenbank

Die MODEL-Datenbank dient als Vorlage für die Erstellung von neuen Datenbanken. Jedesmal, wenn eine Datenbank erstellt wird, wird zuerst die MODEL-Datenbank kopiert und anschließend werden die vom Benutzer angegebenen Parameter/Datenbankoptionen geändert.



Da der SQL Server die TEMPDB bei jedem Neustart neu anlegt, muss die MODEL-Datenbank, die auch für die TEMPDB-Datenbank als Vorlage dient, immer vorhanden sein.

Es ist möglich, die MODEL-Datenbank an die eigenen Erfordernisse anzupassen, so dass zukünftig zu erstellende Datenbanken diese Änderungen reflektieren. Beispielsweise könnten direkt verschiedene Objekte, wie z. B. Tabellen oder Benutzer in die MODEL-Datenbank eingefügt werden, wenn diese immer in allen Datenbanken vorhanden sein sollen.



Eine Veränderung an der MODEL-Datenbank hat keine nachträglichen Auswirkungen auf bereits bestehende Datenbanken!

Physische Eigenschaften

Die MODEL-Datenbank besteht aus zwei Dateien: Der Datendatei `model.mdf` (5 MB) und der Log-Datei `modellog.ldf` (1 MB). Für beide Dateien ist ein automatisches Wachstum konfiguriert.

Einschränkungen bei der Administration

Für die MODEL-Datenbank gelten eine Reihe von Einschränkungen bei der Administration. Eine vollständige Liste dieser Einschränkungen kann dem Microsoft Technet entnommen werden. Hier sollen nur die wichtigsten Einschränkungen aufgelistet werden.

- Es können keine Dateien oder Dateigruppen hinzugefügt werden
- Der Datenbankbesitzer kann nicht geändert werden (Besitzer ist der Nutzer `dbo`)
- Die Datenbank kann nicht gelöscht werden
- Die Datenbank kann nicht umbenannt werden
- Die Primäre Dateigruppe kann nur mit einem speziellen Verfahren `recovered` werden.
- Erstellen von verschlüsselten Objekten, da der Verschlüsselungsschlüssel an die Datenbank gebunden ist. Somit könnte nur MODEL diese Objekte verwenden.



- [ms186388]

16.3.4 Die TEMP-Datenbank

Die TEMPDB-Datenbank ist, wie ihr Name schon sagt, ein Ablageort für temporäre Objekte. Sie steht allen Benutzer zur Verfügung und enthält sowohl Benutzer- als auch Systemobjekte. Des Weiteren wird sie seit SQL Server 2005 auch für die Versionierung von Tabellenzeilen herangezogen. Wie bereits erwähnt, wird die TEMPDB-Datenbank bei jedem Neustart des SQL Server neu erstellt, so dass immer eine bereinigte Version von ihr zur Verfügung steht.



für die TEMPDB-Datenbank sind keine Sicherungsvorgänge zulässig, da temporäre Objekte nicht gesichert werden können.

Physikalische Eingeschafeten

Die TEMPDB-Datenbank besteht aus zwei Dateien: Der Datendatei `tempdb.mdf` (8 MB) und der Log-Datei `templog.ldf` (1 MB). Für beide Dateien ist ein automatisches Wachstum konfiguriert.



Einschränkungen bei Administration und Nutzung

Für die TEMPDB-Datenbank gelten eine Reihe von Einschränkungen bei der Administration. Eine vollständige Liste dieser Einschränkungen kann dem Microsoft Technet entnommen werden. Hier sollen nur die wichtigsten Einschränkungen aufgelistet werden.

- Es können keine Dateien oder Dateigruppen hinzugefügt werden
- Der Datenbankbesitzer kann nicht geändert werden (Besitzer ist der Nutzer DBO)
- Die Datenbank kann nicht gesichert werden.
- Die Datenbank kann nicht gelöscht werden
- Die Datenbank kann nicht umbenannt werden

- Die primäre Dateigruppe kann nicht, wie bei einer anderen Datenbank wiederhergestellt, werden. Hierfür ist ein spezielles Verfahren notwendig.
- Erstellen von verschlüsselten Objekten, da der Verschlüsselungsschlüssel an die Datenbank gebunden ist. Somit könnte nur MODEL diese Objekte verwenden.

Jeder Benutzer kann nur die temporären Objekte benutzen, die er selbst angelegt hat oder für die er vom System zusätzliche Berechtigungen erhalten hat. Es ist theoretisch sogar möglich, einem Benutzer den Zugriff auf die TEMPDB-Datenbank zu verweigern, was aber in der Praxis dazu führt, dass verschiedene Standardroutinen des SQL Server dann für diesen Benutzer nicht mehr funktionieren.



- [ms190768]

16.3.5 Die DISTRIBUTION-Datenbank

Die DISTRIBUTION-Datenbank spielt eine sehr spezielle Rolle unter den Systemdatenbanken. Sie speichert Informationen zu allen Arten von Replikationen, in der SQL Server als Verteiler (Distributor) eingebunden ist.

Die DISTRIBUTION ist die einzige Systemdatenbank, deren Name frei wählbar ist. DISTRIBUTION ist lediglich der Standardname, sofern nichts Anderes angegeben wird.



- [ms183524]

16.3.6 Die MSSQLSYSTEMRESOURCE-Datenbank

Hierbei handelt es sich um eine schreibgeschützte Datenbank, die seit den Tagen von SQL Server 2005 existiert. Sie enthält alle Systemobjekte (Views und Tabellen), die ursprünglich einmal in der MASTER-Datenbank abgelegt waren. Durch diese Auslagerung wurden Updates am SQL Server deutlich vereinfacht, da bei Änderungen an Systemobjekten einfach die komplette MSSQLSYSTEMRESOURCE-Datenbank ausgetauscht werden kann. Die MASTER-Datenbank bleibt unberührt.

Eine weitere Besonderheit der Resourcendatenbank ist, dass ihre Systemobjekte zwar physikalisch in ihr gespeichert sind, dass aber logische Verknüpfungen in alle anderen Datenbanken existieren. Das heißt, dass z. B. die View SYS.DATABASES in der MSSQLSYSTEMRESOURCE-Datenbank gespeichert, aber auch in jeder anderen Datenbank verfügbar ist.



Die **MSSQLSYSTEMRESOURCE**-Datenbank darf keinerlei Benutzerobjekte enthalten und wird deshalb auch nicht im SSMS angezeigt.

Physikalische Eingeschafeten

Die **MSSQLSYSTEMRESOURCE**-Datenbank besteht aus zwei Dateien: Aus der Datendatei `mssqlsystemresource.mdf` und aus der Log-Datei `mssqlsystemresource.ldf`. Da es pro SQL Server-Instanz nur eine Resourcendatenbank gibt, liegt sie zentral im Verzeichnis `C:\Programme\Microsoft SQL Server\MSSQL12.<instance_name>\MDF`.



- [ms190940]

16.3.7 Der Integration Services Katalog - SSISDB

Die SQL Server Integration Services, kurz SSIS, stellen ein mächtiges Werkzeug für sogenannte ETL-Prozesse - die Abkürzung ETL steht für „Extract Transform and Load“ - dar. SSIS kann als eine Art Programmiersprache betrachtet werden. Die in SSIS erstellten Programme werden als „Pakete“ bezeichnet und nach ihrer Erstellung auf alle SQL Server verteilt, wo sie zum Einsatz kommen sollen.

Um die Konfiguration und Bereitstellung von SSIS-Paketen zu vereinfachen, führte der SQL Server 2012 die SSISDB, auch „SSIS-Katalog“ genannt, ein. Mit ihm kommt das „Project Deployment Model“, eine neue Form der Bereitstellung von SSIS-Programmen. Wählen Pakete im Dateisystem abgelegt und durch eine Konfigurationsdatei konfiguriert wurden, werden Projekte im SSIS-Katalog gespeichert und auch deren Konfiguration wird dort abgelegt. Somit entsteht ein zentraler Punkt für SSIS-Programme und deren Konfiguration.

Weitere Vorteile der Nutzung des SSIS-Kataloges sind:

- Durch die Möglichkeit den Katalog in Ordner zu unterteilen, kann eine Berechtigungsstruktur aufgebaut werden. Projekte werden in Ordner gespeichert und Benutzer können Rechte auf Ordner erteilt werden.
- Mehrere Pakete können in einem Projekt zusammenarbeiten, was eine bessere Unterteilung großer SSIS-Programme ermöglicht.

- Projekte und Pakete können Parametrisiert werden um Projekt- und Paketeigenschaften während der Ausführung zu verändern.
- „Serverumgebungen“, „Servervariablen“ und „Serverumgebungsverweise“ ermöglichen es, dass ein Projekt auf den Zustand des SQL Servers, auf dem es ausgeführt wird, reagiert.
- Zentrales Logging vereinfacht das Debugging von Paketen.

• [hh479588]



16.4 Konfigurieren der Servereigenschaften

Die Konfiguration der Eigenschaften einer SQL Server-Instanz ist leider nicht an einer zentralen Stelle möglich. Bisher wurde der SQL Server Configuration Manager genutzt, um z. B. die Eigenschaften der SQL Server-Dienste zu konfigurieren, oder um die Netzwerkeinstellungen der Instanz zu verändern. Es gibt jedoch noch weitere Einstellungen, die nicht über den Configuration Manager geändert werden können, sondern über das SSMS.

Alle Einstellungen, die nicht über den Configuration Manager geändert werden können, werden im Management Studio über den „Servereigenschaften-Dialog“ beeinflusst. Geöffnet wird dieser, durch das Kontextmenü der betreffenden SQL Server-Instanz.

16.4.1 Die Grundeinstellungen

Die Seite „Allgemein“ der Servereigenschaften liefert einen Überblick über verschiedene, unveränderliche Werte der SQL Server-Instanz.

16.4.2 Konfiguration des Arbeitsspeichers

Auf dieser Seite des Servereigenschaften-Dialogs kann der Arbeitsspeicherkonsum der SQL Server Instanz beeinflusst werden. Die beiden Werte „Minimaler Serverarbeitsspeicher (in MB)“ und „Maximaler Serverarbeitsspeicher (in MB)“ geben an, wie viel Arbeitsspeicher für die Instanz reserviert wurde bzw. wie viel sie höchstens verbrauchen kann.

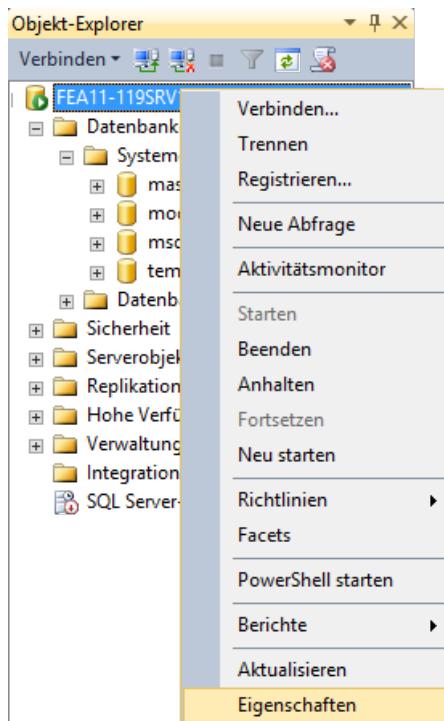


Abb. 16.18:
Aufruf der Server-eigenschaften
einer Instanz

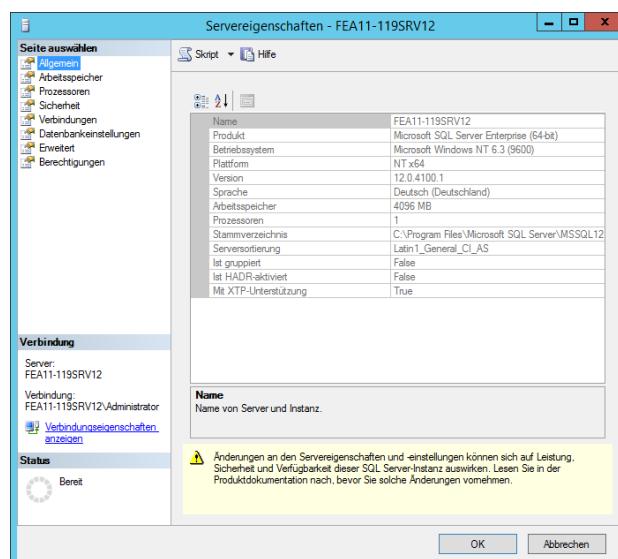


Abb. 16.19:
Seite „Allgemein“
der Servereigen-schaften

💡 Wird für die Option „Minimaler Serverarbeitsspeicher“ ein Wert konfiguriert, bedeutet dies nicht, dass sich die Instanz von Anfang an so viel Arbeitsspeicher nimmt. Es handelt sich vielmehr um eine Zusicherung, dass die Instanz so viel Arbeitsspeicher im Minimum bekommen kann.

Gemäß einer Empfehlung von Microsoft sollten die beiden Optionen minimaler und maximaler Serverarbeitsspeicher immer den gleichen Wert haben, um Verwaltungsarbeiten im RAM zu minimieren.



Ein Wert von 2147483648 beim „Maximalen Serverarbeitsspeicher“ bedeutet, dass der Instanz keine Obergrenze gesetzt wurde.

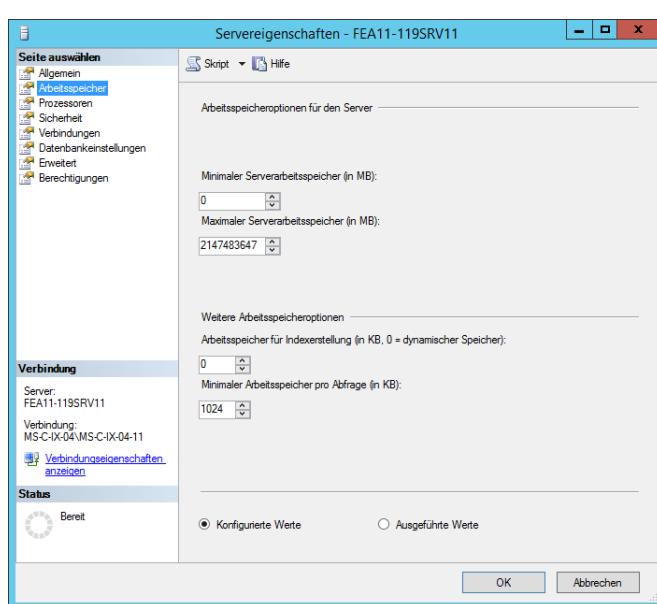


Abb. 16.20:
Seite
„Arbeitsspeicher“
der Servereigen-
schaften

Zu beachten sind in diesem Fenster noch der „Skript“-Button, oben links und die beiden Optionsfelder am unteren Rand. „Konfigurierte Werte“ bedeutet, dass im Fenster die Werte zu sehen sind, die der Nutzer eingestellt hat. Nach einem Umschalten auf „Ausgeführte Werte“ werden die aktuell gültigen Verbrauchswerte der Instanz angezeigt.

Mit Hilfe des „Skript“-Buttons können die im Dialog vorgenommenen Einstellungen in ein SQL-Skript verwandelt werden. Dies ermöglicht das Erlernen der T-SQL-Syntax und auch die Speicherung der Konfiguration einer Instanz in Form eines SQL-Skripts.

16.4.3 Die Prozessoreinstellungen

Die Seite „Prozessoren“ bietet die Möglichkeit, einer SQL Server Instanz bestimmte Prozessoren zu zuweisen (Prozessoraffinität). Dies kann aus zweierlei Gründen nützlich sein:

- Wenn mehrere Instanzen des SQL Server auf einem Computer laufen, kann jede ihre eigenen Prozessoren erhalten, so dass kein „Streit um die Ressourcen“ entsteht.
- Weiterhin kann so verhindert werden, dass das Betriebssystem einen Thread von einem Prozessor auf einen anderen verschiebt, um zusätzlicher Aufwand beim Laden und Entladen der Prozessorcaches zu vermeiden.



Da in Windows Server 2008 die Threadverwaltung grundlegend geändert wurde, soll die Prozessoraffinitätseinstellung in künftigen SQL Server-Versionen nicht mehr existieren.

Da die Einstellung der Prozessoraffinität dynamisch erfolgt, muss der SQL Server nicht neugestartet werden.



- [ms187104]

Eine zweite Einstellungsmöglichkeit ist die „E/A-Affinität“. Mit dieser Option werden die I/O-Prozesse der Instanz an bestimmte Prozessoren gebunden. Der Hintergrund dieser Option ist, wie auch schon bei der Prozessoraffinität, dass die Verschiebung von Threads durch das Betriebssystem von einem Prozessor auf einen anderen verhindert werden soll. In den meisten Fällen, muss die E/A-Affinität nicht konfiguriert werden, weil durch die Standardeinstellung bereits optimale Ergebnisse erzielt werden.



Laut Microsoft sollten niemals die Prozessoraffinität und die E/A-Affinität auf einem Prozessor gleichzeitig aktiviert sein

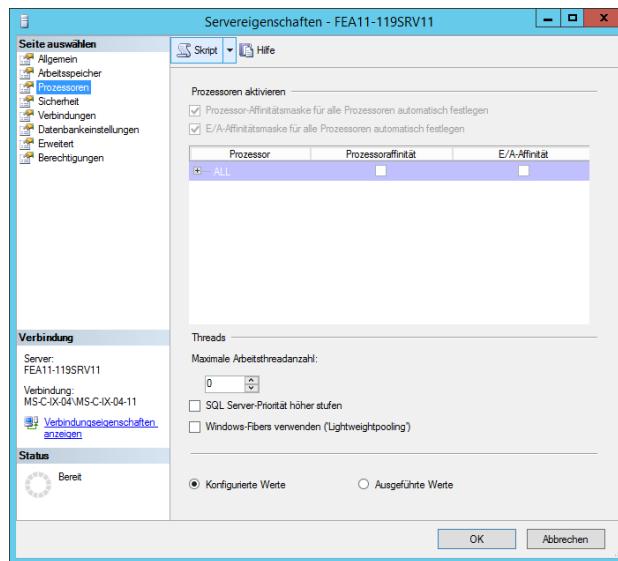
Die Option „Maximale Arbeitsthreadanzahl“ ist für Systeme mit einer sehr hohen Nutzerzahl gedacht. Wird hier ein Wert ungleich 0 eingestellt, kann die Anzahl der Workerthreads für die Instanz begrenzt werden, um eine Überlastung des Servers zu verhindern.

Wird ein Haken bei „SQL Server-Priorität höher stufen“ gesetzt, behandelt das Betriebssystem die Threads des SQL Server vorrangig, was eine Leistungssteigerung beim SQL Server erzielen kann. Hierdurch kann es jedoch auch geschehen, dass anderen Prozessen so viel Leistung entzogen wird, dass diese kaum mehr arbeitsfähig sind.



- [ms189629]

Abb. 16.21:
„Prozessoren“ der
Servereigen-
schaften



16.4.4 Sicherheitseinstellungen

Die Seite „Sicherheit“ besitzt vier verschiedene Optionsgruppen, wobei die Gruppe „Serverauthentifizierung“ die wichtigste ist. Bei der Installation des SQL Servers musste eine Auswahl getroffen werden, welche Form der Authentifizierung von der Instanz zugelassen wird. Es standen die Windows-Authentifizierung und die gemischte Authentifizierung (Windows- + SQL-Authentifizierung) zur Auswahl. Dies kann hier geändert werden.



Das Ändern der Serverauthentifizierung erfordert einen Neustart des SQL Server-Dienstes!

Die beiden Optionsgruppen „Anmeldungsüberwachung“ und „Serverproxykonto“ sind nur noch aus Kompatibilitätsgründen vorhanden. Mit dem SQL Server 2008 wurden diese beiden Dinge durch völlig neue Technologien ersetzt.

- [ms188470]



16.4.5 Verbindungseinstellungen

Die erste Option auf dieser Seite, „Maximale Anzahl von gleichzeitigen Verbindungen“, legt fest, wie viele Benutzer sich mit einer SQL Server-Instanz verbinden können. Sie ist beispielsweise dann interessant,

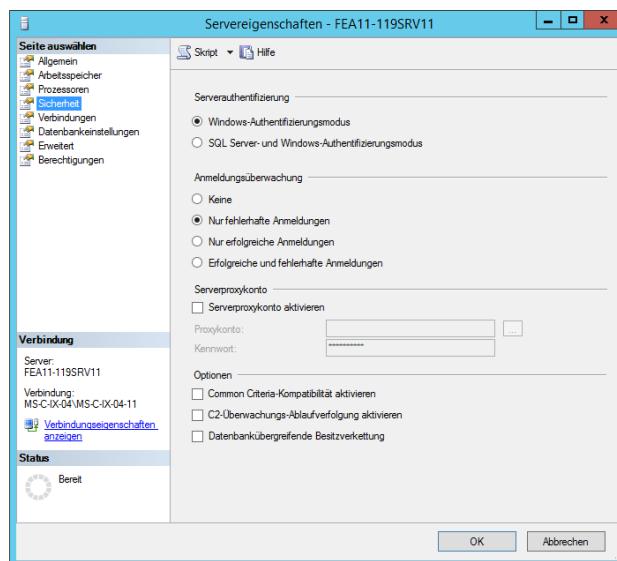


Abb. 16.22:
Seite „Sicherheit“
der Servereigen-
schaften

wenn mit Client-Access-Licenses (CAL) gearbeitet wird und die Anzahl der gleichzeitigen Anmeldungen die Anzahl der CALs nicht überschreiten darf.

Wird die Option „Abfragekontrolle verwenden, um Abfragen mit langer Ausführungszeit zu verhindern“ aktiviert, überwacht der SQL Server für alle Abfragen die Berechnungsdauer ihres Ausführungsplanes. Übersteigt diese Dauer die hier angegebene Zeitspanne, wird die Berechnung abgebrochen. Besonders kostenintensive Abfragen werden so unterbunden.



Zu beachten ist, dass hier nur die Zeitspanne für das Berechnen der Ergebnismenge gilt, nicht aber der Zeitraum, der für das Anzeigen der Ergebnismenge benötigt wird!

Mit den Standardverbindungsoptionen kann festgelegt werden, welche Besonderheiten für alle angemeldeten Clients gelten. Eine vollständige Liste mit den Bedeutungen der einzelnen Optionen, kann in der MSDN, unter dem Link [[ms180124](#)], nachgeschlagen werden.

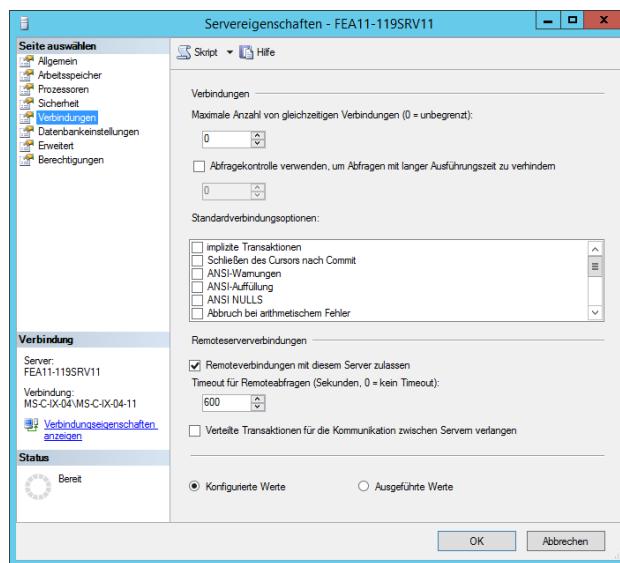


- [[ms180124](#)]

16.4.6 Datenbankeinstellungen

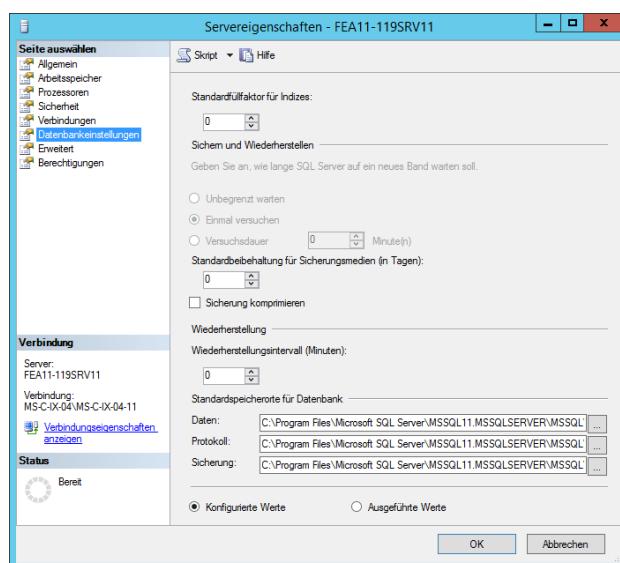
Fast alle Einstellungen auf dieser Seite werden nur selten geändert. Beispielsweise wird der „Standardfüllfaktor für Indizes“, oft nur für bestimmte Indizes angepasst, da der Standardwert 0 meist optimal ist. Auch

Abb. 16.23:
Seite „Verbindungen“
der Servereigen-
schaften



die Einstellungen der Gruppe „Sichern und Wiederherstellen“ werden so gut wie nie allgemein geändert, sondern für alle Sicherungen getrennt.

Abb. 16.24:
Seite „Datenbank-
einstellungen“ der
Servereigen-
schaften



- [ms178521]



16.4.7 Erweiterte Einstellungen

Unter den erweiterten Einstellungen sind zwei, die eine besondere Erwähnung verdienen. „Eigenständige Datenbank aktivieren“ und „FILESTREAM-Zugriffsebene“. Beide werden bei den Themen „Filestream“ bzw. „Partially contained databases“ näher besprochen.

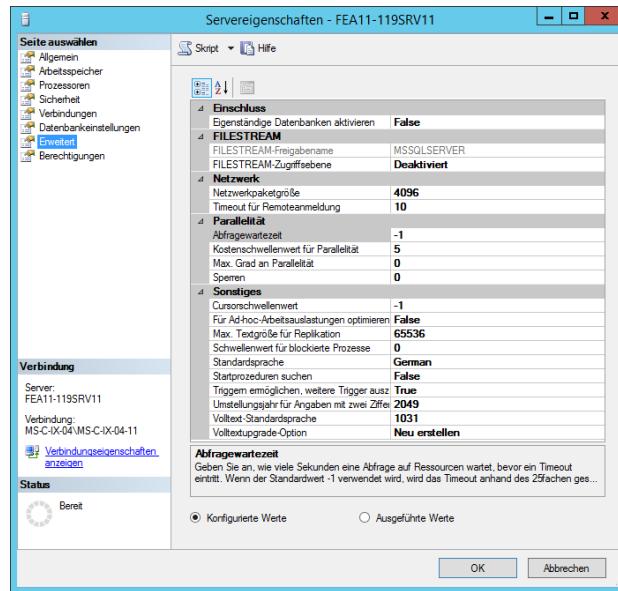


Abb. 16.25:
Seite „Erweitert“
der Servereigen-
schaften

Unter der Rubrik „Netzwerk“ kann die Netzwerkpaketgröße und der Remotetimeout angepasst werden. Eine Veränderung der Netzwerkpaketgröße kann, abhängig von der Art und Weise, wie auf den SQL Server zugegriffen wird, sinnvoll sein. Wenn z. B. hauptsächlich Datenimports getätigter werden, kann eine höhere Paketgröße die Importvorgänge beschleunigen. Bei normalen Lese-/Schreibzugriffen kann sich eine Vergrößerung der Netzwerkpakete aber auch negativ auswirken, da die zu transportierenden Datenmengen viel zu klein für die großen Pakete sind.

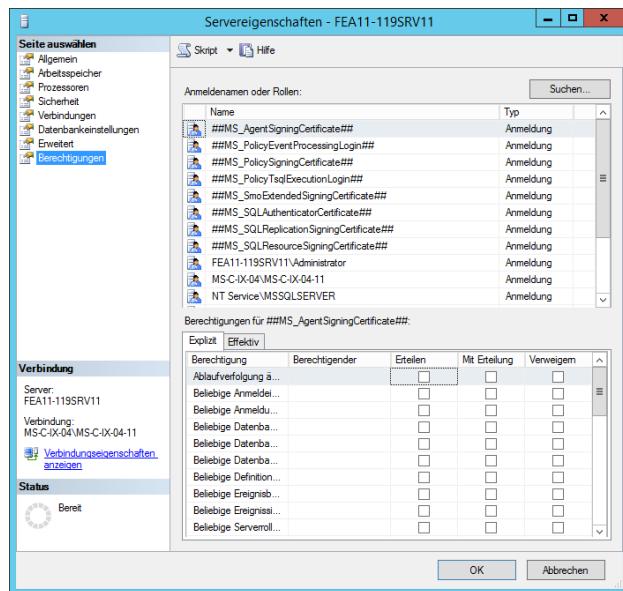


- [ms189357]

16.4.8 Berechtigungen

Hier können den existierenden Logins Berechtigungen auf Server-Ebene erteilt werden. Nähere Informationen zu Logins und Berechtigungen werden im Kapitel „Sicherheit im SQL Server“ bereitgestellt.

Abb. 16.26:
Seite „Datenbank-einstellungen“ der
Servereigen-schaften



16.5 Die gespeicherte Systemprozedur sp_configure

SP_CONFIGURE ist eine standardmäßig vorhandene gespeicherte Systemprozedur, die es ermöglicht, die Servereigenschaften mit Hilfe von T-SQL anzuzeigen oder zuverändern.

16.5.1 Basisoptionen und erweiterte Optionen

SQL Server unterteilt seine Konfigurationsoptionen in zwei Gruppen:

- **Basisparameter:** Hierbei handelt es sich um eine Liste von 18 Parametern, die sehr häufig Änderungen erfahren.
- **Erweiterte parameter:** Dies sind 52 weitere Parameter, die laut Empfehlung von Microsoft, nur von erfahrenen Administratoren oder ausgebildeten SQL Server Technikern geändert werden sollten.

Standardmäßig zeigt SP_CONFIGURE nur die Basisparameter an. Um einen erweiterten Parameter zu sehen bzw. zu ändern, muss der SQL Server erst dementsprechend konfiguriert werden.

16.5.2 Ändern der Konfiguration mit sp_configure

Die allgemeine Syntax für die Anwendung der Systemprozedur SP_CONFIGURE lautet folgendermaßen:

Listing 16.1: Die Syntax von sp_configure

```
sp_configure [[@configname = ] 'option_name'] , [@configvalue = ] 'value' ]]
```

Beispiel ?? zeigt die Benutzung von `sp_configure` zur Änderung der Serverkonfigurationsoption `backup compression default`:

Listing 16.2: Ein erstes Beispiel zur Nutzung von `sp_configure`

```
EXEC sp_configure @configname = 'backup compression default',
                  @configvalue = '1'
```

Das gleiche Kommando kann auch in einer Kurzform ausgeführt werden, indem man die Parameterbezeichner `@configname` und `@configvalue` nicht mit angibt.

Listing 16.3: Die Kurzform der Syntax für `sp_configure`

```
EXEC sp_configure 'backup compression default', '0'
```

Die Tatsache, dass der SQL Server auf diese beiden Statements mit einer Fehlermeldung antwortet zeigt, dass die Konfigurationseinstellung bisher noch nicht wirksam geworden ist.



Die Konfigurationsoption `'backup compression default'` wurde von 1 in 0 geändert. Führen Sie zum Installieren die RECONFIGURE-Anweisung aus.

für jede Konfigurationseinstellung gibt es zwei Werte:

- Den konfigurierten Wert. Dieser wird mit Hilfe von `SP_CONFIGURE` geändert.
- Den aktiven Wert. Dieser wird entweder durch die Anweisung `RECONFIGURE` oder durch einen Neustart der Instanz geändert.

Um die Änderung aus Beispiel ?? bzw. Beispiel ?? zu aktivieren, muss also noch das Kommando `RECONFIGURE` ausgeführt werden.

Listing 16.4: Ein erstes Beispiel zur Nutzung von `sp_configure`

```
EXEC sp_configure 'backup compression default', '1'
RECONFIGURE
```



Vor der Ausführung des `RECONFIGURE`-Kommandos wird der SQL Server Plan cache komplett geleert, d. h. alle Ausführungspläne gehen verloren und müssen neu erarbeitet werden. Dies kann in einer Produktionsumgebung zu einem massiven Performanceeinbruch führen.

16.5.3 Basiskonfigurationseinstellungen anzeigen

Wird die Prozedur SP_CONFIGURE ohne Parameter aufgerufen, zeigt sie die aktuellen Basisserverkonfigurationseinstellungen an.

Listing 16.5: sp_configure zur Anzeige der Basiskonfigurationswerte nutzen

```
EXEC sp_configure
```



name	minimum	maximum	config_value	run_value
allow updates	0	1	0	0
backup compression default	0	1	0	0
clr enabled	0	1	0	0
contained database authentication	0	1	0	0
cross db ownership chaining	0	1	0	0
default language	0	9999	1	1
filestream access level	0	2	0	0
max text repl size (B)	-1	2147483647	65536	65536
nested triggers	0	1	1	1
remote access	0	1	1	1
remote admin connections	0	1	0	0
remote login timeout (s)	0	2147483647	10	10
remote proc trans	0	1	0	0
remote query timeout (s)	0	2147483647	600	600
server trigger recursion	0	1	1	1
show advanced options	0	1	0	0
user options	0	32767	0	0

16.5.4 Erweiterte Konfigurationseinstellungen abfragen

Um erweiterte Konfigurationsoptionen sehen bzw. ändern zu können, muss die Einstellung `show advanced options` auf den Wert 1 eingestellt werden.

Listing 16.6: Einblenden der erweiterten Konfigurationsoptionen

```
EXEC sp_configure 'show advanced options', '1'
RECONFIGURE
```

Ein Aufruf von SP_CONFIGURE zeigt nun eine Liste mit insgesamt 70 Parametern an. Um die erweiterten Parameter wieder auszublenden, muss `show advanced options` auf den Wert 0 zurückgesetzt werden.

- [ms176069]
 - [ms188787]
- 

16.6 Datengewinnung - Systemsichten

Jedes Datenbank Management System bietet dem Administrator einen Weg, um Daten über den Server, die Datenbanken, die Benutzer und viele andere Dinge zu sammeln. In Microsoft SQL Server werden dem Admin Views, die sogenannten Systemsichten, zur Verfügung gestellt, um die gewünschten Daten abfragen zu können.

16.6.1 Katalogsichten

Katalogsichten existieren seit SQL Server 2005 und liefern Informationen, die aus den verschiedenen Teilen der Software stammen (Caches, Pools, SQLOS, usw.). Durch sie kann sich der Administrator ein umfassendes Bild über die Instanz machen, an der er arbeitet. Microsoft baut mit den Katalogsichten ein „Data Dictionary“, gemäß dem SQL-Standard, auf. Mit jeder Version von SQL Server wächst die Anzahl der Katalogsichten. Außerdem verändern sich manche Sichten auch von Version zu Version.

Die Bezeichner der Katalogsichten sind nach einem einfachen Schema aufgebaut.

- Alle Katalogsichten beginnen im Namen mit der Zeichenfolge SYS.
- Der Name lässt immer auf die enthaltenen Informationen schließen.

Beispielsweise bietet die Katalogsicht SYS.DATABASES Infos zu allen Datenbanken innerhalb einer SQL Server-Instanz. Im Verlauf dieser Unterrichtshilfe werden immer wieder die wichtigsten Katalogsichten zum jeweiligen Thema angezeigt.



Die Katalogsichten sind in der Resourcen-Datenbank gespeichert und werden in alle anderen Datenbanken projiziert.



- [ms174365]

16.6.2 Informationsschemasichten

Diese Views haben den gleichen Zweck wie Katalogsichten. Sie zeigen wichtige Informationen über die Instanz und ihre Datenbanken an. Der Unterschied zu den Katalogsichten liegt zum einen darin, dass der Aufbau der Informationsschemasichten im ANSI SQL-Standard genau definiert ist, während die Katalogsichten von den Microsoftentwicklern frei gestaltet werden können. Zum anderen sind die Informationsschemasichten nicht in der Resourcen Datenbank gespeichert, sondern in den Datenbanken selbst.



- [ms186778]

16.6.3 Kompatibilitätssichten

Kompatibilitätssichten dienen, wie ihr Name schon sagt, nur zur Aufrechterhaltung der Kompatibilität. In SQL Server 2000 wurden dem Administrator „Systemtabellen“ zur Verfügung gestellt, deren Aufgabe heute Katalog- und Informationsschemasichten übernommen haben. Damit aber auch noch alte SQL Server 2000-Anwendungen funktionieren, bietet Microsoft mit den Kompatibilitätssichten einen temporären Ausweg.

Sie sind daran zu erkennen, dass ihr Name immer mit SYS beginnt. Zu jeder Kompatibilitätssicht existiert auch eine aktuelle Katalogsicht, beispielsweise gibt es zur View SYSUSERS die Katalogsicht SYS.DATABASE_PRINCIPALS.

Um veraltete Anwendungen auf den neuesten Stand zu bringen, bietet Microsoft in der MSDN eine Tabelle zur Zuordnung von Systemtabellen zu Katalogsichten an ([ms187997]).



- [ms187376]

16.6.4 Dynamische Verwaltungssichten und -funktionen

Dynamische Verwaltungssichten und -funktionen unterscheiden sich von Katalog- und Informationsschemasichten darin, dass sie ihre Daten nicht aus Tabellen generieren, sondern dass sie direkt aus dem SQL Server oder dem Betriebssystem abgreifen. Aus diesem Grund werden sie als dynamische Views bezeichnet, weil ihre Informationen immer ohne Verzögerung direkt aus dem System abgerufen werden.

Mit ihrer Hilfe ist es möglich, interne, versionsspezifische Statusinformationen über den SQL Server abzufragen, um so nach Problemen zu suchen.

Die Namenskonvention für dynamische Verwaltungssichten und -funktionen sieht vor, dass die Bezeichner immer mit SYS.DM_ beginnen. Auch für diese Gruppe von Views werden im Verlauf dieser Unterrichtsunterlage immer wieder die wichtigsten Vertreter zum jeweiligen Thema genannt.



- [ms188754]

16.7 Datenbanken erstellen

Microsoft SQL Server ermöglicht das Erstellen von Datenbanken auf zwei unterschiedlichen Wegen: Mit Hilfe des SQL Server Management Studios, oder mit dem SQL-Kommando `CREATE DATABASE`. Um eine neue Datenbank anlegen zu können, muss der Benutzer:

- ein Mitglied der festen Serverrolle `SYSADMIN` sein,
- oder eines der beiden Serverrechte `CONTROL` bzw. `ALTER`,
- oder das Serverrecht `CREATE DATABASE` haben.

Um eine neue Datenbank anzulegen kopiert SQL Server einfach die `MODEL`-Datenbank, weshalb die primäre Datendatei der neuen Datenbank mindestens genauso groß sein muss, wie die der `MODEL`-Datenbank. Insgesamt darf eine Datenbank unter SQL Server 2014 nicht kleiner als 6 MB sein (primäre Datendatei 5 MB + Log Datei 1 MB). Für sekundäre Datendateien wird eine Mindestgröße von 1 MB angenommen, sofern der Benutzer keine Angabe macht.

16.7.1 Anlegen einer Datenbank mit dem SSMS

1. Klicken Sie im Objekt-Explorer mit der rechten Maustaste auf den Knotenpunkt „Datenbanken“. Es erscheint ein Kontextmenü. Dort wählen sie den Punkt „Neue Datenbank...“.
2. Es öffnet sich der Dialog „Neue Datenbank“.

Abb. 16.27:
Anlegen einer
neuen Datenbank

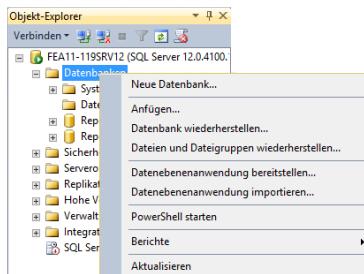
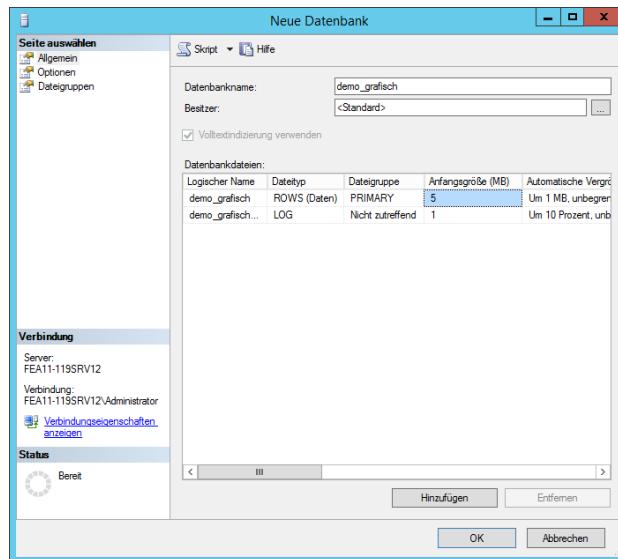


Abb. 16.28:
Der Dialog „Neue
Datenbank“



Auf der Seite „Allgemein“ dieses Dialogfensters können die folgenden Eigenschaften der neuen Datenbank geändert werden:

- **Datenbankname:** Ein innerhalb der SQL Server-Instanz eindeutiger Bezeichner für die neue Datenbank, der maximal 128 Zeichen lang sein darf. Der Datenbankname muss den Regeln für Bezeichner folgen.
- **Besitzer:** Eine SQL Server-Anmeldung, die als Besitzer der Datenbank eingetragen werden soll. Der Datenbankeigentümer hat innerhalb einer Datenbank uneingeschränkte Rechte. Dieser Wert sollte immer auf <Standard> belassen werden.
- **Logischer Name:** Der logische Name Datei ist der Bezeichner, mit dessen Hilfe in T-SQL auf die Datei zugegriffen wird. Er muss innerhalb der Datenbank eindeutig sein und, wie der Datenbankname auch, den Regeln für Bezeichner folgen.
- **Dateityp:** Gibt an, welche Art von Datei angelegt wird. Dies kann sein: Zeilendaten, Protokoll, FileStream.
- **Dateigruppe:** Der Bezeichner der Dateigruppe, zu der die angegebene Datendatei gehören soll. Die erste Dateigruppe hat immer den Bezeichner PRIMARY. Innerhalb dieser Dateigruppe befindet sich auch immer die primäre Datendatei.

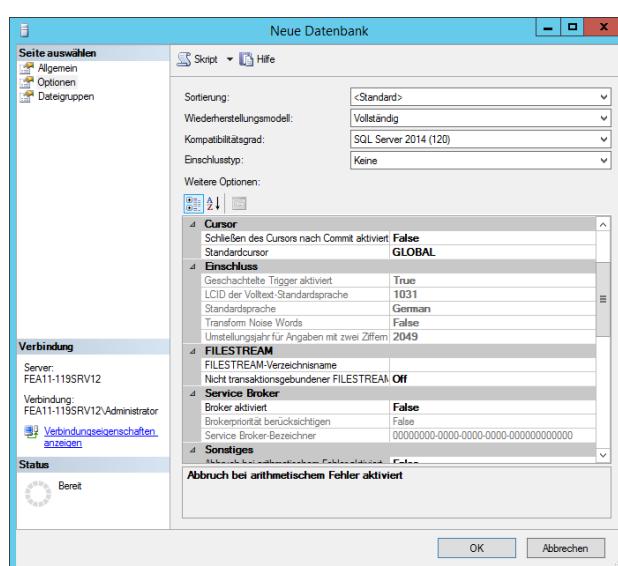
- **Anfangsgröße:** Legt die anfängliche Größe der Daten- und Log-Dateien fest. Es können die Größenangaben KB, MB, GB und TB benutzt werden. Ohne Größenangabe wird automatisch MB angenommen. Der maximalwert in diesem Feld ist 2.147.483.648.
- **Automatische Vergrößerung/Maximale Größe:** Ermöglicht die Konfiguration eines automatischen Wachstums für Datendateien (Wachstumsrate und Größenbeschränkung). Die Standardwachstumsrate beträgt bei Datendateien 1 MB und bei Log-Dateien 10 % (der Prozentsatz bezieht sich immer auf die aktuelle Dateigröße, zum Zeitpunkt des Anwachsens). Die hier angegebene Größe wird immer auf den nächsten, durch 64 KB teilbaren Wert aufgerundet. Durch die Angabe des Wertes 0 wird angezeigt, das kein automatisches Wachstums stattfinden darf. Wird für die Größenbeschränkung kein Wert oder der Wert UNLIMITED angegeben, kann die Datei solange wachsen, bis die physikalischen Grenzen des Datenträger, auf dem sie sich befindet, erreicht sind.
- **Pfad + Dateiname:** Gibt den physikalischen Dateinamen vor. Die Datendatei darf nur auf einem lokalen Datenträger, auf einer SAN oder auf einem iSCSI-Laufwerk abgelegt werden. Der angegebene Pfad muss vor dem Ausführen der `CREATE DATABASE`-Anweisung bereits vorhanden sein.



Es sollte niemals ein komprimiertes Dateisystem für die Ablage von Daten- oder Log-Dateien genutzt werden.

3. Wechseln Sie in das Fenster „Optionen“.

Abb. 16.29:
Verändern der Datenbankoptionen



Hier können alle Datenbankoptionen einzeln gesetzt werden. Viele dieser Einstellungen werden noch im weiteren Verlauf dieser Unterrichtsunterlage behandelt. für eine vollständige Liste der Datenbankoptionen beachten Sie bitte den nächsten Literaturhinweis.

4. Wechseln Sie in das Fenster „Dateigruppen“. Auf dieser Seite können der Datenbank weitere Dateigruppen hinzugefügt werden. Der Haken in der Spalte „Standard“ sagt aus, dass alle Objekte, wie z. B. Tabellen in einer Datei, dieser Dateigruppe angelegt werden, sofern der Benutzer keine andere Angabe macht. Klicken Sie auf OK. Die Datenbank wird erstellt.

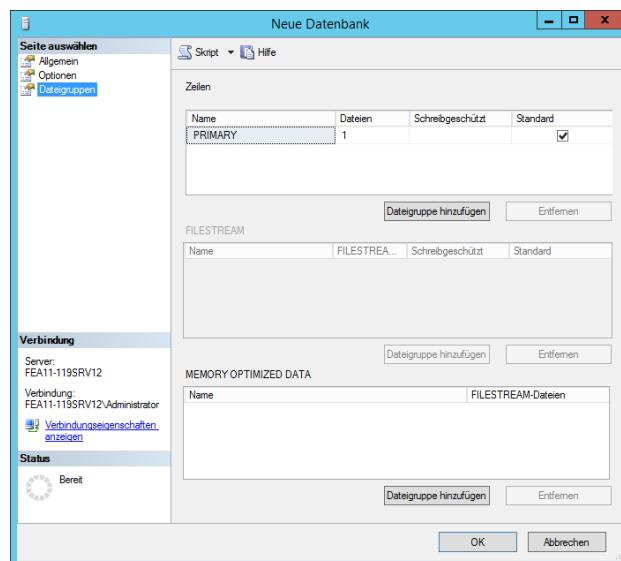


Abb. 16.30:
Anpassen der
Dateigruppen

- [ms187997]



16.7.2 Anlegen einer Datenbank mittels T-SQL (CREATE DATABASE)

Da der SQL Server beim Anlegen einer neuen Bank lediglich eine Kopie der MODEL-Datenbank erstellt, ist die einfachste Form des `CREATE DATABASE`-Statements ohne Parameter.

Listing 16.7: Die einfachste Form von `CREATE DATABASE`

```
CREATE DATABASE demo_sql;
```

Der einzige Wert, der zwingend angegeben werden muss, ist der Name der neuen Datenbank. Die primäre Dateigruppe und die Log-Datei sind 1:1-Kopien der MODEL-Datenbank. Um die primäre Dateigruppe zu beeinflussen, müssen die beiden Schlüsselwörter `ON PRIMARY`, zusammen mit weiteren Angaben angefügt werden.

Listing 16.8: Die Dateigruppe PRIMARY verändern

```
IF EXISTS (SELECT * FROM sys.databases WHERE name='demo_sql')
```

```

DROP DATABASE demo_sql

CREATE DATABASE demo_sql
ON PRIMARY (
    NAME      = 'SALES',
    FILENAME  = 'D:\data\demo_sql\sales01.mdf',
    SIZE      = 100MB,
    MAXSIZE   = 2GB,
    FILEGROWTH = 100MB
)

```

In Beispiel ?? wird eine Datenbank mit zwei Dateien angelegt:

- sales01.mdf: Die primäre Datendatei, mit einer Größe von 100 MB, einer maximalen Größe von 2 GB und einer Wachstumsrate von je 100 MB.
- demo_sql_log.ldf: die Log-Datei der Datenbank. Diese wird mit einer Größe von 25 MB, einer maximalen Größe von 2 TB und einer Wachstumsrate von 10 %. Der Name der Datei wurde aus dem Datenbanknamen und dem suffix _LOG zusammengesetzt.



Wenn kein logischer Name für die Log-Datei angegeben wird, darf der Datenbankname max. 123 Zeichen lang sein, damit das suffix _LOG an den Log-Dateinamen angehängt werden kann.

Listing 16.9: Eine Dateigruppe mit mehreren Dateien ausstatten und die Log-Datei verändern

```

IF EXISTS (SELECT * FROM sys.databases WHERE name='demo_sql')
DROP DATABASE demo_sql

CREATE DATABASE demo_sql
ON PRIMARY (
    NAME      = 'SALES01',
    FILENAME  = 'D:\data\demo_sql\sales01.mdf',
    SIZE      = 100MB,
    MAXSIZE   = 2GB,
    FILEGROWTH = 100MB
),
(
    NAME      = 'SALES02',
    FILENAME  = 'D:\data\demo_sql\sales02.ndf',
    SIZE      = 100MB,
    MAXSIZE   = 2GB,
    FILEGROWTH = 100MB
)
LOG ON (
    NAME      = 'DEMO_SQL_LOG',

```

```
FILENAME      = 'D:\data\demo_sql\demo_sql_log.ldf',
SIZE         = 64MB ,
MAXSIZE      = 1GB ,
FILEGROWTH   = 16MB
)
```

Beispiel ?? zeigt eine nochmals erweiterte Syntax, mit deren Hilfe die Dateigruppe Primary mit zwei Datenbankdatei, sales01.mdf und sales02.ndf ausgestatt wird. Des Weiteren werden die Größe, die Wachstumsrate und die Maximalgröße der Log-Datei verändert.

Im nächsten Beispiel wird gezeigt, wie eine Datenbank mit mehreren Dateigruppen erstellt werden kann. Zusätzlich zur immer existenten Gruppe PRIMARY wird die Dateigruppe CRM mit einer Datendatei angelegt.

Listing 16.10: Eine Dateigruppe mit mehreren Dateigruppen ausstatten und die Log-Datei verändern

```

IF EXISTS (SELECT * FROM sys.databases WHERE name='demo_sql')
DROP DATABASE demo_sql

CREATE DATABASE demo_sql
ON PRIMARY (
    NAME      = 'SALES01',
    FILENAME  = 'D:\data\demo_sql\sales01.mdf',
    SIZE       = 100MB ,
    MAXSIZE   = 2GB ,
    FILEGROWTH = 100MB
),
(
    NAME      = 'SALES02',
    FILENAME  = 'D:\data\demo_sql\sales02.ndf',
    SIZE       = 100MB ,
    MAXSIZE   = 2GB ,
    FILEGROWTH = 100MB
),
FILEGROUP CRM (
    NAME      = 'CRM01',
    FILENAME  = 'D:\data\demo_sql\crm01.ndf',
    SIZE       = 500MB ,
    MAXSIZE   = 16GB ,
    FILEGROWTH = 500MB
)
LOG ON (
    NAME      = 'DEMO_SQL_LOG',
    FILENAME  = 'D:\data\demo_sql\demo_sql_log.ldf',
    SIZE       = 64MB ,
    MAXSIZE   = 1GB ,
    FILEGROWTH = 16MB
)

```



- [ms176061]

16.8 Datenbanken bearbeiten und löschen

16.8.1 Die Datenbank demo_grafisch

In den folgenden Abschnitten und Kapiteln wird für alle weiteren Beispiele die SQL Server-Datenbank DEMO_GRAFISCH benutzt. Sie hat folgenden Aufbau:

Logischer Name	Dateityp	Dateigruppe	Anfangsgröße (MB)	Automatische Vergrößerung	Pfad + Dateiname
PRIMARY	Zeilendaten	PRIMARY	100	Um 100 MB, auf 2048 MB	D:\u01\demo_grafisch\data\primary_01.mdf
CRM	Zeilendaten	CRM	500	Um 100 MB, auf 4096 MB	D:\u01\demo_grafisch\data\crm01.ndf
SALES01	Zeilendaten	SALES	500	Um 100 MB, auf 4096 MB	D:\u01\demo_grafisch\data\sales01.ndf
SALES02	Zeilendaten	SALES	500	Um 100 MB, auf 4096 MB	D:\u01\demo_grafisch\data\sales02.ndf
LOGFILE	Protokoll	--	1	Um 10 %, auf 2097152 MB	D:\u01\demo_grafisch\data\demo_grafisch_log.ldf

16.8.2 Die Eigenschaften einer Datenbank verändern

Die Eigenschaften einer Datenbank können entweder mit Hilfe des SSMS oder durch das SQL-Kommando `ALTER DATABASE` geändert werden. Die folgende Liste zeigt ausschnittsweise, was an einer Datenbank, nach ihrer Erstellung, geändert werden kann.

- Ändern des Datenbanknamens
- Hinzufügen von Datendateien und Dateigruppen
- Hinzufügen von Log-Dateien
- Ändern von Dateieigenschaften (Größe, Wachstumsrate, Maximalgröße, usw.)
- Verändern der physikalischen Position einer Datendatei auf dem Datenträger

Eine vollständige Liste der Möglichkeiten, die das `ALTER DATABASE`-Kommando bietet ist im Microsoft Developer Network abgebildet.



- [ms174269]

Umbenennen einer Datendatei

In der DEMO_GRAFISCH-Datenbank wurde der logische Name der primären Datendatei `primary01.mdf` fälschlicherweise nur mit `PRIMARY`, statt mit `PRIMARY01` angegeben. Dies soll nun geändert werden.

Listing 16.11: Den logischen Namen einer Datendatei ändern

```
ALTER DATABASE demo_grafisch
MODIFY FILE (
    NAME      = 'PRIMARY',
    NEWNAME   = 'PRIMARY01')
```

Hinzufügen einer neuen Dateigruppe

Die DEMO_GRAFISCH-Datenbank benötigt eine neue Dateigruppe namens HRM.

Listing 16.12: Eine neue Dateigruppe hinzufügen

```
ALTER DATABASE demo_grafisch
ADD FILEGROUP HRM
```

Hinzufügen einer Datendatei

Die neu hinzugefügte Dateigruppe HRM benötigt eine Datendatei.

Listing 16.13: Eine Datendatei einer Dateigruppe hinzufügen

```
ALTER DATABASE demo_grafisch
ADD FILE (
    NAME      = 'HRM01',
    FILENAME  = 'D:\data\demo_grafisch\hrm01.ndf',
    SIZE      = 500 MB,
    MAXSIZE   = 4 GB,
    FILEGROWTH = 100 MB )
TO FILEGROUP HRM
```

- [bb522469]



Ändern der Standarddateigruppe

Wenn in einer SQL Server Datenbank ein neues Objekt (Tabelle, Index, Datendatei, ...) angelegt wird, muss es einer Dateigruppe zugeordnet werden. Dies kann auf zwei unterschiedliche Arten geschehen:

- Durch manuelle Angabe des Administrators, oder

- ohne Angabe des Administrators mit der Standarddateigruppe.



Alle Objekte, für die keine Dateigruppenzuordnung gemacht wird, werden automatisch in der Standarddateigruppe angelegt.

Standardmäßig wird die Dateigruppe PRIMARY als Standarddateigruppe angegeben. Dies kann aber nach Belieben geändert werden.

Listing 16.14: Ändern der Standarddateigruppe

```

IF NOT EXISTS (SELECT name
               FROM sys.filegroups
               WHERE is_default=1 AND name = N'SALES')
ALTER DATABASE demo_grafisch
MODIFY FILEGROUP SALES DEFAULT
GO

```

16.8.3 Datenbankoptionen verändern

Datenbankoptionen sind dazu da, um das Verhalten einer Datenbank zu steuern. Der Datenbankadministrator hat mit Hilfe dieser Optionen die Möglichkeit umfangreichen Einfluss auf das Verhalten der Datenbank zu nehmen. Im Folgenden werde einige Beispiele für verschiedene Datenbankoptionen gegeben. Ein vollständige Liste dieser Optionen kann auf der MSDN eingesehen werden.

- [bb522682]



Die Option ANSI_WARNINGS

Mit der Datenbankoption ANSI_WARNINGS wird entschieden, ob Fehlermeldungen ausgegeben werden, wenn z. B. eine Division durch 0 geschieht oder wenn NULL-Werte in einer Aggregatfunktion verarbeitet werden soll.

- **ON:** Es werden ANSI-Standardkonforme Fehlermeldungen ausgegeben.
- **OFF:** Es werden keine Fehlermeldungen erzeugt und im Falle einer Division durch 0 wird der Wert NULL als Ergebnis der Berechnung angenommen.

Listing 16.15: ANSI-Standardkonforme Warnmeldungen ausgeben

```

ALTER DATABASE demo_grafisch
SET ANSI_WARNINGS ON;

```

Die Option ARITHABORT

Mit dieser Option wird gesteuert was im Falle eines arithmetischen Überlaufs oder einer Division durch 0 geschieht.

- **ON:** Das SQL-Statement wird abgebrochen.
- **OFF:** Es wird lediglich eine Warnmeldung ausgegeben, wenn einer der genannten Fehler auftritt.
Das Statement wird aber zuendegeführt, so als hätte es keinen Fehler gegeben.

Listing 16.16: Arithmetische Überläufe und Divisionen durch 0 kontrollieren

```
ALTER DATABASE demo_grafisch  
SET ARITHABORT ON;
```

Die Option CONCAT_NULL_YIELDS_NULL

- **ON:** Eine Verkettungsoperation gibt NULL zurück, wenn einer der beiden Operanden NULL war.
Wird z. B. der Text „Hallo Welt“ mit dem Wert NULL verkettet, ist das Ergebnis NULL.
- **OFF:** Bei Verkettungsoperationen werden NULL-Werte wie leere Zeichenfolgen behandelt. Wird z. B. der Text „Hallo Welt“ mit dem Wert NULL verkettet, ist das Ergebnis „Hallo Welt“.

Listing 16.17: Behandlung von NULL-Werten bei Verkettungsoperationen steuern

```
ALTER DATABASE demo_grafisch  
SET CONCAT_NULL_YIELDS_NULL ON;
```

Das Kompatibilitätslevel der Datenbank ändern

Das Kompatibilitätslevel, gibt an welche Features in einer SQL Server-Datenbank aktiviert werden können bzw. wie sich die Datenbank in verschiedenen Situationen verhält. In einer SQL Server 2014-Instanz können Datenbanken mit dem Kompatibilitätsgrad 90 (SQL Server 2005), 100 (SQL Server 2008), 110 (SQL Server 2012) oder 120 (SQL Server 2014) betrieben werden. Dadurch wird die Migration einer älteren Datenbank auf eine neue SQL Server-Version ermöglicht.

Durch das ändern des Kompatibilitätslevels ist es auch möglich, dass sich eine in SQL Server 2014 erstellte Datenbank verhält, als wäre sie mit dem SQL Server 2005 erstellt worden.

Listing 16.18: Ändern des Kompatibilitätslevels einer Datenbank

```
ALTER DATABASE demo_grafisch2005  
SET COMPATIBILITY_LEVEL 120;
```

In Beispiel ?? wird die fiktive Datenbank DEMO_GRAFISCH2005, die noch mit dem SQL Server 2005 erstellt wurde, auf eine SQL Server 2014-Instanz migriert und der Kompatibilitätsgrad wird auf 120 angehoben. Dadurch werden für diese Datenbank alle SQL Server 2014-Features aktiviert.

16.8.4 Vergrößern und schrumpfen einer Datenbank

Automatisches Wachstum und manuelles Vergrößern

Das automatische Wachstum für eine Datendatei wird, wie bereits demonstriert, über die beiden Dateieigenschaften FILEGROWTH und MAXSIZE kontrolliert. Um eine Datendatei manuell zu vergrößern muss deren Eigenschaft SIZE angepasst werden.

Listing 16.19: Eine Datendatei vergrößern

```
ALTER DATABASE demo_grafisch
MODIFY FILE (
    NAME = 'SALES01',
    SIZE = 2 GB
)
```

Schrumpfen der gesamten Datenbank

Das Schrumpfen einer ganzen Datenbank kann auf einfaches Wege über das SSMS erledigt werden. Über das Kontextmenü der Datenbank kann unter TASKS, VERKLEINERN, DATENBANK der Assistent zum Verkleinern der Datenbank erreicht werden.

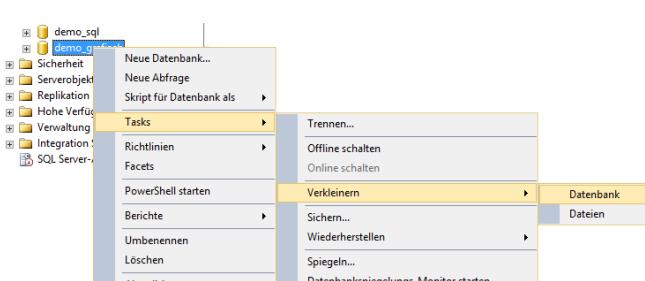


Abb. 16.31:
Den Assistenten
zum Schrumpfen
der Datenbank
starten

Nach einem Klick auf DATENBANK öffnet sich der Assistent.

Der Assistent zeigt an, wie viel Speicherplatz die Datenbank aktuell belegt und wie viel er davon freigeben könnte. Zusätzlich zum reinen Verkleinern kann die Datenbank auch noch reorganisiert (defragmentiert) werden.



Das Schrumpfen einer ganzen Datenbank kann ein sehr zeitaufwendiger und resourcenintensiver Vorgang sein. Es wird daher dringend davon abgeraten eine Datenbank zu Schrumpfen, es sei denn, dass diese Aktion unumgänglich ist.

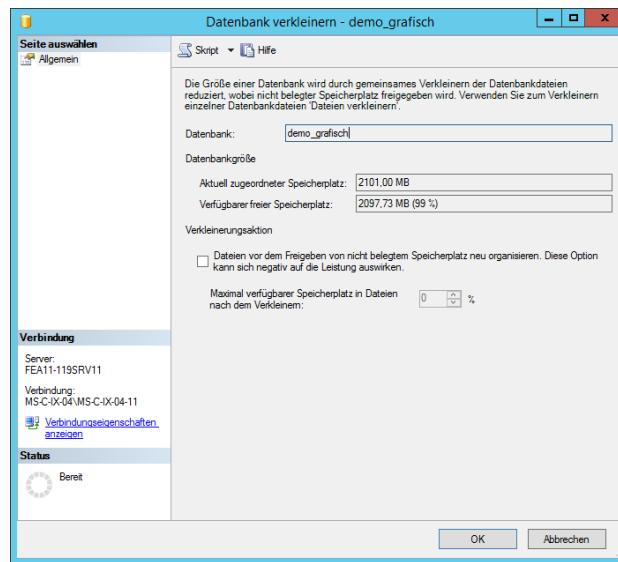


Abb. 16.32:
Der Assistenten zum Schrumpfen der Datenbank

Schrumpfen einzelner Datendateien

SQL Server bietet auch die Möglichkeit nur einzelne Daten- und Protokolldateien zu schrumpfen. Der hierfür zuständige Assistent wird, genau wie beim Schrumpfen der Datenbank auch, über das Kontextmenü der Datenbank erreicht (TASKS, VERKLEINERN, DATENDATEI).

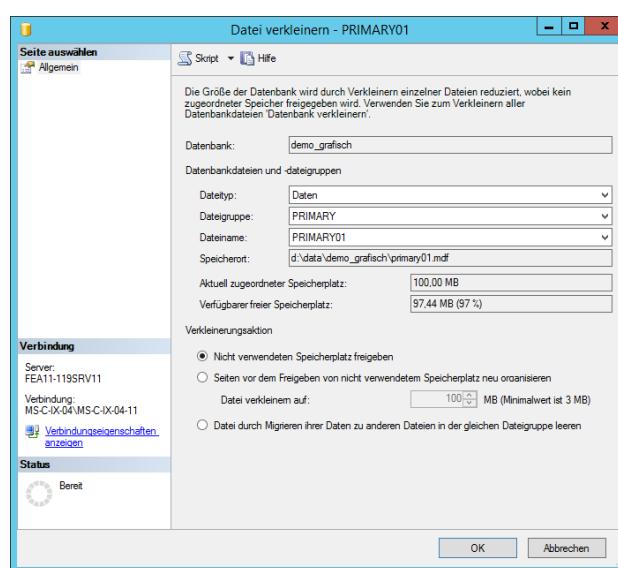


Abb. 16.33:
Der Assistenten zum Schrumpfen einzelner Datendateien

für den Vorgang des Schrumpfens stehen drei verschiedene Optionen zur Verfügung:

- **Nicht verwendeten Speicherplatz freigeben:** Es wird nur der Speicherplatz am Ende der Datendatei freigegeben. Es werden keine Speicherseiten umsortiert.

- **Seiten vor dem Freigeben von nicht verwendetem Speicherplatz neu organisieren:** Vor dem Verkleinern werden zuerst alle Speicherseiten reorganisiert, so dass der Speicherplatz am Ende der Datendatei möglichst frei wird. Anschließend wird verkleinert. Wird zusätzlich die Option DATEI VERKLEINER AUF aktiviert versucht SQL Server die Datendatei auf den angegebenen Wert zu verkleiner, sofern dies möglich ist.
- **Datei durch Migrieren ihrer Daten zu anderen Dateien in der gleichen Dateigruppe leeren:** Es werden alle Speicherseiten in andere Datendateien der gleichen Dateigruppe verschoben, so dass die betroffene Datendatei am Ende leer ist.



Da beim Verkleinern einer Datendatei die Speicherseiten am Ende der Datei nach vorne umsortiert werden, entsteht als Resultat eine hohe Indexfragmentierung. Dies kann die Performance von Abfragen massiv beeinflussen.

16.8.5 Datenbanken trennen und einhängen

Datenbanken trennen

Dem Administrator ist es möglich, eine Datenbank vollständig von einer SQL Server-Instanz zu trennen. Dies kann in verschiedenen Situationen nützlich sein, z. B. wenn eine Datenbank auf einen anderen Server umgezogen werden soll oder wenn eine Datenbank aus der Nutzung genommen werden muss, um sie außerhalb der Arbeitszeiten dann zu löschen. Außerdem ist es möglich, eine zugröße gewordene Log-Datei durch das Trennen und Anfügen der Datenbank auf ihre Mindestgröße zu schrumpfen.

Beim Vorgang des trennens werden alle Spuren der Datenbank aus der Instanz gelöscht. Einzig ein Sicherungsverlauf in der MSDB-Datenbank kann übrig bleiben. Das Trennen der Datenbank von ihrer Instanz geschieht mit der Stored Procedure SP_DETACH_DB, die für Ihre Arbeit nur den Datenbanknamen als Parameter benötigt.

Um eine Datenbank trennen zu können, müssen die folgenden Voraussetzungen gegeben sein:

- Die Datenbank darf aktuell nicht in Verwendung sein,
- sie darf keiner Replikation angehören,
- es darf keinen Database-Snapshot geben,
- sie darf nicht gespiegelt sein,

- es darf keine Systemdatenbank sein,
- die Datenbank darf nicht „fehlerverdächtig“ sein.

Listing 16.20: Trennen der Datenbank demo_grafisch

```
USE master
GO

EXEC sp_detach_db @dbname = 'demo_grafisch'
GO
```



- [ms188031]

Datenbanken anfügen

Zum Anfügen einer Datenbank existieren in SQL Server 2014 zwei Möglichkeiten:

- Die Stored Procedure SP_ATTACH_DB. Sie gilt als veraltet und sollte daher nicht mehr benutzt werden. Außerdem hat sie ein Limit von max. 16 Datendateien, aus denen die anzuhängende Datenbank bestehen darf.
- Das SQL-Kommando CREATE DATABASE ... FOR ATTACH existiert seit SQL Server 2008 und stellt nun die von Microsoft empfohlene Methode zum Anfügen von Datenbanken dar. für diese Kommando existieren keinerlei Einschränkungen bezüglich der Datenbankgröße oder der Anzahl der Datendateien einer Datenbank.

In Beispiel ?? wird die Datenbank DEMO_GRAFISCH wieder an ihre SQL Server-Instanz angefügt. Obwohl sie aus vier Datendateien und einer Log-Datei besteht, genügt es, Pfad + Dateiname der primären Datendatei anzugeben, da dort die Metadaten der anderen Dateien gespeichert sind.



Sollte beim Trennen der Datenbank deren Log-Datei beschädigt oder nicht vorhanden gewesen sein, wird beim wiederanfügen der Datebank eine neue Log-Datei erstellt. Voraussetzung ist, dass die Datenbank nicht schreibgeschützt ist.

Listing 16.21: Anfügen der Datenbank demo_grafisch

```
USE master
GO

CREATE DATABASE demo_grafisch
ON
(
    FILENAME = 'D:\data\demo_grafisch\primary01.mdf',
)
FOR ATTACH
GO
```



Um eine Datenbank anfügen zu können, müssen alle .mdf- und .ndf-Dateien verfügbar sein!

für schreibgeschützte Datenbanken gilt eine Besonderheit: Da die .mdf-Datendatei nicht geändert werden kann, kann auch keine Log-Datei neu erstellt werden. In so einem Fall muss zwingend die bestehende Log-Datei mit angegeben werden.

Listing 16.22: Anfügen der schreibgeschützten Datenbank demo_grafisch

```
USE master
GO

CREATE DATABASE demo_grafisch
ON
(
    FILENAME = 'D:\data\demo_grafisch\primary01.mdf',
),
(
    FILENAME = 'D:\data\demo_grafisch\demo_grafisch_log.ldf'
)
FOR ATTACH
GO
```



- [ms190794]

16.8.6 Systemdatenbanken verschieben

Verschieben Systemdatenbanken

Da Systemdatenbanken nicht einfach ausgehängt werden können, muss ein anderes Verfahren angewandt werden, wenn sie verschoben werden sollen.



Die einzige Systemdatenbank die nicht verschoben werden kann ist die RESOURCE-Datenbank.

Der folgende Ablauf zeigt das Verschieben einer Systemdatenbank am Beispiel der MSDB-Datenbank.

1. Führen Sie ein `ALTER DATABASE ... MODIFY FILE`-Kommando für alle Daten- und Log-Dateien der MSDB-Datenbank durch und geben Sie dabei den neuen Pfad der Dateien an.
2. Stop Sie die SQL Server-Instanz
3. Verschieben Sie die Dateien an ihren neuen Speicherort
4. Starten Sie die SQL Server-Instanz erneut
5. Benutzen Sie die Katalogsicht `SYS.MASTER_FILES`, um das Ergebnis des Verschiebevorganges zu verifizieren.

Listing 16.23: Abfragen der View `SYS.MASTER_FILES`

```
USE
master GO

SELECT name AS logical_name, physical_name AS new_location,
       state_desc as state
FROM   sys.master_files
WHERE  database_id = DB_ID('msdb')
```

Verschieben der Master-Datenbank

Die Technik zum Verschieben der MASTER-Datenbank ähnelt dem Verschiebevorgang für alle anderen Systemdatenbanken sehr. Der wesentliche Unterschied besteht darin, dass nach dem Herunterfahren des SQL Server-Dienstes der Startparameter `-d` geändert werden muss, so dass dieser den neuen Speicherort der Datenbank angibt.

1. Stoppen Sie die SQL Server-Instanz im Database Configuration Assistant.
2. Verschieben Sie die Dateien an ihren neuen Speicherort
3. Ändern Sie den Startparameter -d so, dass er den neuen Speicherort der MASTER-Datenbank angibt.
4. Starten Sie die SQL Server-Instanz neu.
5. Benutzen Sie Katalogsicht die SYS.MASTER_FILES, um das Ergebnis des Verschiebevorganges zu verifizieren.

Listing 16.24: Abfragen der View SYS.MASTER_FILES

```
USE
master GO

SELECT name AS logical_name , physical_name AS new_location ,
       state_desc as state
FROM   sys.master_files
WHERE  database_id = DB_ID('master')
```

16.8.7 Datenbanken löschen

Löschen von Datenbanken und Snapshots

SQL Server bietet das SQL-Kommando `DROP DATABASE` zum Löschen von Datenbanken und Datenbank-Snapshots an. Es kann sowohl Datenbanken von einem Microsoft SQL Server, als auch von einem Windows Azure SQL-Datenbank-Server löschen. Die Syntax dieses Kommandos ist sehr einfach, wie Beispiel ?? zeigt.

Listing 16.25: Löschen der Datenbank demo_sql

```
USE master
GO

DROP DATABASE demo_sql
GO
```

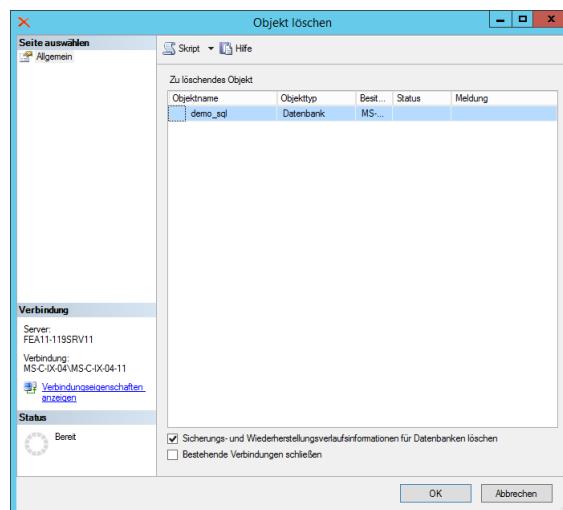
Lediglich der Datenbankname ist zum Löschen der Datenbank notwendig. Der Status einer Datenbank (offline, fehlerverdächtig, ...) ist für das Löschen nicht von Bedeutung. Soll eine einmal gelöschte Datenbank wieder an den Server angefügt werden, kann dies nur mittels eines Backups geschehen.



Das Löschen einer Datenbank beeinhaltet auch das Löschen der Daten- und der Log-Dateien. Befindet sich eine Datenbank im Status OFFLINE, werden deren Datendateien nicht automatisch gelöscht.

Als Alternative zum `DROP DATABASE`-Kommando kann auch das SSMS genutzt werden. Der Dialog zum Löschen der Datenbank ist über das Kontextmenü verfügbar.

Abb. 16.34:
Löschen einer
Datenbank im
SSMS



Einschränkungen beim Löschen von Datenbanken und Snapshots

- Es können keine Systemdatenbanken gelöscht werden.
- Datenbanken, zu denen Benutzerverbindungen bestehen, können nicht gelöscht werden.
- Vor dem Löschen einer Datenbank müssen zuerst alle Datenbank-Snapshots gelöscht werden.
- Das `DROP DATABASE`-Kommando muss im Autocommitt-Modus ausgeführt werden.

16.9 Der DBCC - Database Console Commands

Hinter der Abkürzung DBCC - Database Console Commands - verbergen sich eine ganze Reihe von Befehlen zu unterschiedlichsten Zwecken. Ursprünglich waren diese Kommandos für die Softwaretesting Teams von Microsoft geschaffen worden. Mit ihrer Hilfe sollte es möglich werden, Aufgaben wie Informationsgewinnung, Debugging und Testing auf einfache und schnelle Art und Weise zu erledigen. Nach und nach wuchs der Vorrat an DBCC-Kommandos und einige wurden sogar dokumentiert und



für die Nutzung durch SQL Server-Administratoren freigegeben. Viele DBCC-Befehl sind aber auch heute noch undokumentiert und somit nur der Benutzung durch das Supportpersonal von Microsoft vorbehalten.

Die Benutzung eines undokumentierten DBCC-Kommandos durch einen SQL Server-Administrator kann, im Falle eines Fehlers, den Verlust der Supportberechtigung zur Folge haben. Deshalb sollten niemals undokumentierte DBCC-Kommandos auf einer operativen Datenbank eingesetzt werden.

In SQL Server 2014 sind aktuell 30 verschiedene DBCC-Befehle dokumentiert. Diese können in vier verschiedene Kategorien unterteilt werden:

- Verwaltung
- Information
- Überprüfung
- Sonstiges

Alle DBCC-Kommandos nehmen Argumente entgegen, mit deren Hilfe das Verhalten des jeweiligen Kommandos gesteuert werden kann.

16.9.1 Überprüfungskommandos

DBCC kennt insgesamt sieben dokumentierte Anweisungen zur Überprüfung der Datenbank bzw. von Teilen der Datenbank. Hier eine kleine Auswahl.

- **CHECKDB:** Dient zur Überprüfung der physischen und der logischen Integrität aller Objekte innerhalb einer Datenbank.
- **CHECKFILEGROUP:** Verhält sich genauso, wie [DBCC CHECKDB](#), nur dass die Auswirkungen auf eine Dateigruppe begrenzt bleiben.
- **CHECKTABLE:** Wie [DBCC CHECKDB](#), aber nur für die angegebene Tabelle.
- **CHECKCONSTRAINTS:** Überprüft die Funktionsfähigkeit eines bestimmten oder aller Constraints auf einer Tabelle.

Konsistenzprüfung der Datenbank mit CHECKDB

Mit dem Kommando `CHECKDB` wird die Datenkonsistenz aller Objekte einer Datenbank geprüft. Um dieses Kommando ausführen zu können, muss der Benutzer entweder Mitglied in der festen Serverrolle `SYSADMIN` oder der festen Datenbankrolle `DB_OWNER` sein.

Der Befehl `DBCHECK` ist ein „Sammelkommando“, was bedeutet, dass bei seiner Ausführung im Hintergrund

- `DBCC CHECKCATALOG`,
- `DBCC CHECKALLOC` und
- `DBCC CHECKTABLE`

ausgeführt werden. Zusätzlich zu diesen drei Anweisungen werden noch einige weitere Prüfungen durchgeführt, die hier unerwähnt bleiben.

Die Syntax zur Ausführung von `DBCC CHECKDB` lautet (es handelt sich hier um ein auf das Wesentliche gekürztes Syntaxdiagramm:

Listing 16.26: Die Syntax zu CHECKDB

```
DBCC CHECKDB
[ ( database_name / database_id / 0
    [ , ( REPAIR_ALLOW_DATA_LOSS / REPAIR_FAST / REPAIR_REBUILD ) ]
) ]
[ WITH
{
    [ ESTIMATEONLY ]
    [ , ( PHYSICAL_ONLY / DATA_PURITY ) ]
}
]
```

- **database_name / database_id / 0**: Legt anhand des Namens oder einer ID fest, welche Datenbank geprüft werden soll. Wird dieses Argument nicht angegeben, oder wird eine 0 angegeben, so wird die aktuelle Datenbank überprüft.
- **ESTIMATEONLY**: Es wird berechnet, wie viel Speicherplatz in der TEMPDB benötigt wird, um das `CHECKDB`-Kommando auszuführen. Durch die Angabe dieses Parameters wird die Anweisung nicht ausgeführt.

- **PHYSICAL_ONLY:** Es werden nur physikalische Überprüfungen durchgeführt. Dadurch wird die Ausführungszeit des DBCC-Kommandos deutlich reduziert.
- **DATA_PURITY:** Kommando zur Durchführung einer logischen Konsistenzprüfung von Spaltenwerten, z. B. ob ein Datums-/Zeitwert innerhalb der gültigen Grenzen einer Spalte vom Typ DATETIME liegt. Diese Option wird nur benötigt, wenn eine Datenbank mit einer früheren Version von SQL Server erstellt wurde. Nach einmaliger und erfolgreicher Ausführung dieser Prüfung muss diese Option nie wieder angegeben werden.

Listing 16.27: Physikalische Konsistenzprüfung einer Datenbank mit DBCC

```
USE [master]
GO

DBCC CHECKDB ('demo_grafisch') WITH PHYSICAL_ONLY

Von CHECKDB wurden 0 Zuordnungsfehler und 0 Konsistenzfehler
in der demo_grafisch-Datenbank gefunden.
Die DBCC-Ausf\\"uhrung wurde abgeschlossen. Falls DBCC Fehlermeldungen
ausgegeben hat, wenden Sie sich an den Systemadministrator.
```

Listing 16.28: Konsistenzprüfung einer Datenbank mit DBCC

```
USE [master]
GO

DBCC CHECKDB ('demo_grafisch')

DBCC-Ergebnis fuer 'demo_grafisch'.
Service Broker-Meldung 9675, Status 1: Analysierte Nachrichtentypen: 14.
Service Broker-Meldung 9676, Status 1: Analysierte Dienstverträge: 6.
Service Broker-Meldung 9667, Status 1: Analysierte Dienste: 3.
Service Broker-Meldung 9668, Status 1: Analysierte Dienstwarteschlangen: 3.
Service Broker-Meldung 9669, Status 1: Analysierte Konversationsendpunkte: 0.
Service Broker-Meldung 9674, Status 1: Analysierte Konversationsgruppen: 0.
Service Broker-Meldung 9670, Status 1: Analysierte Remotedienstbindungen: 0.
Service Broker-Meldung 9605, Status 1: Analysierte Konversationspriorität\atnen:0.
DBCC-Ergebnis f\"ur 'sys.sysrscols'.
Es sind 1086 Zeilen auf 14 Seiten f\"ur das sys.sysrscols-Objekt vorhanden.
DBCC-Ergebnis f\"ur 'sys.sysrowsets'.
Es sind 146 Zeilen auf 3 Seiten f\"ur das sys.sysrowsets-Objekt vorhanden.
DBCC-Ergebnis f\"ur 'sys.sysclones'.
Es sind 0 Zeilen auf 0 Seiten f\"ur das sys.sysclones-Objekt vorhanden.
DBCC-Ergebnis f\"ur 'sys.sysallocunits'.
Es sind 169 Zeilen auf 2 Seiten f\"ur das sys.sysallocunits-Objekt vorhanden.

Ergebnis gek\"urzt!!!

Von CHECKDB wurden 0 Zuordnungsfehler und 0 Konsistenzfehler
in der demo_grafisch-Datenbank gefunden.
```

```
Die DBCC-Ausführung wurde abgeschlossen. Falls DBCC Fehlermeldungen
ausgegeben hat, wenden Sie sich an den Systemadministrator.
```

Zur Ausführung von `CHECKDB` wird ein interner Database Snapshot angelegt, eine Art „Abbildung der Datenbank“, vergleichbar mit den Snapshots im Bereich von Virtualisierungssoftware. Dies ermöglicht es dem SQL Server die Konsistenzprüfung der Datenbank so durchzuführen, dass die arbeitenden Benutzer nicht gestört werden.

Sollte `CHECKDB` Fehler in der Datenbank gefunden haben, wird eine Logdatei mit dem Namen `SQLDUMPnnnn.txt` im LOG-Verzeichnis angelegt.

Die drei im Syntaxdiagramm angegebenen Optionen `REPAIR_ALLOW_DATA_LOSS`, `REPAIR_FAST` und `REPAIR_REBUILD` bieten die Möglichkeit gefundene Fehler direkt zu reparieren. Es wird jedoch empfohlen, diese Optionen als das allerletzte Mittel zu betrachten, da durch ihre Anwendung weitere Dateninkonsistenzen entstehen könnten. Statt dessen sollten Backups genutzt werden, um fehlerhafte Datendateien wiederherzustellen.

Konsistenzprüfung einer einzelnen Dateigruppe

Die beiden Anweisungen `CHECKDB` und `CHECKFILEGROUP` sind sich in ihrer Anwendung sehr ähnlich. Der wesentliche Unterschied besteht darin, dass `CHECKDB` alle Dateigruppen, inklusive der Protokolldatei, einer Datenbank prüft, während mit `CHECKFILEGROUP` gezielt eine Dateigruppe geprüft werden kann. Die Syntax für `CHECKFILEGROUP` lautet wie folgt:

Listing 16.29: Die Syntax zu `CHECKFILEGROUP`

```
DBCC CHECKFILEGROUP
[ ( filegroup_name / filegroup_id / 0
) ]
[ WITH
{
    [ ESTIMATEONLY ]
    [ , PHYSICAL_ONLY ]
}
]
```

Listing 16.30: Konsistenzprüfung einer Dateigruppe

```
USE [demo_grafisch]
GO

DBCC CHECKFILEGROUP ('CRM')
```

```
DBCC-Ergebnis f\"ur 'demo_grafisch'.
DBCC-Ergebnis f\"ur 'sys.sysrscols'.
Es sind 1086 Zeilen auf 14 Seiten f\"ur das sys.sysrscols-Objekt vorhanden.
DBCC-Ergebnis f\"ur 'sys.sysrowsets'.
Es sind 146 Zeilen auf 3 Seiten f\"ur das sys.sysrowsets-Objekt vorhanden.

Ergebnis gek\"urzt!!!

Von CHECKFILEGROUP wurden 0 Zuordnungsfehler und 0 Konsistenzfehler
in der demo_grafisch-Datenbank gefunden.
Die DBCC-Ausf\"uhrung wurde abgeschlossen. Falls DBCC Fehlermeldungen
ausgegeben hat, wenden Sie sich an den Systemadministrator.
```

Einzelne Tabellen \u00fcberpr\u00fcfen

Mit **DBCC CHECKTABLE** kann eine einzelne Tabelle auf ihre Konsistenz gepr\u00fcft werden. Die Syntax dieser Anweisung gleicht der Syntax der **CHECKDB**-Abweisung.

Listing 16.31: Konsistenzprüfung einer einzelnen Tabelle

```
USE [demo_grafisch]
GO

DBCC CHECKTABLE ('demo_table')

DBCC-Ergebnis f\"ur 'demo_grafisch.demo_table'.
Es sind 99 Zeilen auf 2 Seiten f\"ur das demo_table-Objekt vorhanden.
Die DBCC-Ausf\"uhrung wurde abgeschlossen. Falls DBCC Fehlermeldungen
ausgegeben hat, wenden Sie sich an den Systemadministrator.
```



- [ms176061]
- [ms187332]
- [ms174338]
- [ms189496]

16.9.2 Verwaltungsanweisungen

Mit den DBCC-Befehlen aus der Gruppe der Verwaltungsanweisungen lassen sich administrative Tätigkeiten an Datenbank, Datendatei, Tabellen und Indizes durchführen. Die zwei wichtigsten Vertreter dieser Gruppe sind:

- **SHRINKDATABASE**: Dient zur Reduzierung der Größe von Daten- und Protokolldateien einer Datenbank.
- **SHRINKFILE**: Reduziert die Größe einer Daten- oder Protokolldatei.

Schrumpfen der Datenbank

Mit dem Kommando **SHRINKDATABASE** kann der Administrator sowohl die Daten- als auch die Protokolldateien einer Datenbank schrumpfen.

Listing 16.32: Die Syntax zu SHRINKDATABASE

```
DBCC SHRINKDATABASE
( database_name / database_id / 0
[ , target_percent ]
[ , ( NOTRUNCATE / TRUNCATEONLY ) ]
)
```

- **database_name / database_id / 0:** Legt anhand des Namens oder einer ID fest, welche Datenbank geschrumpft werden soll. Wird dieses Argument nicht angegeben, oder wird eine 0 angegeben, so wird die aktuelle Datenbank bearbeitet.
- **NOTRUNCATE:** „Defragmentiert“ den Inhalt Datendatei, verkleinert die Datei aber nicht. Es wird kein Speicherplatz freigegeben, es werden lediglich Datenseite verschoben. Ein mit der Option NOTRUNCATE ausgeführtes `DBCC SHRINKDATABASE` wirkt sich nur auf die Datendateien, aber nicht auf die Protokolldatei aus.
- **TRUNCATEONLY:** Der gesamte am Ende einer Datei befindliche Speicherplatz wird freigegeben/ abgeschnitten. Dateien werden somit verkleinert, es findet jedoch keine Defragmentierung innerhalb einer Datei statt. Durch die Angabe des Schlüsselwortes TRUNCATEONLY wirkt sich `DBCC SHRINKDATABASE` auch auf die Protokolldatei aus.
- **target_percent:** Menge an freiem Speicher, der nach der Verkleinerung in der Datendatei enthalten sein soll.



Die Größe einer Datenbank kann niemals unter ihre Mindestgröße geschrumpft werden. Wurde eine Datenbank z. B. mit einer Größe von 10 MB angelegt, kann sie niemals kleiner als 10 MB werden.

Das Schrumpfen einer Datenbank ist ein Online-Vorgang, d. h. während die Datenbank verkleinert wird können andere Nutzer ihre Arbeit fortsetzen.

Weiterhin ist zu beachten, das ein Ausführen von `SHRINKDATABASE` ohne die Angabe von `TRUNCATEONLY` und `NOTRUNCATE` wie folgt abgearbeitet wird:

1. Ausführen von `SHRINKDATABASE` mit `NOTRUNCATE`
2. Ausführen von `SHRINKDATABASE` mit `TRUNCATEONLY`



Werden die beiden Optionen `TRUNCATEONLY` und `NOTRUNCATE` weggelassen, wird das `SHRINKDATABASE`-Kommando zweimal ausgeführt. Zuerst mit `NOTRUNCATE` und anschließend mit `TRUNCATEONLY`

Listing 16.33: Verkleinern einer Datenbank

```
USE [master] GO
DBCC SHRINKDATABASE(demo_grafisch, 25)
```

Das `SHRINKDATABASE`-Kommando aus Beispiel ?? verkleinert die Datendateien der Datenbank DEMO_GRAFISCH um 25 %. Wenn z. B. eine Datendatei eine Größe von 20 MB und eine Mindestgröße von 15 MB hat und mit 8 MB Daten gefüllt ist, wird SQL Server die Datei auf 16 MB verkleinern. Somit verbleiben 8 MB freier Speichern in der Datendatei. Soll diese Datei jedoch um 30 % geschrumpft werden, kann SQL Server dies nicht, da sonst die Mindestgröße unterschritten werden würde.

Bei Protokolldateien verhält sich `SHRINKDATABASE` etwas anders. Dort gibt `target_percent` die Endgröße des Transaktionsprotokolls an und nicht die Menge an freiem Speicher nach dem Verkleinern. Besitzt die Datenbank mehr als nur eine Protokolldatei, wird für jede einzelne Protokolldatei deren Zielgröße berechnet, damit das Protokoll insgesamt auf die von `target_percent` definierte Größe kommt.



Eine Protokolldatei kann immer nur bis zu den Grenzen eines Virtual Log Files verkleinert werden. Insgesamt sollte es vermieden werden eine Protokolldatei zu schrumpfen, da sich dadurch die Anzahl und die Größe der VLFs in ihr verändert.

Am effektivsten ist ein Verkleinerungsvorgang, wenn erst kürzlich viel Speicherplatz in einer Datenbank (z. B. durch `DELETE` oder `TRUNCATE`) freigegeben wurde. Da das Schrumpfen einer Datenbank aber auch entscheidende negative Aspekte besitzt sollte es vermieden werden eine Datenbank wiederholt zu schrumpfen.

Schrumpfen einer Datendatei

Wenn gezielt einzelne Daten- oder Protokolldateien geschrumpft werden sollen kann dies mit der `SHRINKFILE`-Anweisung des DBCC geschehen. Sie unterscheidet sich in einigen Punkten von der `SHRINKDATABASE`-Anweisung:

- Die Zielgröße wird bei `SHRINKFILE` in MB und nicht in Prozent angegeben.
- Die Zielgröße bezeichnet bei `SHRINKFILE` die tatsächliche Größe der Datei nach dem Verkleinerungsvorgang.
- Mit `SHRINKFILE` kann die Mindestgröße einer Datendatei verringert werden.

Die Syntax für `SHRINKFILE` lautet:

Listing 16.34: Die Syntax zu `SHRINKFILE`

```
DBCC SHRINKFILE
  ( { file_name / file_id }
```

```

    { , [ EMPTYFILE ] /
      [
        [ , target_size ] [ , { NOTRUNCATE / TRUNCATEONLY } ]
      ]
    }
)

```

- **file_name / file_id:** Legt anhand des Namens oder einer ID fest, welche Datei geschrumpft werden soll.
- **EMPTYFILE:** Verlagert, wenn möglich, alle Daten aus einer Datendatei in die anderen, um eine Datendatei komplett freizugeben.
- **NOTRUNCATE:** „Defragmentiert“ den Inhalt einer Datendatei, verkleinert die Datei aber nicht. Es wird kein Speicherplatz freigegeben, es werden lediglich Datenseite verschoben. Ein mit der Option NOTRUNCATE ausgeführtes DBCC SHRINKDATABASE wirkt sich nur auf die Datendateien, aber nicht auf die Protokolldatei aus.
- **TRUNCATEONLY:** Der gesamte, am Ende einer Datei befindliche Speicherplatz wird freigegeben/-abgeschnitten. Dateien werden somit verkleinert, es findet jedoch keine Defragmentierung innerhalb einer Datei statt. Durch die Angabe des Schlüsselwortes TRUNCATEONLY wirkt sich DBCC SHRINKDATABASE auch auf die Protokolldatei aus.
- **target_size:** Größe der Datei nach der Verkleinerung.

Das Schrumpfen einer Protokolldatei verläuft analog zum Kommando SHRINKDATABASE.

- [ms190488]
- [ms189493]



16.9.3 Informationsanweisungen

In der Gruppe der Informationsanweisungen gibt es eine Reihe von DBCC-Kommandos die Informationen zu offenen Transaktionen, Auslastung des Transaktionsprotokolls, SQL Statistiken uvm. anzeigen.

Die Auslastung des Transaktionsprotokolls anzeigen

Mit dem Kommando SQLPERF kann zu jeder Datenbank einer SQL Server-Instanz die Auslastung ihrer Protokolldatei angezeigt werden.

Listing 16.35: Die Syntax zu SQLPERF

```
DBCC SQLPERF ( LOGSPACE )
```

Listing 16.36: Anzeigen der Protokolldateiauslastung

```
USE [master]
GO

DBCC (LOGSPACE)
```



Database Name	Log Size (MB)	Log Space Used (%)	Status
master	2,242188	34,77787	0
tempdb	0,7421875	48,09211	0
model	0,9921875	34,49803	0
msdb	19,61719	11,27788	0
ReportServer	6,867188	22,7318	0
ReportServerTempDB	1,039063	60,05639	0
demo_grafisch	5,554688	48,31224	0

- [ms189768]



16.9.4 Sonstige Anweisungen

Mit dem Kommando `DBCC HELP` kann zu jedem anderen DBCC-Befehl eine Syntax-Hilfe abgerufen werden.

Listing 16.37: DBCC Help - Syntaxhilfe aufrufen

```
DBCC HELP |( 'CHECKDB' )|

dbcc CHECKDB
(
    { 'database_name' | database_id | 0 }
    [ , NOINDEX
    | { REPAIR_ALLOW_DATA_LOSS
    | REPAIR_FAST
    | REPAIR_REBUILD
    } ]
)
[ WITH
{
    [ ALL_ERRORMSGS ]
    [ , [ NO_INFOMSGS ] ]
    [ , [ TABLOCK ] ]
    [ , [ ESTIMATEONLY ] ]
    [ , [ PHYSICAL_ONLY ] ]
    [ , [ DATA_PURITY ] ]
    [ , [ EXTENDED_LOGICAL_CHECKS ] ]
}
]

Die DBCC-Ausf\"uhung wurde abgeschlossen.
Falls DBCC Fehlermeldungen ausgegeben hat, wenden Sie sich
an den Systemadministrator.
```

Mit dem Befehl aus Beispiel ?? wird eine Syntax-Hilfe zum Kommando DBCC CHECKDB angezeigt.



- [ms176040]

16.10 Übungen - Verwalten einer SQL Server-Instanz

1. Richten Sie auf Ihrem Übungsserver alle notwendigen Werkzeuge ein, um mit der Domäne MS-C-IX-04.FUS arbeiten zu können.
2. Nehmen Sie Ihren Übungsserver in die Domäne MS-C-IX-04.FUS auf. Achten Sie darauf, dass das Computerkonto Ihres Datenbankservers in der Organisationseinheit IT-SysAdmin\EXER\Serverxx\Computers abgelegt wird, bzw. verschieben Sie es dorthin.
3. Deaktivieren Sie den Zugriff mittels Shared Memory auf Ihre SQL Server-Instanz!
4. Der Zugriff auf Ihre SQL Server-Instanz soll nur noch mittels der öffentlichen IP-Adresse 192.168.111.xx erfolgen können! Alle anderen IPs müssen deaktiviert werden!
5. Erstellen Sie die in der folgenden Auflistung aufgeführten gMSAs in der Organisationseinheit IT-SysAdmin\EXER\Serverxx\MSAs für die Dienste Ihrer SQL Server-Instanz ein. Ersetzen Sie dabei „xx“ durch Ihre Platznummer!
 - MSSQLSRVRXX für den SQL Server-Dienst
 - MSSQLAGENTXX für den SQL Server Agent
 - MSDTSSRVRXX für den SQL Server Integration Services 12.0 Dienst
 - MSRPTSRVRXX für den SQL Server Reporting Services Dienst
 - MSSQLBRWSRXX für den SQL Server-Browser
6. Stoppen Sie die beiden Dienste SQL SERVER REPORTING SERVICES und SQL SERVER INTEGRATION SERVICES 12.0. Konfigurieren Sie beide so, dass Sie manuell gestartet werden müssen.
7. Konfigurieren Sie Ihre Instanz MSSQLSERVER so, dass Sie immer mindestens 200M Arbeitsspeichern zur Verfügung hat und nie mehr als 2G benutzt!
8. Schreiben Sie eine SQL-Abfrage, die anzeigen, ob die Einstellungen aus der vorangegangenen Aufgabe bereits wirksam geworden sind oder nicht!
9. Deaktivieren Sie die Überwachung von fehlerhaften Anmeldungen. Es soll keine Anmeldeüberwachung geben!

10. Fügen Sie Ihre Instanz MSSQLSERVER die Datenbank BANK 2014 hinzu. Der Aufbau der Datenbank kann der folgenden Tabelle entnommen werden. Die Namen der Dateigruppen sind in den logischen Namen der Datendateien enthalten (Primary, HR, CRM, KAM und MAIL).

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_primary_01	10 M	+ 10 M	500 M	D:\u01\bank_2014\data
bank_2014_hr_01	100 M	+ 20 M	500 M	D:\u01\bank_2014\data
bank_2014_hr_02	100 M	+ 20 M	500 M	D:\u01\bank_2014\data
bank_2014_crm_01	50 M	+ 10 M	500 M	E:\u02\bank_2014\data
bank_2014_kam_01	100 M	+ 100 M	500 M	E:\u02\bank_2014\data
bank_2014_mail_01	500 M	+ 100 M	1 G	D:\u01\bank_2014\data
bank_2014_mail_02	500 M	+ 100 M	1 G	D:\u01\bank_2014\data
bank_2014_mail_03	500 M	+ 100 M	1 G	D:\u01\bank_2014\data
bank_2014_log	768 M	+ 768 M	3 G	F:\u03\bank_2014\log

11. Fügen Sie der BANK 2014 die Dateigruppe STAGING hinzu. Benutzen Sie dazu die folgenden Angaben:

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_staging_01	800 M	+ 40 M	2048 M	E:\u02\bank_2014\data

12. Fügen Sie der Dateigruppe CRM eine weitere Datendatei hinzu. Verwenden Sie dazu die folgenden Angaben:

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_crm_02	50 M	+ 10 M	500 M	E:\u02\bank_2014\data

13. Nehmen Sie die im Folgenden beschriebenen Veränderungen an den Datendateien Ihrer Datenbank BANK 2014 vor.

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_primary_01	10 M	+ 10 M	768 M	D:\u01\bank_2014\data
bank_2014_hr_01	100 M	+ 50 M	1 G	D:\u01\bank_2014\data
bank_2014_hr_02	100 M	+ 50 M	1 G	D:\u01\bank_2014\data
bank_2014_kam_01	500 M	+ 100 M	2 G	E:\u02\bank_2014\data

14. Schreiben Sie eine Abfrage, um zu überprüfen, ob die Datendatei ordnungsgemäß angefügt wurde.
15. Recherchieren Sie, wie eine einzelne Datendatei verschoben werden kann und verschieben Sie im Anschluss die Datendatei mit dem Namen BANK_2014_KAM_01 in den Ordner D:\u01\bank_2014\data. Welche Dateiberechtigungen müssen für die betreffende Datei gesetzt werden?
16. Machen Sie die Dateigruppe CRM zur Standarddateigruppe der Datenbank BANK 2014.

17. Konfigurieren Sie für die Datenbankoption QUOTED_IDENTIFIER den Wert TRUE.
18. Ermitteln Sie welchen Wert die Datenbankoption AUTO_CLOSE hat und recherchieren Sie, welche Bedeutung diese Option hat bzw. was sie bewirkt.

19. Verschieben Sie Ihre Datenbank TEMPDB gemäß den folgenden Angaben:

Name	Größe	Wachstum	G Max.	Pfad
tempdev	1 G	+ 200 M	2 G	G:\u04\tempdb\data
templog	200 M	+ 50 M	500 M	F:\u03\tempdb\log

20. Verschieben Sie das Fehlerprotokoll Ihrer Instanz an den folgenden neuen Speicherort: D:\u01\errorlog\sql_s

21. Beim Anlegen der Datenbank ist Ihnen ein Fehler unterlaufen. Statt die Datenbank BANK_2014 zu nennen, haben Sie sie BANK 2014 genannt. Recherchieren Sie, wie eine Datenbank umbenannt werden kann und ersetzen Sie das Leerzeichen im Datenbanknamen durch einen Unterstrich!

16.11 Lösungen - Verwalten einer SQL Server-Instanz

1. Richten Sie auf Ihrem Übungsserver alle notwendigen Werkzeuge ein, um mit der Domäne MS-C-IX-04.FUS arbeiten zu können.

Fügen Sie Ihrem Computer die „Richtlinienverwaltung“ und die „Remoteverwaltungstools“ hinzu!

2. Nehmen Sie Ihren Übungsserver in die Domäne MS-C-IX-04.FUS auf. Achten Sie darauf, dass das Computerkonto Ihres Datenbankservers in der Organisationseinheit IT-SysAdmin\EXER\Serverxx\Computers abgelegt wird, bzw. verschieben Sie es dorthin.

Listing 16.38: Aufnehmen eines Computers in eine Windows Domäne

```
$ou = "OU=Computers,OU=Serverxx,OU=EXER,OU=IT-SysAdmin,DC=MS-C-IX-04-DC=FUS"
Add-Computer -ComputerName FEA11-119SRVxx -
-LocalCredential FEA11-119SRVxx\Administrator -
-DomainName MS-C-IX-04.FUS -
-Credential MS-C-IX_04\Administrator -
-OUPath $ou
-Restart -Force
```

3. Deaktivieren Sie den Zugriff mittels Shared Memory auf Ihre SQL Server-Instanz!

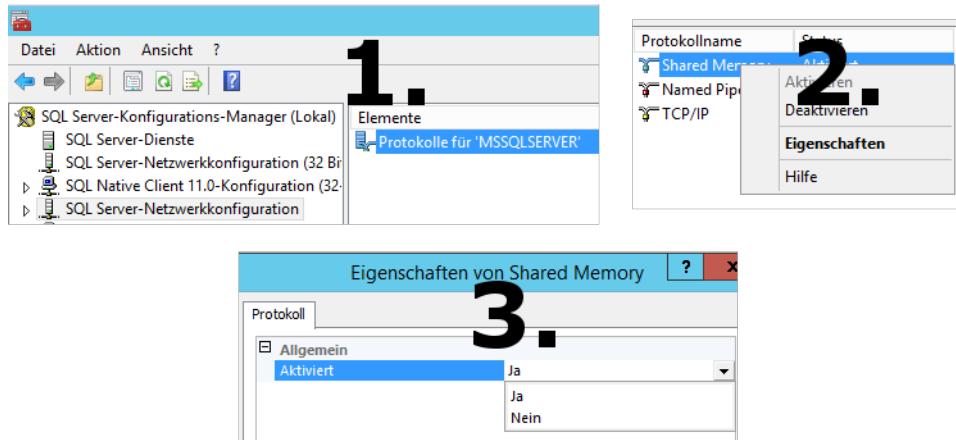
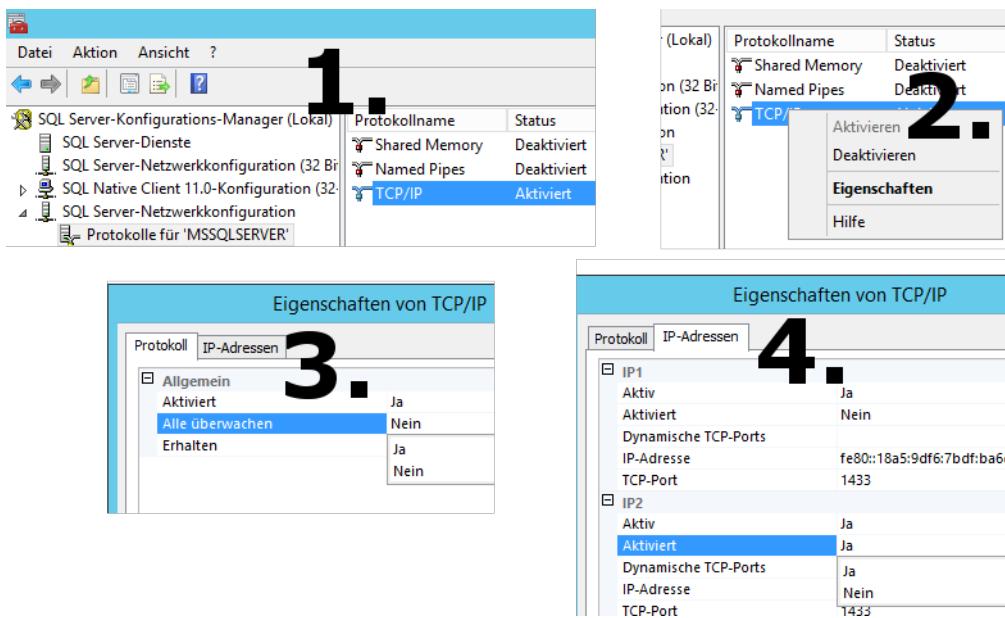


Abb. 16.35:
Den Shared
Memory-Zugriff
deaktivieren

4. Der Zugriff auf Ihre SQL Server-Instanz soll nur noch mittels der öffentlichen IP-Adresse 192.168.111.xx erfolgen können! Alle anderen IPs müssen deaktiviert werden!

Abb. 16.36:
IP-Adressen
deaktivieren



5. Erstellen Sie die in der folgenden Auflistung aufgeführten gMSAs in der Organisationseinheit IT-SysAdmin\EXER\Serverxx\MSAs für die Dienste Ihrer SQL Server-Instanz ein. Ersetzen Sie dabei „xx“ durch Ihre Platznummer!

- MSSQLSRVRXX für den SQL Server-Dienst
- MSSQLAGENTXX für den SQL Server Agent
- MSDTSSRVXXXX für den SQL Server Integration Services 12.0 Dienst
- MSRPTSRVXXXX für den SQL Server Reporting Services Dienst
- MSSQLBRWSRXX für den SQL Server-Browser

Listing 16.39: Die notwendigen gMSAs erstellen

```
$gou = "OU=GROUPS,OU=Serverxx,OU=EXER,OU=IT-SysAdmin,DC=MS-C-IX-04-DC=FUS"
New-ADGroup -Name SQLServerSVCxx -GroupScope Global `
-GroupCategory Security `
-Path $gou

$cou = "OU=Computers,OU=Serverxx,OU=EXER,OU=IT-SysAdmin,DC=MS-C-IX-04-DC=FUS"
$identity = "CN=SQLServerSVCxx," + $cou
$member = "CN=FEA11-119SRVxx," + $cou

Add-ADGroupMember -Identity $identity -Members $member
```

```

$ou = "OU=gMSAs,OU=Serverxx,OU=EXER,OU=IT-SysAdmin,DC=MS-C-IX-04-DC=FUS"

New-ADServiceAccount -Name MSSQLSRVRxx ` 
-DNSHostname FEA11-119SRVAD.MS-C-IX-04.FUS ` 
-Path $ou ` 
-PrincipalsAllowedToRetrieveManagedPassword $identity ` 
-Enabled $true

New-ADServiceAccount -Name MSSQLAGENTxx ` 
-DNSHostname FEA11-119SRVAD.MS-C-IX-04.FUS ` 
-Path $ou ` 
-PrincipalsAllowedToRetrieveManagedPassword $identity ` 
-Enabled $true

New-ADServiceAccount -Name MSDTSSRVRx ` 
-DNSHostname FEA11-119SRVAD.MS-C-IX-04.FUS ` 
-Path $ou ` 
-PrincipalsAllowedToRetrieveManagedPassword $identity ` 
-Enabled $true

New-ADServiceAccount -Name MSRPTSRVRxx ` 
-DNSHostname FEA11-119SRVAD.MS-C-IX-04.FUS ` 
-Path $ou ` 
-PrincipalsAllowedToRetrieveManagedPassword $identity ` 
-Enabled $true

New-ADServiceAccount -Name MSSQLBRWSRxx ` 
-DNSHostname FEA11-119SRVAD.MS-C-IX-04.FUS ` 
-Path $ou ` 
-PrincipalsAllowedToRetrieveManagedPassword $identity ` 
-Enabled $true

#Execute these steps as local admin on your computer

$identity = "CN=MSSQLSRVRxx," + $ou
Install-ADServiceAccount -Identity $identity

$identity = "CN=MSSQLAGENTxx," + $ou
Install-ADServiceAccount -Identity $identity

$identity = "CN=MSDTSSRVRx," + $ou
Install-ADServiceAccount -Identity $identity

$identity = "CN=MSRPTSRVRxx," + $ou
Install-ADServiceAccount -Identity $identity

$identity = "CN=MSSQLBRWSRxx," + $ou
Install-ADServiceAccount -Identity $identity

```

6. Stoppen Sie die beiden Dienste SQL SERVER REPORTING SERVICES und SQL SERVER INTEGRATION SERVICES 12.0. Konfigurieren Sie beide so, dass Sie manuell gestartet werden müssen.

Listing 16.40: Stoppen und umkonfigurieren von Diensten

```
Set-Service -DisplayName 'SQL Server Integration Services 12.0' -
-StartupType Manual

Set-Service -DisplayName 'SQL Server Reporting Services (MSSQLSERVER)' -
-StartupType Manual

Stop-Service -DisplayName 'SQL Server Integration Services 12.0'
Stop-Service -DisplayName 'SQL Server Reporting Services (MSSQLSERVER)'
```

7. Konfigurieren Sie Ihre Instanz MSSQLSERVER so, dass Sie immer mindestens 200M Arbeitsspeichern zur Verfügung hat und nie mehr als 2G benutzt!

Listing 16.41: Ändern der Servereigenschaften

```
EXEC sp_configure 'min server memory (MB)', '200'
EXEC sp_configure 'max server memory (MB)', '2048'
RECONFIGURE
```

8. Schreiben Sie eine SQL-Abfrage, die anzeigen, ob die Einstellungen aus der vorangegangenen Aufgabe bereits wirksam geworden sind oder nicht!

Listing 16.42: Ändern der Servereigenschaften

```
SELECT name, value, value_in_use
FROM sys.configurations
WHERE configuration_id IN (1543, 1544)
GO
```



name	value	value_in_use
min server memory (MB)	200	200
max server memory (MB)	2048	2048

2 Zeilen ausgewählt

9. Deaktivieren Sie die Überwachung von fehlerhaften Anmeldungen. Es soll keine Anmeldeüberwachung geben!

Listing 16.43: Ändern der Servereigenschaften

```
USE [master]
GO

EXEC xp_instance_regwrite N'HKEY_LOCAL_MACHINE',
N'Software\Microsoft\MSSQLServer\MSSQLServer',
```

```
N'AuditLevel', REG_DWORD, 0
GO
```

10. Fügen Sie Ihre Instanz MSSQLSERVER die Datenbank BANK 2014 hinzu. Der Aufbau der Datenbank kann der folgenden Tabelle entnommen werden. Die Namen der Dateigruppen sind in den logischen Namen der Datendateien enthalten (Primary, HR, CRM, KAM und MAIL).

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_primary_01	10 M	+ 10 M	500 M	D:\u01\bank_2014\data
bank_2014_hr_01	100 M	+ 20 M	500 M	D:\u01\bank_2014\data
bank_2014_hr_02	100 M	+ 20 M	500 M	D:\u01\bank_2014\data
bank_2014_crm_01	50 M	+ 10 M	500 M	E:\u02\bank_2014\data
bank_2014_kam_01	100 M	+ 100 M	500 M	E:\u02\bank_2014\data
bank_2014_mail_01	500 M	+ 100 M	1 G	D:\u01\bank_2014\data
bank_2014_mail_02	500 M	+ 100 M	1 G	D:\u01\bank_2014\data
bank_2014_mail_03	500 M	+ 100 M	1 G	D:\u01\bank_2014\data
bank_2014_log	768 M	+ 768 M	3 G	F:\u03\bank_2014\log

Listing 16.44: Ändern der Servereigenschaften

```
USE [master]
GO

CREATE DATABASE [Bank 2014]
CONTAINMENT = NONE
ON PRIMARY
( NAME = N'bank_2014_primary_01',
  FILENAME = N'D:\u01\bank_2014\data\bank_2014_primary_01.mdf',
  SIZE = 10 MB, MAXSIZE = 500 MB, FILEGROWTH = 10 MB ),
FILEGROUP [CRM]
( NAME = N'bank_2014_crm_01',
  FILENAME = N'E:\u02\bank_2014\data\bank_2014_crm_01.ndf',
  SIZE = 50 MB, MAXSIZE = 100 MB, FILEGROWTH = 10 MB ),
FILEGROUP [HR]
( NAME = N'bank_2014_hr_01',
  FILENAME = N'D:\u01\bank_2014\data\bank_2014_hr_01.ndf',
  SIZE = 100 MB, MAXSIZE = 500 MB, FILEGROWTH = 20 MB ),
( NAME = N'bank_2014_hr_02',
  FILENAME = N'D:\u01\bank_2014\data\bank_2014_hr_02.ndf',
  SIZE = 100 MB, MAXSIZE = 500 MB, FILEGROWTH = 20 MB ),
FILEGROUP [KAM]
( NAME = N'bank_2014_kam_01',
  FILENAME = N'E:\u02\bank_2014\data\bank_2014_kam_01.ndf',
  SIZE = 100 MB, MAXSIZE = 500 MB, FILEGROWTH = 100 MB ),
```

```

FILEGROUP [MAIL]
( NAME = N'bank_2014_mail_01',
  FILENAME = N'D:\u01\bank_2014\data\bank_2014_mail_01.ndf',
  SIZE = 500 MB, MAXSIZE = 1 GB, FILEGROWTH = 100 MB ),
( NAME = N'bank_2014_mail_02',
  FILENAME = N'D:\u01\bank_2014\data\bank_2014_mail_02.ndf',
  SIZE = 500 MB, MAXSIZE = 1 GB, FILEGROWTH = 100 MB ),
( NAME = N'bank_2014_mail_03',
  FILENAME = N'D:\u01\bank_2014\data\bank_2014_mail_03.ndf',
  SIZE = 500 MB, MAXSIZE = 1 GB, FILEGROWTH = 100 MB )
LOG ON
( NAME = N'Bank_2014_log',
  FILENAME = N'F:\u03\bank_2014\log\bank_2014_log.ldf',
  SIZE = 768 MB, MAXSIZE = 3 GB, FILEGROWTH = 768 MB )
GO

```

11. Fügen Sie der BANK 2014 die Dateigruppe STAGING hinzu. Benutzen Sie dazu die folgenden Angaben:

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_staging_01	800 M	+ 40 M	2048 M	E:\u02\bank_2014\data

Listing 16.45: Hinzufügen einer Dateigruppe mit Datendatei

```

USE [master]
GO

ALTER DATABASE [Bank_2014]
ADD FILEGROUP [STAGING]
GO

ALTER DATABASE [Bank_2014]
ADD FILE (NAME = N'bank_2014_staging',
          FILENAME = N'E:\u02\bank_2014\data\bank_2014_staging_01.ndf',
          SIZE = 800 MB, MAXSIZE = 2 GB, FILEGROWTH = 40 MB)
TO FILEGROUP [STAGING]
GO

```

12. Fügen Sie der Dateigruppe CRM eine weitere Datendatei hinzu. Verwenden Sie dazu die folgenden Angaben:

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_crm_02	50 M	+ 10 M	500 M	E:\u02\bank_2014\data

Listing 16.46: Hinzufügen einer Datendatei zu einer bestehenden Dateigruppe

```
USE [master]
GO

ALTER DATABASE [Bank 2014]
ADD FILE (NAME = N'bank_2014_crm_02',
           FILENAME = N'E:\u02\bank_2014\data\bank_2014_crm_02.ndf',
           SIZE = 50 MB, MAXSIZE = 500 MB, FILEGROWTH = 10 MB )
TO FILEGROUP [CRM]
GO
```

13. Nehmen Sie die im Folgenden beschriebenen Veränderungen an den Datendateien Ihrer Datenbank BANK 2014 vor.

Name	Größe	Wachstum	G Max.	Pfad
bank_2014_primary_01	10 M	+ 10 M	768 M	D:\u01\bank_2014\data
bank_2014_hr_01	100 M	+ 50 M	1 G	D:\u01\bank_2014\data
bank_2014_hr_02	100 M	+ 50 M	1 G	D:\u01\bank_2014\data
bank_2014_kam_01	500 M	+ 100 M	2 G	E:\u02\bank_2014\data

Listing 16.47: Ändern der Datendateieigenschaften

```
USE [master]
GO

ALTER DATABASE [Bank 2014]
MODIFY FILE ( NAME = N'bank_2014_hr_01', MAXSIZE = 1 GB )
GO

ALTER DATABASE [Bank 2014]
MODIFY FILE ( NAME = N'bank_2014_hr_02', MAXSIZE = 1 GB )
GO

ALTER DATABASE [Bank 2014]
MODIFY FILE ( NAME = N'bank_2014_kam_01', SIZE = 500 MB, MAXSIZE = 2 GB )
GO

ALTER DATABASE [Bank 2014]
MODIFY FILE ( NAME = N'bank_2014_primary_01', MAXSIZE = 768 MB )
GO
```

14. Schreiben Sie eine Abfrage, um zu überprüfen, ob die Datendatei ordnungsgemäß angefügt wurde.

Listing 16.48: Abfragen der Datendateieigenschaften

```
USE [Bank 2014]
GO

SELECT name, size * 8 / 1024 AS Size,
       growth * 8 / 1024 AS Growth,
       max_size * 8 / 1024 AS Max_Size
FROM   sys.database_files
```

15. Recherchieren Sie, wie eine einzelne Datendatei verschoben werden kann und verschieben Sie im Anschluss die Datendatei mit dem Namen BANK_2014_KAM_01 in den Ordner D:\u01\bank_2014\data\ Welche Dateiberechtigungen müssen für die betreffende Datei gesetzt werden?

Listing 16.49: Verschieben einer Datendatei

```
USE [master]
GO

ALTER DATABASE [Bank 2014]
SET OFFLINE;

-- Move Datafile on filesystem
-- Change filesystemrights

ALTER DATABASE [Bank 2014]
MODIFY FILE ( NAME = N'bank_2014_kam_01',
FILENAME = N'D:\u01\bank_2014\data\bank_2014_kam_01.ndf' )
GO

ALTER DATABASE [Bank 2014]
SET ONLINE;
```

16. Machen Sie die Dateigruppe CRM zur Standarddateigruppe der Datenbank BANK 2014.

Listing 16.50: Ändern der Standarddateigruppe

```
USE [Bank 2014]
GO

IF NOT EXISTS (
    SELECT name
    FROM sys.filegroups
    WHERE is_default=1 AND name = N'CRM')
    ALTER DATABASE [Bank 2014]
    MODIFY FILEGROUP [CRM] DEFAULT
GO
```

17. Konfigurieren Sie für die Datenbankoption QUOTED_IDENTIFIER den Wert TRUE.

Listing 16.51: Die Datenbankoption quoted_identifier ändern

```
USE [master]
GO

ALTER DATABASE [Bank_2014]
SET QUOTED_IDENTIFIER ON
WITH NO_WAIT
GO
```

18. Ermitteln Sie welchen Wert die Datenbankoption AUTO_CLOSE hat und recherchieren Sie, welche Bedeutung diese Option hat bzw. was sie bewirkt.

Die Datenbankoption AUTO_CLOSE bewirkt, dass die Datenbank geschlossen wird, sobald sich der letzte noch aktive Nutzer abgemeldet hat. Dies hat zur Folge, dass alle im Arbeitsspeicher befindlichen Teile der Datenbank von dort entfernt werden, wodurch die Performance dieser Datenbank sehr stark negativ beeinflusst wird.

19. Verschieben Sie Ihre Datenbank TEMPDB gemäß den folgenden Angaben:

Name	Größe	Wachstum	G Max.	Pfad
tempdev	1 G	+ 200 M	2 G	G:\u04\tempdb\data
templog	200 M	+ 50 M	500 M	F:\u03\tempdb\log

Listing 16.52: Verschieben der Systemdatenbank tempdb

```
USE [master]
GO

ALTER DATABASE [tempdb]
MODIFY FILE ( NAME = N'tempdev',
               FILENAME = N'G:\u04\tempdb\data\tempdb.mdf' )
GO

ALTER DATABASE [tempdb]
MODIFY FILE ( NAME = N'templog',
               FILENAME = N'F:\u03\tempdb\log\templog.ldf' )
GO

-- Shutdown the instance

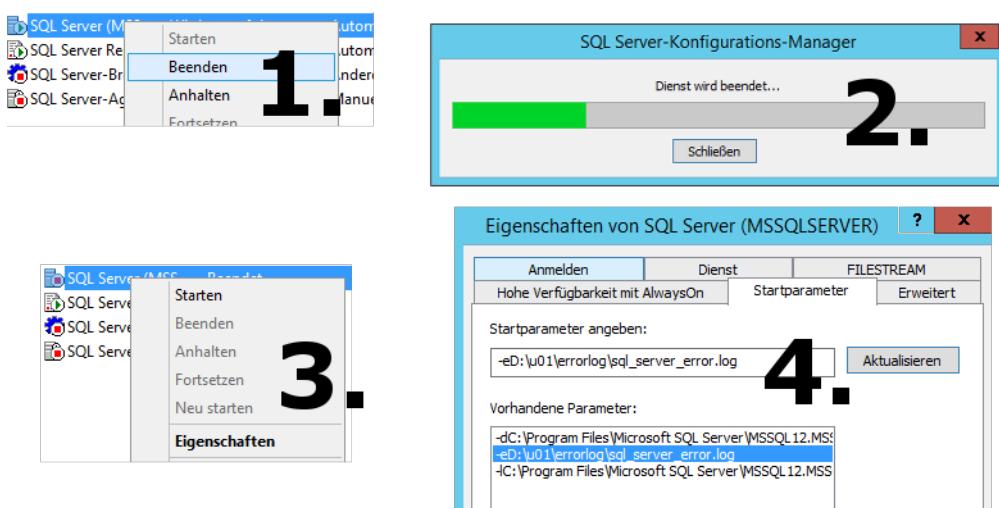
-- Move file on filesystem to their new locations

-- Change filesystemrights

-- Startup the instance
```

20. Verschieben Sie das Fehlerprotokoll Ihrer Instanz an den folgenden neuen Speicherort: D:\u01\errorlog\sql.log

Abb. 16.37:
Verschieben des
errorlog



21. Beim Anlegen der Datenbank ist Ihnen ein Fehler unterlaufen. Statt die Datenbank BANK_2014 zu nennen, haben Sie sie BANK 2014 genannt. Recherchieren Sie, wie eine Datenbank umbenannt werden kann und ersetzen Sie das Leerzeichen im Datenbanknamen durch einen Unterstrich!

Listing 16.53: Umbenennen einer Datenbank

```
USE [master]
GO

ALTER DATABASE [Bank 2014]
Modify Name = Bank_2014
GO
```

17 Filestream

Inhaltsangabe

17.1 FileStream

Mit dem FileStream-Feature ist es möglich, nicht strukturierte Daten (PDFs, Videos, Audiodateien, Bilder), sogenannte BLOB-Daten, auf dem Dateisystem abzulegen und diese gleichzeitig mit der Datenbank zu verknüpfen. Der Zugriff auf die Daten kann dann mittels TSQL, aber auch über das Dateisystem erfolgen.

17.1.1 Wie funktioniert FileStream

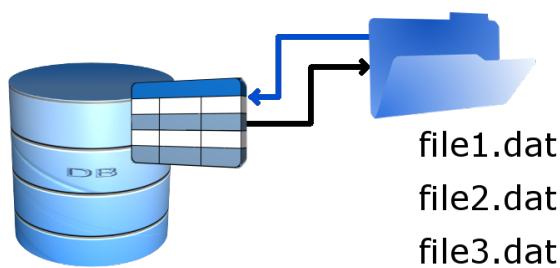
Implementierung im Dateisystem

Die Realisierung von FileStream geschieht mit Hilfe der in Microsoft Windows integrierten „Win32-Dateisystemschnittstelle“ (API¹). Diese API ermöglicht es dem SQL Server einen Win32-Namespace² zu überwachen um so alle Änderungen, die dort stattgefunden haben, zu registrieren.

Implementierung in SQL Server

Für den Zugriff mittels TSQL muss in einer Datenbank eine FileStream-Dateigruppe erstellt werden, in der dann, seit der Version 2012 des SQL Servers, eine sogenannte „FileTable“ plaziert werden kann. FileTables sehen aus wie normale Tabellen, greifen aber, unter Zuhilfenahme von SQL Server-eigenen Routinen, auf das Dateisystem zu. So kann mit `SELECT` der Verzeichnisinhalt ausgelesen und mittels `INSERT`, `UPDATE` und `DELETE` geändert werden.

Abb. 17.1:
FileStream



¹API = Application Programming Interface

²Namespaces sind UNC-Pfade, die den Zugriff auf Betriebssystemressourcen (Netzwerkfreigaben, Geräte, WMI, u. ä ermöglichen)

Der Neue - VARBINARY(max)

Um die Speicherung von BLOBs in der FileTable zu ermöglichen bringt der SQL Server einen neuen Datentyp mit, den VARBINARY(MAX)-Typ. Dieser kann unstrukturierte Binärdaten in der Datenbank und im Dateisystem ablegen. Bei der Ablage der Daten im FileStream gibt es keine Größenbeschränkung für diesen Datentyp.



Wird eine VARBINARY(MAX)-Spalte nicht im Dateisystem, sondern in der Datenbank abgelegt, besitzt sie eine Größenbeschränkung von 2 GB.

17.1.2 FileStream konfigurieren

Das FileStream-Feature ist nicht standardmäßig aktiviert. Wird es benötigt, muss der Administrator sowohl im SQL Server Configurations Manager, als auch im Management Studio einige Klicks vornehmen.

FileStream im SQL Server Configurations Manager

In einem ersten Schritt muss das FileStream-Feature im SQL Server Configurations Manager für den Windows-Dienst MSSQLSERVER aktiviert werden.

1. Öffnen Sie den SQL Server Configurations Manager
2. Wählen Sie die Rubrik „SQL Server-Dienste“
3. Öffnen Sie die Eigenschaften des SQL Server-Dienstes (MSSQLSERVER)
4. Klicken Sie die Registerkarte „FILESTREAM“ an.
5. Aktivieren Sie die beiden Optionen „FILESTREAM für Transact-SQL-Zugriff aktivieren“ und „FILESTREAM für E/A-Dateizugriff aktivieren“.
6. Starten Sie den SQL Server-Dienst neu!

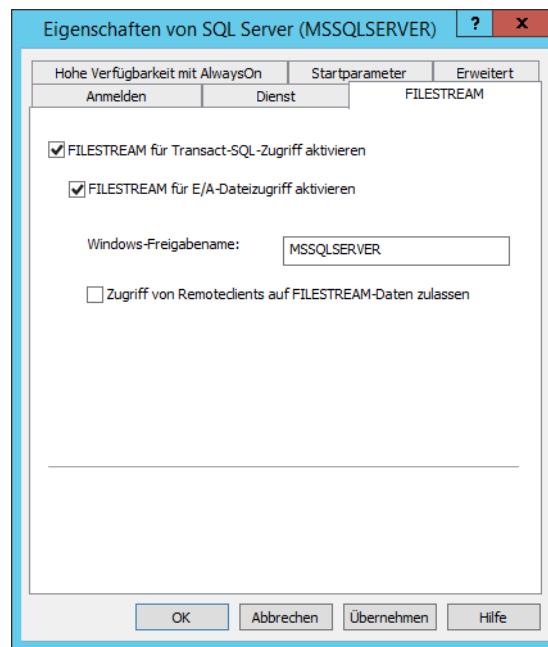


Abb. 17.2:
Die Registerkarte
FILESTREAM

FileStream im Management Studio

Der zweite Schritt ist das Aktivieren des FileStream-Features auf Instanzebene. Obwohl beide Schritte identisch und somit redundant erscheinen, ist die Trennung zwischen dem Windows-Dienst und der Instanz durchaus sinnvoll. Auf den Windows-Dienst sollten im Normalfall nur Betriebssystemadministratoren Zugriff haben, nicht jedoch die Datenbankadministratoren³. Umgekehrt sollte ein Betriebssystemadministrator keinen Zugriff auf die SQL Server-Instanz haben.

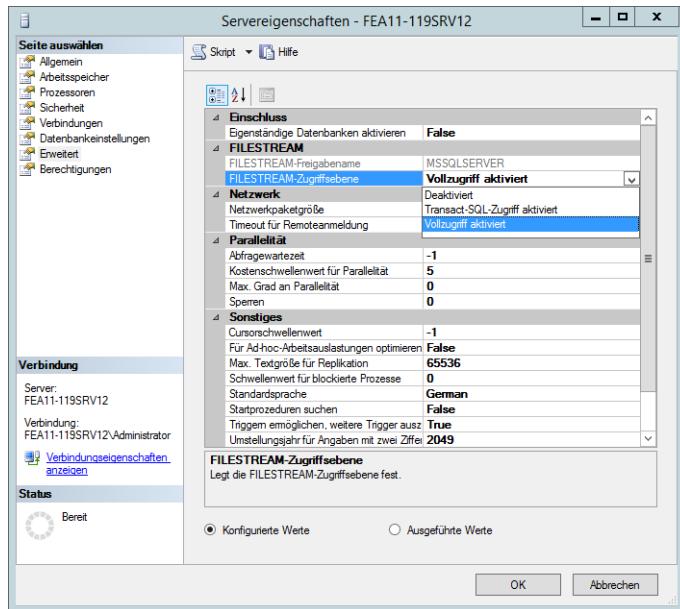
1. Öffnen Sie die Eigenschaften Ihrer SQL Server-Instanz
2. Klicken Sie auf die Rubrik „Erweitert“
3. Geben Sie für die Datenbankoption „FILESTREAM-Zugriffsebene“ den Wert „Vollzugriff aktiviert“ an.



- [llobelwp20101212ss2fp2o4]

³Anmerkung des Autors: In der Praxis herrscht hier meist Personalunion bei diesen beiden Zuständigkeiten!

Abb. 17.3:
Filestream im
Management
Studio



17.1.3 Eine FileStream-Dateigruppe anlegen

Filestream-Dateigruppen sind spezielle Dateigruppen. Sie stellen die Schnittstelle zum Dateisystem her, die dann von den FileTables genutzt werden kann.

Anlegen der Dateigruppe

Eine FileStream-Dateigruppe wird auf die gleiche Art und Weise angelegt, wie eine normale Dateigruppe, lediglich das Schlüsselwort `CONTAINS FILESTREAM` kommt hinzu.

Listing 17.1: Eine FileStream-Dateigruppe anlegen

```
USE [Bank_2014]
GO

ALTER DATABASE Bank_2014
ADD FILEGROUP FSstorage CONTAINS FILESTREAM;
GO
```

Mit dem Kommando aus Beispiel ?? wird der Datenbank BANK_2014 eine FileStream-Dateigruppe namens FSSTORAGE hinzugefügt. Genau wie bei normalen Dateigruppen wird diese als Standarddateigruppe markiert, da sie die erste ist. Sollten noch weitere FileStream-Dateigruppen existieren, kann diese Markierung, unabhängig von den normalen Dateigruppen, verschoben werden.

Hinzufügen eines Data Containers

Die Dateigruppe FSSTORAGE existiert zwar, aber noch können dort keine Daten abgelegt werden, da noch kein „Speicherort“ definiert wurde. Mit Hilfe der `ADD FILE`-Klausel des `ALTER DATABASE`-Statements kann ein Verzeichnis angegeben werden, in dem die FileStream-Daten abgelegt werden. Dieses Verzeichnis wird als „Data Container“ bezeichnet.



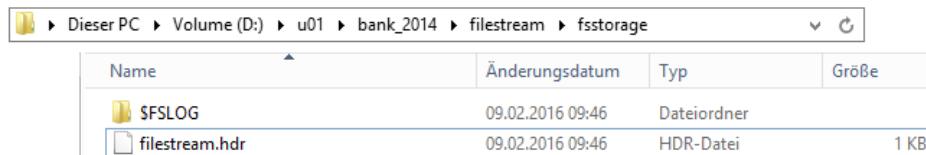
Damit die Definition des Data Containers erfolgreich sein kann, muss der Pfad D:\u01\bank_2014\filestream schon vorher existieren. Das Verzeichnis fsstorage darf jedoch noch nicht existieren.

Listing 17.2: Einen Data-Container hinzufügen

```
USE [Bank_2014]
GO

ALTER DATABASE Bank_2014
ADD FILE (
    NAME      = 'fsstorage',
    FILENAME  = 'D:\u01\bank_2014\filestream\fsstorage',
) TO FILEGROUP FSstorage;
GO
```

Die Auswirkungen von Beispiel ?? sind, dass das Verzeichnis `fsstorage` angelegt wird. Darin befinden sich nun die Datei `filestream.hdr` und das Verzeichnis `$FSLOG`.



Name	Änderungsdatum	Typ	Größe
\$FSLOG	09.02.2016 09:46	Dateiordner	
filestream.hdr	09.02.2016 09:46	HDR-Datei	1 KB

Abb. 17.4:
Ein Data Container

Die Datei `filestream.hdr` ist eine wichtige Headerdatei, ohne die die FileStream-Dateigruppe nicht funktionieren kann.

17.2 FileTables

In der soeben fertiggestellten FileStream-Dateigruppe können nun FileTables angelegt werden. Jede FileTable erstellt einen eigenen Verzeichnisbaum innerhalb der Dateigruppe. Das hat den Vorteil, dass mehrere FileTables, für unterschiedliche Zwecke in der gleichen FileStream-Dateigruppe, angelegt werden können und das trotzdem die Daten sauber von einander getrennt gespeichert werden.

17.2.1 Konfigurieren des Features FileTable

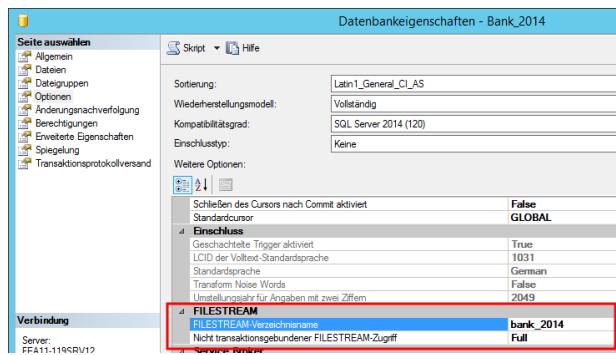
Für den transaktionsbasierten Zugriff mit TSQL sind keine weiteren vorbereitenden Schritte notwendig, da sich FileTables wie ganz normale Tabellen verhalten. Lediglich der externe Zugriff über das Dateisystem muss noch konfiguriert werden.

Konfigurationstätigkeiten auf Datenbankebene

Nachdem das FileStream-Feature auf Instanz-Ebene bereits aktiviert und konfiguriert wurde, müssen auf Datenbankebene noch zwei weitere Eigenschaften eingestellt werden.

- **Nicht transaktionaler Zugriff:** Der externe Zugriff auf die FileStream-Daten einer Datenbank kann in drei Stufen geregelt werden:
 - **Off:** Kein externe Zugriff
 - **ReadOnly:** Nur lesender Zugriff
 - **Full:** Lese- und Schreibzugriff
- **Database-level directory:** Für jede Datenbank muss ein eigener Verzeichnisknotenpunkt angegeben werden. Dies geschieht in den Eigenschaften der Datenbank.

Abb. 17.5:
Der
Datenbankpfad in
der FileStream-
Freigabe



Wird eine neue Datenbank angelegt, enthält diese kein Database-level Directory. Dieses muss im Nachhinein durch eine `ALTER DATABASE`-Anweisung oder mittels der grafischen Oberfläche angegeben werden.

Beispiel ?? zeigt, wie die FileStream-Einstellungen einer Datenbank mit Hilfe von SQL geändert werden können.

Listing 17.3: FileStream-Einstellungen einer Datenbank ändern

```
USE [Bank_2014]
GO

ALTER DATABASE Bank_2014
SET FILESTREAM ( NON_TRANSACTED_ACCESS = FULL ,
```

```
DIRECTORY_NAME = N'directory_name' );
```

GO

- [gg509097]



Für die Einrichtung eines Database-Level Directory gelten die folgenden Anforderungen:

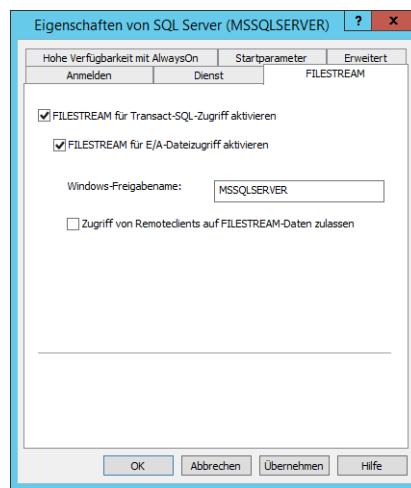
- Das Verzeichnis kann solange geändert werden, wie die Datenbank noch keine FileTable enthält.
- Jede Datenbank muss ihr eigenes Verzeichnis haben.
- Wird eine vorhandene Datenbank auf SQL Server 2014 aktualisiert, wird der Wert für DIRECTORY_NAME auf NULL zurückgesetzt
- Beim Löschen der Datenbank werden alle Inhalte im Database-Level Directory gelöscht.

Der Aufbau des externen Zugriffspfades

Der externe E/A-Zugriff auf eine FileTable wird mit Hilfe einer virtuellen Netzwerksfreigabe geregelt. Die gesamte Verzeichnisstruktur der Freigabe wird innerhalb der Datenbank gespeichert und über das Betriebssystem bereitgestellt. Der UNC-Pfad der Freigabe baut sich wie folgt auf:

- \\servername: Der DNS-Name des Datenbankservers
- Instanz-Level-Freigabename: Der Name der Freigabe auf Instanz-Ebene. Dieser wird im SQL Server Configuration Manager angegeben. Standardmäßig lautet er MSSQLSERVER. Um diesen Wert ändern zu können, muss der SQL Server-Dienst neustartet werden.

Abb. 17.6:
Die Registerkarte
FILESTREAM



- Database-Level directory
- FileTable directory: Ein Verzeichnis, das beim anlegen der FileTable angegeben werden muss.

Der vollständige UNC-Pfad (Namespace) lautet somit:

```
\server\Instanz-Level-Freigabenname\Database-level directory\FileTable-directory
```



- [gg492087]

17.2.2 Anlegen und Ändern von FileTables

Anlegen einer FileTable

Eine FileTable ist nicht nur wegen ihrer Funktionalität eine besondere Tabelle, sondern auch aufgrund der Tatsache, dass sie ein festes Spaltenschema besitzt. Das bedeutet, dass beim Anlegen einer FileTable keine Spaltendefinition angegeben werden muss bzw. angegeben werden kann. Lediglich drei Angaben sind erlaubt:

- **FILETABLE_DIRECTORY**: Ein Verzeichnis, unter dem die Dateien der FileTable abgelegt werden. Dieses Verzeichnis ist Teil des externen Zugriffspfades für die FileTable.
- **FILETABLE_COLLATE_FILENAME**: Eine Sortierung für die Dateinamen. Die benutzte Sortierung darf nicht nach Groß- und Kleinschreibung unterscheiden, damit die Windows-Dateisystemsemantik nicht verletzt wird.
- Die Bezeichner für die drei festgelegten Constraints einer FileTable:
 - **FILETABLE_PRIMARY_KEY_CONSTRAINT_NAME**
 - **FILETABLE_STREAMID_UNIQUE_CONSTRAINT_NAME**
 - **FILETABLE_FULLPATH_UNIQUE_CONSTRAINT_NAME**

Beschränkungen für Pfad- und Dateinamen

Beim Anlegen von Dateien und Verzeichnissen innerhalb einer FileTable muss folgendes beachtet werden:

- Der Verzeichnisname kann maximal 255 Zeichen aufweisen.
- Er muss den Anforderungen an einen gültigen Dateisystem-Verzeichnisnamen genügen.

- Er muss für jede FileTable eindeutig sein.
- Wird kein Verzeichnisname angegeben, wird der Name der FileTable als Verzeichnisname genutzt.
- Eine FileTable kann maximal 15 Verzeichnisebenen enthalten. Wird diese Maximalzahl erreicht, kann die unterste Ebene keine Dateien mehr enthalten, da diese als weitere Ebene gelten würden.
- Der Windows Explorer und viele andere Anwendungsprogramme geben eine maximale Dateinamenslänge von 260 Zeichen vor. Mit Hilfe einer FileTable ist es möglich, Dateinamen zu erzeugen, die wesentlich länger sind. Solche Dateien können weder durch den Explorer noch durch andere Anwendungsprogramm angezeigt werden.



- [gg492087]

Beispiel ?? zeigt, wie mit Hilfe des `CREATE TABLE AS FileTable`-Statements eine FileTable angelegt wird.

Listing 17.4: Anlegen einer Filetable

```
USE [Bank_2014]
GO

CREATE TABLE ExternalResources AS FileTable
WITH (
    FileTable_Directory = 'ExternalResources',
    FileTable_Collate_Filename = Latin1_General_CI_AS
);
GO
```

Um eine Liste aller in der Datenbank vorhandenen FileTables zu erhalten, kann die View `SYS.FILETABLES` abgefragt werden.



- [gg509088]

Das Schema einer FileTable

Eine FileTable ist eine ganz gewöhnliche SQL Server-Tabelle mit der Einschränkung, dass ihr Schema fest vorgegeben ist. Die vier wichtigsten Spalten aus diesem Schema sind in Tabelle ?? zu sehen.

Bezeichner	Datatype	Beschreibung
stream_id	uniqueidentifier ROWGUIDCOL	Eindeutiger Schlüsselwert
file_stream	VARBINARY(max) FILESTREAM	Der binäre Content der FileTable (BLOB)
name	NVARCHAR(255)	Der Name der Datei
path_locator	hierarchyid	Logische Position der Datei innerhalb des SQL Server Namespaces

Die PATH_LOCATOR-Spalte ist der Primärschlüssel einer FileTable. Sie besitzt den spezielle Datentyp HIERARCHYID, der mit dem SQL Server 2008 eingeführt wurde. Mit der Hilfe dieser Spalte ist es möglich, die Position des BLOBs im Dateisystem zu ermitteln, um dann auf die Datei zuzugreifen.

Zusätzlich zu diesen vier Spalten existieren noch 13 weitere, deren Werte automatisch berechnet werden.



- [gg492084]
- [tallan20120101iafsiad]

Ändern und Löschen einer FileTable

Da FileTables ein festgelegtes Schema haben, kann der Administrator auch keine Änderung daran vornehmen. Lediglich die Erstellung von Indizes, Triggern und Constraints, sowie einige andere, weniger erwähnenswerte Dinge sind möglich.

Gelöscht wird eine FileTable mit Hilfe des `DROP TABLE`-Statements. Zu beachten ist, dass der gesamten Dateisysteminhalt der FileTable mitgelöscht wird.

17.2.3 Benutzen von FileTables

Befüllen mit Daten

Eine FileTable kann auf unterschiedliche Art und Weise mit Daten gefüllt werden. Im einfachsten Falle werden Dateien einfach mit Hilfe des externen Zugriffspfades in die FileTable hineinkopiert. Ein solcher Zugriff ist jedoch nicht durch das Transaktionskonzept der Datenbank abgesichert. Soll ein transaktionaler Zugriff erfolgen, bieten sich Tools wie z. B. bcp an. Sollte der Zugriffspfad einer FileTable nicht bekannt sein, kann dieser mit der Funktion FILETABLEROOTPATH ermittelt werden.

Listing 17.5: Zugriffspfad zu einer FileTable ermitteln

```
USE [Bank_2014]
GO

SELECT FileTableRootPath('ExternalResources') AS Rootpath
```

**Rootpath**

\\FEA11-119SRV12\MSSQLSERVER\bank_2014\ExternalResources
1 Zeile

Müssen Dateien aus einer normalen Tabelle (gespeicherte Pfade und Dateinamen) in eine FileTable migriert werden, können SQL-Anweisungen wie das `OPENROWSET` oder das `INSERT INTO ... SELECT * FROM` genutzt werden.

Listing 17.6: Mit TSQL eine Datei in eine FileTable einfügen

```
USE [Bank_2014]
GO

INSERT INTO ExternalResources (name, File_Stream)
SELECT 'bank_sql_server.bak', *
FROM OPENROWSET(BULK N'H:\bank_sql_server.bak', SINGLE_BLOB) AS Content
```



- [\[sqlarticlesagwwf\]](#)

Den transaktionalen Zugriff aktivieren/deaktivieren

Im Falle dessen, dass Wartungsarbeiten an einer Datenbank mit aktiven FileTables durchgeführt werden müssen, kann es notwendig sein, den nicht transaktionalen Zugriff auf die FileTables vorübergehend zu deaktivieren, da der Administrator anderenfalls keinen exklusiven Zugriff auf die Datenbank erhalten kann.

Beispiel ?? zeigt, wie der nicht transaktionale Zugriff auf alle FileTables einer Datenbank deaktiviert bzw. reaktiviert wird.

Listing 17.7: Deaktivieren und reaktivieren des Zugriffs auf FileTables

```
USE [Bank_2014]
GO

-- Deactivate Access
```

```

ALTER DATABASE bank_2014
SET FILESTREAM ( NON_TRANSACTED_ACCESS = OFF );
GO

-- Reactivate Access
ALTER DATABASE bank_2014
SET FILESTREAM ( NON_TRANSACTED_ACCESS = FULL );
GO

```

Das Kommando in Beispiel ?? hat den Nachteil, dass es blockiert, falls noch offene Dateihandles in einer der FileTables sind. Entweder müssen die Benutzer alle Dateien schließen oder der Administrator muss den Zugriff der Benutzer auf die Dateien unterbrechen.



Beim schließen von offenen Dateihandles durch den Administrator kann es passieren, dass Benutzer nicht gespeicherte Informationen verlieren.

Wird das `ALTER DATABASE`-Kommando um den Zusatz `WITH ROLLBACK IMMEDIATE` ergänzt, werden automatisch alle offenen Dateihandles geschlossen.

Listing 17.8: Deaktivieren und des Zugriffs auf FileTables und gleichzeitiges Schließen aller offenen Dateihandles

```

USE [Bank_2014]
GO

-- Deactivate Access
ALTER DATABASE bank_2014
SET FILESTREAM ( NON_TRANSACTED_ACCESS = OFF )
WITH ROLLBACK IMMEDIATE;
GO

```

Offene Dateihandles beenden

Manchmal kann es notwendig werden, ein offenes Dateihandle zu schließen, z. B. dann, wenn dies aufgrund eines Fehlers nicht automatisch geschieht. Mit Hilfe der Dynamic Management View `SYS.DM_FILESTREAM_NON_TRANSACTED_HANDLES` können alle offenen handles abgefragt werden, um sie dann unter Zuhilfenahme der Stored Procedure `SP_KILL_FILESTREAM_NON_TRANSACTED_HANDLE` zu schließen.

Listing 17.9: Abfragen der offenen FileStream-Dateihandles

```

USE [Bank_2014]
GO

SELECT database_id, object_id, handle_id

```

```
FROM sys.dm_filestream_non_transacted_handles;
GO
```

database_id	object_id	handle_id
0	0	235
8	581577110	525
8	581577110	491

3 Zeilen ausgewählt



Zum schließen des Handles wird der Wert der Spalte HANDLE_ID benötigt.

Listing 17.10: Schließen eines offenen FileStream-Dateihandles

```
USE [Bank_2014]
GO

EXEC sp_kill_filestream_non_transacted_handles @handle_id = 525;
GO
```



- [ff929106]
- [ff929168]

17.3 Projekt - Benutzen von FileStream und FileTables

1. Bereiten Sie Ihre SQL Server-Instanz auf die Benutzung von FileStream vor. Der Bezeichner für die FileStream-Freigabe lautet: MSSQLSERVER. Der Bezeichner für das Verzeichnis der Datenbank BANK_2014 lautet: BANK_2014. Aktivieren Sie den nichttransaktionalen Zugriff auf den FileStream.
2. Führen Sie das Skript ADD_FOTO_COLUMN.SQL aus. Dieses Skript fügt der Tabelle MITARBEITER in der Datenbank BANK_2014 eine Spalte namens FOTO hinzu. Kopieren Sie in das Verzeichnis E:\u02\bank_2014\data\fotos.
3. Legen Sie eine neue FileStream-Dateigruppe names EXTERNALRESOURCES an. Das FileStream-Verzeichnis soll D:\u01\bank_2014\filestream\fsstorage sein. Benutzen Sie den Bezeichner BANK_2014_EXTERNALRESOURCES als logischen Namen für das FileStream-Verzeichnis.
4. Erstellen Sie eine neue FileTable mit dem Namen FOTOS. Benutzen Sie als Bezeichner für das FileTable-Verzeichnis: MITARBEITER_FOTOS.
5. Benutzen Sie die „OpenRowset“-Funktionalität des SQL Servers, um die Fotos der Mitarbeiter in die FileTable FOTOS zu importieren. Verknüpfen Sie die Inhalt der Tabelle MITARBEITER mit den Fotos in der Tabelle FOTOS. Welches Attribut der FileTable eignet sich dafür? Benutzen Sie migrate_fotos_to_filetable_template.sql. hierfür als Vorlage.



• [ms190312]

18 Schemaobjekte verwalten

Inhaltsangabe

18.1 Heap Organized Tables

Heap-Organized Tables sind die Basisdatenstruktur einer Datenbank. Daten werden in Zeilen und Spalten abgelegt. Jede Tabelle wird mit einem Namen und einer Anzahl von Spalten definiert. Jede Spalte hat einen Bezeichner, einen Datentyp und eine Länge.

Bei dieser Art von Tabelle werden die Zeilen unsortiert, in der Reihenfolge ihrer Erstellung abgelegt, was sich jedoch mit der Zeit ändern kann. Wird ein Heap über einen längeren Zeitraum genutzt, passiert es früher oder später, dass die Database Engine Zeilen verschiebt, um diese effizienter zu speichern. Somit ist die Reihenfolge der Zeilen in einem Heap nicht klar vorhersagbar.

Abb. 18.1:
Heap-Organized
Table

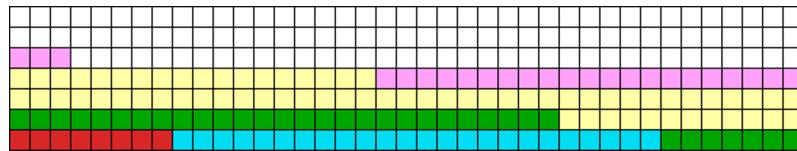


Abbildung ?? zeigt, dass die Tabellenzeilen, hier durch rote, blaue, grüne und gelbe Kästchen dargestellt, einfach nacheinander abgelegt werden. Dies ist vergleichbar mit einem „Haufen“ (engl. Heap) bei dem alle Elemente einfach aufeinander geworfen werden.

Abb. 18.2:
Die Struktur eines
Heaps

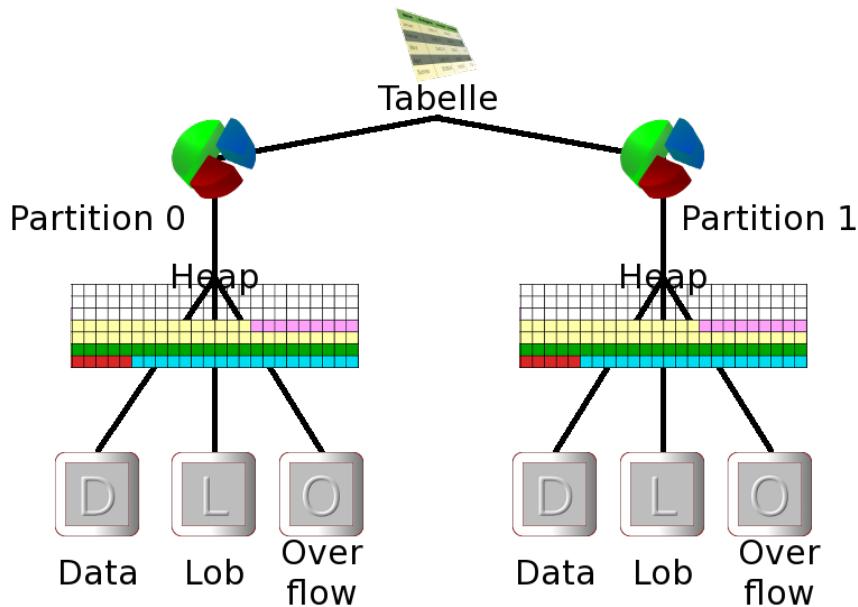


Abbildung ?? zeigt den Aufbau einer als Heap organisierten Tabelle. Jede Tabelle dieser Art:

- besteht aus einer oder mehreren Partitionen,
- in jeder Partition liegt ein Heap und
- jeder Heap besteht im Minimum aus einer „Data Allocation Unit“ und kann zusätzlich noch „LOB Allocation Units“ und „Row Overflow Space“ enthalten.

18.1.1 Partitionen

Den Begriff „Partition“ definiert Microsoft wie folgt: „A partition is a user-defined unit of data organization“. Das heißt, dass Partition benutzerdefinierte Datenstrukturen sind, die eine Tabelle oder einen Index enthalten können. Partitionen im Sinne des Microsoft SQL Server kann man sich genauso vorstellen, wie Partitionen in Bezug auf Festplatten. Sie dienen dazu, ein großes Ganzes in mehrere Stücke zu unterteilen (eine Datendatei wird in Stücke unterteilt → eine Festplatte wird in mehrere Stücke unterteilt). Eine Partition hat folgende Eigenschaften:

- In einer Partition ist immer nur eine Tabelle/ein Index enthalten.
- Jede Partition gehört zu genau einer Dateigruppe. Sie kann sich nicht auf mehrere verteilen.
- Wenn eine SQL-Abfrage auf eine Tabelle ausgeführt wird, werden automatisch alle Partitionen, aus denen die Tabelle besteht zusammengefasst, d. h., dass Partitionen in einem `SELECT`-Statement nicht berücksichtigt werden müssen. Daraus folgt: Partitionen sind für den Benutzer absolut transparent.
- Partitionen können mittels der View `sys.partitions` angezeigt werden.

Besteht eine Tabelle aus mehreren Partitionen, liegt in jeder Partition eine Heap-Struktur. Die Zeilen der Tabelle werden dann horizontal über die Heaps verteilt. Gleiches gilt für Indizes.



18.1.2 Allocation Units

Jeder Heap wird nochmals unterteilt in sogenannten „Allocation Units“. Eine Allocation Unit ist eine Sammlung von Seiten (Datenseiten, Indexseiten, LOB-Seiten, usw.) und dient dazu, die Verwaltung der Seiten zu vereinfachen. Es gibt drei verschiedene Seitentypen:

- **IN_ROW_DATA:** Diese Allocation Units enthalten nur Daten- oder Indexseiten und dienen zur Speicherung von „normalen“ Daten (Textdaten bis zu einer Länge von 8.060 Byte).
- **ROW_OVERFLOW_DATA:** Hier werden Daten von Datentypen mit variabler Länge (VARCHAR, NVARCHAR, VARBINARY oder SQL_VARIANT) abgelegt, sobald deren Inhalt die Länge von 8.060 Byte überschreitet.
- **LOB_DATA:** Unter einem LOB¹ versteht man umfangreiche Text- oder Binärdaten, welche die Datenbank unformatiert ablegen soll. So kann z. B. ein MS Word-Dokument als LOB in einer Datenbank gespeichert werden. Zur Ablage von LOBs werden heutzutage hauptsächlich die vier Datentypen VARCHAR(max), NVARCHAR(max), VARBINARY(max) und XML genutzt.



Jeder Heap kann maximal eine Allocation Unit von jedem Typ in einer Partition haben!



- [ms175012]
- [ms188706]
- [ms189051]

18.1.3 Index Allocation Maps

Index Allocation Maps - kurz IAM-Pages - haben die Aufgabe, eine Verkettung zwischen allen Speicherseiten/Extents herzustellen, die zu einer Allocation Unit gehören. Mit ihrer Hilfe kann also erkannt werden, welche Page bzw. welches Extent zu welcher Allocation Unit gehört.

Diese Seiten können einen bis zu 4 GB großen Bereich in einer Datendatei, der als „GAM interval“ bezeichnet wird, verwalten. An der Bezeichnung „GAM interval“ kann man erkennen, dass sie in engem Zusammenhang mit den GAM- und SGAM-Seiten einer Datenbank steht, denn zu jeder GAM-/SGAM-Seitenkombination gibt es eine IAM-Page.



Eine IAM-Page kann immer nur den Speicherplatz eines einzelnen GAM intervals verwalten. Ist eine Datendatei größer als 4 GB oder besteht eine Datenbank aus mehr als einer Datendatei, werden mehrere verkettete IAM-Pages angelegt. Man spricht in so einem Fall von einer „IAM-Chain“.

¹LOB = Large Object Block

Ihr Aussehen unterscheidet sich von dem anderer Seiten leicht. Nach dem üblichen 96 Byte großen Header kommt in einer IAM-Page ein „IAM-Header“. Dieser Header zeigt auf die erste Speicherseite, die von dieser IAM-Page verwaltet wird. Des Weiteren enthält der IAM-Header insgesamt acht Einträge, in denen auf die ersten acht Extents (Mixed Extents) einer Allocation Unit verwiesen wird.

Listing 18.1: Inhalt eines IAM-Page Headers

```
PAGE HEADER:

Page @0x00000000F16FA000

m_pageId = (1:120)          m_headerVersion = 1          m_type = 10
m_typeFlagBits = 0x0        m_level = 0            m_flagBits = 0x200
m_objId (AllocUnitId.idObj) = 120      m_indexId (AllocUnitId.idInd) = 256

Metadata: AllocUnitId = 72057594045792256
Metadata: PartitionId = 72057594040549376          Metadata: IndexId = 0
Metadata: ObjectId = 245575913   m_prevPage = (0:0)      m_nextPage = (0:0)
pminlen = 90                  m_slotCnt = 2           m_freeCnt = 6
m_freeData = 8182             m_reservedCnt = 0       m_lsn = (97:106:7)
m_xactReserved = 0            m_xdesId = (0:0)        m_ghostRecCnt = 0
m_tornBits = -1344281328      DB Frag ID = 1

Allocation Status

GAM (1:2) = ALLOCATED          SGAM(1:3) = ALLOCATED
PFS (1:1) = 0x70 IAM_PG MIXED_EXT ALLOCATED    O_PCT_FULL     DIFF (1:6) = CHANGED
ML (1:7) = NOT MIN_LOGGED

IAM: Header @0x0000000007C6A064 Slot 0, Offset 96

sequenceNumber = 0              status = 0x0          objectId = 0
indexId = 0                     page_count = 0
start_pg = (1:0)

IAM: Single Page Allocations @0x0000000007C6A08E

Slot 0 = (1:166)      Slot 1 = (1:178)      Slot 2 = (1:179)
Slot 3 = (1:180)      Slot 4 = (1:181)      Slot 5 = (1:183)
Slot 6 = (1:184)      Slot 7 = (1:188)
```

Die rot markierten Bereiche in Beispiel ?? zeigen:

- die PageID (1:120),
- den Seitentyp einer IAM-Seite (10),

- die SeitenIDs der GAM- (1:2) und SGAM-Pages (1:3),
- die erste Speicherseite, die sich im GAM interval dieser IAM-Page befindet (1:0)
- und die Liste der „IAM Single Page Allocations“ mit ihren acht Slots, in die die ersten acht Seiten einer Allocation Unit eingetragen werden.



Wenn eine Allocation Unit mehr als eine IAM-Page hat, wird nur bei der ersten IAM-Page der Bereich „IAM Single Page Allocations“ gefüllt.

Der Speicherbereich nach dem IAM Header wird von einer 8.000 Bit großen Bitmap belegt. Genau wie bei einer GAM-Page steht jedes Bit stellvertretend für ein Extent, genauer gesagt für ein Uniform Extent.

Listing 18.2: Die Extent Allocation Status Slots einer IAM-Page

IAM:	Extent Alloc	Status Slot	Slot 1 @0x000000007C6A0C2
(1:0)	- (1:240)	= NOT ALLOCATED	
(1:248)	-	= ALLOCATED	
(1:256)	-	= NOT ALLOCATED	
(1:264)	- (1:272)	= ALLOCATED	
(1:280)	- (1:312)	= NOT ALLOCATED	
(1:320)	-	= ALLOCATED	
(1:328)	-	= NOT ALLOCATED	
(1:336)	- (1:352)	= ALLOCATED	
(1:360)	- (1:384)	= NOT ALLOCATED	
(1:392)	-	= ALLOCATED	
(1:400)	-	= NOT ALLOCATED	
(1:408)	- (1:672)	= ALLOCATED	
(1:680)	- (1:792)	= NOT ALLOCATED	

Beispiel ?? zeigt diese Bitmap in Textform. Die Bedeutung dieser Anzeige ist:

- Der Bereich mit den Datapages 0 bis einschließlich 240 in Datendatei Nummer 1 ist nicht von der hier verwalteten Allocation Unit belegt.
- Das Extent mit den Speicherseiten 248 bis einschließlich 255 ist durch die aktuell Allocation Unit belegt.
- Das Extent beginnt bei Datapage 256 ist nicht durch diese Allocation Unit belegt.

Die restlichen Zeilen aus Beispiel ?? sind analog zu interpretieren.



- [sqlityTIAMLSwt]
- [ms187501]
- [ms188270]

18.1.4 Die Zeilen ID - RID

Eine Zeilen ID² (kurz RID) ist ein eindeutiges Identifikationsmerkmal für eine Zeile in einem Heap. Sie besteht aus:

- der Dateinummer,
- der Datenseitennummer,
- und einer Slotnummer (Row Offset)

Mit diesen drei Angaben kann die Position einer Zeile innerhalb einer SQL Server Datenbank eindeutig bestimmt werden. Intern wird die RID als „Physical Locator“, kurz „physloc“ bezeichnet.

Dem Administrator ist es möglich, die RID sichtbar zu machen. Dies kann er mit Hilfe einer undokumentierten Pseudospalte namens `%%PHYSLOC%%` und der ebenfalls undokumentierten Funktion `SYS.FN_PHYSLOCFORMATTER`.

Listing 18.3: Auslesen der RID

```
USE [demo_grafisch]
GO

SELECT %%physloc%% AS [%%physloc%%],
       sys.fn_PhysLocFormatter(%%physloc%%) AS [File:PageID:Slot]
FROM   dbo.TESTTAB
```



- [stovfl909155]
- [hh213609]



²engl. Row ID

18.2 Speicheroptimierte Tabellen (SOT)

Mit dem SQL Server 2014 kommt eine neue Form der Datenspeicherung und Verarbeitung, das „In-Memory OLTP“. In-Memory OLTP ist direkt in die Database Engine des SQL Server integriert. Es ist keine zusätzliche Anwendung notwendig, um diese neue Technologie zunutzen. In-Memory OLTP wird durch zwei Dinge umgesetzt:

- Speicheroptimierte Tabellen (SOT)
- Systeminterne Kompilierung

18.2.1 Das Konzept der Speicheroptimierten Tabellen

Aufbau und Erstellung

Diese Art von Tabellen trägt ihren Namen, da sie, im Gegensatz zu den Heaps, komplett im Arbeitsspeicher gehalten wird. Auch die Verarbeitung der Daten geschieht direkt im Arbeitsspeicher.

Die Zeilen in einer SOT werden mit Hilfe von 8-Byte Zeigern adressiert und nicht, wie die Zeilen in einem Heap, mit Seitennummern und Offsets.



Eine Datenbank kann sowohl Heaps als auch SOTs enthalten!

Um Speicheroptimierte Tabellen nutzen zu können, muss eine neue Art von Dateigruppe, mit der Option `CONTAINS MEMORY_OPTIMIZED_DATA`, erstellt werden.

Listing 18.4: Erstellen einer In-Memory Dateigruppe

```
USE [master]
GO

ALTER DATABASE [demo_grafisch]
ADD FILEGROUP [in_memory_oltp]
CONTAINS MEMORY_OPTIMIZED_DATA
GO

ALTER DATABASE [demo_grafisch]
ADD FILE (
    NAME = 'in_memory_oltp',
    FILENAME = 'D:\u01\demo_grafisch\in_memory\in_memory_oltp'
```

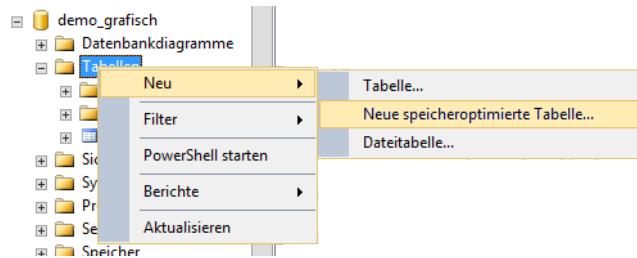
```
)  
GO
```



Wurde eine In-Memory-Filegroup erst einmal angelegt, kann sie nicht mehr gelöscht werden. Hierzu müsste die gesamte Datenbank gelöscht und neuangelegt werden.

Nach dem Erstellen der Dateigruppe kann eine erste SOT erstellt werden. Dies geschieht jedoch nicht wie gewohnt mittels eines grafischen Dialogs, sondern mit Hilfe einer TSQL-Vorlage. Dies kann mittels des Kontextmenüs im Objekt-Explorer aufgerufen werden (NEU → Neue Speicheroptimierte Tabelle).

Abb. 18.3:
Eine SOT
erstellen



Listing 18.5: Erstellen einer Speicheroptimierten Tabelle

```
-- Create Memory Optimized Table Template
-- Use the Specify Values for Template Parameters command (Ctrl-Shift-M)
-- to fill in the parameter values below.
-- This template creates a memory optimized table and indexes on
-- the memory optimized table.
-- The database must have a MEMORY_OPTIMIZED_DATA filegroup before the
-- memory optimized table can be created.
-- -- --
USE <database , sysname , AdventureWorks>
GO

--Drop table if it already exists.
IF OBJECT_ID('<schema_name , sysname , dbo>,
            <table_name , sysname , sample_memoryoptimizedtable>','U') IS NOT NULL
    DROP TABLE <schema_name , sysname , dbo>.
            <table_name , sysname , sample_memoryoptimizedtable>
GO

CREATE TABLE <schema_name , sysname , dbo>.
            <table_name , sysname , sample_memoryoptimizedtable> (
<column_in_primary_key , sysname , c1>
    <column1_datatype , , int> <column1_nullability , , NOT NULL>,
<column2_name , sysname , c2>
    <column2_datatype , , float> <column2_nullability , , NOT NULL>,
<column3_name , sysname , c3>
    <column3_datatype , , decimal(10,2)> <column3_nullability , , NOT NULL>
INDEX
<index3_name , sysname , index_sample_memoryoptimizedtable_c3>
    NONCLUSTERED (<column3_name , sysname , c3>),
CONSTRAINT <constraint_name , sysname , PK_sample_memoryoptimizedtable>
    PRIMARY KEY NONCLUSTERED (<column1_name , sysname , c1>),
INDEX <index2_name , sysname , hash_index_sample_memoryoptimizedtable_c2> HASH
(<column2_name , sysname , c2>) WITH
(BUCKET_COUNT = <sample_bucket_count , int , 131072>)
WITH (MEMORY_OPTIMIZED = ON,
      DURABILITY = <durability_type , ,
SCHEMA_AND_DATA>)
GO
```

Die entscheidende Klausel, die bewirkt, dass eine Speicheroptimierte Tabelle erstellt wird, steht am Ende des `CREATE TABLE`-Statements und lautet `WITH MEMORY_OPTIMIZED = ON`.

Zu beachten ist der Kommentar im Kopf der Vorlage. Es wird darauf hingewiesen, dass durch das Drücken von STRG+SHIFT+M ein Dialogfenster zur Ersetzung der Vorlageparameter geöffnet werden kann. Dies erspart dem Nutzer lästiges Suchen und Ersetzen von Werten.

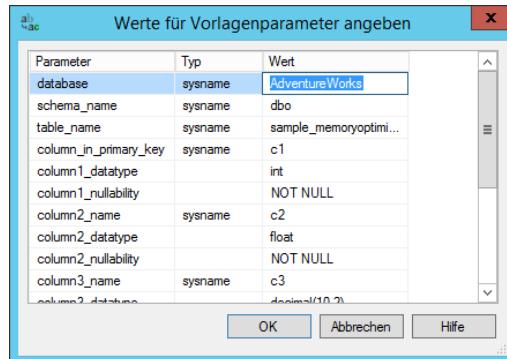


Abb. 18.4:
Werte für Vorlagenparameter angeben

Ein einfaches Beispiel für eine SOT könnte so aussehen:

Listing 18.6: Erstellen einer Speicheroptimierten Tabelle

```
USE [demo_grafisch]
GO

IF OBJECT_ID('dbo.aktie_in_memory', 'U') IS NOT NULL
    DROP TABLE dbo.aktie_in_memory
GO

CREATE TABLE dbo.aktie_in_memory
(
    Aktie_ID NUMERIC NOT NULL,
    Name      VARCHAR(25) NOT NULL,
    WKN       NUMERIC,
    ISIN      VARCHAR(12) NOT NULL,
    CONSTRAINT aktie_in_memory_pk PRIMARY KEY NONCLUSTERED (Aktie_ID)
)
WITH (MEMORY_OPTIMIZED = ON,
      DURABILITY = SCHEMA_AND_DATA)
GO
```



- [dn205318]

Beständigkeit der Daten - Persistenz

Obwohl eine SOT vollständig im Arbeitsspeicher liegt, garantiert Microsoft, dass Inhalt und Änderungen an einer SOT konsistent und persistent sind. Das heißt, dass alle Regeln bezüglich der Veränderung von Daten (`INSERT`, `UPDATE` und `DELETE`) auch für SOTs gelten. Um die Daten persistieren zu können, existiert eine Kopie der Tabelle auf dem Datenträger.

Einschränkungen bei der Benutzung

Für die Benutzung Speicheroptimierter Tabellen gibt es einige Einschränkungen:

- es können keine `FOREIGN KEY`- oder `CHECK`-Constraints benutzt werden,
- es können keine Trigger für SOTs erstellt werden,
- die Tabellendefinition kann nicht mittels `ALTER TABLE` geändert werden.
- es können keine clustered Indizes für SOTs erstellt werden,
- um Indizes auf Character-Spalten erstellen zu können, muss eine binäre Sortierreihenfolge für die Datenbank gewählt werden, z. B. `LATIN1_GENERAL_BIN2`

Diese Liste von Einschränkungen zeigt nur die Wesentlichsten, sie ist nicht vollständig.



- [dn133182]

Vorteile Speicheroptimierter Tabellen

Ob die Nutzung einer SOT vorteilhaft, im Vergleich zur Nutzung eines Heaps ist, hängt von der genutzten Anwendung ab. Eine generelle Aussage kann hierzu nicht getroffen werden.

SOTs entfallen ihre Vorteile, wenn:

- eine Anwendung hauptsächlich umfangreiche gespeicherte Prozeduren nutzt, anstatt kurzer TSQL-Aufrufe.

- hauptsächlich mit systemintern kompilierten Prozeduren zugegriffen wird und nicht Adhoc-TSQL
- die Anwendung Parallelität in hohem Maße nutzt.

Zusammenfassend kann man sagen, dass SOTs für große Anwendung, die eine hohe Zahl von Transaktionen pro Minute verarbeiten müssen vorteilhaft sind.



- [dn511014]

18.2.2 Systeminterne Kompilierung

Unter der Systeminternen Kompilierung versteht Microsoft, dass für den Zugriff auf SOTs, statt interpretiertem TSQL, nativ kompilierter TSQL-Code genutzt wird.

Was ist interpretierter TSQL-Code?

Wenn ein Nutzer ein SQL-Statement an den SQL Server schickt, wird dies durch die Relational Engine aufgenommen und verarbeitet. Wie in ?? beschrieben, nimmt zuerst der Command Parser seine Arbeit auf und verwandelt das lesbare SQL-Statement in den sogenannten Query Tree. Dieser wird durch den Query Optimizer verändert/verbessert und zuletzt durch den Query Executor, Zeile für Zeile, ausgeführt.

Das zeilenweise Analysieren und Ausführen von Quellcode wird als „Interpretieren“ bezeichnet. Diese Form der Programmausführung bietet ein hohes Maß an Flexibilität. Sie hat allerdings den Nachteil, dass sie sehr langsam ist, weil für jede einzelne Zeile alle Vorgänge (Einlesen, Analysieren, Ausführen) erneut durchgeführt werden müssen.

Was versteht man unter nativ kompiliert?

Ein Kompiler hat, genauso wie ein Interpreter, die Aufgabe, Quellcode für die Ausführung durch die CPU vorzubereiten. Der Unterschied zwischen den Beiden besteht im jeweils angewendeten Verfahren. Ein Kompiler übersetzt ein Programm vollständig in eine „maschinenlesbare Sprache“. Bezogen auf TSQL heißt das, dass der Kompiler das komplette TSQL-Statement in eine Sprache übersetzt, die ohne Umwege von der CPU ausgeführt werden kann³.

³Der Begriff „nativ“ bezieht sich darauf, dass die Sprache direkt von der CPU ausgeführt werden kann

18.2.3 Systemintern kompilierte Tabellen und Prozeduren

Systemintern kompilierte Tabellen

Wird eine Tabelle erstellt, wird deren Definition in die Metadaten der betreffenden Datenbank eingetragen. Dies gilt sowohl für Heaps, als auch für Speicheroptimierte Tabellen. Bei den letzteren wird die Definition zusätzlich auch in nativ kompilierter Form, in einer DLL-Datei abgelegt. So kann eine SOT den immensen Geschwindigkeitsvorteil erreichen, denn ihre Definition liegt immer in prozessorlesbarer Form vor und muss nicht erst bei Bedarf neu interpretiert werden.

Ändern von systemintern kompilierten Tabellen

Da die Definition einer systemintern kompilierten Tabelle nicht nur in den Metadaten einer Datenbank, sondern auch in einer DLL-Datei gespeichert ist, kann sie nicht so einfach geändert werden, wie bei einer normalen Tabelle.

Da der Vorgang zum Ändern der Definition einer systemintern kompilierten Tabelle sehr umständlich und komplex ist, wird an dieser Stelle lediglich auf die Microsoft MSDN verwiesen.



- [dn269114]

Systemintern kompilierte gespeicherte Prozeduren

Gespeicherte Prozeduren sind kleine, in der Sprache TSQL geschriebene, Programme. Dies müssen, genau wie jedes andere TSQL-Statement, bei jeder Ausführung interpretiert werden. Um auch hier einen Geschwindigkeitsvorteil erzielen zu können, gibt es seit SQL Server 2014 die Möglichkeit auch solche Programme in nativen Code kompilieren zu lassen. Es gilt somit das Gleiche, wie für nativ kompilierte Tabellen.

Haupteinsatzzweck von systemintern kompilierten gespeicherten Prozeduren ist die Implementierung von Geschäftslogik für SOTs, da es für diese ja keine Foreign Key und Check-Constraints geben kann.



Zum besseren Verständnis der Thematiken „Speicheroptimierte Tabellen“ und „Systeminterne Kompilierung“ empfiehlt es sich, den Artikel [dn205319] zu lesen.



- [DynLinLib]
- [KleKonSS2IMO Aid]
- [dn249342]
- [dn205319]

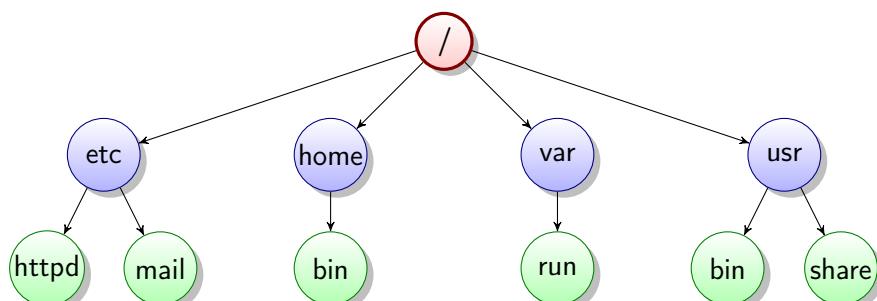
18.3 Bäume in der Informatik

Indizes sind optionale, mit einer Tabelle verbundene, Strukturen. Sie werden benutzt, um die Geschwindigkeit von Abfragen zu erhöhen. Dies geschieht, indem sie die Anzahl der Datenträgerzugriffe pro Abfrage verringern.

Es können beliebig viele Indizes zu einer Tabelle erstellt werden, solange sich die Kombinationen der Schlüsselspalten für die Indizes unterscheiden. Dabei kann eine Spalte in mehreren Kombinationen vorkommen und ein Index kann sich über mehrere Spalten erstrecken.

Indizes sind logisch und physisch unabhängig von den Daten in der Tabelle, an die sie angehängt sind. Deshalb benötigen Sie ihren eigenen Speicherplatz auf dem Datenträger. Ein Index kann erstellt und gelöscht werden, ohne das dadurch Tabellen oder andere Speicherstrukturen beeinflusst werden. Ihre Verwaltung geschieht automatisch durch die Datenbank, wenn DML-Operationen auf ihren Basistabellen stattfinden. Auch Anwendungen werden durch das Löschen eines Index nicht beeinträchtigt, nur der Zugriff auf die Nutzdaten kann langsamer werden.

Bäume stellen in der Informatik eine wichtige hierarchische Datenstruktur dar, die für unterschiedlichste Zwecke genutzt werden kann. Das bekannteste Beispiel für Bäume sind die „Verzeichnisbäume“ eines Dateisystems.



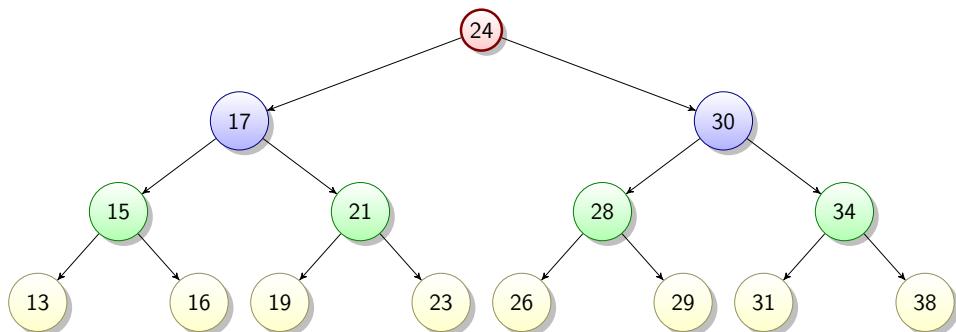
Die Abbildung zeigt den Unix-Verzeichnisbaum, anhand dessen einige Merkmale eines Baumes erkennbar sind.

- Jedes Element eines Baumes wird als „Knoten“ bezeichnet.
- Die Verbindungslien/Pfeile zwischen den Knoten sind „Kanten“.
- Ein Knoten, der keinen Vorgänger hat, wird als „Wurzel“ bezeichnet (im Diagramm rot).
- Alle Knoten ohne Nachfolger sind „Blätter“ bzw. „Leaves“ (im Diagramm grün)
- Knoten, die sowohl einen Vorgänger als auch einen Nachfolger besitzen sind „Zweige“ oder „Branches“ (im Diagramm blau).

18.3.1 Wichtige Eigenschaften von Bäumen

Es gibt diverse Arten von Bäumen in der Informatik (z. B. Binärbaum, B-Baum, B^+ -Baum oder B^* -Baum, uvm.). Die einfachste Form ist der Binärbaum. Daher wird dieser hier dazu benutzt werden, um die wichtigsten Grundlagen für das Verständnis von Bäumen zu legen.

Der im Folgenden abgebildete Binärbaum dient als Grundlage für alle weiteren Erläuterungen.



Von Knoten, Kanten und Ebenen

Wichtige Eigenschaften eines Baumes sind:

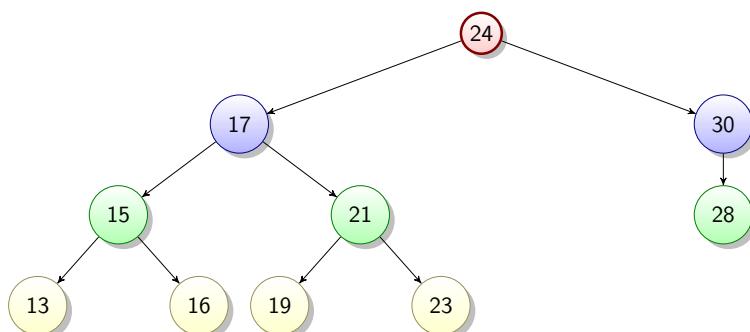
- **Tiefe:** Die Tiefe eines Knotens ist sein Abstand zur Wurzel (Anzahl der Kanten zwischen ihm und der Wurzel).
- **Ebene/Level:** Die Menge aller Knoten der gleichen Tiefe wird als Ebene bzw. Level bezeichnet.

- **Baumhöhe:** Die Höhe eines Baumes wird durch die maximale Tiefe, die ein Knoten erreichen kann, bestimmt.
- **Vollständigkeit:** Ein Baum gilt als vollständig, wenn alle Ebenen, bis auf die Unterste, komplett gefüllt sind. Die unterste Ebene muss von links nach rechts gefüllt sein.

Daraus ergibt sich, für den Binärbaum, folgendes:

- Die blauen Knoten bilden eine Ebene mit der Tiefe 1.
- Die grünen Knoten bilden eine Ebene mit der Tiefe 2.
- Die gelben Knoten bilden eine Ebene mit der Tiefe 3.
- Die Höhe des Baumes ist mit dem Wert 3 anzugeben. Dabei werden alle Ebenen, außer der Wurzel gezählt.
- Der Baum ist vollständig, da die blaue Ebene mit zwei Knoten und die grüne Ebene mit vier Knoten voll besetzt ist.
- Bäume wachsen in der Informatik von oben nach unten.

Die folgende Abbildung zeigt einen unvollständigen Binärbaum.



In der zweiten Ebene (grün) fehlt ein Element. Sie ist nicht mehr voll besetzt, was bedeutet, dass der Baum unvollständig ist.

Elemente und Höhe des Binärbaumes

Um die Anzahl der Elemente eines beliebigen vollständigen Binärbaumes zu berechnen, müssen verschiedene Informationen bekannt sein:

- In einem Binärbaum hat jeder Knoten höchstens zwei Nachfolger.
- Die Höhe des Baumes ist für die Berechnung notwendig.
- Die Ebenen werden nummeriert, beginnend mit dem Wert 0 bei der Wurzel.

Mit Hilfe dieser Informationen kann nun die maximale Anzahl der Elemente des Baumes (hier mit dem Buchstaben n bezeichnet) berechnet werden:

$$\begin{aligned} n &= 2^0 + 2^1 + 2^2 + 2^3 \\ n &= 15 \end{aligned}$$

Der Wert 2 in dieser Formel röhrt daher, dass jeder Knoten höchstens zwei Nachfolger haben kann. Die Exponenten 0, 1, 2 und 3 sind die Nummern der Ebenen. Berechnet man die einzelnen Zweierpotenzen ergibt sich:

$$n = 1 + 2 + 4 + 8$$

Die Wurzelebene hat genau ein Element. Die zweite Ebene hat höchstens zwei, die dritte Ebene höchstens 4 und die Blatt ebene höchstens 8 Elemente. Diese Formel lässt sich auf folgende Schreibweise verkürzen: $n = 2^{h+1} - 1$. Denn es gilt:

$$2^{h+1} - 1 = (2^0 + 2^1 + 2^2 + \dots + 2^h) - 1$$

Der Buchstabe h steht für die Höhe des Baumes.

Dieses Rechenbeispiel zeigt, dass die Höhe des Baumes und die Anzahl der Elemente in einer direkten Beziehung zu einander stehen. Daraus folgt, wenn die Anzahl der Elemente des Binärbaumes bekannt

ist, kann die Höhe des Baumes errechnet werden. Dies geschieht durch Umstellen der Formel: $n = 2^{h+1} - 1$.

$$2^{h+1} - 1 = n \quad | + 1 \quad (18.1)$$

$$2^{h+1} = n + 1 \quad | \log_2 \quad (18.2)$$

$$h + 1 = \log_2(n + 1) \quad | - 1 \quad (18.3)$$

$$h = \log_2(n + 1) - 1 = \log_2(n) \text{ abgerundet} \quad (18.4)$$

Das $\log_2(n + 1) - 1 = \log_2(n) \text{ abgerundet}$ gilt lässt sich beweisen:

$$n = 15 \quad (18.5)$$

$$\log_2(15 + 1) - 1 = 3 \quad (18.6)$$

$$\log_2(15) = 3.91(\text{Abrunden!}) \quad (18.7)$$



Mit Hilfe der Formel $\log_2(n)$ kann die Höhe eines Binärbaumes berechnet werden. Dabei ist zu beachten, dass das Ergebnis immer auf die ganze Zahl **abgerundet** werden muss.

Zur Wiederholung: Die Höhe eines Baumes wird durch die maximale Tiefe, die ein Knoten erreichen kann, bestimmt. Die Tiefe eines Knotens ist sein Abstand von der Wurzel (Anzahl der Kanten zwischen ihm und der Wurzel).

Die Zahl 3 aus dem vorangegangenen Beispiel gibt somit die Anzahl der Kanten zwischen dem Wurzelknoten und einem Blattknoten an. Damit ist aber noch nicht die Frage geklärt: „Wie viele Knoten müssen maximal gelesen werden, um ein bestimmtes Objekt aufzufinden?“. Die Antwort lautet: Die maximale Anzahl der Lesezugriffe ist die Höhe des Baumes + 1.

Dies kann am Beispiel des Knotens mit der Nummer „19“ nachgewiesen werden. Um diesen Knoten aufzufinden, müssen die Knoten „24“, „17“ und „21“ gelesen werden. Anschließend noch die „19“ selbst. Dies sind vier Zugriffe. Die Höhe des Baumes wird mit $h = \log_2(n) = 3.91$ berechnet, was abgerundet 3 ergibt. Für die Anzahl der Lesezugriffe muss die Zahl 3.91 aufgerundet werden, was den Wert 4 ergibt.

Bäume als Hilfsmittel zur Suche von Datensätzen

Um demonstrieren zu können, welch wichtige Rolle Bäume beim Auffinden von Datensätzen spielen, wird zu aller erst eine Tabelle benötigt. Dies soll hier die Tabelle KUNDE, aus dem Schema BANK, sein. Sie umfasst insgesamt 561 Zeilen.

Um nun einen bestimmten Datensatz zu finden, gibt es zwei unterschiedliche Verfahren:

- **Full Table Scan:** Das zeilenweise Durchsuchen der Tabelle, bis zum Auffinden des richtigen Datensatzes wird als Full Table Scan bezeichnet.
- **Index Scan:** Bei einem Index Scan wird der Suchbaum/Index nach dem gewünschten Datensatz durchsucht.

Diese beiden Methoden unterscheiden sich wesentlich in der Anzahl der Lesevorgänge, die zum Auffinden eines Datensatzes benötigt werden. Beim Full Table Scan gilt die Formel: $x = \frac{n}{2}$, wobei x die Anzahl der Lesevorgänge und n die Anzahl der Datensätze darstellt. Für die Tabelle KUNDE bedeutet dies konkret: $x = \frac{561}{2} \rightarrow x \approx 280$. Es werden also ca. 280 Lesevorgänge benötigt, um in der Tabelle KUNDE einen ganz bestimmten Datensatz zu finden.

Anders sieht dies bei der Nutzung eines Suchbaumes aus. Wie zuvor beschrieben, wird in einem Binärbaum die Anzahl der Lesevorgänge mit der Formel: $\log_2(n)$ angegeben, wobei n die Anzahl der Datensätze darstellt. Bezogen auf die Tabelle KUNDE bedeutet dies: $x = \log_2(561) \rightarrow \log_2(561) = 9.13 \rightarrow x \approx 10$. Mit Hilfe eines Suchbaumes würden also nur durchschnittlich 10 Zugriffe, statt der 280 benötigt. Dies stellt eine Kostenreduzierung von ca. 96 % dar.



In modernen Datenbanken kommen keine Binärbäume, sondern B*-Bäume zum Einsatz! Der Binärbaum wurde hier nur als einfaches Beispiel gewählt.

18.3.2 B*-Baum Indizes

B*-Bäume (gesprochen B-Stern Baum) sind eine vielfach weiterentwickelte Variante der Binärbäume. Urheber dieser Baumart ist Donald Ervin Knuth⁴. Einer der wesentlichsten Unterschiede zu den Binärbäumen ist, dass in einem B*-Baum ein Knoten beliebig viele Nachfolger haben kann, nicht mehr nur Zwei. Das bedeutet, dass die Kosten für die Suche eines Datensatzes noch weiter reduziert werden, da der Baum sehr viel breiter und damit flacher wird.

⁴Donald Ervin Knuth: US-Amerikanischer Informatiker und Professor an der Stanford University. Autor der Buchserie: „The Art of Computer Programming“ und Erfinder des Textsatzsystems „TeX“

B*-Baum Indizes sind die häufigste Index-Variante in einer Datenbank. Sie bestehen aus einer Reihe von Seiten (Daten- und Indexseiten) und enthalten die indizierten Werte in sortierter Reihenfolge. Jeder Wert muss dabei ein Schlüssel zugeordnet werden, mit dessen Hilfe der Wert gefunden werden kann.

Wird beispielsweise die Spalte NAME der Tabelle BANK indiziert, legt Microsoft SQL Server den Namen der jeweiligen Bank als Schlüssel und ein Merkmal zur Identifizierung der betreffenden Tabellenzeile im Index ab. Mit besagtem Merkmal ist es möglich, eine Tabellenzeile direkt, ohne Umwege, aufzufinden und zu lesen. Abbildung ?? zeigt beispielhaft, wie ein B*-Baum Index aussehen könnte.

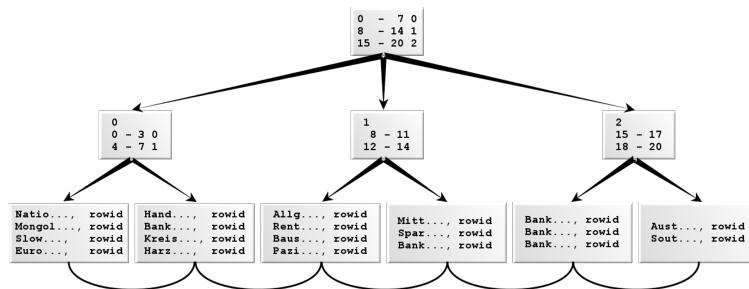


Abb. 18.5:
Ein B*-Baum
Index

Der Rootnode und die Branchnodes enthalten die Schlüsselwerte, in der Form: von - bis und die Adresse des dazugehörigen Indexblocks. Zum Beispiel bedeutet 0 - 7 0, dass die Schlüsselwerte null bis sieben im Branchnode Nummer null zu finden sind. Die erste Zeile in den Branchnodes ist die Indexblocknummer (0, 1 und 2).

In den Leafnodes stehen dann die indizierten Schlüsselwerte und die Zeilenidentifikationsmerkmale. Alle Leafnodes sind untereinander verbunden, so dass ein direkter Wechsel von einem Leafnode in den nächsten stattfinden kann, ohne dabei über den zuständigen Branchnode gehen zu müssen.

18.4 Indexentwurf im SQL Server

Der Microsoft SQL Server kennt eine Reihe verschiedener Indextypen. Die meisten davon sind in der soeben vorgestellten Form, dem B*-Baum, implementiert. Die Gemeinsamkeit aller Indextypen ist, dass sie einem Objekt, meist einer Tabelle, fest zugeordnet sind.

Der richtige Indexentwurf stellt für den Administrator eine große Herausforderung dar, da diese Thematik sehr komplex ist und viele Dinge gut bedacht werden müssen. Jeder Index stellt eine Verbesserung der Leseperformance dar, er bringt aber auch Kosten mit sich, da er gepflegt werden muss. Eine Formel für den richtigen Indexentwurf gibt es nicht. Oftmals hilft nur das Experimentieren mit unterschiedlichen Entwürfen.

18.4.1 Allgemeine Hilfestellungen für den Indexentwurf

Wie bereits erwähnt, gibt es keinen roten Faden, der den Administrator durch den Arbeitsprozess des Indexentwurfs leiten könnten. Es gibt aber durchaus einige Überlegungen von Microsoft, die dem Admin als Hilfestellung dienen können.

Welche Merkmale hat die Datenbank?

Datenbanken können im Wesentlichen auf zwei unterschiedliche Arten genutzt werden:

- Als **Online Transaction Processing Database** (OLTP): Die Datenbank besteht aus einer vielzahl kleiner und Großer Tabellen. Die Anzahl der DML-Vorgänge in der Datenbank ist mindestens genauso hoch, wie die Anzahl der Lesevorgänge.
- Als **Decision Support System** (DSS): Die Datenbank besteht aus einigen wenigen extrem großen Tabellen und einer kleinen bis mittleren Anzahl kleiner Tabellen. Schreibvorgänge sind sehr sehr selten. Neue Daten werden ausschließlich mittels eines täglichen Importvorganges eingefügt. Die Anzahl der Lesevorgäng ist hoch und auch die Komplexität der Abfragen ist groß.

Indizes in einer OLTP-Datenbank

Bei einer OLTP-Datenbank besteht das Problem, dass vor der Indexerstellung immer eine Kostennutzen-abwägung getroffen werden muss, da bei jedem DML-Vorgang die betroffenen Indizes gepflegt werden müssen. Microsoft empfiehlt hier:

- Die Anzahl der Indizes sollte gering gehalten werden, so viele wie nötig, aber nur so wenige wie möglich.
- Sofern sich Indizes auf eine Kombination von Spalten beziehen, sollte diese so schmal wie möglich gehalten werden.
- Auf einer Tabelle mit sehr geringer Volatilität⁵ kann dem Optimizer eine bereite Auswahl an verschiedenen Indizes geboten werden, damit diese sich den Besten heraussuchen kann.
- Kleine Tabellen sollten unter Umständen überhaupt nicht indiziert werden, da es kostengünstiger sein kann die gesamte Tabelle zu lesen, anstatt einen passenden Index auszuwählen.

⁵Volatilität = Änderungshäufigkeit

- Indizes sollten für solche Spalten erstellt werden die häufig in Prädikaten (`WHERE`-Klauseln) vorkommen.
- Bedenken Sie die Reihenfolge, in der Sie die Spalten in den Index aufnehmen gut. Diese sollte sich immer nach der Häufigkeit der Nutzung einer Spalte richten.

Die letzte Aussage der vorangegangenen Aufzählung soll hier mit einem Beispiel erläutert werden. Ein Index wird über die Spalten PLZ und ORT der Tabelle MITARBEITER werden in der genannten Reihenfolge in einem Index erfasst.

Ein Benutzer führt eine Abfrage mit dem Prädikat: `WHERE plz = 90411 AND ort = 'N"urnberg'` durch. In diesem Fall kommt der Index zur bestmöglichen Anwendung, da die Spaltenreihenfolge in der `WHERE`-Klausel und die im Index die gleiche ist.

Der gleiche Benutzer führt nun erneut eine Abfrage aus und ändert dabei das Prädikat folgendermaßen: `WHERE ort = 'N"urnberg' AND plz = 90411` durch. Auch hier kann der Index genutzt werden, da jedoch die Spaltenreihenfolge in der `WHERE`-Klausel eine andere ist, als im Index ist die Performance nicht so hoch, wie im ersten Beispiel.

Ein weiteres Beispiel: Der Nutzer fragt nur nach der Postleitzahl (`WHERE plz = 90411`). Wiederum kann der Index zur Anwendung gebracht werden. Die Leseperformance ist genauso hoch, wie im ersten Fall, allerdings sollte der Admin überlegen, ob er den Index nicht auf die Spalte PLZ begrenzt, wenn die Benutzer nur sehr selten nach dem Ort fragen.

Im vierten und letzten Beispiel gibt der Nutzer folgendes Prädikat an: `WHERE ort = 'N"urnberg'`. In diesem Fall kann der Index nicht zum Einsatz gebracht werden, da die Sortierung/Gruppierung im Index die Spalte ORT von der Spalte PLZ abhängig macht.

Indizes in einer DSS-Datenbank

Grundsätzlich gelten in einer DSS-Datenbank die gleichen Regeln für den Indexentwurf, wie in einer OLTP-Datenbank. Folgendes sollte aber trotzdem beachtet werden:

- Da Abfragen in einem DSS meist sehr komplex und zeitaufwendig sind, sollte gerade hier darauf wertgelegt werden, dass die Spaltenreihenfolge und die Spaltenauswahl im Index, der Spaltenreihenfolge und der Spaltenauswahl der meisten Prädikate entsprechen. So kann eine bestmögliche Ausnutzung der Indizes gewährleistet werden.

- Für DSS-Datenbanken kennt der SQL Server seit der Version 2012 einen neuen Indextyp, den Columnstore-Index. Dieser bietet wesentliche Performancesteigerungen im DSS-Bereich, ist jedoch in einer OLTP-Datenbank völlig fehl an Platz.

18.4.2 Die Sortierreihenfolge von Indizes festlegen

Beim Anlegen eines Indizes kann eine Sortierreihenfolge für die Schlüsselspalten angegeben werden. Sofern die Zeilen einer Tabelle häufig in sortierter Reihenfolge ausgegeben werden, ist es ratsam, einen sortierten Index auf die Spalten zu legen, nach denen in den Abfragen sortiert wird. Findet eine Abfrage einen vorsortierten Index, müssen die Ergebnissezeilen nicht erneut sortiert werden.

Ein Beispiel hierzu:

Auf den Spalten VORNAME und NACHNAME der Tabelle MITARBEITER liegt ein Index mit der Sortierung nachname DESC, vorname ASC. Ein Benutzer gibt in seiner Abfrage die ORDER BY-Klausel ORDER BY Nachname DESC, vorname ASC an. Da Vor- und Nachnamen im Index schon vorsortiert sind, kann bei der Ausführung der Abfrage die ORDER BY-Klausel vernachlässigt werden.

In einer weiteren Abfrage ändert der Nutzer seine ORDER BY-Klausel dahingehend, dass er die Sortierung der beiden Spalten vertauscht: ORDER BY nachname ASC, vorname DESC. Hier zeigt sich nun die Flexibilität des SQL Servers, denn dieser kann auch hier den Index benutzen, obwohl die ORDER BY-Klausel eine andere Sortierung vorgibt, als der Index.

- [jj835095]



18.5 Die verschiedenen Indextypen und deren Merkmale

18.5.1 Gruppierte Indizes - Clustered Indexes

Gruppierte Indizes sind eine Mischung aus einem Index und einer Tabelle. Sie stellen ein B-Baumstruktur für die effiziente Datensatzsuche bereit und gleichzeitig speichern sie selbst die Daten (Der Index ist die Tabelle - die Tabelle ist der Index). Nachdem bei dieser Indexstruktur die Daten direkt im Index gespeichert sind, entfällt der zusätzlich Zugriff auf eine Heap-Struktur, was wiederum für ein hohes Maß an Leseperformance sorgt.

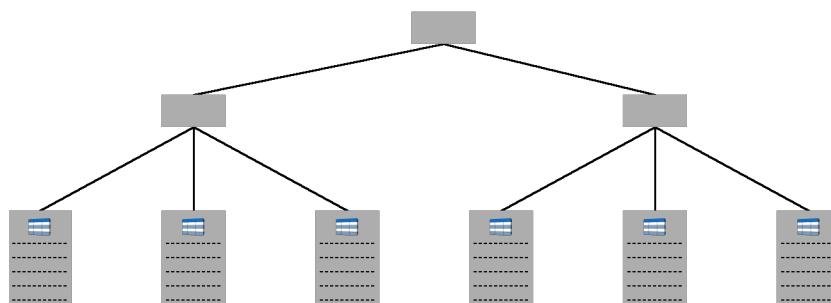


Abb. 18.6:
Schematische
Darstellung eines
clustered Index

Eine wichtige Eigenschaft des gruppierten Indexes ist es, dass er die Daten in sortierter Form speichert. Die Sortierung erfolgt mit Hilfe der Schlüsselwerte, wobei der Administrator entscheiden kann, ob er eine auf- oder absteigende Sortierung haben möchte. Dadurch, dass die Daten im Index sortiert sind, kann eine Abfrage, die nach einem Wertebereich fragt (z. B. WHERE bankfiliale_ID BETWEEN 10 AND 15

oder `WHERE geburtsdatum >= '01.01.1993')` sehr performant ausgeführt werden, weil die benötigten Zeilen physikalisch nebeneinander liegen.

Des Weiteren bringt die Vorsortierung einen Performance gewinn, da eine Abfrage mit einer `ORDER BY`-Klausel die Daten schon in passender Form vorfindet (siehe ??).

Ein gruppierter Index kann auf zwei Arten erstellt werden:

- mit Hilfe des `CREATE INDEX`-Statements
- oder durch das Anlegen eines `PRIMARY KEY`-Constraints.

Listing 18.7: Erstellen eines gruppierten Indizes

```
USE [demo_grafisch]
GO

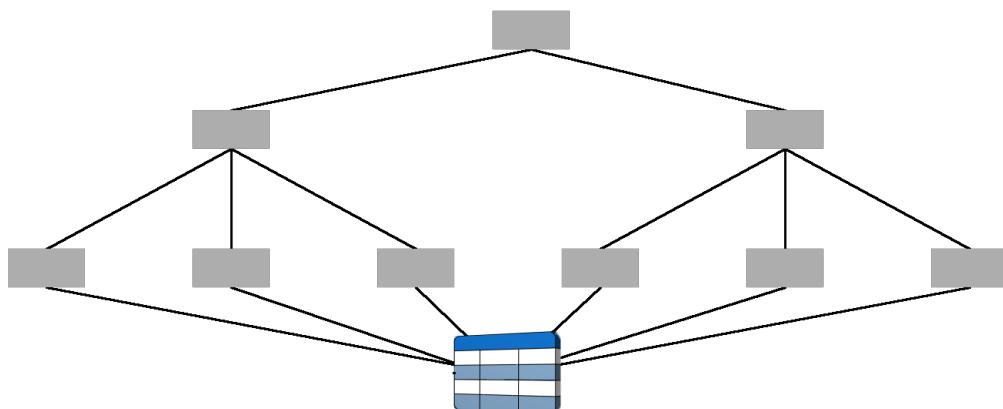
CREATE CLUSTERED INDEX idx_cl_aktie
ON demo_grafisch.dbo.aktie(aktie_ID DESC)
GO
```

- [ms186342]

18.5.2 Nicht gruppierter Index - Nonclustered Indexes

Ein nicht gruppierter Index ist eine eigenständige Struktur, die zu einem bestimmten Objekt, Tabelle oder View, gehört. Er enthält in seiner Leaf-Ebene Indexschlüsselwerte (z. B. Nachnamen, Postleitzahlen, Personalnummern) und Zeilenlokatoren (RID oder Schlüssel eines gruppierten Indizes). Die Indexschlüsselwerte dienen als Suchwerte und die RIDs zum Auffinden der betroffenen Zeilen.

Abb. 18.7:
Schematische
Darstellung eines
Nonclustered
Index



Ideal ist diese Art von Index für Abfragen, die nach einem ganz bestimmten Wert oder nach Werten aus einem Bereich suchen, sofern das Prädikat der Abfrage eine genaue Übereinstimmung verlangt. Beispielsweise sind Klauseln wie `WHERE plz = 90411` beziehungsweise `WHERE plz IN ('90411', '90762', '83262')` Prädikate mit genauer Übereinstimmung. Im Gegensatz dazu verlangt das Prädikat `WHERE nachname LIKE 'WEI%'` keine genaue Übereinstimmung.

- [ms189605]



Genau wie gruppierte Indizes werden die nicht gruppierten Indizes ebenfalls mit dem `CREATE INDEX`-Statement erstellt, allerdings kommt hier das Schlüsselwort `NONCLUSTERED` hinzu.

Listing 18.8: Erstellen eines nicht gruppierten Index

```
USE [demo_grafisch]
GO

CREATE NONCLUSTERED INDEX idx_isin
ON demo_grafisch.dbo.aktie(isin DESC)
GO
```

Beispiel ?? zeigt die Erstellung eines nicht gruppierten Indizes auf die Spalte ISIN, der Tabelle AKTIE. Die Sortierung der Spalte ISIN wird in diesem Beispiel explizit mit angegeben, da eine absteigende Sortierung gewünscht ist.



Es können maximal 999 verschiedene Indizes auf einer Tabelle erstellt werden.

Soll sich ein Index über mehrere Spalten erstrecken, müssen diese als Kommaseparierte Liste in den runden Klammern angegeben werden.

Listing 18.9: Erstellen eines nicht gruppierten Index mit mehreren Spalten

```
USE [demo_grafisch]
GO

CREATE NONCLUSTERED INDEX idx_aktie
ON demo_grafisch.dbo.aktie(isin DESC, name ASC)
GO
```



Die Anzahl der Schlüsselspalten in einem Index ist auf 16 begrenzt. Weiterhin darf der Indexschlüssel die maximale Breite von 900 Byte nicht überschreiten.



- [ms189280]

Datenspalten in einen Index einschließen

SQL Server bietet die Möglichkeit an, in einen nicht gruppierten Index, zusätzlich zu den Schlüsselspalten, auch Nichtschlüsselspalten aufzunehmen. Diese zusätzlichen Spalten werden zwar bei der Indexdurchsuchung nicht berücksichtigt, aber sie können die Performance einer Abfrage deutlich erhöhen, da die Ergebnisse direkt aus dem Index ausgegeben werden. Auf diese Weise unterbleiben weitere Zugriffe auf Tabellen oder gruppierte Indizes. Im Idealfall enthält ein nicht gruppierter Index alle Spalten als Nichtschlüsselspalten, die von einer Abfrage benötigt werden. Dadurch wäre die größt mögliche Performance gegeben.



Enthält ein Index alle Spalten, die von einer Abfrage benötigt werden, in Form von Schlüssel- oder Nichtschlüsselspalten, wird als „abdeckend“ bezeichnet.

Mit Hilfe von Nichtschlüsselspalten in einem Index ist es möglich, die Größenbeschränkung für Indexschlüssel, die bei 900 Byte liegt, zu umgehen, bzw. die Breite der Indexschlüssel möglichst klein zu halten. Hierzu ein Beispiel:

Es wird ein Index auf die drei folgenden Spalten gelegt:

- VORNAME VARCHAR(30)
- NACHNAME VARCHAR(35)
- SOZVERSNR VARCHAR(20)

Bei Spalten des Typs VARCHAR wird 1 Byte pro Zeichen benötigt, d. h. die Schlüsselbreite des Indexes beträgt 85 Byte. Die Abfragen, welche diesen Index benutzen lauten meist so ähnlich, wie das folgende Beispiel:

Listing 18.10: Beispielabfrage

```
USE [bank]
GO

SELECT vorname, nachname
FROM Mitarbeiter
WHERE SozVersNr = 'D370941F-6CD-6C07977'
```

Beispiel ?? zeigt, dass eigentlich nur die Spalte SOZVERSNR zur Suche genutzt wird. Daraus folgt, dass auch nur die Spalte SOZVERSNR als Schlüsselspalte in den Index aufgenommen werden müsste. Die Spalten VORNAME und NACHNAME könnten stattdessen als Nichtschlüsselspalten in den Index aufgenommen werden, wodurch sich die Schlüsselbreite verringern und der Index effizienter werden würde. Auch würde dadurch dessen Pflege einfacher und zusätzlich könnten die Werte der Spalten VORNAME und NACHNAME trotzdem direkt aus dem Index gelesen werden.

Nichtschlüsselspalten werden mittels der **INCLUDE**-Klausel in einen Index aufgenommen. Beispiel ?? zeigt, wie der in Beispiel ?? thematisierte Index, jetzt mit VORNAME und NACHNAME als Nichtschlüsselspalten erstellt wird.

Listing 18.11: Ein Index mit Nichtschlüsselspalten

```
USE [bank]
GO

CREATE NONCLUSTERED INDEX idx_sozversnr
ON bank.dbo.mitarbeiter(sozversnr ASC)
INCLUDE (vorname, nachname)
GO
```



Ein Index kann maximal 1024 Spalten haben. Davon muss eine Spalte als Schlüsselspalte genutzt werden und bis zu 1023 können als Nichtschlüsselspalten genutzt werden.

Bei der Nutzung der **INCLUDE**-Klausel müssen einige Regeln beachtet werden:

- Nichtschlüsselspalten können nur für nicht gruppierte Indizes angegeben werden
- Spalten mit einem der Datentypen TEXT, NTEXT und IMAGE können nicht als Nichtschlüsselspalten genutzt werden.
- Berechnete Spalten können unter bestimmten Bedingungen als Nichtschlüsselspalten genutzt werden.
- Spalten dürfen nicht als Schlüssel- und gleichzeitig als Nichtschlüsselspalten angegeben werden.
- Spaltennamen dürfen sich in der INCLUDE-Klausel nicht wiederholen.
- Soll eine Nichtschlüsselspalte aus ihrer Tabelle gelöscht werden, so müssen erst alle betroffenen Indizes gelöscht werden.

- Ist eine Tabellenspalte als Nichtschlüsselspalte in einem Index aufgenommen kann an ihr nur die `NULL`- bzw. `NOT NULL`-Eigenschaft geändert werden. Außerdem darf die Länge von Spalten der Typen `VARCHAR`, `NVARCHAR` und `VARBINARY` vergrößert werden.



- [ms190806]

18.5.3 Eigenschaften von Indizes

Eindeutigkeit

Die Aufgabe eines eindeutigen Indexes ist es, zu garantieren, dass in der indizierten Spalte keine redundanten Werte vorkommen. Die Nutzung eines solchen Indexes ist jedoch nur dann sinnvoll, wenn die Werte einer Spalte dafür geeignet sind. Häufig sind dies Spalten mit IDs oder anderen Nummern, während Spalten wie `VORNAME`, `NACHNAME` oder `GEBURTSdatum` fast immer redundante Werte enthalten und somit nicht geeignet sind.



Das Attribut der Eindeutigkeit kann sich sowohl auf gruppierte, als auch auf nicht gruppierte Indizes beziehen, wobei gruppierte Indizes immer als eindeutige Indizes angelegt werden. Dies gelingt durch einen einfachen Trick auch dann, wenn die Werte der indizierten Spalte Redundanzen enthalten.

Wird ein eindeutiger Index auf eine Gruppe von Spalten, z. B. `VORNAME`, `NACHNAME` und `GEBURTSdatum` gelegt, so erzwingt er die Eindeutigkeit der Kombinationen dieser Werte. Es dürfen dann niemals zwei Zeilen über die gleiche Wertekombination verfügen.

Um einen eindeutigen Index zu erstellen gibt es zwei Wege:

- Erstellen eines `PRIMARY KEY`- oder `UNIQUE`-Constraints
- Ausführen einer `CREATE UNIQUE INDEX`-Anweisung.

Da ein eindeutiger Index fast immer zur Gewährleistung der Eindeutigkeit von Werten eingesetzt wird, empfiehlt Microsoft, den Index nicht manuell, sondern mit einem `UNIQUE`- oder `PRIMARY KEY`-Constraint zusammen anzulegen. Denn beide Constraints erstellen automatisch einen eindeutigen Index im Hintergrund. Bei einem `PRIMARY KEY`-Constraint ist zusätzlich zu beachten, dass automatisch ein gruppierter Index erstellt wird, sofern der Administrator dies nicht explizit anders wünscht.



Auch eindeutige Indizes dürfen zusätzliche Nichtschlüsselpalten enthalten.

- [ms187019]



Filterung

Ein nicht gruppierter Index kann für bestimmte Zwecke mit einem Filter ausgestattet werden. Das bedeutet, dass er nicht mehr alle Werte seiner indizierten Spalten enthält, sondern nur noch solche, die einem Filterprädikat genügen. Somit eignen sich solche optimierten Indizes besonder für Abfragen, die meist mit einer fest definierten Teilmenge der Daten einer Tabelle arbeiten.

Durch die Anwendung eines Filters auf einem Index können folgende Vorteile erreicht werden:

- Verbesserung der Abfrageperformance,
- reduzierter Pflegeaufwand für den Index,
- reduzierter Speicherplatzbedarf

Die Abfrageperformance kann sich bei der Nutzung eines gefilterten Indizes deutlich verbessern, da der Index weniger Daten enthält und somit kleiner und auch genauer ist. Der Pflegeaufwand für den Index reduziert sich, weil ein Index nur dann gewartet werden muss, wenn sich einer der Werte ändert, der im Index aufgenommen wurde. Da in einem gefilterten Index nur noch relevante Werte enthalten sind, ist die Wahrscheinlichkeit, dass sich einer davon ändert geringer.

Besonders nützlich sind gefilterte Indizes beispielsweise dann, wenn eine Spalte sehr viele NULL-Werte enthält oder wenn einige fest Datenkategorien in ihr vorkommen.

Die Spalte STAATSANGEHOERIGKEIT der Tabelle EIGENKUNDE enthält nur die Werte „Belgisch“, „Dänisch“, „Deutsch“ und TÜRKISCH. Wenn sich eine Abfrage häufig nur auf deutsche Staatsbürger bezieht könnte hier ein gefilterter Index Vorteile bringen.

Um einen gefilterten Index anzulegen, der sich nur auf deutsche Staatsbürger bezieht, wird dem `CREATE INDEX` Kommando eine `WHERE`-Klausel angehängt.

Listing 18.12: Einen gefilterten Index erstellen

```
USE [bank]
```

```
GO
```

```
CREATE NONCLUSTERED INDEX idx_statsangehoerigkeit
ON bank.dbo.eigenkunde(statsangehoerigkeit)
WHERE statsangehoerigkeit = 'Deutsch'
GO
```



- [cc280372]

18.5.4 Der Columnstore-Index

Der mit dem SQL Server 2012 eingeführte Columnstore-Index speichert die Daten in einer völlig anderen Art und Weise als die herkömmlichen B*-Baum-Indizes und ist speziell für DSS-System entwickelt worden. Er bietet im DSS-Umfeld eine bis zu zehnfache Abfrageleistung und eine bis zu siebenfache höhere Kompressionsrate an, als dies bei normalen Indizes möglich ist. Besonders wichtig für seinen Einsatz ist, dass die Daten hauptsächlich durch Dataloads (Massenladenvorgänge) geschrieben werden.



- [gg492088]

18.6 Administrative Tätigkeiten an Indizes

18.6.1 Indizes reorganisieren

Indizes müssen bei jeder DML-Aktion von der Database Engine gepflegt werden, d. h. dem Index werden neue Datensätze hinzugefügt oder es werden bestehende gelöscht bzw. geändert. Durch diese Tätigkeiten bilden sich mit der Zeit „Lücken“ im Index. Die Informationen im Index liegen nicht mehr in der korrekten logischen Reihenfolge vor, sondern sie sind verstreut. Die Folge davon ist, dass die logische Reihenfolge der Indexeinträge nicht mehr zur physikalischen Reihenfolge der indizierten Datensätze passt. Beim Lesen der Datensätze kann kein sequenzieller Zugriff mehr erfolgen, es muss „hin und her gesprungen“ werden, um die Datensätze zu lesen. Die beiden folgenden Abbildungen sollen den Unterschied zwischen einem fragmentierten und einem nicht fragmentierten Index verdeutlichen. In Abbildung ?? ist die Reihenfolge der Indexeinträge 1., 2. und 3. (hier rot, grün und blau dargestellt) die gleiche, wie die der Indizierten Zeilen in der abhängigen Tabelle. Sollen mit einer Abfrage beispielsweise die Zeilen 2 und 3 gelesen werden, kann dies mit einem einfachen sequenziellen Zugriff erfolgen.

Abb. 18.8:
Ein nicht
fragmentierter
Index

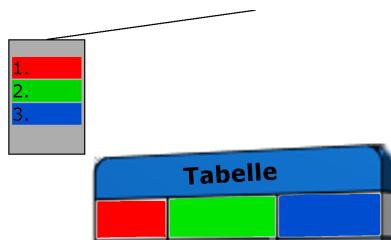
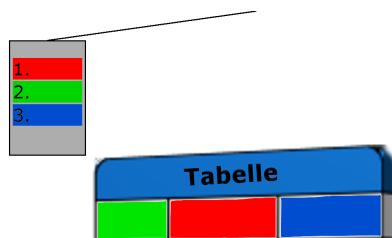


Abbildung ?? zeigt eine fragmentierten Index. Die Reihenfolge der Indexeinträge und die Reihenfolge der indizierten Zeilen in der Tabelle stimmen nicht mehr überein. Das Lesen der Datensätze wird dadurch aufwendiger.

Die Fragmentierung eines Indexes kann durch eine Neuerstellung oder eine Neuorganisation behoben werden. Im Folgenden werden diese beiden unterschiedlichen Methoden näher erläutert.

Abb. 18.9:
Ein fragmentierter
Index



Indexfragmentierung erkennen

Um den Grad der Fragmentierung eines Indexes feststellen zu können, bringt SQL Server die Systemfunktion `SYS.DM_DB_INDEX_PHYSICAL_STATS` mit. Diese liefert dem User drei Angaben:

- **avg_fragmentation_in_percent:** Prozentsatz der logischen Fragmentierung des Indizes (falsche Reihenfolge der Seiten im Index)
- **fragment_count:** Anzahl der Fragmente im Index (physisch aufeinanderfolgende Blattseiten)
- **avg_fragment_size_in_pages:** Durchschnittliche Anzahl der Seiten in einem Fragment im Index.

Für die Entscheidung, ob der Index Neuorganisiert oder Neuerstellt werden muss ist der Werte der Spalte `AVG_FRAGMENTATION_IN_PERCENT` entscheidend. Microsoft liefert folgenden Anhalt:

- $5\% < x \leq 30\%$: Der Index sollte Neuorganisiert werden
- $30\% < x$: Der Index sollte Neuerstellt werden.



Diese Richtwerte sind nur ein grober Anhalt, wann die eine und wann die andere Methode zur Anwendung kommen sollte. Der Administrator muss durch Experimentieren herausfinden, was für sein eigenes System adäquat ist.

Fragmentierung sehr kleiner Indizes

Wenn ein Index sehr klein ist (kleiner als 8 Indexseiten) kann es sein, dass der Administrator einer Fragmentierung nicht entgegenwirken kann, weil die Indexseiten in Mixed-Extents gespeichert werden. Da in einem Mixed-Extent Seiten von mehreren Objekten liegen, kann diese Form von Extent nicht einfach Neuorganisiert/Umsortiert werden.

Neuorganisieren

Beim Neuorganisieren eines Indexes werden die Seiten der Blattebene physisch neu angeordnet, so dass sie wieder in der richtigen Reihenfolge vorliegen. Diese Vorgehensweise ist dann interessant, wenn nur einige Teile des Indexes betroffen sind. Wenn beispielsweise in den ersten 10 % eines Indexes eine hohe Fragmentierung herscht, während die restlichen 90 % vollkommen in Ordnung sind, kann eine Neuorganisation gegenüber einer Neuerstellung von Vorteil sein. Die Neuorganisation betrachtet nur die fragmentierten 10 % des Indexes und lässt den Rest unangetastet, was sehr viel Zeit und Rechenleistung spart.

Listing 18.13: Den Fragmentierungsgrad eines Indizes feststellen

```
USE [bank]
GO

SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(N'Bank'), OBJECT_ID('dbo.konto'),
NULL, NULL, NULL) AS a
JOIN sys.indexes AS b
ON (a.object_id = b.object_id AND a.index_id = b.index_id)
GO
```



index_id	name	avg_fragmentation_in_percent
1	konto_pk	23,0769230769231

1 Zeile ausgewählt

Beispiel ?? zeigt eine logische Fragmentierung von 23 % für den gruppierten Index KONTO_PK an. Gemäß der Empfehlung von Microsoft ist hier eine Neuorganisation angebracht.

Listing 18.14: Den Index KONTO_PK reorganisieren

```
USE [bank]
GO

ALTER INDEX konto_pk
ON dbo.Konto
REORGANIZE;
GO
```

Nach erfolgreicher Neuorganisation des Indizes wird eine verbleibende Fragmentierung von 16,66 % für diesen Index angezeigt. Die Fragmentierung konnte also um ca. $\frac{1}{3}$ reduziert werden.



Die Neuorganisation eines Indexes erfolgt immer online, d. h. Benutzer können während der Reorganisation weiter auf den Index zugreifen. Indexoptionen können während der Neuorganisation nicht geändert werden.

Neuerstellen eines Indexes

Beim Neuerstellen eines Indexes wird der komplette Index neu gebaut. Dieser Prozess kann wahlweise online (nur in der Enterprise Edition) oder offline erfolgen.

Der Fragmentierungsgrade des Indizes EINGENKUNDEN_PK wird aktuell mit ca. 94 % angegeben. Laut Microsoft könnte hier mittels des Neuaufbaus des Indexes die Fragmentierung reduziert werden.

Listing 18.15: Den Index EIGENKUNDEN_PK neuerstellen

```
USE [bank]
GO

ALTER INDEX eigenkunden_pk
ON dbo.Eigenkunden
REBUILD
WITH (ONLINE = ON);
GO
```

Beispiel ?? zeigt wie der Index EIGENKUNDEN_PK online neuerstellt wird.



Es sei darauf hingewiesen, dass die Online-Neuerstellung eines Indexes ein vielfaches mehr an Ressourcen kostet, als die Offline-Variante. Zudem wird auch das Transaktionsprotokoll stärker belastet, wenn der Index online neuerstellt wird.



- [ms189858]
- [mssRSSiutOo]

18.6.2 Deaktivieren von Indizes

Durch das Deaktivieren eines Indexes werden alle lesenden und schreibenden Zugriffe auf ihn unterbunden. Bei einem gruppierten Index werden sogar alle Zugriff auf die zugrundeliegenden Tabellendaten unmöglich. Dies wird häufig vor Massenladevorgängen gemacht oder wenn ein Index konstant für schlechte Abfrageperformance sorgt.

Wird ein Index deaktiviert, so gilt grundsätzlich:

- Die Metadaten (Definition) eines deaktivierten Indexes, der auf eine Tabelle verweist, werden beibehalten, d. h. der Index geht nicht verloren.
- Der Index wird bei DML-Vorgängen nicht mehr gewartet.
- Er wird nicht mehr bei der Erstellung von Ausführungsplänen berücksichtigt.
- Es kann kein neuer Index erstellt werden, der den gleichen Namen benutzt, wie ein bereits vorhandener deaktiverter Index.
- Mit Hilfe von `ALTER INDEX ALL REBUILD` können alle deaktivierten Indizes auf einer Tabelle erneut erstellt und aktiviert werden.
- Ein deaktiverter Index kann nicht gelöscht werden.

Deaktivieren von gruppierten Indizes

Zusätzlich zu den allgemeinen Regeln gelten beim Deaktivieren eines gruppierten Indexes folgenden Regeln:

- Es werden alle `FOREIGN KEY`-Constraints deaktiviert, die sich auf den deaktivierten gruppierten Index beziehen.
- Wenn es sich bei dem gruppierten Index um einen eindeutigen Index handelt, werden auch alle `PRIMARY KEY`- und `UNIQUE`-Constraints deaktiviert, die sich auf den Index beziehen.
- Beim Deaktivieren eines gruppierten Indexes werden auch alle nicht gruppierten Indizes deaktiviert, die sich auf den gruppierten Index beziehen.
- `DBCC CHECKDB` kann keine Informationen zu einem deaktivierten gruppierten Index liefern.

Deaktivieren von nicht gruppierten Indizes

Zusätzlich zu den allgemeinen Regeln gelten beim Deaktivieren eines nicht gruppierten Indexes folgende Regeln:

- Ein deaktivierter nicht gruppierter Index kann online oder offline neuerstellt werden.

Indizes mit TSQL deaktivieren

Sowohl gruppierte als auch nicht gruppierte Indizes können mit Hilfe des `ALTER INDEX`-Kommando deaktiviert werden.

Listing 18.16: Den Index EIGENKUNDEN_PK deaktivieren

```
USE [bank]
GO

ALTER INDEX eigenkunden_pk
ON dbo.Eigenkunden
DISABLE;
GO
```

Zum Deaktivieren aller Indizes, die auf ein bestimmtes Objekt zeigen, existiert die Anweisung `ALTER INDEX ALL`.

Listing 18.17: Alle Indizes auf der Tabelle EIGENKUNDEN

```
USE [bank]
GO

ALTER INDEX ALL
ON dbo.Eigenkunden
DISABLE;
GO
```



- [ms177456]

18.6.3 Aktivieren von Indizes

Bevor ein deaktivierter Index wieder genutzt werden kann, muss er zuerst aktiviert werden. Dies gilt nicht nur für den betroffenen Index, sondern auch für alle von ihm abhängigen Indizes und evtl. auch Constraints.

Auswirkungen auf Constraints

- Nach dem Neuerstellen/Aktivieren eines Indexes müssen alle Constraints, manuell, durch das Kommando `ALTER TABLE CHECK CONSTRAINT`, aktiviert werden.
- `PRIMARY KEY`- und `UNIQUE`-Constraints werden durch die Neuerstellung eines Indizes, von dem sie abhängig sind, automatisch aktiviert.

Auswirkungen auf gruppierte Indizes

- Gruppierte Indizes können nicht online neuerstellt werden.

Auswirkungen auf nicht gruppierten Indizes

- Deaktivierte nicht gruppierte Indizes müssen explizit, mittels `ALTER INDEX`-Kommando reaktiviert werden, es sei denn ein gruppierter Index, von dem sie abhängig sind, wird mittels `ALTER INDEX ALL` neuerstellt.

Weiterhin treten die folgenden Auswirkungen auf einen deaktivierten nicht gruppierten Index auf:

Aktion auf gruppierten Index	Ergebnis für nicht gruppierten Index
<code>ALTER INDEX REBUILD</code>	Er bleibt deaktiviert
<code>ALTER INDEX REBUILD ALL</code>	Er wird neu erstellt und aktiviert
<code>DROP INDEX</code>	Er bleibt deaktiviert
<code>CREATE INDEX WITH DROP_EXISTING</code>	Er bleibt deaktiviert

Wird der gruppierte Index neu erstellt, treten die gleichen Auswirkungen auf, wie bei einem `ALTER INDEX ALL REBUILD`. Die folgende Tabelle zeigt auf, welche Aktionen für einen nicht gruppierten Index, der einem gruppierten Index zugeordnet ist, zulässig sind und welche Auswirkungen diese haben.

Aktion auf nicht gruppierten Index	Beide Indizes sind deaktiviert	Der gruppierte Index ist aktiviert
<code>ALTER INDEX REBUILD</code>	Fehler	Aktion erfolgreich
<code>DROP INDEX</code>	Aktion erfolgreich	Aktion erfolgreich
<code>CREATE INDEX WITH DROP_EXISTING</code>	Fehler	Aktion erfolgreich



- [ms190645]
- [ms177456]

18.6.4 Verschieben eines Indexes in eine andere Dateigruppe

Indizes können mittels der Anweisung `CREATE INDEX WITH DROP_EXISTING = ON` in eine andere Dateigruppe verschoben werden. Sollte es sich bei einem Index um einen gruppierten Index handeln, werden die Tabellendaten zusammen mit dem Index verschoben.

Das folgende Beispiel zeigt, wie der Index EIGENKUNDEN_PK in die Dateigruppe INDEXDATA verschoben wird.

Listing 18.18: Den Index EIGENKUNDEN_PK in eine andere Dateigruppe verschieben

```
USE [bank]
GO

CREATE UNIQUE CLUSTERED INDEX eigenkunden_pk
ON dbo.Eigenkunden (eigenkunden_id)
WITH (DROP_EXISTING = ON)
ON Indexdata;
GO
```



- [ms175905]

18.7 Übungen - Schemaobjekte verwalten

Benutzen Sie für die Durchführung der Übungen die Datenbank BANK_2014.

1. Erstellen Sie die Tabelle BANK, gemäß Ihrem ER-Modell, in der Dateigruppe KAM. Machen Sie das Attribut BANK_ID zum Primärschlüssel und achten sie darauf, dass die Tabelle als Heap erstellt wird.
 2. Ist es sinnvoll, die Tabelle BANK als Heap anzulegen? Begründen Sie Ihre Aussage!
-
-
-

3. Legen Sie ein **UNIQUE**-Constraint auf die Spalte BIC der Tabelle BANK. Die Tabelle muss weiterhin in Form eines Heaps gespeichert bleiben.
4. Fügen Sie Ihrer Datenbank die Dateigruppe IN_MEMORY hinzu und bereiten Sie alles vor, um eine SOT darin anzulegen.
5. Legen Sie die Tabelle BUCHUNG gemäß Ihrem ER-Modell als SOT an und fügen Sie zwischen die beiden Spalten BUCHUNGSDATUM und KONTO_ID die Spalte BUCHUNGSTEXT VARCHAR(200) ein! Legen Sie auf die Spalte BUCHUNGSDATUM einen Index. Legen Sie auch auf die Spalte BUCHUNGSTEXT einen Index.
6. Erstellen Sie die Tabelle EIGENKUNDE, gemäß Ihrem ER-Modell, in der Dateigruppe KAM und legen Sie ein **UNIQUE**-Constraint auf die Spalte PERSONALAUSWEISNR. Die Sortierung des Indexes, der zu diesem Constraint gehört soll absteigend sein.
7. Erstellen Sie die Tabelle MITARBEITER, gemäß Ihrem ER-Modell, in der Dateigruppe KAM und legen Sie ein **UNIQUE**-Constraint auf die Spalte SOZVERSNR. Schließen Sie die Spalten NACHNAME und VORNAME in den Index als Nichtschlüsselpalten ein. Hinweis: Das Anlegen des **UNIQUE**-Constraints muss über einen „Umweg“ geschehen!
8. Legen Sie einen nicht gruppierten Index auf die Tabelle EIGENKUNDE (Spalte ABLAUFDATUM). Der Index soll nur die Zeilen enthalten, die einen Personalausweis zeigen, der vor dem 01.01.2016

abgelaufen ist. Schließen Sie die Spalte PERSONALAUSWEISNR als Nichtschlüsselspalte in den Index ein.

9. Benutzen Sie für die Durchführung dieser Aufgabe die Datenbank „bank“!

Ermitteln Sie den Prozentsatz der logischen Indexfragmentierung der beiden Indizes MITARBEITER_PK und GIROKONTO_PK.

10. Benutzen Sie für die Durchführung dieser Aufgabe die Datenbank „bank“!

Ergreifen Sie für jeden der beiden Indizes die geeignete Maßnahme, um deren Fragmentierung zu reduzieren.

11. Erstellen Sie die Tabelle GIROKONTO gemäß Ihrem ER-Modell. Legen Sie den Primärschlüssel dieser Tabelle mit einem clustered Index an.
12. Erstellen Sie einen nicht gruppierten Index auf der Spalte GUTHABEN der Tabelle GIROKONTO.
13. Deaktivieren Sie den gruppierten Index der Tabelle GIROKONTO und beantworten Sie die folgenden Fragen:

Können die Daten der Tabelle GIROKONTO noch selektiert werden?

Welche Auswirkungen hatte das Deaktivieren des clustered Index auf den nicht gruppierten Index der Spalte GUTHABEN

14. Suchen Sie einen Weg, den gruppierten und den nicht gruppierten Index der Tabelle GIROKONTO in einem Zug zu reaktivieren.

mindestens drei sinnvolle Fragen!

15. Lesen Sie den Artikel [utilUDPaC] und notieren Sie sich mindestens drei sinnvolle Fragen!

18.8 Lösungen - Schemaobjekte verwalten

Benutzen Sie für die Durchführung der Übungen die Datenbank BANK_2014.

1. Erstellen Sie die Tabelle BANK, gemäß Ihrem ER-Modell, in der Dateigruppe KAM. Machen Sie das Attribut BANK_ID zum Primärschlüssel und achten sie darauf, dass die Tabelle als Heap erstellt wird.

```
USE [Bank_2014]
GO

CREATE TABLE bank (
    Bank_ID      NUMERIC NOT NULL,
    BIC          VARCHAR(11) NOT NULL,
    Name          VARCHAR(50) NOT NULL,
    Strasse       VARCHAR(50),
    Hausnummer   VARCHAR(20),
    PLZ           CHAR(5),
    Ort           VARCHAR(20),
    Rating        VARCHAR(3),
    CONSTRAINT bank_pk PRIMARY KEY NONCLUSTERED (Bank_ID)
)
ON KAM;
GO
```

2. Ist es sinnvoll, die Tabelle BANK als Heap anzulegen? Begründen Sie Ihre Aussage!

Die Nutzung einer Tabelle in Form eines Heaps, anstatt eines gruppierten Indizes, kann sinnvoll sein, wenn die Tabelle sehr klein ist und somit das Lesen der gesamten Tabelle kostengünstiger ist, als das Durchsuchen eines clustered Index.

In der Praxis kann es hierfür noch weitere Gründe geben, die allerdings den Rahmen dieser Unterrichtsunterlage deutlich sprengen würden.

3. Legen Sie ein **UNIQUE**-Constraint auf die Spalte BIC der Tabelle BANK. Die Tabelle muss weiterhin in Form eines Heaps gespeichert bleiben.

```
USE [Bank_2014]
GO

ALTER TABLE bank
ADD CONSTRAINT bic_uk UNIQUE NONCLUSTERED (BIC);
GO
```

4. Fügen Sie Ihrer Datenbank die Dateigruppe IN_MEMORY hinzu und bereiten Sie alles vor, um eine SOT darin anzulegen.

```
USE [Bank_2014]
GO

-- Memory optimized filegroup anlegen
ALTER DATABASE [Bank_2014]
ADD FILEGROUP [in_memory]
CONTAINS MEMORY_OPTIMIZED_DATA;
GO

-- Es muss eine Filestream Datendatei geben,
-- um Tabellen anlegen zu koennen.
ALTER DATABASE [Bank_2014]
ADD FILE (
    NAME = 'bank_2014_in_memory_01',
    FILENAME = 'D:\u01\bank_2014\filestream\in_memory'
)
TO FILEGROUP [in_memory]
GO
```

5. Legen Sie die Tabelle BUCHUNG gemäß Ihrem ER-Modell als SOT an und fügen Sie zwischen die beiden Spalten BUCHUNGSDATUM und KONTO_ID die Spalte BUCHUNGSTEXT VARCHAR(200) ein! Legen Sie auf die Spalte BUCHUNGSDATUM einen Index. Legen Sie auch auf die Spalte BUCHUNGSTEXT einen Index.

```
USE [Bank_2014]
GO

CREATE TABLE Buchung (
    Buchungs_ID      NUMERIC NOT NULL,
    Betrag           NUMERIC(12,2),
    Buchungsdatum    DATETIME2 NOT NULL,
    Buchungstext     VARCHAR(200) COLLATE LATIN1_GENERAL_BIN2 NOT NULL,
    Konto_ID         NUMERIC,
    Transaktions_ID  NUMERIC,
    CONSTRAINT Buchung_pk PRIMARY KEY NONCLUSTERED (Buchungs_ID),
    INDEX            idx_Buchungsdatum NONCLUSTERED (Buchungsdatum),
    INDEX            idx_Buchungstext NONCLUSTERED (Buchungstext)
)
WITH (
    MEMORY_OPTIMIZED = ON
)
GO
```

6. Erstellen Sie die Tabelle EIGENKUNDE, gemäß Ihrem ER-Modell, in der Dateigruppe KAM und legen Sie ein **UNIQUE**-Constraint auf die Spalte PERSONALAUSWEISNR. Die Sortierung des Indexes, der zu diesem Constraint gehört soll absteigend sein.

```
USE [Bank_2014]
GO

CREATE TABLE Eigenkunde (
    Kunden_ID        NUMERIC NOT NULL,
    Geburtsdatum     DATETIME2,
    Strasse          VARCHAR(50),
    Hausnummer       VARCHAR(15),
    PLZ              CHAR(5),
    Ort              VARCHAR(20),
    PersonalausweisNr VARCHAR(9),
    Ausstellungsdatum DATETIME2,
    Ablaufdatum      DATETIME2,
    Staatsangehoerigkeit VARCHAR(20),
    Geburtsort        VARCHAR(50),
    Telefon          VARCHAR(15),
    Emailadresse     VARCHAR(25),
    CONSTRAINT eigenkunde_pk PRIMARY KEY (Kunden_ID),
    CONSTRAINT personalausweisNr_uk UNIQUE (PersonalausweisNr DESC)
)
ON KAM;
```

GO

7. Erstellen Sie die Tabelle MITARBEITER, gemäß Ihrem ER-Modell, in der Dateigruppe KAM und legen Sie ein **UNIQUE**-Constraint auf die Spalte SOZVERSNR. Schließen Sie die Spalten NACHNAME und VORNAME in den Index als Nichtschlüsselspalten ein. Hinweis: Das Anlegen des **UNIQUE**-Constraints muss über einen „Umweg“ geschehen!

```
USE [Bank_2014]
GO

CREATE TABLE Mitarbeiter (
    Mitarbeiter_ID      NUMERIC NOT NULL,
    Vorname              VARCHAR(30),
    Nachname             VARCHAR(35),
    Vorgesetzter_ID      NUMERIC,
    Bankfiliale_ID       NUMERIC,
    Geburtsdatum         DATETIME2,
    SozVersNr            VARCHAR(20),
    Gehalt               NUMERIC(12,2),
    Strasse              VARCHAR(50),
    Hausnummer           VARCHAR(20),
    PLZ                  CHAR(5),
    Ort                  VARCHAR(20),
    Provision            NUMERIC,
    CONSTRAINT Mitarbeiter_pk PRIMARY KEY (Mitarbeiter_ID)
)
ON KAM;
GO

CREATE UNIQUE NONCLUSTERED INDEX idx_SozVersNr
ON dbo.Mitarbeiter (SozVersNr)
INCLUDE (Vorname, Nachname);
GO
```

8. Legen Sie einen nicht gruppierten Index auf die Tabelle EIGENKUNDE (Spalte ABLAUFDATUM). Der Index soll nur die Zeilen enthalten, die einen Personalausweis zeigen, der vor dem 01.01.2016 abgelaufen ist. Schließen Sie die Spalte PERSONALAUSWEISNR als Nichtschlüsselspalte in den Index ein.

```
USE [Bank_2014]
GO

CREATE NONCLUSTERED INDEX idx_ablaufdatum
ON Eigenkunde (Ablaufdatum)
INCLUDE (PersonalausweisNr)
WHERE Ablaufdatum < CONVERT(DATETIME2, '2016-01-01');
GO
```

9. Benutzen Sie für die Durchführung dieser Aufgabe die Datenbank „bank“!

Ermitteln Sie den Prozentsatz der logischen Indexfragmentierung der beiden Indizes MITARBEITER_PK und GIROKONTO_PK.

```
USE [Bank_2014]
GO

SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(N'Bank'),
OBJECT_ID('dbo.mitarbeiter'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b
ON (a.object_id = b.object_id AND a.index_id = b.index_id);
GO

/* 1 mitarbeiter_pk 80 -> REBUILD */

SELECT a.index_id, name, avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(DB_ID(N'Bank'),
OBJECT_ID('dbo.girokonto'), NULL, NULL, NULL) AS a
JOIN sys.indexes AS b
ON (a.object_id = b.object_id AND a.index_id = b.index_id);
GO

/* 1 girokonto_pk 25 -> REORGANIZE */
```

10. Benutzen Sie für die Durchführung dieser Aufgabe die Datenbank „bank“!

Ergreifen Sie für jeden der beiden Indizes die geeignete Maßnahme, um deren Fragmentierung zu reduzieren.

```
USE [Bank_2014]
GO

/* 1 mitarbeiter_pk 80 -> REBUILD */

ALTER INDEX mitarbeiter_pk
ON Mitarbeiter
REBUILD;

/* 1 girokonto_pk 25 -> REORGANIZE */

ALTER INDEX girokonto_pk
ON Girokonto
```

```
REORGANIZE;
```

11. Erstellen Sie die Tabelle GIROKONTO gemäß Ihrem ER-Modell. Legen Sie den Primärschlüssel dieser Tabelle mit einem clustered Index an.

```
USE [Bank_2014]
GO

CREATE TABLE Girokonto (
    Konto_ID      NUMERIC NOT NULL,
    Gebuehr       NUMERIC(12,2),
    Guthaben      NUMERIC(12,2),
    Habenzins     NUMERIC(5,2),
    Kreditrahmen  NUMERIC(12,2),
    CONSTRAINT girokonto_pk PRIMARY KEY CLUSTERED (Konto_ID)
)
ON KAM;
```

12. Erstellen Sie einen nicht gruppierten Index auf der Spalte GUTHABEN der Tabelle GIROKONTO.

```
USE [Bank_2014]
GO

CREATE NONCLUSTERED INDEX idx_guthaben
ON Girokonto (Guthaben);
GO
```

13. Deaktivieren Sie den gruppierten Index der Tabelle GIROKONTO und beantworten Sie die folgenden Fragen:

Können die Daten der Tabelle GIROKONTO noch selektiert werden?

Welche Auswirkungen hatte das Deaktivieren des clustered Index auf den nicht gruppierten Index der Spalte GUTHABEN

```
USE [Bank_2014]
GO

ALTER INDEX girokonto_pk
ON Girokonto
DISABLE;
```

14. Suchen Sie einen Weg, den gruppierten und den nicht gruppierten Index der Tabelle GIROKONTO in einem Zug zu reaktivieren.

```
USE [Bank_2014]
GO

ALTER INDEX ALL
ON Girokonto
REBUILD;
```

15. Lesen Sie den Artikel [utilUDPaC] und notieren Sie sich mindestens drei sinnvolle Fragen!

19 Sicherheit im SQL Server

Inhaltsangabe



Die in diesem Kapitel genannten Informationen beziehen sich nur auf das Absichern einer SQL Server-Instanz. Für andere Softwareprodukte können unter Umständen andere bzw. zusätzliche Einstellungen und Maßnahmen notwendig sein.

Das Absichern von Anwendungen und Diensten ist eine vielschichtige Aufgabe und bedarf genauer Planung, sowie guter Kenntnis der Materie. Grundsätzlich sind es vier Ebenen, die es abzusichern gilt:

- Der Serverraum
- Das Netzwerk
- Das Betriebssystem
- Die Anwendung / der Dienst

Abhängig davon, wie sensibel die zu schützenden Daten sind, muss jede dieser Ebene mit entsprechendem Aufwand gesichert werden.

19.1 Sicherheit auf Betriebssystemebene

Ein wesentlicher Bestandteil der Sicherheit auf Betriebssystemebene ist ein gut durchdachtes Rollen und Rechtekonzept. Jeder Benutzer sollte nur auf die Bestandteile des Betriebssystems Zugriff haben, auf die er auch Zugriff haben muss. Das beginnt damit, dass jeder Benutzer ein eigenes Nutzerkonto hat, das mit den notwendigen Berechtigungen ausgestattet ist.

Zu beachten ist hierbei, dass nicht nur „Benutzer aus Fleisch und Blut“ Nutzerkonten verwenden. Jeder Windowsdienst wird im Kontext eines Benutzerkontos ausgeführt, was bedeutet, dass der Dienst die Berechtigungen seines Nutzerkontos besitzt und einsetzen kann.



Benutzerkonten, die von Windowsdiensten verwendet werden, werden im Folgenden als „Dienstkonten“ bezeichnet.

Um unliebsame Zwischenfälle zu vermeiden sollte der Administrator das „Principle of least privilege“ (POLP) beachten. Dieses Prinzip besagt, dass jeder Benutzer nur die Rechte eingeräumt bekommt, die er für seine Arbeit zwingend braucht.

19.1.1 NTFS-Zugriffsrechte

Der Microsoft SQL Server speichert verschiedene Dateiarten in unterschiedlichen Verzeichnissen. Bei fast allen dieser Ordner muss nur der SQL Server-Dienst selbst Zugriff haben, der Datenbankadministrator benötigt dagegen nur in Ausnahmefällen Zugriff.

Die Standardverzeichnisse des SQL Server

In Kapitel ?? wurde bereits das Standardinstallationsverzeichnis des SQL Server angesprochen. Es befindet sich in C:\Program Files\Microsoft SQL Server. In diesem Ordner liegen die folgenden Bestandteile:

- die instanzabhängigen Komponenten der Database Engine, der Reporting Services und der Analysis Services
- die gemeinsam genutzten Komponenten,
- das Datenverzeichnis data,
- das Protokollverzeichnis data
- und das Sicherungsverzeichnis backup

Für all diese Verzeichnisse gilt: Normale Benutzer benötigen keinen Zugriff und auch Administratoren müssen dort nur in äußerst seltenen Fällen zugreifen. Nur die SQL Server-Dienste müssen Vollzugriff auf die dort befindlichen Dateien haben.



Anmerkung des Autors: Im Unterordner binne liegen die ausführbaren Dateien für das Management Studio und alle anderen Komponenten, die Sie während dieses Lehrgangs benötigen. Aus diesem Grund werden die Zugriffsrechte für dieses Verzeichnis nicht verändert. Im Normalfall hat der Administrator einen eigenen „Adminrechner“, auf dem die Client-Tools, samt Management Studio, installiert sind, weshalb auch er dort keinen Zugriff haben muss.

Erstellen einer sinnvollen Verzeichnisstruktur

Sofern der SQL Server mit Standardeinstellungen betrieben wird, werden neue Datenbanken im Datenverzeichnis `data`, unterhalb des Installationsordners angelegt. Da dieser Ordner üblicherweise auf dem Systemdatenträger `C:\` liegt, sollten dort keine Datenbanken liegen, weil sonst die Gefahr besteht, dass dieser Datenträger vollläuft und das gesamte System blockiert wird. Des Weiteren sollten, wie in ?? bereits erwähnt, bestimmte Datenbanken, wie z. B. die `TEMPDB`, auch aus Performancegründen auf andere Datenträger verteilt werden.

Die Aufgabe des Administrators ist es nun, sich eine sinnvolle Verzeichnisstruktur für die Verteilung zu überlegen. Besagte Struktur sollte dabei die folgenden Punkte berücksichtigen:

- Anwendungsdaten (Programme und Programmbibliotheken) und Datenbanken sollten getrennt von einander liegen.
- Es sollte möglich sein, ein „Inplace Upgrade“ der Datenbanksoftware durchzuführen. Dazu müssen kurzzeitig zwei Versionen des SQL Servers auf einem Server betrieben werden können.
- Datenbanken und Protokolldateien sollten auf getrennten Datenträgern liegen.
- Möglichst einfach durch NTFS-Rechte zu sichernde Verzeichnisstruktur.

Verschiedene Softwareanbieter haben bereits solche Lösungen erarbeitet und stellen diese der Allgemeinheit zur Verfügung. Ein Lösungsansatz, der all diese Punkte berücksichtigt, ist die „Oracle Flexible Architecture“ - kurz OFA. Auch wenn die OFA für die Aufteilung von Oracle-Datenbankdateien gedacht ist, ist sie dennoch so allgemein gehalten, dass sie auch auf andere DBMS, wie z. B. den SQL Server, anwendbar ist.

Die OFA-Struktur sieht folgendes vor:

- Ein Verzeichnisbaum beginnt mit einem kurzen, eindeutigen Namen, der von keinen äußeren Faktoren, wie z. B. einer Hardwarebezeichnung, abhängig ist. Außerdem sollte der Verzeichnisname eine laufende Nummer enthalten.
- Das erste Unterverzeichnis drückt etwas über seinen Inhalt aus, z. B. `app` für Applications oder `data` für Datenbankdateien.
- In der zweiten Verzeichnisebene kann die Versionsnummer des genutzten DBMS dargestellt werden, z. B. 120 für den SQL Server 2014.

- Die nächste Verzeichnisebene gibt den Namen der Datenbank an, z. B. Bank_2014

Beispiele für eine OFA-Kompatible Verzeichnisstruktur könnten somit sein:

D:\u01\data\120\Bank_2014

D:\u01\filestream\120\Bank_2014

D:\u01\log\120\Bank_2014

D:\u01\backup\120\Bank_2014

Festlegen der NTFS-Rechte mit Hilfe der OFA

Die einfachste Möglichkeit, die passenden NTFS-Berechtigungen innerhalb der OFA zu verteilen besteht darin, den Mechanismus der „Vererbung“ auszunutzen. Der Administrator muss lediglich die NTFS-Rechte an der obersten Verzeichnisebene, im vorangegangenen Beispiel war dies u01, ändern und diese werden weiter nach unten vererbt. Orientiert man sie dabei, wie von Microsoft empfohlen, am AGDLP-Schema, so bedeutet dies:

1. Erstellen einer globalen Gruppe im AD, z. B. mit dem Namen G-MSSQLSERVER120.
2. Aufnehmen der gMSAs aller SQL Server-Dienste in die globale Gruppe.
3. Erstellen einer domänenlokalen Gruppe im Active Directory, beispielsweise mit dem Namen L-MSSQLSERVER120.
4. Aufnehmen der globalen Gruppe in die domänenlokale Gruppe.
5. Deaktivieren der Vererbung für den Ordner D:\u01
6. Zuteilen des NTFS-Rechts „Vollzugriff“ an die domänenlokale Gruppe auf das Verzeichnis D:\u01.
7. Aufrufen der Erweiteren Sicherheitseinstellungen und ausführen der Funktion „Alle Berechtigungseinträge für untergeordnete Objekte durch vererbbare Berechtigungseinträge von diesem Objekt ersetzen“. Dadurch werden alle Unterordner mit den korrekten Berechtigungen versehen.
8. Kontrollieren, dass die Vererbung des NTFS-Rechte für alle Unterordner von D:\u01 aktiviert ist.

9. Zuteilen des NTFS-Rechts „Vollzugriff“ an die weiteren Ordner u02, u03, usw.

19.1.2 Die Windows-Firewall

Mit Hilfe der Windows-Firewall kann ein weiteres Stück Sicherheit erzeugt werden. Sie ermöglicht es, die Kommunikation zwischen dem Datenbankserver und seinen Clients sowohl mit statischen als auch mit dynamischen Ports zu regeln.

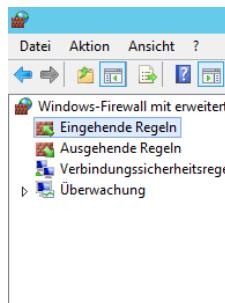


Da es die unterschiedlichsten Firewall-Produkte auf dem Markt gibt, wird die Windows-Firewall hier nur als stellvertretendes Beispiel herangezogen, um aufzuzeigen, wie die Netzwerkkommunikation zwischen Client und Server gesichert werden kann.

Die Verwaltung der Windows-Firewall erfolgt mit dem MMC-Plugin „Windows-Firewall mit erweiterter Sicherheit“. Dieses kann durch die „Ausführen“-Funktion des Windows Startmenüs mit WF.msc gestartet werden.

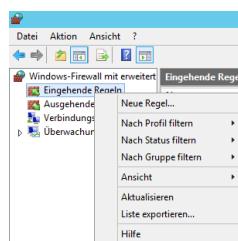
1. Klicken Sie im Firewallmenü, links außen, auf die Option „Eingehende Regel“.

Abb. 19.1:
Das Windows-
Firewall
MMC-Plugin



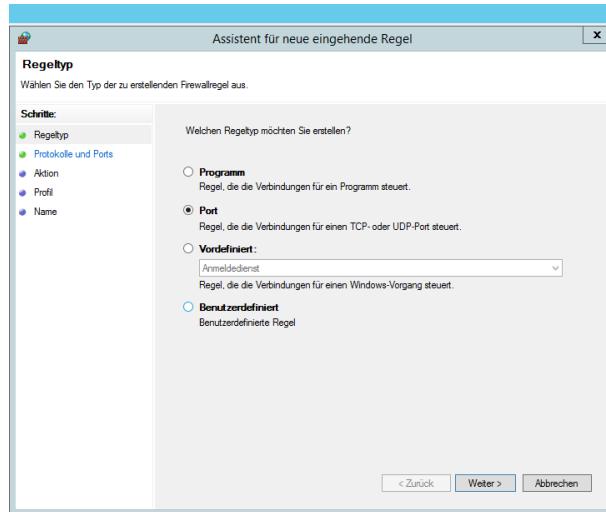
2. Öffnen Sie mit der rechten Maustaste das Kontextmenü und wählen Sie die Option „Neue Regel“.

Abb. 19.2:
Neue eingehende
Regel erstellen



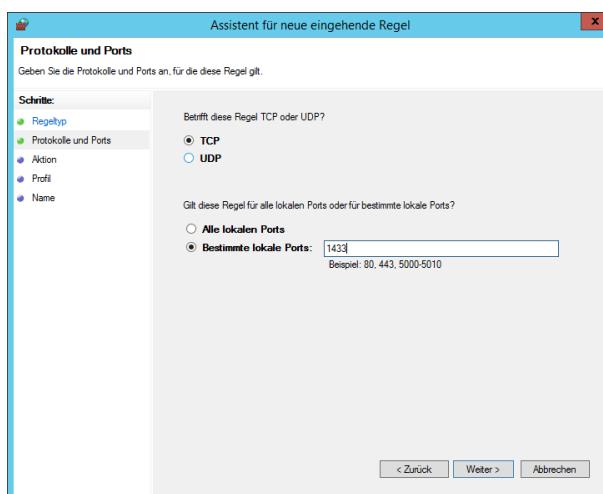
3. Wählen Sie in Schritt eins, des „Assistenten für eingehende Regeln“ die Option „Port“.

Abb. 19.3:
Eine Regel für
Ports erstellen



4. Klicken Sie auf „Weiter“
5. Wählen Sie in Schritt zwei die Optionen „TCP“ und „Bestimmte lokale Ports“ und geben Sie die Portnummer 1433 an.

Abb. 19.4:
TCP-Port 1433
ansprechen



6. Klicken Sie auf „Weiter“.
7. Im Schritt „Aktion“ des Assistenten muss die Option „Verbindung zulassen“ aktiviert bleiben.
8. Klicken Sie auf Weiter

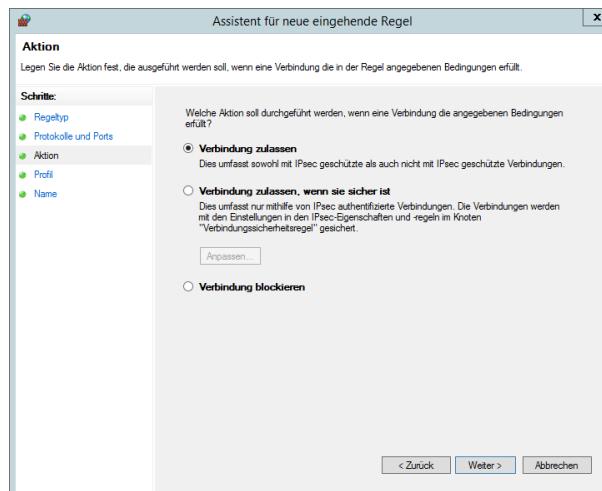


Abb. 19.5:
Zulassen der
Verbindung

9. Die eingehende Regel soll für alle drei Profile gelten.

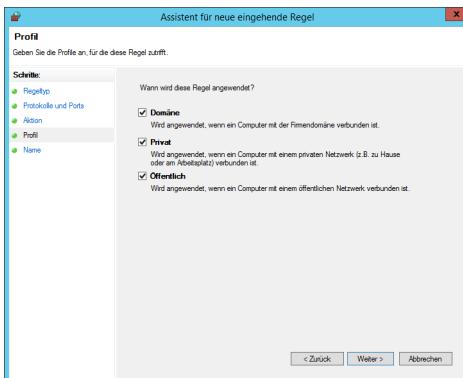


Abb. 19.6:
Anwenden der
Regel auf alle
Netzwerkprofile

10. Vergeben Sie den Namen „Database Engine“ für die Policy.

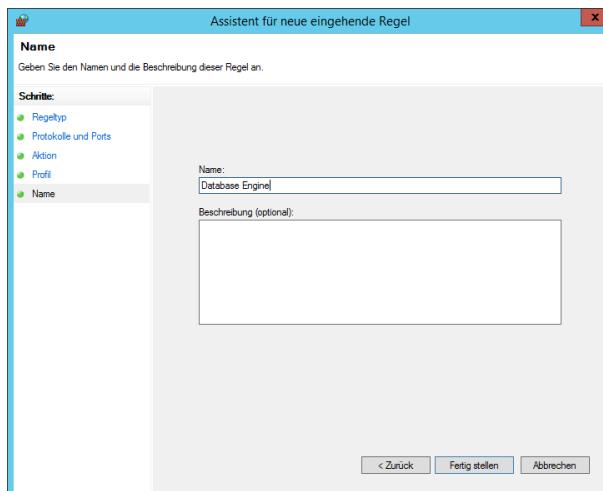


Abb. 19.7:
Fertigstellen des
Assistenten

- [ms175043]



19.2 Konfigurieren der Authentifizierung

Für die Authentifizierung der Clients kann der SQL Server zwei verschiedene Protokolle benutzen: NTLM und Kerberos. NTLM ist die Standardmethode und wird immer dann genutzt, wenn Kerberos nicht konfiguriert wurde oder nicht genutzt werden kann.

19.2.1 Unterschiede zwischen NTLM und Kerberos

NT LAN Manager Authentifizierung (NTLM)

Das LAN Manager-Protokoll, ursprünglich von Microsoft entwickelt, kam lange Zeit nur in Microsoft-produkten vor. Mit Hilfe von Reverse Engineering hat dieses Protokoll aber auch Einzug in andere Softwareprodukte erhalten. Ende der 1990er Jahre erfolgte dann die Weiterentwicklung des LAN Manager-Protokolls zum NT LAN Manager-Protokoll. NTLM basiert auf der folgenden Vorgehensweise:

1. Der Client sendet seinen Benutzernamen im Klartext an den Server.
2. Der Server sendet im zweiten Schritt eine Zufallszahl an den Client. Diese Zahl wird als „Challenge“ bezeichnet.
3. Der Client sendet eine mit dem Hash-Wert seines Benutzerpasswortes verschlüsselte Antwort, die „Response“ zurück.
4. Der Server verschlüsselt nun seinerseits die Challenge mit dem Benutzerpasswort und vergleicht das Ergebnis mit der Response. Bei Wertgleichheit gilt der Client als authentifiziert.

Die Größte Schwachstelle dieses Protokolls ist die unverschlüsselte Übertragung des Benutzernamens. Außerdem konnte es bei der ersten Version des NTLM-Protokolls vorkommen, dass lange Passwörter schwächer waren als Kurze, weshalb NTLMv2 entwickelt wurde.

Das Kerberos-Protokoll

Das Kerberos-Protokoll hat seinen Ursprung im „Needham-Schroeder-Protokoll“, welches 1978 als Protokoll zur Authentifizierung in unsicheren Netzwerken entworfen wurde. Kerberos gilt als äußerst sicher, weshalb Microsoft die Nutzung von Kerberos als Standard vorschlägt. Die Authentifizierung wird durch einen extra Kerberos-Dienst vorgenommen, der als „vertrauenswürdige dritte Partei“ (Trusted Third Party) oder auch als „Key Distribution Center“ (KDC) bezeichnet wird. In der Welt von Microsoft übernimmt die Rolle des KDC der Domain Controller einer Active Directory Domäne. Die Vorteile von Kerberos gegenüber NTLM sind:

- **Multilaterale Authentifizierung:** Kerberos-Dienst, Zielserver und Client authentifizieren sich gegenseitig, wodurch Man-In-The-Middle-Attacken vermieden werden.

- **Sichere Datenübertragung:** Verschlüsselte Übertragung aller Informationen.
- **Single-Sign-On:** Vermeidung mehrfacher Anmeldevorgänge.

Der Mechanismus den Kerberos verwendet basiert auf der Verteilung von verschlüsselten Tickets. Diese Tickets stellen für die Clients die Berechtigung dar auf bestimmte Dienste (die Services) zuzugreifen. Für jeden Service den ein Client benutzen möchte muss er sich ein eigenes Ticket holen.

Ein Kerberos-Authentifizierungsvorgang läuft wie folgt ab:

1. Der Client meldet sich beim Kerberos-Server und fordert ein „Ticket Granting Ticket“ (TGT) an. Bei einem TGT handelt es sich um eine kleine verschlüsselte Datei, welche die IP-Adresse des Clients, eine Session-ID und eine Gültigkeitsdauer beinhaltet. Zur Anforderung des TGT muss der Client unter Umständen ein Passwort angeben.
2. Der Server prüft, ob sich der Client in seiner „Realm DB“ befindet und erteilt dem Client das TGT.
3. Der Client kann nun ein „Service Ticket“ für einen speziellen Dienst beim KDC anfordern. Hierfür sendet er sein TGT und den SPN des gewünschten Dienstes an den KDC.
4. Der KDC erteilt dem Client das Service Ticket
5. Der Client kann mit Hilfe des Service Tickets nun den gewünschten Service benutzen.

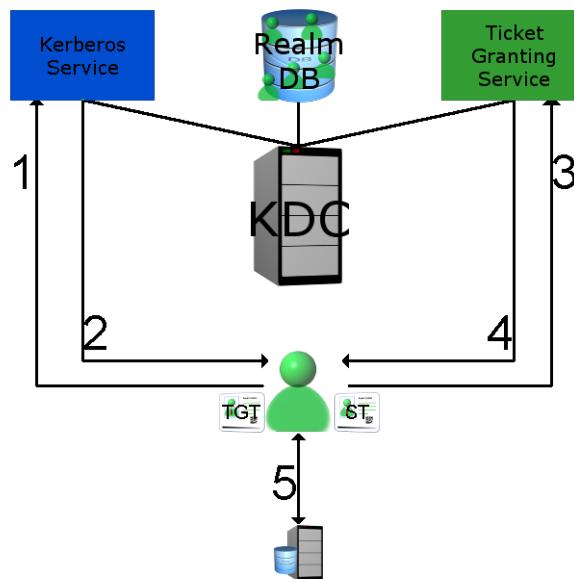
- [cc280744]
- [cc280745]
- [wikipediaTicketGrantingTicket]



19.2.2 Service Principal Names (SPN)

In einer Active Directory Umgebung darf kein Zugriff auf einen Dienst, ohne eine vorherige Authentifizierung des Clients erfolgen. Für die Authentifizierung benutzt Microsoft Windows das Protokoll „Kerberos“, und ein Bestandteil dieses Protokolls sind die sogenannten „Service Principal Names“, abgekürzt „SPN“. Ein SPN, zu Deutsch „Dienst Prinzipal Name“, ist vereinfacht ausgedrückt ein Name für einem Windows Dienst, mit dem dieser sich im Active Directory registriert.

Abb. 19.8:
Das Kerberos-
verfahren



Aufbau eines SPN

Ein SPN besteht aus maximal vier Teilen, die durch Slashes (/) von einander getrennt werden.

<service class>/<host>:<port number>/<service name>

- **service class:** Eine Zeichenfolge, welche die Art des Dienstes identifiziert, beispielsweise steht MSSQLSVC für den SQL Server Dienst.
- **host:** Abhängig vom jeweiligen Dienst ist dies die IP-Adresse, der NetBIOS-Name oder der FQDN¹ des betreffenden Servers auf dem der Dienst bereitgestellt wird. Dieser Teil wird manchmal auch als „Instance Name“ bezeichnet.
- **port number:** Eine optional anzugebende Portnummer, mit deren Hilfe mehrere Instanzen, des gleichen Dienstes, auf einem Host unterschieden werden können.
- **service name:** Ebenfalls eine optionale Angabe, die zur Unterscheidung unterschiedlicher Instanzen ein und des selben Dienstes auf einem Host benutzt werden kann.

Der Teil Service Class wird oft auch als „Service Type“ bezeichnet. Es existieren eine Reihe sogenannter „Well-known services“, wie z. B. www für Web Services, HTTP für Webseiten-/server oder auch LDAP für Directory Services. Grundsätzlich muss der Service Type aber nicht mehr als eine eindeutige Zeichenfolge sein.

¹FQDN = Fully Qualified Domain Name

Der SPN der SQL Server Standardinstanz besteht aus zwei bis drei Bestandteilen:

MSSQLSvc/fea11-119srv20.ms-c-ix-04.fus

Das Beispiel zeigt einen SPN des Typs SQL Server, der auf dem Host FEA11-119SRV20.MS-C-IX-04.FUS bereitgestellt wird. Optional kann hier auch eine Portnummer mit aufgenommen werden, falls der SQL Server nicht auf seinem Standardport 1433 betrieben wird.

MSSQLSvc/fea11-119srv20.ms-c-ix-04.fus:1423

Da der SQL Server ein replizierbarer Dienst ist, kann es mehrere parallele Instanzen auf einem Host geben. In solch einem Fall ist der Service Name von Bedeutung, da er den Namen der benannten SQL Server Instanz enthält.

MSSQLSvc/fea11-119srv20.ms-c-ix-04.fus/CRM

MSSQLSvc/fea11-119srv20.ms-c-ix-04.fus/ERM

Beidesmals sind die zwei vorderen Teile gleich, nur der Service Name ist unterschiedlich.



Wichtig ist der Service Name besonders dann, wenn ein Verbindungsprotokoll zum Einsatz kommt, dass keine Portnummern kennt, wie z. B. Named Pipes.

- [ms677601]
- [qzaidi20101012]



Verbreitung von SPNs - Service Publication

Wie eingangs bereits erwähnt, muss ein Dienst seinen SPN im Active Directory registrieren, damit Clients zu ihm Kontakt aufnehmen können. Der Vorgang während dem der Dienst seinen SPN im AD registriert wird als „Service Publication“ bezeichnet.

Unter normalen Umständen wird ein SPN von einem Setup-Programm erzeugt und im Active Directory registriert. Bei SQL Server 2014 werden in einem Installationsschritt die Dienstkonten für die verschiedenen Konten abgefragt und nach Abschluß des Schrittes im AD registriert.



Damit die Service Publication funktionieren kann, muss der Benutzer, der die Installationsroutine ausführt die Berechtigung besitzen Objekte im Active Directory anzulegen.

Wird nach erfolgter Service Publication das Konto eines Dienstes geändert, muss die SPN beim alten Dienstkonto gelöscht und mit dem neuen registriert werden. Dieser Schritt kann entfallen, wenn MSAs genutzt werden, da diese ihre SPN selbst verwalten.

Wo werden die SPN-Informationen abgelegt?

Ein SPN ist immer mit einem Benutzer- oder Computerkonto verknüpft. Welche Kontoart benutzt wird hängt davon ab, welcher Kontotyp als Dienstkonto festgelegt wurde.

- **Virtual Account:** Da ein virtuelles Konto das Computerkonto des lokalen Computers benutzt, um auf Netzwerkressourcen zuzugreifen, wird in diesem Fall der SPN mit dem Computerkonto registriert.
- **Benutzerkonto/MSA:** Hier wird der SPN mit dem betreffenden Benutzerkonto registriert. Der einzige Unterschied zwischen Benutzerkonto und MSA ist der, dass MSAs die notwendigen Berechtigungen haben, um ihre SPNs selbst zu verwalten.



SPNs werden im LDAP-Attribut „servicePrincipalName“ aufgelistet. Dessen Inhalt kann im ADSI-Editor sichtbar gemacht werden.



- [ms191153]
- [cc755804]

19.2.3 Umstellung auf Kerberos

Um das Authentifizierungsverfahren auf Kerberos umzustellen muss der Administrator lediglich einen einzigen Schritt durchführen:

Dem SQL Server-Dienstkonto muss ein gültiger Service Principal Name (SPN) zugeordnet werden.

Die Zuordnung des SPN kann manuell oder automatisch erfolgen.

Automatische Zuordnung eines SPN

Die automatische Zuordnung eines SPN für den SQL Server-Dienst geschieht beim Starten des Dienstes. Die Voraussetzung dafür ist, dass sein Dienstkonto die Berechtigung hat einen SPN zu registrieren. Standardmäßig haben nur die Konten „Lokale System“ und „Netzwerk Dienst“ die Berechtigung einen SPN zu registrieren. Allen anderen Konten muss dieses Recht erst erteilt werden. Gehen Sie hierzu wie folgt vor:

1. Starten Sie das MMC-Plugin „ADSI-Editor“

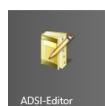


Abb. 19.9:
Starten des
ADSI-Editors

2. Öffnen Sie per Rechtsklick auf „ADSI-Editor“ das Kontextmenü und klicken Sie auf „Verbindung herstellen“.

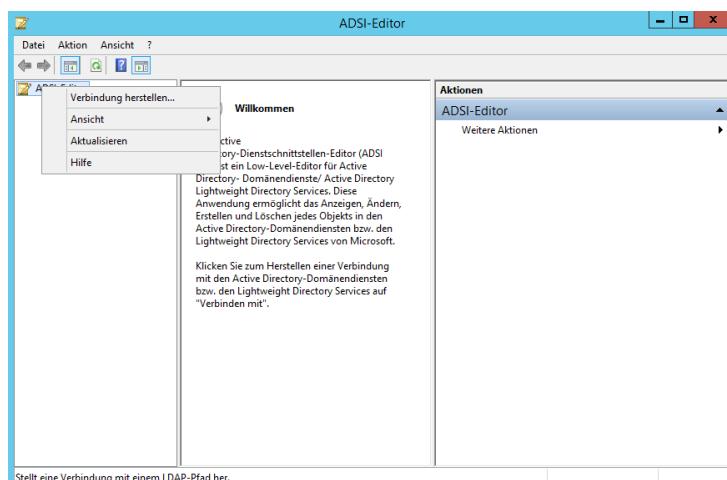


Abb. 19.10:
Herstellen einer
Verbindung zur
Domäne

3. Klicken Sie auf OK um die Verbindung zur Domäne herzustellen.

4. Navigieren Sie im Verzeichnisbaum bis zum gewünschten gMSA.

5. Öffnen Sie das Kontextmenü des gMSA und klicken Sie auf „Eigenschaften“

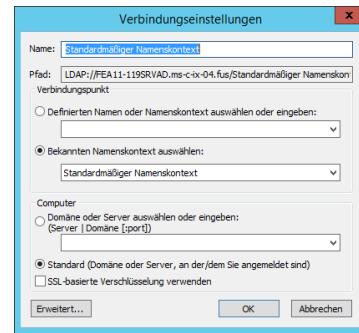


Abb. 19.11:
Verbindungs-einstellungen

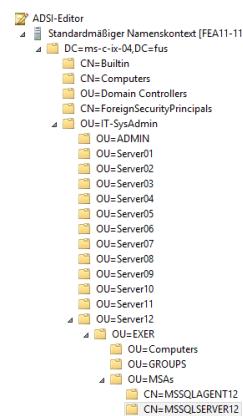


Abb. 19.12:
Navigation im
Verzeichni

6. Wählen Sie die Registerkarte „Sicherheit“ und klicken Sie auf das Benutzerkonto „SELBST“.

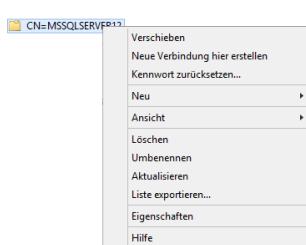


Abb. 19.13:
Der
Eigenschaften-
dialog des
gMSA

7. Klicken Sie auf „Erweitert“.

8. Wählen Sie im Dialog „Erweiterte Sicherheitseinstellungen“ den Benutzer „SELBST“ aus. Dieser Eintrag existiert u. U. mehrfach. Klicken Sie auf die Schaltfläche „Bearbeiten“ sobald Sie den richtigen Benutzer gefunden haben.

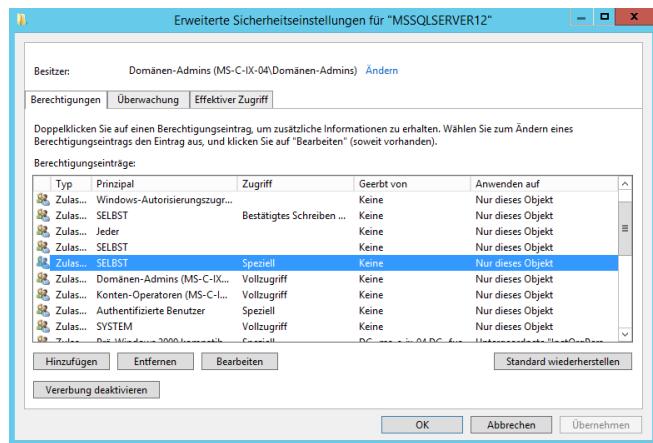


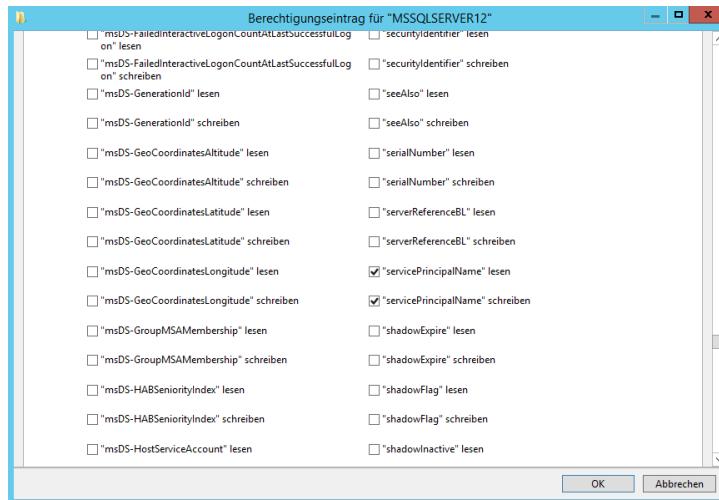
Abb. 19.14:
Erweiterte
Sicherheits-
einstellungen

9. Scrollen Sie im Dialog „Berechtigungseintrag“ soweit nach unten, bis Sie die beiden Berechtigungen

- ServicePrincipalName lesen
- ServicePrincipalName schreiben

gefunden haben und wählen Sie diese aus.

Abb. 19.15:
Auswählen der
Berechtigungen



10. Schließen Sie die Eigenschaftsdialoge und den ADSI-Editor.

11. Starten Sie den SQL Server-Dienst neu.

Im ErrorLog des SQL Server kann nach dem Neustart ersehen werden, ob die automatische Zuordnung des SPN funktioniert hat. Es erscheint die folgende Meldung:

The SQL Server Network Interface library successfully registered the Service Principal Name (SPN)
MSSQLSvc/FEA11-119SRVXX.ms-c-ix-04.fus for the SQL Server service.

Beim Fehlschlagen der Zuordnung wird die folgende Fehlermeldung angezeigt:

The SQL Server Network Interface library could not register the Service Principal Name (SPN) MSSQLSvc/FEA11-119SRVXX.ms-c-ix-04.fus for the SQL Server service. Windows return code: 0xXXXXXX, state: 15. Failure to register a SPN might cause integrated authentication to use NTLM instead of Kerberos.

Sollte diese Fehlermeldung erscheinen, kann sie durch die manuelle Zuordnung eines SPN behoben werden.



- [btmcn210tsnilwutrs]
- [mswcttssnilcnrtspns]
- [ms191153]

Manuelle Zuordnung eines SPN

1. Starten Sie das MMC-Plugin „ADSI-Editor“

2. Öffnen Sie per Rechtsklick auf „ADSI-Editor“ das Kontextmenü und klicken Sie auf „Verbindung herstellen“.
3. Klicken Sie auf OK um die Verbindung zur Domäne herzustellen.
4. Navigieren Sie im Verzeichnisbaum bis zum gewünschten gMSA.
5. Öffnen Sie das Kontextmenü des gMSA und klicken Sie auf „Eigenschaften“
6. Scrollen Sie auf der Registerkarte „Attribut-Editor“ soweit nach unten, bis Sie den Eintrag „servicePrincipalName“ gefunden haben. Wählen Sie ihn aus und klicken Sie auf „Bearbeiten“.

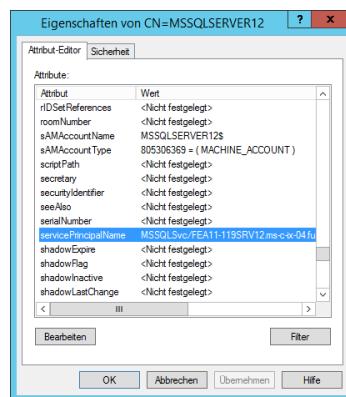


Abb. 19.16:
Der
Attribut-Editor

7. Fügen Sie den SPN MSSQLSvc/FEA11-119SRVXX.ms-c-ix-04.fus hinzu.

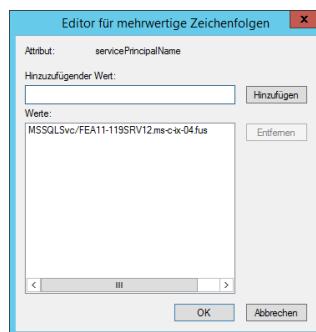


Abb. 19.17:
Das Attribut
servicePrincipal-
Name

8. Schließen Sie alle Dialogfenster und den ADSI-Editor.
9. Starten Sie den SQL Server-Dienst neu.

Die Zuordnung eines SPN überprüfen

Ob die Zuordnung des SPN funktioniert hat kann mit Hilfe des Kommandos setspn in einer Eingabeaufforderung überprüft werden.

```
setspn -L MS-C-IX-04\MSSQLSERVER12$
```

Bei erfolgreicher Zuordnung erscheint eine Liste der SPNs, die dem gMSA MSSQLSERVER12\$ zugeordnet sind.

Abb. 19.18:
Ergebnis von
setspn.exe

```
Administrator: Eingabeaufforderung
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Windows\system32>setspn -L MS-C-IX-04\MSSQLSERVER12$>
Registrierte Dienstprinzipalnamen (SPNs) für CN=MSSQLSERVER12, OU=MSAs, OU=EVER, OU=Server12, OU=IT-SysAdmin, DC=ms-c-ix-04, DC=fus:
MSSQLSvc/FEA11-119SRU12.ms-c-ix-04.fus

C:\Windows\system32>
```

Wird Kerberos nun wirklich verwendet?

Die Katalogsicht `SYS.DM_EXEC_CONNECTIONS` kann darüber Auskunft geben, ob Kerberos für die Authentifizierung genutzt wird oder nicht.

Listing 19.1: Wie wird authentifiziert?

```
SELECT net_transport, client_net_address, encrypt_option, auth_scheme
FROM sys.dm_exec_connections;
```



NET_TRANSPORT	CLIENT_NET_ADDRESS	ENCRYPT_OPTION	AUTH_SCHEME
TCP	192.168.111.43	TRUE	KERBEROS

1 Zeile ausgewählt

19.3 Von Privilegien, Rollen, Prinzipalen und Securables

19.3.1 Privilegien

Zuweisen, Entziehen und Verweigern

Um mit Datenbankobjekten arbeiten zu können, müssen Benutzer dazu berechtigt werden bestimmte Aktionen auszuführen. Dies kann z. B. das Verbinden mit der SQL Server-Instanz, das Lesen von Tabellenzeilen oder auch das Ändern von Werten in Tabellen sein. Solche Berechtigungen werden im Datenbankumfeld als „Privilegien“ bezeichnet. Privilegien können Nutzern zugewiesen oder entzogen werden. Zusätzlich dazu kennt der Microsoft SQL Server auch die Möglichkeit einem Benutzer ein Privileg explizit zu verweigern.

Die Auswirkungen beim Zuweisen, Entziehen oder Verweigern von Privilegien sind vergleichbar mit den Auswirkungen bei der Verwaltung von NTFS-Rechten.

- **Zuweisung:** Einem Benutzer wird ein Privileg erteilt. Er kann dadurch eine bestimmte Aktion ausführen.
- **Entzug:** Einem Benutzer wird ein zugeteiltes Privileg weggenommen, er kann eine Aktion nicht mehr ausführen.
- **Verweigerung:** Einem Benutzer wird die Ausübung eines Privileges explizit verboten. Dabei ist es unerheblich, ob dem betroffenen Benutzer ein Privileg erteilt wurde oder nicht. Durch die Verweigerung kann er bestimmte Aktionen nicht ausführen.



Zuteilung, Entzug und Verweigerung von Privilegien wirken sich immer augenblicklich auf den oder die betroffenen Benutzer aus.

Bei der Zuteilung von Privilegien an Benutzer sollte immer nach dem LUA-Prinzip („Least-Privileged User Account“) vorgegangen werden, was bedeutet, dass einem Benutzer immer nur ein minimaler Privilegiensatz zugewiesen werden soll. Wichtig ist, dass dieses Prinzip auch schon bei der Entwicklung von Datenbankanwendungen genutzt wird, da nur so ein konsequent nach LUA erstelltes Rollen- Rechtekonzept entsteht.

Durchführung der Berechtigungsprüfung

Um zu überprüfen, ob ein Prinzipal den Zugriff auf eine Resource erhält oder nicht, muss der SQL Server einen relativ komplexen Vorgang ausführen, in dessen Rahmen nach einer Berechtigungszuteilung gesucht wird, die einen Prinzipal mit ausreichenden Berechtigungen ausstattet. Hierbei müssen aber blockierende DENY-Anweisungen, überlappende Gruppenmitgliedschaften oder Besitzverkettungen berücksichtigt werden.

Die folgende Aufzählung zeigt, welche Schritte der SQL Server bei der Berechtigungsprüfung vollziehen muss. Die Reihenfolge kann von Vorgang zu Vorgang etwas anders sein.

1. Umgehe die gesamte Berechtigungsprüfung, wenn der betroffene Prinzipal ein Administrator (festen Serverrolle sysadmin) oder der Eigentümer einer Datenbank ist.
2. Erteilung des Zugriffs, sofern die Besitzverkettung anwendbar ist.
3. Identitäten, die dem Betroffenen User zuordenbar sind, aggregieren, um den Sicherheitskontext zu erstellen.



Als Sicherheitskontext wird die Gruppe der Prinzipale bezeichnet, die sich auf das aktuelle Login bzw. den Benutzer beziehen und die Berechtigungen für die Zugriffüberprüfung einbringen.

4. Auflisten der Berechtigungen, die für diesen Berechtigungsbereich zugewiesen oder verweigert worden sind.



Als Berechtigungsbereich wird das Securable und alle Securable Classes bezeichnet, in denen sich das Securable befindet. Beispielsweise befindet sich eine Tabelle (Securable) in einem Schema (Securable Class) welches sich in einer Datenbank (weitere Securable Class) befindet.)

5. Analysieren, welche Berechtigung für den aktuellen Vorgang benötigt wird.
6. Die Berechtigungsprüfung ist beendet (erfolgreich oder nicht erfolgreich).

Die Prüfung gilt als erfolgreich, wenn die benötigte Berechtigung für eine Identität innerhalb des Sicherheitskontextes für ein Securable im Berechtigungsbereich gefunden wurde (in Kurzform: Wenn der betroffene Benutzer das entsprechende Zugriffsrecht hat).

Namenskonventionen für Privilegien

Microsoft hat die Privilegien, die innerhalb des SQL Server zur Verfügung stehen, nach dem folgenden Namensschema benannt:

- **alter**: Räumt einem Prinzipal die Möglichkeit ein alle Eigenschaften, mit Ausnahme des Besitzes, eines Securables zu ändern. Wird dieses Privileg auf einen Gültigkeitsbereich erteilt, so ist damit nicht nur das Ändern, sondern auch das Erstellen und das Löschen von Securables enthalten.
- **alter any**: Durch den Zusatz **any** ist es möglich alle Securables eines Types zu Erstellen, zu Ändern oder zu Löschen. Diese Berechtigung kann nur für die Gültigkeitsbereiche Datenbank oder Server erteilt werden und betrifft, im Gegensatz zu **alter**, automatisch immer alle Securables in einem dieser Bereiche.
- **view definition**: Ermöglicht dem Empfänger Einsicht in die Metadaten eines Securables zu nehmen.
- **control**: Erteilt Privilegien, die denen des Besitzers eines Securables sehr ähnlich sind. Der Empfänger kann alle Eigenschaften eines Securables ändern (wie bei **alter**) und er kann auch Berechtigungen auf das Securable erteilen.

Diese Liste zeigt nur einen kurzen Ausschnitt aus dem Benennungsschema. Die vollständige Version, sowie eine Liste aller in SQL Server enthaltenen Privilegien kann unter dem folgenden Literaturhinweis eingesehen werden.

- [ms191291]

19.3.2 Rollen

Rollen Sie für eine Datenbank das, was unter Microsoft Windows Gruppen sind. Mit Hilfe von Rollen können Benutzer gruppiert und Privilegien zugeteilt werden. Sie werden benutzt, um die Erstellung und die Verwaltung eines Rechtekonzeptes zu vereinfachen. Grundsätzlich ist es einfacher eine Rolle mit Privilegien auszustatten und anschließend Benutzer der Rolle zuzuweisen bzw. sie daraus zu entfernen, als jedem Benutzer einen eigenen Privilegiensatz zu geben. Übersichtlichkeit und Verwaltbarkeit eines Systems werden durch die Nutzung von Rollen deutlich gesteigert.



Rollen können geschachtelt werden (Rollen werden Rollen zugewiesen). Jedoch sollte der Administrator/Entwicklern von einer allzu tiefen Verschachtelung absehen, da sonst die Performance einer Anwendung verschlechtert werden kann.



- [bb669084]

19.3.3 Prinzipale

Ein Prinzipal ist ein Objekt bzw. eine Entität, die in der Lage ist Ressourcen anzufordern und zu benutzen. In Bezug auf Microsoft Windows sind Benutzerkonten und Sicherheitsgruppen Prinzipale, da sowohl Benutzern als auch Gruppen Rechte zugewiesen werden können, mit deren Hilfe die Prinzipale dann Ressourcen, wie z. B. Dateien benutzen können.

Der SQL Server versteht unter anderem Rollen, Logins und Datenbankbenutzer als Prinzipale.



- [ms181127]

19.3.4 Securables - Sicherungsfähige Elemente

Securables (deutsche Übersetzung: Sicherungsfähige Elemente) sind Entitäten die durch Privilegien gesichert werden können. Sie stellen das Gegenstück zu den Prinzipalen dar. Ein Prinzipal muss bestimmte Privilegien besitzen, um auf ein Securable zugreifen zu können. Durch die Möglichkeit Securables in einander zu schachteln entstehen hierarchische Bereiche in der Datenbank, die wiederum selbst Securables darstellen und somit auch durch das Autorisierungssystem des SQL Server geschützt werden können.



- [ms190401]

19.4 Sicherheit auf Serverebene

19.4.1 Securables und Prinzipale

Securables

Auf Serverebene kennt der Microsoft SQL Server die folgenden Securables:

- Anmeldungen (Logins)
- Datenbanken
- Serverrollen
- Endpunkte
- Verfügbarkeitsgruppen

Einige dieser Securables werden im weiteren Verlauf dieser Unterrichtsunterlagen noch näher erläutert.

- [ms190401]



Prinzipale

Auf Serverebene kennt der SQL Server zwei Prinzipale: Anmeldungen (Logins) und Serverrollen.

Die Verknüpfung eines Benutzerkontos mit dem SQL Server wird als „Login“, zu Deutsch „Anmeldung“ bezeichnet. Der SQL Server kennt drei verschiedene Arten von Anmeldungen:

- Windows-Benutzer-Logins
- Windows-Gruppen-Logins
- SQL Server-Logins

Zusätzlich zu den aufgelisteten Anmeldetypen gibt es noch weitere, die allerdings lediglich zur Codesignierung genutzt werden können.

19.4.2 Serverrollen

Seit SQL Server 2005 werden Rollen auf Serverebene bereitgestellt, um dem Administrator die Verwaltung der Privilegie auf einem Server zu vereinfachen.

Feste Serverrollen

Der SQL Server 2014 bringt 9 feste Serverrollen mit, wobei jede dieser Rollen einen unveränderlichen Satz von Berechtigungen besitzt.

- **sysadmin:** Mitglieder werden zu Serveradministratoren, da die Rolle alle Privilegien beinhaltet, die auf Serverebene existieren.
- **bulkadmin:** Mitglieder können die SQL-Anweisung `BULK INSERT`-Ausführen.
- **dbcreator** Diese Rolle enthält alle Privilegien, um ihren Mitgliedern das Erstellen, Ändern, Löschen und Wiederherstellen von beliebigen Datenbanken zu ermöglichen.
- **diskadmin:** Mitglieder dieser Rolle können die Dateien der Datenbanken auf dem Datenträger verwalten.
- **processadmin** Mitglieder können Prozesse beenden, die innerhalb einer SQL Server-Instanz laufen.
- **public:** Diese Rolle verhält sich wie die Windows-Benutzergruppe `Jeder`.
- **securityadmin:** Mitglieder dieser Rolle können Anmeldungen verwalten und Berechtigungen auf Serverebene zuteilen, entziehen oder verweigern. Sofern ein Mitglied Zugriff auf eine Datenbank hat kann es auch dort Privilegien zuteilen, verweigern oder entziehen. Die Rolle `securityadmin` sollte mit der Rolle `sysadmin` gleichgestellt werden.
- **serveradmin:** Mitglieder dieser Rolle können Serverkonfigurationsoptionen verändern und den Server herunterfahren.
- **setupadmin:** Diese Rolle ermöglicht es ihren Mitgliedern Verbindungsserver mit Hilfe von T-SQL hinzuzufügen und zu entfernen.



Bei der Benutzung von Festen Serverrollen ist große Vorsicht geboten. Jedes Mitglied einer Festen Serverrolle kann andere Anmeldungen als Mitglied zur eigenen Rolle hinzufügen.



- [ms188659]
- [defsaldr2]

Eigene Serverrollen erstellen

1. Erweitern Sie im Objekt-Explorer die SQL Server-Instanz innerhalb derer die neue Serverrolle erstellt werden soll.
2. Wählen Sie das Register „Sicherheit“ und darin „Serverrollen“.
3. Öffnen Sie das Kontextmenü des Registers „Serverrollen“ und klicken Sie auf „Neue Serverrolle...“. Es erscheint der Dialog „Neue Serverrolle“



Abb. 19.19:
„Neue
Serverrolle...“

4. Geben Sie einen Namen für die Serverrolle ein und wählen Sie das Securable, dessen Privilegien Sie verwalten möchten.
5. Wählen Sie im unteren Bereich des Dialogs die benötigten Privilegien aus.
6. Klicken Sie auf „OK“.

Beispiel ?? zeigt den Vorgang der Rollenerstellung mit Hilfe von T-SQL.

Listing 19.2: Eine Serverrolle erstellen

```
USE [master]
GO

CREATE SERVER ROLE [UserHelpDesk]
GO
```

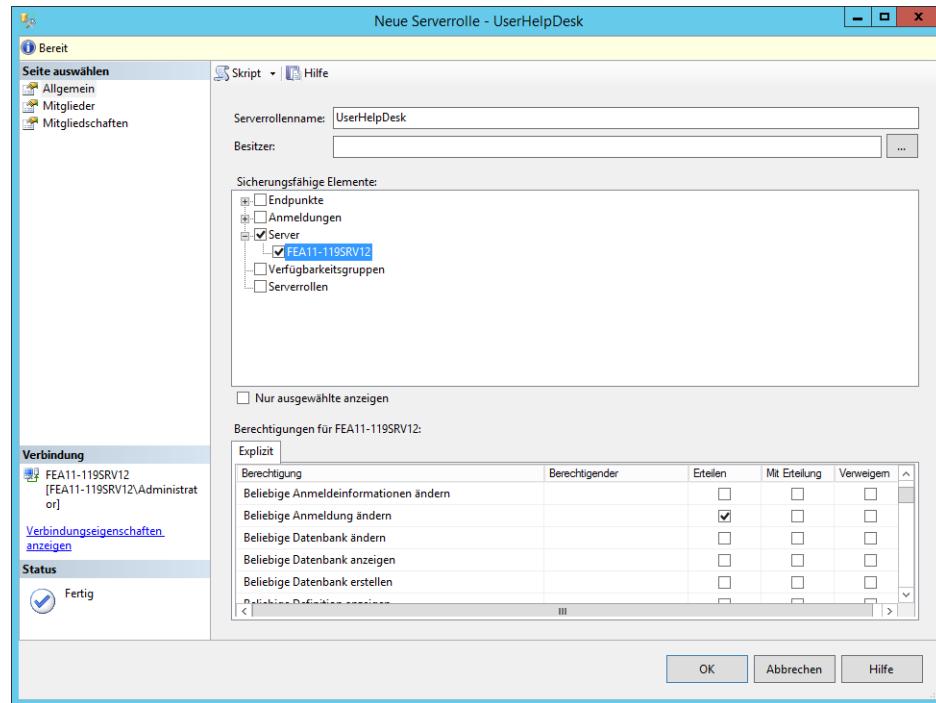


Abb. 19.20:
Der Dialog „Neue Serverrolle“

```
GRANT alter any login, alter any connection, impersonate any login
TO [UserHelpDesk]
GO
```



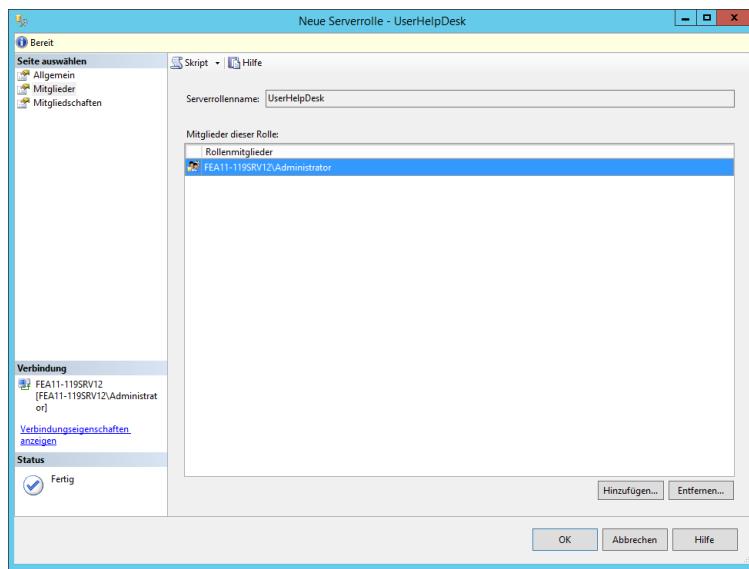
- [ee677627]
- [ee677610]

Mitglieder einer Rolle verwalten

Anmeldung und Serverrollen können Mitglieder in einer Serverrolle werden. Die Verwaltung der Mitglieder einer Rolle kann entweder mittels SSMS oder mittels T-SQL erfolgen.

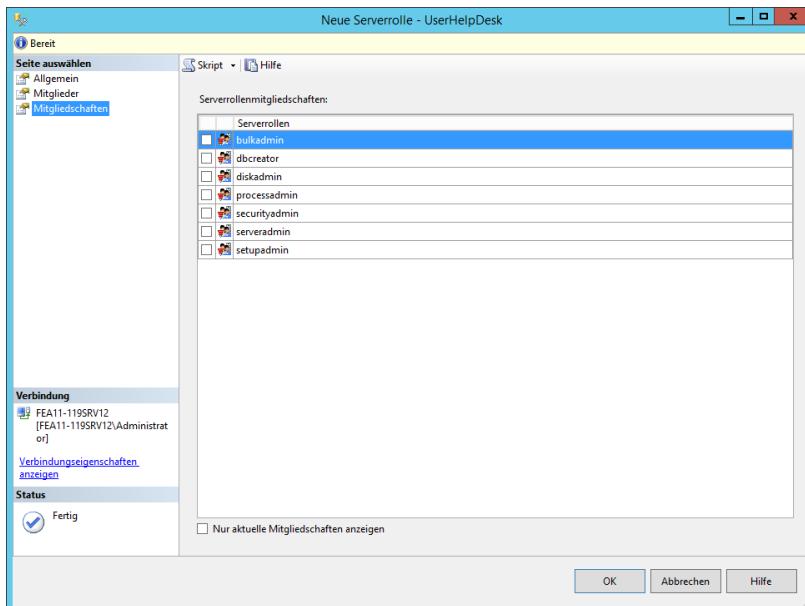
1. Öffnen Sie im Objekt-Explorer das Kontextmenü der Serverrolle und wählen Sie den Menüpunkt „Eigenschaften“.
2. Wechseln Sie links außen auf die Seite „Mitglieder“.
3. Klicken Sie auf „Hinzufügen“ um der Rolle neue Mitglieder hinzuzufügen oder wählen Sie ein Mitglied, das Sie entfernen möchten und klicken Sie auf „Entfernen“

Abb. 19.21:
Der Dialog „Neue
Serverrolle“



4. Wechseln Sie auf die Seite „Mitgliedschaften“. Hier kann die Rolle selbst Mitglied einer anderen Rolle werden.

Abb. 19.22:
Der Dialog „Neue
Serverrolle“



5. Klicken Sie auf „OK“.

Listing 19.3: Hinzufügen und entfernen von Mitgliedern zu/aus einer Serverrolle

```
ALTER SERVER ROLE [UserHelpDesk] ADD MEMBER [MS-C-IX-04\G-UserHelpdesk]
GO

ALTER SERVER ROLE [UserHelpDesk] DROP MEMBER [MS-C-IX-04\G-UserHelpdesk]
GO
```

- [ms189775]



Selbsterstellte Serverrollen löschen

Um eine selbsterstellte Serverrolle löschen zu können, müssen zuerst alle Mitglieder der Rolle gelöscht werden.

1. Öffnen Sie im Objekt-Explorer das Kontextmenü der Serverrolle und wählen Sie den Menüpunkt „Löschen“.
2. Klicken Sie auf „OK“.

Listing 19.4: Entfernen einer Serverrolle

```
DROP SERVER ROLE [UserHelpDesk]
GO
```



- [ee677643]

19.4.3 Privilegienverwaltung auf Serverebene

In SQL Server existieren auf Serverebene ca. 30 bis 40 verschiedene Privilegien, welche alle einen rein administrativen Charakter haben. Das bedeutet, dass diese Privilegien nur Administratoren und nicht den „gewöhnlichen“ Benutzer zugeordnet werden sollten.

Securables auf Serverebene

Auf Serverebene existieren in SQL Server 2014 insgesamt 5 verschiedene Securables:

- Login
- Endpoint
- Server
- Availabilitygroup
- Server role

Die wichtigsten Server-Privilegien im Überblick

- `control server`: Ermöglicht genau wie die Rolle `sysadmin` die vollständige Kontrolle über eine SQL Server Instanz. Der einzige Unterschied ist, dass ein Mitglied der Serverrolle `sysadmin` nicht durch ein DENY aufgehalten werden kann.
- `alter any database`: Ermöglicht das Anlegen und Löschen beliebiger Datenbanken.
- `alter any login`: Erlaubt das Anlegen, Ändern und Löschen von Logins. Logins, welche der Serverrolle `sysadmin` angehören können nur dann verändert werden, wenn man zusätzlich das Privileg `control server` besitzt.
- `alter settings`: Gibt dem Benutzer die Möglichkeit, die Konfiguration des Servers mit `sp_configure` zu verändern.
- `connect sql`: Ermöglicht es eine Verbindung zum Server aufzubauen. Dieses Privileg wird einem frisch erstellten Login automatisch zugewiesen.
- `view any database`: Erlaubt es die Metadaten aller Datenbanken mit Hilfe von `SYS.DATABASES`, `SYSDATABASES` oder `SP_HELPDB` sehen zu können.
- `view server state`: Ermöglicht es Einsicht in die Serverkonfiguration nehmen zu können.



- [mssqltips1714]
- [practdbacocsp]

Die wichtigsten Login-Privilegien im Überblick

- `control`: Ermöglicht die vollständige Kontrolle über ein Login. Es beinhaltet die anderen drei Privilegien `alter`, `impersonate` und `view definition`.
- `alter`: Ermöglicht es Veränderungen an einem Login vorzunehmen.
- `impersonate`: Der Besitzer dieses Privileges kann die Identität des betroffenen Logins annehmen.
- `view definition`: Ermöglicht es die Metadaten des betroffenen Logins sehen zu können.

Privilegien auf Serverebene erteilen



Privilegien auf Serverebene können nur dann erteilt werden, wenn die aktuelle Datenbank MASTER ist. Ist dies nicht der Fall, antwortet der SQL Server mit der Fehlermeldung 4621.

Listing 19.5: Privilegien auf ein Server- oder Login-Objekt erteilen

```
USE [master]
GO

GRANT control server TO SQLLogin;

GRANT view any definition TO SQLLogin;

GRANT view any database TO SQLLogin
WITH GRANT OPTION;

GRANT impersonate ON LOGIN::SQLLOGIN TO Adventureworks2012\sqluser;

GRANT view definition ON LOGIN::SQLLOGIN TO Adventureworks2012\sqluser;
```

Die Angabe **WITH GRANT OPTION** bei der Vergabe eines Privileges bewirkt, dass der betroffene Benutzer dieses Privileg weitergeben kann.



- [ms178640]
- [ms186717]

Privilegien auf Serverebene Entziehen/Verwehren

Das Entziehen und Verwehren von Privilegien auf Serverebene funktioniert mit den beiden Kommandos **REVOKE** und **DENY**. Wahlweise kann hierzu aber auch das SSMS genutzt werden.

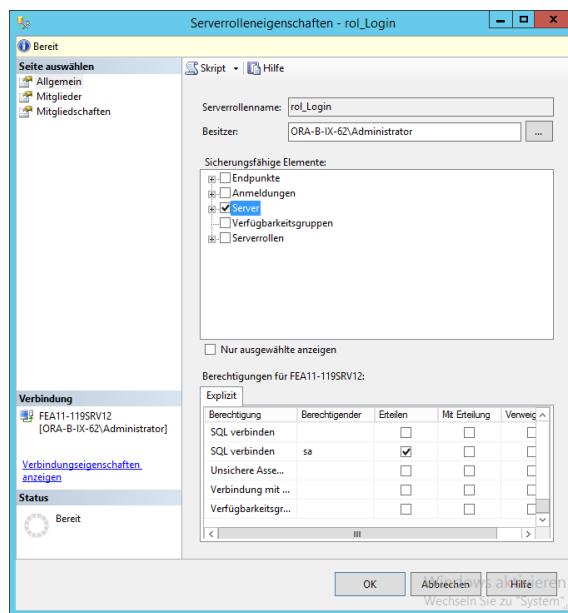
Listing 19.6: Privilegien von einem Server- oder Login-Objekt entziehen

```
USE [master]
GO

REVOKE control server FROM SQLLogin;

REVOKE GRANT OPTION FOR view any database FROM SQLLogin;
```

Abb. 19.23:
Rechtevergabe
auf Serverebene
im SSMS



```
REVOKE view any database FROM SQLLogin CASCADE;

REVOKE impersonate ON LOGIN::SQLLOGIN FROM Adventureworks2012\sqluser;
```

Das Kommando `REVOKE GRANT OPTION` entzieht nicht das Privileg selbst, sondern nur die `GRANT OPTION`. Durch die Angaben von `CASCADE` wird ein kaskadierender Privilegienentzug des `view any database` Privileges ausgeführt, was bedeutet, dass nicht nur der eigentliche Besitzer dieses Privileges, sondern auch alle nachgeordneten Besitzer das `view any database` Privileg verlieren.

Die Syntax des `DENY`-Kommandos stimmt mit der des `REVOKE`-Kommandos nahezu überein. Lediglich das Schlüsselwort `FROM` muss gegen `TO` ausgetauscht werden.

Listing 19.7: Privilegien von einem Server- oder Login-Objekt entziehen

```
USE [master] GO

DENY control server TO SQLLogin;

DENY view any database TO SQLLogin CASCADE;

DENY impersonate ON LOGIN::SQLLOGIN TO Adventureworks2012\sqluser;
```



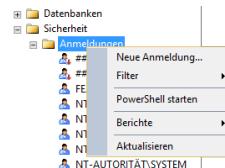
- [ms186226]
- [ms186710]

19.4.4 Verwalten von Logins

Erstellen von Windows-Logins

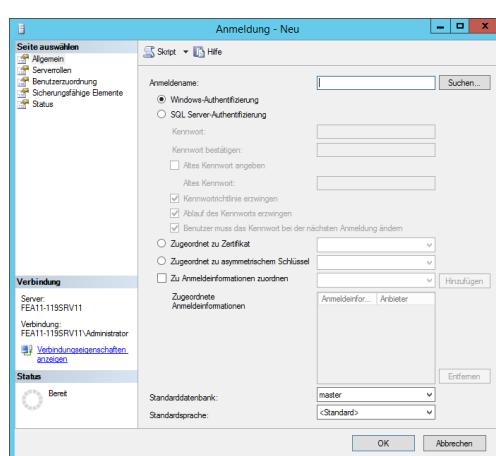
1. Öffnen Sie im Objekt-Explorer den Reiter „Sicherheit“!
2. Öffnen Sie das Kontextmenü des Reiters „Anmeldungen“ und klicken Sie auf „Neue Anmeldung“!

Abb. 19.24:
Ein neues Login
anlegen



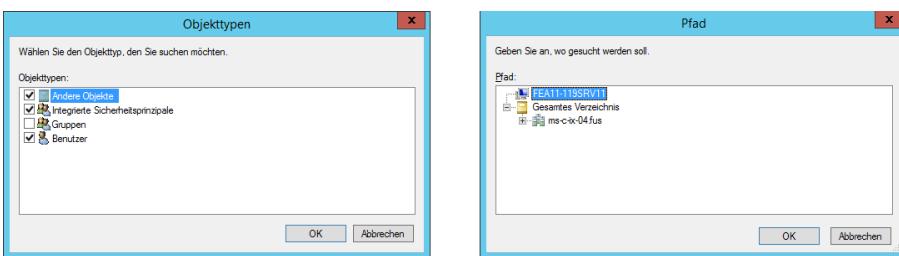
3. Klicken Sie im Dialog „Anmeldung - Neu“ rechts oben auf die Schaltfläche „Suchen...“!

Abb. 19.25:
Den gewünschten
Windows-
Benutzer
suchen



4. Wählen Sie den gewünschten Objekttyp und den Suchpfad aus!

Abb. 19.26:
Objekttyp und
Suchpfad
festlegen



5. Geben Sie den vollständigen Namen des Windows-Benutzerkontos ein!

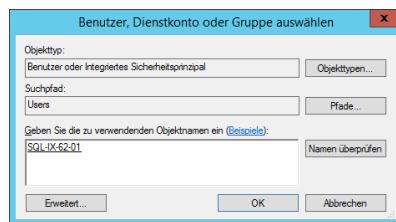


Abb. 19.27:
Eingeben des
Benutzernamens

6. Klicken Sie auf „OK“!

7. Wählen Sie Standarddatenbank für den Login aus. Dies ist die Datenbank mit der ein Benutzer nach der Anmeldung automatisch verbunden wird!

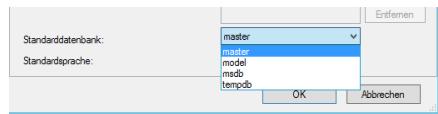


Abb. 19.28:
Festlegen der
Standard-
datenbank

8. Klicken Sie auf „OK“!

Soll das neue Login mit Hilfe von T-SQL angelegt werden, muss hierfür das Kommando `CREATE LOGIN` mit dem Zusatz `FROM WINDOWS` benutzt werden.

Listing 19.8: Anlegen eines Windows-Logins mit T-SQL

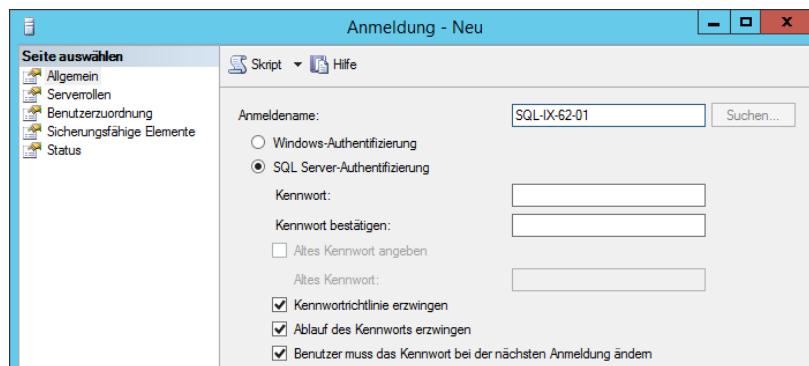
```
USE [master]
GO

CREATE LOGIN [MS-C-IX-04\SQL-IX-62-01]
FROM WINDOWS
WITH
    DEFAULT_DATABASE=[master]
GO
```

Erstellen eines SQL Server-Logins

1. Öffnen Sie im Objekt-Explorer den Reiter „Sicherheit“!
2. Öffnen Sie das Kontextmenü des Reiters „Anmeldungen“ und klicken Sie auf „Neue Anmeldung“!
3. Wählen Sie im Dialog „Anmeldung - Neu“ die Option „SQL Server-Authentifizierung“ aus und geben Sie einen Namen für das Login an!

Abb. 19.29:
Erstellen eines
SQL Server
Logins



4. Geben Sie ein Passwort für das Login an!
5. Wählen Sie die gewünschten Passwortrichtlinien aus!
6. Wählen Sie Standarddatenbank für den Login aus. Dies ist die Datenbank mit der ein Benutzer nach der Anmeldung automatisch verbunden wird!
7. Klicken Sie auf „OK“!

Listing 19.9: Anlegen eines SQL Server-Logins mit T-SQL

```
USE [master]
GO

CREATE LOGIN [SQL-IX-62-01]
WITH
    PASSWORD=N'P@ssw0rd', MUST_CHANGE ,
    DEFAULT_DATABASE=[master],
    CHECK_EXPIRATION=ON,
    CHECK_POLICY=ON
GO
```

Ändern von Logins

Mit Hilfe des `ALTER LOGIN`-Kommandos bzw. mittels des SSMS ist es möglich, alle Eigenschaften eines Logins zu verändern. Hierbei müssen die folgenden Punkte beachtet werden:

- Die Bezeichner von Domänenanmeldungen/Windows-Anmeldungen müssen in eckige Klammern gesetzt werden.
- Das Deaktivieren eines Logins wirkt sich nicht auf aktuell offene Sessions aus.



- [ms189828]

Löschen von Logins

Das Löschen eines Logins geschieht mit dem SQL-Kommando `DROP LOGIN`. Dabei ist zu beachten, dass ein Login, welches gerade benutzt wird, nicht gelöscht werden kann. Zuerst müssen alle Sessions, die sich auf das zulöschende Login beziehen beendet werden. Auch können Logins, welche Besitzer von Objekten sind nicht gelöscht werden. Der Objektbesitz muss erst an andere Logins übergeben werden.



- [ms188012]

19.4.5 Den Authentifizierungsmodus wählen

Der Microsoft SQL Server unterscheidet zwei verschiedene Arten der Authentifizierung:

- **Windows Authentifizierung:** Bei dieser Art von Authentifizierung werden verschiedene Windows Konten bzw. Gruppen mit dem SQL Server verknüpft. Durch diese Verknüpfung gelten Sie dann als Vertrauenswürdig und es erfolgt keine weitere Abfrage von Passwörtern oder anderen Sicherheitsmerkmalen.
- **Gemischter Modus:** Im gemischten Modus wird zusätzlich zur Windows Authentifizierung noch eine SQL Server-Authentifizierung unterstützt. Dabei speichert/verwaltet des SQL Server eigene Benutzerkonten. Dieser Modus gilt jedoch als unsicher, da bei der Benutzung der SQL Server-Authentifizierung der Benutzername und das Passwort über das Netzwerk übertragen werden.



Die Windows Authentifizierung wird auch als „Integrated Security“ bezeichnet.

Grundsätzlich empfiehlt Microsoft die Benutzung der Integrierten Sicherheit. Die SQL Server-Authentifizierung ist nur in wenigen Ausnahmefällen sinnvoll. Beispiele für solche Situationen sind:

- Es ist kein Domänencontroller im Netzwerk vorhanden (arbeiten in einer Arbeitsgruppe),
- Anmeldungen aus unterschiedlichen, nicht vertrauenswürdigen Domänen sind notwendig,
- für Web-Anwendungen

Integrated Security

Wird dieser Authentifizierungsmodus gewählt, werden Logins für Benutzerkonten und -gruppen des lokalen Windows-Servers oder einer vertrauenswürdigen Domäne im SQL Server erstellt. Hierbei gibt es zwei Punkte die zu beachten sind:

- Bei einem Windows-Gruppen-Login, haben alle Mitglieder dieser Gruppe die gleichen Zugriffsrechte auf den SQL Server.
- Existiert ein Windows-Benutzer-Login für einen Benutzer und ist dieser Benutzer gleichermassen Mitglied in einer Windowsgruppe, für die es ein Windows-Gruppen-Login gibt, so haben die Einstellungen des Windows-Benutzer-Logins Vorrang.

SQL Server Authentifizierung

Bei diesem Verfahren muss der Administrator die Benutzerkonten im SQL Server erstellen und verwalten. Gespeichert werden der Account und ein Hash-Wert des Passwortes in der MASTER-Datenbank. Wird der SQL Server auf Windows Server 2003 oder höher betrieben, können sogar die Kennwortrichtlinien des Active Directory auf die SQL Server Benutzerkonten angewendet werden, um so die Sicherheit dieser Konten zu erhöhen.

Bei der Installation des SQL Servers wird standardmäßig das SQL Server Benutzerkonto SA angelegt. Dieses Konto hat volle administrative Rechte und sollte daher gut gesichert werden. Vergeben Sie ein starkes Kennwort und lassen Sie den Account gesperrt.

19.4.6 Distributed Administrator Connection - DAC

Die Distributed Administrator Connection wird vom SQL Server zur Verfügung gestellt, damit der Administrator in speziellen Fällen, in denen keine normale Verbindung zur Instanz mehr möglich ist eine Problemdiagnose betreiben kann. Sie unterstützt Sicherheitsfeatures, wie z. B. Datenbankverschlüsselung und viele andere.



Nur Mitglieder der festen Serverrolle `sysadmin` können eine DAC aufbauen.

Verbindungsauflaufbau mit der DAC

Der Aufbau einer Verbindung mit Hilfe der DAC geschieht nicht über den Standardnetzwerkport 1433, sondern über Port 1434. Diese geschieht, um im Falle dessen, dass es Probleme auf dem Port 1433 gibt, trotzdem noch eine DAC aufgebaut werden kann. Standardmäßig kann eine DAC nur lokal erstellt werden, d. h. der Client muss sich direkt auf dem Datenbankserver befinden.



Wenn Port 1434 nicht zur Verfügung steht, wird beim Starten der Instanz ein DAC-Port dynamisch ausgewiesen und im Error Log notiert.

Im Folgenden wird gezeigt, wie mit Hilfe von `sqlcmd.exe` eine DAC zur Instanz erstellt wird.

Listing 19.10: SQLCMD.exe mit DAC benutzen

```
sqlcmd.exe -S 127.0.0.1 -A

sqlcmd.exe -S 127.0.0.1,1434

sqlcmd.exe -S FEA11-119SRV12 -U admin -P xxx -A

sqlcmd.exe -S FEA11-119SRV12 -U admin -P xxx -A -d Master
```

Der Schalter `-S` gibt an, zu welcher Instanz die Verbindung aufgebaut werden soll und der spezielle Schalter `-A` sagt aus, dass versucht werden soll, eine DAC zu benutzen.

Remoteverbindungen mit der DAC

Remote DAC Verbindungen können zugelassen werden, jedoch wird aus Sicherheitsgründen davon abgeraten. Falls remote DAC erforderlich sein sollte, wird dieses Feature durch die Einstellung `remote admin connections` konfiguriert.

Listing 19.11: Konfigurieren von Remote DAC Verbindungen

```
USE [master]
GO

sp_configure 'remote admin connections', 1;
GO

RECONFIGURE;
GO
```

Unmittelbar nach dem für `remote admin connections` der Wert 1 gesetzt wurde, wird der DAC Listener gestartet. Dies geschieht ohne Instanz-Neustart.

Um remote DAC wieder zu deaktivieren, muss für die Einstellung `remote admin connections` der Wert 0 gesetzt werden.



- [ms190468]

Einschränkungen

- Es ist nur eine DAC Verbindung pro Instanz erlaubt
- Die DAC versucht eine Verbindung zur Standarddatenbank des benutzten Logins herzustellen. Gelingt dies nicht, so wird die Fehlermeldung 4060 ausgegeben.
- SQL-Kommandos, die parallele Ausführung benutzen, wie z. B. `BACKUP` oder `RESTORE` sind bei Nutzung der DAC verboten.
- Für eine DAC stehen nur minimale Ressourcen zur Verfügung. Es sollten deshalb keine Ressourcenintensiven SQL-Kommandos ausgeführt werden.

19.5 Sicherheit auf Datenbankebene

19.5.1 Securables und Prinzipale

Securables

Auf Datenbankebene kennt der SQL Server 2014 insgesamt 16 verschiedene Securables. Im Folgenden werden die Prinzipale Datenbankrolle, Schema und Benutzer näher erläutert.

- [ms190401]



Prinzipale

Innerhalb von Datenbanken existieren drei verschiedene Prinzipale:

- Datenbankbenutzer
- Datenbankrollen
- Anwendungsrollen

19.5.2 Datenbankbenutzer

Damit sich ein Login mit einer Datenbank verbinden kann, muss dem Login ein Benutzer in der betreffenden Datenbank zugeordnet werden. Der Datenbankbenutzer stellt dann die Identität des Logins innerhalb der Datenbank dar. Beide, das Login und auch der Datenbankbenutzer können den gleichen Namen tragen. Dies ist aber nicht zwingend erforderlich.

Logins können in mehreren Datenbanken unterschiedlichen Datenbankbenutzern zugeordnet werden. Innerhalb einer Datenbank kann aber nur eine einzige Zuordnung zwischen einem Login und einem Benutzer erfolgen.

Der Gast-Benutzer

Das Gast-Benutzerkonto bietet die Möglichkeit, die direkte Zuordnung von Logins und Datenbankbenutzern auszuhebeln. Jedes Login muss normalerweise zwingend einem Datenbankbenutzer zugeordnet werden sein, um innerhalb einer Datenbank arbeiten zu können. Ist das Gast-Benutzerkonto einer Datenbank aktiviert, kann jedes Login auch ohne explizite Benutzerzuordnung eine Verbindung zu der betreffenden Datenbank herstellen.



Das Gast-Benutzerkonto ist ein integriertes Benutzerkonto, welches in jeder Datenbank existiert. Standardmäßig ist es deaktiviert und seine Verwendung ohne zwingenden Grund sollte vermieden werden. Der Gast-Account kann nicht gelöscht werden.

Der Benutzer „dbo“

Das Benutzerkonto dbo ist ein in jede Datenbank fest integriertes Konto, welches implizt alle Berechtigungen zum Ausführen beliebiger Operationen auf der jeweiligen Datenbank besitzt. Implizit bedeutet, dass für dieses Konto schlicht und einfach keinerlei Berechtigungsprüfung durchgeführt wird. Dies hat zur Folge, dass auch die explizite Verweigerung eines Privilegs keine Auswirkung auf den Nutzer dbo hat.

Von besonderer Bedeutung ist dieses Konto, wenn eine beschädigte Datenbank wiederhergestellt werden muss. Da die Zuordnung, zwischen Login und Datenbankbenutzer in der Datenbank MASTER einer jeden Instanz gespeichert wird, ist trotz Beschädigung der Datenbank bekannt, wer dem Benutzer dbo zugeordnet ist. Nur so kann eine Wiederherstellung der Datenbank erfolgen.



Alle Mitglieder der Festen Serverrolle sysadmin sind automatisch dem Benutzerkonto dbo zugeordnet.



- [bb669065]

Verschiedene Benutzertypen

Microsoft SQL Server kennt eine Reihe verschiedener Datenbankbenutzertypen für unterschiedliche Einsatzzwecke. Es wird unterschieden in:

- **SQL-Benutzer mit Anmeldename:** Diese Art von Benutzerkonto kommt häufig dann zum Einsatz, wenn ein Benutzer außerhalb der Windows-Domäne Zugriff auf den SQL Server benötigt. Ein klassischer Fall wäre z. B. ein Außendienstmitarbeiter. Die Verwaltung der Benutzersicherheit wird bei dieser Kontoart komplett vom SQL Server übernommen.
- **SQL-Benutzer ohne Anmeldename:** Benutzerkonten ohne Anmeldename kommen nur in Partially Contained Databases zum Einsatz. Ein Erläuterung deren Bedeutung/Benutzung erfolgt an entsprechender Stelle in diesem Manuskript.
- **Benutzer mit Zuordnung zu einem Zertifikat:** Benutzer die aus einem Zertifikat erstellt wurden können sich nicht am SQL Server anmelden. Sie dienen zur Ausführung von Prozessen (z. B. Datenbankspiegelung, Bulk Copy u. ä.) oder zur Verbesserung der Sicherheit von Datenbankanwendungen.
- **Benutzer mit Zuordnung zu einem asymmetrischen Schlüssel:** Genau wie die Benutzer mit Zuordnung zu einem Zertifikat können sich auch diese Benutzer nicht an der Datenbank anmelden. Auch ihr Einsatzzweck ist mit dem der Benutzer mit Zuordnung zu einem Zertifikat identisch.
- **Windows-Benutzer:** Mit dieser Kontoart ist es möglich, eine direkte Zuordnung zwischen einem Windows-Benutzer bzw. einer Windows-Benutzergruppe und einem SQL Server Benutzerkonto herzustellen. Die Verwaltung der Benutzersicherheit wird dabei von Windows übernommen.

- [bb895327]
- [somarskoggranperm]



19.5.3 Datenbankbenutzer verwalten

SQL-Benutzer mit Anmeldename erstellen

1. Wählen Sie im Objekt-Explorer den Ordner „Datenbanken“.
2. Klicken Sie auf die gewünschte Datenbank und navigieren Sie dort zum Ordner SICHERHEIT.

3. Öffnen Sie das Kontextmenü der Option BENUTZER und klicken Sie auf NEUER BENUTZER
4. Wählen Sie im Dropdownmenü BENUTZERTYP den Punkt SQL-BENUTZER MIT ANMELDENAME
5. Geben Sie im Feld BENUTZERNAME einen gültigen Benutzernamen an.
6. Geben Sie im Feld ANMELDENAME den Namen eines existierenden Logins an oder klicken Sie auf die Schaltfläche rechts neben dem Feld, um ein Login auszuwählen.
7. Klicken Sie auf OK.

Das Anlegen eines SQL-Benutzers mit Anmeldename kann auch mittels SQL-Statement geschehen.

Listing 19.12: Anlegen eines SQL-Benutzers mit Anmeldename

```
USE [master]
GO

-- Das Login muss im Vorfeld bereits erstellt worden sein.
CREATE LOGIN [SQL-IX-62-01]
WITH PASSWORD=N'password',
DEFAULT_DATABASE=[model],
CHECK_EXPIRATION=OFF,
CHECK_POLICY=OFF
GO

-- Anlegen des Datenbankbenutzers
CREATE USER [SQL-IX-62-01]
FOR LOGIN [SQL-IX-62-01];
GO
```



Wenn Sie bei der Auswahl des Logins ein Login mit Windows-Authentifizierung auswählen wird SQL Server automatisch einen Windows-Benutzer erstellen.

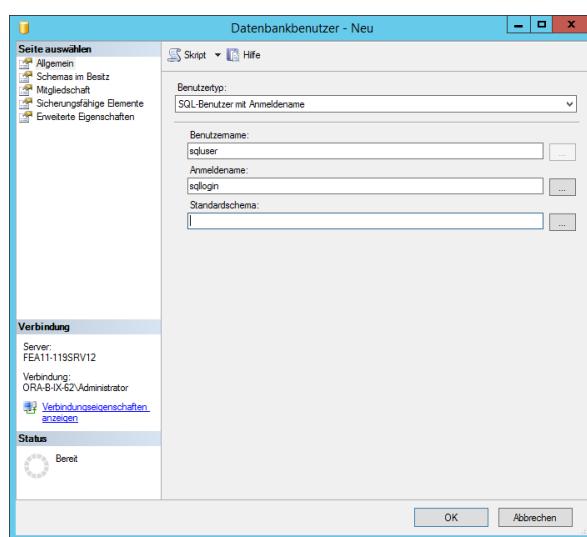


Abb. 19.30:
Anlegen eines
SQL-Benutzers
mit
Anmeldename

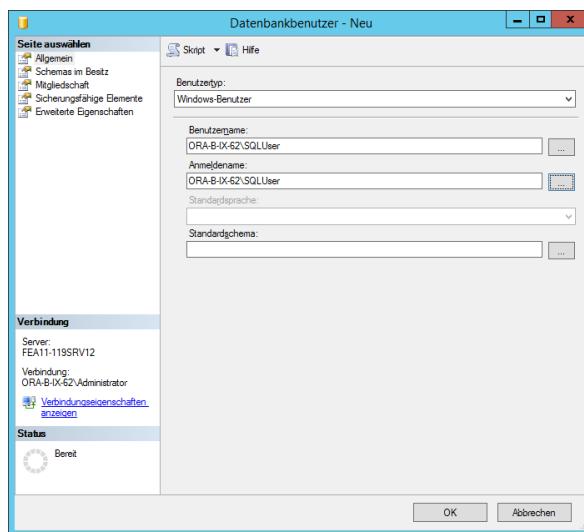
Anlegen von Windows-Benutzern

1. Wählen Sie im Objekt-Explorer den Ordner „Datenbanken“.
2. Klicken Sie auf die gewünschte Datenbank und navigieren Sie dort zum Ordner SICHERHEIT.
3. Öffnen Sie das Kontextmenü der Option BENUTZER und klicken Sie auf NEUER BENUTZER
4. Wählen Sie im Dropdownmenü BENUTZERTYP den Punkt „Windows-Benutzer“
5. Geben Sie im Feld BENUTZERNAME den Namen eines bereits existierenden Windows-Benutzeraccounts an. Achten Sie hierbei auf die korrekte Schreibweise: [Rechnername]\[Benutzername] bzw. [Domäne]\[Benutzername]. Ein Beispiel wäre der Benutzername „ORA-B-IX-62\SQLUser“.
6. Geben Sie im Feld ANMELDENAME den Namen eines existierenden Logins an oder klicken Sie auf die Schaltfläche rechts neben dem Feld, um ein Login auszuwählen.
7. Klicken Sie auf OK.



Achten Sie darauf, dass Sie bei der Auswahl des Logins ein Login mit Windows-Authentifizierung auswählen, da sie anderenfalles eine Fehlermeldung erhalten werden.

Abb. 19.31:
Anlegen eines
Windows-
Benutzers



Das Anlegen eines SQL-Benutzers mit Anmeldenamen kann auch mittels SQL-Statement geschehen.

Listing 19.13: Anlegen eines Windows-Benutzers

```
USE [master]
GO

-- Das Login muss im Vorfeld bereitstellt worden sein.
CREATE LOGIN [ORA-B-IX-62\sqluser]
FROM WINDOWS
WITH DEFAULT_DATABASE=[model]
GO

-- Anlegen des Datenbankbenutzers
CREATE USER [SQL-IX-62-01]
FOR LOGIN [SQL-IX-62-01];
GO
```



Wird **FOR LOGIN** weggelassen, wird der Benutzer einem gleichnamigen Login zugewiesen.

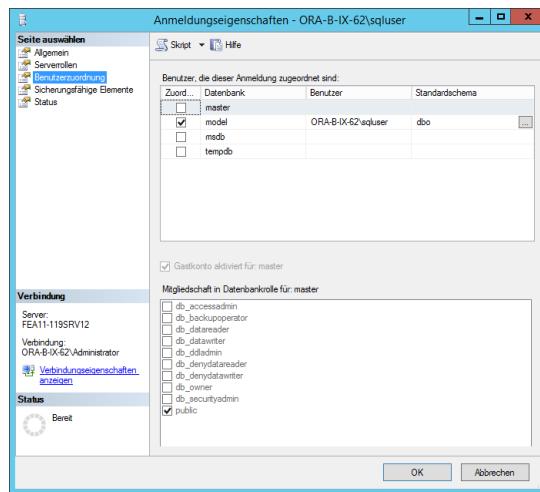


- [ms173463]

Benutzerzuordnungen

Als Benutzerzuordnung wird die Verbindung eines Logins zu einem Datenbankbenutzer in einer Datenbank bezeichnet. Diese Zuordnung kann im Menü SICHERHEIT - ANMELDUNGEN in den Eigenschaften des jeweiligen Logins auf der Seite BENUTZERZUORDNUNG eingesehen werden.

Abb. 19.32:
Ansicht der
Benutzerzuord-
nungen



Der Eigenschaftsdialog BENUTZERZUORDNUNG bietet einen alternativen Weg zum Erstellen von Datenbankbenutzern. Nach der Erstellung eines Logins muss lediglich bei der/den gewünschten Datenbank(en) ein Häckchen gesetzt werden und SQL Server richtet automatisch einen passenden Datenbankbenutzer ein.

Schemas im Besitz

Der Eigenschaftsdialog SCHEMAS IM BESITZ eines jeden Datenbankbenutzers zeigt die Schemata, welche in der Datenbank existieren, zu der der Benutzer gehört. Hier kann der Benutzer den Besitz eines Schemas übernehmen.



Der Nutzen und die Bedeutung von Datenbankschemata wird im weiteren Verlauf dieses Kapitels noch erläutert.

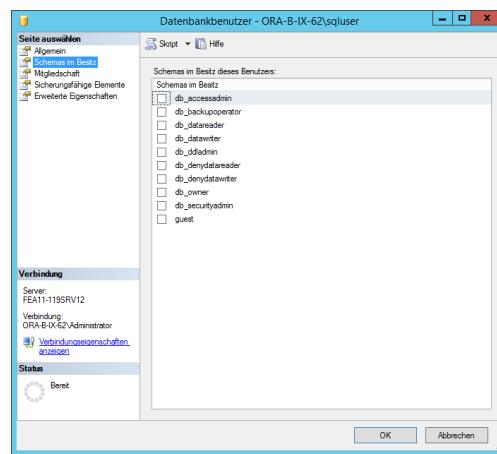


Abb. 19.33:
Schemas im
Besitz des
Benutzers

Mitgliedschaft

Im Eigenschaftsdialog MITGLIEDSCHAFT kann der Benutzer einer Datenbankrolle als Mitglied hinzugefügt werden.

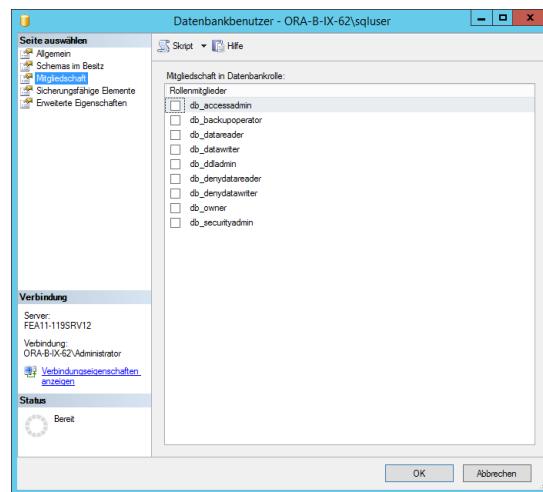


Abb. 19.34:
Rollenmitglied-
schaften eines
Benutzers

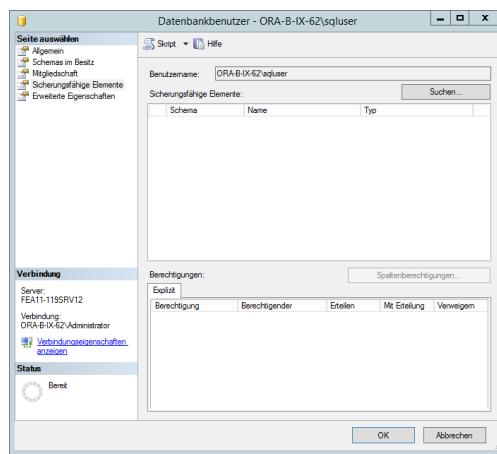
Sicherungsfähige Elemente

Mit Hilfe des Dialoges SICHERUNGSFÄHIGE ELEMENTE können einem Datenbankbenutzer direkt Privilegien für Securables zugewiesen werden.



Die Vergabe von Privilegien sollte niemals direkt an Benutzer erfolgen. Benutzen Sie hierfür ein Rollen- und Rechtekonzept. Ein Beispiel für ein solches Konzept wird noch weiteren Verlauf dieses Manuskriptes gegeben.

Abb. 19.35:
Sicherungsfähige
Elemente eines
Benutzers



- [aa337545]



Allgemeine Änderungen an einem Benutzerkonto vornehmen

Grundsätzlich können alle Attribute eines Benutzers im Nachhinein geändert werden. Beispielsweise können der Benutzername und das zugeordnete Login verändert werden.

Listing 19.14: Umbenennen eines Benutzers

```
USE [model]
GO

ALTER USER [SQLUser]
WITH NAME = [SQLLooser];
GO
```



Das Umbenennen eines Benutzers funktioniert nicht im Management Studio.

Für das Umbenennen eines Windows-Benutzers muss der gültige Name eines existenten Windows-Benutzeraccounts angegeben werden.

Soll dem Benutzer ein anderes Login zugewiesen werden, so muss beachtet werden, dass das neue Login den gleichen Typ (SQL Server- bzw. Windowsauthentifizierung) haben muss und das es bereits existieren muss.

Listing 19.15: Umbenennen eines Benutzers

```
USE [model]
GO

ALTER USER [SQLLooser]
```

```
WITH LOGIN = [SQLLooserLogin];
GO
```



- [ms176060]

Datenbankbenutzer löschen

Mit Hilfe des `DROP USER`-Anweisung können Benutzer aus einer Datenbank gelöscht werden. Zu Beachten ist dabei, dass Besitzer von Securables erst dann gelöscht werden können, wenn der Besitz an andere Nutzer übergeben worden ist.

Um einen Datenbankbenutzer im Management Studio zu löschen öffnen Sie das Kontextmenü des Benutzers und klicken Sie auf LÖSCHEN.



- [ms189438]

19.5.4 Das Problem der verwaisten Benutzer

Jeder Datenbankbenutzer muss einem Login zugeordnet sein, um für die Anmeldung an der SQL Server Instanz genutzt werden zu können. Als verwaist wird ein Benutzer dann bezeichnet, wenn es für ihn keine Zuordnung zu einem Login gibt. Dies kann folgende Gründe haben:

- Das zugeordnete Login wurde gelöscht
- Die Datenbank des Benutzers wurde in einer anderen Instanz wiederhergestellt.
- Die Datenbank des Benutzers wurde an eine andere Instanz angefügt.

Um zu ermitteln, ob es in einer Datenbank verwaiste Benutzerkonten gibt und welche Konten davon betroffen sind, stellt Microsoft den Administratoren die System gespeicherte Prozedur `SP_CHANGE_USERS_LOGIN` zur Verfügung.



Diese Procedure kann nicht für Benutzerkonten genutzt werden, die auf Windows-Logins basieren. Microsoft hat angekündigt, dass diese Procedure in neueren Versionen des Microsoft SQL Server nicht mehr zur Verfügung stehen wird.

Listing 19.16: Ermitteln der verwaisten Benutzerkonten

```
USE [model]
GO

sp_change_users_login @Action='Report';
GO
```

UserName	UserSID
SQLLooser	0x3CE2559A2F2C674F824B27C672ABB3F1



Um dem Benutzerkonto SQLLOOSER ein neues Login zuzuweisen sollte das `ALTER USER`-Statement genutzt werden, so wie es in Beispiel ?? zu sehen ist.

- [ms175475]
- [ms174378]



19.5.5 Privilegien auf Datenbankebene

Auf Datenbankebene existieren in SQL Server 2014 insgesamt 16 Securables, wie z. B. Schemata, Benutzer und Datenbankrollen. Auf dieses Securables kann eine Vielzahl von Privilegien angewandt werden. Im Folgenden werden die Wichtigsten davon aufgezählt und kurz erläutert:

- **control:** Dies ist das umfangreichste Privileg, das einem Benutzer für ein Securable verliehen werden kann. Es schließt alle anderen Privilegen auf Datenbankebene ein.
- **view definition:** Erlaubt dem Benutzer die Metadaten der betroffenen Objekte einsehen zu können, ohne jedoch direkten Zugriff auf das Securable zu haben.
- **alter:** Dieses Privileg erlaubt es dem Benutzer ein Objekt zu verändern. Welcher Art diese Veränderung ist hängt dabei vom Objekt selbst ab. Einer Rolle können beispielsweise Mitglieder hinzugefügt werden, während bei einem Schema der Besitzer geändert werden kann.

- `take ownership`: Erlaubt dem Benutzer den Besitz für ein Objekt zu übernehmen.
- `impersonate`: Dieses Privileg kann auf Logins (Serverebene) und Benutzer (Datenbankebene) erteilt werden und ermöglicht es, die Identität eines anderen Logins/Benutzers anzunehmen.

Zusammenhänge zwischen Securable- und Datenbankebene

Die in SQL Server implementierte Berechtigungshierarchie lässt es zu, dass einige Privilegien, die direkt auf eine Datenbank erteilt werden, sich auch auf deren Securables auswirken. Beispiele hierfür sind die Privilegien `alter any schema`, `alter any role` und `alter any user`. Hat ein Benutzer das Privileg `alter any role`, dann besitzt er implizit das `alter`-Privileg für alle Rollen, die in der Datenbank existieren bzw. zukünftig existieren werden.

Die folgende Tabelle zeigt ausschnittsweise, welche datenbankseitigen Privilegien sich implizit auf Securables auswirken.

Übergeordnetes Privileg	Basisprivileg	Securable
<code>control</code>	<code>control</code>	Zertifikat
<code>control</code>	<code>control</code>	Asymmetrischer Schlüssel
<code>control</code>	<code>control</code>	Schema
<code>take ownership</code>	<code>control</code>	Asymmetrischer Schlüssel
<code>alter any certificate</code>	<code>alter</code>	Zertifikat
<code>alter any role</code>	<code>alter</code>	Datenbankrolle
<code>alter any schema</code>	<code>alter</code>	Schema

Die Tabelle ?? ist ein Ausschnitt aus der Tabelle „SQL Server and SQL Database Permissions“ aus dem Artikel „Permissions (Database Engine)“ ([ms191291]).

Zusammenhänge zwischen Datenbank- und Serverebene

Ähnliche Zusammenhänge, wie sie gerade beschrieben wurden, existieren auch zwischen Server und Datenbankebene.

Übergeordnetes Privileg	Basisprivileg	Securable
<code>control server</code>	<code>alter any asymmetric key</code>	Datenbank
<code>control server</code>	<code>alter any role</code>	Datenbank
<code>control server</code>	<code>alter any schema</code>	Datenbank
<code>alter any database</code>	<code>alter</code>	Datenbank

alter any certificate	alter	Zertifikat
view any definition	view definition	Datenbank

Die Tabelle ?? ist ein Ausschnitt aus der Tabelle „SQL Server and SQL Database Permissions“ aus dem Artikel „Permissions (Database Engine)“ ([[ms191291](#)]).

- [[ms190401](#)]
- [[ms175536](#)]
- [[ms191291](#)]



Privilegien auf Datenbankebene erteilen/entziehen und verweigern

Das Erteilen, Entziehen und Verweigern von Privilegien auf Datenbankebene entspricht dem Erteilen, Entziehen und Verweigern von Privilegie auf Serverebene.

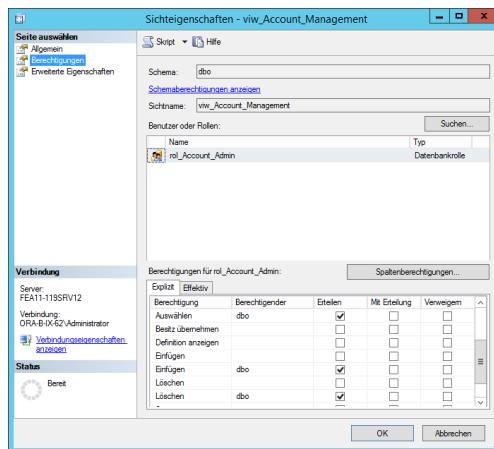


Abb. 19.36:
Privilegien auf
Datenbankebene
verwalten

Listing 19.17: Privilegien auf Datenbankebene erteilen, entziehen und verweigern

```
USE [Bank]
GO

GRANT create table
TO bob;

GRANT alter any role
TO bob;

GRANT create view
TO bob
WITH GRANT OPTION;

REVOKE create table
```

```

FROM bob;

REVOKE GRANT OPTION FOR control ON USER::[Bank]
FROM [bob] CASCADE
GO

GRANT control ON USER::[Bank]
TO [bob]
GO

DENY impersonate ON USER::[Bank]
TO [bob]
GO

```

Das Kommando `GRANT CREATE VIEW TO bob WITH GRANT OPTION;` erteilt dem Benutzer BOB das `create view` und gleichzeitig die Möglichkeit dieses an andere Benutzer weiterzugeben. Dies ist der gleiche Mechanismus wie auch auf Serverebene.

Um auf Datenbankebene nur die `GRANT OPTION` zu entziehen, muss hier zusätzlich das Schlüsselwort `CASCADE` angegeben werden. So werden automatisch alle Privilegien entzogen, die der Benutzer aufgrund des Besitzes der `GRANT OPTION` erteilt hat.

Die Kombination der beiden letzten Kommandos zeigt wie ein Benutzer das `control`-Privileg erhalten und trotzdem nicht die Identität des betroffenen Benutzerkontos übernehmen kann. Dies funktioniert, in dem ihm das `control`-Privileg erteilt und im Gegenzug das `impersonate`-Privileg verweigert wird.



- [ms188396]
- [ms187728]

19.5.6 Rollen

Die Datenbankrolle „db_owner“

Die Datenbankrolle `db_owner` gehört zu den Festen Datenbankrollen, die in jeder Datenbank existieren. Sie räumt ihren Mitgliedern alle Berechtigungen ein, die zur Administration einer Datenbank notwendig sind. Diese Rolle sollte nicht mit dem Datenbankbenutzer `dbo` verwechselt werden. Es existieren zwei wesentliche Unterschiede zwischen der Rolle `db_owner` und dem Benutzer `dbo`:

- Mitgliedern der Rolle db_owner können Priviliegien explizit verweigert werden, dem Benutzer dbo nicht, da für ihn keine Berechtigungsprüfung erfolgt.
- Die Auflistung der Mitglieder der Rolle db_owner wird in jeder Datenbank separat gespeichert. Ist die Datenbank beschädigt, kann nicht mehr festgestellt werden, wer ein Mitglied dieser Rolle ist. Somit können Mitglieder der Rolle db_owner nur unbeschädigte Datenbanken wiederherstellen. Der Benutzer dbo kann auch beschädigte Datenbanken wiederherstellen.



- [gc20110407msdvdo]

Die Datenbankrolle „public“

Die Datenbankrolle public ist mit der Windows-Benutzergruppe Jeder vergleichbar. Alle aktiven Benutzer innerhalb einer Datenbank sind automatisch Mitglied dieser Rolle.



Die Datenbankrolle public kann nicht gelöscht werden. Da alle aktiven Benutzer automatisch Mitglied dieser Rolle werden, sollte sie nur minimale Privilegien haben.

Weitere Feste Datenbankrollen

Neben den beiden Rollen db_owner und public existieren in SQL Server 2014 noch acht weitere Feste Datenbankrollen. Diese sind:

- db_accessadmin: Erlaubt den Zugriff auf eine Datenbank zu regeln. Mitglieder können Logins den Benutzerkonten in der Datenbank zuordnen.
- db_backupoperator: Mitglieder können eine Sicherung der Datenbank anfertigen.
- db_datareader: Erteilt das Recht, alle Tabellen in der Datenbank zu lesen.
- db_datawriter: Erteilt das Recht, in alle Tabellen der Datenbank zu schreiben.
- db_ddladmin: Mitglieder können alle DDL-Befehle innerhalb einer Datenbank ausführen.

- db_denydatareader: Verweigert das Recht Tabellen in der Datenbank zu lesen. Mitglieder können keinerlei Daten abfragen.
- db_denydatawriter: Verweigert das Recht DML-Befehle auf die Tabellen einer Datenbank anzuwenden. Mitglieder können keine Daten ändern.
- db_securityadmin: Ermöglicht das Ändern von Rollenmitgliedschaften und die Verwaltung von Privilegien.



- [ms189121]
- [ms189612]

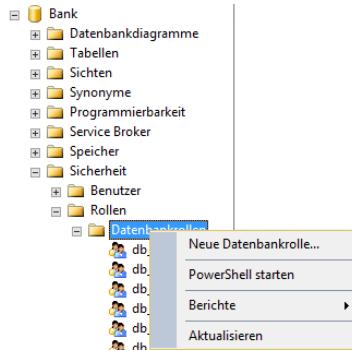
Besondere Rollen der MSDB-Datenbank

Die Datenbank MSDB enthält einige spezielle Feste Datenbankrollen die in der MSDB unter [ms189121] aufgeführt sind.

Eigene Datenbankrollen erstellen

1. Erweitern Sie im Objekt-Explorer die Datenbank, innerhalb derer die neue Rolle erstellt werden soll.
2. Wählen Sie das Register „Sicherheit“, „Rollen“ und darin „Datenbankrollen“.

Abb. 19.37:
Erstellen einer
Datenbankrolle



3. Öffnen Sie das Kontextmenü des Registers „Datenbankrollen“ und klicken Sie auf „Neue Datenbankrolle“. Es erscheint der Dialog „Datenbankrolle - Neu“

4. Tragen Sie im Feld „Rollenname“ den Bezeichner der Rolle ein.

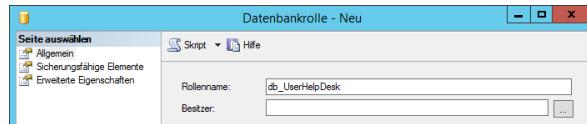


Abb. 19.38:
Den
Rollenbezeichner
eingeben

5. Wählen Sie die Seite „Sicherungsfähige Elemente“ aus.

6. Klicken Sie auf die Schaltfläche „Suchen“ und wählen Sie die Securables aus, für die Sie Privilegien festlegen wollen.

7. Erteilen Sie in der unteren Fensterhälfte die benötigten Privilegien.

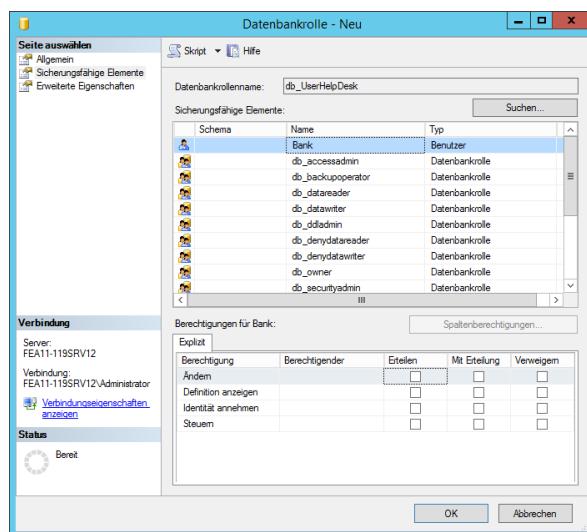


Abb. 19.39:
Auswählen der
Securables

8. Klicken Sie auf „OK“.

Listing 19.18: Erstellen einer Datenbankrolle

```
USE [Bank]
GO

CREATE ROLE userhelpdesk AUTHORIZATION dbo;
GO
```

- [ms187936]

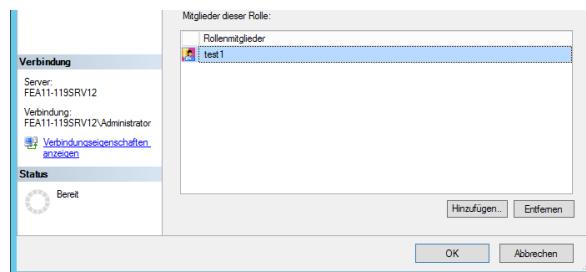


Rollenmitglieder verwalten

Das Hinzufügen von Mitgliedern zu einer Datenbankrolle kann mittels des Managementstudios oder mit Hilfe von T-SQL geschehen. Wenn Sie das Management Studio benutzen möchten, gehen Sie wie folgt vor:

1. Erweitern Sie im Objekt-Explorer die Datenbank, innerhalb derer die neue Rolle erstellt werden soll.
2. Wählen Sie das Register „Sicherheit“, „Rollen“ und darin „Datenbankrollen“.
3. Öffnen Sie das Kontextmenü der Datenbankrolle und klicken Sie auf „Eigenschaften“. Es erscheint der Dialog „Eigenschaften der Datenbankrolle“.
4. Im unteren Bereich dieses Dialogs können Benutzer oder andere Rollen als Mitglieder hinzugefügt oder entfernt werden.

Abb. 19.40:
Mitglieder zu
einer Rolle
hinzufügen



Die beiden folgenden T-SQL Statements zeigen, wie einer Rolle Mitglieder hinzugefügt bzw. entfernt werden können.

Listing 19.19: Ändern einer Datenbankrolle

```
USE [Bank]
GO

ALTER ROLE userhelpdesk
ADD MEMBER weidinger;
GO

ALTER ROLE userhelpdesk
DROP MEMBER weidinger;
GO
```

Datenbankrollen umbenennen

Soll eine Datenbankrolle umbenannt werden, muss das T-SQL Kommando `ALTER ROLE` mit der Klausel `WITH NAME` benutzt werden.

Listing 19.20: Umbenennen einer Datenbankrolle

```
USE [Bank]
GO

ALTER ROLE [db_UserHelpDesk]
WITH NAME = [db_HelpDesk]
GO
```

Wahlweise kann eine Rolle auch im Objekt-Explorer umbenannt werden.



Feste Datenbankrollen können nicht umbenannt werden!

- [ms189775]

Selbsterstellte Datenbankrollen löschen

Damit eine Datenbankrolle gelöscht werden kann, müssen zuerst alle Mitglieder und alle Securables aus der Rolle entfernt werden.

1. Öffnen Sie im Objekt-Explorer das Kontextmenü der Datenbankrolle und wählen Sie den Menüpunkt „Löschen“.
2. Klicken Sie auf „OK“.

Listing 19.21: Entfernen einer Datebankrolle

```
DROP ROLE [db_UserHelpDesk];
GO
```

- [ms174988]

19.6 Datenbankschemata

19.6.1 Securables

Innerhalb eines Schemas können sich die folgenden Securables befinden:

- Type
- XML-Schemaauflistung
- Aggregat (Objekt)
- Funktion
- Verfahren
- Queue
- Synonym
- Tabelle
- Sicht



- [ms190401]

Privilegien auf Schemaebene

Auf Schemaebene existieren in SQL Server 2014 insgesamt 12 Objektprivilegien, wie z. B. `select`, `insert`, `update` und `delete`. Diese steuern den direkten Zugriff auf die Datensätze innerhalb des Schemas bzw. der Datenbank.

Zugewiesen, entzogen und verweigert werden Objektprivilegien mit den SQL-Kommandos `grant`, `revoke` und `deny`.

Listing 19.22: Erteilen von Objektprivilegien

```
USE [Bank]
GO
```

```

GRANT select ON viw_eigenkunden_kontaktdaten
TO bob;
GO

GRANT select ON OBJECT::viw_eigenkunden_kontaktdaten
TO bob;
GO

GRANT select ON viw_eigenkunden_adressen
TO bob
WITH GRANT OPTION;
GO

GRANT insert, update, delete ON eigenkunden
TO bob;
GO

```

Der Entzug und das Verweigern von Objektprivilegien erfolgt parallel zum Entzug und zur Verweigerung von Privilegien auf Datenbankebene.

- [ms188371]
- [ms187965]



Ownership chaining - Besitzverkettungen

Wenn in einer SQL Server Datenbank mehrere Objekte ein und des selben Besitzers/Schemas sequentiell von einander abhängig sind, spricht man von einer Besitzverkettung bzw. engl. Ownership chain. Beispiel ?? zeigt eine Besitzverkettung, zwei Views die sequentiell auf den Tabellen KUNDE und EIGENKUNDE aufbauen.

Listing 19.23: Eine Besitzverkettung

```

USE [Bank]
GO

CREATE VIEW dbo.viw_eigenkunden_adressen
AS
    SELECT vorname, nachname, strasse, hausnummer, plz, ort,
           TelefonNr, EmailAdresse
    FROM eigenkunde ek INNER JOIN Kunde k
        ON (k.kunden_id = ek.kunden_id);
GO

CREATE VIEW dbo.viw_eigenkunden_kontaktdaten

```

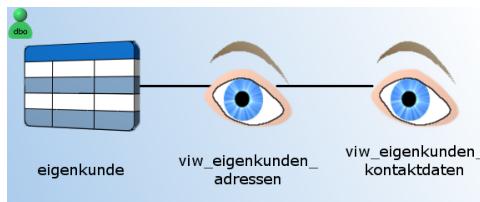
```

AS
SELECT TelefonNr , EmailAdresse
FROM eigenkunde;
GO

```

Die beiden Views aus dem vorangegangenen Beispiel werden beide im Schema DBO (Besitzer ist der Datenbankbenutzer DBO) erstellt, in dem auch die Tabelle EIGENKUNDE liegt. Dadurch haben alles drei Objekte den gleichen Besitzer und eine Besitzverkettung ist entstanden.

Abb. 19.41:
Eine
Besitzverkettung
dreier Objekte



Dem Datenbankbenutzer BOB wird nun in Beispiel ?? das select-Privileg auf die View VIW_EIGENKUNDEN_KONTAKTDATEN eingeräumt.

Listing 19.24: Bob erhält Leserechte

```

USE [Bank]
GO

GRANT select ON viw\_eigenkunden\_kontaktdaten
TO bob;
GO

```

Sobald der Benutzer BOB nun die View VIW_EIGENKUNDEN_KONTAKTDATEN abfragt, wird er die entsprechenden Datensätze aus der Tabelle EIGENKUNDE zurückgeliefert bekommen.

SQL Server prüft im Falle einer Besitzverkettung die Zugriffsrechte wie folgt:

1. Wer ist der Besitzer des Objekts (VIW_EIGENKUNDEN_KONTAKTDATEN), auf das Zugriff genommen wird? Antwort: DBO.
2. Ist DBO auch der Besitzer der weiteren Objekte (VIW_EIGENKUNDEN_ADRESSEN, EIGENKUNDE)? Antwort: ja.
3. Hat der Benutzer BOB die notwendigen Privilegien, um in gewünschter Art und Weise auf VIW_EIGENKUNDEN_KONTAKTDATEN zuzugreifen? Antwort: ja.

Da BOB das `select`-Privileg auf die View `VIW_EIGENKUNDEN_KONTAKTDATEN` hat, welche auf die View `VIW_EIGENKUNDEN_ADRESSEN` und die Tabelle `EIGENKUNDE` aufbaut bekommt er Zugriff auf die Datensätze der Tabelle, ohne das weitere Zugriffsrechte geprüft werden würden.



Der Zugriff auf ein Objekt mittels einer Besitzverkettung ist performanter, da nur für das Objekt, auf welches unmittelbar Zugriff genommen wird die Zugriffsrechte geprüft werden.

Unterbrochene Besitzverkettungen

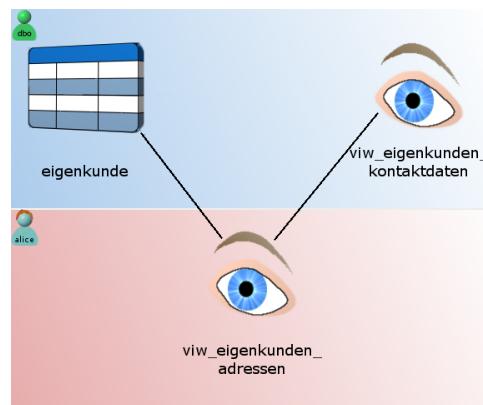


Abb. 19.42:
Eine
unterbrochene
Besitzverkettung

Abbildung ?? zeigt ein Beispiel für eine unterbrochene Besitzverkettung. Weil die View `VIW_EIGENKUNDEN_ADRESSEN` einen anderen Besitzer (ALICE) hat, als die zweite View, `VIW_EIGENKUNDEN_KONTAKTDATEN`, wird der Zugriffsmechanismus nicht mehr funktionieren. SQL Server prüft hier wie folgt:

1. Wer ist der Besitzer des Objekts (`VIW_EIGENKUNDEN_KONTAKTDATEN`), auf das Zugriff genommen wird? Antwort: DBO.
2. Ist DBO auch der Besitzer der weiteren Objekte (`VIW_EIGENKUNDEN_ADRESSEN`, `EIGENKUNDE`)?
Antwort: **nein**.
3. Hat der Benutzer BOB die notwendigen Rechte, um in gewünschter Weise auf die View `VIW_EIGENKUNDEN_KONTAKTDATEN` zugreifen zu können? Antwort: ja.
4. Hat der Benutzer BOB die notwendigen Rechte, um in gewünschter Weise auf die View `VIW_EIGENKUNDEN_ADRESSEN` zugreifen zu können? Antwort: **nein**.
5. Auswertung: Der gewünschte Zugriff kann nicht erfolgen.



- [ms188676]
- [bb669084]

Seiteneffekte der Besitzverkettung

Das größte Problem der Besitzverkettung ist, dass dieser Mechanismus die Überprüfung der Zugriffsrechte innerhalb der Besitzkette komplett umgeht.

Abb. 19.43:
Seiteneffekte
einer
Besitzverkettung

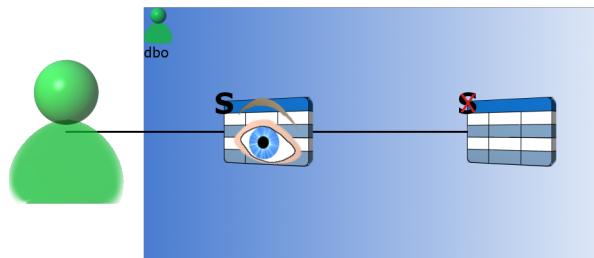


Abbildung ?? zeigt eine Situation in der der Benutzer BOB das select-Privileg auf einer View besitzt, wobei im gleichzeitig das select-Privileg auf die zugrundeliegende Tabelle explizit verweigert wurde. Beispiel ?? zeigt den SQL-Code zu dieser Situation.

Listing 19.25: Seiteneffekte der Besitzverkettung

```
USE [Bank]
GO

GRANT select ON viw_eigenkunden_kontaktdaten
TO bob;

DENY select ON Eigenkunde
TO bob;
GO
```

Das Problem in diesem Falle ist, dass BOB trotz des expliziten DENY die Datensätze der Tabelle EIGENKUNDE abfragen können. SQL Server verfährt hier wie folgt:

1. Wer ist der Besitzer des Objekts (VIW_EIGENKUNDEN_KONTAKTDATEN), auf das Zugriff genommen wird? Antwort: DBO.
2. Ist DBO auch der Besitzer der Tabelle EIGENKUNDE? Antwort: ja.
3. Hat der Benutzer BOB die notwendigen Rechte, um in gewünschter Weise auf die View VIW_EIGENKUNDEN_KONTAKTDATEN zugreifen zu können? Antwort: ja.

4. Auswertung: Der gewünschte Zugriff kann erfolgen.

Weil BOB nicht direkt auf die Tabelle EIGENKUNDE zugreift, sondern nur indirekt mit Hilfe der View VIW_EIGENKUNDEN_KONTAKTDATEN, für die er die notwendige select-Berechtigung hat, wird das deny nicht weiter geprüft und auch nicht berücksichtigt.

Obwohl ein deny eine entsprechende Aktion definitiv unmöglich machen soll, kann dies in diesem Fall nicht durchgesetzt werden. Der Berechtigungsprüfungsmechanismus des SQL Server wird einfach „ausgehebelt“.

19.6.2 Erstellen eines Rollen- und Rechtekonzeptes

Eine rollenbasierte Rechtezuteilung ist ein intelligentes und flexibles Hilfsmittel für den Administrator, zur effizienten Verwaltung einer großen Anzahl von Benutzern, mit dessen Hilfe diesen die benötigten Berechtigungen erteilt werden können. Rollen- und Rechtekonzepte eignen sich besonders als Autorisierungsverfahren für Systeme mit einer hohen Volatilität². Nicht oder nur bedingt geeignet sind Rollen- und Rechtebasierte Konzepte für Systeme mit wenigen Benutzern, da hier der Nutzen schnell durch den zusätzlichen Aufwand nicht mehr zu rechtfertigen ist.

Vorteile eines Rollen- und Rechtekonzeptes

- Viele Einzelberechtigungen werden als „Tätigkeiten“ in Rollen zusammengefasst. Beispielsweise können Tätigkeiten wie ein Login oder das Verwalten von Benutzern über solche Rollen geregelt werden.
- Die Komplexität des Autorisierungssystems sinkt, da wenig bis gar keine Einzelberechtigungen erteilt und verwaltet werden müssen.
- Neue Benutzer können sehr einfach in das System integriert werden.

Vorteile eines Rollen- und Rechtekonzeptes

- Hoher Planungsaufwand, um ein konsistentes rollenbasiertes Autorisierungssystem zu erstellen.
- Erfordert gut ausgebildetes Administrationspersonal

²lat. volatilis = fliegend, flüchtig, hier die Änderungshäufigkeit

- Eventuell vorhandene Überschneidungen müssen geregelt werden (eine Person benötigt die Rechte aus Abteilung A und nur einen Teil der Rechte für Abteilung B)

Beispiel für ein Rollen- und Rechtekonzept in SQL Server

Abb. 19.44:
Beispiel eines
Rollenkonzepts

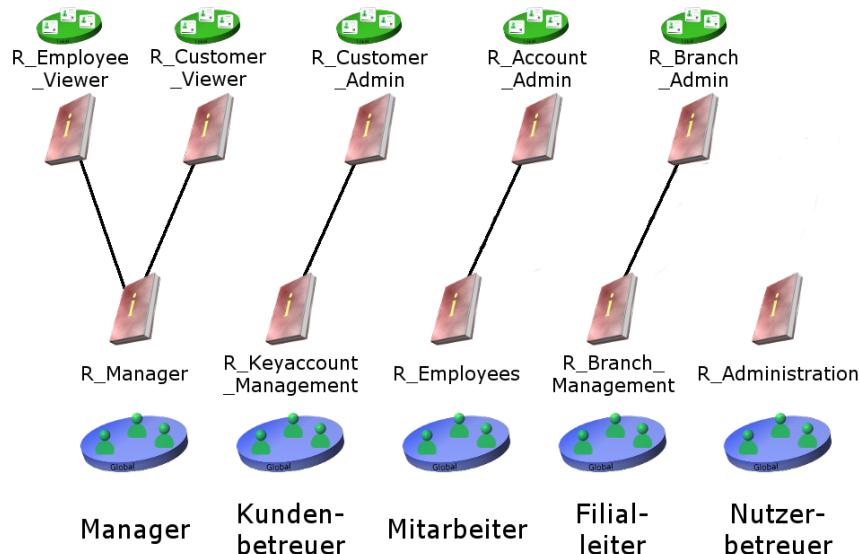


Abbildung ?? zeigt ein zweistufiges Rollen- und Rechtekonzept. Auf der unteren Ebene sind dabei die sogenannten „Personenrollen“ zu sehen. Diese Rollen können wie Benutzergruppen betrachtet werden. In einer Personenrolle werden alle Benutzerkonten zusammengefasst, die ein gemeinsames Interesse bzw. eine gemeinsame Aufgabe haben. In der Beispielabbildung sind das z. B. Filialleiter, Nutzerbetreuer oder Kundenberater.

Die obere Ebene zeigt die „Funktionsrollen“. Diese Rollen dienen zur Gruppierung von Berechtigungen, welche dann eine bestimmte Tätigkeit ermöglichen. Die Rolle R_LOGIN beispielsweise beinhaltet das connect_sql-Privileg, das benötigt wird, um sich mit dem Datenbankserver zu verbinden. Die folgende Aufzählung gibt einen detaillierten Überblick über das hier vorgestellte Konzept. Die darin benutzten Abkürzungen sind: **Select**, **Insert**, **Update** und **Delete**. Sie stellen jeweils die Privilegien dar, die für ein bestimmtes Objekt zugewiesen werden. Die Präfixes an den Objektnamen haben folgende Bedeutung: **rol:** Rolle, **viw:** View, **tab:** Tabelle, **udf:** User Defined Function (Benutzerdefinierte Funktion) und **trg:** Trigger.

Funktionsrollen:

- ROL_EMPLOYEE_VIEWER
 - S VIW_EMPLOYEES

- ROL_CUSTOMER_VIEWER
 - S VIW_CUSTOMERS
- ROL_CUSTOMER_ADMIN
 - SIUD VIW_CUSTOMERS
 - SU VIW_ACCOUNTS
- ROL_ACCOUNT_ADMIN
 - SIUD VIW_ACCOUNT_MANAGEMENT
- ROL_BRANCH_ADMIN
 - S UDF_EMPLOYEES_AT_BRANCH
 - S UDF_CUSTOMERS_AT_BRANCH
- Serverrolle ROL_LOGIN
 - connect sql

Personenrollen:

- ROL_MANAGEMENT: Beinhaltet alle Manager der Bank (Mitarbeiter_IDs 1 bis 5)
- ROL_KEYACCOUNT_MANAGEMENT: Alle Mitarbeiter die als Kundenberater arbeiten
- ROL_EMPLOYEES: Alle Mitarbeiter die nicht in einer beratenden Funktion tätig sind
- ROL_BRANCH_MANAGEMENT: Die Gruppe der Filialleiter (Mitarbeiter_IDs 8 bis 27)
- ROL_ADMINISTRATION: Die Nutzerbetreuer der Organisation (Mitarbeiter_IDs 6 und 7)

Die Zusammenhänge zwischen den benutzten Views und Funktionen stellen sich wie folgt dar:

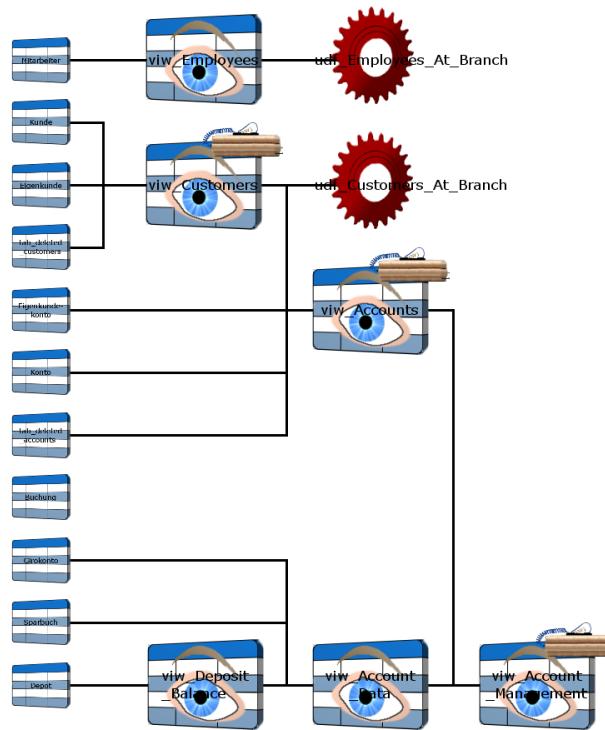


Abb. 19.45:
Zusammenhang
der Objekte

Die drei Views VIW_CUSTOMERS, VIW_ACCOUNTS und VIW_ACCOUNT_MANAGEMENT werden durch Trigger überwacht. Diese haben die Aufgabe DML-Operationen auf die Views zu ermöglichen, da dies anderen Falls nicht möglich wäre.

Mit Hilfe der beiden benutzerdefinierten Funktionen UDF_EMPLOYEES_AT_BRANCH und UDF_CUSTOMERS_AT_BRANCH ist es der Gruppe der Filialleiter möglich, auf die Angestellten ihrer Filialen bzw. auf die Kunden, welche in ihren Filialen betreut werden, zu zugreifen.

Listing 19.26: Selektieren aus einer User Defined Table-function

```
USE [Bank]
GO

-- Select all employees from branch number 10
SELECT *
FROM    udf_Employees_At_Branch(10);

-- Select all customers coached in branch number 10
SELECT *
FROM    udf_Customers_At_Branch(10);
```

19.7 Datenbankverschlüsselung

Die SQL Server-Verschlüsselung ist ein Mechanismus der es ermöglicht, sensible Daten, mit Hilfe eines Schlüssels oder eines Kennwortes, durch einen zusätzlichen Zugriffsschutz gegen Datenverlust zu sichern. Gesetzt den Fall, dass die Zugriffsschutzbarrieren der Datenbank umgangen werden und ein „Angreifer“ Daten entwendet, kann die Verschlüsselung eine probate Möglichkeit darstellen, die Beute für den Dieb unbrauchbar zu machen.



Man muss sich bewußt sein, dass jede Verschlüsselung ein „Hindernis auf Zeit“ ist.

19.7.1 Aufbau der Transparenten Datenbankverschlüsselung (TDE)

Seit der Version 2005 stellt Microsoft SQL Server eine hierarchische Infrastruktur zur Verschlüsselung von Daten und zur Verwaltung von Schlüsseln bereit. Dabei kommen Passwörter, symmetrische und asymmetrische Schlüssel, sowie Zertifikate zum Einsatz.

Die Basis für die SQL Server-Verschlüsselungshierarchie ist die „Windows Operating System level Data Protection API“, kurz „DPAPI“. Diese wird dazu benutzt den „Diensthauptschlüssel“ / „Service Master Key“, kurz „SMK“ zu schützen.



SQL Server 2012 benutzt den AES-Algorithmus zur Verschlüsselung des SMK.

Abbildung ?? - Architektur der transparenten Datenbankverschlüsselung (TDE) - Quelle: [bb934049]

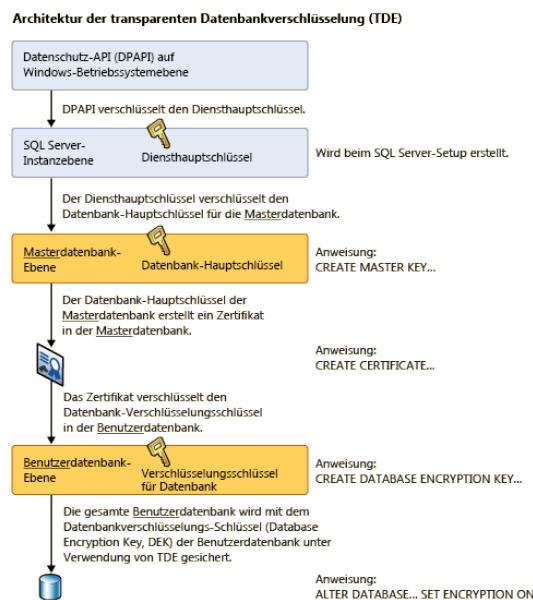
Der Service Master Key

Beim SMK handelt es sich um einen symmetrische Schlüssel der beim Setup des SQL Servers aus

- den Anmeldeinformationen des SQL Server-Dienstkontos und
- den Anmeldeinformationen des Computers, auf dem der SQL Server installiert wird

erstellt wird. Verschlüsselt wird er mit dem Computerschlüssel des Computers, auf dem SQL Server installiert wird. Es existiert immer nur ein SMK pro SQL Server.

Abb. 19.46:
Architektur der transparenten Datenbankverschlüsselung (TDE)



Der SMK dient zur Verschlüsselung von folgenden Bestandteilen:

- Datenbankhauptschlüssel/„Database Master Key“ (DMK),
- Kennwörtern für Verbindungsserver,
- Anmeldeinformationen

Entschlüssel werden kann der SMK nur von dem Dienstkonto, unter dem er erstellt wurde oder von einem Prinzipal, der auf die Anmeldeinformationen des Computers zugreifen kann. Damit es bei einem Wechsel des SQL Server-Dienstkontos nicht zu Problemen kommt, sollte immer der SQL Konfigurations Manager genutzt werden. Dieser entschlüsselt und verschlüsselt den SMK automatisch.

Muss der Computer, auf dem Microsoft SQL Server installiert ist, neu erstellt werden, sollte der Domänenbenutzer, der zuvor als Dienstkonto genutzt wurde auch weiterhin genutzt werden. Dieser kann ohne Problem den SMK wiederherstellen. Soll ein anderer Domänenbenutzer als Dienstkonto für den SQL Server verwendet werden, muss der SMK mittels Backup and Recovery wiederhergestellt werden.

Soll SQL Server auf einen anderen Computer verschoben werden, bleibt nur die Möglichkeit, den SMK mittels Backup and Recovery wiederherzustellen.



Der SMK ist der Stamm der Verschlüsselungshierarchie. Sollte er verloren gehen, sind die weiteren Bestandteile der Hierarchie unbrauchbar.



- [ms189060]

Der Database Master key (DMK)

Der Database Master Key bzw. der Datenbankhauptschlüssel ist ein symmetrischer Schlüssel, der in der MASTER-Datenbank gespeichert wird. Für jede Datenbank innerhalb einer SQL Server-Instanz wird ein eigener, eindeutiger DMK erzeugt, dessen Aufgabe es ist, private Schlüssel von Zertifikaten oder asymmetrischen Schlüsseln zu sichern. Dies geschieht, in dem mit Hilfe des DMK ein Serverzertifikat innerhalb der MASTER-Datenbank erzeugt wird. Dieses Zertifikat verschlüsselt dann den „Datenbankverschlüsselungsschlüssel“, der in jeder Datenbank angelegt werden kann.



- [bb964742]

Das Serverzertifikat

Zertifikate sind digital signierte Securables, die einen öffentlichen und optional auch einen privaten Schlüssel enthalten. In SQL Server werden Zertifikate mit dem Standard IETF X.509v3 erstellt. Sie können verwendet werden, um Objekte zu signieren oder zu verschlüsseln. Im konkreten Fall des Serverzertifikates wird es dazu benutzt, um den Database encryption key einer Datenbank zu generieren.

Die privaten Schlüssel, welche mit SQL Server zu einem Zertifikat generiert werden sind 1024 Bits lang. Werden private Schlüssel aus einer externen Quelle importiert, haben diese eine Länge zwischen 384 und 4096 Bits. Damit ein Zertifikat in Zusammenhang mit TDE genutzt werden kann, muss der private Schlüssel eine Länge von genau 3456 Bits aufweisen.



Der private Schlüssel muss dem öffentlichen Schlüssel entsprechen, der mit dem Zertifikat zusammenhängt.



- [bb895327]

Der Database encryption key (DEK)

Der Datenbankverschlüsselungs Schlüssel bzw. Database encryption key ist ein symmetrischer Schlüssel, der für die Verschlüsselung der Daten innerhalb einer Datenbank zuständig ist. Er wird direkt in der Datenbank gespeichert, für die er erzeugt wurde.

19.7.2 Einrichten der Transparenten Datenbankverschlüsselung

Erstellen eines Datenbankhauptschlüssels (DMK)

Listing 19.27: Erstellen eines Datenbankhauptschlüssels

```
USE [Bank]
GO

CREATE MASTER KEY
ENCRYPTION BY PASSWORD = '2S0d0V5sMehI0#DVe4x9i8z6';
GO
```



- [aa337551]
- [ms174382]

Erstelle des benötigten Serverzertifikates

Listing 19.28: Erstellen des Serverzertifikates

```
USE [Bank]
GO

CREATE CERTIFICATE dek_server_certificate
WITH SUBJECT = 'DEK Server certificate';
```

Alternativ kann das Serverzertifikat auch aus einer Zertifikatsdatei importiert werden.

Listing 19.29: Erstellen des Serverzertifikates aus einer Datei

```
USE [Bank]
GO

CREATE CERTIFICATE dek_server_certificate
FROM FILE = 'D:\certs\dek_server_certificate.cer',
WITH PRIVATE KEY (FILE = 'D:\certs\dek_server_certificate_key.pk',
DECRYPTION BY PASSWORD = '2S0d0V5sMehI0#DVe4x9i8z6');
GO
```



Der private Schlüssel eines Zertifikates wird standardmäßig mit dem Datenbankhauptschlüssel verschlüsselt. Sollte kein DMK vorhanden sein muss ein Passwort angegeben werden, da dies anderen Falls zu einem Fehler führt.



- [ms187798]

Erzeugen des Datenbankverschlüsselungs Schlüssels (DEK)

Listing 19.30: Erstellen des Datenbankverschlüsselungs Schlüssels

```
USE [Bank]
GO

CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE dek_server_certificate;
GO
```



- [bb677241]

Aktivieren der Verschlüsselung einer Datenbank

Listing 19.31: Aktivieren der Verschlüsselung

```
USE [Bank]
GO

ALTER DATABASE Bank
SET ENCRYPTION ON;
GO
```



- [bb934049]

19.8 Verschlüsselung der Kommunikation - SSL

Der Secure Socket Layer (SSL) stellt eine einfache Möglichkeit zur Verschlüsselung des Datenverkehrs auf Protokollebene dar. Der SQL Server bietet diese Art der Verschlüsselung seit der Version 2005 an. Ob SSL genutzt werden kann oder nicht hängt aber im Wesentlichen von der genutzten Clientsoftware ab. Beispielsweise können die „Microsoft Data Access Components“ (MDAC) SSL erst ab der Version 2.6 SSL benutzen.

Ein zweites, durch den Secure Socket Layer mitgeliefertes Feature ist die Identitätskontrolle für den Datenbankserver. Diese funktioniert jedoch nur, wenn kein selbstsigniertes Zertifikat für SSL verwendet wird.



Die zur Verschlüsselung verwendete Schlüsselbreite, 40 oder 128 Bit, hängt vom genutzten Betriebssystem ab.

19.8.1 Vorbereitende Maßnahmen



Für die Nutzung von SSL sollte in jedem Falle ein eigenes Zertifikat erstellt werden. Steht des SQL Server kein Zertifikat zur Verfügung, nutzt er ein selbstsigniertes Zertifikat. Dies verringert die Sicherheit von SSL massiv.

Erstellen eines Certificate Signing Request

Der erste Schritt der bei der Erstellung eines Zertifikats getan werden muss, ist die erstellung einer Zertifikatsanforderung, auch „Certificate Signing Request“ (CSR) genannt. Dieser wird üblicherweise auf dem Rechner erstellt, auf dem auch das Zertifikat genutzt werden soll. In einem verschlüsselten Textblock enthält er Informationen, die später im Zertifikat vorkommen. Der CSR wird dann von einer Zertifizierungstelle bzw. Certificate Authority signiert. Aus dem signierten CSR entsteht dann das Zertifikat.



- [sslshopwiaccsr]

1. Starten Sie die Microsoft Management Console (MMC).

2. Wählen Sie im Menü „Datei“ den Eintrag „snap-In hinzufügen/entfernen“.
3. Wählen Sie aus der Liste „Verfügbare Snap-Ins“ das Snap-In „Zertifikate“ mit einem Doppelklick aus.

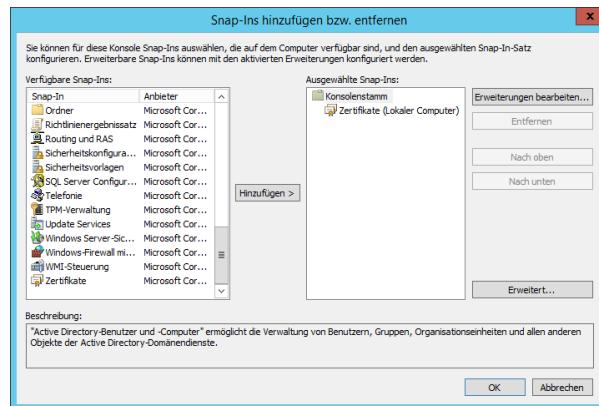


Abb. 19.47:
Das Zertifikats-Snap-In benutzen

4. Wählen Sie im neu erschienenen Dialogfenster die Option „Computerkonto“ und klicken Sie auf „Weiter“.

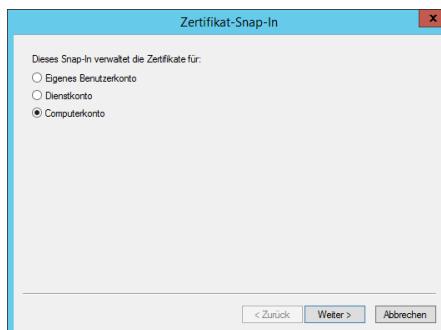


Abb. 19.48:
Das Zertifikats-Snap-In benutzen

5. Wählen Sie im Dialog „Computer auswählen“ die Option „Lokalen Computer (Computer, auf dem diese Konsole ausgeführt wird)“ und klicken Sie auf Fertigstellen.

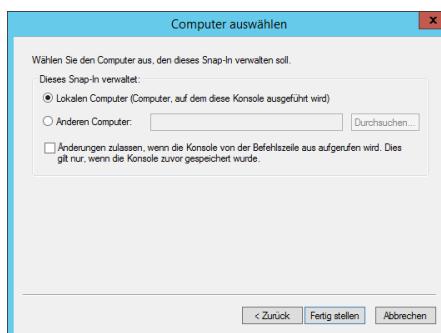


Abb. 19.49:
Das Zertifikats-Snap-In benutzen

6. Klicken Sie links außen den Eintrag „Eigene Zertifikate“ an und öffnen Sie das Kontextmenü.

7. Wählen Sie „Alle Aufgaben“ → „Erweiterte Vorgänge“ → „Benutzerdefinierte Anforderung erstellen“:

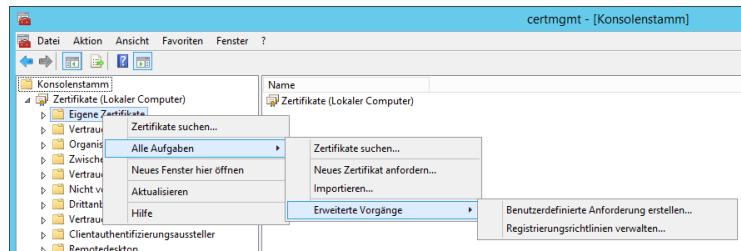
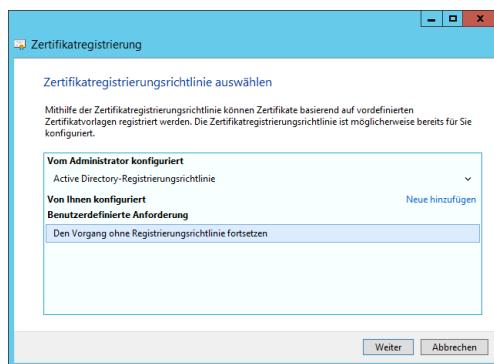


Abb. 19.50:
Einen benutzerdefinierten Request erstellen

8. Klicken Sie im Vorbereitungs-Dialog auf „Weiter“.
9. Wählen Sie im nächsten Schritt den Anforderungstyp: „Den Vorgang ohne Registrierungsrichtlinie fortsetzen“

Abb. 19.51:
Zertifikatsregistrierungsrichtlinie wählen



10. Klicken Sie auf „Weiter“.

- Wählen Sie als Vorlagetyp für die benutzerdefinierte Anforderung: „(Keine Vorlage) Legacyschlüssel“ und als Anforderungsformat „PKCS#10“. Klicken Sie auf „Weiter“.

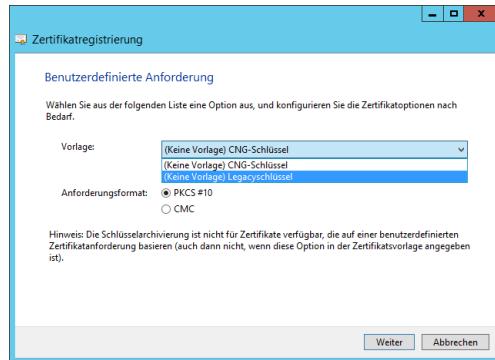


Abb. 19.52:
Anforderungs-
vorlage und
-format auswählen

- Klicken Sie im Schritt „Zertifikatsinformationen“ auf „Details“ und anschließend auf die Schaltfläche „Eigenschaften“. Es öffnet sich der Dialog „Zertifikateigenschaften“.

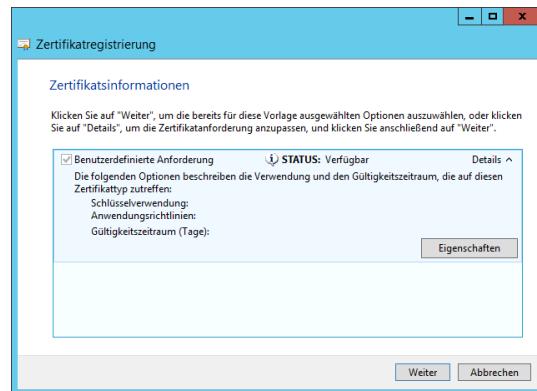


Abb. 19.53:
Anpassen der
CSR-
Eigenschaften

- Geben Sie einen Anzeigenamen und eine kurze Beschreibung für das Zertifikat ein, so wie in Abbildung ?? zu sehen.

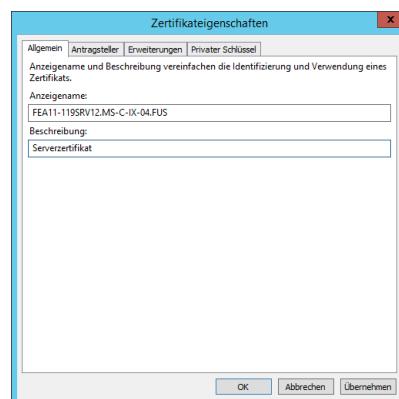


Abb. 19.54:
Allgemeine
Angaben zum
Zertifikat

- Wechseln Sie auf die Registerkarte „Antragsteller“. Erstellen Sie die für das Zertifikat notwendigen Einträge:

Antragstellernname:

- Allgemeiner Name: FEA11-119SRV12.MS-C-IX-04.FUS
- Land/Region: DE
- E-Mail: SQL-IX-62-12@MS-C-IX-04.FUS
- Ort: Feldafing
- Organisation: Bundeswehr
- Organisationseinheit: FueUstgSBw

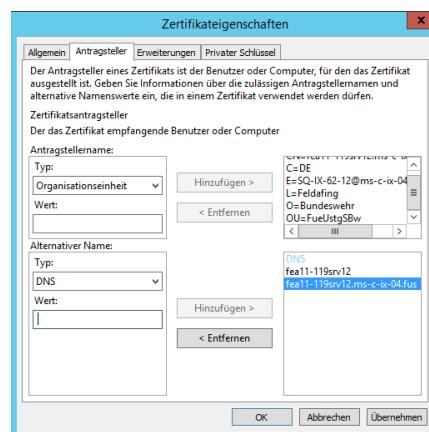


Tauschen Sie die Platznummer 12 jeweils durch Ihre eigene Platznummer aus!

Alternativer Name:

- DNS: FEA11-119SRV12
- DNS: FEA11-119SRV12.MS-C-IX-04.FUS

Abb. 19.55:
Angaben zum
Antragsteller



15. Wechseln Sie auf die Registerkarte „Erweiterungen“.
16. Öffnen Sie das Menü „Erweiterte Schlüsselverwendung (Anwendungsrichtlinien)“.

17. Wählen Sie „Serverauthentifizierung“ bei den Verfügbaren Optionen.

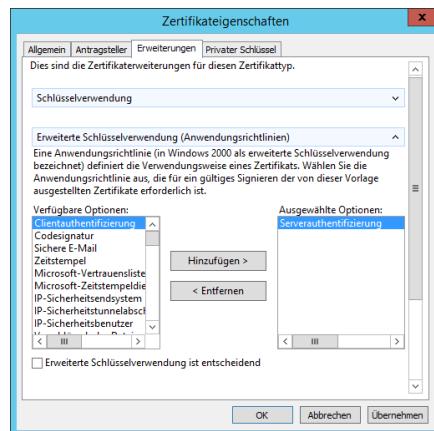


Abb. 19.56:
Zertifikats-
optionen
auswählen

18. Wechseln Sie auf die Registerkarte „Privater Schlüssel“.

19. Öffnen Sie das Menü „Schlüsseloptionen“.

20. Wählen Sie 2048 Byte als Schlüsselgröße und aktivieren Sie die Option „Privaten Schlüssel exportierbar machen“.

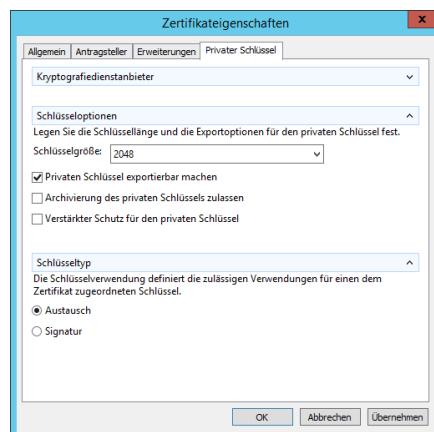


Abb. 19.57:
Eigenschaften des
privaten
Schlüssels ändern

21. Wählen Sie im Menü „Schlüsseltyp“ die Option „Austausch“.

22. Öffnen Sie das Menü „Schlüsselberechtigungen“.

23. Aktivieren Sie die Option „Benutzerdefinierte Berechtigungen verwenden“ und klicken Sie auf „OK“.

24. Klicken Sie auf „Weiter“.

25. Geben Sie als Dateinamen D:\FEA11-119SRV12.MS-C-IX-04.FUS.csr ein und achten Sie darauf, dass als Dateiformat „Base 64“ gewählt ist.
26. Klicken Sie auf „Fertigstellen“

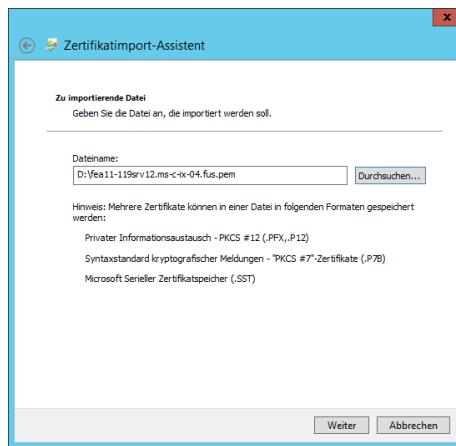
Die erstellte Anforderung muss nun durch eine CA signiert werden.

Importieren des Serverzertifikats

Das Importieren des Serverzertifikats geschieht, genau wie das Erstellen des Requests, mit Hilfe des Zertifikatplugins auf dem Datenbankserver.

1. Starten Sie das Zertifikatsplugin.
2. Klicken Sie links außen den Eintrag „Eigene Zertifikate“ an und öffnen Sie das Kontextmenü.
3. Wählen Sie „Alle Aufgaben“ → „Erweiterte Vorgänge“ → „Importieren“.
4. Klicken Sie im Willkommensdialog auf „Weiter“.
5. Geben Sie als Dateinamen D:\fea11-119srv12.ms-c-ix-04.fus.pem ein.

Abb. 19.58:
Importieren eines
Zertifikats



6. Wählen Sie im nächsten Schritt die Option „Alle Zertifikate in folgendem Speicher speichern“ und wählen Sie als Speicher „Eigene Zertifikate“ aus. Klicken Sie auf „Weiter“.
7. Klicken Sie auf „Fertigstellen“.

Abb. 19.59:
Auswählen des
Zertifikats-
speichers

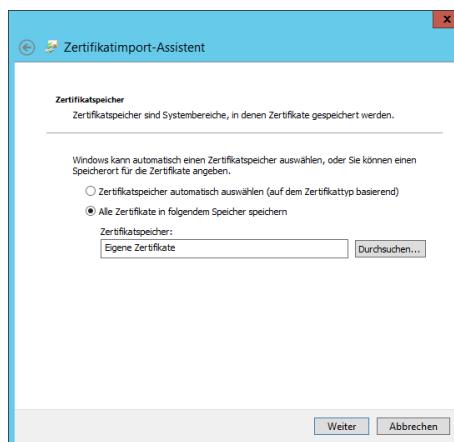
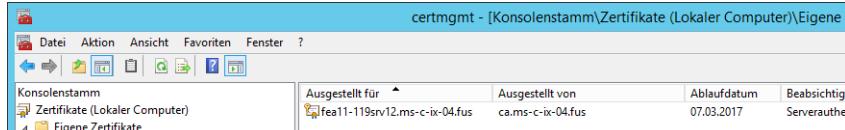


Abb. 19.60:
Das importierte
Server-Zertifikat



Es ist wichtig, dass das Zertifikat in den Zertifikatsspeicher des Computerkontos importiert wird. Nur so können verschiedene Benutzerkonten berechtigt werden, auf das Zertifikat zuzugreifen.



Festlegen der Zugriffsrechte auf den privaten Schlüssel

Nach dem Import des Zertifikats in den Speicher des Computerkontos können bisher nur die drei folgenden Konten darauf zugreifen:

- Lokaler Dienst (NT AUTHORITY\LOCALSERVICE)
- Lokales System (NT AUTHORITY\localsystem)
- Netzwerkdienst (NT AUTHORITY\NETWORKSERVICE)

Damit das gMSA-Dienstkonto des SQL Server (MS-C-IX-04\MSSQLSERVER12\$) das Zertifikat nutzen kann, muss im der Zugriff auf den privaten Schlüssel des Zertifikats eingeräumt werden.

1. Öffnen Sie den Zertifikatsmanager für das Computerkonto.
2. Klicken Sie mit der rechten Maustaste auf das Zertifikat und wählen Sie aus dem Kontextmenü „Alle Aufgaben“ → „Private Schlüssel verwalten“.
3. Fügen Sie das Dienstkonto des SQL Servers hinzu und geben Sie im Leserechte auf das Zertifikat.



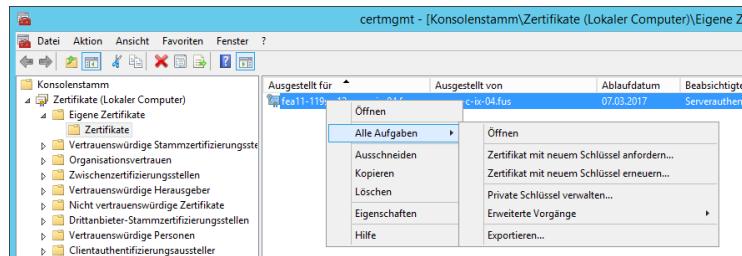


Abb. 19.61:
Verwalten der
privaten Schlüssel

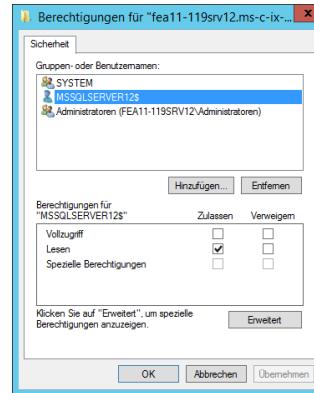
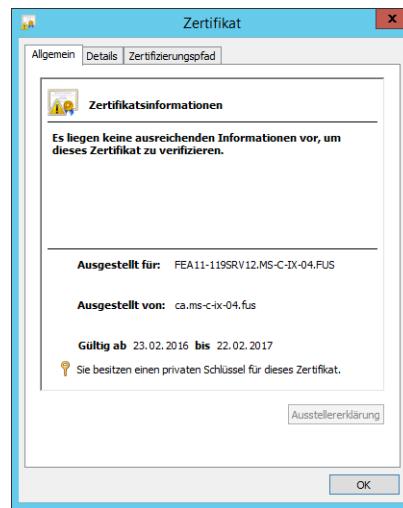


Abb. 19.62:
Verwalten der
privaten Schlüssel

Importieren des CA-Zertifikats

Um die Echtheit des Serverzertifikats überprüfen zu können, muss der SQL Server auch das CA-Zertifikat importieren.

Abb. 19.63:
Fehlende
Zertifikats-
informationen



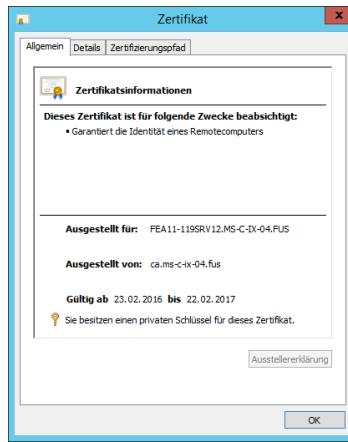
Die Vorgehensweise ist die gleiche, wie beim Import des Serverzertifikats. Es ist lediglich darauf zu achten, dass im zweiten Schritt des Assistenten als Zertifikatsspeicher „Vertrauenswürdige Stammzertifizierungsstellen“ gewählt werden muss. Die anschließende Sicherheitswarung muss mit „Ja“ bestätigt werden.

Abb. 19.64:
Das importierte
CA-Zertifikat

CA 沃通根证书	CA 沃通根证书	08.08.2039	Serverauthentifizier...	CA 沃通根证书
ca.ms-c-ix-04.fus	ca.ms-c-ix-04.fus	20.02.2026	<Alle>	<Keine>
CCA India 2007	CCA India 2007	04.07.2015	Serverauthentifizier...	CCA India 2007
CCA India 2011	CCA India 2011	11.03.2016	Serverauthentifizier...	CCA India 2011
Certeurope Root CA 2	Certeurope Root CA 2	28.03.2037	Serverauthentifizier...	CertEurope

Mit Hilfe des CA-Zertifikats kann nun die Echtheit des Serverzertifikats nachgewiesen werden.

Abb. 19.65:
Vollständige
Certificate chain



19.8.2 Konfigurieren der serverseitigen Verschlüsselung

Auf Seiten des SQL Servers muss in der Netzwerkkonfiguration das Serverzertifikat angegeben und die Verschlüsselung aktiviert werden. Diese Schritte werden alle im SQL Server Configuration Manager ausgeführt.

1. Starten Sie den SQL Server Configuration Manager.
2. Öffnen Sie die mit Hilfe des Kontextmenüs die Eigenschaften der SQL Server Netzwerkprotokolle.

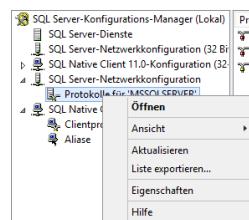


Abb. 19.66:
Die SQL
Server-Netzwerk-
protokolle

3. Wechseln Sie im Eigenschaftendialog auf die Registerkarte „Zertifikat“ und wählen Sie in der Dropdownbox „Zertifikat“ das installierte Serverzertifikat aus.

Hier können nur solche Zertifikate ausgewählt werden, die vorher mit dem Zertifikatsplugin der MMC installiert wurden und die den von Microsoft formulierten Anforderungen für ein Serverzertifikat entsprechen.





- [ms189067]
- [ms191192]

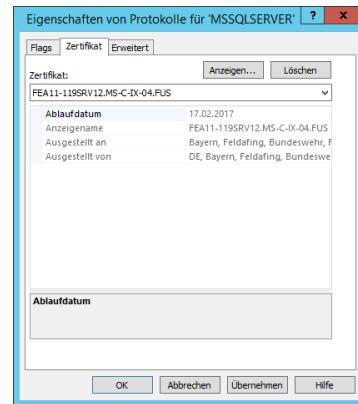


Abb. 19.67:
Die Registerkarte „Zertifikat“

- Wechseln Sie zurück auf die Registerkarte „Flags“. Aktivieren Sie dort die Option „Verschlüsselung erzwingen“.

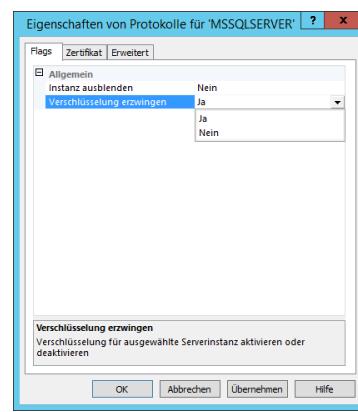


Abb. 19.68:
Die Registerkarte „Flags“

- Klicken Sie auf „OK“ und starten Sie den SQL Server-Dienst neu.



Eine unverschlüsselte Kommunikation mit dem SQL Server ist nun nicht mehr möglich.

Probleme bei der Auswahl des Zertifikats

Es kann vorkommen, dass der SQL Server nicht in der Lage ist, ein korrekt erstelltes und in den Speicher des Computerkontos importiertes Zertifikat zu erkennen. Dies äußert sich so, dass das Zertifikat auf der Registerkarte „Zertifikat“ der Netzwerkprotokolleigenschaften des Servers nicht zur Auswahl steht.

In einem solchen Fall, kann zu einer Notlösung gegriffen werden. Der Fingerprint des Zertifikats wird direkt in die Registry eingetragen. Dieser Vorgang wird von Microsoft als offizielle Lösung empfohlen.

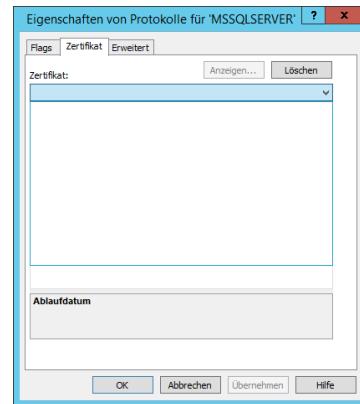


Abb. 19.69:
Kein Zertifikat
zur Auswahl

- Lesen Sie die Eigenschaft „Fingerabdruck“ aus Ihrem Zertifikat aus.

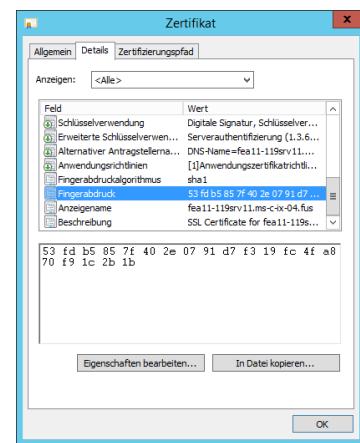


Abb. 19.70:
Der
Fingerabdruck
des Zertifikats

- Öffnen Sie den Registrierungsseditor mit dem Kommando „regedit.exe“
- Navigieren Sie zu dem folgenden Schlüssel:

```
\HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Microsoft SQL Server\MSSQL12.MSSQLSERVER
```

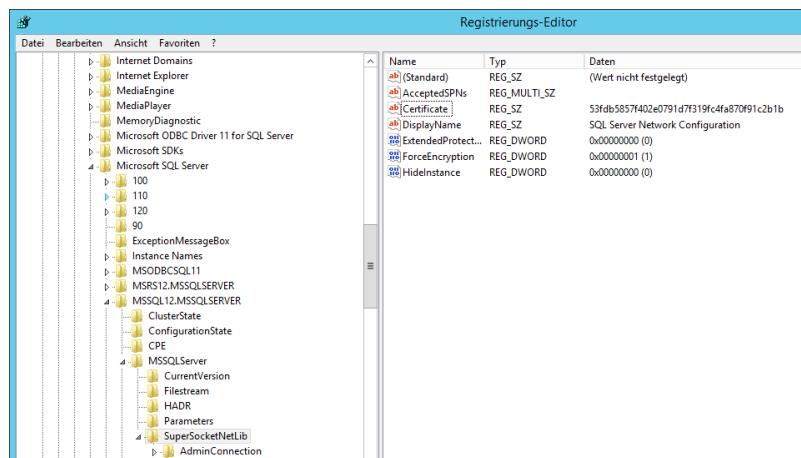
```
\MSSQLServer\SuperSocketNetLib
```

- Wählen Sie dort den Registryeintrag „Certificate“ und tragen Sie den Fingerprint Ihres Zertifikats ein.



Der Fingerabdruck muss ohne Leerzeichen eingetragen werden. Es dürfen auch keine unsichtbaren Steuerzeichen in den Registryeintrag hineinkopiert werden. Das Eintragen sollte deshalb von Hand erfolgen.

Abb. 19.71:
Eintragen des
Fingerabdrucks



Der SQL Server-Dienst muss nun neugestartet werden. Sofern mit dem Zertifikat alles in Ordnung ist, wird der Dienst wie erwartet starten.

19.8.3 Konfigurieren der clientseitigen Verschlüsselung

Auf der Clientseite kann die Verschlüsselung auf zwei unterschiedliche Arten konfiguriert werden:

- Die Verschlüsselung kann für jede Applikation einzeln aktiviert/deaktiviert werden.
- Die Verschlüsselung kann für alle Anwendungen auf einem Client erzwungen werden.

Wichtig ist, dass der Clientrechner entweder das original Serverzertifikat oder das Zertifizierungsstammstellenzertifikat zur Verfügung haben muss, um eine sichere SSL-Verbindung mit dem Datenbankserver herstellen zu können.

Aktivieren/Deaktivieren der Verschlüsselung in einer Anwendung

Um eine verschlüsselte Verbindung zu einem SQL Server aufzubauen zu können, muss einen Client-Anwendung einen Datenbanktreiber benutzen, der diese Möglichkeit bietet. Ein Beispiel für eine solche Anwendung ist das Microsoft SQL Server Management Studio. Es benutzt den „SQL Native Client 11.0“ von Microsoft. Um die Verschlüsselung zu aktivieren, muss im AnmeldeDialog die Option „Verbindung verschlüsseln“ auf der Registerkarte „Verbindungseigenschaften“. Dies kann durch einen Klick auf die Schaltfläche „Optionen“ erreicht werden.

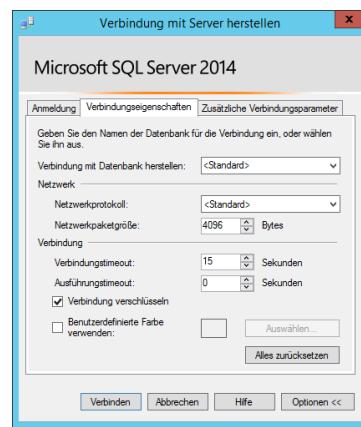


Abb. 19.72:
Verschlüsseln der
Verbindung im
SSMS

Erzwingen der Verschlüsselung für alle Anwendungen

Im SQL Server Configuration Manager können die SQL Native Client Komponenten so konfiguriert werden, dass alle Anwendungen, welche diesen Datenbanktreiber benutzen, ihre Verbindung verschlüsseln.

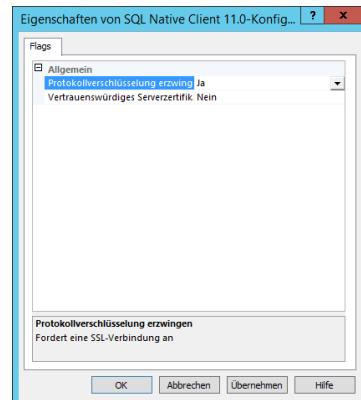


Abb. 19.73:
Erzwingen der
Verschlüsselung
auf dem Client

Dieser Dialog wird über den Eintrag „Eigenschaften“ des Kontextmenüs des Eintrags „SQL Native Client 11.0-Konfiguration“ erreicht. Die beiden dort sichtbaren Optionen haben folgende Bedeutung:

- **Protokollverschlüsselung erzwingen:** Wird diese Option auf den Wert „Ja“ eingestellt, fordert der Client eine SSL-Verbindung zum Server an. Der Verbindung wird dann in jedem Fall verschlüsselt.
- **Vertrauenswürdiges Serverzertifikat:** Solange diese Option den Wert „Nein“ hat, überprüft der Client die Identität des Servers. So können u. U. Man-In-The-Middle-Attacken verhindert werden.



Für die Überprüfung der Serveridentität muss dem Client entweder das original Serverzertifikat oder das Zertifizierungsstammstellenzertifikat zur Verfügung stehen. Der Import dieses Zertifikats verläuft analog zum Import auf dem Datenbankserver.

19.8.4 Ist die Verbindung auch wirklich verschlüsselt?

Die Antwort auf diese Frage kann nur die Katalogsicht `SYS.DM_EXEC_CONNECTIONS` geben. Sie enthält eine Spalte `ENCRYPT_OPTION`, welche für verschlüsselte Verbindungen den Wert `TRUE` anzeigt.

Listing 19.32: Verschlüsselungseigenschaften der Verbindung abfragen

```
SELECT net_transport, client_net_address, encrypt_option
FROM sys.dm_exec_connections;
```

NET_TRANSPORT	CLIENT_NET_ADDRESS	ENCRYPT_OPTION
Shared memory	<local machine>	TRUE
TCP	192.168.111.43	TRUE
2 Zeilen ausgewählt		

19.8.5 Nutzen der Option „Erweiterter Schutz“

Der Erweiterte Schutz ist eine Funktion, die seit Windows Server 2008 R2/Windows 7 durch das Betriebssystem zur Verfügung gestellt wird. Sie beinhaltet zwei Komponenten, die Dienstbindung und die Kanalbindung, welche in Kombination als Abwehrmaßnahme gegen Relay-Attacken funktionieren. Standardmäßig ist der Erweiterte Schutz nicht aktiviert, dies muss manuell durch den Administrator geschehen.

Wovor schützt der Erweiterte Schutz?

Laut Microsoft kann dieser Mechanismus vor zwei Angriffsarten schützen:

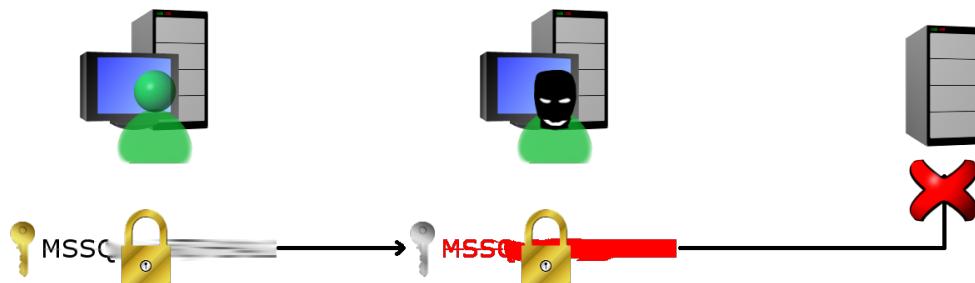
- **Lockangriff:** Der Client wird dazu verleitet (angelockt) freiwillige eine Verbindung zu einem Angreifer herzustellen.
- **Spoofingangriff:** Bei einem solchen Angriff wird dem Client eine manipulierte Route zum Ziel zur Verfügung gestellt, so dass er unbemerkt beim Angreifer landet.

Welche Schutzmaßnahmen bietet der Erweiterte Schutz?

Durch das Aktivieren des Erweiterten Schutzes werden zwei Mechanismen in Gang gesetzt:

- **Dienstbindung:** Bei der Dienstbindung signiert der Client den Service Principal Name (SPN) des Servers und sendet diesen verschlüsselt an den Server. Durch die Signatur des SPN kann der Server erkennen von welchem Client der SPN stammt. Durch die Verschlüsselung kann kein anderer Client den SPN für seine Zwecke missbrauchen.

Abb. 19.74:
Schematische
Darstellung der
Dienstbindung



- **Kanalbindung:** Bei der Kanalbindung wird ein sicherer Kanal zwischen dem Client und dem Server eingerichtet. Diese Methode wird mit Hilfe des TLS (ehemals SSL) Protokolls umgesetzt.



- [ff487261]

Den Erweiterten Schutz in Windows aktivieren

Die Funktion „Erweiterter Schutz“ kann nur durch zwei Registry-Einträge aktiviert werden. Der Administrator muss prüfen, ob die Schlüssel

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\SuppressExtendedProtection`

und

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\LMCompatibilityLevel`

bereits existiert. Falls dies nicht der Fall ist, müssen die Schlüssel manuell angelegt werden (Schlüsseltyp = DWORD).

1. Starten Sie den Registrierungseditor mit `regedit.exe`

2. Navigieren Sie durch den Registrierungsbaum bis Sie den folgenden Schlüssel erreichen: HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\SecureBoot\SecurePackages\SuppressExtendedProtection
3. Öffnen Sie das Kontextmenü mit der rechten Maustaste und wählen Sie „Neu“ → „DWORD-Wert (32-Bit)“

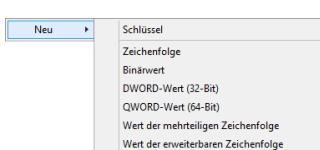


Abb. 19.75:
Anlegen eines
DWORD-
Schlüssels in der
Registry

4. Geben Sie dem neuen Wert den Namen SuppressExtendedProtection.

Notification Packages	REG_MULTI_SZ	rassfm sccl
ProductType	REG_DWORD	0x00000007 (7)
restrictanonymous	REG_DWORD	0x00000000 (0)
restrictanonymoussam	REG_DWORD	0x00000001 (1)
SecureBoot	REG_DWORD	0x00000001 (1)
Security Packages	REG_MULTI_SZ	''
SuppressExtendedProtection	REG_DWORD	0x00000000 (0)

Abb. 19.76:
Der Schlüssel
Suppress-
Extended-
Protection

5. Der Schlüssel muss den Wert 0 haben.

6. Öffnen Sie das Kontextmenü mit der rechten Maustaste und wählen Sie „Neu“ → „DWORD-Wert (32-Bit)“

7. Geben Sie dem neuen Wert den Namen LmCompatibilityLevel.

SuppressExtendedProtection	REG_DWORD
LmCompatibilityLevel	REG_DWORD

Abb. 19.77:
Der Schlüssel
LmCompatibility-
Level

8. Doppelklicken Sie auf den Wert LmCompatibilityLevel.

9. Geben Sie den Wert 3 ein und klicken Sie auf OK.

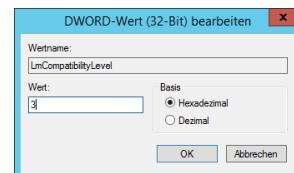


Abb. 19.78:
Der Schlüssel
LmCompatibility-
Level

10. Starten Sie den Windows Server neu.

Den Erweiterten Schutz für SQL Server aktivieren

Der Erweiterte Schutz für den SQL Server wird mit Hilfe von drei Einstellungen im SQL Server-Konfigurations-Manager verwaltet:

- **Erzwingen der Verschlüsselung:** Durch die Konfiguration dieser Eigenschaft auf den Wert Ein wird die Kanalbindung erzwungen, da der SQL Server die Verschlüsselung einer jeden Verbindung erzwingt.
- **Erweiterter Schutz:** Diese Eigenschaft besitzt drei unterschiedliche Werte: Aus, Zulässig und Erforderlich.

Durch die Einstellung Zulässig wird die Dienstbindung bei allen Clients benutzt die dies unterstützen. Sollte ein Client keine Dienstbindung zur Verfügung stellen wird die Verbindung ohne Dienstbindung aufgebaut.

Durch die Einstellung Erforderlich wird die Dienstbindung für alle Clients erzwungen. Sollte ein Client keine Dienstbindung zur Verfügung stellen wird keine Verbindung aufgebaut.

- **Akzeptierte NTLM-SPNs:** Diese Eigenschaft wird nur dann benötigt, wenn ein Datenbankserver durch mehrere SPN identifiziert wird. In diesem Fall kann hier eine durch Semikola separierte Liste von Service Principal Names angegeben werden.

Zufinden sind diese Eigenschaften durch einen Rechtsklick auf den Punkt „Protokolle für 'MSSQLSERVER'“

Abb. 19.79:
Aktivieren des
Erweiterten
Schutzes



Die Einstellung „Verschlüsselung erzwingen“ muss auf den Wert „Ja“ eingestellt werden, um die Kanalbindung zu ermöglichen.

Auf der Registerkarte „Erweitert“ muss die Einstellung „Erweiterter Schutz“ auf den Wert „Erforderlich“ eingestellt werden, um Kanal- und Dienstbindung zu erzwingen.



Abb. 19.80:
Aktivieren des
Erweiterten
Schutzes

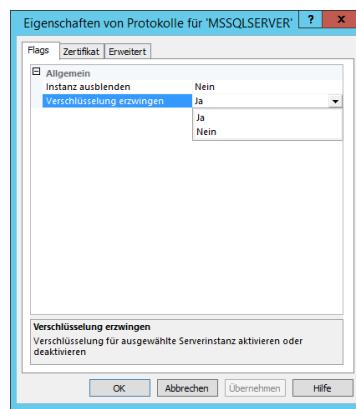
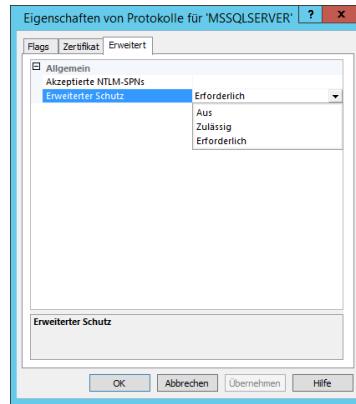


Abb. 19.81:
Aktivieren des
Erweiterten
Schutzes



- [ff487261]

20 Partially contained databases

Inhaltsangabe

20.1 Partially contained databases - Teilweise eigenständige Datenbanken

Die Architektur des Microsoft SQL Server sieht vor, dass Einstellungen, die zum Betrieb einer Datenbank notwendig sind, an verschiedenen Orten gespeichert werden. Beispielsweise werden Logins in der MASTER-Datenbank gespeichert, um instanzweit zur Verfügung zu stehen, während Benutzerkonten direkt in der betreffenden Datenbank abgelegt werden. Ein anderes Beispiel sind Jobs des SQL Server Agent. Diese werden in der MSDB-Datenbank abgelegt, können sich aber auf Datenbanken auswirken.

Die durch diese Umstände bewirkte Verflechtung von Instanz und Datenbanken macht es schwierig eine Datenbank auf einen anderen Server / in eine andere Instanz zu migrieren. Mit Hilfe des Konzeptes der „Teilweise eigenständigen Datenbank“ (engl. Partially contained databases) versucht Microsoft diesem Problem entgegen zu treten.

In einer Partially contained database werden verschiedene Metadaten in der Datenbank selbst und nicht in der MASTER-Datenbank gespeichert. Der Fachbegriff dafür, dass bestimmte Objekte ohne Verbindung zur Instanz oder zu einer anderen Datenbank existieren können lautet „Kapselung“



Von großer Bedeutung ist das Wort „Partially“, da nicht alle Metadaten in der Datenbank selbst abgelegt werden. Diese Art von Datenbank ist tatsächlich nur zu einem gewissen Teil unabhängig.

20.1.1 Was genau bedeutet Kapselung?

Bei der Kapselung geht es darum, dass Objekte in einer Datenbank keinerlei Bezüge/Verknüpfungen nach außen haben dürfen. Ein Beispiel hierfür ist die Systemview `SYS.TABLES`. Die Basisobjekte, auf welche sich diese View bezieht, befinden sich alle in der betroffenen Datenbank selbst. Ein Gegenbeispiel wäre ein Objekt, wie z. B. ein SQL-Benutzer mit Anmeldename. Ein solches Benutzerkonto bezieht sich auf ein Windows-Login und dieses Login liegt in der MASTER-Datenbank. Somit existiert ein Bezug nach außen.

Man kann somit zwei Arten von Objekten unterscheiden:

- **Contained entities:** Ein Objekt wird dann als „vollständig enthalten“ bezeichnet, wenn es die Datenbankbegrenzung nie überschreitet, sich also nur auf Funktionen stützt, welche in der Datenbank enthalten sind.

- **Uncontained entities:** Ein Objekt, welches die Datenbankgrenze überschreitet wird als „nicht enthalten“ bezeichnet, da es Verknüpfungen zu Objekten außerhalb seiner eigenen Datenbank hat.



Als „Datenbankbegrenzung“ wird die Grenze zwischen einer Datenbank und der Instanz bzw. die Grenze zwischen zwei Datenbanken bezeichnet. Sie ist eine eindeutig definierbare Trennlinie zwischen den Funktionen der Datenbank (dem Datenmodell) und den Funktionen der Instanz (dem Verwaltungsmodell).

20.1.2 Welche Probleme werden gelöst?

- Die Datenbank übernimmt die Authentifizierung von Benutzern komplett. Es sind keine Logins mehr von Nöten.
- Weitere Metadaten, z. B. eine Liste der Datenbanken, von denen die Partially contained database abhängig ist und weitere Systemeinstellungen, die zum Betrieb der Datenbank notwendig sind, werden in der Datenbank selbst gespeichert.
- Temporäre Tabellen werden direkt im Kontext der Partially contained database gespeichert, so dass es keine Konflikte mehr bei Sortierreihenfolgen zwischen der TEMPDB und der Datenbank selbst gibt.

20.1.3 Einschränkungen für Partially contained databases

Das Konzept der teilweise eigenständigen Datenbanken bringt nicht nur Vorteile mit sich, es existieren auch Einschränkungen. Beispielsweise unterstützen diese Datenbank weder das Feature der Replikation noch das Change-Data-Capture. Eine vollständige Liste mit allen Einschränkungen ist in der MSDN enthalten.



- [ff929071]

20.2 Partially contained databases erzeugen

20.2.1 Konfigurieren der Instanz

Bevor Partially contained databases erstellt werden können muss zuerst die Authentifizierung für eigenständige Datenbanken eingeschaltet werden. Dies wird durch eine einfache Instanzeinstellung vorgenommen.

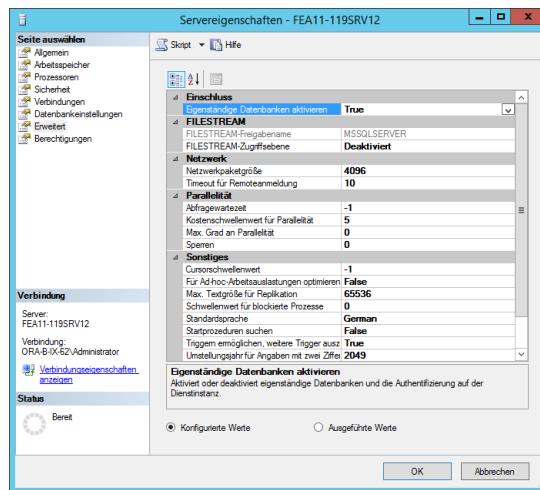


Abb. 20.1:
Eigenständige
Datenbanken
aktivieren

Listing 20.1: Eigenständige Datenbanken aktivieren

```
USE [master]
GO

EXEC sp_configure 'contained database authentication', 1
RECONFIGURE
GO
```

20.2.2 Erstellen der Datenbank

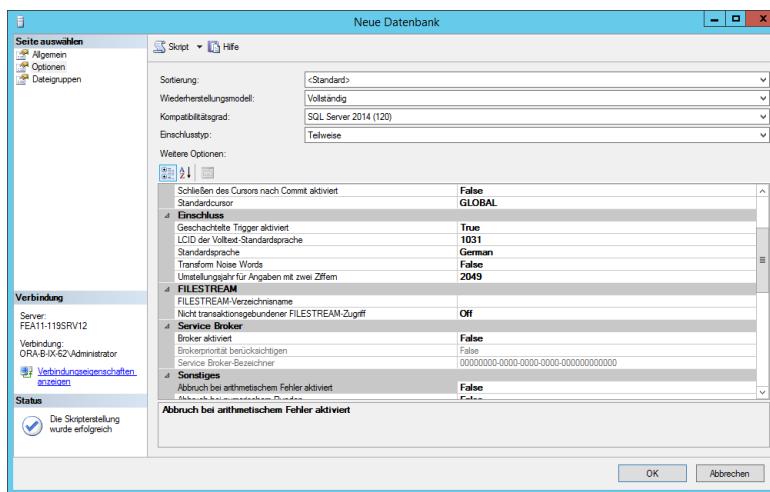
Eine neue Partially contained database erstellen

Eine teilweise eigenständige Datenbank wird mit Hilfe des `CREATE DATABASE`-Statements und dem Zusatz `CONTAINMENT = PARTIAL` erzeugt. Wahlweise kann dies aber auch im SSMS geschehen.

Listing 20.2: Erstellen einer teilweise eigenständigen Datenbank

```
USE [master]
GO
```

Abb. 20.2:
Erzeugen einer
teilweise
eigenständigen
Datenbank



```

CREATE DATABASE [Bank_PCD]
CONTAINMENT = PARTIAL
ON PRIMARY ( NAME = N'Bank_PCD',
FILENAME = N'D:\120\bank_pcd\data\Bank_pcd.mdf',
SIZE = 1GB, MAXSIZE = 2GB,
FILEGROWTH = 512MB)
LOG ON ( NAME = N'Bank_PCD_Log',
FILENAME = N'D:\120\bank_pcd\log\Bank_pcd.ldf',
SIZE = 100MB, MAXSIZE = 2GB,
FILEGROWTH = 10%)
GO

```

Den Containmenttype einer Datenbank ändern

Soll eine bestehende Datenbank in eine Partially contained database migriert werden, so geschieht dies mit dem **ALTER DATABASE**-Statement.

Listing 20.3: Migration einer Datenbank in eine teilweise eigenständige Datenbank

```

USE [master]
GO

ALTER DATABASE [Bank_PCD] SET CONTAINMENT = PARTIAL;
GO

```

- [hh534404]



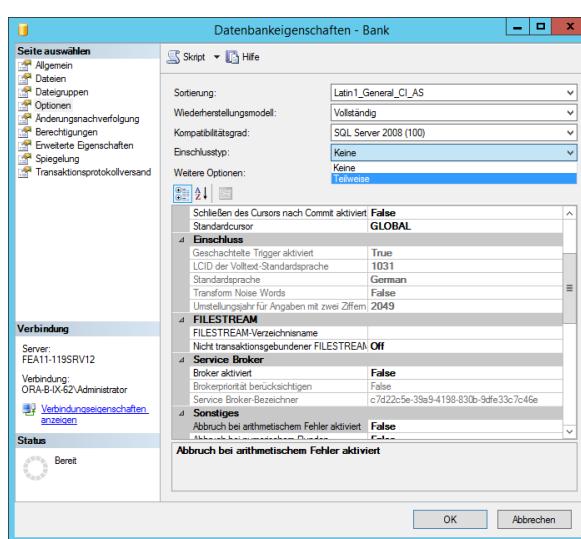


Abb. 20.3:
Migration einer
Datenbank in eine
teilweise
eigenständige
Datenbank

Identifizieren der Datenbankkapselung

Wurde eine Datenbank in eine Partially contained database migriert, stellt sich trotz Migration die Frage, ob wirklich alle Objekte innerhalb der Datenbank die Datenbankbegrenzung einhalten. Um dies herauszufinden, gibt es zwei Mechanismen:

- Die View `SYS.DM_DB_UNCONTAINED_ENTITIES` zeigt alle Objekte an, die möglicherweise die Datenbankbegrenzung überschreiten.
- Das Extended Event `DATABASE_UNCONTAINED_USAGE` wird immer dann ausgelöst, wenn zur Laufzeit eines Programms eine uncontained Entity gefunden wird.



Die View `SYS.DM_DB_UNCONTAINED_ENTITIES` kann Entitäten, welche dynamisches SQL enthalten nicht korrekt auswerten und deshalb auch nicht feststellen, ob solche Objekte die Datenbankbegrenzung verletzen oder nicht.



- [ff929071]

20.2.3 Contained users verwalten

Grundsätzlich sind die „enthaltenen Benutzerkonten“ bzw. Contained Users mit den normalen Benutzerkonten identisch, bis auf eine Ausnahme: Es wird kein Login für einen Contained user erstellt. Die Metadaten eines Contained users werden in der Datenbank selbst und nicht in der MASTER-Datenbank abgelegt.

Es gibt zwei Arten Contained users:

- **SQL Benutzer mit Kennwort:** Diese Art von Benutzerkonto ist das Pendant zur Kombination „SQL Benutzer mit Anmeldename und Login mit SQL Server-Authentifizierung“. Der Unterschied besteht darin, dass kein Login für ihn angelegt wird. Das Passwort für ein solches Benutzerkonto wird in der Datenbank selbst verwaltet.
- **Windows-Benutzer:** Der Windows Benutzer ist mit der Kombination „SQL Benutzer mit Anmeldename und Login mit Windows-Authentifizierung“ vergleichbar.

Zu beachten ist, dass das Authentifizierungsverfahren bei beiden Benutzertypen unterschiedlich abläuft. Meldet sich ein Benutzer mit einem Windows-Account an, wird zuerst nach einem Login mit Windows-Authentifizierung gesucht und erst wenn kein passendes Login gefunden wird, wird nach einem Windows-Benutzer gesucht. D. h. die Authentifizierung wird zuerst auf Instanzebene und dann auf Datenbankebene versucht.

Bei einer Anmeldung mit einem SQL Server-Account, wird zuerst nach einem SQL Benutzer mit Kennwort und dann nach einem Login mit SQL Server-Authentifizierung gesucht. Hier wird also zuerst auf Datenbankebene und dann auf Instanzebene authentifiziert.

Contained users erstellen

Contained Users können wie bereits in ?? gezeigt, mit Hilfe des SSMS oder mit SQL-Kommandos erstellt werden. Die beiden folgenden Statements zeigen wie ein SQL Benutzer ohne Anmeldename und ein Windows-Benutzer erstellt werden.

Listing 20.4: Erstellen eines SQL Benutzers mit Kennwort

```
USE [Bank_PCD]
GO

CREATE USER [sql_user_without_login]
WITH PASSWORD = 'P@ssw0rd'
GO
```

Listing 20.5: Erstellen eines Windows-Benutzers

```
USE [Bank_PCD]
GO

CREATE USER [ORA-B-IX-62\sqluser]
GO
```

Contained users können nur innerhalb einer eigenständigen Datenbank erstellt werden. Versucht man einen contained user in einer nicht eigenständigen Datenbank zu erstellen antwortet SQL Server mit den folgenden Fehlermeldung.

Listing 20.6: Fehler beim Erstellen eines SQL Benutzers mit Kennwort in einer nicht eigenständigen Datenbank

```
USE [Bank]
GO

CREATE USER [sql_user_without_login]
WITH PASSWORD = 'P@ssw0rd'
GO

Meldung 33233, Ebene 16, Status 1, Zeile 3
Sie k\"onnen nur einen Benutzer mit einem Kennwort in einer eigenst\"andigen
Datenbank erstellen.
```

Login an einer Partially contained database

Für die Anmeldung an einer Partially contained database muss in den Verbindungseigenschaften der Name der gewünschten Datenbank angegeben werden.

Abb. 20.4:
Login an einer
Partially
contained
database



Alternativ zur Anmeldung mittels SSMS kann auch mit Hilfe des SQLCMD-Tools eine Verbindung zu einer Contained database hergestellt werden.

Listing 20.7: Login an einer Partially contained database mit dem SQLCMD-Tool

```
sqlcmd -S FEA11-119SRV12\MSSQLSERVER -U sql_user_without_login -P P@sswOrd
-d Bank_PCD
```



- [bmmcspb202ss2pcdp1]
- [hh534404]

Sicherheitsrelevante Einstellungen

Bei der Benutzung von eigenständigen Benutzerkonten gibt es einige kritische Punkte, die man in seine Überlegungen mit einbeziehen sollte. Darunter fallen die Datenbankeigenschaft AUTO_CLOSE und das Systemprivileg alter any user.

Durch das Setzen der Datenbankeigenschaft AUTO_CLOSE wird eine Datenbank geschlossen, sobald sich der letzte Benutzer von ihr abgemeldet hat. Da bei einer eigenständigen Datenbank die Authentifizierung nicht durch die Instanz, sondern durch die Datenbank selbst gemacht wird, bedeutet dies, dass nach dem Schließen der Datenbank zusätzliche Ressourcen aufgewandt werden müssen, um sie wieder zu öffnen. Wenn sich nun ein einziger Benutzer an der Datenbank wiederholt an- und abmeldet, kann dem Server daraus eine große Belastung entstehen, was sich letzten endes wie eine Denial of Service Attacke auswirkt.

Das Systemprivileg alter any user stellt in einer nicht eigenständigen Datenbank kein allzu großes Risiko dar, weil die Authentifizierung auf Instanzebene abläuft. In einer eigenständigen Datenbank jedoch, kann mit Hilfe dieses Privileges einem Benutzer Zugang zu einer Datenbank verschafft bzw. ein neuer Benutzer erstellt werden. Daher sollten dieses Privileg und die beiden Rollen DB_OWNER und DB_SECURITYADMIN, welche in Besitz dieses Privileges sind, mit größter Vorsicht gehandhabt werden.



Wenn in einer Datenbank das Gast-Konto aktiviert ist, besteht zusätzlich die Gefahr, dass sich die „illegal erstellten“ Benutzerkonto Zugang zu dieser Datenbank verschaffen.

Eine ähnliche Gefahrt droht von den beiden Systemprivilegien alter any database (fixed server role dbcreateor) und control database (fixed server role db_owner). Wenn ein Benutzer eines der beiden Privilegien besitzt, kann er den status einer Datenbank auf „teilweise eigenständig“ setzen. Daraus resultiert wiederum das soeben beschriebene Problem mit eigenständigen Benutzerkonten, die sich illegal Zugang zu anderen Datenbanken verschaffen können.



- [ff929055]

Das Passwort eines eigenständigen Benutzerkontos ändern

Listing 20.8: Ändern des Passwortes eines eigenständigen Benutzers

```
USE [Bank_PCD]
GO

ALTER USER [sql_user_without_login]
WITH
    PASSWORD = 'Pa$$w0rd',
    OLD_PASSWORD = 'P@ssw0rd';
GO
```

Benutzerkonten migrieren

Um alle Benutzerkonten des Types „SQL Benutzer mit Anmeldename“, die auf einem Login mit Passwort-Authentifizierung basieren in einen „SQL Benutzer mit Passwort“ zu konvertieren, stellt Microsoft eigens die Stored Procedure `sp_migrate_user_to_contained` zur Verfügung. Das folgende Beispiel ist der MSDN ([ff929139]) entnommen und zeigt, wie ein solcher Vorgang für eine Partially contained database abläuft.

Listing 20.9: Benutzerkonten migrieren

```
DECLARE @username sysname;
DECLARE user_cursor CURSOR
FOR
    SELECT dp.name
    FROM sys.database_principals AS dp
    JOIN sys.server_principals AS sp
    ON dp.sid = sp.sid
    WHERE dp.authentication_type = 1 AND sp.is_disabled = 0;
OPEN user_cursor
FETCH NEXT FROM user_cursor INTO @username
WHILE @@FETCH_STATUS = 0
BEGIN
    EXECUTE sp_migrate_user_to_contained
    @username = @username,
    @rename = N'keep_name',
    @disablelogin = N'disable_login';
    FETCH NEXT FROM user_cursor INTO @username
END
CLOSE user_cursor;
DEALLOCATE user_cursor;
```

- [ff929139]



21 Auditing

Inhaltsangabe

Auditing ist das Überwachen und Aufzeichnen von ausgewählten Aktionen, die innerhalb der Datenbank stattfinden. Es kann auf Einzelnen oder auf einer Kombination von Faktoren (z. B. Nutzername, Anwendung, Anmeldezeit, usw.) basieren. Auditing üblicherweise für die folgenden Zwecke genutzt:

- Abschreckung von Nutzern, vor unerlaubten Zugriffen auf Objekte, außerhalb ihres Verantwortungsbereiches.
- Daten über bestimmte Aktivitäten in der Datenbank sammeln
- Verdächtige Nutzeraktivitäten aufdecken
- Aufdecken von Problemen mit Autorisations- und Zugriffskontrollmechanismen

In Microsoft SQL Server kann das Auditing auf zwei Ebenen stattfinden, als Serverüberwachung oder als Datenbanküberwachung.



Datenbanküberwachungen sind nicht in allen SQL Server Editionen verfügbar. Sie existieren lediglich in der Enterprise, der Developer oder der Eval-Edition.

21.1 Bestandteile des SQL Server Auditings

21.1.1 Überwachungen

Überwachungen sind Objekte die auf Server-Instanzebene erstellt werden. Sie regeln verschiedene Dinge, wie z. B.:

- Was passiert, wenn das Auditing nicht durchgeführt werden kann?
- Wo werden die Auditing-Daten gespeichert?
- Maximale Größe/Anzahl der Auditing-Dateien?

Überwachungen stellen die Grundlage für das Auditing dar. Alle weiteren Komponenten basieren auf Ihnen und können deshalb auch nur dann erstellt werden, wenn eine Überwachung existiert.

21.1.2 Serverüberwachungsspezifikationen

In einer Serverüberwachungsspezifikation kann der Administrator eine Liste von Überwachungsaktionsgruppen erstellen, um so verschiedene Ereignisse/Aktionen auf Server-Instanzebene zu überwachen.

Nach Aktivierung der Serverüberwachungsspezifikation werden die gesammelten Überwachungsdaten an eine Überwachung gesendet, die diese wiederum in einem Überwachungsziel speichert.



Überwachungen und Serverüberwachungsspezifikationen stehen in einer 1:1-Beziehung zueinander. Jede Überwachung kann nur mit genau einer Serverüberwachungsspezifikation verbunden werden und eine Serverüberwachungsspezifikation ist ohne Überwachung nicht funktionsfähig.

21.1.3 Datenbanküberwachungsspezifikationen

Datenbanküberwachungsspezifikationen sind in ihrer Arbeits- und Funktionsweise mit den Serverüberwachungsspezifikationen identisch. Sie zeichnen Überwachungsaktionen auf und leiten diese an eine Überwachung weiter.



Genau wie bei den Serverüberwachungsspezifikationen gilt auch hier, dass zwischen Datenbanküberwachungsspezifikation und Überwachung eine 1:1-Beziehung existiert.

21.1.4 Überwachungsaktionen und Überwachungsaktionsgruppen

Unter Überwachungsaktionen bzw. Aktionsgruppen versteht man einzelne Ereignisse und Gruppen von Ereignissen, die im SQL Server auftreten und überwacht werden können. Sie existieren auf Server- und Datenbankebene, außerdem gibt es Überwachungsereignisse, die zur Überwachung des Auditing notwendig sind.



Diese Ereignisse bzw. Gruppen basieren auf den „Extended Events“, welche noch an späterer Stelle in dieser Unterlage behandelt werden.



- [cc280663]

21.1.5 Überwachungsziele

Unter dem Begriff „Überwachungsziel“ versteht der SQL Server einen Speicherort für die Auditing-Datensätze. Hierfür stehen drei Möglichkeiten zur Verfügung:

- **Datei:** Die Auditing-Datensätze werden in einer Datei bzw. eine Gruppe von Dateien auf einem Datenträger abgelegt.
- **Anwendungsprotokoll:** SQL Server kann seine Auditing-Daten in das zentrale Auditing-Protokoll des Betriebssystems, das Anwendungsprotokoll schreiben.
- **Sicherheitsprotokoll:** SQL Server kann seine Auditing-Daten in das Sicherheitsprotokoll, eine gut geschützte Alternative zum Anwendungsprotokoll, schreiben.

21.2 Erstellen und verwalten von Überwachungen

21.2.1 Konfiguration und Benutzung von Überwachungszielen

Für die Erstellung einer Überwachung muss ein Überwachungsziel ausgewählt werden. Dieses Ziel bestimmt, wo die Überwachungsdaten gespeichert werden.

Ein Dateipfad als Überwachungsziel

Datensätze in das Anwendungsprotokoll schreiben

Das Windows-Anwendungsprotokoll ist eine zentrale Sammelstelle für Ereignisse des Betriebssystems und anderer Softwareprodukte. Grundsätzlich kann jede Software auf unkomplizierte Weise ihre Einträge in das Anwendungsprotokoll schreiben und jeder Benutzer hat lesenden Zugriff darauf. Es ist kein zusätzlicher Konfigurationsaufwand notwendig, um dieses Protokoll zu benutzen. Daraus resultiert, dass das Anwendungsprotokoll zwar einfach und praktisch zu Handhaben, jedoch auch sehr unsicher ist.



Dieses Überwachungsziel ist ungeeignet, wenn der Administrator des SQL Servers und der „Sicherheitsbeauftragte“ zwei unterschiedliche Personen sind, da der SQL Server Administrator nicht am Zugriff auf das Anwendungsprotokoll gehindert werden kann.

Benutzung des Sicherheitsprotokolls vorbereiten

21.2.2 Erstellen von Überwachungen mit dem SSMS

Erstellen der Überwachung

- Wählen Sie im Objekt-Explorer das Menü „Sicherheit“ auf Instanzebene und öffnen Sie dort das Kontextmenü des Punktes „Überwachungen“.

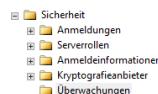


Abb. 21.1:
Erstellen einer
Überwachung

- Klicken Sie auf „Neue Überwachung...“

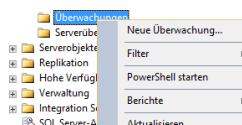


Abb. 21.2:
Erstellen einer
Überwachung

- Es öffnet sich der Dialog „Überwachung erstellen“

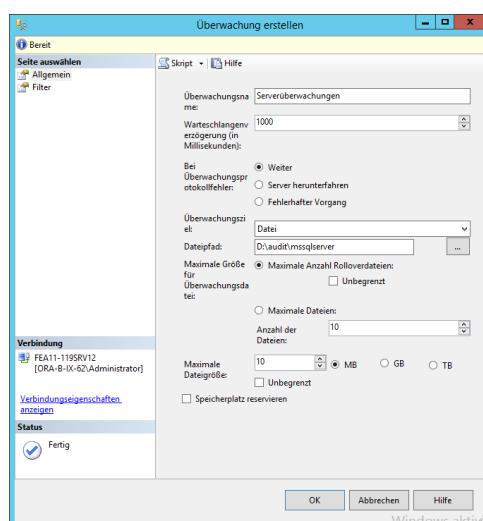


Abb. 21.3:
Erstellen einer
Überwachung

- Tragen Sie die folgenden Angaben ein:

- **Überwachungsname:** Serverüberwachungen
- **Überwachungsziel:** Datei
- **Dateipfad:** D:\audit\mssqlserver (Pfad vorher anlegen)
- **Überwachungsdateien:** Option „Unbegrenzt“ entfernen
- **Anzahl der Dateien:** 10
- **Maximale Dateigröße:** 10 MB

5. Klicken Sie auf OK.

Bedeutung der Optionen für das Überwachungsziel

- **Dateipfad:** In diesem Verzeichnis wird ein Ordner angelegt, welcher die Überwachungsdaten aufnimmt.
- **Anzahl der Dateien:** Durch diese Angabe wird festgelegt, nach wie vielen Dateien der SQL Server anfängt, die erste Datei wieder zu überschreiben (Rollover). Dadurch wird eine zyklische Wiederverwendung der Audit-Dateien erreicht.
- **Maximale Dateigröße:** Gibt die maximale Größe einer Audit-Datei an.

Bedeutung der Optionen für Überwachungsprotokollfehler

Es kann vorkommen, dass der SQL Server aus einem unbekannten Grund nicht mehr in der Lage ist die Daten des Auditings zu speichern. Da Auditing aber meist dazu benutzt wird, um sicherheitskritische Systeme zu überwachen, ist der Ausfall des Überwachungsziels selbst auch als kritisch zu bewertender Vorfall. Der Administrator hat drei verschiedene Optionen zur Verfügung, um das Verhalten des SQL Servers in einem solchen Fall zu beeinflussen und angemessen auf den Ausfall zu reagieren.

- **Weiter:** Das Auditing läuft weiter und der SQL Server versucht auch weiterhin Daten in das Überwachungsziel zu schreiben. Diese Option ermöglicht es unter Umständen, dass Benutzer Aktivitäten auf dem Server ausführen, welche durch Ihre Sicherheitsrichtlinien verboten sind, ohne dass dies protokolliert werden kann.



Wählen Sie diese Option nur dann aus, wenn die Verfügbarkeit des Datenbankmoduls wichtiger ist als die Sicherheit des Systems.

- **Server herunterfahren:** Ist diese Option ausgewählt, wird der Server automatisch beim Ausfall des Überwachungszieles heruntergefahren.



Das Herunterfahren des Servers bietet zwar ein hohes Maß an Sicherheit, es kann aber auch für Denial-Of-Service Attacken genutzt werden. Verwenden Sie diese Option nur dann, wenn die Sicherheit des Servers höher bewertet wird, als die Verfügbarkeit.

- **Fehlerhafter Vorgang:** Diese Option stellt einen Kompromiss aus „Weiter“ und „Server herunterfahren“ dar. Der Server blockiert alle Aktionen, welche eine Überwachung nach sich ziehen würden. Andere nicht überwachte Aktivitäten sind problemlos möglich. Der Server bleibt verfügbar.



- [cc280525]

21.2.3 Erstellen von Überwachungen mit SQL

Wahlweise kann eine Überwachung auch mittels SQL-Statement erstellt werden.

Listing 21.1: Erstellen einer Überwachung

```
USE [master]
GO

CREATE SERVER AUDIT [Server\"überwachungen"]
TO FILE
( FILEPATH = N'D:\audit\mssqlserver',
  MAXSIZE = 10 MB,
  MAX_ROLLOVER_FILES = 10,
  RESERVE_DISK_SPACE = OFF
)
WITH
( QUEUE_DELAY = 1000,
  ON_FAILURE = CONTINUE
);

GO
```

Beispiel ?? zeigt, wie die Überwachung „Serverüberwachung“, die zuvor mit Hilfe des SSMS erstellt wurde mittels SQL-Kommando kreiert wird.

21.2.4 Überwachungsziel Datei

NTFS-Berechtigungen

21.2.5 Überwachungsziel Anwendungsprotokoll

21.2.6 Überwachungsziel Sicherheitsprotokoll

21.3 Auditing auf Serverebene

21.3.1 Serverüberwachungsspezifikationen

21.4 Auditing auf Datenbankebene

22 Backup und Recovery

Inhaltsangabe

22.1 Datenbank-Snapshots

DB-Snapshots (S. 155)

Verschieben von Systemdatenbanken nach einem Hardwarefehler

Nach einem Hardwarefehler kann es sein, dass Datendateien von Systemdatenbanken beschädigt wurden oder verloren gegangen sind. In so einem Fall kann es erforderlich sein, den Speicherort einer Systemdatenbank auf einen anderen Datenträger zu verschieben und die Datenbank dann dort wiedherzustellen. Problematisch dabei ist, dass die SQL Server-Instanz nicht gestartet werden kann, wenn nicht alle Systemdatenbanken verfügbar und voll funktionsfähig sind. Hier hilft dem Administrator das Traceflag 3608 weiter.

1. Starten des SQL Server-Dienstes mit minimaler Konfiguration und mit dem Traceflag 3608.

Listing 22.1: Starten des SQL Server unter Umgehung des automatischen Recoveries

```
NET START MSSQLSERVER /f /T3608
```



Das Traceflag 3608 sorgt dafür, dass der SQL Server beim Start kein Recovery an seinen Datenbanken durchführt. Einzige Ausnahme ist die MASTER-Datenbank.

2. Führen Sie ein `ALTER DATABASE ... MODIFY FILE`-Kommando für alle Daten- und Log-Dateien der MSDB-Datenbank durch und geben Sie dabei den neuen Pfad der Dateien an.
3. Stop Sie die SQL Server-Instanz
4. Verschieben Sie die Dateien an ihren neuen Speicherort
5. Starten Sie die SQL Server-Instanz erneut (ohne /f und /T)
6. Benutzen Sie Katalogsicht `SYS.MASTER_FILES`, um das Ergebnis des Verschiebevorganges zu verifizieren.

Listing 22.2: Abfragen der View `SYS.MASTER_FILES`

```
USE
master GO

SELECT
```

```
name AS logical_name
, physical_name AS new_location
, state_desc as state
FROM sys.master_files
WHERE database_id = DB_ID('msdb')
```

- [ms188396]



<https://technet.microsoft.com/en-us/library/jj853293.aspx>

Teil IV

Appendix

Microsoft SQL Server