

Datenbankadministration

Oracle 11g

Florian Weidinger

1. Ausgabe: 09.05.2007

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.

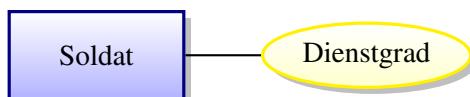
1.3.1 Definitionen und Syntaxregeln



- **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.

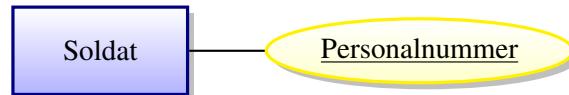


Um ein Modell übersichtlich und verständlich zu halten, sollte konsequent immer nur eine der drei zur Verfügung stehenden Methoden für die Identifizierung von Objekten genutzt werden!

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

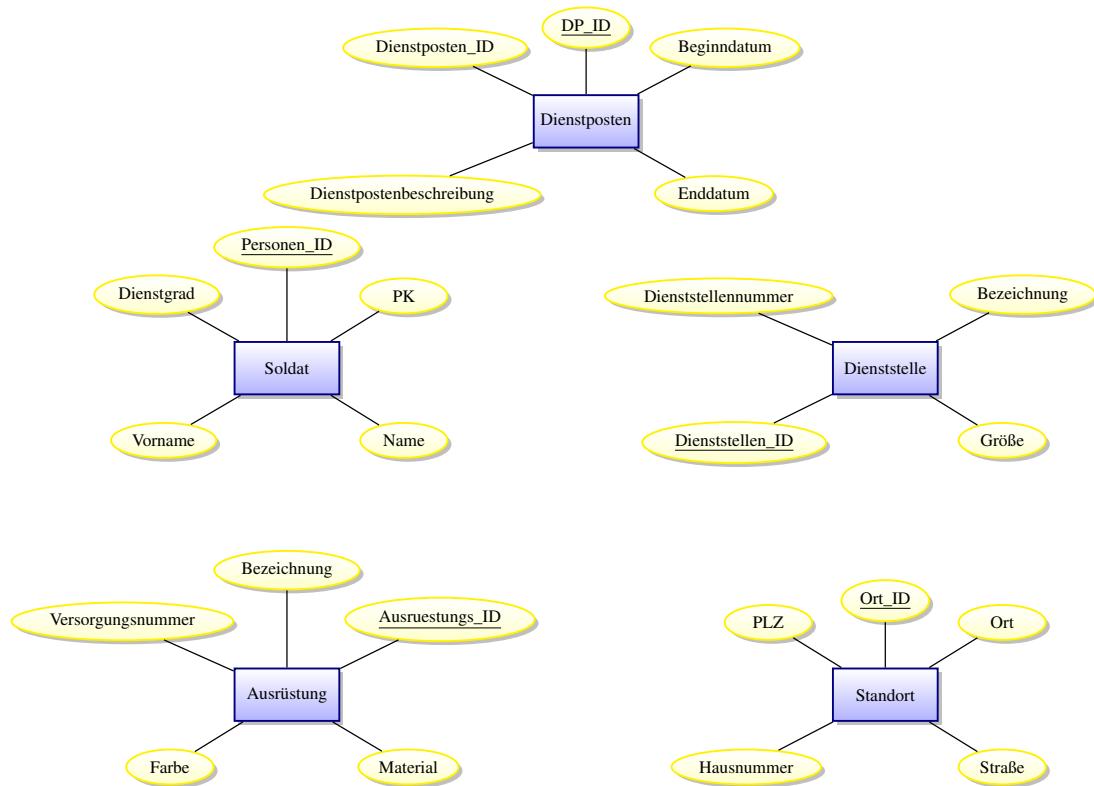
An dieser Stelle wird die Entscheidung getroffen, dass für die Fortführung dieses Modells organisatorische Einheiten als identifizierende Eigenschaften eingeführt werden!

- 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

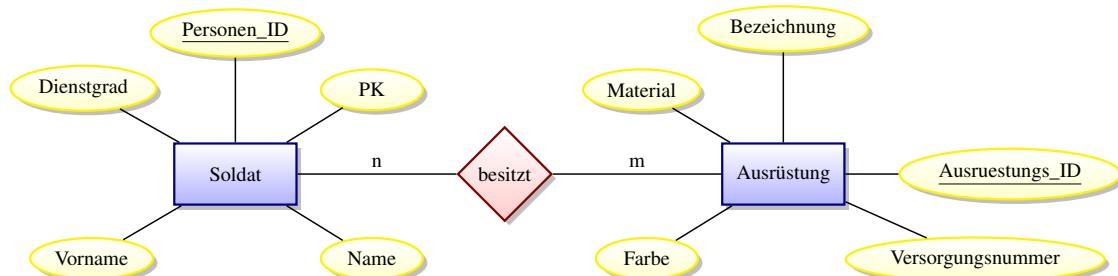
2.1.3 Kardinalität

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)

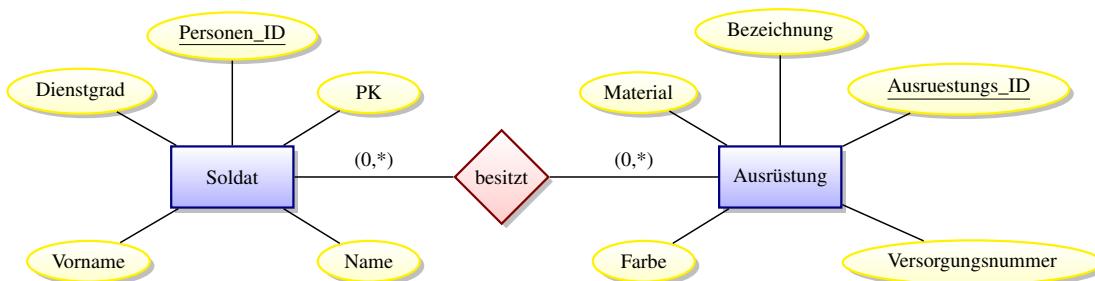
Welche Werte können angenommen werden bzw. wofür stehen die Buchstaben?

- 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.

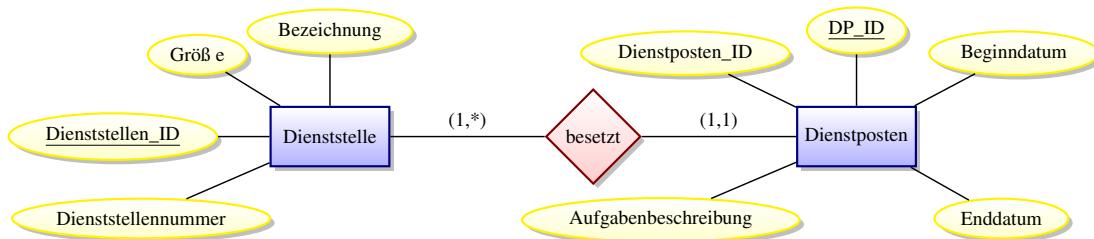
- Die (Min,Max)-Notation: Man betrachtet zunächst ein Objekt a auf der Seite des Objekttyps A und schreibt die Kardinalität auf die gleiche Seite des Beziehungstyps, an den Objekttyp A.
- (Chen)-Notation: Inverse Bezeichnung der Kardinalitäten wie bei der (Min,Max)-Notation. Die Kardinalität des Objekts 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.

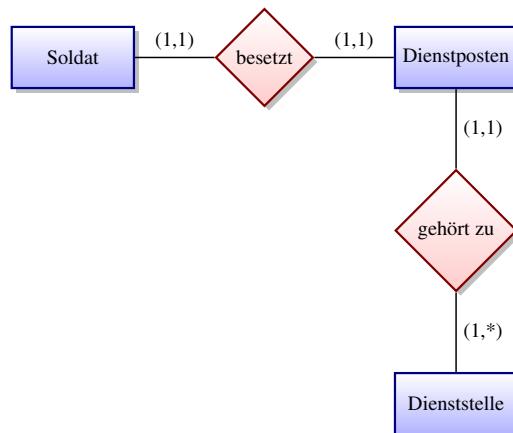
Nun können die Brücken aber so angelegt sein, dass es von Dateninsel A nach Dateninsel B zwei (oder mehr) verschiedene Wege gibt. Sind dann eine oder mehrere Brücken überflüssig? Bei der Datenmodellierung spricht man in solchen Fällen von **redundanten Beziehungstypen**. Das sind Beziehungstypen, die 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.

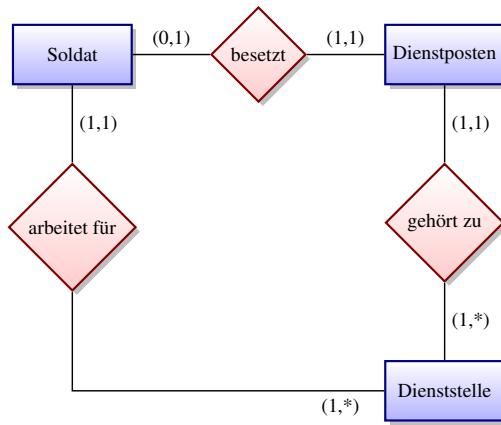
Der Verdacht auf einen redundanten Beziehungstyp ergibt sich i.d.R. bei zyklischen Beziehungstyp-Strukturen. Kann man nun aber allein aus strukturellen Merkmalen des Datenmodells die Redundanz ableiten? Wenn dies so wäre, könnten automatisierte Optimierungsprozesse diese Redundanz wieder entfernen. Zur 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.

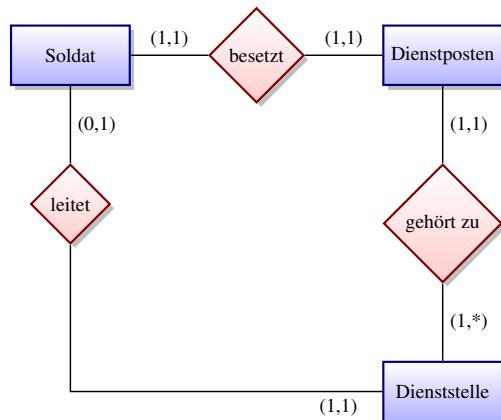
Nun kommt die „arbeitet für“-Beziehung hinzu, so dass auf Grund der entstehenden zyklischen Struktur zwei Wege vom Soldaten zur Dienststelle führen. Ist dieser Beziehungstyp nun redundant? Betrachten wir zwei verschiedene Interpretationen dieses Beziehungstyps.

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.

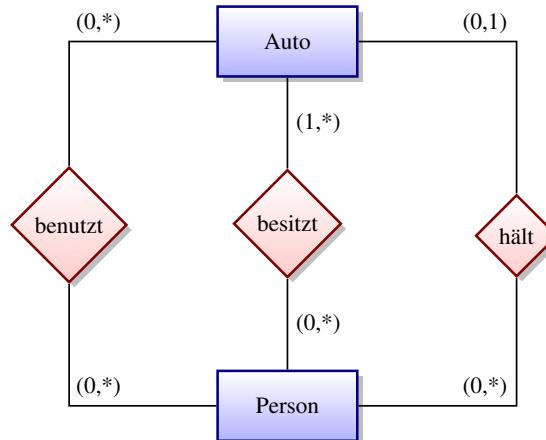
2.4 Parallelle Beziehungstypen

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?

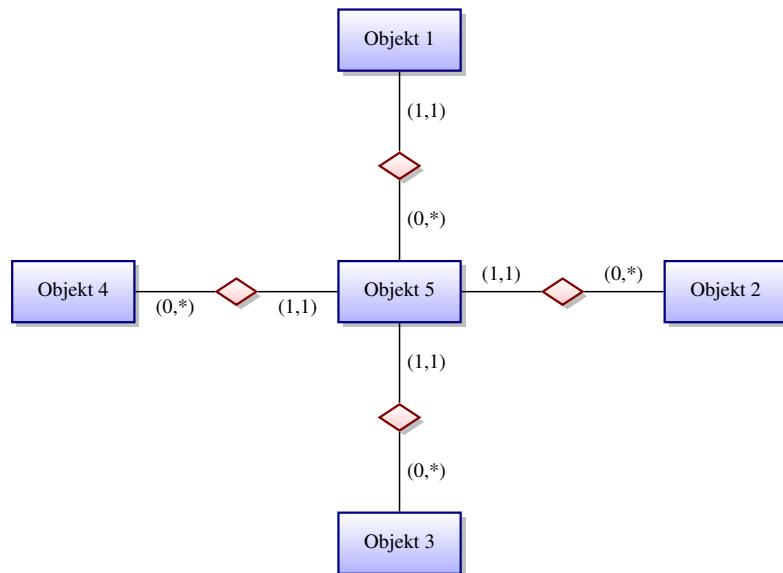
Nehmen wir an Sie wollen wie in der folgenden Abbildung dargestellt, für eine Personengruppe und eine definierte Menge von Autos drei Beziehungstypen modellieren.

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.

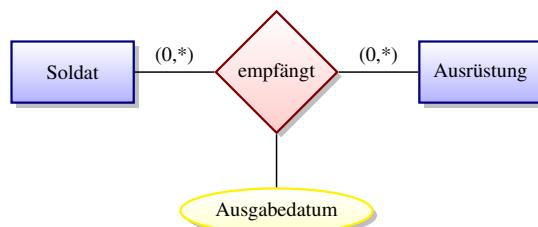


Der Objekttyp 5 steht hier mit vier anderen Objekttypen in Beziehung. Auch die übrigen Objekttypen können mit anderen Objekttypen in Beziehung stehen. Eine evtl. Problematik ergibt sich erst durch die Transformation der Beziehungstypen, da bei diesem Prozess weitere Fremdschlüssel, in die dann entstandenen 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

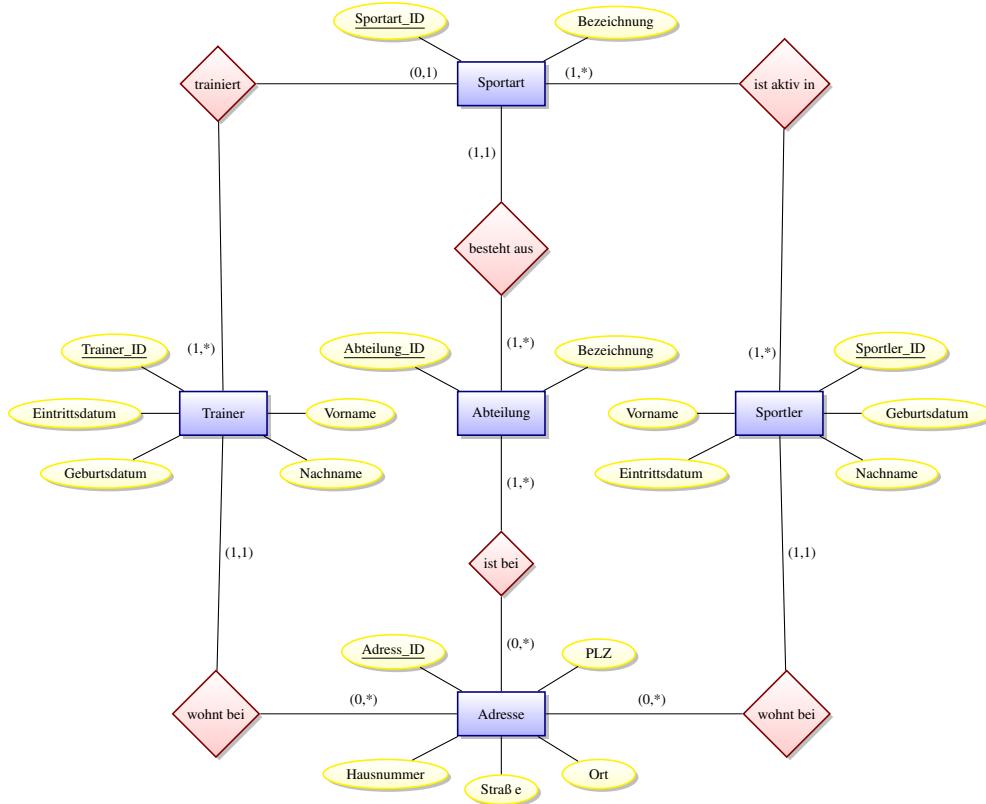
Entwerfen Sie, basierend auf der folgenden Lage, ein ER-Modell, inklusive der Beziehungen zwischen den Entitäten.

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

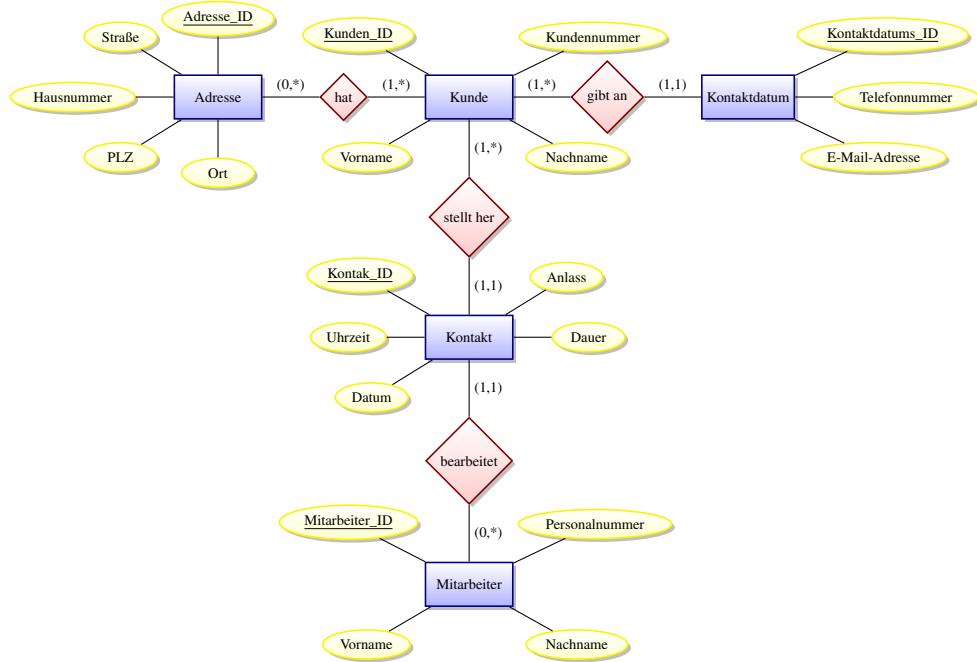


Transformation

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

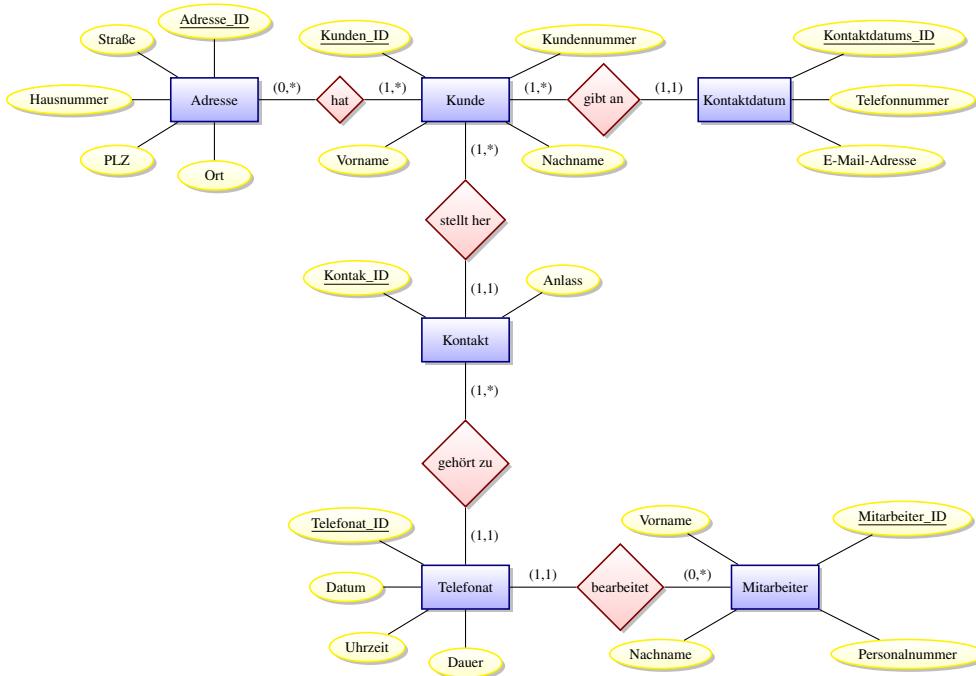
Vorgaben



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)

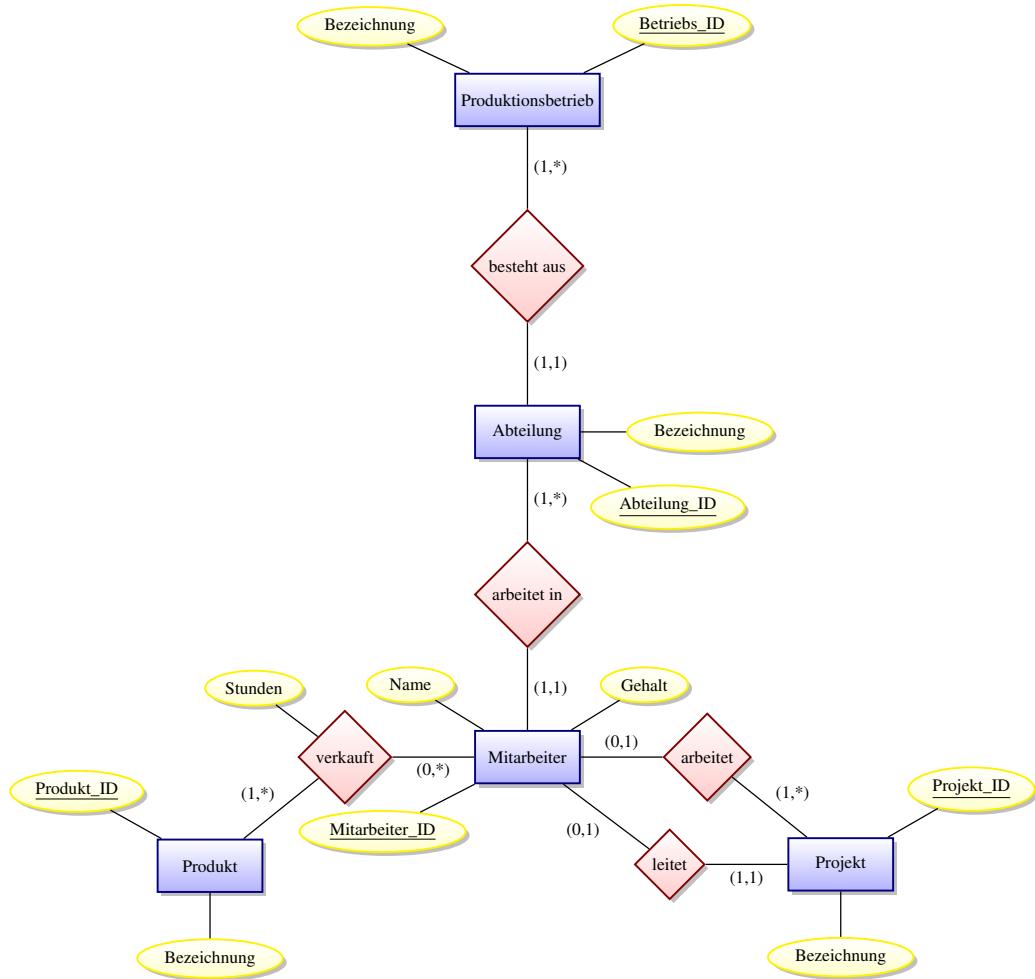
Zusatzaufgabe



Transformation

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

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

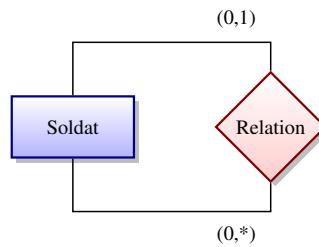
Bei der traditionellen Informationsspeicherung, mit Hilfe von Karteikärtchen, wird der rekursive Beziehungstyp durch Querverweise, innerhalb des Karteikastens, realisiert. Im betrachteten Vertretungsbeispiel würden auf der Karteikarte des Mitarbeiters die Personalnummern seiner möglichen Vertreter notiert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

3.1.2 Syntaxregeln für rekursive Beziehungstypen

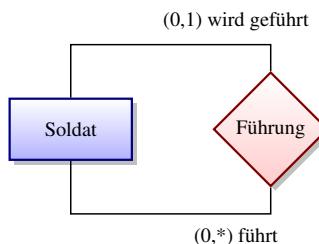
- Ein rekursiver Beziehungstyp zur Kennzeichnung eines sachlogischen Zusammenhangs zwischen den Objekten ein- und desselben Objekttyps A wird als eine Verbindungsline dargestellt, die den

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.

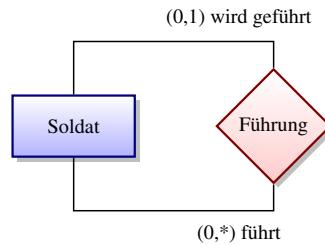
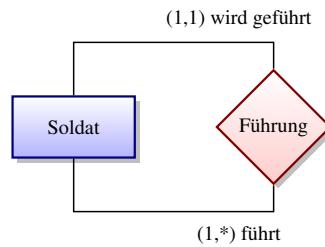
Tabelle 3.1: Mögliche Kombinationen von rekursiven Beziehungstypen

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:

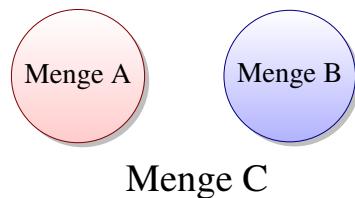
1. Vollständige Überdeckung und Überlappung nicht zugelassen (= disjunkt, kein gemeinsames Element)

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

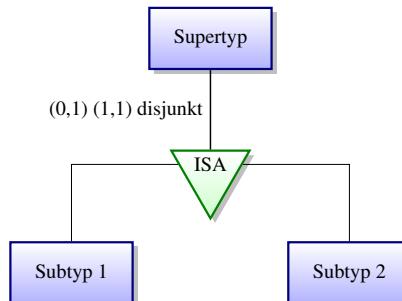
Wenn man die Objektmengen dieser 1. Beziehungsart grafisch darstellt, ergibt sich diese Abbildung:



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

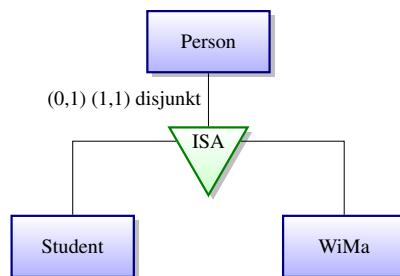
Tabelle 3.3: Student

Person_ID	Matrikelnummer
1	65420
4	66530

Tabelle 3.4: WiMa

Person_ID	Gehalt
2	3000
3	3150

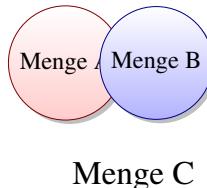
Zusammenfassend ergibt sich für den 1. Fall die folgende Abbildung:



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

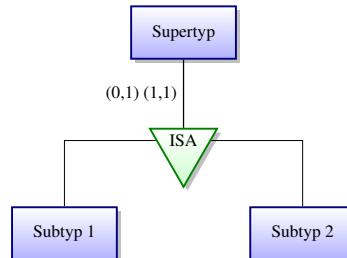
Wenn man die Objektmengen dieser 2. Beziehungsart grafisch darstellt, ergibt sich die folgende Abbildung:



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

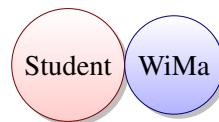
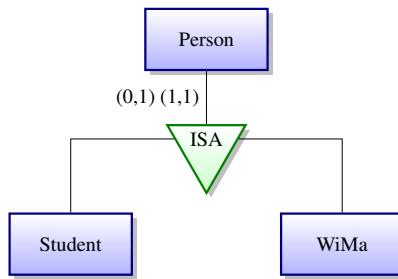
Tabelle 3.6: Student

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:

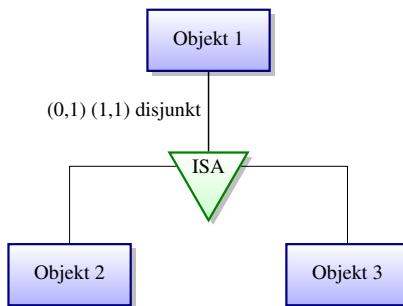
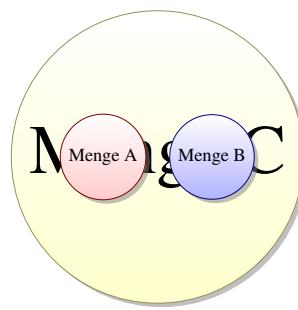


3.3.5 Nicht vollständige Überdeckung und Disjunkt

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

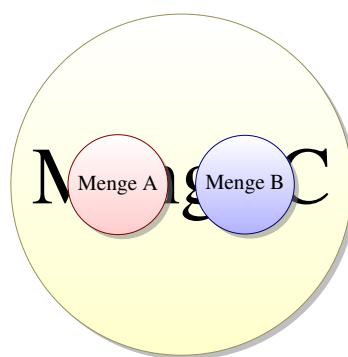
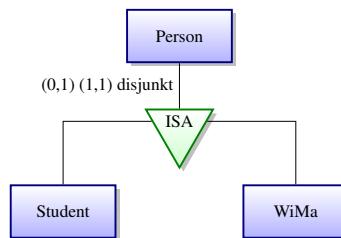
Person_ID	Matrikelnummer
1	65420
4	66530
5	66460

Tabelle 3.10: WiMa

Person_ID	Gehalt
2	3000
3	3150

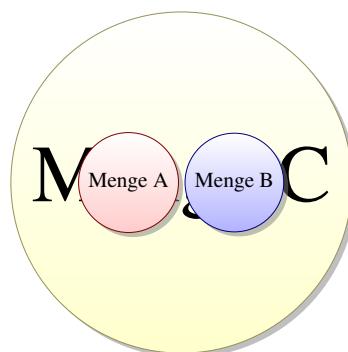
Die Person mit der Person_ID 6 kommt in den Subtypen nicht vor, da sie ein Praktikant ist und nur die Attribute des Supertyps in sich vereint.

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

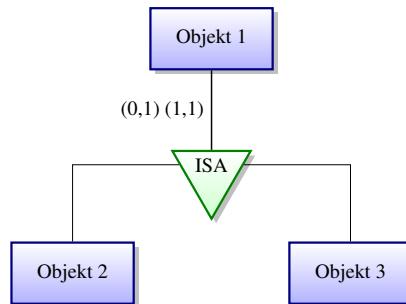


3.3.6 Nicht vollständige Überdeckung und nicht Disjunkt

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

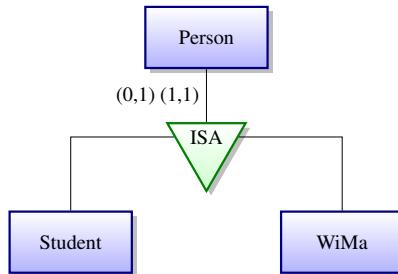
Tabelle 3.13: WiMa

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:



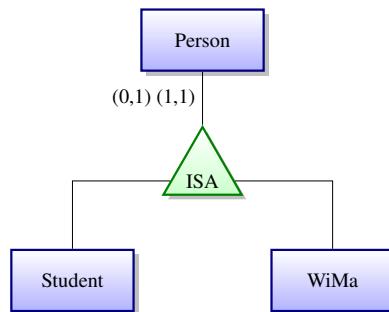
3.4 Generalisierung

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.

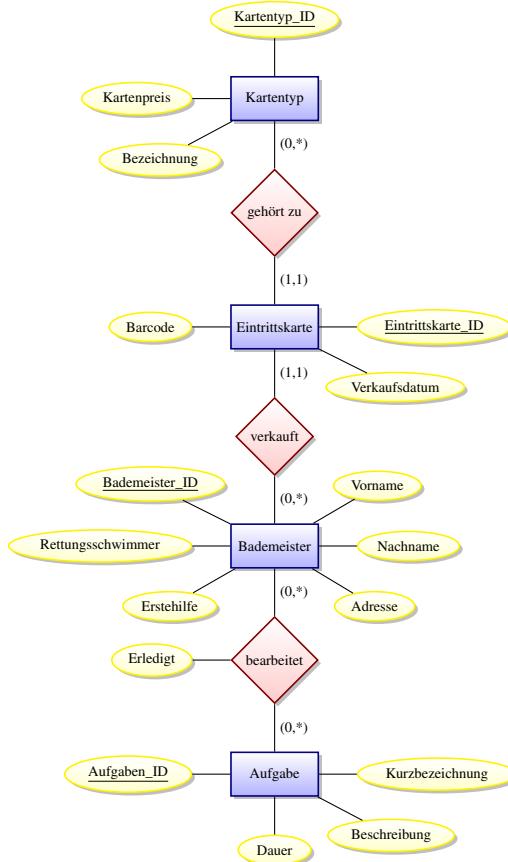
Wird eine „normale“ Behandlung an einem Diensthund durchgeführt, so besteht diese aus mindestens einer oder mehreren Behandlungspositionen (Operation, Medikamentengabe, usw.) die einzeln zu speichern sind. Dabei ist eine erfasste Behandlungsposition immer eindeutig einer Behandlung eines Diensthundes 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

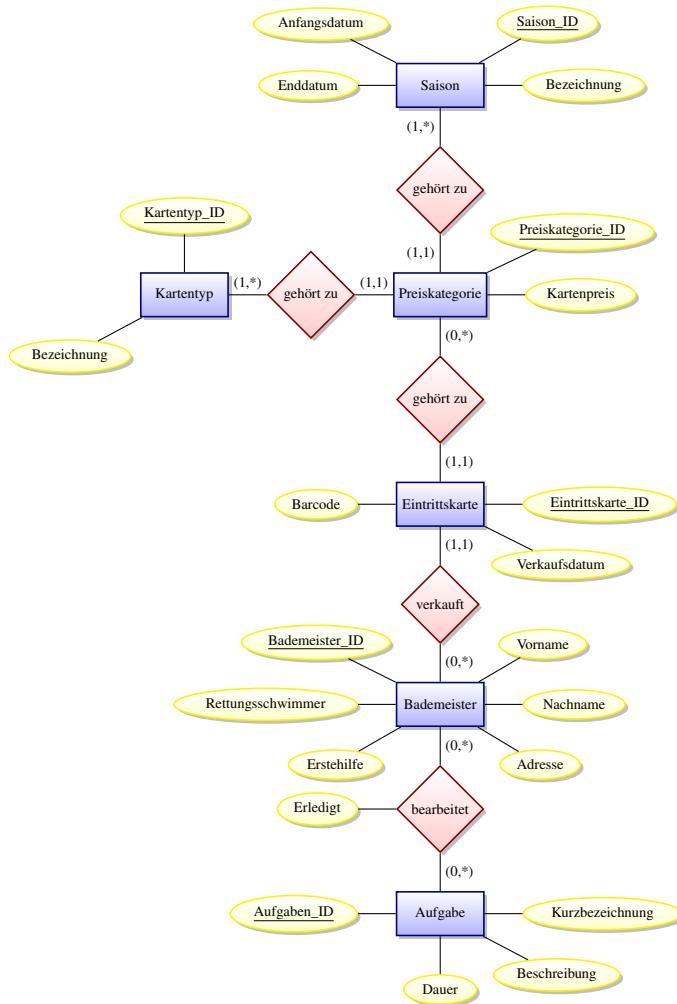
Vorgaben



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



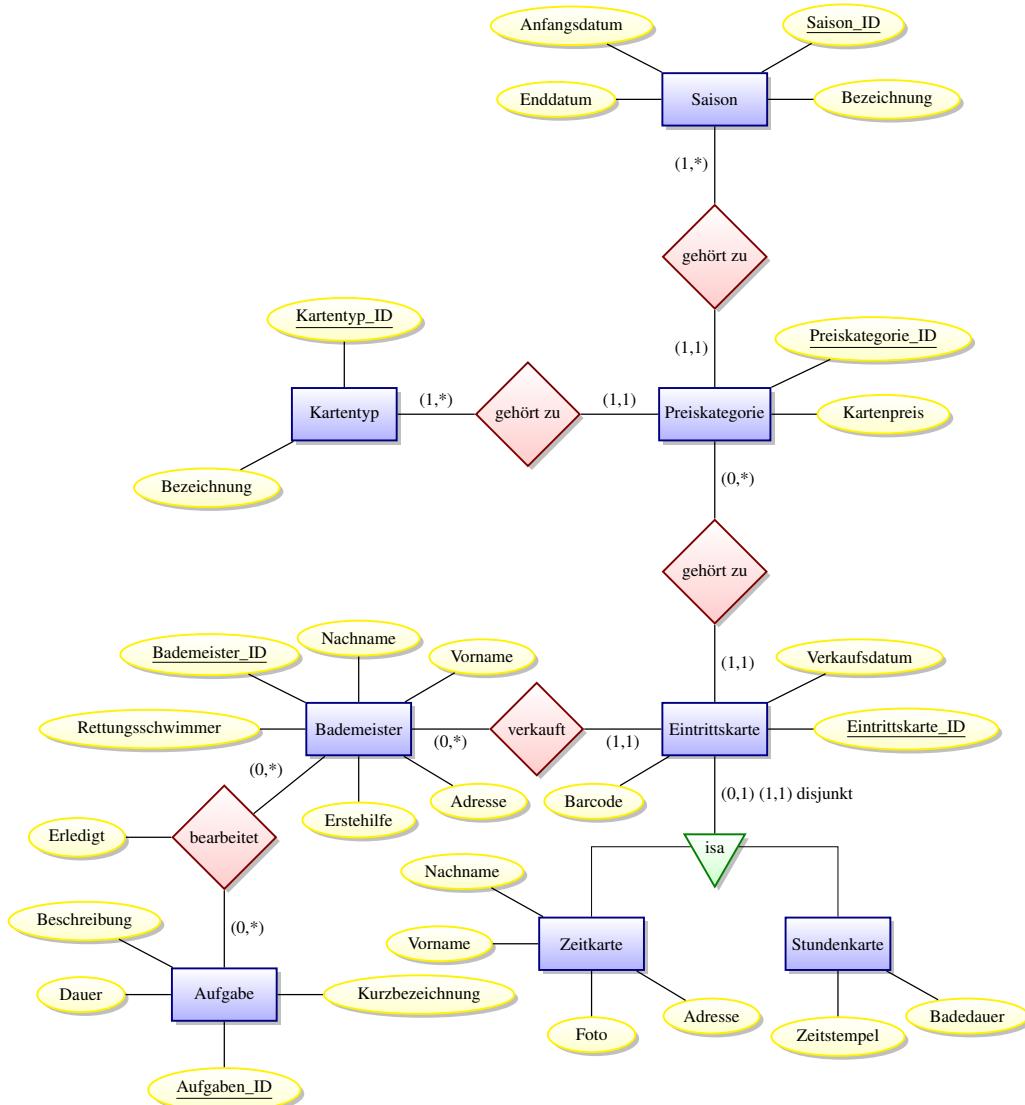
Transformation Zusatzaufgabe 1

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

Kartentyp
Preiskategorie
Eintrittskarte
Saison

(Kartentyp_ID
(Preiskategorie_ID
(Eintrittskarte_ID
↑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

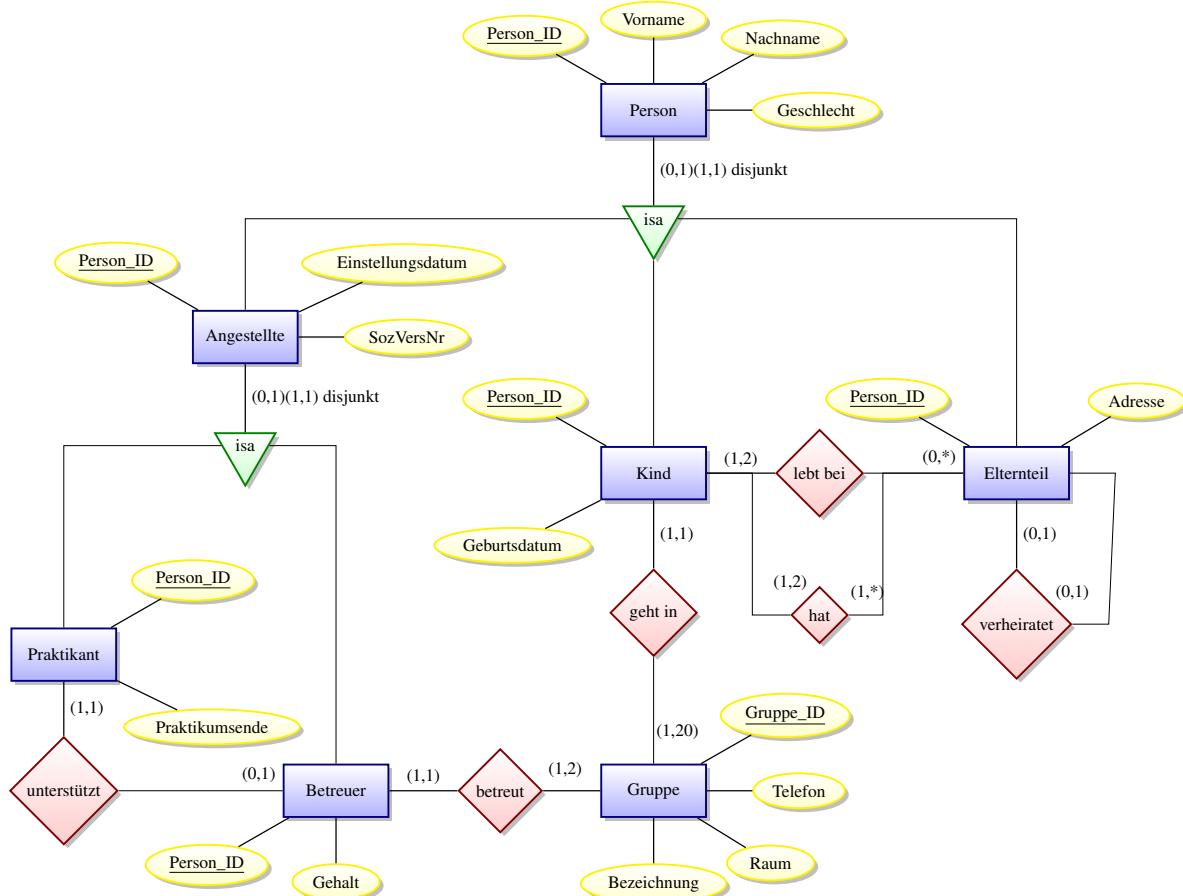
Personal (Personal_ID, Vorname, Nachname, Adresse)

Bademeister (\uparrow Personal_ID \uparrow , Rettungsschwimmer, Ersthilfe)

Ehemalige (\uparrow Personal_ID \uparrow , Kündigungsdatum)

3.6.2 Übungsaufgabe Kindergarten

Vorgaben

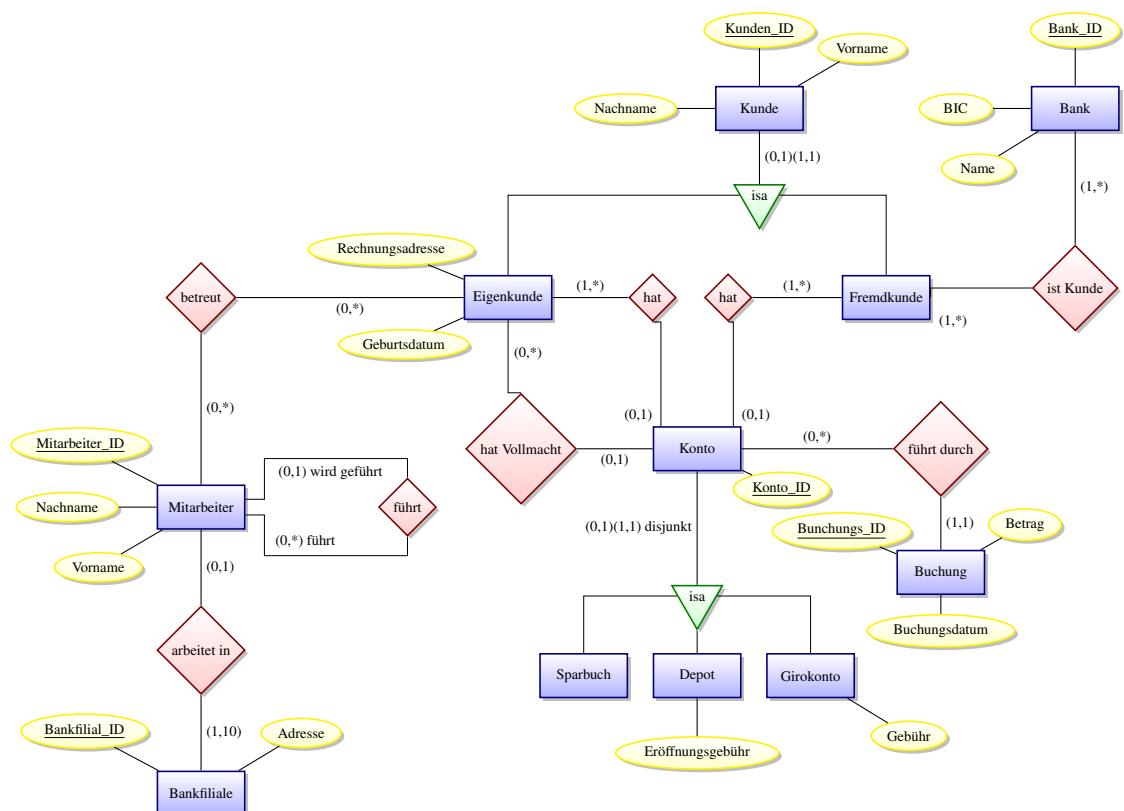


Transformation

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)

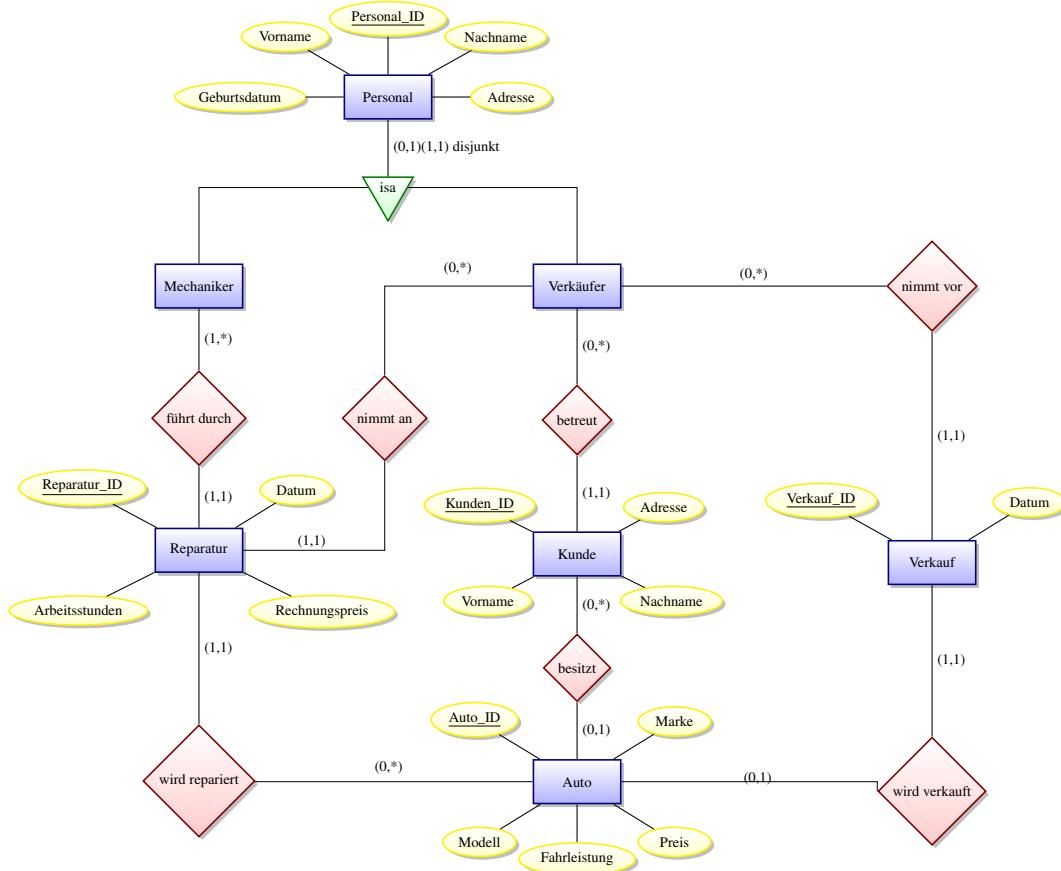
Transformation Variante 2

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

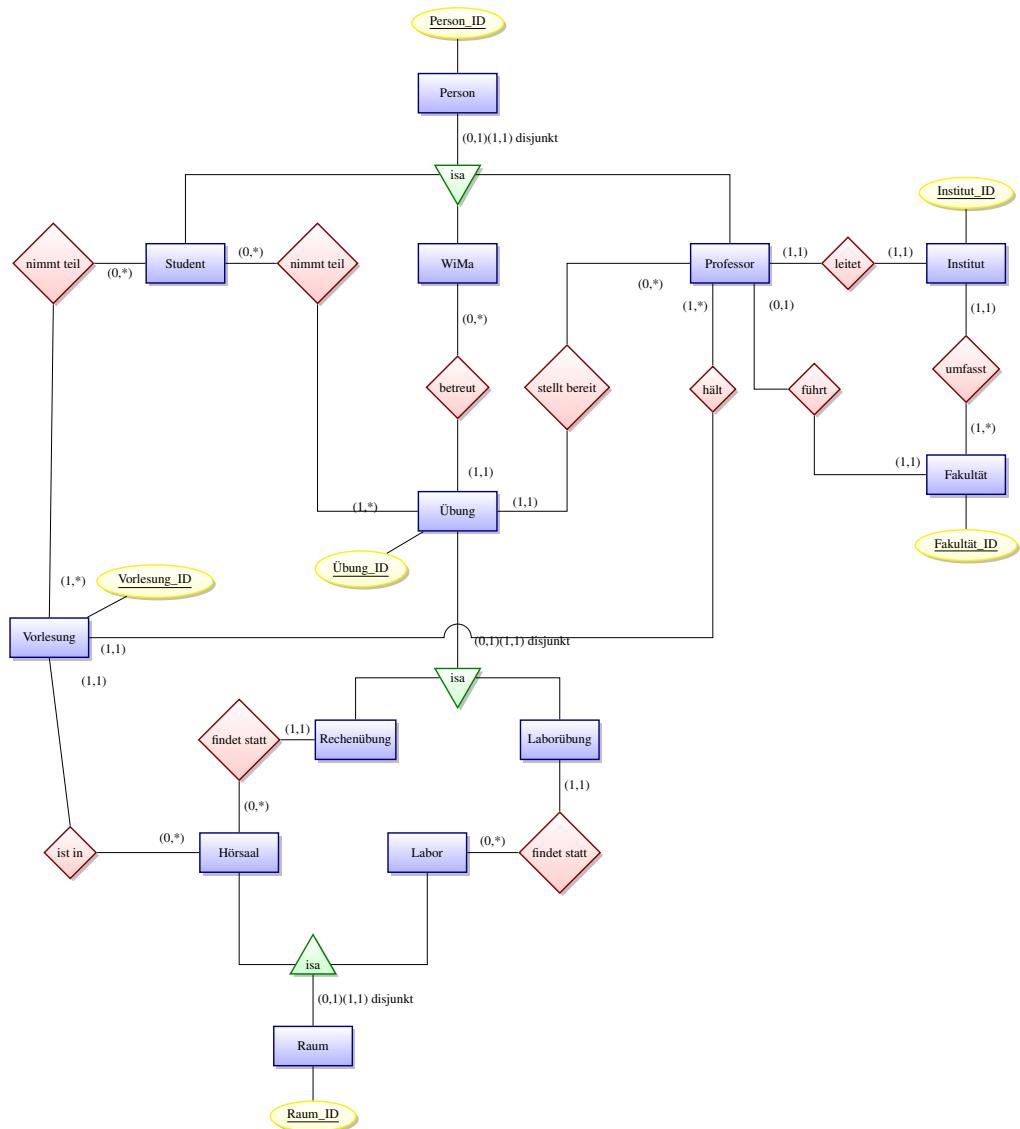


Transformation

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



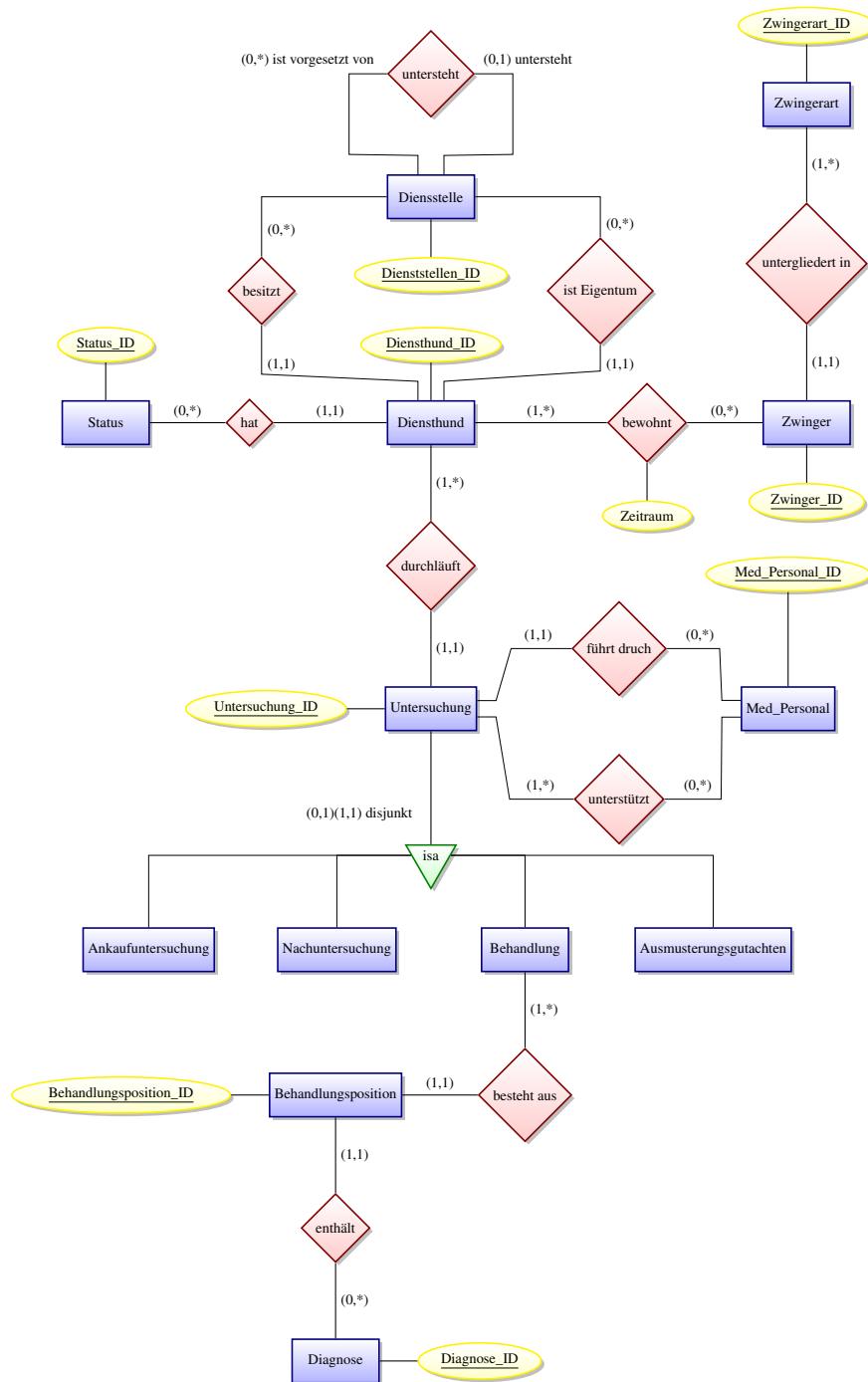
Aus Platzgründen sind nur die Primärschlüssel im Modell eingezeichnet.

Transformation

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

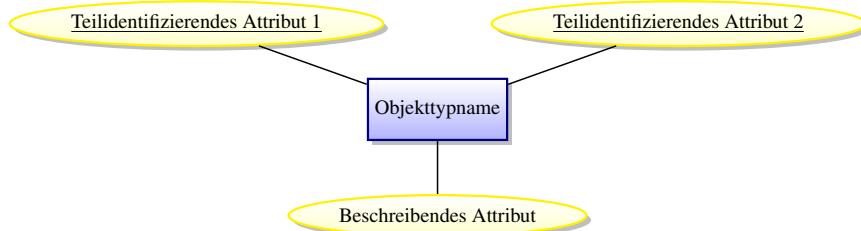
4.2 Transformation von Objekttypen

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



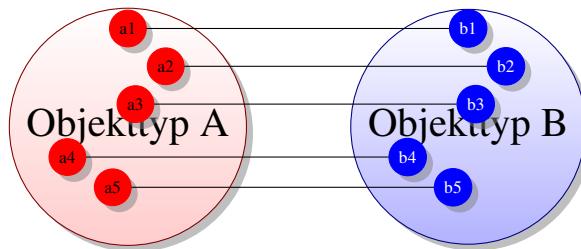
4.3 Transformation binärer Beziehungstypen

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.

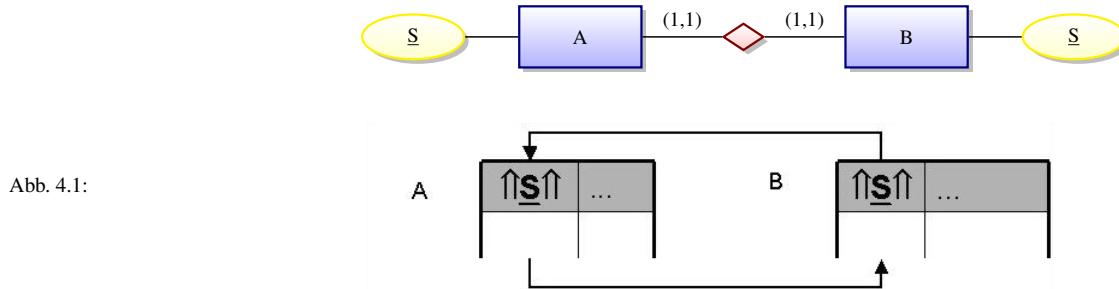
Es gibt aber auch Fälle, bei denen ein (1,1):(1,1) Beziehungstyp durchaus sinnvoll ist. Will man beispielsweise die Informationen über Objekte in öffentliche (z.B. Mitarbeiter-Offen(Personalnummer, Name, TelNr, Abteilung)) und vertrauliche Daten (z.B. MitarbeiterVS(Personalnummer, Gehalt, Konfession)) 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.

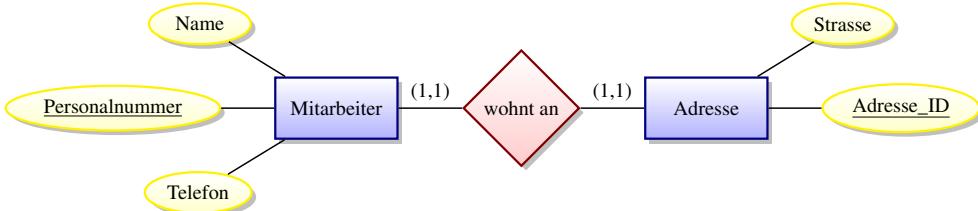


Eine Transaktion ist eine Folge von Operationen, bei der sichergestellt wird, dass entweder alle Operationen fehlerfrei beendet werden oder das keine der Operationen ausgeführt wird.

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.

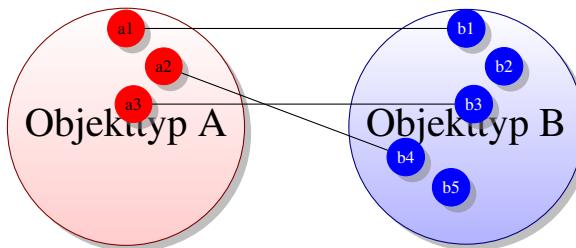


Mitarbeiter(Personalnummer, Name, Telefon, ↑Adresse_ID↑ [NN] [UN])

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

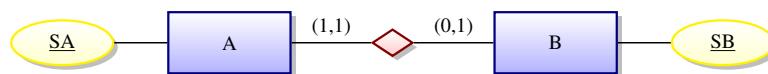
Fall 1: $\#A$ unwesentlich kleiner als $\#B$

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.

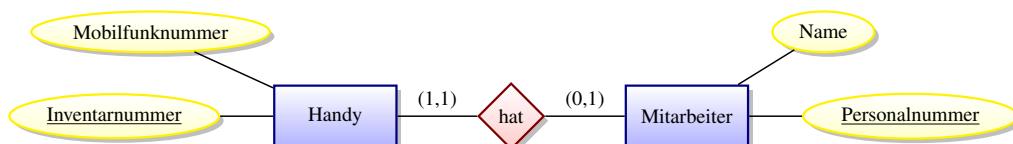
Transformationsregel T03 für den (1,1):(0,1) Beziehungstyp (selten realisierte Optionalität)

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.

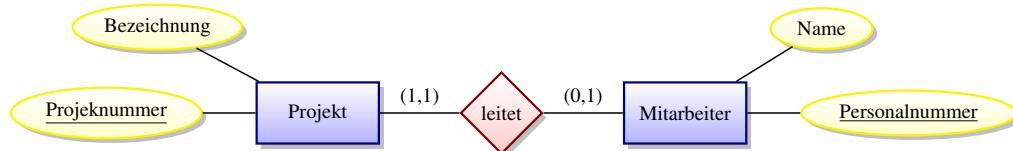
Transformationsregel T04 für den (1,1):(0,1) Beziehungstyp (oft realisierte Optionalität)

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.



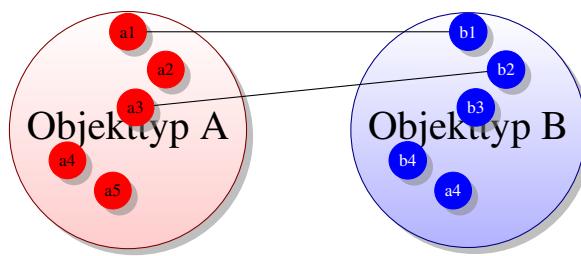
(A:) Projekt(Projektnummer, Bezeichnung, \uparrow Personalnummer \uparrow [NN, UN])

(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.

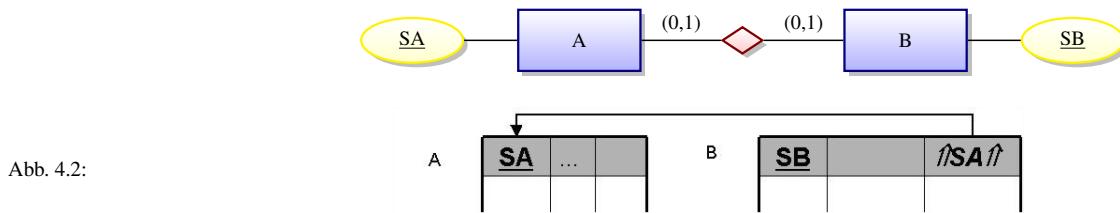


Wir nehmen für die folgenden Betrachtungen - ohne Beschränkung der Allgemeinheit - an, dass es höchstens so viele B-Objekte gibt wie A-Objekte ($\#A \geq \#B$, andernfalls müssen die Objekttypen einfach die Seiten tauschen). Da sowohl die A-Objekte als auch die B-Objekte unabhängig voneinander existieren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

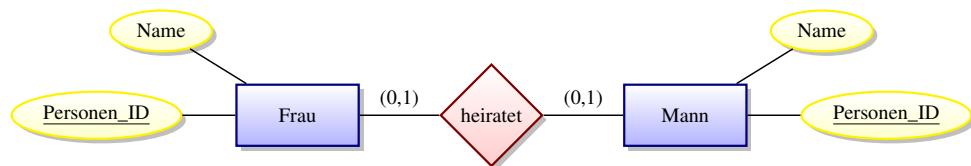
Transformationsregel T05 für den (0,1):(0,1) Beziehungstyp

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.



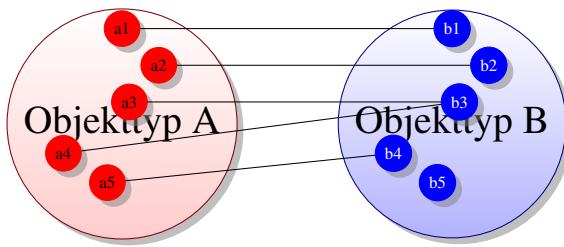
(A:) Frau(Personen_ID, Name)

(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

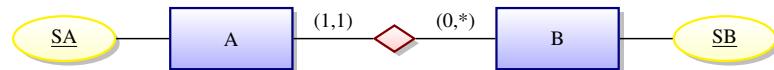
Beim $(1,1):(0,*)$ Beziehungstyp ist jedes Objekt B mit keinem, einem oder mehreren Objekten des Objekttyps A verbunden. Ein A-Objekt ist dagegen immer mit genau einem B-Objekt gekoppelt. Die folgende Abbildung zeigt dies schematisch.



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.

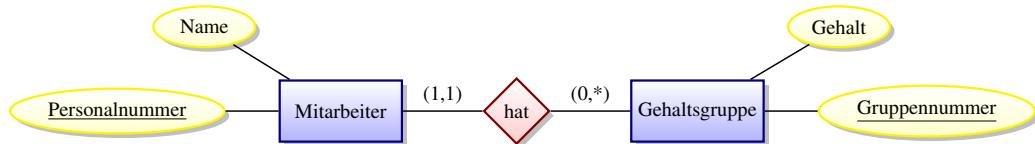
Transformationsregel T06 für den (1,1):(0,*) Beziehungstyp

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.



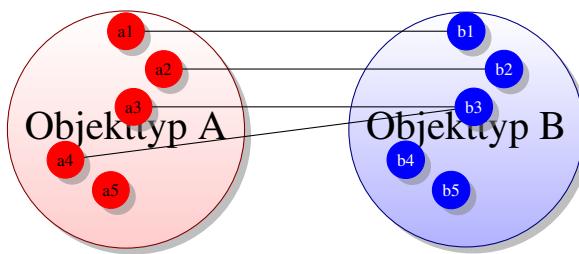
(B:) Gehaltsgruppe(Gruppennummer, Gehalt)

(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

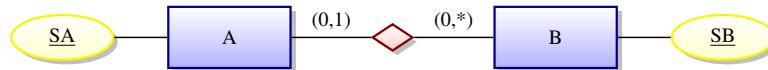
Beim (0,1):(0,*) Beziehungstyp kann ein Objekt des Typs B mit keinem, einem oder mehreren A-Objekten gekoppelt sein. Ein A-Objekt kann aber zu höchstens einem B-Objekt in Beziehung stehen.



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

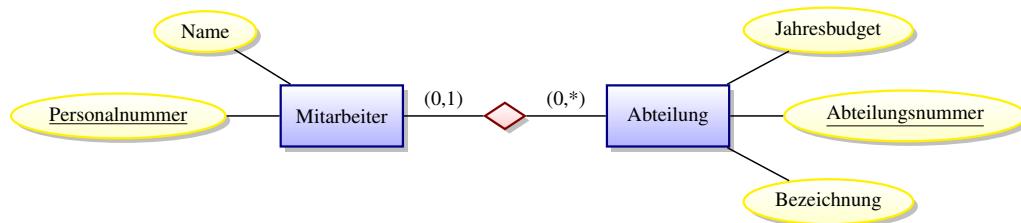
Transformationsregel T07 für den (0,1):(0,*) Beziehungstyp

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)

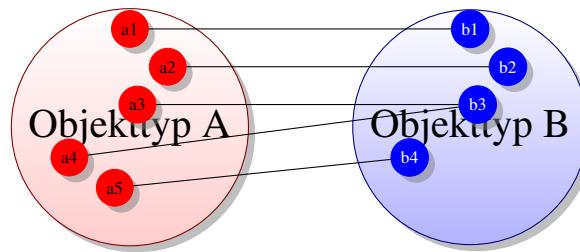
(A:) Mitarbeiter(Personalnummer, Name, \uparrow Abteilungsnummer \uparrow)

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.



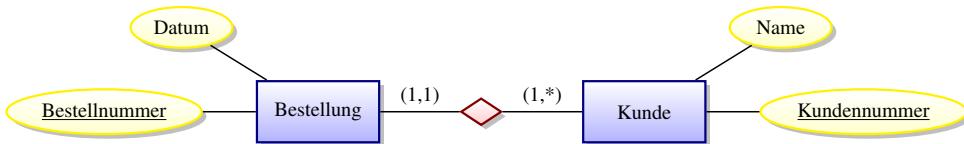
In der Literatur zum physischen Datenbankmodell (auch relationales Modell) wird der Beziehungstyp (1,1):(1,*) gewöhnlich als „meistverbreitetste“ Art von Beziehungstypen bezeichnet. Das ist aber falsch, denn dieser Beziehungstyp lässt sich im physikalischen Datenbankmodell gar nicht repräsentieren. Der 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.



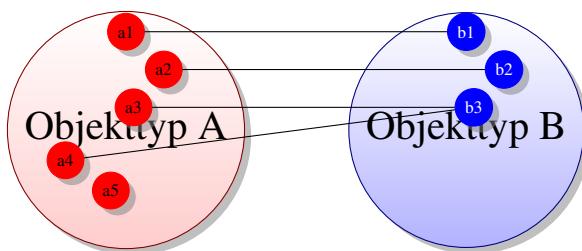
(B:) Kunde(Kundennummer, Name)

(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

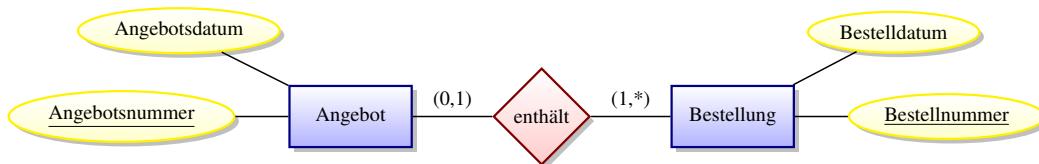
Der (0,1):(1,*) Beziehungstyp unterscheidet sich vom (1,1):(1,*) Beziehungstyp dadurch, dass nicht jedes Objekt des Objekttyps A mit einem Objekt des Objekttyps B gekoppelt sein muss. Es müssen aber wieder alle Objekte aus B mindestens eine Beziehung zu Objekten aus A haben.



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,*)).



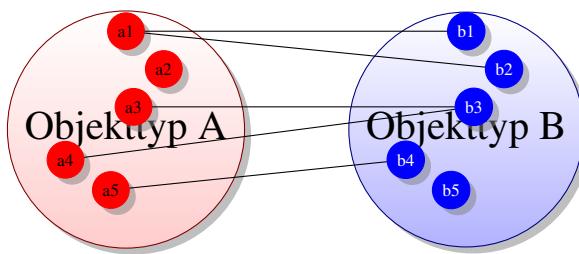
(B:) Bestellung(Bestellnummer, Bestelldatum)

(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.

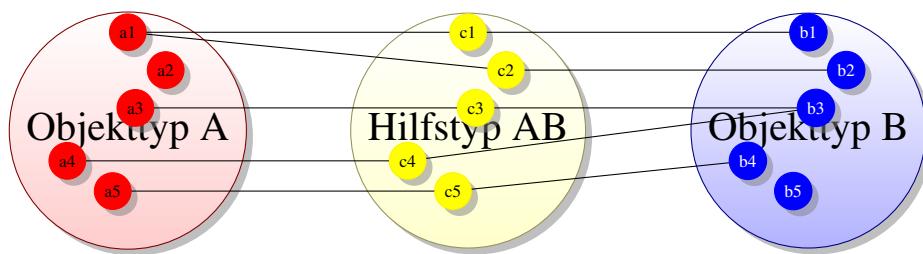
4.3.8 Der (0,*)(0,*) Beziehungstyp

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

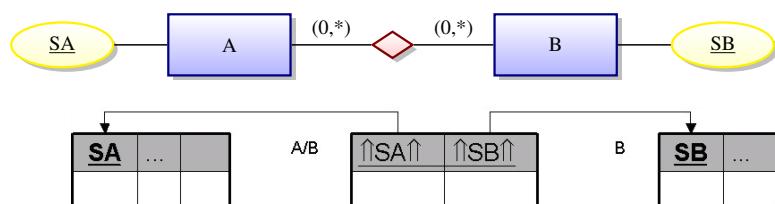
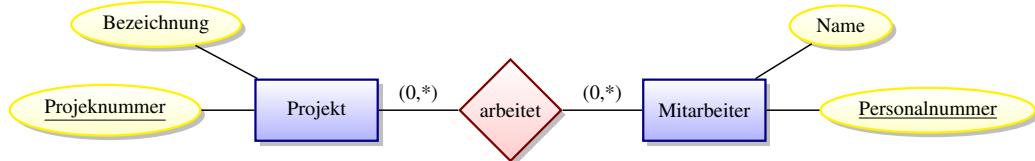


Abb. 4.3:

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.



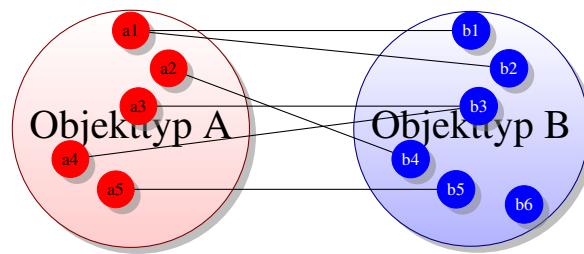
(B:) Mitarbeiter(Personalnummer, Name)

(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.



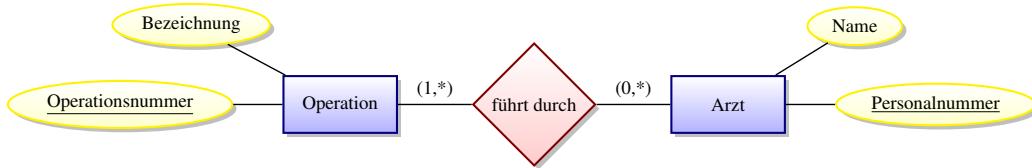
Analog zum (0, *):(0,*) Beziehungstyp muss der (1, *):(0,*) Beziehungstyp vor seiner Transformation in das physische Datenbankmodell durch die Einführung eines neuen Objekttyps A/B in einen (1,1):(0,*) Beziehungstyp und einen (1, *):(1,1) Beziehungstyp umgewandelt werden.



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)

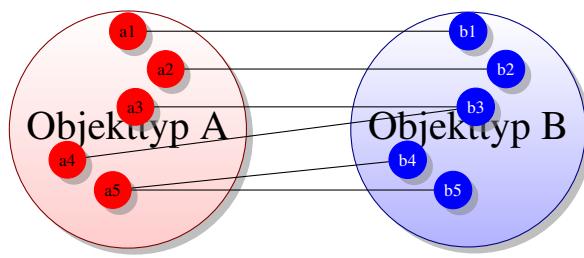
(A:) Operation(Operationsnummer, Bezeichnung)

(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

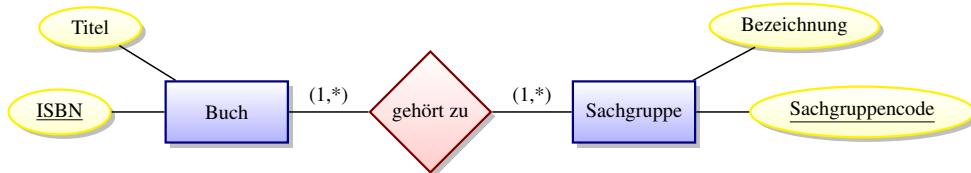
Beim $(1, *):(1, *)$ Beziehungstyp ist jedes Objekt des Objekttyps A mit einem oder mehreren Objekten des Objekttyps B verbunden, ebenso wie jedes B-Objekt mit einem oder mehreren A-Objekten gekoppelt ist.



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↑)

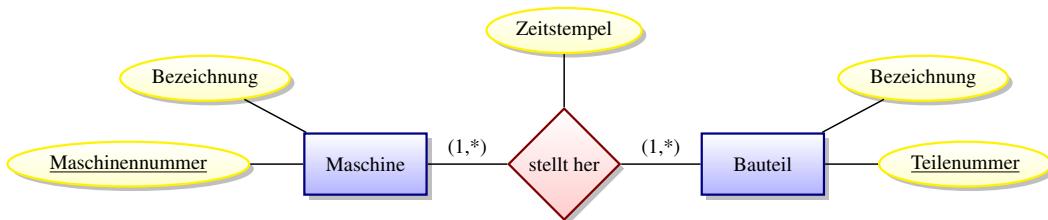
Eine Zeile der Hilfstabelle „Buch/Sachgruppe“ ordnet ein Buch einer Sachgruppe zu. Da weder der FK \uparrow ISBN \uparrow noch der FK \uparrow Sachgruppencode \uparrow für sich unikal sind, können mehrere Zeilen der Hilfstabelle auf dasselbe Buch oder auch auf dieselbe Sachgruppe verweisen. Die Tabellentypbeschreibungen 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

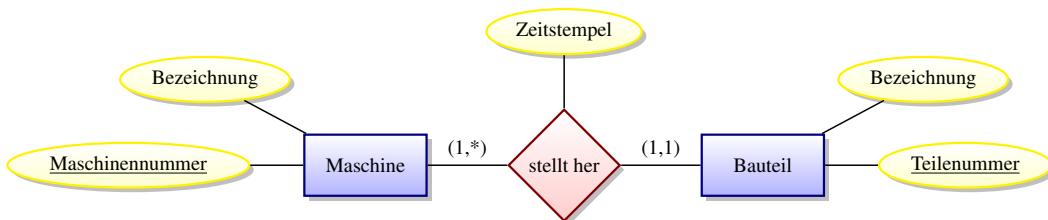


(A:) **Maschine**(Maschinennummer, Bezeichnung)

(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

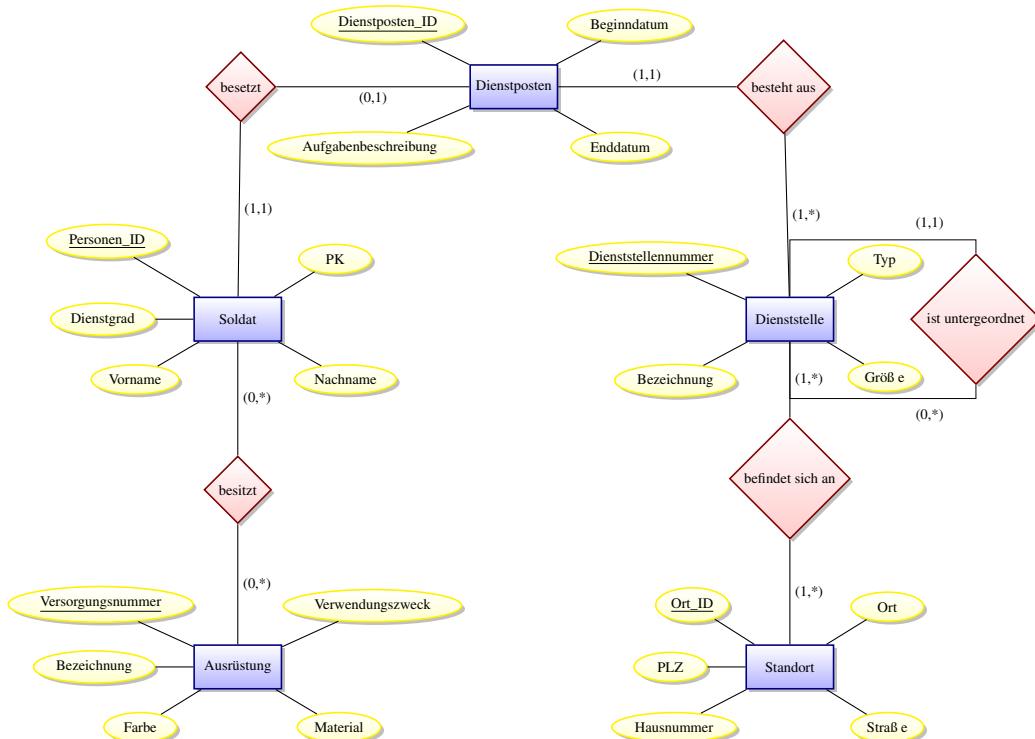
Da bei Beziehungen des Typs 1:1 nur extrem selten Beziehungsattribute vorkommen, muss in solchen Fällen eine Einzelfallbetrachtung durchgeführt werden. Daher wird an dieser Stelle auf ein Transformationsbeispiel verzichtet.

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.

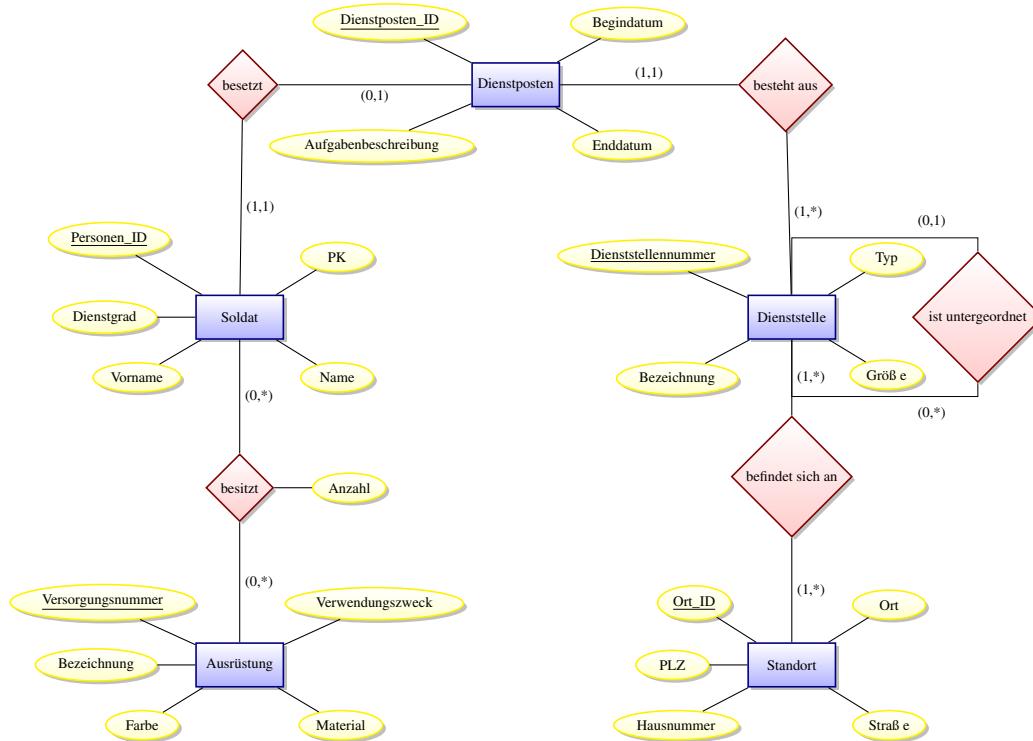
Dienstposten	(<u>Dienstposten_ID</u> , Beginndatum, Enddatum, Aufgabenbeschreibung)
Soldat	(<u>Personen_ID</u> , PK, Vorname, Nachname, Dienstgrad)
Ausrüstung	(<u>Versorgungsnummer</u> , Bezeichnung, Verwendungszweck, Material, Farbe)
Dienststelle	(<u>Dienststellenummer</u> , Bezeichnung, Typ, Größe)
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:

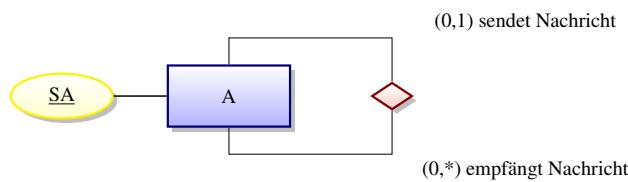
SoldatAusrüstung(Personen_ID + Versorgungsnummer, Anzahl)

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

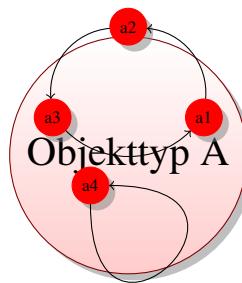
Da bei Rekursiv-Beziehungstypen die miteinander gekoppelten Objekte beide im selben Objekttyp liegen, können sie nicht - wie bei den binären Beziehungstypen - durch ihre Zugehörigkeit zum jeweiligen Objekttyp unterschieden werden. Bei Rekursiv-Beziehungstypen müssen die Objekte hinsichtlich ihrer Rolle 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])

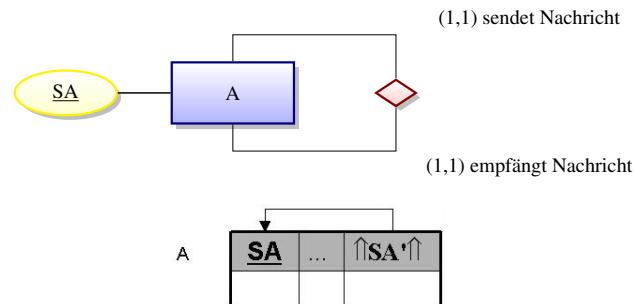


Abb. 4.4:

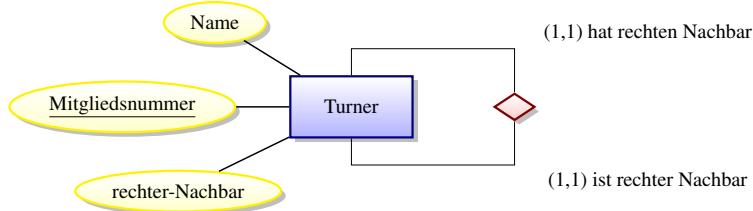
Beispiel – Kreis einer Turnergruppe

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).

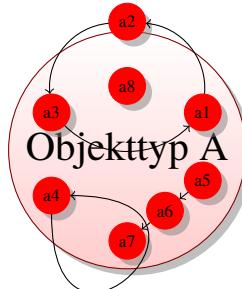
Schwierig wird bei diesem Beziehungstyp die Speicherung der Daten so vorzunehmen, dass keine Integritätsverletzung auftritt. Es sind entsprechende software-technische Maßnahmen zu treffen, die hier nicht weiter erläutert werden sollen.



Turner(Mitgliedsnummer, Name, \uparrow rechter-Nachbar \uparrow [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])

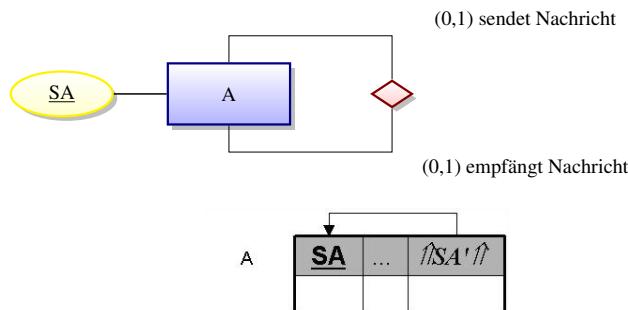
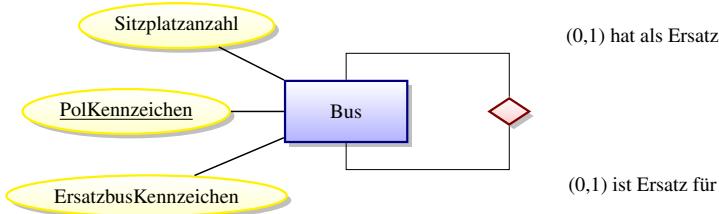


Abb. 4.5:

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).



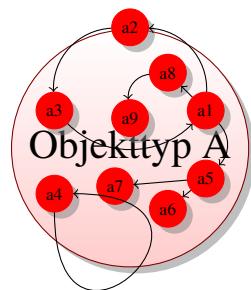
Bus(PolKennzeichen, Sitzplatzanzahl, $\uparrow\!\!/\!\!\downarrow\text{ErsatzbusKennzeichen}$ [UN])

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>	⇒ Tabelle A mit PK <u>SA</u>
(1,1):(0,*) Rekursiv-Beziehungstyp	⇒ <u>SA</u> wird in A mit anderen Attributbezeichnungen „gedoppelt“ und als eingabepflichtiger nicht-unikaler FK $\uparrow\!SA'\!\uparrow$ vereinbart, der auf den Sender verweist. ($\uparrow\!SA'\!\uparrow$ [NN])

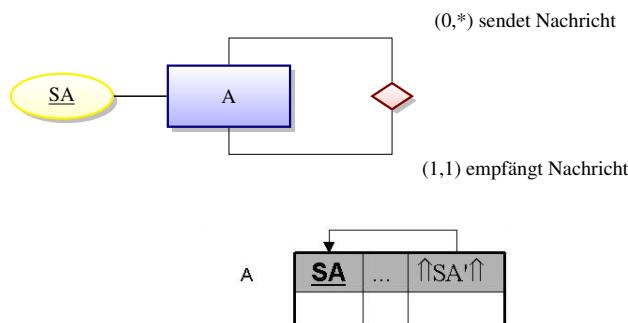
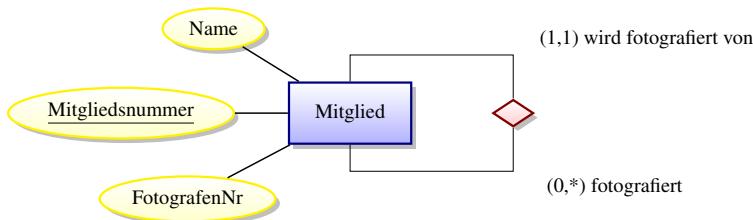


Abb. 4.6:

Beispiel - Fotos der Vereinsmitglieder

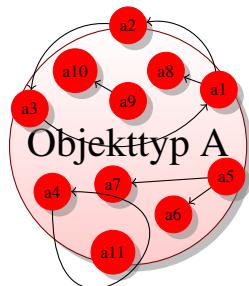
Nehmen wir als Beispiel an, dass für die Festzeitschrift eines Vereins von allen Mitgliedern ein Foto benötigt wird. Im Interesse der Kostendämpfung wird kein externer Fotograf hinzugezogen. Jedes Mitglied muss genau einmal fotografiert werden. Manche Mitglieder fotografieren gar nicht, andere müssen 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, $\uparrow\!FotografenNr\!\uparrow$ [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. ($\uparrow SA' \uparrow$)

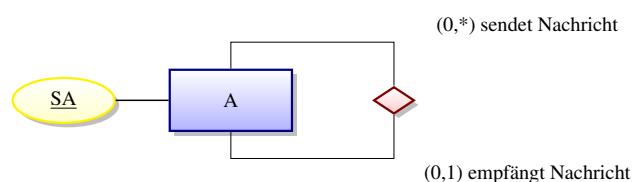
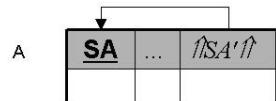
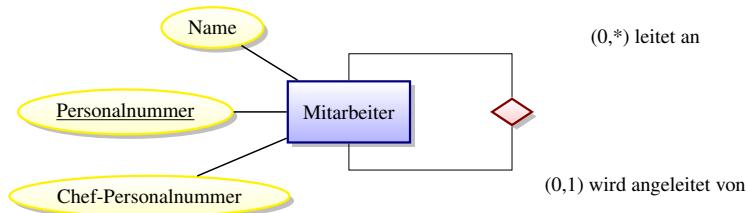


Abb. 4.7:



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:

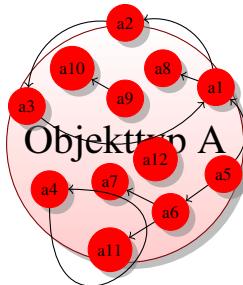
- *Mehr-Objekt-Zyklen*: Der Mitarbeiter mit der Personalnummer „0815“ kann Chef des Mitarbeiters „0816“ sein. Dieser kann den Mitarbeiter „0817“ anleiten, der wiederum Chef des Mitarbeiters „0815“ ist.
- *Ein-Objekt-Zyklen*: Ein Mitarbeiter kann als sein eigener Chef ausgewiesen werden, wenn nämlich 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.

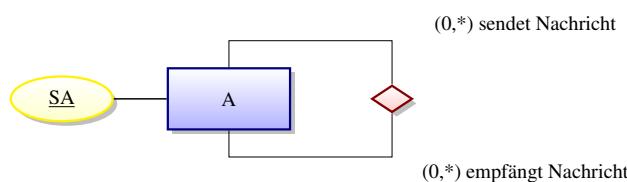
4.5.5 Der $(0,*)(0,*)$ Rekursiv-Beziehungstyp

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.

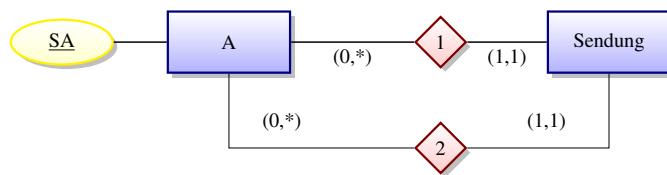
Es lassen sich beliebig strukturierte Netzwerke darstellen. Insbesondere können dies Polyhierarchien sein. In einer Polyhierarchie hat jedes Objekt keines, eines oder mehrere untergeordnete Objekte. Jedes Objekt kann keines, eines oder mehrere übergeordnete Objekte besitzen. Als Sonderfall sind natürlich 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:

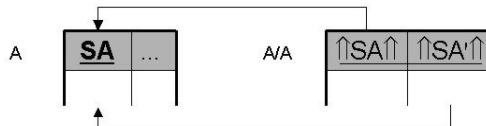


Stellt man den Objekttyp „Sendung“ im physischen Datenbankmodell als Koppel-Tabelle A/A dar, so enthält er den PK des Objekttyps A zweimal, als eingabepflichtige FK, die auf den Empfänger bzw. auf den Sender einer Nachricht verweisen. Beide FK sind für sich genommen nicht-unikal, so dass ein Objekt in 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

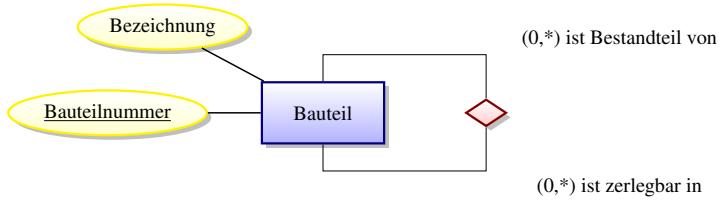
ER-Modell		physisches Datenmodell
Objekttyp A mit Schlüssel <u>SA</u>	\Rightarrow	Tabelle A mit PK <u>SA</u>
(0,*)(0,*) Rekursiv-Beziehungstyp	\Rightarrow	Koppel-Tabelle A/A, die <u>SA</u> in zwei Exemplaren, als eingabepflichtige, nicht-unikale Fremdschlüsselelemente enthält: als Empfänger-Fremdschlüssel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

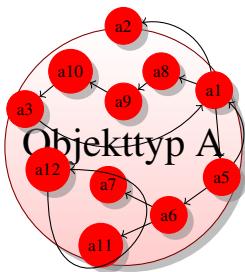
- *Ein-Objekt-Zyklen*: Ein Bauteil kann als sein eigenes Bestandteil ausgewiesen werden, wenn seine Bauteilnummer in einer Zeile der Koppel-Tabelle als Wert beider FK angegeben wird.
- *Mehr-Objekte-Ketten*: Ein Bauteil kann als Bestandteil nur ein einziges anderes Bauteil haben. Seine Nummer kommt dann 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.

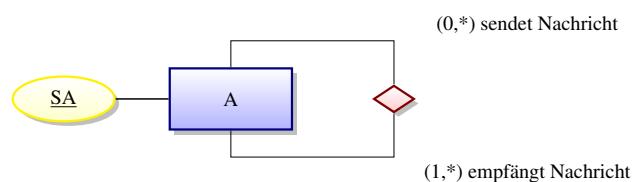
Die beschriebenen Situationen kann das Datenbankmanagementsystem nicht verhindern. Sie müssen durch entsprechende Anwendungsprogrammierung unterbunden werden.

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

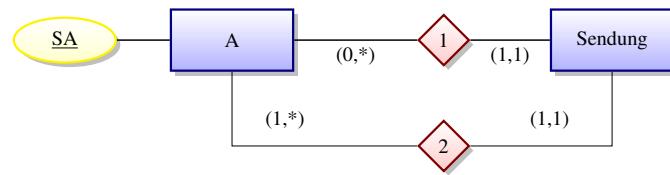
Beim (1,*):(0,*) Rekursiv-Beziehungstyp kann ein Objekt keine, eine oder mehrere Nachrichten senden, es muss aber stets mindestens eine Nachricht empfangen. Die folgende Abbildung vermittelt einen Eindruck von möglichen Beziehungsstrukturen, ohne alle Varianten abzudecken.



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.



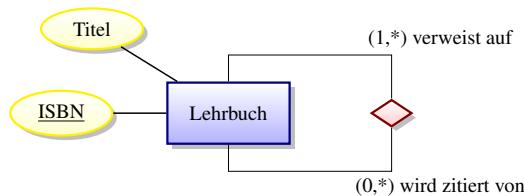
wird umgewandelt in:



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.

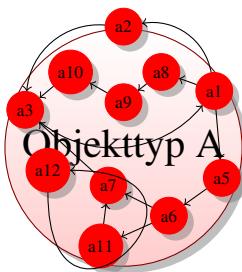
Die Tabellentypbeschreibungen lassen folgende Situationen zu, die nicht der Semantik entsprechen.

- *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.

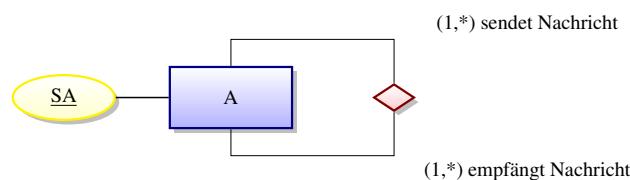
Die beschriebenen „pathologischen“ Situationen lassen sich nur durch die Anwendungsprogrammierung vermeiden.

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

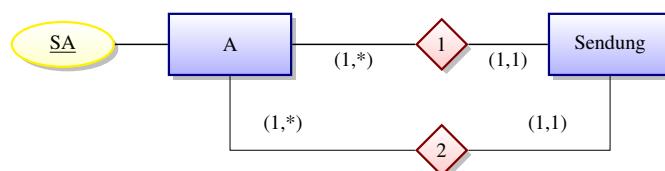
Der $(1,*)(1,*)$: $(1,*)$ Rekursiv-Beziehungstyp unterscheidet sich nur in einem Punkt, vom zuvor behandelten Rekursiv-Beziehungstyp $(1,*)(0,*)$. Ein Objekt muss mindestens eine Nachricht senden. Diese Forderung hat jedoch entscheidende Konsequenzen für die möglichen Beziehungsstrukturen, von denen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



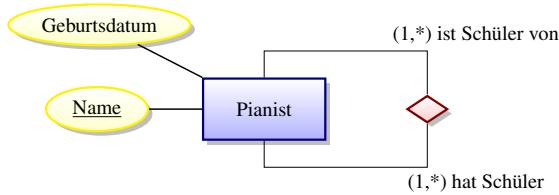
wird umgewandelt in:



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).

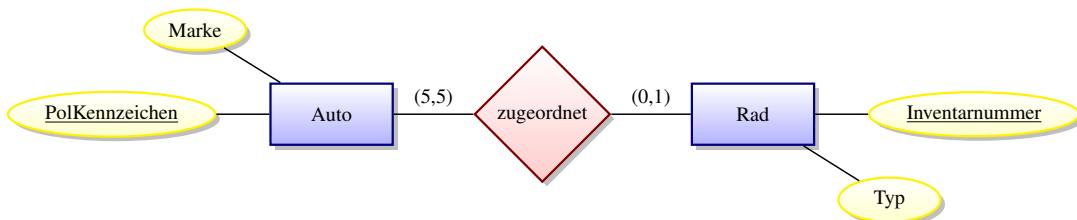
Diese unzulässigen Strukturen sind durch die Anwendungsprogrammierung zu verhindern.

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.



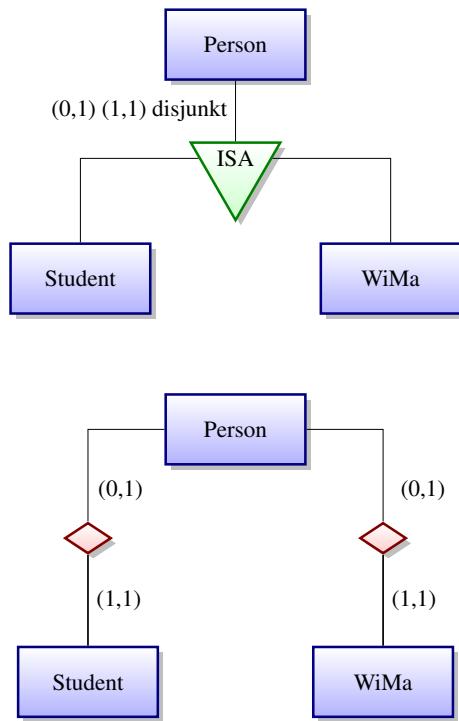
Transformationsregel T14 Kardinalitäts-Beschränkung

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

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

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

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
INTERVAL '8' DAY	Ein Intervall von 8 Tagen
INTERVAL '8 4' DAY TO HOUR	Ein Intervall von 8 Tagen und 4 Stunden
INTERVAL '8 4:25' DAY TO MINUTE	Ein Intervall von 8 Tagen, 4 Stunden und 25 Minuten
INTERVAL '8 4:25:10' DAY TO SECOND	8 Tage, 4 Stunden, 25 Minuten und 10 Sekunden
INTERVAL '120' DAY(3)	120 Tage (Präzision 3!!!)
INTERVAL '3' HOUR	3 Stunden
INTERVAL '3:18' HOUR TO MINUTE	3 Stunden und 18 Minuten
INTERVAL '3:18:10' HOUR TO SECOND	3 Stunden, 18 Minuten und 10 Sekunden
INTERVAL '210' HOUR	210 Stunden
INTERVAL '18' MINUTE	18 Minuten
INTERVAL '18:10' MINUTE TO SECOND	18 Minuten und 10 Sekunden
INTERVAL '120' MINUTE	120 Minuten
INTERVAL '10' SECOND	10 Sekunden
INTERVAL '180' SECOND	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;
```



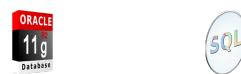
Mit NULL	Ohne NULL
	30000.000000
	30000.000000
...	...
2400.000000	2400.000000
2500.000000	2500.000000
101 Zeilen ausgewählt	

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 gemäßige 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

1. 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";
```

2. 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";
```

3. 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;
```

4. 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) = '0 Juppiter, magnus numerus'
    RAISERROR ('failed fourth test',16,1)
if NOT dbo.ToRomanNumerals(2725) = 'MMDCCXXV'
    RAISERROR ('failed fifth test',16,1)
if NOT dbo.ToRomanNumerals(949) = 'CMXLIX'
    RAISERROR ('failed Sixth test',16,1)
if NOT dbo.ToRomanNumerals(154321) = N'ccc|           MMMMCCCXXI'
    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.

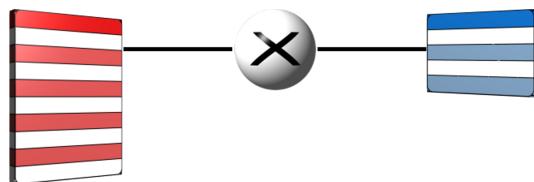


Abb. 8.1:
Left-Outer-Join

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
```

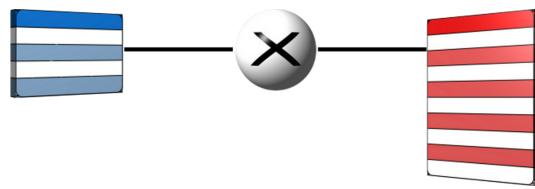


Abb. 8.2:
Der
Right-Outer-Join

```
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!

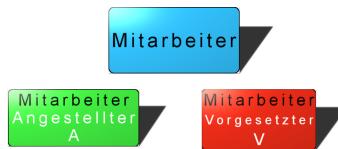


Abb. 8.3:
Gespaltene
Persönlichkeit -
Eine Tabelle,
zwei Aliase

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.

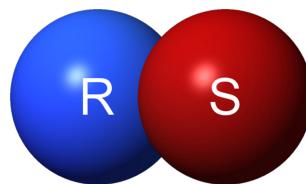
8.4.2 Die SET-Operatoren

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

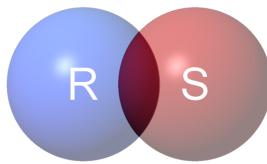
In der ersten Abfrage wird eine Spalte, mit Aliasnamen PERSONENTYP eingefügt. Sie bezieht ihren Wert nicht aus einer Tabelle, sondern sie enthält einfach nur den Buchstaben „K“ für Kunde. Die gleiche Spalte muss nun auch in der zweiten Abfrage eingeführt werden, da beide Abfragen, wie bereits erwähnt, 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**

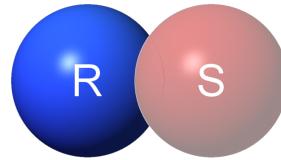
Alsleben
Aschersleben
Bernburg
Borne
Bördehue
25 Zeilen ausgewählt

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;
```

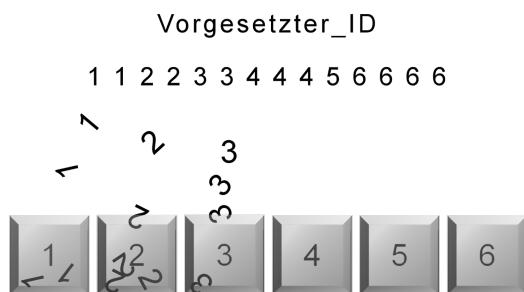


VORGESETZTER_ID

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 (GEBURTS DATUM)	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	Weiß
Petra	Berger
Karolin	Lange
Tom	Hartmann
28 Zeilen ausgewählt	

Auf die gleiche Art und Weise, wie in Beispiel ?? gezeigt, können 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 Ergebnisse Zeilen 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 view_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 Tabelenzeile 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 Oracle Datenbankarchitektur

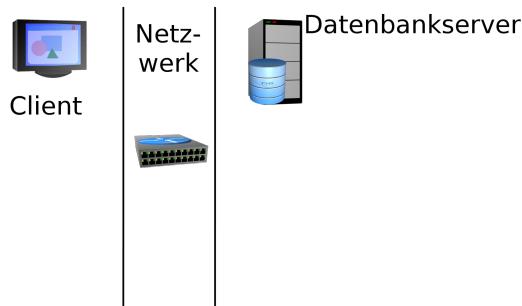
Inhaltsangabe

14.1 Oracle und die Client-Server-Architektur

Oracle-Datenbanken sind dazu geschaffen, um riesige Datenmengen zu verwalten und diese in einer Multi-User-Umgebung einer großen Nutzeranzahl zur Verfügung zu stellen. Eine solche Umgebung kann auf unterschiedliche Arten realisiert werden. Die einfachste davon ist eine Client-Server-Architektur, bestehend aus:

- Client
- Netzwerk
- Datenbankserver

Abb. 14.1:
Oracle
Datenbank-
architektur



14.1.1 Der Client

Nutzerprozess

Eine Client-Anwendung, die sich mit einer Oracle-Datenbank verbündet, wird als „Nutzerprozess“ bezeichnet. Beispiele für solche Nutzerprozesse sind:

- SQL*Plus
- SQL*Developer
- J*Developer
- Enterprise Manager Console

Connection

Eine Connection ist eine physikalische Verbindung zwischen einem Client und dem Datenbankserver. Sind Client und Datenbankserver eins, wird die Connection durch einen Interprozesskommunikationsmechanismus (IPC) erzeugt. Bei zwei unterschiedlichen Rechnern, wird die Connection über ein Netzwerkprotokoll, wie beispielsweise TCP/IP realisiert.



Eine Connection stellt die Grundlage für eine Verbindung mit einer Oracle-Datenbank dar.

Session

Während die Connection eine physikalische Verbindung zwischen dem Client und dem Datenbankserver darstellt, ist eine Session eine Kommunikationsverbindung zwischen dem Nutzerprozess und der Datenbank. Der Aufbau einer Session erfolgt, sobald sich ein Benutzer bei der Datenbank authentifiziert hat.



Eine Session stellt einen Zugang zur Datenbank dar.

Oracle ist in der Lage, mehrere Sessions über ein und die selbe physikalische Connection zu öffnen. Zum Beispiel kann sich ein Nutzer mit Name/Passwort „SCOTT/TIGER“ beliebig oft von einem Rechner aus, an der Datenbank anmelden.

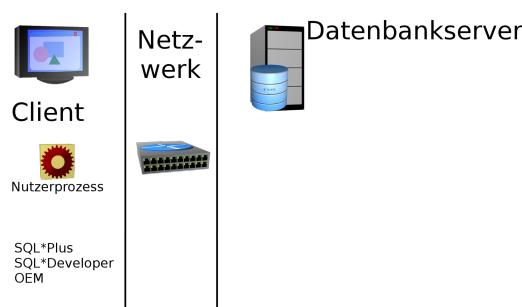


Abb. 14.2:
Oracle
Datenbank-
architektur

14.1.2 Der Datenbankserver

Instanz und Datenbank

Ein Oracle-Datenbankserver besteht aus zwei großen Teilen:

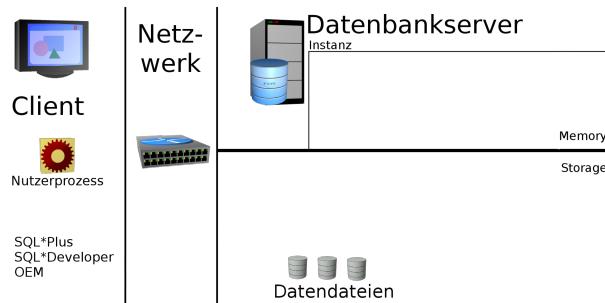
- **Instanz:** Die Instanz ist das Herzstück einer Oracle-Datenbank. Es handelt sich dabei um eine Menge von Arbeitsspeicherstrukturen und Prozessen, die ein schnelles und effizientes Arbeiten mit den Daten ermöglichen. Sie gliedern sich grob in drei Teile: System Global Area, Program Global Area und Hintergrundprozesse.
- **Datenbank:** Dies ist die Menge aller Dateien, aus denen der Oracleserver besteht. Es sind sowohl die Installationsdateien der Oracle-Software, als auch die Datendateien gemeint, welche die Nutzdaten enthalten, sowie alle weiteren Dateien.



Die Begriffe Instanz und Datenbank, werden oft als Synonyme verwendet. Tatsächlich handelt es sich aber um zwei sehr unterschiedliche Dinge.

Während eine Instanz immer nur auf genau eine Datenbank zugreifen kann, kann eine Datenbank gleichzeitig von mehreren Instanzen gemountet¹ werden.

Abb. 14.3:
Oracle
Datenbank-
architektur



Serverprozess

Ein Serverprozess stellt das serverseitige Pendant zu einem Nutzerprozess dar. Er ist die eigentliche „Arbeitsmaschine“, nimmt die Anforderungen eines Nutzerprozesses entgegen und verarbeitet diese innerhalb der Instanz.

¹engl. to mount = anschließen



Ein Nutzerprozess benötigt immer einen Serverprozess um arbeitsfähig zu sein. Dadurch wird gewährleistet, dass keine Clientanwendung direkt in der Datenbank arbeitet und das somit auch bestimmte Regeln gewahrt bleiben.

Um besser erklären zu können, was ein Serverprozess ist, soll als bildliches Beispiel ein Ober in einem Restaurant dienen. Kein Gast geht in einem Lokal in die Küche und bereitet sich sein Essen selbst zu. Statt dessen wird er bei einem Ober (dem Serverprozess) eine Bestellung aufgeben, welcher diese dann an den Koch weiterreicht.

Bei ihm angekommen, wird die Bestellung zubereitet bzw. im Sinne einer Datenbank werden die geforderten Datensätze zusammengestellt. Anschließend hat der Ober noch die Aufgabe, das Essen / das Ergebnis dem Gast zu servieren.

Der einzige Haken an diesem Beispiel ist, dass der Serverprozess Ober und Küchenpersonal in Personalunion ist. Er nimmt Anforderungen entgegen, verarbeitet diese innerhalb der Instanz und reicht das Ergebnis an den Nutzerprozess zurück.

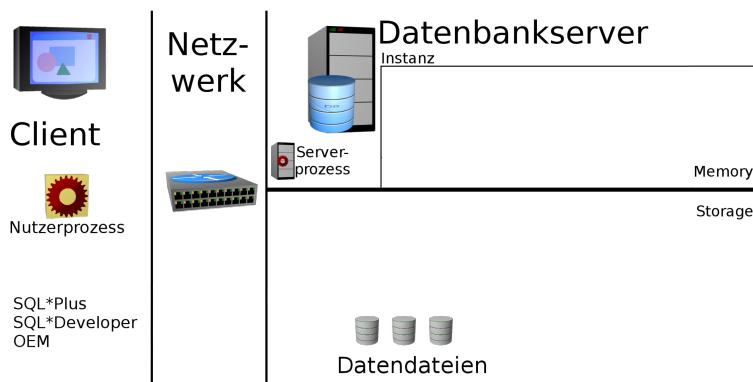
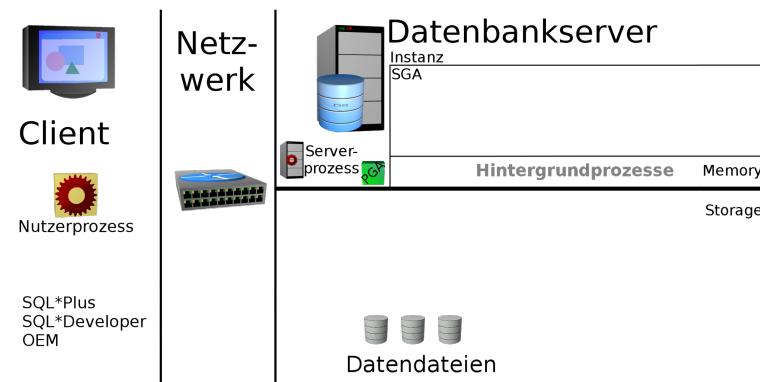


Abb. 14.4:
Oracle
Datenbank-
architektur

14.2 Die System Global Area (SGA)

Die System Global Area (kurz SGA) ist eine Speicherstruktur mit variabler Größe innerhalb einer Instanz. Alle von den Serverprozessen ausgeführten Arbeitsschritte benötigen in irgendeiner Form die SGA. Sie enthält Nutz- und Metadaten, sowie andere Kontrollinformationen zu einer bestimmten Datenbank. Die Erstellung der SGA erfolgt automatisch beim Instanzstart. Durch das Herunterfahren einer Instanz wird die SGA zerstört.

Die SGA teilt sich in verschiedene Regionen, mit unterschiedlichem Inhalt.

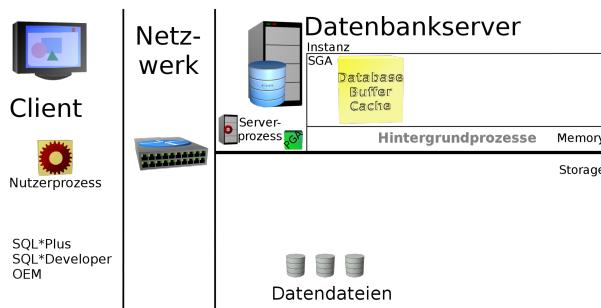
Abb. 14.5:
Oracle
Datenbank-
architektur

14.2.1 Die Fixed SGA

Ein sehr kleiner Teil der SGA wird als „Fixed SGA“ bezeichnet. Hierbei handelt es sich um einen betriebssystemabhängigen, unveränderlichen Teil der SGA, der als eine Art „Einstiegspunkt“ von allen Serverprozessen genutzt wird. Er enthält die Speicheradressen anderer Strukturen innerhalb der SGA. Der Datenbankadministrator hat keinerlei Kontrolle über diesen Teil der SGA.

14.2.2 Der Database Buffer Cache

Der Database Buffer Cache ist für die Zwischenspeicherung der zuletzt benutzten Oracleblöcke zuständig. Da eine theoretische Chance besteht, dass ein Block der bereits einmal benötigt wurde, auch noch weitere Male benötigt wird, kann so die Anzahl der Zugriffe auf den Datenträger reduziert und die Arbeitsgeschwindigkeit erhöht werden.

Abb. 14.6:
Oracle
Datenbank-
architektur

Neue oder geänderte Daten werden nicht sofort auf den Datenträger geschrieben. Um die Anzahl der Festplattenzugriffe zu reduzieren und somit die Performance der Datenbank zu steigern, werden die Daten im Database Buffer Cache gesammelt. Abhängig von bestimmten Kriterien, wird der Inhalt des Caches dann auf den Datenträger geschrieben.

14.2.3 Der Shared Pool

Der Teil der SGA, der „Shared Pool“ genannt wird, ist für die Speicherung von Informationen bezüglich ausführter SQL-Statements zuständig. Er gliedert sich seit Oracle 11g in drei Bereiche:

- Library Cache
- Data Dictionary Cache
- Result Cache

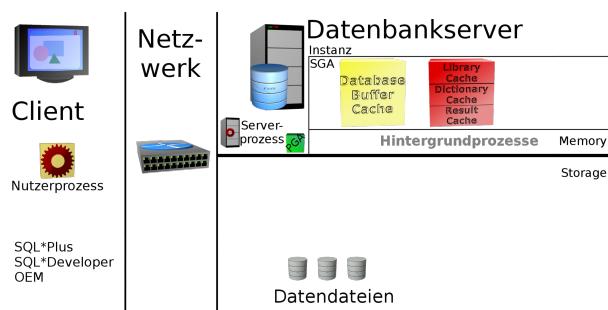


Abb. 14.7:
Oracle
Datenbank-
architektur

Der Library Cache

Der Library Cache enthält Informationen über alle ausgeführten SQL- und PL/SQL-Statements. Setzt ein Nutzer ein SQL-Statement ab, werden Informationen darüber in Form eines Ausführungsplanes im Library Cache abgelegt. Wird exakt das gleiche Statement von einem anderen Nutzer ausgeführt, können die vorhandenen Informationen im Library Cache wiederverwendet werden, was die Ausführungsgeschwindigkeit wesentlich erhöht.

Oracle nutzt für jedes SQL-Statement soviel Speicherplatz im Shared Pool wie notwendig. Ist kein Platz mehr für weitere Informationen vorhanden, wird nach einem modifizierten LRU-Algorithmus wieder Speicher freigegeben.



Die Abkürzung LRU steht für „Least Recently Used“, was soviel bedeutet wie: „Am wenigsten benötigt“. Ein LRU-Algorithmus hat die Aufgabe die am wenigsten benötigten Informationen aus einer Speicherstruktur zu entfernen, um so Speicherplatz freizugeben.

Der Dictionary Cache

Das Data Dictionary ist eine Sammlung von Datenbanktabellen und Views, die Metadaten über die gesamte Datenbank enthalten. Oracle muss ständig im laufenden Betrieb auf dieses Data Dictionary zugreifen. Um häufige Datenträgerzugriffe zu vermeiden, werden benötigte Informationen aus dem Data Dictionary im Data Dictionary Cache zwischengespeichert.

Der Result Cache

Beim Result Cache handelt es sich um ein neues Feature der Oracle Version 11g. Dieser Cache speichert keine Datenblöcke, sondern Ergebniszeilen. Der Vorteil dieser Vorgehensweise liegt auf der Hand.

Wird ein SQL-Statement häufig ausgeführt, ohne das sich die Datenbasis ändert, kann das Ergebnis direkt aus dem Result Cache übermittelt werden. Sinnvoll genutzt werden kann der Result Cache immer dann, wenn:

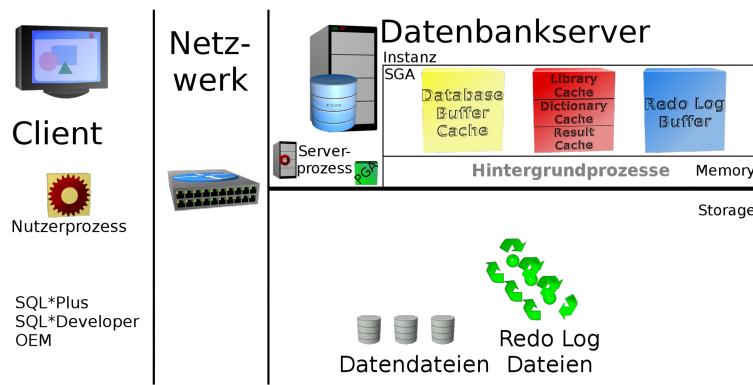
- SQL-Abfragen sehr rechenintensiv sind,
- das Ergebnis einer SQL-Abfrage nahezu unveränderlich ist,
- sehr viel Arbeitsspeicher vorhanden ist.

14.2.4 Der Redo Log Buffer

Für jede Änderung, die an einer Oracle-Datenbank durchgeführt wird, wird ein sogenannter Redo Record erzeugt. Ein Redo Record protokolliert jeweils eine Änderung und macht diese somit nachvollziehbar. Durch Redo Records wird Datensicherheit dahingehend gewährleistet, dass im Falle eines Crashes alle protokollierten Änderungen wieder in die Datenbank eingepflegt werden können.

Im Redo Log Buffer werden die zuletzt erzeugten Redo Records zwischengespeichert. Er ist als Ringpuffer (beim Erreichen des letzten Eintrags wird der erste Eintrag wieder überschrieben) organisiert. In bestimmten Zeitabständen oder wenn andere Bedingungen erfüllt sind, wird der Inhalt des Redo Log Buffers in die Redo Log Dateien geschrieben.

Abb. 14.8:
Oracle
Datenbank-
architektur



14.3 Die Program Global Area (PGA)

Während die SGA für alle Nutzer relevante Informationen beinhaltet, ist die PGA ein Speicherbereich, der sessionabhängige Informationen enthält. Das heißt jeder Nutzer hat seine eigene PGA. Eine PGA wird beim Starten eines Serverprozesses angelegt und gehört zu genau einem bestimmten Serverprozess.

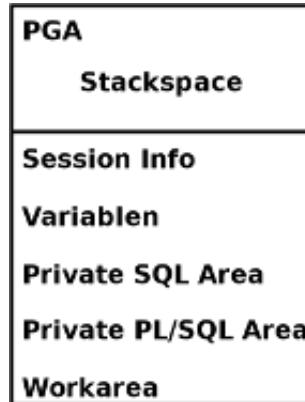


Abb. 14.9:
Die Program
Global Area

Abbildung ?? zeigt den Aufbau einer PGA. Sie besteht im Wesentlichen aus einem Bereich namens „Stackspace“. Dieser beinhaltet Informationen über die Nutzersession, verschiedene Variablen der Nutzersession, private Informationen zur Abarbeitung von SQL-Statements und eine Workarea. Die Workarea ist ein Bereich, in dem Hash- und Sortieroperationen durchgeführt werden.



- [CNCPT1237]

14.4 Memory Management

Unter dem Begriff Memory Management werden alle Aufgaben und Einstellungen zusammengefasst, die sich um die Dimensionierung der Speicherkomponenten von SGA und PGA drehen. Bis zur Version Oracle 9i konnte das Memory Management nur manuell durch den Administrator durchgeführt werden. Diese Technik wurde als „Manual Shared Memory Management“ bezeichnet. Der Administrator musste die Größen für alle SGA-Komponenten manuell eingeben und, falls notwendig, sie den aktuellen Gegebenheiten anpassen. Dies ermöglichte dem Admin zwar eine sehr genaue Kontrolle über die SGA und alle PGAs, jedoch konnten durch schlechte Einstellungen auch viele Fehler gemacht und die Performance der Instanz stark gesenkt werden.

Mit Oracle 10g wurde dann das „Automatic Shared Memory Management“ eingeführt, wodurch ein automatisches Tuning der SGA-Komponenten und der PGAs erfolgte. Bei diesem neuen Verfahren musste der DBA lediglich noch zwei Speichergrößen angeben: Eine Speichermenge für alle SGA-Komponenten und einen Speicherpool für alle PGAs. Oracle 10g vergrößerte/verkleinerte dann die Speicherbereiche der SGA automatisch, so dass diese immer den aktuellen Erfordernissen entsprachen und die Instanz eine optimale Performance erreichen konnte.

Aus dem Speicherpool der PGAs wurden dann die einzelnen PGAs der Serverprozesse erstellt. Dadurch dass auch hier nur noch ein einzelner Wert angegeben werden musste, konnte Oracle auch das automatische Anpassen der PGA-Größen übernehmen, so dass jeder Serverprozess die für ihn optimale Speichermenge zur Verfügung hatte.

Mit Oracle 11g kam nun das „Automatic Memory Management“, was die Verwaltung von SGA- und PGA-Speicher noch weiter vereinfacht und ein besseres Tuning der Speicherstrukturen erlaubt. Der DBA muss nur einen einzigen Wert angeben, nämlich die Gesamtmenge an Arbeitsspeicher, die für SGA und PGAs verfügbar sein soll. Oracle 11g übernimmt die gesamte Verwaltung aller Speicherkomponenten der SGA und PGAs. Dadurch das nun nur noch ein Limit für beide existiert, teilen sich SGA und PGA den gleichen Speicher, was eine höhere Flexibilität bei der Größenanpassung von SGA und PGA bedeutet.

Tabelle ?? fasst noch einmal alle Arten des Memory Management und deren Features zusammen.

Tabelle 14.1: Oracle Shared Memory Management

Oracle	Memory Management	Auto. Tuning	Einstellungen
bis Oracle 9i	Manual Shared Memory Management	PGA-Größe	Alle Komponenten der SGA einzeln
ab Oracle 10g	Automatic Shared Memory Management	Größe aller Komponenten der SGA und die Speichergrößen der PGAs	SGA-Zielgröße und aggregierter Speicher aller PGAs

ab Oracle 11g	Automatic Memory Management	Größe aller Komponenten der SGA und die Speichergrößen der PGAs	Speichermenge der gesamten Instanz (SGA + PGA)
---------------	-----------------------------	---	--

14.5 Überblick über die Struktur einer Oracle Datenbank

14.5.1 Datendateien

Jede Oracle Datenbank besteht aus einer oder mehreren Datendateien. Sie enthalten alle Nutz- und Metadaten, die dort in Form von logischen Datenstrukturen, wie z. B. Tabellen und Indizes gespeichert werden.



- Eine Datendatei kann nur zu einer Datenbank gehören.
- Das verwendete Dateisystem begrenzt die Anzahl der Datendateien, die angelegt werden können.

14.5.2 Die Parameterdatei

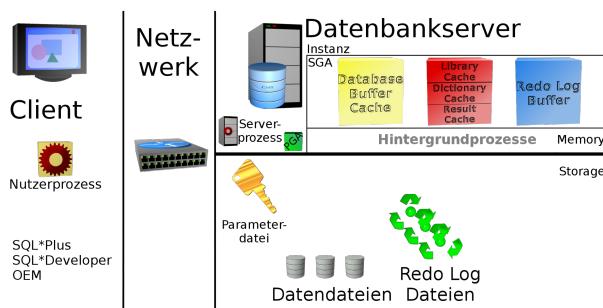
Beim Start einer Oracle Datenbank muss eine Vielzahl von Parametern gesetzt werden. Diese Parameter werden in so genannten Parameterdateien (PFile oder SPFile) zusammengefasst.

Es gibt zwei unterschiedliche Arten von Parameterdateien:

- Die Parameterdatei (PFile): Sie ist eine statische Sammlung von Parametern in einer Textdatei. Es können keine dynamischen Änderungen vorgenommen werden. Jede Änderung an den Initialisierungsparametern der Datenbank muss auch an der Parameterdatei vorgenommen werden.
- Die Server-Parameterdatei (SPFile): Hierbei handelt es sich um eine dynamische Parameterdatei, in Form einer Binärdatei. Mit Hilfe einer Server-Parameterdatei ist es möglich, Änderungen an einem Initialisierungsparameter in einem einzigen Schritt vorzunehmen. Diese Änderungen können dann wahlweise nur an der Instanz, nur in der Server-Parameterdatei oder aber an Instanz und Server-Parameterdatei gleichzeitig getätigkt werden.



Abb. 14.10:
Oracle
Datenbank-
architektur



Eine Server-Parameterdatei darf niemals mit einem Texteditor verändert werden, da Oracle eine solche Datei sofort als korrupt erkennt und nicht mehr benutzen kann.

Oracle empfiehlt die Nutzung von Server-Parameterdateien. Es gibt sie seit Oracle 9i. Während in Oracle 9i standardmäßig noch Parameterdateien genutzt wurden, wird seit Oracle 10g automatisch jede Datenbank, die mit dem Database Configuration Assistant angelegt wird, mit einer Server-Parameterdatei erstellt.

14.5.3 Kontrolldateien

Die Kontrolldatei ist ein wesentlicher Bestandteil einer jeden Oracle-Datenbank. Sie enthält alle Informationen über den Aufbau der jeweiligen Datenbank. Dies sind beispielsweise:

- Datenbankname
- Namen und Speicherorte aller Datendateien
- Zeitstempel der Datenbankerstellung
- Informationen über Backups

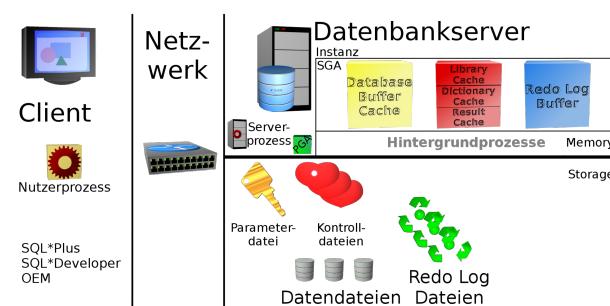


Abb. 14.11:
Oracle
Datenbank-
architektur

Aus Gründen der Ausfallsicherheit wird die Kontrolldatei gespiegelt, d. h. sie wird mehrfach mit gleichem Inhalt gespeichert. Der Verlust aller Kontrolldateien ist ein besonders kritischer Fall, da die Datenbank nur mit großem Aufwand wieder geöffnet werden kann.

Gebraucht wird die Kontrolldatei an verschiedenen Stellen während des Datenbankbetriebs. Beispielsweise beim Start einer Oracleinstanz. Die Kontrolldatei stellt dann die Informationen über Namen und Speicherorte der Datendateien und anderer wichtiger Dateien bereit, die für den Startvorgang geöffnet werden müssen.

Wird das physische Layout der Datenbank geändert, z. B. dadurch, dass eine neue Datendatei hinzugefügt wird, wird diese Änderung sofort in der Kontrolldatei vermerkt, damit diese immer den aktuellen Stand der Datenbank wiedergibt.

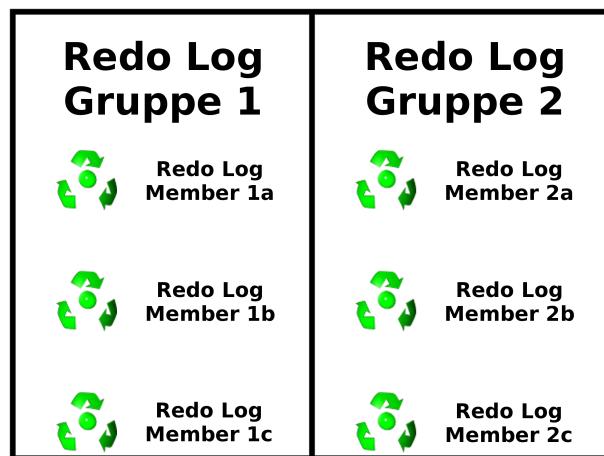
14.5.4 Redo Log Dateien

Aufbau und Funktion

Für den Betrieb einer Oracle Datenbank wird ein Set aus „Redo Log Gruppen“, umgangssprachlich als „Redo Logs“ bezeichnet, benötigt. Eine Redo Log Gruppe besteht aus einer oder mehreren Redo Log Dateien, die auch als Redo Log Member bezeichnet werden.

Die primäre Funktion der Redo Logs ist das Aufzeichnen aller Änderungen, die an den Daten vorgenommen wurden (Nutz- und Metadaten).

Abb. 14.12:
Redo Log
Gruppen und
Member



Die Redo Log Member nehmen den Inhalt des Redo Log Buffers auf. Im Falle eines Recovery benutzt die Datenbank die Redo Logs, um alle Änderungen der Nutzer wieder in die Datenbank einzuarbeiten und sie so auf den letzten Stand vor dem Crash zu bringen.

Spiegelung der Redo Logs

Um die Redo Logs vor Ausfällen zu schützen, hat eine Oracle Datenbank die Möglichkeit sie zu spiegeln und die einzelnen Kopien auf mehrere verschiedene Datenträger zu verteilen.

In Abbildung ?? wird angenommen, dass es zwei Redo Log Gruppen (Gruppe 1 und Gruppe 2) mit je drei Membern (Member a, b und c) gibt. Die Member sind auf zwei Datenträgern (/u02 und /u03) verteilt gespeichert.

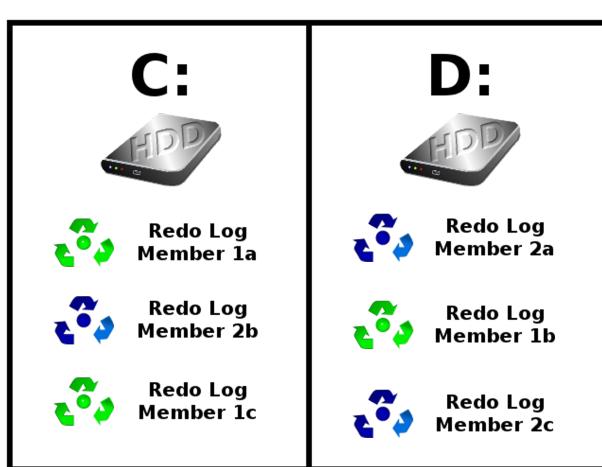


Abb. 14.13:
Verteilung und
Spiegelung von
Redo Log
Membern

Empfehlungen

Grundsätzlich gelten die folgenden Empfehlungen für Redo Log Dateien/-Gruppen:

- Jede Redo Log Gruppe sollte mindestens zwei Member haben (besser drei).
- Es sollten niemals alle Member einer Redo Log Gruppe auf dem gleichen Datenträger liegen.
- Alle Redo Log Gruppen sollten immer die gleich Anzahl Member haben und damit symmetrisch sein.
- Die Größe der Redo Log Member sollte so angepasst werden, dass kein Platz auf den Sicherungsmedien verschwendet wird.



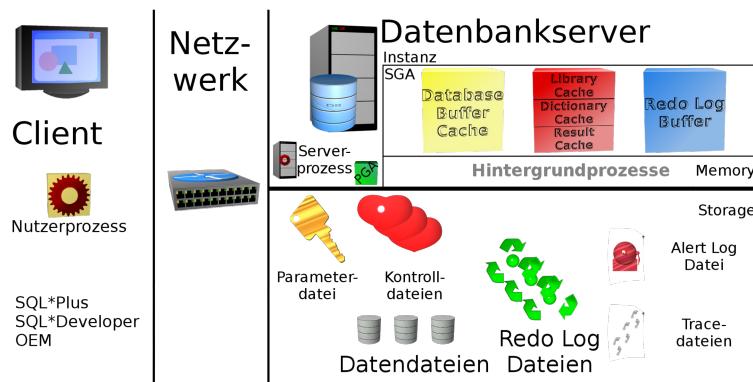
Zu beachten ist, dass die Redo Logs nur gegen System- oder Medienfehler schützen können (z. B. im Falle eines Stromausfalls), nicht aber gegen Fehleingaben eines Nutzers.

14.5.5 Archivierte Log Dateien

Um eine Oracle Datenbank, im Falle eines Fehlers, wiederherstellen zu können, werden die in den Redo Log Dateien gespeicherten Informationen benötigt. Da die Redo Logs jedoch zyklisch überschrieben werden, stehen diese Informationen nur für eine begrenzte Zeitspanne zur Verfügung. Um diese wichtigen Informationen für einen längeren (theoretisch unbegrenzten) Zeitraum verfügbar machen zu können, bietet Oracle die Möglichkeit, Kopien der Redo Logs anzulegen. Diese Kopien werden dann als „Archive Logs“ oder „Archivierte Log Dateien“ bezeichnet.

14.5.6 Die Alert Log Datei und Trace-Dateien

Abb. 14.14:
Oracle
Datenbank-
architektur



Trace-Dateien

Eine Oracle-Datenbank kennt zwei unterschiedliche Arten von Trace-Dateien. Einerseits sind dies Trace-Dateien, die auf Anforderung des Administrators erzeugt werden. Diese enthalten meist Performance- oder Diagnose-Informationen die zur Fehlerbehebung bzw. zum Tuning der Datenbank eingesetzt werden können.

Andererseits entstehen die meisten Trace-Dateien aufgrund von Fehlern bei den Hintergrundprozessen. Jeder Hintergrundprozess hat seine eigene Trace-Datei in die er Informationen schreibt.

Die Inhalte einer Trace-Datei sind teils für den Datenbankadministrator und teils für den Oracle-Support bestimmt.

Listing 14.1: Eine Tracedatei des Log Writers

```
/u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_lgwr_12331.trc
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - Production
With the Partitioning, OLAP and Data Mining options
ORACLE_HOME = /u01/app/oracle/product/11.2.0
System name: Linux
Node name: FEA11-119WD01
Release: 3.10.5.1-100.fc19
Version: #1 SMP Mon Jul 21 02:06:29 EDT 2011
Machine: i686
Instance name: ORCL
Redo thread mounted by this instance: 1
Oracle process number: 6
Unix process pid: 12331, image: oracle@FEA11-119WD01 (LGWR)
```

Die Alert Log Datei

Die „Alert Log Datei“ nimmt eine Sonderstellung unter den Trace-Dateien ein. Sie ist ein chronologisch sortiertes Protokoll, in dem alle in der Datenbank auftretenden Ereignisse und Fehler protokolliert werden. Man könnte sie auch als das „Tagebuch“ der Datenbank bezeichnen.

Seit Oracle 11g R1 existiert die Alert Log Datei in zwei Versionen, als Text-Datei und als XML-Datei. Beide gehören zum ebenfalls neu erschienenen „Automatic Diagnostic Repository“, kurz ADR. Das ADR ist im Wesentlichen ein Verzeichnis, dass von Oracle genutzt wird, um die verschiedenen Diagnose-Informationen geordnet abzulegen. Mit Hilfe dieser Dateien und des ADRCI (Automatic Diagnostic Repository Commandline Interface) können sogenannte Incident Packages geschnürt und an den Oracle Support verschickt werden.

Die beiden Alert Log Dateien befinden sich in Unterverzeichnissen des ADR:

- **Textdatei:** Sie heißt **alert_<SID>.ora**. Die Abkürzung „SID“ steht für „System Identifier“, den Namen der Datenbankinstanz. Bei einer Instanz mit Namen „ORCL“ hieße die Alert Log Datei dann: **alert_ORCL.ora**. Sie wird im Verzeichnis \$ADR_BASE/diag/product_type/product_id/instance_id/trace erzeugt.
- **XML-Datei:** Der Name der XML-Datei lautet **log.xml**. Diese wird im Verzeichnis \$ADR_BASE/diag/product_type/p angelegt.

- [CNCPT005]



Listing 14.2: Die Alert Log Textdatei

```
Fri Aug 14 13:33:43 2011 Process J000 died, see its trace file
Fri Aug 14 13:33:43 2011
kkjcre1p: unable to spawn jobq slave process
Fri Aug 14 13:33:43 2011
Errors in file /u01/app/oracle/admin/ORCL/bdump/orcl_cjq0_23189.trc:
Fri Aug 14 13:35:33 2011
Errors in file /u01/app/oracle/admin/ORCL/bdump/orcl_j000_18573.trc:
ORA-00600: Interner Fehlercode, Argumente: [keltnfy-ldmInit], [46]
Fri Aug 14 13:35:34 2011
Errors in file /u01/app/oracle/admin/ORCL/bdump/orcl_j000_18573.trc:
ORA-00600: Interner Fehlercode, Argumente: [keltnfy-ldmInit], [46]
Fri Aug 14 13:35:35 2011
Process J000 died, see its trace file
Fri Aug 14 13:35:35 2011
kkjcre1p: unable to spawn jobq slave process
Fri Aug 14 13:35:35 2011
Errors in file /u01/app/oracle/admin/ORCL/bdump/orcl_cjq0_23189.trc:
```

14.6 Die Oracle Hintergrundprozesse



Ein Prozess ist ein Thread oder ein ähnlicher Mechanismus eines Betriebssystems, der einen oder mehrere Arbeitsschritte durchführen kann (einige Betriebssysteme verwenden dafür die Bezeichnungen „Job“ oder „Task“). Jeder Prozess hat seinen eigenen Speicherbereich im Arbeitsspeicher.

Die Aufteilung der Arbeit in einzelne Prozesse geschieht, um die Datenbank Multi-user-fähig zu machen. Durch diese Aufteilung können sich (theoretisch) beliebig viele Nutzer mit einer Oracle-Datenbank verbinden, da jeder Nutzer seinen eigenen Serverprozess bekommt.

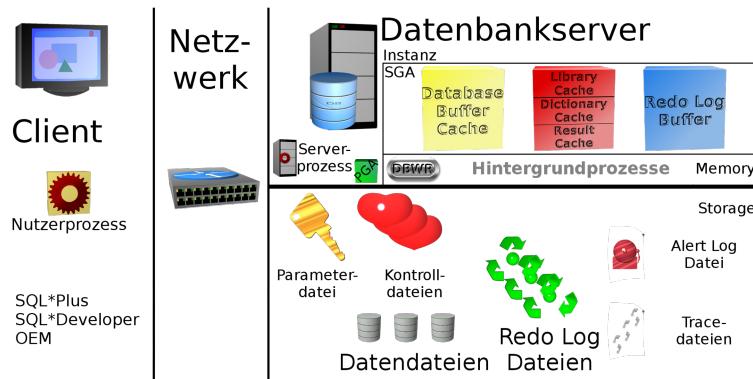
Eine Oracleinstanz kann viele verschiedene Hintergrundprozesse haben. Es sind jedoch nicht immer alle Hintergrundprozesse aktiv. Im Folgenden werden die wichtigsten Hintergrundprozesse beschrieben.

14.6.1 Der Database Writer (DBWn)

Der Database Writer ist für das Rückübertragen aller modifizierten Blöcke aus dem Database Buffer Cache in die Datendateien verantwortlich. Auch wenn meist ein einziger DBWn-Prozess für eine Da-

tenbank ausreichend ist, können mehrere gestartet werden, um die Performance des Systems zu erhöhen.

Abb. 14.15:
Oracle
Datenbank-
architektur



Da durch Schreibzugriffe die Anzahl der freien Blöcke im Database Buffer Cache abnimmt, ist es die Aufgabe des Database Writers, diese Anzahl nie unter einen bestimmten Schwellenwert fallen zu lassen. Wird der Schwellenwert dennoch unterschritten, müssen die Nutzerprozesse auf den Database Writer warten, bis der Schwellenwert wieder überstiegen wird.



Der Database Writer schreibt, wenn eine der folgenden Bedingungen erfüllt ist:

- Wenn sich zu viele modifizierte Blöcke im Database Buffer Cache befinden (Schwellenwert: weniger als 3 % freie Blöcke).
- Wenn ein Serverprozess zu lange nach freien Blöcken suchen muss.
- Beim Auslösen eines Checkpoints.
- Wenn die Instanz heruntergefahren wird (außer bei einem shutdown abort).
- Wenn ein Tablespace Offline oder Read Only gesetzt wird.
- Wenn eine Speicherstruktur, wie z. B. eine Tabelle gelöscht wird.

Aufbau und Organisation des Database Buffer Caches

Der Database Buffer Cache ist in zwei Listen aufgeteilt:

- Least recently used list (LRU)
- Buffer Checkpoint Queue

Die Buffer Checkpoint Queue ist eine Liste der geänderten Blöcke (dirty Blocks) des Buffer Caches, die noch nicht auf den Datenträger zurückgesichert wurden. Diese werden in aufsteigender Reihenfolge, nach ihrer Low Redo Byte Adress (low RBA) gespeichert.

Die LRU List enthält drei Blockarten:

- freie Blöcke (**clean blocks**)

Ein Block gilt als clean, wenn er bisher noch nicht verwendet wurde oder aber, wenn sein Inhalt nach einer Modifikation auf die Festplatte zurückübertragen wurde (d. h. das Blockabbild im Buffer Cache und der Originalblock auf der Festplatte haben den gleichen Inhalt).

- Blöcke die aktuell in Verwendung sind (**pinned blocks**)
- modifizierte Blöcke (**dirty blocks**)

Wird ein Block im Buffer Cache verändert, dann ändert sich sein Status auf dirty (d. h. Das Blockabbild im Buffer Cache hat einen anderen Inhalt als der Originalblock auf der Festplatte). Blöcke mit diesem Status müssen erst noch auf die Festplatte übertragen werden.

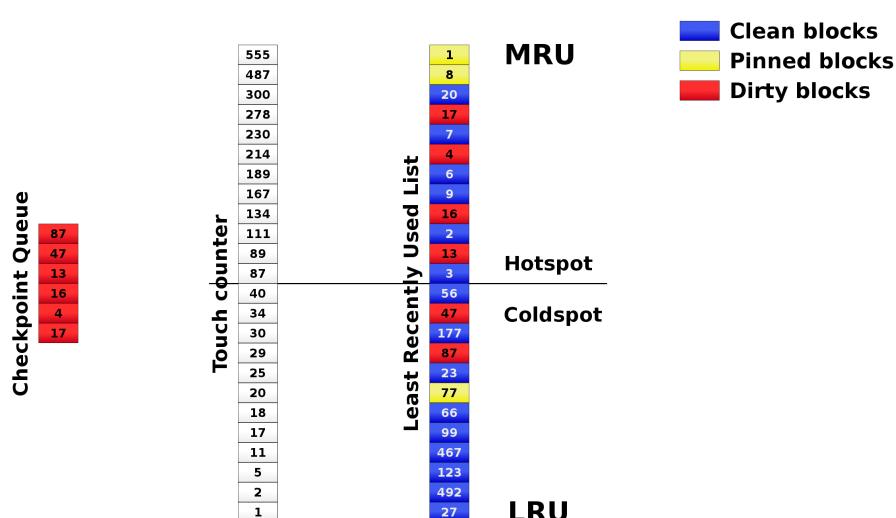


Abb. 14.16:
Aufbau des
Database Buffer
Caches

Jeder Block in der LRU hat einen „Touchcounter“, einen Zähler, der die Anzahl der Zugriffe auf den Datenblock zählt. Er wird für die Verwaltung der Blöcke in der LRU benötigt.

Verwaltung des Database Buffer Caches

Sucht ein Serverprozess einen bestimmten Block im Database Buffer Cache, beginnt er mit seiner Suche am MRU²-Ende der LRU³-Liste, da die Wahrscheinlichkeit, dass sich der gesuchte Block dort befindet

²MRU = Most Recently Used

³LRU = Least Recently Used

höher ist, als die, dass er sich am LRU-Ende befindet. Der Serverprozess durchsucht die LRU-Liste sequenziell, Block für Block.

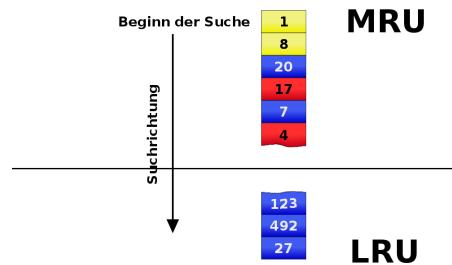


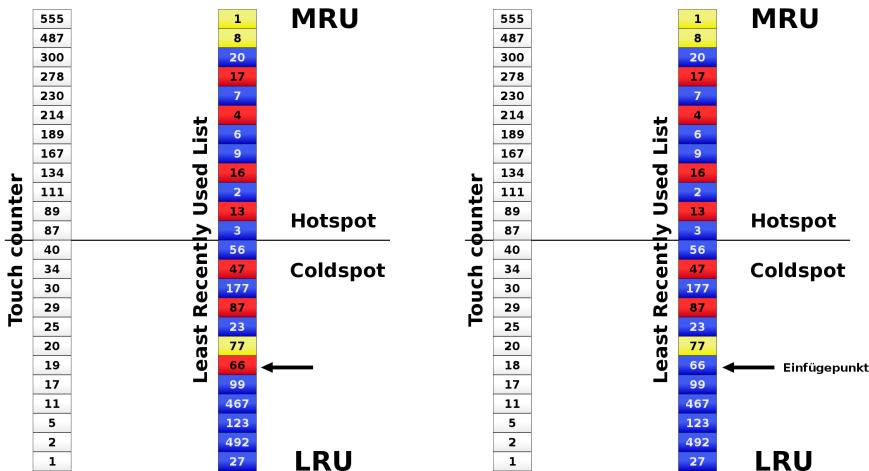
Abb. 14.17:
Suche nach freien
Blöcken im
Buffer Cache

Wird der gesuchte Block gefunden (cache hit), kann er verarbeitet werden. Wird er nicht gefunden (cache miss), muss der Block in den Database Buffer Cache geladen werden.

Beim Laden eines Blockes in den Database Buffer Cache, wird er auf halber Höhe des Coldspot eingeschrieben. Um einen neuen Block in den Buffer Cache übertragen zu können, muss der Serverprozess vorher nach freien Blöcken suchen. Dabei beginnt er seine Suche am LRU-Ende der LRU-Liste. Trifft er dabei auf dirty blocks, notiert er die Blockadressen dieser Blöcke in der Checkpoint Queue, so dass der Database Writer diese in die Datendateien übertragen kann.

Wurden genügend freie Blöcke gefunden, können die neuen Blöcke in den Buffer Cache übertragen werden. Werden keine freien Blöcke gefunden, muss der Serverprozess den Database Writer damit beauftragen, die Checkpoint Queue abzuarbeiten, um Platz im Database Buffer Cache zu schaffen.

Abb. 14.18:
Einfügen neuer
Blöcke im Buffer
Cache



Der Touchcounter zählt die Zugriffe auf einen Datenblock. Wird eine bestimmte Anzahl Zugriffe erreicht, wird der Block relativ von seiner aktuellen Position aus, um 50 % nach oben verschoben. So kann der Block aus dem Coldspot an das obere Ende der LRU-Liste, den Hotspot, wandern. Dabei verdrängt er andere Blöcke nach unten. Wird auf einen Block nur selten zugegriffen, erreicht er irgendwann den Coldspot und bekommt früher oder später den Status clean.



Die Serverprozesse sind dafür zuständig, angeforderte Blöcke in den Database Buffer Cache zu übertragen und diese dort zu lesen bzw. nach den Anforderungen der Nutzer entsprechend zu modifizieren. Der Database Writer ist als einziger dafür zuständig modifizierte Blöcke auf die Festplatte zurückzuübertragen.

14.6.2 Der Log Writer (LGWr)

Der Log Writer Prozess, der oft auch als Redo Thread bezeichnet wird, ist zuständig für das Management der Redo Logs. Er schreibt die Redo Log Einträge (Redo Records) im Redo Log Buffer in die Redo Log Dateien. Der Redo Log Buffer ist zyklisch aufgebaut. Wenn der Log Writer alle Einträge aus dem Buffer in die Dateien geschrieben hat, können die alten Werte im Redo Log Buffer überschrieben werden.

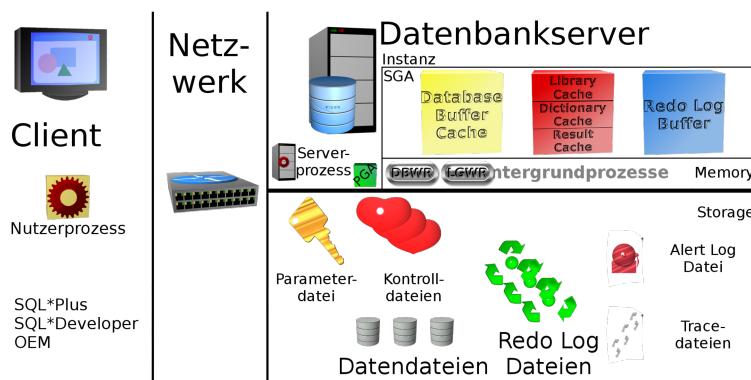


Abb. 14.19:
Oracle
Datenbank-
architektur

Der Log Writer tritt in Aktion, wenn die folgenden Bedingungen erfüllt sind:

- Periodisch alle 3 Sekunden (Timeout)
- Wenn sich der Redo Log Buffer bis zu einem Drittel gefüllt hat (Standardgröße 512 KB)
- Wenn im Redo Log Buffer mehr als 1 MB Redo Informationen enthalten sind
- Bevor der Database Writer schreibt
- Wenn ein Nutzer eine Transaktion mit `COMMIT` beendet



Der Log Writer Prozess schreibt synchron in alle Member der aktiven Log Gruppe. Ist einer defekt oder nicht verfügbar, setzt der Log Writer das Schreiben in den anderen Dateien fort und gibt eine Fehlermeldung in seiner Trace-Datei und der Alert Log Datei aus.



Es sollten immer so viele Redo Log Gruppen vorhanden sein, dass der Log Writer zu jeder Zeit eine freie Gruppe finden kann, ohne warten zu müssen.

Der Log Writer und der Database Writer sind zwei Prozesse, die von einander abhängig sind. Bevor der Database Writer einen dirty block in die Datendateien schreiben darf, müssen vorher alle mit diesem Block verbundenen Redo Log Einträge durch den Log Writer in die Redo Logs übertragen worden sein. Dies wird durch das *write-ahead Protokoll* gewährleistet.

Log Switch

Wenn der Log Writer eine Gruppe zu 100 % gefüllt hat wechselt er in die nächste freie Gruppe. Dieser Vorgang des Wechsels einer Redo Log Gruppe heißt Log Switch. Aus der Forderung, dass der Log Writer immer eine freie Gruppe benötigt, in die er wechseln kann, resultiert die Forderung, dass mindestens zwei Redo Log Gruppen vorhanden sein müssen. Findet der Log Writer keine freie Gruppe zum Wechseln, bleibt die Datenbank stehen.

Der Log Writer benutzt alle vorhandenen Gruppen der Reihe nach. Ist er bei der letzten Gruppe angekommen, versucht er die erste Gruppe erneut zu nutzen, d. h. die Redo Log Gruppen werden im Kreis immer wieder genutzt. Der Inhalt einer bereits befüllten Gruppe geht bei erneuter Nutzung verloren. Um einen solchen Verlust zu vermeiden, kann ein weiterer Hintergrundprozess, der Archiver eingeschaltet werden.



Als Faustregel gilt: Es sollte ca. alle 20 Minuten ein Log Switch stattfinden. Dies kann durch die Größe der Redo Log Dateien beeinflusst werden.

Die Log Sequence Number (LSN)

Wenn der Log Writer beginnt, eine Redo Log Gruppe zu benutzen, ordnet er ihr eine Log Sequence Number zu. Die Log Sequence Number ist eine fortlaufende Nummer, anhand derer die Reihenfolge der Nutzung der Redo Log Gruppe erkannt werden kann. Wenn der Log Writer z. B. die Gruppe 1 zu erst benutzt, erhält diese Gruppe die LSN 1. Nach einem Log Switch auf Gruppe 2 erhält Gruppe 2 die LSN 2. Bei einem weiteren Log Switch hin zu Gruppe 1 erhält Gruppe 1 die LSN 3 usw.

Status einer Redo Log Gruppe

Jede Redo Log Gruppe hat einen bestimmten Status. Der Status gibt an, ob die Gruppe gerade benutzt wird, bzw. ob sie für die Nutzung frei ist. Der Status kann sein:

- **Current:** Die Redo Log Gruppe, die aktuell durch den Log Writer benutzt wird hat den Status Current.
- **Active:** Eine Redo Log Gruppe, die für ein Instance-Recovery benötigt wird hat den Status Active. Eine Redo Log Gruppe wird solange für ein Instance-Recovery benötigt, bis der Database Writer alle betreffenden Oracle-Blöcke aus dem Database Buffer Cache in die Datendateien geschrieben hat. Hat der DBWn seine Arbeit vollendet, sind die Informationen dieser Redo Log Gruppe nicht mehr für ein Instance-Recovery von Nötigen und der Status der Redo Log Gruppe wechselt zu *Inactive*.
- **Inactive:** Redo Log Gruppen, die nicht mehr für ein Instance-Recovery benötigt werden haben diesen Status.

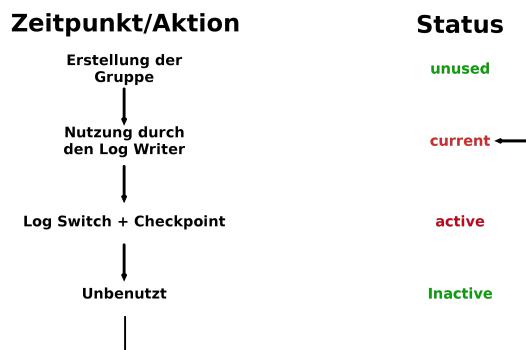


Abb. 14.20:
Status einer Redo Log Gruppe

14.6.3 Der Checkpoint Prozess (CKPT)

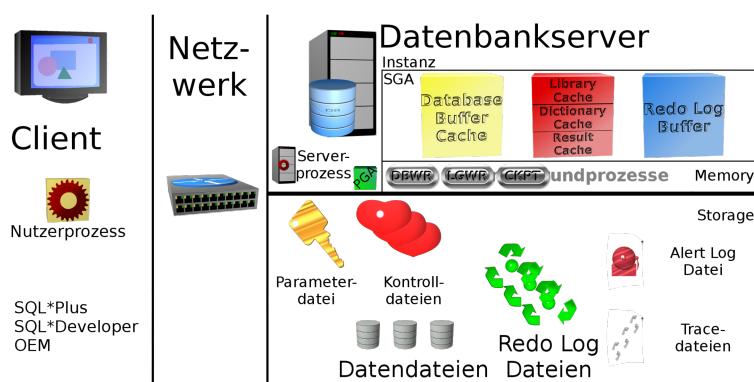


Abb. 14.21:
Oracle Datenbank-architektur

Der Checkpoint Prozess, kurz CKPT, gehört seit Oracle 8 zum Kreise der Prozesse, die zwingend notwendig sind für den Betrieb einer Oracle-Datenbank. Sein Name leitet sich von einem Ereignis ab, das in regelmäßigen Intervallen auftritt, dem Checkpoint.

Checkpoints

Die offizielle Definition eines Checkpoints lautet:



Ein Checkpoint ist ein Ereignis, das durch andere Ereignisse ausgelöst wird (Log Switch, Manuell).

Checkpoints haben die Aufgabe eine Datenbank konsistent zu halten. Dies geschieht, in dem bei jedem Auftreten eines Checkpoints die geänderten Daten aus dem Database Buffer Cache auf den Datenträger geschrieben werden, so wie dies bereits beschrieben wurde. Die hauptsächliche Last eines Checkpoints tragen somit der DB Writer und der Log Writer.

Nach dem Abschluss eines Checkpoints muss aber noch Verwaltungsarbeit durchgeführt werden. Die Datenbank muss sich „notieren“, dass, bzw. wann der Checkpoint beendet wurde. Dies geschieht, in dem der Checkpoint Prozess (CKPT), am Ende eines Checkpoints die System Change Number, kurz SCN, in die Kontrolldateien und die Header aller Datendateien schreibt.

Die System Change Number (SCN)



Die System Change Number (SCN) ist eine fortlaufende Nummer, anhand derer das Alter einer Oracledatenbank bestimmt werden kann. Sie wird durch die verschiedensten Nutzeraktionen und durch die Arbeit der Hintergrundprozesse inkrementiert.

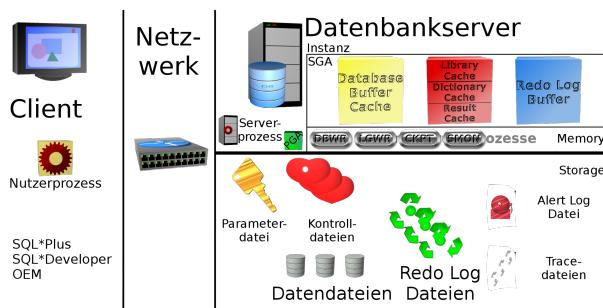
Die aktuelle SCN kann mit der Prozedur GET_SYSTEM_CHANGE_NUMBER, aus dem PL/SQL-Paket DBMS_FLASHBACK herausgefunden werden. Sie ist in der Kontrolldatei eingetragen und wird auch als „Referenz-SCN“ bezeichnet. Die SCN des letzten Checkpoint ist im Feld CHECKPOINT_CHANGE# in der V\$-View V\$DATABASE zu finden.



- [p665]

14.6.4 Der System Monitor (SMON)

Abb. 14.22:
Oracle
Datenbank-
architektur



Der SMON kann als eine Art „Reinigungskraft der Datenbank auf Betriebssystemebene“ verstanden werden. Zu seinen Aufgaben zählt:

- **Instance-Recovery:** Wird die Datenbank nach einem Crash hochgefahren, muss der SMON die inkonsistente Datenbank in einen konsistenten Zustand überführen.
- **Aufräumarbeiten:** Der SMON muss immer wieder Aufräumarbeiten in verschiedenen Systemtabellen der Datenbank (z. B. OBJ\$) durchführen.
- **Undo Segmente schrumpfen:** Der SMON schrumpft Undo Segmente automatisch auf ihre optimale Größe.
- **Abschalten von Undo Segmenten:** Legt der DB-Admin fest, dass ein Undo Segment abgeschaltet werden soll, so führt der SMON diese Tätigkeit aus.

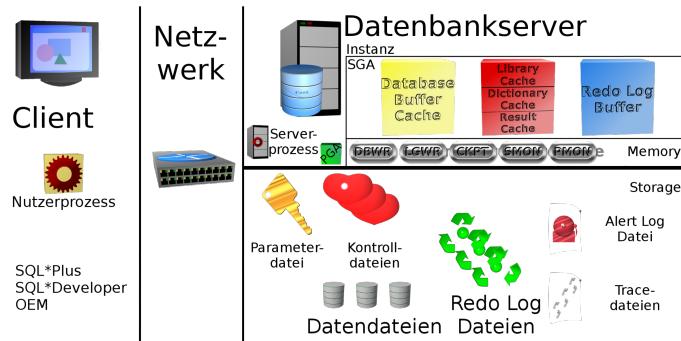
14.6.5 Der Prozess Monitor (PMON)

Der Prozess Monitor stellt die Ergänzung zum System Monitor dar. Er ist die „Reinigungskraft auf Datenbankebene“. Zu seinen Aufgaben zählt:

- **Serverprozesse aufräumen:** Wird eine Session abnormal getrennt bleibt der zugehörige Serverprozess als sogenannter „Zombieprozess“ stehen. Die Aufgabe des Prozessmonitors ist es, solche Zombies zu beenden.
- **Sperren lösen**
- **Zurückrollen von Transaktionen:** Alle Transaktionen die vor einem Instanz-Crash nicht beendet wurden, müssen durch den PMON zurückgerollt werden.

- **Hintergrundprozesse überwachen:** PMON ist dafür zuständig die anderen Hintergrundprozesse zu überwachen und diese bei Bedarf neuzustarten.
- **Dynamic Service Registration**

Abb. 14.23:
Oracle
Datenbank-
architektur

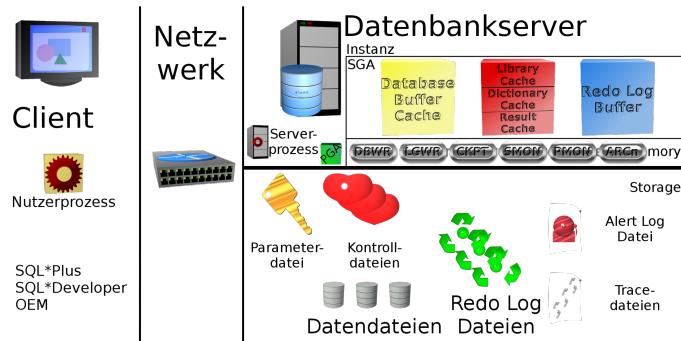


14.6.6 Der Archiver (ARCn)

Der Archiver Prozess ist dafür zuständig, gefüllte Redo Log Dateien, nach einem „Log switch“, an einem definierten Speicherort zu sichern. Eine Redo Log Gruppe muss bei aktiver Archivierung erst archiviert werden, bevor sie durch den Log Writer erneut genutzt werden kann. Ist die Archivierung nicht aktiviert, kann eine Redo Log Gruppe sofort wieder verwendet werden.

Sind alle Member einer Redo Log Gruppe beschädigt oder nicht mehr vorhanden, kann der Archiver diese Gruppe nicht mehr archivieren. Dies führt in dem Moment zum Stillstand der Datenbank, in dem der Log Writer versucht, diese Gruppe erneut zu nutzen.

Abb. 14.24:
Oracle
Datenbank-
architektur



- [CNCPT020]



Eine Redo Log Gruppe ist in zwei Fällen für die Nutzung frei:

- Direkt nach ihrer Erstellung
- Wenn die Gruppe unbenutzt ist und wenn die Archivierung beendet wurde.

15 Installieren einer Oracle 11g Release 2 Datenbank

Inhaltsangabe

15.1 Aufgaben und Verantwortungsbereich eines DBA

Der Verantwortungsbereich eines Datenbankadministrators kann die folgenden Aufgaben einschließen:

- Installation und Upgrade des Datenbankservers und der Nutzeranwendungen
- Planen und Überwachen des Speicherbedarfs der Datenbank auf dem Datenträger
- Erstellen von Storage Strukturen in der Datenbank (z. B. Tabellen und Indizes)
- Nutzerkonten verwalten und überwachen der Sicherheit
- Performance Monitoring und Tuning
- Entwickeln von Backup- und Recoverystrategien
- Verwalten von archivierten Daten
- Durchführen von Backup und Recovery

15.1.1 Planen und durchführen einer Oracle Installation

Lesen der Releasenotes

Die Releasenotes sind mit Sicherheit der am wenigsten beachtete Teil einer Software Dokumentation. So wenig Beachtung sie jedoch finden, um so wichtiger sind sie tatsächlich. Sie enthalten wichtige Neuerungen zur Vorgängerversion und andere Hinweise, die bei einem Update auf eine neue Version, der Auswahl der notwendigen Hardware und anderen Preinstallation Tasks von großer Bedeutung sein können.



- [e23557]
- [e23558]

Evaluieren der vorhandenen Hardware

Vor der Installation sollte der DBA prüfen, wie die vorhandene Hardware bestmöglich durch die Oracle-Datenbank und die benötigten Anwendungsprogramme genutzt werden kann. Diese Überprüfung sollte die folgenden Überlegungen einschließen:

- Wie viele Datenträger sind für die Datenbank verfügbar?
- Wie viel Arbeitsspeicher ist für die Oracle Instanz verfügbar?

- [BABFDGHJ]
- [i1011417]



Planen der Datenbank

Zum Planen einer Datenbank gehören die folgenden Aufgaben:

- Planen der logischen Speicherstrukturen der Datenbank
- Prüfen des Datenbankdesigns
- Entwickeln von Backupstrategien

Die Planung der logischen Speicherstrukturen der Datenbank ist wichtig, um deren Einflüsse auf die Systemperformance erkennen zu können. Beispielsweise ist es entscheidend, ob in einer Datenbank große Objekte, wie z. B. Bilder oder ISO-Images gespeichert werden oder nur normale Text/Zahlen-Werte. Auch das Datenbankdesign ist entscheidend für die Performance der Datenbank. Meist hat der DBA zwar keinen direkten Einfluss auf das Design, er kann aber Hinweise und Verbesserungsvorschläge geben.

15.1.2 Download und Installation von Patches

Um die Datenbank sicherer zu machen und um Bugs zu beheben, muss der DBA von Zeit zu Zeit Patches oder Patchsets installieren. Ein Patch (auch „single interim patch“ genannt) behebt ein einzelnes spezielles Problem und ist nicht bei jeder Installation notwendig. Ein Patchset ist eine Sammlung von Patches, die so erstellt wurde, das sie für jeden Kunden passt. Ein Patchset hat eine Releasenummer. Wurde

beispielsweise die Oracle Version 11.2.0.0 installiert, dann hat das erste Patchset die Versionsnummer 11.2.0.1.

15.2 OFA - Die Optimal Flexible Architecture

15.2.1 Überblick über die OFA

Die Optimal Flexible Architecture ist eine Menge von Namenskonventionen und Konfigurationsrichtlinien die zuverlässige Oracle Installationen mit minimalem Wartungsaufwand sicherstellen sollen. Der OFA-Standard wurde entwickelt um:

- eine große Menge komplexer Anwendungen zusammen mit deren Nutzdaten auf einem Datenträger zu speichern und dabei Bottlenecks und schlechte Performance zu vermeiden,
- administrative Routineaufgaben zu erleichtern, wie z. B. Backups,
- das Wechseln zwischen verschiedenen Oracle Instanzen zu erleichtern,
- es dem Administrator zu ermöglichen angemessen auf das Wachstum einer Datenbank reagieren zu können,
- die Fragmentierung von Teilen der Datenbank möglichst gering zu halten.

Die OFA kann als eine Art „Sammlung von guten Manieren“ bei der Erstellung von Verzeichnissen für Oracle angesehen werden. Der Oracle Universal Installer platziert automatisch alle Oracle Komponenten in Verzeichnissen, die gemäß OFA erstellt wurden. Auch wenn die Nutzung einer OFA kein Muss ist, wird dies in jedem Falle von Oracle empfohlen.

Der Oracle Universal Installer (OUI) trennt die Datenbanksoftware von den Nutzdaten. Er legt die Software im Oracle Homeverzeichnis \$ORACLE_HOME ab. Die Nutzdaten werden in \$ORACLE_BASE/oradata abgelegt. \$ORACLE_BASE und \$ORACLE_HOME sind Umgebungsvariablen die Verzeichispfade enthalten.

Der Vorteil an dieser Methode ist, wenn eine neue Version der Datenbanksoftware installiert wird, hat diese ein neues Oracle Homeverzeichnis. Sie kann auf Zuverlässigkeit getestet werden und die alte Version kann nach erfolgreichen Tests der neuen Version gelöscht werden, ohne dass dabei Probleme entstehen.

15.2.2 Charakteristika einer OFA-Konformen Installation

Eine OFA-Konforme Installation besitzt die folgenden Charakteristika:

- Unabhängige Unterverzeichnisse

Dateien unterschiedlicher Arten werden in unterschiedliche Unterverzeichnisse gelegt. Somit ist eine Dateiart wenig bis gar nicht betroffen, wenn eine andere verändert oder gelöscht wird.

- Konsequente Namenskonventionen für die Dateien der Datenbank

Die Dateien der Oracle-Datenbank können einfach anhand der Verzeichnisstruktur von anderen Dateien unterschieden werden. Datenbankdateien unterschiedlicher Oracle Versionen können auf die gleiche Art und Weise sehr einfach von einander unterschieden werden.

- Integrität der Oracle-Home-Verzeichnisse

Oracle-Home-Verzeichnisse können hinzugefügt, verschoben oder gelöscht werden, ohne dass die betroffene Software dadurch beschädigt werden würde.

- Metadaten unterschiedlicher Datenbanken von einander trennen

Die strikte Trennung administrativer Daten von einer Datenbank gegenüber allen anderen erleichtert administrative Arbeiten deutlich und schafft eine übersichtliche und zuverlässige Struktur.

- Strikte Trennung unterschiedlicher Arten von Nutzdaten

Nutzdaten unterschiedlicher Arten können von einander getrennt werden, um eine bessere Performance zu schaffen

- Verteilung von I/O-Last auf alle verfügbaren Datenträger

15.2.3 Die OFA-Konventionen

Die OFA schreibt folgende Dateiendungen für Datenbankdateien vor:

- *.ctl Kontrolldatei
- *.dbf Datendatei
- *.log Redo Log Datei

Das Oracle Base-Verzeichnis

Die Umgebungsvariable \$ORACLE_BASE enthält den Pfad des Oracle Basisverzeichnisses. Es stellt die Wurzel des Oracleverzeichnisbaumes dar. Der OUI setzt für \$ORACLE_BASE automatisch den Wert

/u01/app/oracle

Das Oracle Homeverzeichnis

Die Umgebungsvariable \$ORACLE_HOME enthält das Oracle Homeverzeichnis. Es ist der Speicherort für die Oracle Datenbanksoftware. \$ORACLE_HOME ist ein Unterverzeichnis von \$ORACLE_BASE. Zum Beispiel:

/u01/app/oracle/product/11.2.0/orcl



- [BABHAIJ]

15.3 Durchführen einer Oracle Installation auf Linux

In dieser Sektion wird die Installation einer Oracle 11g R2 Datenbank auf einem Oracle Enterprise Linux 6 System beschrieben. Alle Voreinstellungen die am Betriebssystem gemacht werden müssen, sind bereits getätigt worden. An dieser Stelle wird nur noch die reine Softwareinstallation beschrieben.

15.3.1 Der Installationsbeginn

Das Starten der Installation erfolgt durch den Aufruf: `/media/CDROM/runInstaller`. Die Pfadangabe `/media/CDROM` kann von System zu System variieren. Hierbei handelt es sich um den Pfad, an dem das CD/DVD-Laufwerk oder die ISO-Datei gemountet wurde. `runInstaller` ist das Kommando zum Starten des Oracle Universal Installer, kurz OUI.

Schritt 1- Sicherheitsupdates

Der OUI ist ein grafisches Werkzeug zur Installation der Oracle-Datenbank-Software. Der Setup-Vorgang umfasst insgesamt neun Schritte.

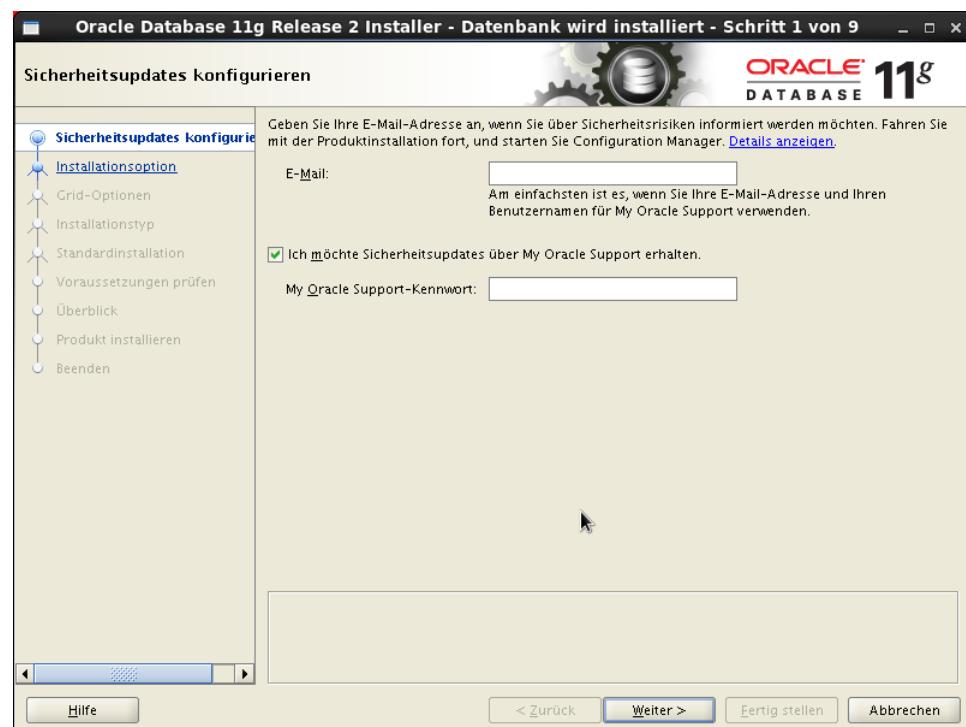


Abb. 15.1:
Schritt 1 - Sicherheitsupdates

In Schritt 1 der Installation wird das Fenster für die Konfiguration von Sicherheitsupdates angezeigt. Hier kann der Administrator seine E-Mail-Adresse eingeben, um über Sicherheitsupdates informiert zu werden. Zusätzlich kann er auch sein „My Oracle Support“ Kennwort angeben, so dass Sicherheitsupdates direkt bezogen werden können.

Sollte der Administrator in diesem Schritt „vergessen haben“, seine E-Mail-Adresse anzugeben, wird er prompt auf diesen „Fehler“ hingewiesen.

Ein Klick auf den „Ja“-Button löst das Problem.

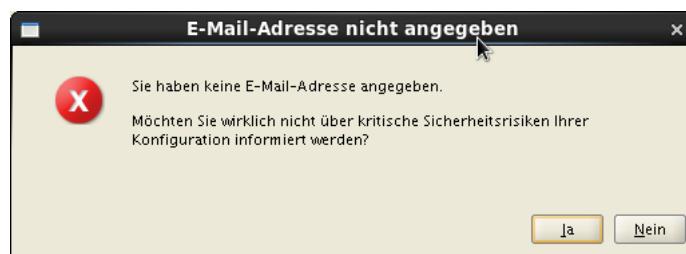


Abb. 15.2:
E-Mail-Adresse
vergessen?

Schritt 2 - Installationsoptionen

In diesem Installationsschritt wird abgefragt, ob:

- Die Software installiert und eine neue Datenbank angelegt werden soll,
- nur die Software installiert werden soll oder
- ob ein Upgrade einer bestehenden Datenbank durchgeführt werden soll.

Schritt 3 - Grid-Installation Optionen

Dieser Schritt ist seit Oracle 11g neu. Die Version 11 der Oracle-Datenbank beinhaltet eine Software, die als „Grid Infrastructure“ bezeichnet wird. Diese wird jedoch nur dann benötigt, wenn eine Real Application Cluster Installation durchgeführt werden soll.

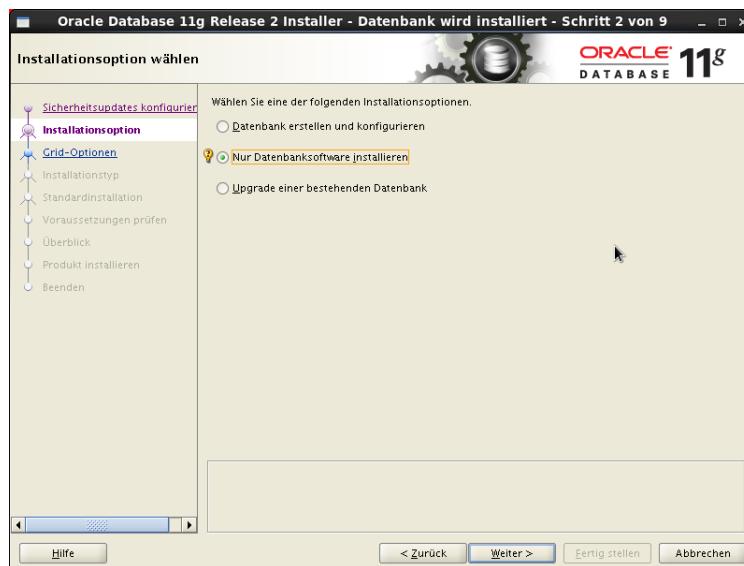


Abb. 15.3:
Schritt 2 - Installa-
tionsoptionen

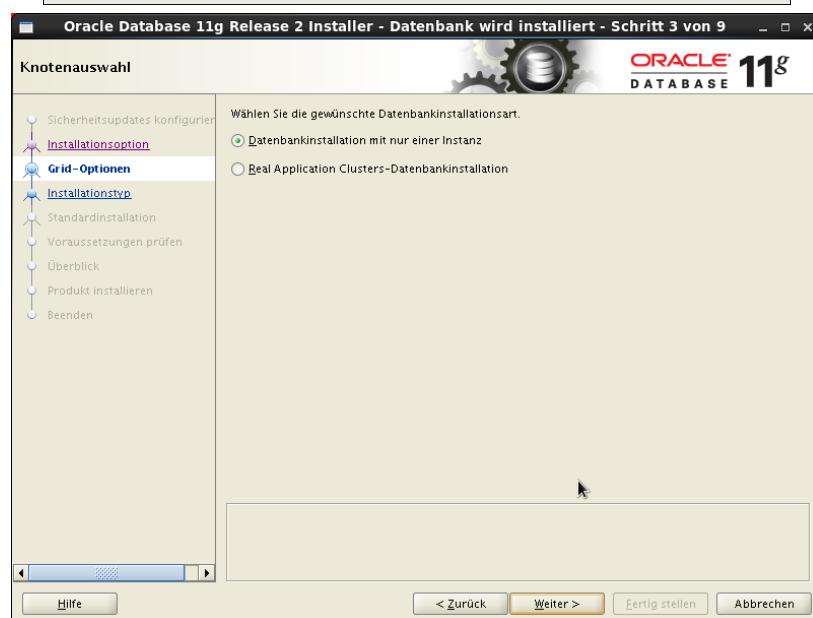


Abb. 15.4:
Schritt 3 -
Grid-Optionen

Schritt 4 - Produktsprachen

Oracle 11g R2 wird standardmäßig in den Sprachen Deutsch und Englisch installiert. Zusätzlich stehen weitere 45 Sprachen zur Verfügung.

Schritt 5 - Datenbank Edition

Die Oracle-Datenbank existiert in verschiedenen Editionen. Es gibt die:

- Enterprise Edition

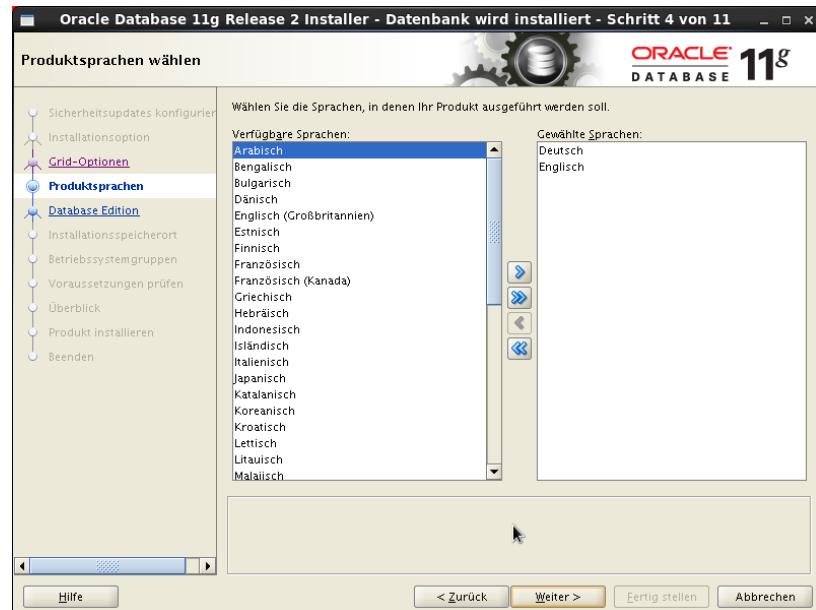


Abb. 15.5:
Schritt 4 -
Produktsprachen

- Standard Edition
- Standard Edition One
- Personal Edition (nur unter Windows)

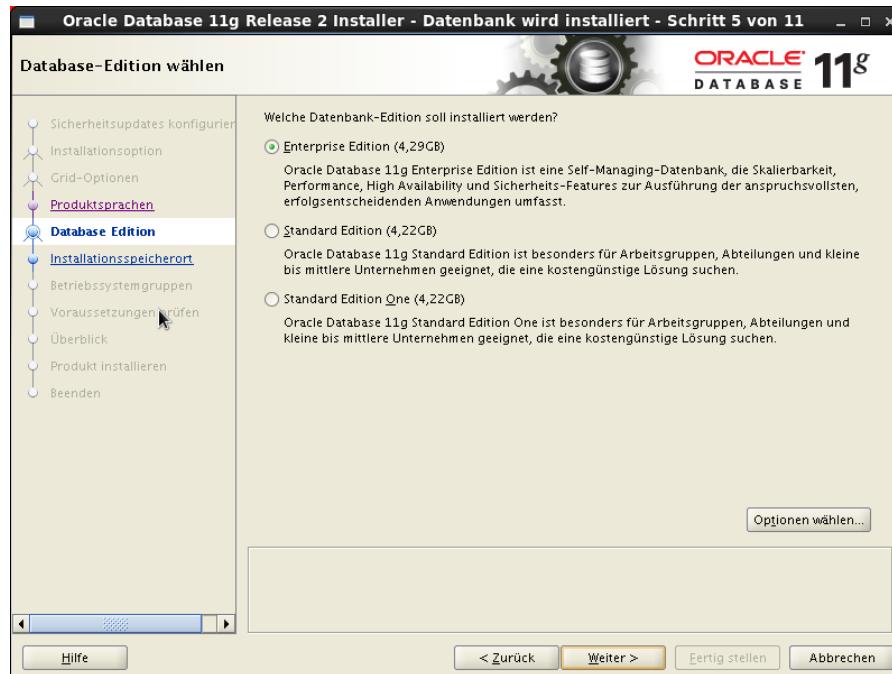


Abb. 15.6:
Schritt 5 -
Datenbank
Edition

Der Unterschied zwischen diesen Editionen liegt in deren Funktionsumfang und in den Lizenzkosten. Die Enterprise Edition ist das umfangreichste Paket. Sie enthält alle Features und sie bietet die Mög-

lichkeit „Datenbank Optionen“ hinzu zu installieren, um die Fähigkeiten der Datenbank noch mehr zu erweitern.

Die Standard Edition hat einen eingeschränkten Funktionsumfang. Sie ist nur noch für mittlere Unternehmen geeignet, da Features fehlen, die im Umgang mit großen Datenmengen absolut unverzichtbar sind. Für diese Edition ist es auch nicht möglich, Datenbank Optionen hinzuzufügen.

Die Standard Edition One ist das kleinste Softwarepaket. Sie hat einen sehr stark eingeschränkten Funktionsumfang und ist somit nur noch für kleine Unternehmen oder Abteilungen gedacht.

Sollte die Enterprise Edition ausgewählt worden sein, so können über den Button „Optionen wählen“ weitere Datenbankoptionen hinzugefügt werden.

Abb. 15.7:
Schritt 5 -
Datenbank
Optionen
hinzufügen



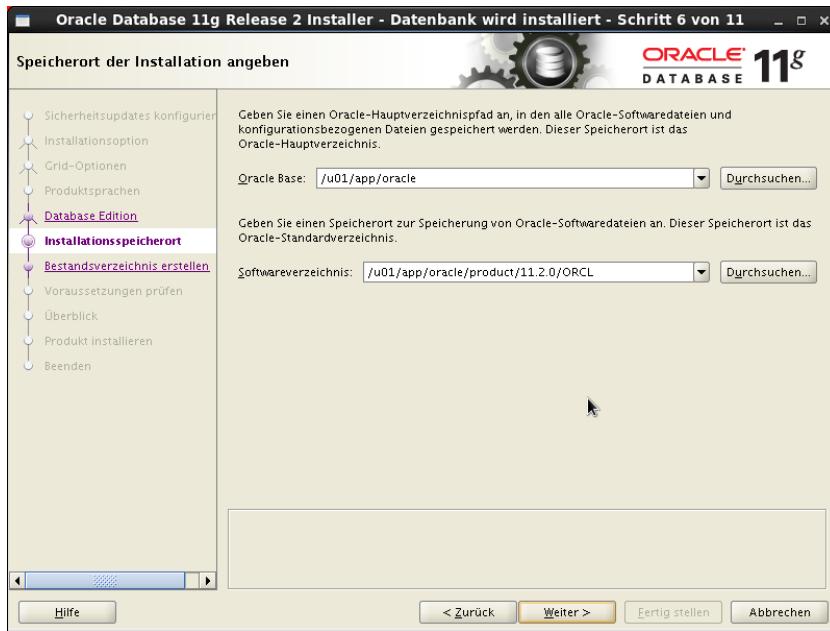
Einige Datenbankoptionen müssen eigenständig lizenziert werden, wodurch zusätzliche Lizenzkosten entstehen!

Schritt 6 - Installationsspeicherort

Um diesen Schritt erfolgreich zu beenden, müssen zwei Angaben gemacht werden:

- **Oracle Base:** Das Oracle Base-Verzeichnis ist die Wurzel des Installationspfades. Alle weiteren Angaben beziehen sich standardmäßig auf dieses Verzeichnis.
- **Softwareeverzeichnis:** Dies ist das \$ORACLE_HOME-Verzeichnis. Hierhinein wird die Oracle-Software installiert. Bei diesem Verzeichnis handelt es sich um ein Unterverzeichnis von \$ORACLE_BASE.

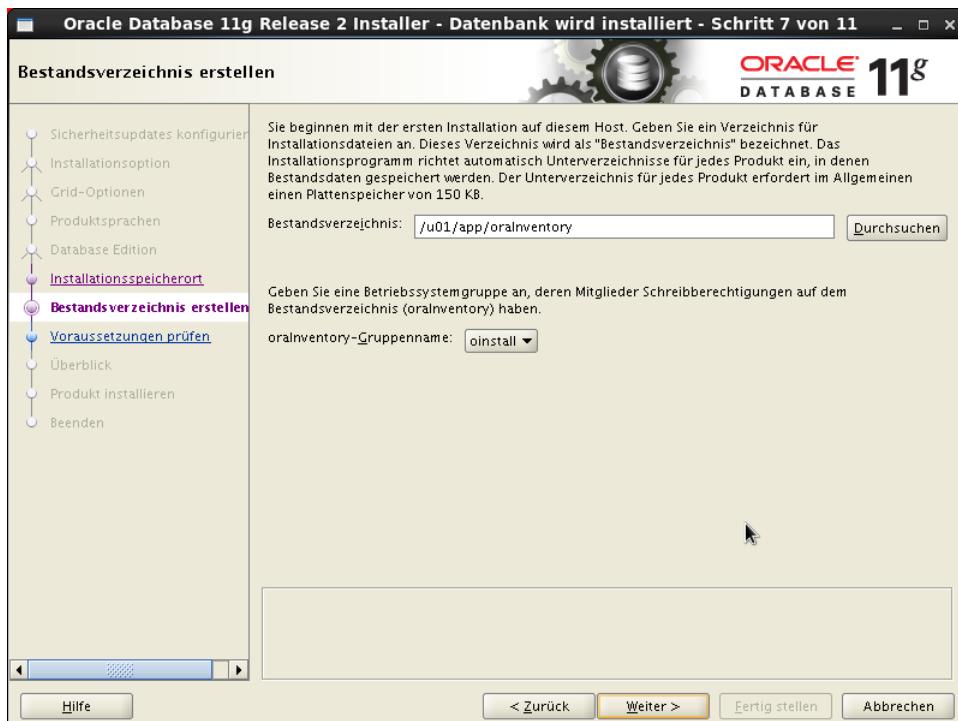
Abb. 15.8:
Schritt 6 - Installationsspeicherort



Schritt 7 - Bestandsverzeichnis erstellen (nur Linux)

Beim Oracle Bestandsverzeichnis handelt es sich um ein Verzeichnis, in dem alle installierten Oracle-Produkte Bestandsdateien anlegen. Es wird bei der ersten Installation eines beliebigen Oracle-Produktes angelegt. Alle Produkte erstellen dort Unterverzeichnisse für ihre eigenen Bestandsdaten.

Abb. 15.9:
Schritt 7 - Bestandsverzeichnis erstellen



Schritt 8 - Berechtigte Betriebssystemgruppen

Bei der Installation der Oracle-Datenbank werden zwei Betriebssystemgruppen angelegt, welche für die „Betriebssystemauthentifizierung“ relevant sind: OSDBA und OSOPER. Mitglieder der Gruppe OSDBA haben innerhalb der Datenbank alle Berechtigungen. Wie der Name der Gruppe sagt, sollten nur DBAs Mitglied sein.

Alle Mitglieder der Betriebssystemgruppe OSOPER haben eingeschränkte administrative Rechte. Für die tägliche Arbeit eines Oracle-DBAs ist diese Gruppe faktisch irrelevant.

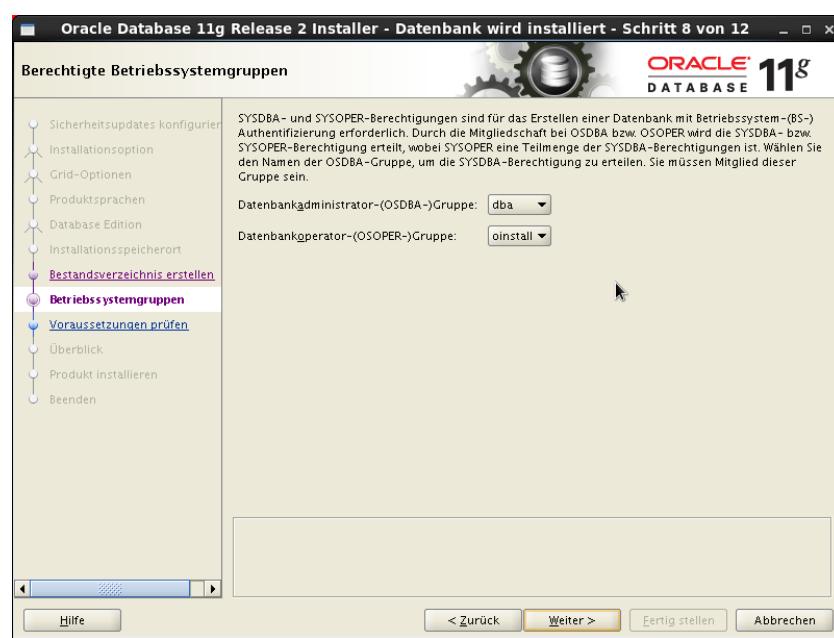


Abb. 15.10:
Schritt 8 -
Berechtigte
Betriebssystem-
gruppen

Schritt 9 - Voraussetzungen prüfen

In diesem Schritt werden alle Bedingungen geprüft, die betriebssystemseitig vor der Installation erfüllt sein müssen.

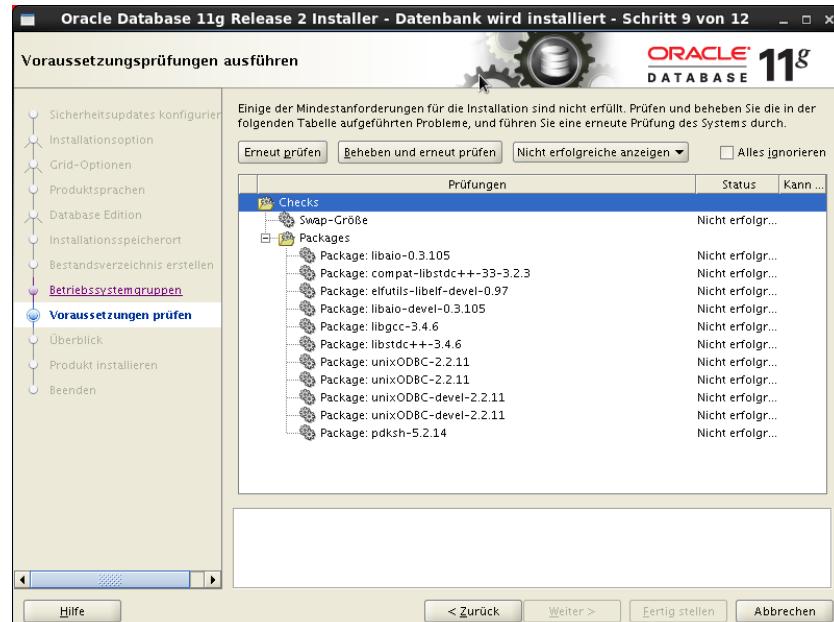
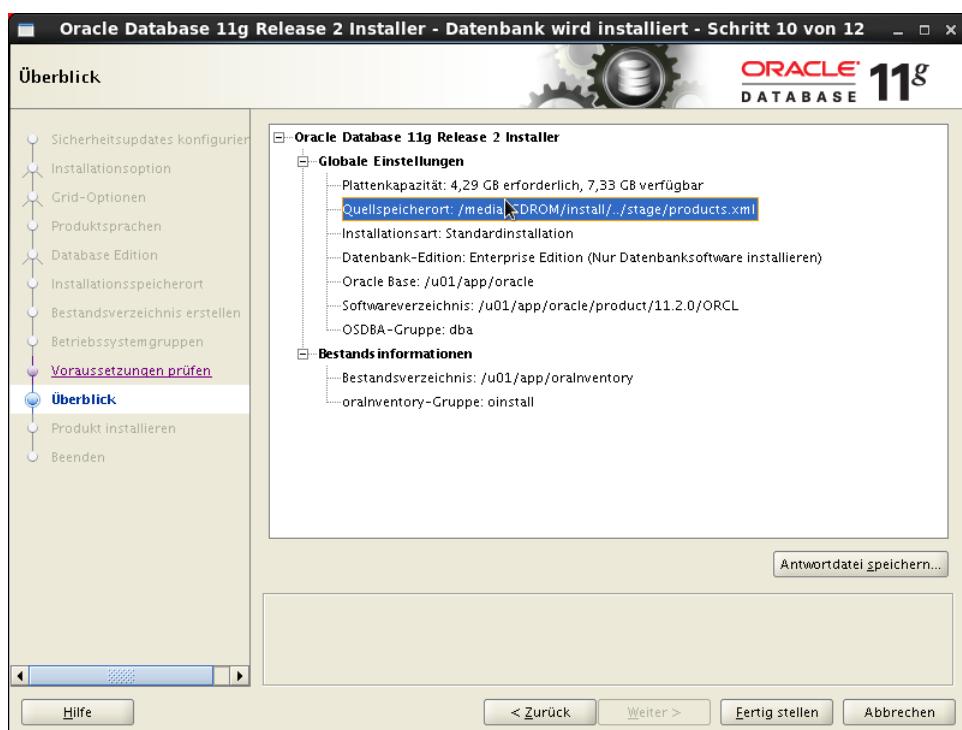


Abb. 15.11:
Schritt 9 -
Voraussetzungen
prüfen

Schritt 10 - Überblick

Bevor der Nutzer mit einem Klick auf den Button „Fertig stellen“ die Installation startet wird ein Überblick über alle gewählten Einstellungen gegeben. Erstmalig wird hier auch die Möglichkeit geboten eine Antwortdatei mit allen getätigten Einstellungen erstellen zu lassen. Diese kann an späterer Stelle für eine automatisierte Installation genutzt werden.

Abb. 15.12:
Schritt 10 -
Überblick



Schritt 11 - Produkt installieren

Hier erfolgt nun die eigentliche Installation.

Unter Linux/Unix ist es notwendig zwei Aufgaben als Benutzer „root“ durchzuführen, da diese lokale Administratorrechte erfordern. Beide Aufgaben liegen in Form von Shell-Skripten vor. Um diese Skripte ausführen zu können, muss ein neues Terminalfenster geöffnet und die Identität des Nutzers root angenommen werden.

Nach dem Ausführen beider Skripte kann das neue Terminalfenster wieder geschlossen und das Fenster „Konfigurationsskripte ausführen“ mit einem Klick auf „OK“ geschlossen werden.

Abb. 15.13:
Schritt 11 -
Produkt
installieren

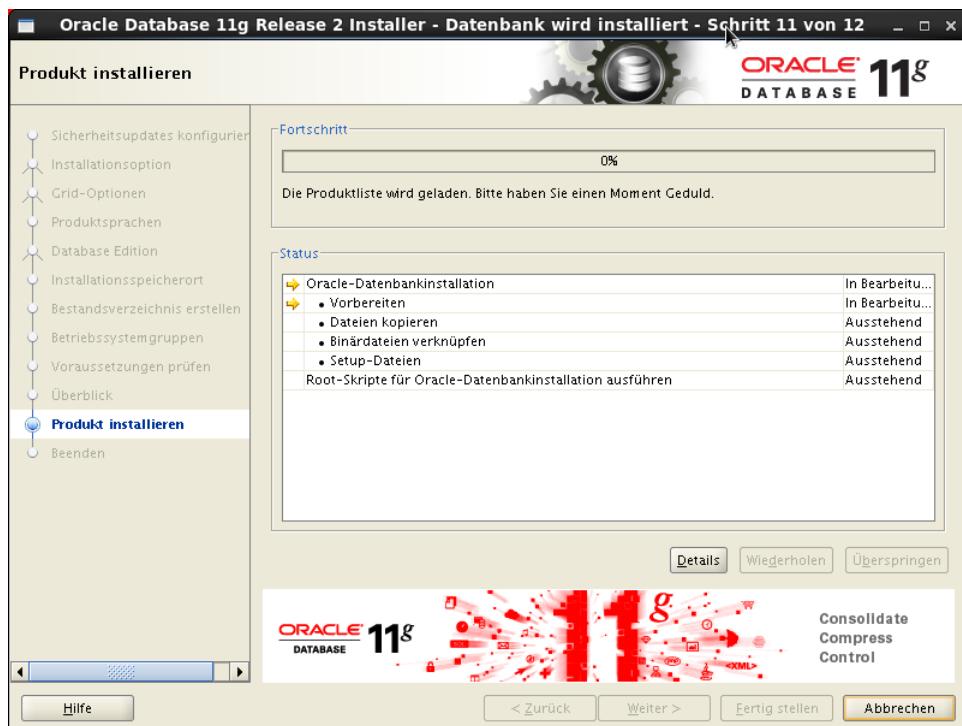
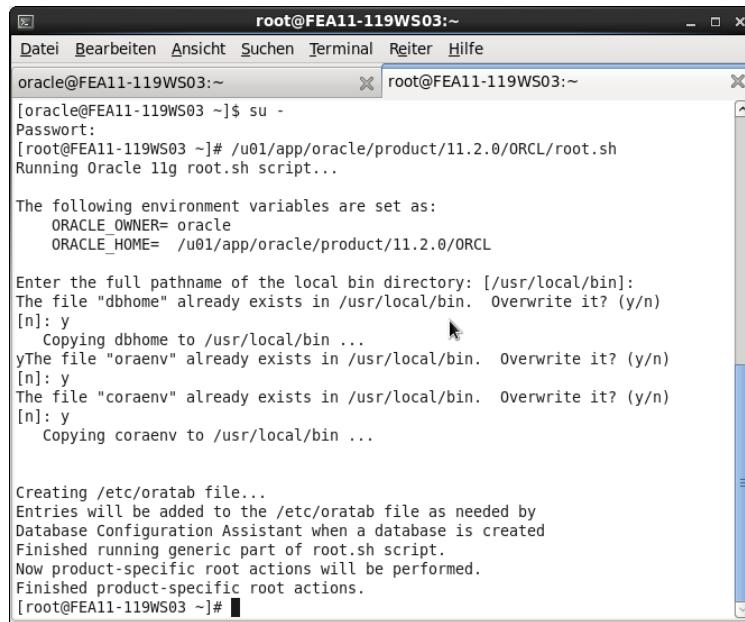


Abb. 15.14:
Schritt 11 - Das
Skript
„oraInstRoot.sh“

```
root@FEA11-119WS03:~ Datei Bearbeiten Ansicht Suchen Terminal Reiter Hilfe
oracle@FEA11-119WS03:~ X root@FEA11-119WS03:~ X
[oracle@FEA11-119WS03 ~]$ su -
Passwort:
[root@FEA11-119WS03 ~]# /u01/app/oraInventory/orainstRoot.sh
Berechtigungen ändern von/u01/app/oraInventory.
Lese- und Schreibberechtigungen für Gruppe werden hinzugefügt.
Lese-, Schreib- und Ausführungsberechtigungen für World werden entfernt.

Ändern des Gruppennamen von /u01/app/oraInventory zu oinstall.
Die Ausführung des Skripts ist abgeschlossen.
[root@FEA11-119WS03 ~]#
```

Abb. 15.15:
Schritt 11 - Das Skript „root.sh“



```

root@FEA11-119WS03:~$ su -
Passwort:
[root@FEA11-119WS03 ~]# /u01/app/oracle/product/11.2.0/ORCL/root.sh
Running Oracle 11g root.sh script...

The following environment variables are set as:
ORACLE_OWNER= oracle
ORACLE_HOME= /u01/app/oracle/product/11.2.0/ORCL

Enter the full pathname of the local bin directory: [/usr/local/bin]:
The file "dbhome" already exists in /usr/local/bin. Overwrite it? (y/n)
[n]: y
      Copying dbhome to /usr/local/bin ...
yThe file "oraenv" already exists in /usr/local/bin. Overwrite it? (y/n)
[n]: y
The file "coraenv" already exists in /usr/local/bin. Overwrite it? (y/n)
[n]: y
      Copying coraenv to /usr/local/bin ...

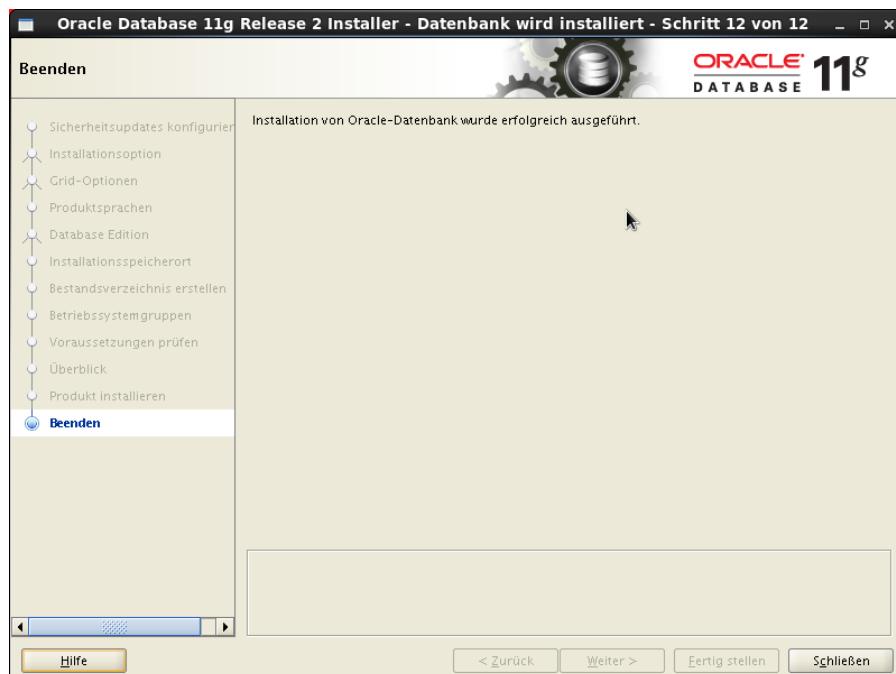
Creating /etc/oratab file...
Entries will be added to the /etc/oratab file as needed by
Database Configuration Assistant when a database is created
Finished running generic part of root.sh script.
Now product-specific root actions will be performed.
Finished product-specific root actions.
[root@FEA11-119WS03 ~]#

```

Schritt 12 - Beenden

Zu guter Letzt erhält man noch die Bestätigung, dass die Installation erfolgreich verlaufen ist. Mit einem Klick auf „Schließen“ kann der Oracle Universal Installer nun geschlossen werden.

Abb. 15.16:
Schritt 12 - Beenden



15.4 Software installieren/deinstallieren

Während der Installation der Oracle-Software wird auch eine Kopie des Oracle Universal Installers installiert. Diese befindet sich im Oraclesoftwareverzeichnis, Unterordner `oui/bin`. Mit Hilfe dieses OUI kann weitere Software nachinstalliert bzw. können Komponenten deinstalliert werden.

Unter Microsoft Windows kann der OUI aus dem Startmenü geöffnet werden. Für Linux-Systeme muss die Kommandozeile genutzt werden.

```
/u01/app/oracle/product/11.2.0/ORCL/oui/bin/runInstaller
```

Nach dem Start erfolgt die übliche Begrüßung.



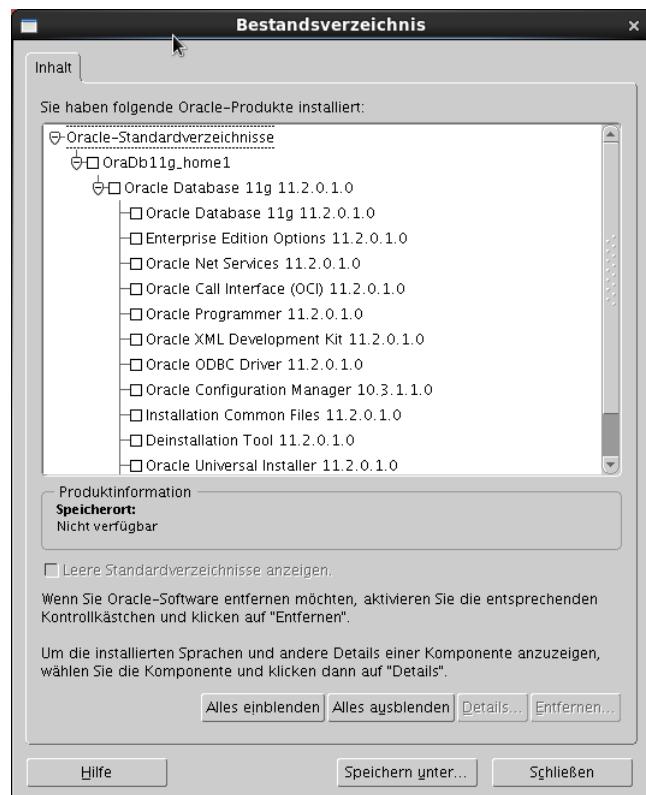
Abb. 15.17:
Willkommen im
OUI

Hier erfolgt nun die Auswahl, was als nächstes geschehen soll. Mit einem Klick auf den Button „Installierte Produkte“ kann das Oracle Bestandsverzeichnis abgefragt werden.

Durch anhaken/markieren einzelner Komponenten und anklicken der Schaltfläche „Entfernen“ können Teile der Software deinstalliert werden.

Zurück im Willkommensfenster kann mit einem Klick auf die Schaltfläche „Weiter“, die Installation einer weiteren Oracle-Instanz begonnen werden. Die Installationsschritte bleiben dabei die gleichen, wie sie soeben beschrieben wurden.

Abb. 15.18:
Auflistung aller
installierten
Produkte



16 Erstellen einer Oracle Datenbank

Inhaltsangabe

16.1 Überlegungen vor der Erstellung

Eine Oracle Datenbank kann auf zwei verschiedene Arten erstellt werden:

- **Database Configuration Assistant (DBCA):** Der DBCA kann während der Installation durch den OUI oder später von Hand gestartet werden. Er stellt ein grafisches Interface für die Erstellung von Datenbanken zur Verfügung.
- **CREATE DATABASE:** Das SQL-Kommando `CREATE DATABASE` bietet die Möglichkeit zur manuellen Erstellung einer Datenbank. Wird eine Datenbank auf diesem Weg erstellt, anstatt mit dem DBCA, müssen zusätzliche Schritte ausgeführt werden, bis die Datenbank voll funktionsfähig ist.

Bevor aber eine dieser beiden Möglichkeiten genutzt werden kann, muss zuerst einiges an Vorarbeit geleistet werden. Die folgenden Überlegungen sollen dabei helfen, eine Oracle Datenbank planvoll zu erstellen.

16.1.1 Kapazitätenplanung

Für jedes Datenbankobjekt, wie z. B. eine Tabelle, kann anhand der verwendeten Datentypen, der Spaltenanzahl und der geschätzten maximalen Anzahl Zeilen eine durchschnittliche Größe berechnet werden. Um den Speicherbedarf einer Datenbank realistisch einschätzen zu können sollte dies für jedes Objekt geschehen. Folgendes Beispiel verdeutlicht die Kapazitätsplanung einer einzelnen Tabelle.

Tabelle: `tblKunden`

Spalten:

1.)	KundenID	NUMBER(4)	3 Byte
2.)	Name	VARCHAR2(30)	30 Byte
3.)	Vorname	VARCHAR2(25)	25 Byte
4.)	Strasse	VARCHAR2(50)	50 Byte
5.)	PLZ	VARCHAR2(5)	5 Byte
6.)	Ort	VARCHAR2(30)	30 Byte

Voraussichtliche Anzahl Zeilen : 300

Die Gesamtgröße der Tabelle `TBLKUNDEN` wird somit wie folgt berechnet:

$$300 * (3 + 30 + 25 + 50 + 5 + 30) = 42.900 \text{ Byte}$$

16.1.2 Planen des physischen Datenbanklayouts

Unter dem physischen Layout einer Oracle Datenbank versteht man die Aufteilung der selben in einzelne Dateien und die Verteilung dieser Dateien auf verschiedene Datenträger. Der Einsatz von Mirroring- und Striping-Mechanismen wird ebenfalls zum physischen Datenbanklayout gezählt. Eine vernünftige Planung kann sich entscheidend auf Performance und Ausfallsicherheit der Datenbank auswirken.

Oracle Managed Files (OMF) und Automatic Storage Management (ASM)

OMF und ASM sind zwei Mechanismen, die den Administrator bei der Verwaltung einer Datenbank unterstützen sollen. Beide sind aber nicht zwingend notwendig für den Betrieb.

Auswählen des globalen Datenbanknamens

Der globale Datenbankname ist der Name der Datenbank, der sie eindeutig identifiziert, z. B. „ORCL“.

Initialisierungsparameter

Die Initialisierungsparameter beeinflussen das Verhalten einer Oracle-Instanz. Eine ungünstige Konfiguration kann sich negativ auf die Performance der Datenbank auswirken.

Der Datenbankzeichensatz

Je nach dem welcher Zeichensatz für die Datenbank ausgewählt wurde kann diese verschiedene Sprachen darstellen und andere nicht. Dieser Punkt gewinnt im multinationalen Betrieb eine erhöhte Bedeutung. Für die Auswahl des richtigen Datenbankzeichensatzes sind die folgenden Punkte entscheidend:

- Welche Sprachen sollen in der Datenbank dargestellt werden (jetzt und in Zukunft)?
- Ist der Zeichensatz im verwendeten Betriebssystem verfügbar?
- Welchen Zeichensatz nutzen die Clients?



- [NLSPG002]

Auswählen der korrekten Zeitzone(n)

Die gewählte Zeitzone ist dann entscheidend, wenn Clients in unterschiedlichen Zeitzonen stehen. Sie ist dafür zuständig, eine korrekte Umrechnung der Datum/Uhrzeit-Angaben des Clients in die Zeitzone des Servers sicherzustellen.



- [i1006705]

Festlegen der Oracle Datenblockgröße

Die Standard Oracle Blockgröße beträgt 8 Kb. Abhängig vom verwendeten Dateisystem und dem Verwendungszweck der Datenbank muss dieser Wert angepasst werden, um die Datenbank performant zu machen.

Abschätzen der passenden Größe für den Sysaux-Tablespace

Der SYSAUX-Tablespace wird automatisch bei der Datenbankerstellung generiert. Er beinhaltet Nutzdaten für verschiedene Oracle Features wie z. B. Oracle Text, Ultra Search, Log Miner, Oracle Spatial und andere. Erstellt wird dieser Tablespace abhängig von den gewählten Oracle Features, in der richtigen Größe. Der Administrator muss sich darum kümmern, dass dieser Tablespace auf einem Datenträger mit genügend freiem Speicher erstellt wird.

Einplanen von Standardtablespaces für Nutzer

Wenn Nutzer neue Datenbankobjekte anlegen, werden diese im „Defaulttablespace“ des jeweiligen Nutzers abgelegt. Wurde einem Nutzer kein Defaulttablespace zugewiesen oder existiert kein Tablespace, außer dem SYSAUX-Tablespace, wird automatisch dieser verwendet. SYSTEM stellt jedoch das Herzstück einer Oracle Datenbank dar und sollte deshalb niemals als Defaulttablespace dienen.

16.2 Der Database Configuration Assistant (DBCA)

Der DBCA stellt ein grafisches Werkzeug dar, mit dessen Hilfe folgende Tätigkeiten durchgeführt werden können:

- Eine Datenbank erstellen
- Konfigurieren einer Datenbank
- Datenbanken löschen
- Datenbanktemplates erstellen und verwalten

Gestartet wird der DBCA unter Windows mit Hilfe des Startmenüs. Unter Linux muss, in einem Terminalfenster, der gesamte Pfad zur Datei dbca, also \$ORACLE_HOME/bin/dbca angegeben werden. Dies kann jedoch nur dann funktionieren, wenn bereits die Umgebungsvariable \$ORACLE_HOME gesetzt wurde. Andernfalls muss der Pfad manuell angegeben werden: /u01/app/oracle/product/11.2.0/<SID>/bin/dbca (<SID> stellt hier einen Platzhalter für die SID der Datenbank dar.). Wie gewohnt, wird der Nutzer zu aller erst begrüßt.



Abb. 16.1:
Willkommen im
DBCA

Nach einem Klick auf den Button „Weiter“ muss zuerst die gewünschte Tätigkeit ausgewählt werden.

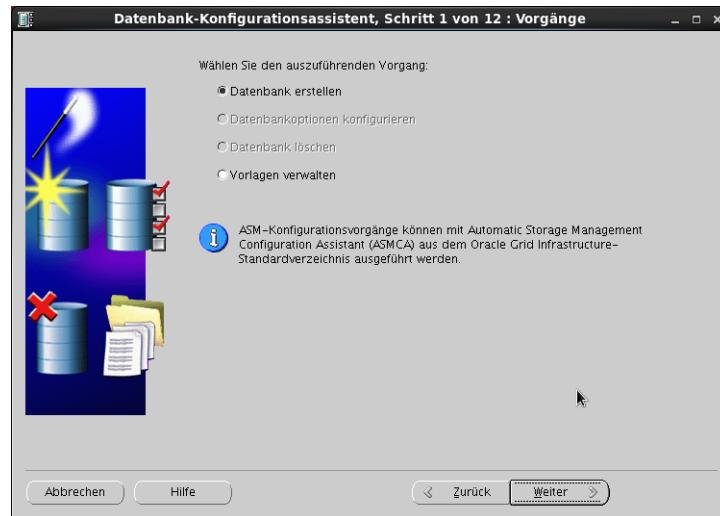


Abb. 16.2:
Schritt 1 -
Vorgänge

16.2.1 Eine Datenbank mit dem DBCA erstellen

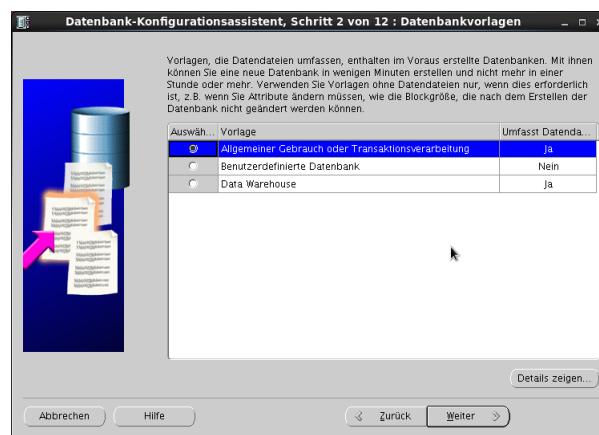
Für die Erstellung einer Datenbank bietet der DBCA zwei Varianten:

- Erstellen einer benutzerdefinierten Datenbank
- Erstellen einer Datenbank mittels einer Vorlage

Die erste Option erlaubt es, eine Datenbank komplett selbst zu gestalten, jedoch mit dem Nachteil, dass hier sehr viel Arbeit zu leisten ist. Die Erstellung einer Datenbank aus einer Vorlage hingegen bietet nahezu genauso viele Möglichkeiten, entlastet aber den Ersteller deutlich.

Schritt 2 - Datenbankvorlagen

Abb. 16.3:
Schritt 2 - Daten-
bankvorlagen



Standardmäßig werden zwei Vorlagen durch den DBCA angeboten:

- Allgemeiner Gebrauch oder Transaktionsverarbeitung
- Data Warehouse

Diese beiden unterscheiden sich hauptsächlich in den gespeicherten Initialisierungsparametern, mit denen die fertige Datenbank gestartet wird. Welche der beiden Vorlagen genutzt wird hängt davon ab, welchen Einsatzzweck die Datenbank haben soll.

Ein Datawarehouse, auch als Decision Support System bezeichnet, ist eine Datenbank, die oft eine sehr große Datenmenge hält und durch aufwendige SQL-Anfragen analysiert wird. Das heißt, dass für eine solche Datenbank eine Optimierung der Lesevorgänge im Vordergrund steht. Schreibvorgänge haben hier lediglich eine untergeordnete Rolle, da sie selten erfolgen und meist große Datenmengen am Stück, in die Datenbank importiert werden.

In einer OLTP - Online Transaction Processing - Datenbank ist das Verhältnis zwischen Lese- und Schreibzugriffen ganz anders. Daraus folgt, dass die Datenbank für beide Vorgänge so optimal wie möglich eingestellt werden sollte (Kompromisslösung).

Ein Klick auf den Button „Details zeigen...“ zeigt die Einstellungen der Vorlagen.

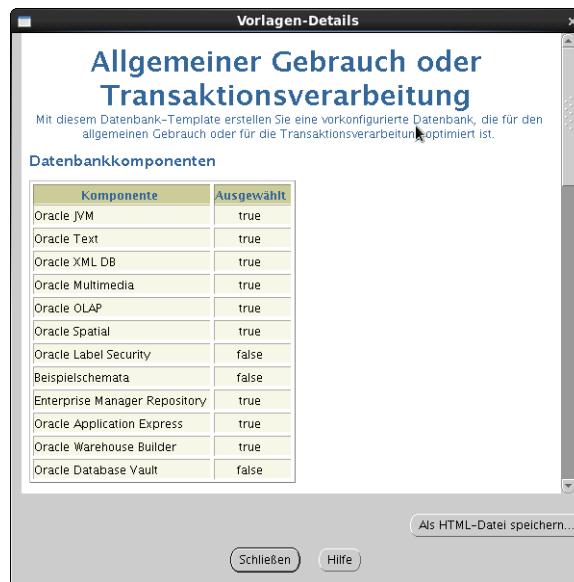


Abb. 16.4:
Schritt 2 -
Vorlagen-Details

Schritt 3 - Datenbank-ID

Nach einem Klick auf „Weiter“ muss in Schritt 3 der Datenbank ein Name gegeben werden. Zu diesem Thema gibt es jedoch einiges zu sagen.

Eine Oracle-Datenbank hat, ähnlich wie ein Mensch, einen Vornamen und einen Nachnamen. Korrekt ausgedrückt hat eine Oracle-Datenbank einen Datenbanknamen und eine Datenbankdomäne. Der Datenbankname kann maximal acht Zeichen umfassen und sollte dafür genutzt werden, um den Einsatzzweck der Datenbank wieder zu spiegeln. Datenbanknamen wie ORCL oder ORA11G sind maximal in Trainingsumgebungen sinnvoll. In einer Produktivumgebung sollten Namen wie DSS für ein Decision Support System oder PERS für eine Personalverwaltung genutzt werden. Evtl. ist es auch nützlich, dem Datenbanknamen eine Versionsnummer anzuhängen, um im Falle einer Migration die alte und die neue Datenbank am Namen unterscheiden zu können.

Die Datenbankdomäne ist der zweite Namensbestandteil. Sie kann dazu genutzt werden, um dem Datenbanknamen zusätzliche Informationen hinzuzufügen. Wenn beispielsweise zu einer produktiven Datenbank ein Testsystem hinzugefügt werden soll, könnten die Domänen PROD.ORACLE.LOCAL und TEST.ORACLE.LOCAL dazu genutzt werden, um beide Systeme zu unterscheiden. Fügt man beide Informationen zusammen, den Datenbanknamen und die Datenbankdomäne, so erhält man den „Globalen Datenbanknamen“.



Der Globale Datenbankname besteht aus dem Datenbanknamen und der Datenbankdomäne. Er wird im Format db_name . db_domain angegeben.

In der Welt der Oracle-Datenbanken hat jedoch nicht nur die Datenbank einen Namen, sondern auch die zu ihr gehörende Instanz. Der Instanzname wird im Betriebssystem, in einer Umgebungsvariablen namens ORACLE_SID hinterlegt. Die Abkürzung SID (gesprochen S-ID) steht für System Identifier. Welche Länge die SID haben darf ist Betriebssystemspezifisch, jedoch sind auf nahezu allen Plattformen acht Zeichen ohne Probleme möglich. Da die Instanz keine Domäne als Ergänzung hat, kann hier eine Unterscheidung, z. B. zwischen Test- und Produktivsystem nur im Instanznamen selbst erfolgen, beispielsweise „ppers“ oder „tpers“.

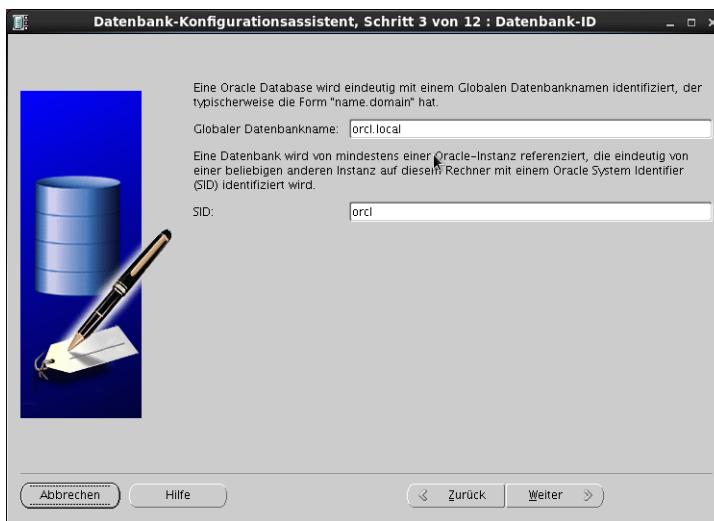


Der Instanznamen muss nicht gleich dem Datenbanknamen sein. Es empfiehlt sich jedoch beide Namen gleich zu wählen, um Datenbank und Instanz zusammen finden zu können.

Schritt 4 - Verwaltungsoptionen

Dieser Schritt gliedert sich in zwei Registerkarten. Auf dem Register „Enterprise Manager“ kann gewählt werden, wie die Datenbank verwaltet werden soll.

Abb. 16.5:
Schritt 3 -
Datenbank-ID



- **Enterprise Manager Grid Control:** Grid Control ist eine zentrale Verwaltungskonsole für verschiedene Oracle-Produkte, wie z. B. die Datenbank oder den Application Server. Sie kann für die Verwaltung beliebig vieler Systeme genutzt werden.
- **Enterprise Manager Database Control:** Hierbei handelt es sich um eine lokale Verwaltungskonsole für nur eine Datenbank.
- **Kein Enterprise Manager:** Es ist nicht zwingend notwendig für die Verwaltung einer Oracle-Datenbank den Enterprise Manager einzusetzen. Soll lediglich SQL*Plus genutzt werden oder sind Produkte von Drittanbietern im Einsatz, kann der Enterprise Manager einfach weggelassen werden.

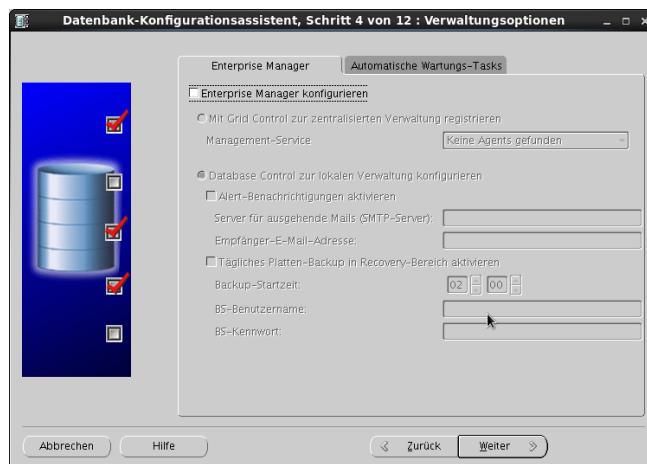


Abb. 16.6:
Schritt 4 -
Verwaltungs-
optionen

Die Option „Enterprise Manager konfigurieren“ kann seit Oracle 11g erst genutzt werden, wenn ein Listener (Siehe „Konfigurieren der Oracle Netzwerkumgebung“) konfiguriert und eine Datenbank erstellt wurde.



Die Registerkarte „Automatische Wartungs-Tasks“ bietet dem Administrator die Möglichkeit, automatische Wartungs-Tasks der Datenbank zu aktivieren. Dabei handelt es sich um Jobs, wie das Sammeln von Performance-Statistiken oder das Generieren von Berichten.



Abb. 16.7:
Schritt 4 -
Verwaltungsoptionen

Schritt 5 - Datenbank-ID-Daten

In Schritt 5 müssen Passwörter für die beiden Datenbankbenutzer SYS und SYSTEM festgelegt werden. Der Nutzer SYS ist ein Account mit uneingeschränkten Rechten. Er ist von zentraler Bedeutung in einer Oracle-Datenbank. Diese Tatsache sollte sich in der Komplexität seines Passwortes reflektieren.

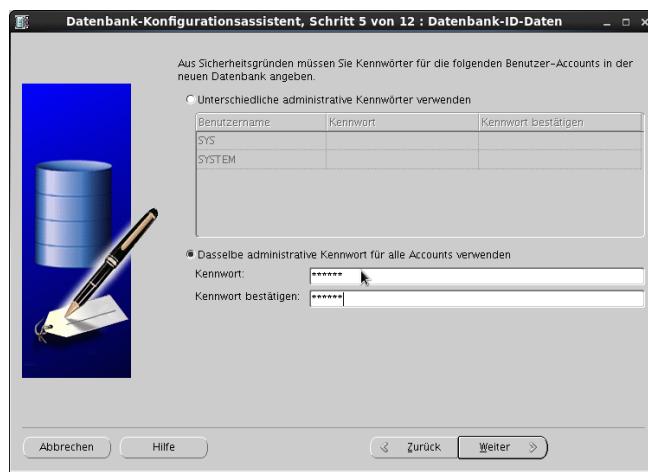
Das Benutzerkonto SYSTEM hat ebenfalls administrative Berechtigungen, ist jedoch stärker eingeschränkt, als SYS. Für diesen Nutzer sollte ebenso ein sicheres Passwort gewählt werden, da er das Recht hat, den gesamten Datenbankinhalt zu exportieren.

Schritt 6 - Speicherort von Datenbankdateien

Eine Oracle Datenbank besteht aus einer Vielzahl unterschiedlicher Dateien. Damit diese erstellt werden können, muss ein Speicherort vorgegeben werden. Der DBCA bietet dazu die beiden grundsätzlichen Möglichkeiten:

- Speichern der Dateien in einem Dateisystem
- Nutzung der Oracle-Eigenen Technologie „Automatic Storage Management“

Abb. 16.8:
Schritt 5 -
Datenbank-ID-
Daten



Egal welche der beiden Methoden gewählt wurde, es muss nun noch die Auswahl des genauen Speicherortes getroffen werden. Es kann der Speicherort aus der Vorlage genutzt, ein eigenes Verzeichnis für alle Dateien angegeben oder die Technologie „Oracle Managed Files“, die an späterer Stelle noch besprochen wird, genutzt werden.

Falls keine dieser drei Möglichkeiten die richtige ist, kann am Ende des Assistenten der Speicherort einer jeden einzelnen Datei geändert werden.

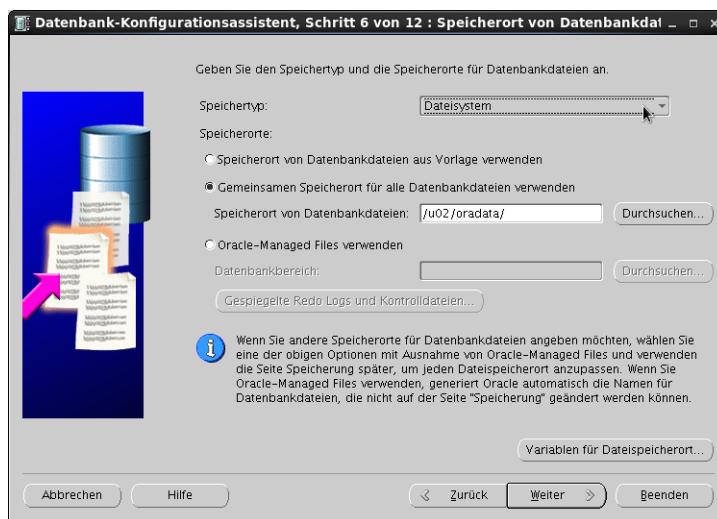


Abb. 16.9:
Schritt 6 -
Speicherort von
Datenbankdateien

Um herauszufinden, was hinter der Angabe {ORACLE_BASE}/oradata steckt, kann der Button „Variablen für Dateispeicherort“ angeklickt werden.

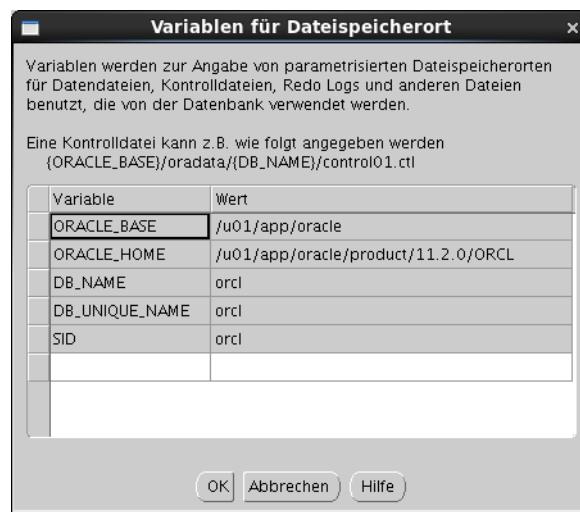


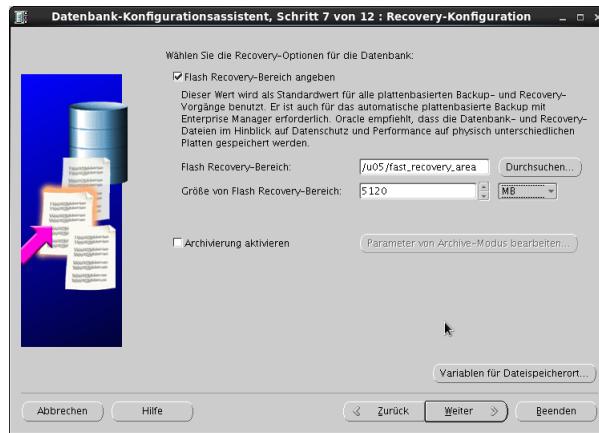
Abb. 16.10:
Schritt 6 -
Variablen für
Dateispeicherort

Schritt 7 - Recovery-Konfiguration

Dieser Dialog bietet zwei Optionen:

- **Flash Recovery-Bereich angeben:** Hierbei handelt es sich um ein von der Datenbank überwachtes Verzeichnis, welches als Puffer für Backups dient.
- **Archivierung aktivieren:** Diese Option aktiviert den Archiver-Hintergrundprozess, der dafür sorgt, dass automatisch Kopien der benutzten Redo Log Dateien angelegt werden.

Abb. 16.11:
Schritt 7 -
Recovery-
Konfiguration

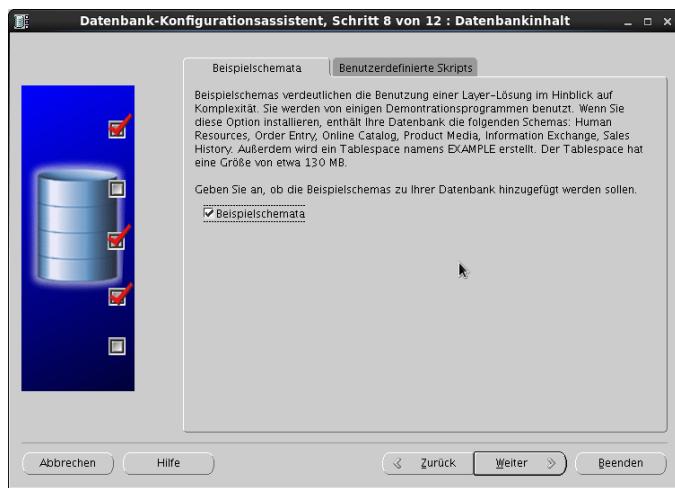


Schritt 8 - Datenbankinhalt

Hier ist es möglich, die Datenbank mit Beispielinhalten für Test- und Schulungszwecke zu füllen. Produktivsysteme sollten niemals die Beispielschemata enthalten, da diese bekannt sind und eine zusätzliche Angriffsfläche darstellen.

Auf der zweiten Registerkarte „Benutzerdefinierte Skripts“ können eigene SQL-Skripte angegeben werden, die der DBCA automatisch nach Erstellung der Datenbank ausführt.

Abb. 16.12:
Schritt 8 -
Datenbankinhalt

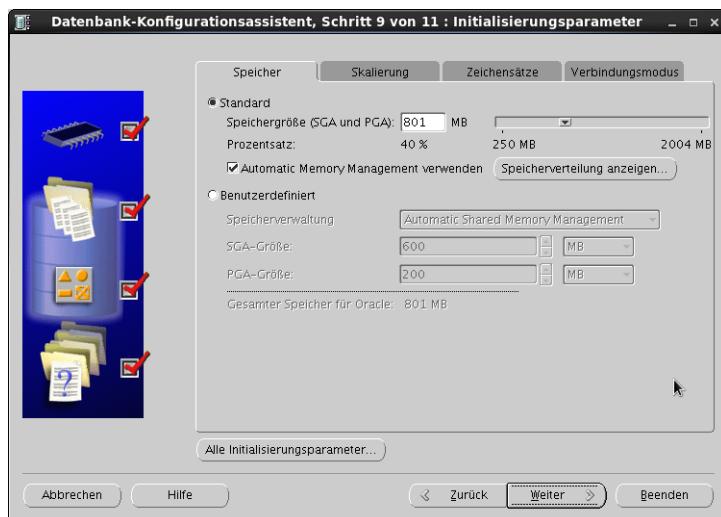


Schritt 9 - Initialisierungsparameter

Dies ist der wohl umfangreichste Dialog des DBCA. Auf der Registerkarte „Speicher“ muss die Entscheidung getroffen werden, wie der Arbeitsspeicher der Instanz verwaltet werden soll. Folgende Methoden stehen zur Verfügung:

- **Automatic Memory Management:** Bei dieser Option wird der Instanz eine Speichermenge zugeordnet, die diese dann vollständig autark verwaltet. Es wird keine Unterscheidung in SGA und PGA gemacht. Der Standard sind 40 % der gesamten Arbeitsspeichermenge des Servers.

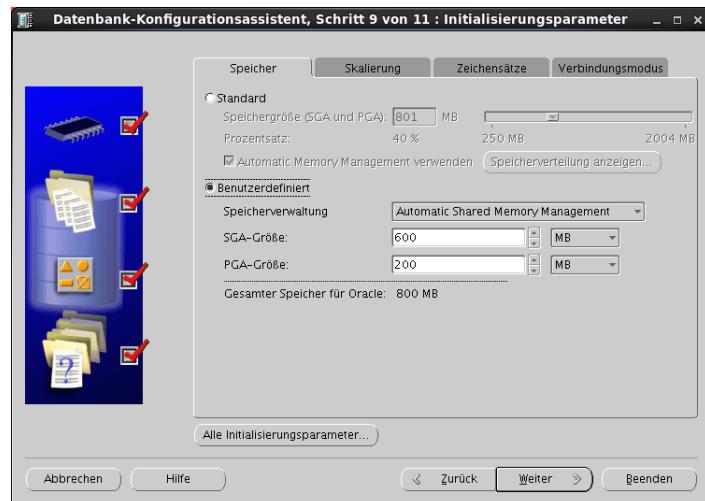
Abb. 16.13:
Schritt 9 -
Automatic
Memory
Management



- **Automatic Shared Memory Management:** Dies ist der Vorgänger zum Automatic Memory Management. Hier werden der Instanz zwei getrennte Werte für die Speichermenge der SGA und der aggregierten PGAs gegeben. Beide Speicherbereiche werden unabhängig von einander von der Instanz verwaltet.

Abb. 16.14:

Schritt 9 -
Automatic Shared
Memory
Management



- **Manual Shared Memory Management:** Mit dieser Variante muss für jeden Speicherpool in der SGA und für die aggregierten PGAs ein eigener Wert angegeben werden. Auch wenn der DBA hier die größten Einflussmöglichkeiten hat, so sollte doch eine der beiden automatischen Verwaltungsoptionen genutzt werden, da diese sich immer wieder den aktuellen Gegebenheiten anpassen.

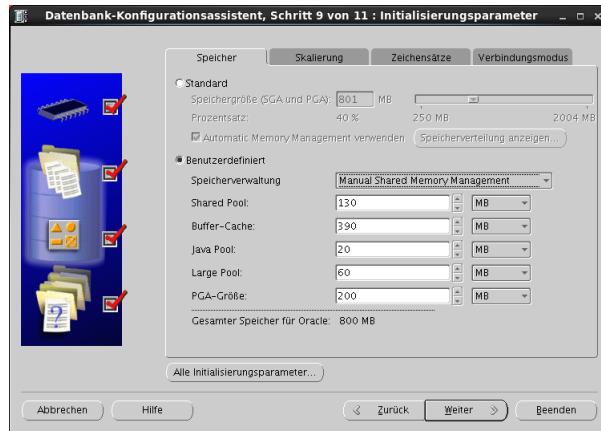


Abb. 16.15:
Schritt 9 - Manual
Shared Memory
Management

Nach der Einstellung der Speicherverwaltung kann nur auf der zweiten Registerkarte, mit dem Namen „Skalierung“, die Option „Prozesse“ geändert werden. Dieser Wert ist aus zwei Gründen interessant:

- lizenzrechtlich: Für jeden Client muss eine Client-Access-Licence vorliegen.
- Serverauslastung: Damit der Server nicht überlastet wird.

Der zweite Wert, die „Blockgröße“ kann nicht geändert werden, wenn die Datenbank aus einer Vorlage heraus erstellt wird. Nur bei einer benutzerdefinierten DB ist dies möglich.



Abb. 16.16:
Schritt 9 -
Skalierung

Auf der dritten Registerkarte können zwei Zeichensätze für die Datenbank ausgewählt werden, der Datenbankzeichensatz und der länderspezifische Zeichensatz.



Ein Zeichensatz besteht, wie sein Name besagt, aus einer Menge von Zeichen (Buchstaben, Ziffern, Sonderzeichen). Jedem Zeichen wird ein numerischer Code zugeordnet. Dieser wird vom Computer benötigt, um die Zeichen verarbeiten zu können.

Abb. 16.17:
Schritt 9 -
Zeichensätze



Der Datenbankzeichensatz wird für die folgenden Aufgaben verwendet:

- Speichern von Daten in den Datentypen CHAR, VARCHAR2, CLOB und LONG,
- Speichern von Objektbezeichnern, Spaltennamen und Variablenbezeichnern,
- Speichern von SQL und PL/SQL-Quellcode.

Bevor der Datenbankzeichensatz ausgewählt wird, sollten die folgenden Überlegungen angestellt werden:

- Welche Landessprachen muss die Datenbank jetzt und in Zukunft unterstützen?
- Ist der gewünschte Zeichensatz auch auf dem Betriebssystem des Datenbankservers verfügbar und welche Zeichensätze nutzen die Clients?
- Kommen die genutzten Anwendungen mit dem Zeichensatz zurecht?
- Gibt es Performance-Probleme oder andere Einschränkungen bei der Nutzung dieses Zeichensatzes?



Der Datenbankzeichensatz kann nach der Datenbankerstellung nur unter einer Bedingung geändert werden: Der neue Zeichensatz muss eine strikte Obermenge des Aktuellen sein. Oft ist dies nicht der Fall, weshalb der Datenbankzeichensatz, auf Empfehlung von Oracle, immer AL32UTF8 sein sollte, da dieser am umfassendsten ist.

Der länderspezifische Zeichensatz dient dazu, um Unicode-Zeichen in einer Datenbank zu speichern, die keinen Unicode-Datenbankzeichensatz nutzt. Nur die Datentypen NCHAR, NVARCHAR2 und NCLOB unterstützen diesen alternativen Zeichensatz. Auf der Registerkarte „Verbindungsmodus“ kann die Art und Weise gewählt werden, wie sich Clients standardmäßig mit der Datenbank verbinden sollen. Die beiden Modi „Dedizierter Server“ und „Shared Server“ werden später im Skript noch näher erläutert.

Schritt 10 - Datenbankspeicherung

In diesem vorletzten Schritt können die Speicherorte aller Datenbankdateien geändert werden. Des Weiteren ist es möglich, verschiedene Optionen für einzelne Dateiarten zu ändern.

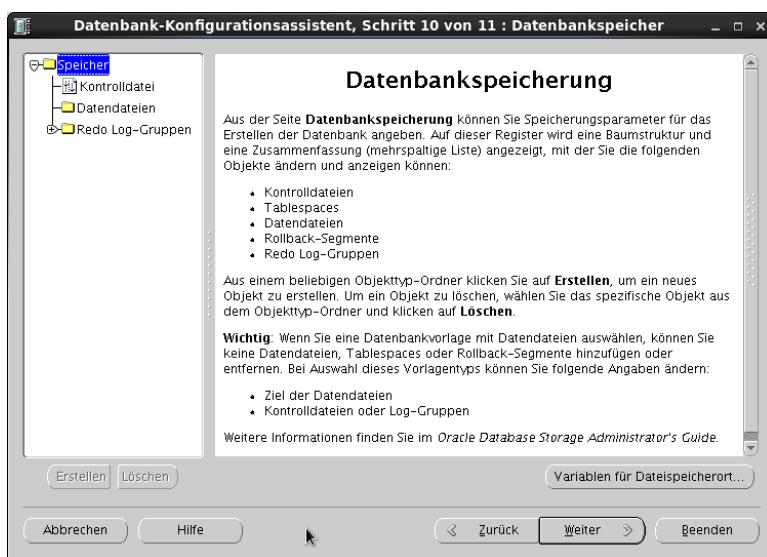


Abb. 16.18:
Schritt 10 - Daten-
bankspeicher

Schritt 11 - Optionen für das Erstellen

In Schritt 11 von 11 bleibt nur die Auswahl, was mit den soeben getätigten Einstellungen geschehen soll. Soll damit eine neue Datenbank erstellt, ein neues Template kreiert oder ein SQL-Skript generiert werden. Diese Optionen sind kombinierbar.

An dieser Stelle muss der Nutzer auf den „Beenden“-Button klicken, um eine Zusammenfassung des Erstellvorgangs angezeigt zu bekommen.

Abb. 16.19:
Schritt 11 -
Optionen für das
Erstellen

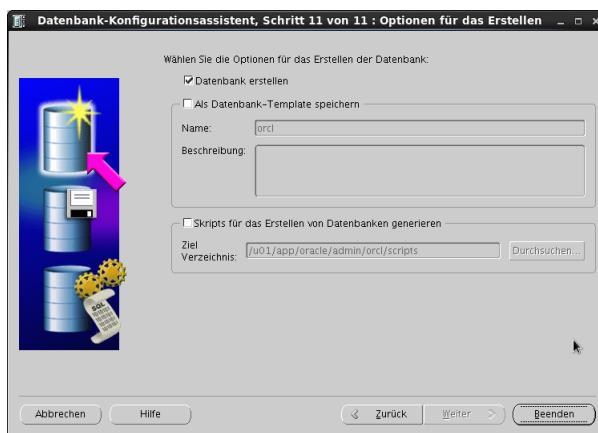
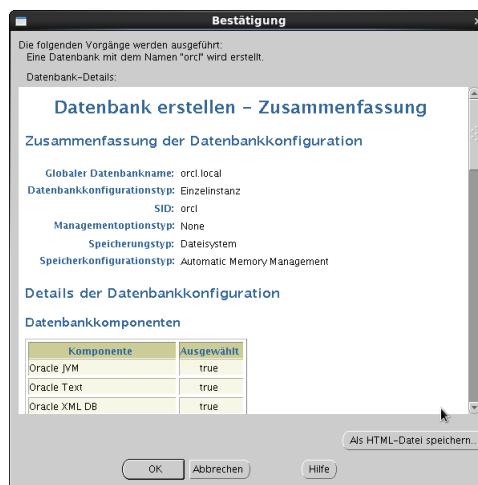


Abb. 16.20:
Schritt 12 -
Bestätigung



Nach einem Klick auf „OK“ bleibt nur noch abzuwarten, bis die Datenbank fertig ist.

Abb. 16.21:
Schritt 13 - Daten-
bankerstellung



16.2.2 Konfigurieren einer Datenbank mit dem DBCA

Der DBCA bietet die Möglichkeit, Konfigurationseinstellungen von Datenbankoptionen zu ändern. Folgende Optionen stehen zur Verfügung:

- Einbinden von installierten Komponenten in die Datenbank
- Ändern des Verbindungsmodus

Schritt 1 - Starten des DBCA

Nach dem Start des DBCA muss die Option „Datenbankoptionen konfigurieren“ ausgewählt werden.

Schritt 2 - Datenbank

Im zweiten Schritt wird eine Auswahl aller installierten Datenbanken geboten. Hier ist die Richtige auszuwählen.

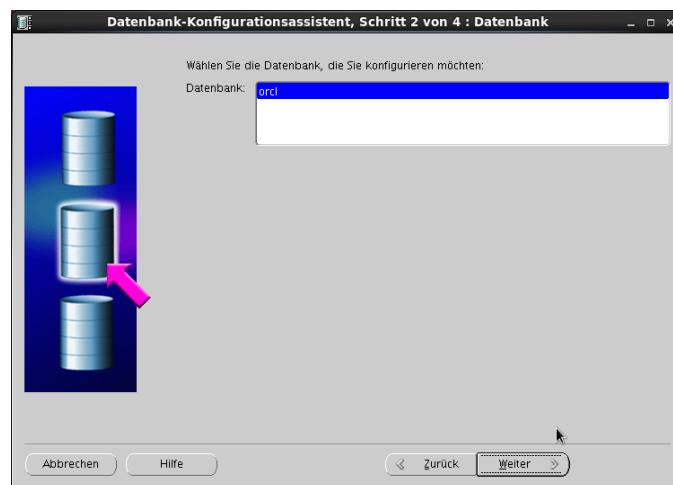


Abb. 16.22:
Schritt 2 -
Datenbank

Nachdem die Datenbank ausgewählt wurde, werden deren Einstellungen eingelesen.



Abb. 16.23:
Schritt 2 -
Datenbank

Schritt 3 - Verwaltungsoptionen

Wie unter Schritt 4 der Datenbankerstellung beschrieben kann hier der Enterprise Manager für die Datenbank konfiguriert werden.

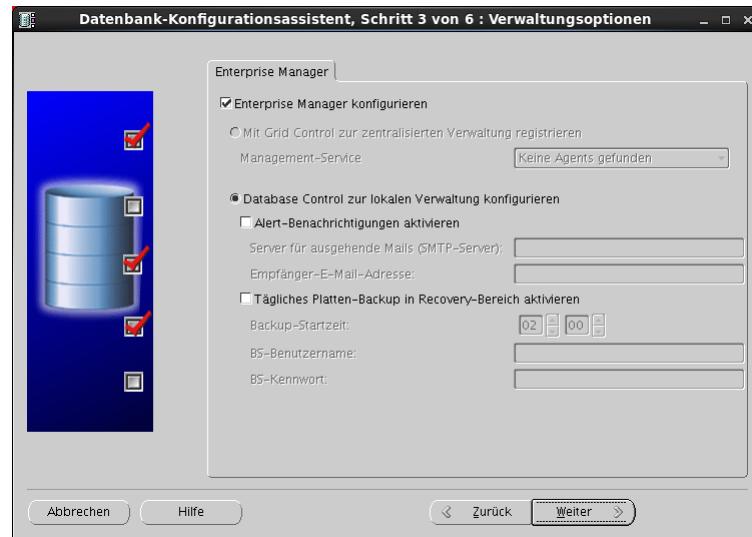


Abb. 16.24:
Schritt 3 -
Verwaltungsoptionen

Schritt 4 - Datenbankinhalt

Hier können verschiedene Datenbankoptionen hinzugefügt oder entfernt werden. Dies ist nur möglich, wenn es sich um eine benutzerdefinierte Datenbank handelt.

Abb. 16.25:
Schritt 4 -
Datenbankinhalt



Schritt 5 - Verbindungsmodus

Im letzten Schritt ist das Ändern des Verbindungsmodus möglich.

Abb. 16.26:
Schritt 5 - Verbin-
dungsmodus



17 Verwalten einer Oracle Instanz

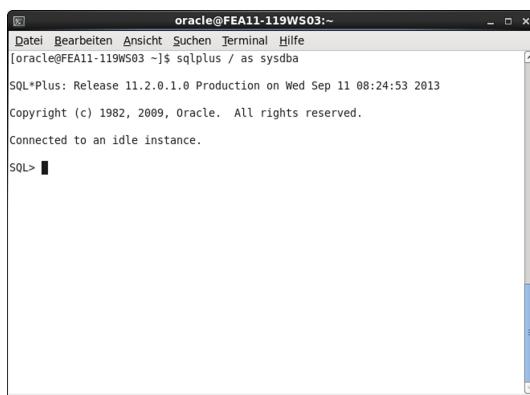
Inhaltsangabe

Eine Oracle-Umgebung besteht immer aus zwei Teilen: Datenbank und Instanz. Beim Start-Up, dem „Hochfahren der Datenbank“, wird die Instanz als Shared-Memory-Block im Arbeitsspeicher erzeugt. Dieser wird in die SGA und die PGA unterteilt. Des Weiteren werden verschiedene Hintergrundprozesse gestartet, die die Arbeit in der SGA verrichten.

17.1 Das SQL*Plus-Tool

Das SQL*Plus-Tool (Aussprache: sequel plus) ist ein textbasiertes, interaktives Tool, welches hauptsächlich für administrative Aufgaben gedacht ist. Es wird automatisch bei jedem Oracle-Datenbankserver mit installiert. Ebenso kann es mit Hilfe der Oracle Client Tools auf einem Rechner ohne Datenbankserver installiert werden.

Abb. 17.1:
Das
SQL*Plus-Tool

A screenshot of a terminal window titled "oracle@FEA11-119WS03:~". The window contains the following text:

```
oracle@FEA11-119WS03:~$ Datei Bearbeiten Ansicht Suchen Terminal Hilfe
[oracle@FEA11-119WS03 ~]$ sqlplus / as sysdba
SQL*Plus: Release 11.2.0.1.0 Production on Wed Sep 11 08:24:53 2013
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to an idle instance.

SQL> ■
```

The window has a standard Linux-style menu bar at the top. The main area shows the SQL*Plus command-line interface with the prompt "SQL>" followed by a blue square cursor.

Das SQL*Plus-Tool kennt drei verschiedene Arten von Befehlen:

- SQL-Kommandos
- PL/SQL-Kommandos
- SQL*Plus-Befehle

Während SQL- und PL/SQL-Befehle durch die Datenbank verarbeitet werden, bleiben SQL*Plus-Kommandos lokal. Sie dienen nur zur Formatierung der Anzeige in SQL*Plus.

17.1.1 Die erste Anmeldung

Das Anmelden an einer Datenbank mit Hilfe des SQL*Plus-Tools verläuft in drei Schritten:

1. Öffnen eines Terminalfensters

2. Auswählen der gewünschten Instanz

3. Starten von SQL*Plus

Um eine Instanz auswählen zu können, liefert Oracle das Shell-Skript oraenv aus.

```
[oracle@FEA11-119WS03 ~]$ . oraenv
ORACLE_SID = [orcl] ? orcl
```

Abb. 17.2:
Das oraenv
Shell-Skript



Beachten Sie den Punkt im Aufruf: . oraenv

Der Start des SQL*Plus-Tools erfolgt mittels der Kommandozeile: `sqlplus / as sysdba`. Die Bedeutung des Zusatzes `/ as sysdba` wird an späterer Stelle noch behandelt.

17.1.2 Die wichtigsten SQL*Plus-Befehle

Tabelle 17.1: Die wichtigsten SQL*Plus-Befehle

Befehl	Beispiel	Erläuterung
<code>show user</code>	<code>show user</code>	Zeigt den aktuellen Benutzernamen an.
<code>conn[ect]</code>	<code>conn hr/hr</code>	Öffnet eine Session mit dem angegebenen Nutzernamen/Passwort.
<code>disconn[ect]</code>	<code>disconn</code>	Beendet die aktuelle Session.
<code>exit</code>	<code>exit</code>	Beendet die aktuelle Session und schließt das SQL*Plus-Tool.
<code>ho[st]</code>	<code>ho</code>	Verlässt SQL*Plus und wechselt in einen Terminal. Das SQL*Plus-Tool bleibt im Hintergrund geöffnet. Durch die Eingabe von <code>exit</code> kann zurückgewechselt werden.
<code>desc[ribe]</code>	<code>desc employees</code>	Zeigt die Definition einer Tabelle an.
<code>ed[it]</code>	<code>ed</code>	Öffnet das letzte SQL-Statement in einem Editor. Das Statement kann geändert und erneut ausgeführt werden.
<code>col xx format aNN</code>	<code>col mail format a20</code>	Breite einer Tabellenspalte der Typen CHAR, VARCHAR2 oder DATE begrenzen. <code>xx</code> steht für einen Spaltenbezeichner und <code>NN</code> steht für die Breite.

Befehl	Beispiel	Erläuterung
col xx format NN	col salary 999999	Begrenzt die Breite einer Spalte des Typs NUMBER. Jede 9 steht für eine Stelle, d. h. 999 erzeugt eine dreistellige Spalte.
set linesize NN	set linesize 300	Zeilenlänge auf NN Zeichen begrenzen.
set long NN	set long 4000	Begrenzt die Breite einer Spalte des Typs LONG auf NN Zeichen.
set pagesize NN	set pagesize 50	Seitenhöhe auf NN Zeilen begrenzen. Die Höhe gibt an, nach wie vielen Zeilen die Spaltenüberschriften wiederholt werden.
set serveroutput on off	set serveroutput on	Für PL/SQL-Blöcke wird die Bildschirmausgabe ein- oder ausgeschaltet.
startup	startup	Startet eine Oracle-Instanz.
shutdown	shutdown	Schließt eine Oracle-Instanz.
[l]ist	l	Letztes SQL-Kommando anzeigen.
[r]erun	r	Letztes SQL-Kommando wiederholen.

17.2 Der Start-Up-Prozess

Unter dem Begriff Start-Up versteht Oracle das „Hochfahren der Datenbank“. Da die Datenbank jedoch nur eine Sammlung von Dateien ist, ist somit die Ausdrucksweise „Hochfahren der Datenbank“ inkorrekt. Richtiger Weise muss es heißen „Erstellen der Instanz“, da beim Start-Up eine Instanz erstellt und mit ihrer Datenbank verbunden wird.

Der Start-Up-Prozess verläuft in drei Schritten, die Start-Up-Phasen genannt werden. Jede Phase hat einen eigenen Namen, einen bestimmten Zweck und ist für unterschiedliche, meist administrative Tätigkeiten notwendig. Die Namen der Start-Up-Phasen lauten:

- NOMOUNT
- MOUNT
- OPEN

17.2.1 Die NOMOUNT-Phase

Nach dem ersten Schritt des Start-Ups befindet sich die Datenbank in der NOMOUNT-Phase. Das bedeutet, dass die Instanz erstellt, aber noch nicht an die Datenbank angeschlossen wurde. Es existiert eine Instanz ohne Datenbank.



Um den Start-Up einer Instanz durchführen zu können, müssen die beiden Umgebungsvariablen ORACLE_SID und ORACLE_HOME gesetzt sein. Dies geschieht beim Ausführen des Shell-Skriptes: . oraenv.

Das Starten der Instanz geschieht mit dem SQL*Plus-Befehl `startup`. Diesem Kommando können die Zusätze `nomount`, `mount` oder `open` mitgegeben werden, um die jeweils gewünschte Start-Up-Phase zu erreichen.

Listing 17.1: Einen Start-Up bis zur NOMOUNT-Phase durchführen

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Aug 27 10:20:29 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to an idle instance.

SQL> startup nomount
ORACLE instance started.

Total System Global Area  643084288 bytes
Fixed Size                  2215984 bytes
Variable Size                222302160 bytes
Database Buffers            411041792 bytes
Redo Buffers                 7524352 bytes
```

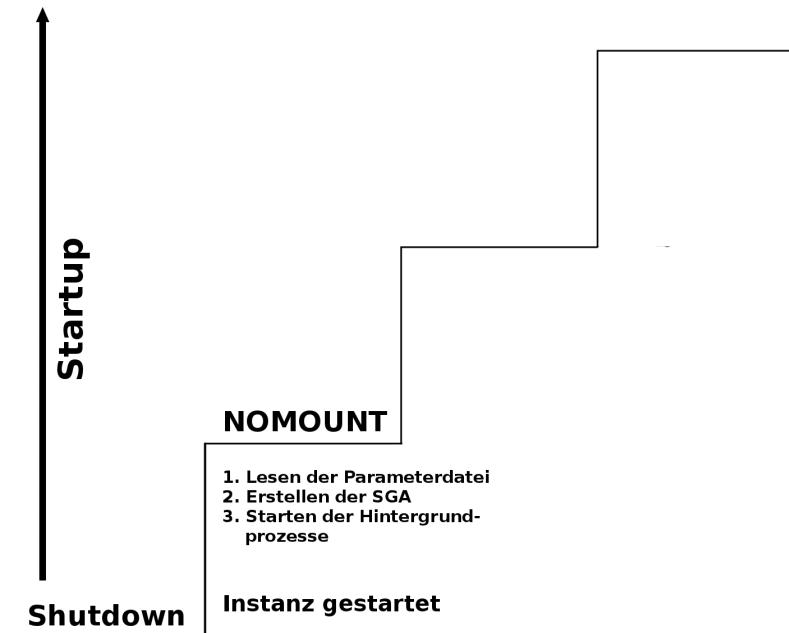


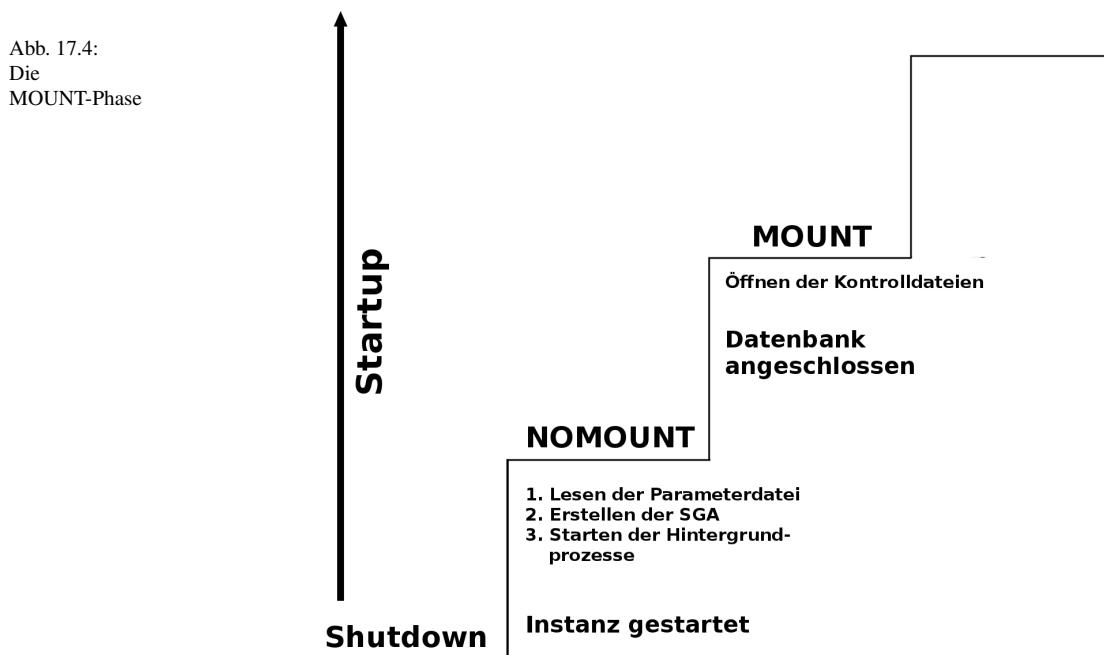
Abb. 17.3:
Die NOMOUNT-Phase

Nach Erreichen der NOMOUNT-Phase sind drei Dinge geschehen:

1. Die Parameter-/Serverparameterdatei wurde gelesen
2. Die SGA wurde mit den gelesenen Parametern erstellt
3. Die Oracle-Hintergrundprozesse wurden gestartet

17.2.2 Die MOUNT-Phase

Der Begriff „mounten“ bedeutet, dass eine Instanz mit ihrer Datenbank verbunden wird. Dieser Vorgang wird dadurch realisiert, dass die Instanz die Kontrolldatei der Datenbank liest, um ihr die Speicherorte der Daten- und der Redo Log Dateien zu entnehmen. Der Pfad zu den Kontrolldateien ist in der Parameterdatei, im Parameter `control_files` festgelegt.



In dieser Phase ist die Instanz exklusiv geöffnet, d. h. nur Administratoren haben Zugriff, normale Nutzer noch nicht. Die MOUNT-Phase ist für administrative Tätigkeiten, wie z. B. Recovery nach einem Datenverlust oder das Verschieben von Datenbankdateien vorgesehen.

Wie eine Instanz die MOUNT-Phase erreichen kann, hängt davon ab, ob sie geschlossen ist oder ob sie sich in der NOMOUNT-Phase befindet. Für eine geschlossene Datenbank wird mit Hilfe des Kommandos `startup mount` die Instanz bis in die MOUNT-Phase gebracht. Existiert die Instanz aber bereits in der NOMOUNT-Phase, muss sie mittels des Befehls `ALTER DATABASE MOUNT` in die MOUNT-Phase versetzt werden.



Eine Instanz die bereits mit `startup` gestartet wurde, kann nicht nochmal gestartet werden. Ihr Status muss stattdessen mit `ALTER DATABASE` geändert werden.

Beispiel ?? zeigt wie eine Instanz reagiert, wenn sie nach einen Start-Up erneut gestartet werden soll.

Listing 17.2: ORACLE läuft noch. Erst stoppen.

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Aug 27 10:23:58 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> startup mount
ORA-01081: cannot start already-running ORACLE - shut it down first
```

In Beispiel ?? ist zu sehen, dass das `startup nomount`-Kommando, die Instanz startet und in die NOMOUNT-Phase versetzt. Das zweite Kommando, `startup mount` quittiert Oracle mit der Fehlermeldung: „ORA-01081: cannot start already-running ORACLE - shut it down first“. Dies geschieht, da die Instanz bereits gestartet worden war.



Eine Instanz muss erst heruntergefahren werden, bevor sie erneut gestartet werden kann.

Im folgenden Beispiel wird die Instanz korrekt von der NOMOUNT-Phase in die MOUNT-Phase gehoben.

Listing 17.3: Das Kommando ALTER DATABASE MOUNT

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Aug 27 10:26:35 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> ALTER DATABASE MOUNT;

Database altered.
```

17.2.3 Die OPEN-Phase

Eine Oracle-Datenbank zu öffnen bedeutet, die Daten- und Redo Log Dateien zu öffnen und sie den „normalen Nutzern“ zugänglich zu machen. Das Öffnen einer geschlossenen Datenbank geschieht mit dem Kommando `startup` oder wahlweise auch mit `startup open`.

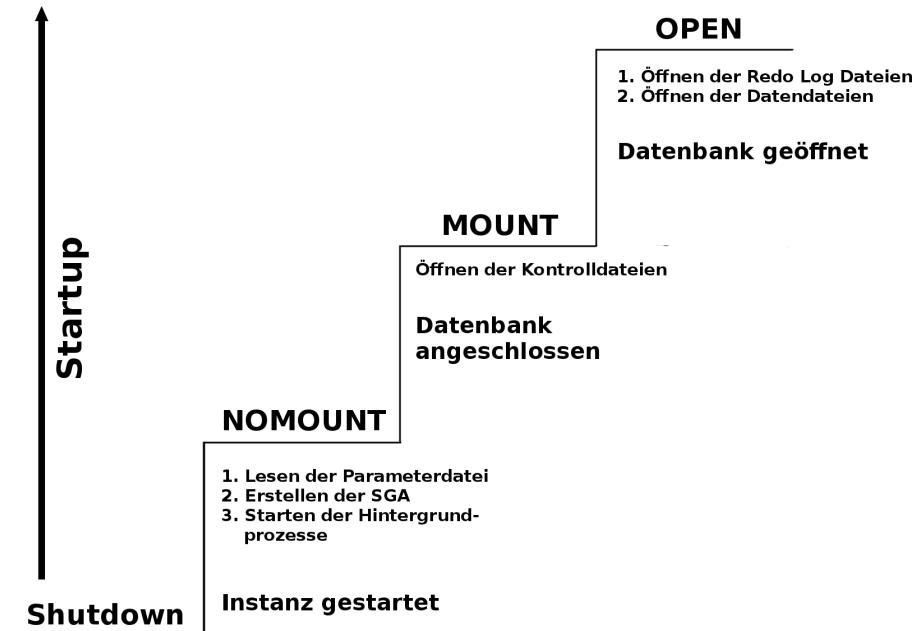


Abb. 17.5:
Die OPEN-Phase

Listing 17.4: Starten der Instanz und öffnen der Datenbank

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Aug 27 10:36:12 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to an idle instance.

SQL> startup open
ORACLE instance started.

Total System Global Area  643084288 bytes
Fixed Size                  2215984 bytes
Variable Size                222302160 bytes
Database Buffers            411041792 bytes
Redo Buffers                 7524352 bytes
Database mounted.
Database opened.
```

Für den Wechsel zwischen MOUNT- und OPEN-Phase gilt das Gleiche, wie für den Wechsel zwischen NOMOUNT- und MOUNT-Phase, er muss mittels `ALTER DATABASE` erfolgen, da die Instanz bereits gestartet ist.

Listing 17.5: Wechsel zwischen MOUNT- und OPEN-Phase

```
ORACLE instance started.

Total System Global Area  643084288 bytes
Fixed Size                  2215984 bytes
Variable Size                222302160 bytes
Database Buffers            411041792 bytes
Redo Buffers                 7524352 bytes
Database mounted.
SQL> ALTER DATABASE OPEN;
Database altered.
```

17.2.4 Das Hochfahren einer Instanz erzwingen

In einigen Fällen kommt es vor, dass eine Instanz nicht auf normalem Wege gestartet werden kann. Sollte dies geschehen, muss „der Instanzstart erzwungen werden“. Sinnvoll ist ein solches Vorgehen aber nur in den folgenden Situationen:

- Wenn die Instanz nicht gestartet werden kann
- Wenn ein Herunterfahren der Instanz nicht möglich ist

Mit dem Kommando `startup force` wird eine Instanz gezwungen, neu zu starten. Durch einen solchen „gewaltsamen“ Neustart entsteht jedoch Datenverlust, weil die Instanz ohne vorherigen Checkpoint von ihrer Datenbank getrennt wird. Alle nicht abgeschlossenen Transaktionen der Nutzer und sämtliche Sessions werden abgebrochen.

Zur Behebung des Datenverlusts, wird beim Neustart ein „Instance-“ bzw. „Crash recovery“ durchgeführt. Dies wird durch einen Eintrag im Alert-Log File belegt.

Listing 17.6: Ein erzwungener Neustart der Instanz und seine Folgen - Der Eintrag im Alert Log File

```
Mon Aug 26 08:59:27 2013 ALTER DATABASE OPEN
Beginning crash recovery of 1 threads
parallel recovery started with 2 processes
Started redo scan
Completed redo scan
read 119 KB redo, 82 data blocks need recovery
Started redo application at
  Thread 1: logseq 4, block 277
Recovery of Online Redo Log: Thread 1 Group 1 Seq 4 Reading mem 0
  Mem# 0: /u02/oradata/orcl/redo01.log
Completed redo application of 0.11MB
Completed crash recovery at
  Thread 1: logseq 4, block 515, scn 1037873
```

```
82 data blocks read, 82 data blocks written, 119 redo k-bytes read
```

17.3 Der Shutdown-Vorgang

Als „Shutdown“ wird der Vorgang des Herunterfahrens der Instanz bezeichnet. Welche Einzelschritte bei einem Shutdown geschehen, hängt davon ab, welche der vier Arten des Shutdowns gewählt wurde. Folgende Shutdown-Arten gibt es:

- Shutdown Normal
- Shutdown Transactional
- Shutdown Immediate
- Shutdown Abort

Ein Shutdown kann mit Hilfe eines Datenbank-Tools eingeleitet werden, z. B. SQL*Plus.

Nur Nutzer mit dem Privileg SYSDBA oder SYSOPER können einen Shutdown durchführen.



17.3.1 Shutdown NORMAL

Der Shutdown NORMAL ist die gründlichste Art des Shutdown. Er führt folgende Einzelschritte durch:

1. Es werden keine neuen Connections zur Datenbank zugelassen
2. Es wird gewartet, bis alle Transaktionen der Nutzer abgeschlossen sind
3. Es wird gewartet, bis alle Nutzer ihre Session beendet haben
4. Ein Checkpoint wird ausgelöst

Durch das Auslösen des Checkpoints werden alle geänderten Daten aus dem Database Buffer Cache in die Datendateien zurückgeschrieben. Daraus folgt, dass ein Shutdown NORMAL die Datenbank in einem konsistenten Zustand belässt.

Durchgeführt wird ein Shutdown NORMAL mit dem Kommando `shutdown normal` oder einfach nur `shutdown`.

Listing 17.7: Durchführen eines Shutdown NORMAL

```
SQL> shutdown normal
Database closed.
Database dismounted.
ORACLE instance shut down.
```



Problematisch an einem Shutdown NORMAL ist, dass in einer Umgebung mit sehr vielen Nutzern, vermutlich nie alle Nutzer ihre Session beenden, was bedeutet, dass der Shutdown-Vorgang auch nie vollendet werden kann.

17.3.2 Shutdown TRANSACTIONAL

Der Shutdown TRANSACTIONAL ist eine Abstufung des Shutdown NORMAL. Er führt folgende Einzelschritte durch:

1. Es werden keine neuen Connections zur Datenbank zugelassen
2. Es wird gewartet, bis alle Transaktionen der Nutzer abgeschlossen sind
3. Noch aktive Nutzersessions werden automatisch beendet
4. Ein Checkpoint wird ausgelöst

Hier wird also lediglich auf das Ende aller noch offenen Transaktionen gewartet, nicht aber darauf, dass sich alle Nutzer vom System abmelden. Dies kann einen Shutdown-Vorgang deutlich beschleunigen, bzw. es wird dadurch eine realistische Chance für die Vollendung des Shutdowns eingeräumt.



Bei einem Shutdown TRANSACTIONAL ist es sinnvoll die Nutzer vorher zubenachrichtigen, da diese ab einem Zeitpunkt X, ihre Arbeit nicht mehr fortsetzen können.

Listing 17.8: Durchführen eines Shutdown TRANSACTIONAL

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Aug 27 10:50:05 2013
```

```
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> shutdown transactional
Database closed.
Database dismounted.
ORACLE instance shut down.
```

Der Unterschied zwischen einem Shutdown NORMAL und einem Shutdown TRANSACTIONAL ist im Alert Log File zu sehen.

Listing 17.9: Der Shutdown TRANSACTIONAL im Alert Log

```
Shutting down instance (transactional)
Shutting down instance: further logons disabled
Stopping background process CJQ0
Stopping background process QMNC
Stopping background process MMNL
Stopping background process MMON
All transactions complete. Performing immediate shutdown
License high water mark = 3
All dispatchers and shared servers shutdown
ALTER DATABASE CLOSE NORMAL
```

Der Eintrag „All transactions complete. Performing immediate shutdown“ weisst darauf hin, dass nach dem Ende der letzten Transaktion sofort mit dem Shutdown begonnen wird. Bei einem Shutdown NORMAL fehlt diese Zeile, wie in Beispiel ?? zu sehen ist.

Listing 17.10: Der Shutdown NORMAL im Alert Log

```
Shutting down instance (normal)
Stopping background process SMCO
Shutting down instance: further logons disabled
Mon Aug 26 10:41:26 2013
Stopping background process CJQ0
Stopping background process QMNC
Stopping background process MMNL
Stopping background process MMON
License high water mark = 4
All dispatchers and shared servers shutdown
ALTER DATABASE CLOSE NORMAL
```

Ein Shutdown TRANSACTIONAL überführt die Datenbank in einen konsistenten Zustand.



17.3.3 Shutdown IMMEDIATE

Der Shutdown IMMEDIATE ist genau das, was sein Name besagt. Die Datenbank wird sofort, ohne auf offene Transaktionen oder Nutzer zu warten heruntergefahren. Da diese Art des Shutdowns sehr radikal ist, sollten auf jeden Fall alle Nutzer informiert werden.

Nur in den folgenden Situationen sollte ein Shutdown IMMEDIATE durchgeführt werden:

- Vor einem automatisierten Backup
- Um im Falle eines Stromausfalles die DB so schnell wie möglich herunterzufahren
- Im Falle einer Datenbankfehlfunktion, wenn es nicht möglich ist, alle Nutzer vorher zu benachrichtigen

Bei diesem Shutdown werden folgende Einzelschritte durchgeführt:

- Es werden keine neuen Connections zur Datenbank zugelassen
- Alle aktiven Transaktionen werden zurückgerollt
- Noch aktive Nutzersessions werden automatisch beendet
- Es wird ein Checkpoint gesetzt



Obwohl alle Transaktionen abgebrochen und alle Sessions geschlossen werden, wird die Datenbank in einem konsistenten Zustand hinterlassen.

Listing 17.11: Durchführen eines Shutdown IMMEDIATE

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Aug 27 10:52:09 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> shutdown immediate
Database closed.
```

```
Database dismounted.
ORACLE instance shut down.
```

17.3.4 Shutdown ABORT

Der Shutdown ABORT ist die einzige Shutdown-Variante, bei der die Datenbank in einem inkonsistenten Zustand hinterlassen wird. Hier wird die Instanz einfach von der Datenbank getrennt, ohne dass vorher ein Checkpoint ausgelöst wird. Aus diesem Grund sollte ein Shutdown ABORT nur dann eingesetzt werden, wenn es notwendig ist.

Folgende Einzelschritte werden bei einem Shutdown ABORT ausgeführt:

- Es werden keine neuen Connections zur Datenbank zugelassen
- Alle aktiven Transaktionen werden nicht zurückgerollt, sondern sofort abgebrochen (Inkonsistenz!)
- Noch aktive Nutzersessions werden abgebrochen

Listing 17.12: Durchführen eines Shutdown ABORT

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Tue Aug 27 10:52:09 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> shutdown abort
ORACLE instance shut down.
```



Ein Neustart der Datenbank nach einem `shutdown abort` erfordert ein Instance recovery.

- [i1006091]



17.4 Verwalten der Parameterdatei

Die Parameterdatei bzw. Serverparameterdatei ist die erste Datei, die benötigt wird, um eine Instanz hochfahren zu können. Sie enthält eine Liste mit Konfigurationsparametern, den sogenannten „Initialisierungsparametern“.

Oracle sucht seine Parameterdatei/Serverparameterdatei beim Hochfahren der Instanz in einem betriebssystemabhängigen Standardverzeichnis. Dieses ist:

- **Windows:** %ORACLE_HOME%\database
- **UNIX:** \$ORACLE_HOME/dbs

Beim Durchsuchen des entsprechenden Standardpfades wird nach drei verschiedenen Dateinamen gesucht:

- spfile\$ORACLE_SID.ora (z. B. spfileorcl.ora)
- spfile.ora
- init\$ORACLE_SID.ora (z. B. initorcl.ora)

Bei den ersten beiden Dateinamen handelt es sich um Serverparameterdateien, beim Dritten um eine Parameterdatei.



Kann Oracle keine dieser Dateien finden, wird der Startvorgang abgebrochen.



- [i1124822]

17.4.1 Initialisierungsparameter administrieren

Initialisierungsparameter haben unterschiedliche Aufgaben:

- Benennen von Objekten wie z. B. Kontrolldateien oder Betriebssystemverzeichnissen

- Beeinflussen von Kapazitäten, wie z. B. der Größe der SGA

Es kann aus unterschiedlichen Gründen notwendig sein, die Werte von Initialisierungsparametern zu verändern. Welche Auswirkungen diese Änderungen haben, hängt von den Charakteristiken der Datenbank und anderen Faktoren ab.

Statische und dynamische Parameter

Initialisierungsparameter werden in zwei Gruppen eingeteilt:

- **Statische Parameter:** Die Änderung an einem solchen Parameter wird erst nach einem Instanzneustart wirksam.
- **Dynamische Parameter:** Änderungen an dynamischen Parametern werden sofort wirksam.

Die View V\$SYSTEM_PARAMETER gibt Aufschluss darüber, ob ein Parameter statisch oder dynamisch ist.

Listing 17.13: Unterscheiden zwischen dynamischen und statischen Parametern

```
SQL> col name format a30
SQL> SELECT name, issys_modifiable
  2  FROM v$system_parameter;
```

In der Spalte ISSYS_MODIFIABLE können drei verschiedene Werte vorkommen:

- **IMMEDIATE:** Es handelt sich um einen **dynamischen** Parameter.
- **FALSE:** Der Parameter ist **statisch**.
- **DEFERRED:** Änderungen an einem so markierten Parameter haben nur auf neue Sessions, die nach der Änderung erstellt wurden eine Auswirkung.

Initialisierungsparameter ändern

Initialisierungsparameter können per **ALTER SYSTEM**-SQL-Kommando oder im Enterprise Manager geändert werden. Beispiel ?? zeigt wie der Parameter LICENSE_MAX_SESSIONS von 0 auf 50 geändert wird.



Der Parameter LICENSE_MAX_SESSIONS legt fest, wie viele gleichzeitige Verbindungen zur Datenbank möglich sind. Ist der Schwellenwert von LICENSE_MAX_SESSIONS erreicht, können sich nur noch Nutzer an der Datenbank anmelden, die das Privileg `restricted_session` haben.

Listing 17.14: license_max_sessions wird geändert

```
SQL> show parameter license_max_sessions

NAME                                     TYPE        VALUE
-----
license_max_sessions                      integer     0

SQL> ALTER SYSTEM
  2 SET license_max_sessions = 50 SCOPE=both;
System altered.
```

Das `ALTER SYSTEM`-Kommando hat in Beispiel ?? zwei Klauseln:

- `SET <parameter> = wert`: Gibt den zu verändernden Parameter und den neuen Wert an. Welche Werte zulässig sind, hängt vom jeweiligen Parameter ab.
- `SCOPE = <scope>`: Mit der `SCOPE`-Klausel wird geregelt, wo die Änderung vollzogen wird.

Die `SCOPE`-Klausel kennt drei Werte für `<scope>`:

- `SCOPE=both`: Die Änderung erfolgt in der SGA und im SPFILE.
- `SCOPE=memory`: Die Änderung erfolgt nur in der SGA.
- `SCOPE=spfile`: Die Änderung erfolgt nur im SPFILE. Damit der neue Wert wirksam wird, muss die Instanz neu gestartet werden.



`SCOPE=both` ist die Standardeinstellung und kann deshalb entfallen.



- [i2053602]

17.4.2 Sessionparameter

Im Gegensatz zu Initialisierungsparametern, die systemweite Gültigkeit haben, sind Sessionparameter nur innerhalb der Session eines Nutzers gültig. Beispielsweise kann der Sessionparameter `nls_language` von jedem Nutzer, der das `ALTER SESSION` besitzt, geändert werden. D. h. während Nutzer A mit deutschen Spracheinstellungen arbeitet, kann Nutzer B zur gleichen Zeit in englischer Sprache arbeiten.

Sessionparameter ändern

Änderungen an Sessionparametern werden mit dem SQL-Kommando `ALTER SESSION` durchgeführt, dessen Syntax dem `ALTER SYSTEM`-Kommando sehr ähnlich ist.

Geändert werden können einige dynamische Initialisierungsparameter, sowie alle Sessionparameter. Welche Initialisierungsparameter betroffen sind, kann wiederum mit Hilfe der View `V$SYSTEM_PARAMETER` und der Spalte `ISSES_MODIFIABLE` ermittelt werden. Eine Liste der Sessionparameter kann aus der Oracle-Onlinedokumentation entnommen werden.

Listing 17.15: Sessionmodifizierbare Initialisierungsparameter

```
SQL> col name format a30
SQL> SELECT name, isses_modifiable
  2  FROM v$system_parameter;
```

- [autoId0]
- [sthref3228]



Listing 17.16: Beispiel für ALTER SESSION

```
SQL> show parameter nls_language
NAME                                     TYPE        VALUE
-----                                     -----
nls_language                           string      GERMAN

SQL> ALTER SESSION
  2  SET nls_language = 'AMERICAN';
Session altered.
```

Das Ergebnis des `ALTER SESSION`-Statements aus Beispiel ?? kann nicht mit Hilfe des SQL*Plus-Kommandos `show parameter` geprüft werden, da dieses nur Systemparameter anzeigt, aber keine Sessionparameter. Die Werte aller NLS-Sessionparameter können mit Hilfe der View `V$NLS_PARAMETERS` angezeigt werden.

Listing 17.17: Sessionparameter mit Hilfe von `V$NLS_PARAMETERS` ermitteln

```
SQL> col parameter format a30
SQL> col value format a30

SQL> SELECT *
  2  FROM    v$nls_parameters
  3  WHERE   parameter LIKE 'NLS_LANGUAGE';

PARAMETER          VALUE
-----
NLS_LANGUAGE        AMERICAN
```

17.4.3 SPFiles/PFiles generieren

Oracle bietet verschiedene Möglichkeiten, um ein PFile bzw. ein SPFile zu erzeugen:

- SPFile aus einem PFile
- SPFile aus den aktuellen Initialisierungsparametern der Instanz
- PFile aus einem SPFile
- PFile aus den aktuellen Initialisierungsparametern der Instanz

Ein SPFile generieren

Da ein SPFile eine Binärdatei ist, kann es nicht von Hand erstellt werden. Oracle stellt das Kommando `CREATE SPFILE` zur Verfügung, um ein SPFile aus einem PFile zu generieren.

Beispiel ?? zeigt die Syntax des `CREATE SPFILE`-Kommandos. Angaben in eckigen Klammern sind optional.

Listing 17.18: `CREATE SPFILE`

```
CREATE SPFILE = [pfad/dateiname.ora]
FROM PFILE = [pfad/dateiname.ora];
```



Die **SPFILE**-Klausel kann wahlweise den Dateinamen der Zielfile annehmen. Wird kein Dateiname angegeben, wird die Datei `ORACLE_HOME/dbs/spfile<SID>.ora` angelegt. Dies funktioniert jedoch nur, wenn die Instanz heruntergefahren ist, da Oracle sonst den Speicherort der Datei `ORACLE_HOME/dbs/spfile<SID>.ora` schützt, so dass diese nicht überschrieben werden kann.

Ist die Instanz gestartet, muss für die **SPFILE**-Klausel ein Dateiname angegeben werden, da Oracle sonst die Fehlermeldung „ORA-32002: cannot create SPFILE already being used by the instance“ anzeigt.

Dient ein PFile als Quelle kann hier optional der Name der Quelldatei angegeben werden. Ohne Dateiname wird die Datei `ORACLE_HOME/dbs/init<SID>.ora` als Quelle genutzt. Ist diese Datei nicht vorhanden, antwortet Oracle mit der Fehlermeldung:

```
ORA-01078: failure in processing system parameters
LRM-00109: could not open parameter file
'/u01/app/oracle/product/11.2.0/ORCL/dbs/initorcl.ora'
```

Durch die Angabe von `MEMORY`, statt `PFILE`, kann ein SPFile mit den Werten der aktuellen Initialisierungsparameter erzeugt werden. Beispiel ?? zeigt verschiedene Varianten des **CREATE SPFILE**-Kommandos.

Listing 17.19: Beispiele für **CREATE SPFILE**

```
SQL> CREATE SPFILE
  2  FROM    PFILE;

File created.

SQL> CREATE SPFILE = '/home/oracle/spfileorcl.ora'
  2  FROM    PFILE;

File created.

SQL> CREATE SPFILE = '/home/oracle/spfileorcl.ora'
  2  FROM    PFILE      = '?/dbs/initorcl.ora'

File created.

SQL> CREATE SPFILE = '/home/oracle/spfileorcl.ora'
  2  FROM    MEMORY;

File created.
```



Das ? in einer Pfadangabe dient als Synonym für die Umgebungsvariable ORACLE_HOME.



Wird kein Quell-SPFile angegeben, wird der Wert des Initialisierungsparameters spfile benutzt, um das aktuelle SPFile zu ermitteln. Wird kein Dateiname für das PFile angegeben, wird im Verzeichnis ORACLE_HOME/dbs die Datei init<SID>.ora angelegt.

Ein PFile generieren

Analog zum Kommando `CREATE SPFILE` existiert das `CREATE PFILE`-Statement.

Listing 17.20: Beispiele für `CREATE PFILE`

```
SQL> CREATE PFILE
  2  FROM    SPFILE;

SQL> CREATE PFILE = '/home/oracle/initorcl.ora'
  2  FROM    SPFILE;

SQL> CREATE PFILE = '/home/oracle/initorcl.ora'
  2  FROM    SPFILE      = '?/dbs/initorcl.ora'

SQL> CREATE PFILE = '/home/oracle/initorcl.ora'
  2  FROM    MEMORY;
```

17.4.4 Hochfahren einer Instanz mit alternativer Parameterdatei

Soll beim Hochfahren der Instanz eine alternative Parameterdatei genutzt werden. Das Schlüsselwort `pfile` dient dazu, den Namen der alternativen Parameterdatei anzugeben. Es kann nur eine Parameterdatei verarbeiten, keine Serverparameterdatei!

Listing 17.21: Start mit alternativer Parameterdatei

```
SQL> startup pfile='/home/oracle/initorcl.ora';
ORACLE instance started.

Total System Global Area  643084288 bytes
Fixed Size                  2215984  bytes
Variable Size                222302160  bytes
Database Buffers            411041792  bytes
Redo Buffers                 7524352  bytes
Database mounted.
Database opened.
```



- [i1006091]

17.5 Memory Management

Wie in Abschnitt ?? bereits beschrieben, kennt Oracle drei verschiedene Arten des Memory Managements: Manual Shared Memory Management, Automatic Shared Memory Management und seit Oracle 11g auch Automatic Memory Management. Die Konfiguration dieser Memory Management Modi funktioniert mit Hilfe von Initialisierungsparametern.

17.5.1 Manual Shared Memory Management

Beim Manual Shared Memory Management müssen alle Komponenten der SGA bzw. der PGAs einzeln definiert werden. Für die SGA-Komponenten existieren die folgenden Initialisierungsparameter:

- db_cache_size: Legt die Größe des Database Buffer Caches fest.
- shared_pool_size: Dimensioniert den Shared Pool.
- large_pool_size: Gibt die Größe des Large Pool an.
- java_pool_size: Definiert die Größe des Java Pools.
- streams_pool_size: Legt die Größe des Streams Pools fest.
- log_buffer: Gibt die Größe des Redo Log Buffers an.



Auf einige Komponenten der SGA, wie z. B. den Large Pool oder den Streams Pool wird in dieser Unterlage nicht näher eingegangen! Zudem existieren noch weitere Parameter, die über den Horizont dieses Skriptes hinausgehen.

Die aktuellen Werte dieser Parameter können mit der View V\$SGAINFO abgefragt werden.

Listing 17.22: Größe der SGA-Komponenten ermitteln

```
SQL> SELECT name, bytes
  2  FROM v$sgainfo;
```

NAME	BYTES
Fixed SGA Size	2217264
Redo Buffers	6926336
Buffer Cache Size	260046848
Shared Pool Size	121634816
Large Pool Size	4194304
Java Pool Size	4194304
Streams Pool Size	0
Shared IO Pool Size	0
Granule Size	4194304
Maximum SGA Size	764121088
Startup overhead in Shared Pool	71303168
Free SGA Memory Available	364904448

17.5.2 Automatic Shared Memory Management (ASMM)

Mit dem Automatic Shared Memory Management kamen in Oracle 10g zwei neue Initialisierungsparameter: `sga_target` und `pga_aggregate_target`.

Um ASMM zu aktivieren, müssen folgende Voraussetzungen gegeben sein:

- Der Parameter `sga_target` muss einen Wert größer 0 haben.
- Es muss ein SPFile benutzt werden.
- Der Parameter `statistics_level` muss einen der beiden Werte „typical“ oder „all“ haben.
- Der Parameter `shared_pool_size` muss einen Wert größer 0 haben.

Mit dem `sga_target` wird die Speichergesamtmenge für alle Komponenten der SGA gesetzt. Er ist dynamisch und kann somit jederzeit geändert werden. Seine maximale Größe wird durch den statischen Parameter `sga_max_size` begrenzt (Hardlimit).

Damit die Einstellungen für die SGA-Komponenten auch nach einem Neustart der Instanz erhalten bleiben, werden in der Parameterdatei zusätzliche Initialisierungsparameter geführt: `_db_cache_size`, `_shared_pool_size`, `_large_pool_size`, `_java_pool_size` und `_streams_pool_size`. Diese Parameter, die an dem doppelten Unterstrich vor ihrem Namen zu erkennen sind, dienen als Zwischenspeicher bis zum nächsten Instanzstart.

Zusätzlich zu sga_target kann der DBA Parameter, wie db_cache_size, shared_pool_size oder large_pool_size dazu benutzen, um Mindestgrößen für diese Speicherbereiche zu setzen. Hierzu ein Beispiel:

Listing 17.23: Ein Rechenbeispiel

```
SQL> ALTER SYSTEM
2  SET sga_target = 600M;

System altered.

SQL> ALTER SYSTEM
2  SET shared_pool_size = 50M;

System altered.

SQL> ALTER SYSTEM
2  SET db_cache_size = 200M;
```

Wird eine Instanz, wie in Beispiel ?? konfiguriert, entfallen mindestens 50 Megabyte auf den Shared Pool, mindestens 200 Megabyte auf den Database Buffer Cache und 350 Megabyte können auf alle anderen Komponenten der SGA verteilt werden.

Zur Verwaltung der PGAs gibt es `pga_aggregate_target`. Dieser Parameter legt die gesamte Speichermenge fest, die für alle PGAs zur Verfügung steht. Statt einer fixen PGA-Größe, die der DBA selbst festlegen muss, kann die Datenbank nun selbst entscheiden, wie groß jede einzelne PGA werden kann. Serverprozesse die eine größere PGA benötigen können somit mehr Speicherplatz erhalten, als andere.

Problematisch an dieser Form der Speicherverwaltung ist, dass wenn ein zu kleiner Wert für `sga_target` und ein zu großer für `pga_aggregate_target` gesetzt wird, die Datenbank keinen automatischen Ausgleich schaffen kann, falls die SGA mehr Memory benötigt. Hier muss dann der DBA eingreifen und selbstständig die Werte anpassen.

17.5.3 Automatic Memory Management

Das mit Oracle 11g neu eingeführte Automatic Memory Management vereinfacht die Situation des DBAs erneut. Es kommen zwei neue Parameter hinzu:

- `memory_target`: Dieser dynamische Parameter definiert die Größe aller Speicherkomponenten der Instanz (SGA + PGA).
- `memory_max_target`: Dies ist ein statischer Parameter, der als hartes Limit für `memory_target` fungiert.

Der Vorteil dieser neuen Methode ist, dass die Datenbank automatisch einen Ausgleich zwischen SGA und PGA schaffen kann, falls eine der beiden Seiten zu wenig Speicher hat. Für das Automatic Memory Management gelten folgende Regeln:

Tabelle 17.2: Regeln für das Automatic Memory Management

<code>memory_target</code>	<code>sga_target</code>	<code>pga_aggregate_target</code>	<code>sga_max_size</code>	Auswirkung
$x \neq 0$	$x \neq 0$	$x \neq 0$	-	Die 2 Parameter <code>sga_target</code> sowie <code>pga_aggregate_target</code> sind Mindestwerte.
$x \neq 0$	$x \neq 0$	n. a.	-	$memory_target - sga_target = pga_aggregate_target$
$x \neq 0$	n. a.	n. a.	-	$memory_target - \text{Wert von } pga_aggregate_target$ ergibt <code>sga_target</code>
$x \neq 0$	0	0	0	<code>pga_aggregate_target: 40 %</code> und <code>sga_target: 60 %</code> von <code>memory_target</code>

memory_target	sga_target	pga_aggregate_target	sga_max_size
---------------	------------	----------------------	--------------

Listing 17.24: Auszug aus einer Parameterdatei

```
orcl.__db_cache_size=260046848
orcl.__java_pool_size=4194304
orcl.__large_pool_size=4194304
orcl.__pga_aggregate_target=268435456
orcl.__sga_target=402653184
orcl.__shared_pool_size=121634816
orcl.__streams_pool_size=0
*.compatible='11.2.0.0.0'
*.db_block_size=8192
*.db_domain='local'
*.db_name='orcl'
*.memory_max_target=730M
*.memory_target=640M
```

17.6 Verwaltung von Kontrolldateien

Eine Kontrolldatei ist eine Binärdatei, welche die Struktur einer Oracle-Datenbank aufzeichnet. Jede Oracle-Datenbank benötigt eine Kontrolldatei. Sie beinhaltet folgende Informationen:

- Globaler Datenbankname
- Dateinamen und Speicherorte der Daten- und Redo Log Dateien
- Zeitstempel der Datenbankerstellung
- Die aktuelle Log Sequence Number
- Checkpoint-Informationen

Um eine Datenbank öffnen zu können, muss der Schreibzugriff auf die Kontrolldateien möglich sein. Generiert werden diese Dateien während der Datenbankerstellung. Standardmäßig wird immer nur eine erstellt, der Administrator sollte jedoch dafür sorgen, dass mehrere Sicherheitskopien der Kontrolldatei auf mehreren Speichermedien (Spiegelung) zur Verfügung stehen.

17.6.1 Namensgebung für Kontrolldateien

Welche Kontrolldateien die Datenbank benutzt, wird durch den Initialisierungsparameter `control_files` in der Serverparameterdatei festgelegt. Er kann eine Liste von Dateinamen enthalten, wie das folgende Beispiel zeigt.

Listing 17.25: Der Parameter control_files

```
...
CONTROL_FILES='/u02/oradata/orcl/control01.ctl',
              '/u05/fast_recovery_area/orcl/control02.ctl'
...
```



Es werden alle angegebenen Kontrolldateien geöffnet und parallel in diese geschrieben.

17.6.2 Multiplexing der Kontrolldateien

Jede Oracle-Datenbank sollte mindestens zwei Kopien einer Kontrolldatei haben, die auf verschiedenen Datenträgern gespeichert sind. Wurde eine Kontrolldateikopie während des laufenden Betriebs beschädigt, stürzt die Instanz meist sofort ab.

Nach der Behebung des Medienfehlers kann die kaputte Kontrolldatei durch eine funktionsfähige Kopie erneuert werden.

Die Datenbank verwendet ihre Kontrolldateien wie folgt:

- Es wird gleichzeitig in alle Kontrolldateien geschrieben.
- Nur die erste aufgelistete Kontrolldatei wird gelesen
- Wird eine Kopie beschädigt, stürzt die Instanz meistens ab.

Eine Variante Kontrolldateien zu spiegeln ist, sie auf allen Datenträgern, die eine Redo Log Datei enthalten zu verteilen, da auch die Redo Logs gespiegelt werden sollten.

17.6.3 Hinzufügen und löschen von Kontrolldateikopien

Hinzufügen einer weiteren Kontrolldateikopie

Die zum Hinzufügen einer Kontrolldateikopie notwendigen Schritte sind:

1. Instanz herunterfahren und in die NOMOUNT-Phase bringen

Listing 17.26: Hinzufügen von Kontrolldateikopien 1

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.

SQL> startup nomount
ORACLE instance started.
```

2. Kopieren einer funktionsfähigen Kontrolldateikopie an den neuen Speicherort

Listing 17.27: Löschen von Kontrolldateikopien 3

```
SQL> host cp /u02/oradata/orcl/control01.ctl /u03/oradata/orcl/control03.ctl
```

3. Den Parameter control_files mit ALTER SYSTEM bearbeiten.

Listing 17.28: Hinzufügen von Kontrolldateikopien 2

```
SQL> show parameter control_files

NAME                                TYPE        VALUE
-----
control_files                        string      /u02/oradata/orcl/control01.ctl,
                                         /u05/fast_recovery_area/orcl/control02.ctl

SQL> ALTER SYSTEM
  2 SET control_files='/u02/oradata/orcl/control01.ctl',
     '/u05/fast_recovery_area/orcl/control02.ctl',
     '/u03/oradata/orcl/control03.ctl'
  3 SCOPE=spfile;
```

4. Neustart der Instanz

Löschen von Kontrolldateikopien

Zum Löschen einer Kontrolldateikopie sind drei Schritte notwendig:

1. Instanz herunterfahren und in die NOMOUNT-Phase bringen

Listing 17.29: Löschen von Kontrolldateikopien 1

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
```

```
SQL> startup nomount
ORACLE instance started.
```

2. Den Parameter `control_files` mit `ALTER SYSTEM` bearbeiten.

Listing 17.30: Löschen von Kontrolldateikopien 2a

```
SQL> show parameter control_files

NAME                      TYPE        VALUE
-----
control_files              string      /u02/oradata/orcl/control01.ct
                           , /u05/fast_recovery_area/orc
                           l/control02.ctl
```

Listing 17.31: Löschen von Kontrolldateikopien 2b

```
SQL> ALTER SYSTEM
  2  SET control_files='/u02/oradata/orcl/control01.ctl',
  3  SCOPE=spfile;
```

3. Löschen der Kontrolldateikopie mit Hilfe des Betriebssystems

Listing 17.32: Löschen von Kontrolldateikopien 3

```
SQL> host rm -f /u05/fast_recovery_area/orcl/control02.ctl
```

4. Neustart der Instanz



- [i1006088]

17.7 Verwalten der Redo Logs

Für den Betrieb einer Oracle-Datenbank wird ein Set aus „Redo Log Gruppen“, umgangssprachlich als „Redo Logs“ bezeichnet, benötigt. Eine Redo Log Gruppe besteht aus einer oder mehreren Redo Log Dateien, die auch als Redo Log Member bezeichnet werden. Die primäre Funktion der Redo Logs ist das Aufzeichnen aller Änderungen, die an den Daten vorgenommen wurden (Nutz- und Metadaten).

17.7.1 Redo Log Gruppen und Member erstellen

Obwohl die Konfiguration der Redo Log Gruppen bereits vor der Installation einer Datenbank erdacht werden sollte, kann es notwendig werden, weitere Redo Log Gruppen oder neue Member zu erstellen. Für die Erstellung von Redo Log Gruppen und Membern muss der Nutzer das Privileg `ALTER DATABASE` besitzen.

Das Erzeugen einer neuen Redo Log Gruppe geschieht mit `ALTER DATABASE ADD LOGFILE`.

Listing 17.33: Erzeugen einer Redo Log Gruppe

```
SQL> ALTER DATABASE
  2  ADD LOGFILE '/u02/oradata/orcl/redo04a.log'
  3  SIZE 50M;
```



Es sollte immer eine vollständige Pfadangabe für die Member verwendet werden, da diese sonst im Verzeichnis ORACLE_HOME/dbs erstellt werden.

Das obige Statement kann durch die Angabe der Redo Log Gruppen Nummer erweitert werden:

Listing 17.34: Erzeugen einer Redo Log Gruppe mit Angabe der Gruppennummer

```
SQL> ALTER DATABASE
  2 ADD LOGFILE GROUP 5
  3 ('/u02/oradata/orcl/redo05a.log',
  4   '/u03/oradata/orcl/redo05b.log')
  5 SIZE 50M;
```

Die Angabe von **GROUP 5** sorgt dafür, das die Datenbank versucht, die neue Redo Log Gruppe mit der Nummer 5 zu erstellen. Dies kann aber nur funktionieren, wenn diese Nummer noch nicht belegt ist. Die beiden neu erstellten Gruppen können durch die View v\$LOG sichtbar gemacht werden.

Listing 17.35: Die View v\$LOG

```
SQL> SELECT group#, members,
  2         bytes / POWER(1024, 2) AS Megabytes
  3     FROM v$log
  4    ORDER BY group#;
```

GROUP#	MEMBERS	MEGABYTES
1	1	50
2	1	50
3	1	50
4	1	50
5	2	50

Die Spalte **MEMBERS** zeigt, dass die Gruppen 1 bis 4 mit jeweils nur einem Member angelegt wurden, was eine sehr gefährliche Konfiguration darstellt. Wird dieser eine Member beschädigt, ist folglich die gesamte Gruppe beschädigt. Dies führt zu Problemen bei einem Datenbank-Recovery-Vorgang, wenn der Memberausfall nicht rechtzeitig bemerkt wird.

Redo Log Member einer Gruppe hinzufügen

Um einer Redo Log Gruppe einen neuen Member hinzuzufügen, wird das SQL-Kommando **ALTER DATABASE ADD LOGFILE** zusammen mit dem Dateinamen des Members, sowie der Nummer der betroffenen Redo Log Gruppe verwendet.

Listing 17.36: Hinzufügen eines Members zu einer Redo Log Gruppe

```
SQL> ALTER DATABASE
  2  ADD LOGFILE MEMBER '/u03/oradata/orcl/redo01b.log'
  3  TO GROUP 1;
```

Passend zur View V\$LOG, die Informationen zu allen Log Gruppen anzeigt, gibt es auch die View V\$LOGFILE. Dies befasst sich mit den Redo Log Members.

Listing 17.37: Informationen über Redo Log Member sammeln

```
SQL> col member format a50
SQL> SELECT group#, member
  2  FROM v$logfile;

GROUP# MEMBER
-----
3 /u02/oradata/orcl/redo03.log
2 /u02/oradata/orcl/redo02.log
1 /u02/oradata/orcl/redo01.log
1 /u03/oradata/orcl/redo01b.log
4 /u02/oradata/orcl/redo04a.log
5 /u02/oradata/orcl/redo05a.log
5 /u03/oradata/orcl/redo05b.log
```

17.7.2 Redo Logs verschieben/umbenennen

Es gibt Situationen, die es notwendig machen, Redo Log Member an einen anderen Speicherort zu verschieben. Dies ist beispielsweise dann der Fall, wenn ein Datenträger aus dem System entfernt werden soll oder wenn ein Datenträger eine zu hohe Auslastung hat, weil sich zu viele Redo Log Dateien, Datendateien und Kontrolldateien darauf befinden.

Das Verschieben/Umbenennen besteht im Wesentlichen aus zwei Schritten:

1. Verschieben/Umbenennen der Redo Log Datei mit Betriebssystemmitteln
2. Ändern des Dateipfades in der Datenbank

Für diesen Vorgang muss der Nutzer das Privileg `ALTER DATABASE` auf Seiten der Datenbank und entsprechende Rechte im Betriebssystem haben.

1. Abfragen von `V$LOGFILE`

Listing 17.38: Speicherorte der Member ermitteln

```
SQL> col member format a50
SQL> SELECT group#, member
  2  FROM v$logfile;

GROUP# MEMBER
-----
3 /u02/oradata/orcl/redo03.log
2 /u02/oradata/orcl/redo02.log
1 /u02/oradata/orcl/redo01.log
1 /u03/oradata/orcl/redo01b.log
```

```
4 /u02/oradata/orcl/redo04a.log  
5 /u02/oradata/orcl/redo05a.log  
5 /u03/oradata/orcl/redo05b.log
```

2. Instanz herunterfahren

Listing 17.39: Instanz herunterfahren

```
SQL> shutdown immediate
```

3. Verschieben der Datei /u02/oradata/orcl/redo01.log mit Betriebssystemmitteln nach /u02/oradata/orcl/redo01a.log

Listing 17.40: Verschieben der Redo Log Datei mit BS Mitteln

```
SQL> host mv /u02/oradata/orcl/redo01.log /u02/oradata/orcl/redo01a.log
```

4. Datenbank in die MOUNT-Phase bringen

Listing 17.41: Datenbank MOUNTen

```
SQL> startup mount
```

5. Ändern des Dateipfades in der Datenbank

Listing 17.42: Redo Log Datei umbenennen

```
SQL> ALTER DATABASE  
2 RENAME FILE '/u02/oradata/orcl/redo01.log'  
TO '/u02/oradata/orcl/redo01a.log';
```

6. Datenbank öffnen

Listing 17.43: Datenbank öffnen

```
SQL> ALTER DATABASE OPEN;
```

17.7.3 Löschen von Redo Log Gruppen und Membern

Eine Redo Log Gruppe löschen

Sollte eine Redo Log Gruppe nicht mehr von Nöten sein, kann diese gelöscht werden. Hierfür benötigt der Nutzer wiederum das ALTER DATABASE-Privileg. Außerdem sollten vor dem Löschen die folgenden Punkte bedacht werden:

- Jede Instanz benötigt mindestens zwei Redo Log Gruppen mit einer beliebigen Anzahl von Membern.

- Nur eine Redo Log Gruppe, die den Status Inactive oder Unused hat, kann gelöscht werden.
- Falls die Archivierung für Redo Logs aktiviert ist, sollte vorher geprüft werden, ob die zu löschen Gruppe bereits archiviert wurde.

- Prüfen, ob die Redo Log Gruppe inaktiv ist und evtl. bereits archiviert wurde

Listing 17.44: Status der Redo Logs prüfen

```
SQL> SELECT group#, archived, status
  2  FROM v$log;
```

- Durchführen eines Log Switches und eines Checkpoints, damit die Gruppe in den Status „inactive“ wechselt.

Listing 17.45: Log Switch + Checkpoint durchführen

```
SQL> ALTER SYSTEM SWITCH LOGFILE;
SQL> ALTER SYSTEM CHECKPOINT;
```

- Redo Log Gruppe löschen

Listing 17.46: Log Gruppe löschen

```
SQL> ALTER DATABASE DROP LOGFILE GROUP 5;
```

- Löschen aller Memberdateien der Redo Log Gruppe mit Betriebssystemmitteln.

Listing 17.47: Log Gruppe löschen

```
SQL> host rm /u02/oradata/orcl/redo05a.log /u03/oradata/orcl/redo05b.log
```

Redo Log Member löschen

Um einen Member aus einer Redo Log Gruppe löschen zu können, benötigt der Nutzer das Privileg ALTER DATABASE. Außerdem sollten vor dem Löschen die folgenden Punkte bedacht werden:

- Beim Löschen eines Members aus einer Redo Log Gruppe wird die Redo Log Konfiguration kurzzeitig asymmetrisch. Dieser Zustand sollte so schnell wie möglich bereinigt werden.
- Nur in einer Redo Log Gruppe, die den Status Inactive oder Unused hat, können Member gelöscht werden.
- Falls die Archivierung für Redo Logs aktiviert ist, sollte vorher geprüft werden, ob die zu löschen Gruppe bereits archiviert wurde.
- Der letzte Member einer Redo Log Gruppe kann nicht gelöscht werden. Es muss dann die ganze Gruppe gelöscht werden.

1. Prüfen, ob die Redo Log Gruppe inaktiv ist und evtl. bereits archiviert wurde

Listing 17.48: Status der Redo Logs prüfen

```
SQL> SELECT group#, archived, status  
2   FROM v$log;
```

2. Durchführen eines Log Switches und eines Checkpoints, damit die Gruppe in den Status „inactive“ wechselt.

Listing 17.49: Log Switch + Checkpoint durchführen

```
SQL> ALTER SYSTEM SWITCH LOGFILE;  
  
SQL> ALTER SYSTEM CHECKPOINT;
```

3. Redo Log Member löschen

Listing 17.50: Log Member löschen

```
SQL> ALTER DATABASE  
2   DROP LOGFILE MEMBER '/u03/oradata/orcl/redo01b.log';
```

Listing 17.51: Log Member löschen

```
SQL> host rm -f /u03/oradata/orcl/redo01b.log
```

Zuletzt muss die betreffende Datei noch betriebssystemseitig gelöscht werden.

17.7.4 Defekte Member bearbeiten

Status von Redo Log Membern

Wird ein Redo Member beschädigt, erhält er einen Fehlerstatus. Dieser kann in der View V\$LOGFILE aus der Spalte STATUS ersehen werden. Es gibt folgende Stati:

- **NULL** (Kein Wert): Ist die Statusspalte leer, ist der Redo Log Member voll funktionsfähig.
- **INVALID**: Aus einem nicht näher definierten Grund, kann auf die Datei nicht zugegriffen werden.
- **STALE**: Der Inhalt der Redo Log Member Datei ist nicht vollständig. Dies kann entstehen, wenn während der Nutzung einer Redo Log Gruppe die Instanz abstürzt.
- **DELETED**: Dieser Status zeigt an, dass die Datei gelöscht wurde.

Leeren einer Redo Log Gruppe

Wenn im laufenden Betrieb der Datenbank eine Redo Log Gruppe zerstört wird und damit die Archivierung unmöglich geworden ist, muss die beschädigte Gruppe geleert werden. Der Vorteil dieser Methode ist, dass die Datenbank hierzu nicht heruntergefahren werden muss. Weiterhin gibt es zwei Fälle, in denen das Leeren einer Redo Log Gruppe die einzige Möglichkeit darstellt, das Problem zu lösen:

- Wenn es nur zwei Redo Log Gruppen gibt
- Wenn in der Redo Log Gruppe mit dem Status Current eine Redo Log Datei defekt ist

Um eine Redo Log Gruppe zu leeren, die gerade in der Archivierung befindlich ist, wird folgendes Kommando verwendet:

Listing 17.52: Redo Log Gruppe leeren

```
SQL> ALTER DATABASE
  2  CLEAR LOGFILE GROUP 3;
```

Zum Leeren einer noch nicht archivierten Redo Log Gruppe, muss das SQL-Schlüsselwort **UNARCHIVED** hinzugefügt werden.

Listing 17.53: Eine nicht archivierte Redo Log Gruppe leeren

```
SQL> ALTER DATABASE
  2  CLEAR UNARCHIVED LOGFILE GROUP 3;
```

Dieses Statement verhindert, dass die betreffende Redo Log Gruppe jemals archiviert wird.



Zu beachten ist: Wird eine Redo Log Gruppe geleert, die zum Recovery der Datenbank benötigt wird, sind alle Backups, die dieses Redo Log benötigen nutzlos und es sollte sofort ein neues Backup der Datenbank angefertigt werden. In der Alert Log Datei werden die ungültigen Datenbankbackups aufgelistet (nur mit RMAN erstellte Backups).

17.7.5 Informationen über Redo Log Gruppen/Member sammeln

- [i1007497]



17.8 Verwalten der archivierten Redo Logs

17.8.1 Was sind archivierte Redo Logs

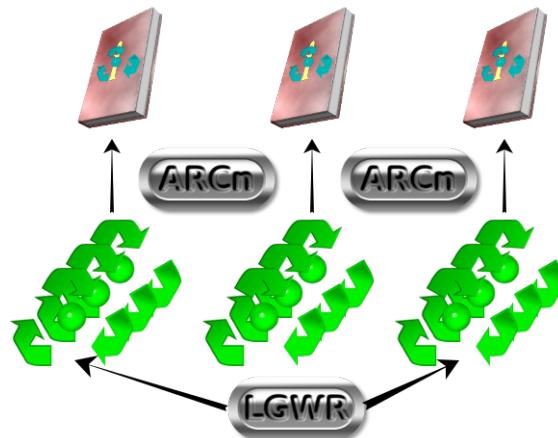
Es ist möglich, die gefüllten Redo Log Gruppen einer Oracle-Datenbank an einem oder mehreren anderen Speicherorten zu sichern. Diese gesicherten Redo Logs werden als „Archived Redo Logs“ oder einfach als „Archive Logs“ bezeichnet. Der Prozess des Sicherns der Redo Logs heißt „Archiving“¹.



Wird eine Redo Log Gruppe durch multiplexing auf mehrere Speicherorte verteilt, archiviert der Archiver-Prozess immer nur eine der identischen Kopien der Redo Log Dateien.

Damit die Redo Logs einer Oracle-Datenbank archiviert werden, muss die Datenbank in den „Archivelog-Modus“ versetzt werden. Das Gegenstück zum Archivelog-Modus ist der „Noarchivelog-Modus“. Im Noarchivelog-Modus findet keine Archivierung statt. Welcher der beiden Modi für die Datenbank verwendet werden sollte, hängt davon ab, ob Datenverlust akzeptabel ist oder nicht.

Abb. 17.6:
Archivierung der
Redo Logs



17.8.2 Für die Archivierung notwendige Initialisierungsparameter

Anzahl der Archiver-Prozesse festlegen

Der Initialisierungsparameter `log_archive_max_processes` legt die Anzahl der zu startenden Archiver-Prozesse fest (Standardwert ist 4). Eine Veränderung dieses Standardwertes ist normalerweise nicht

¹archiving = engl. Archivierung, Sicherung

notwendig, da die Datenbank selbstständig zusätzliche Archiver-Prozesse startet, wenn dies erforderlich erscheint.

Ein möglicher Grund für eine Änderung dieses Parameters ist z. B., dass es performanter ist, gleich die richtige Anzahl Archiver-Prozesse zu starten. Es können bis zu 30 Archiver-Prozesse gleichzeitig gestartet werden.

Speicherorte der Archiver-Prozesse festlegen

Zuständig für die Festlegung der Speicherorte der Archive Logs sind die Initialisierungsparameter `log_archive_dest_1` bis `log_archive_dest_31`. Um beispielsweise die beiden Speicherorte `/u02/backup/archive_1` und `/u03/backup/archive_logs` festzulegen, müssen die beiden Initialisierungsparameter `log_archive_dest_1` und `log_archive_dest_2` wie folgt angepasst werden:

Listing 17.54: `log_archive_dest`-Parameter setzen

```
SQL> ALTER SYSTEM
  2 SET log_archive_dest_1='LOCATION=/u02/backup/archive_logs';
SQL> ALTER SYSTEM
  2 SET log_archive_dest_2='LOCATION=/u03/backup/archive_logs';
```

Dateinamen der Archive Logs konfigurieren

Um das Format des Dateinamens für die Archive Logs zu setzen wird der Parameter `log_archive_format` verwendet. Der komplette Name eines Archive Logs setzt sich dann aus den Parameter `log_archive_dest_n` + `log_archive_format` zusammen.

Listing 17.55: `log_archive_format`-Parameter setzen

```
SQL> ALTER SYSTEM
  2 SET log_archive_format='arch_%s_%r_%t.log',
  3 SCOPE=spfile;
```



Hat der Initialisierungsparameter `compatible` einen Wert größer oder gleich 10.0, müssen im Parameter `log_archive_format` die Platzhalter `%r`, `%s` und `%t` zwingend verwendet werden. Falls nicht wird die Fehlermeldung „ORA-19905: `log_archive_format` must contain `%s`, `%t` and `%r`“ ausgelöst.

Die Konfiguration der Archive Log Destination kann mittels der View `V$ARCHIVE_DEST` abgefragt werden.

Listing 17.56: Die Konfiguration der Archive Log Destination abfragen

```
SQL> col dest_name format a20
```

```
SQL> col destination format a30
SQL> col error format a40
SQL> set linesize 300
```

Listing 17.57: Die Konfiguration der Archive Log Destination abfragen - Fortsetzung

```
SQL> SELECT dest_name, status, destination, error
  2  FROM v$archive_dest
  3 WHERE dest_id < 3;
```

DEST_NAME	STATUS	DESTINATION	ERROR
LOG_ARCHIVE_DEST_1	VALID	/u02/backup/archive_logs	
LOG_ARCHIVE_DEST_2	VALID	/u03/backup/archive_logs	

Status der Speicherorte

Jede Archive Log Destination ist immer mit einem Status versehen, der etwas über ihre Funktionsfähigkeit und ihre Nutzung aussagt.

- **Valid/Invalid:** Es wurde ein gültiger bzw. ein ungültiger Speicherort angegeben. Gültig heißt, dass der Speicherort existieren muss.
- **Enabled/Disabled:** Ist der Speicherort aktiviert (wird genutzt) oder deaktiviert (wird nicht genutzt)?
- **Active/Inactive:** Ist der Speicherort zugänglich oder aufgrund eines Fehlers unzugänglich?

Tabelle Tabelle ?? zeigt verschiedene Kombinationen dies Stati.

Tabelle 17.3: Mögliche Zustände der LOG_ARCHIVE_DEST_n-Speicherorte

Status	Eigenschaften			Bedeutung
	Valid	Enabled	Active	
VALID	Ja	Ja	Ja	Es wurde ein gültiger Speicherort angegeben, der auch erreichbar ist.
INACTIVE	Nein	–	–	Es wurde kein Speicherort angegeben.
ERROR	Ja	Ja	Nein	Es trat ein Fehler auf, als versucht wurde, am angegebenen Speicherort eine Datei zu erstellen.
FULL	Ja	Ja	Nein	Kein freier Speicher am Speicherort verfügbar.
DEFERRED	Ja	Nein	Ja	Der Speicherort wurde durch den Nutzer zeitweilig deaktiviert.
DISABLED	Ja	Nein	Nein	Der Speicherort musste aufgrund eines Zugriffsehlers zeitweilig deaktiviert werden.
BAD PARAM	–	–	–	Ein nicht näher definierbarer Fehler ist aufgetreten (z. B. wurde ein ungültiger Wert für einen Parameter angegeben).

Probleme mit fehlerhaften Speicherorten

Es kann vorkommen, dass die Speicherung eines Archive Logs an einem der angegebenen Speicherorte fehlschlägt. Oracle stellt verschiedene Möglichkeiten bereit, wie auf solche Ausfälle reagiert werden kann. Eine Möglichkeit besteht darin, mit dem Parameter `log_archive_min_succeed_dest` anzugeben, auf wie vielen Speicherorten die Archivierung als erfolgreich gelten muss, ehe die zu archivierende Redo Log Gruppe wieder verwendet werden kann. Standardwert für diesen Parameter ist 1. Der Maximalwert ist 10.

In Kombination dazu kann mit jedem `log_archive_dest_n`-Parameter ein Speicherort als *optional* (Standardwert) oder als *mandatory*² deklariert werden. Je nachdem, wie der Parameter `log_archive_min_succeed_dest` eingestellt ist, muss die Archivierung mindestens an allen als mandatory deklarierten Speicherorten erfolgreich gewesen sein, bevor der Log Writer-Prozess die betreffende Redo Log Gruppe wieder verwenden kann.

Die folgende Tabelle soll das Verhalten der Datenbank verdeutlichen:

- Es wurden 4 Speicherorte deklariert, 2 als optional und 2 als mandatory
- `log_archive_min_succeed_dest` wird auf die Werte 1 bis 5 eingestellt. Die Spalte Wert zeigt an, auf welchen Wert dieser Parameter gesetzt wurde. Die Spalte Auswirkungen beschreibt welche Auswirkungen diese Einstellung hat.

Tabelle 17.4: Auswirkungen des Parameters `log_archive_min_succeed_dest`

Wert	Auswirkungen
1	Die Datenbank ignoriert den für <code>log_archive_min_succeed_dest</code> eingestellten Wert und nimmt stattdessen die Anzahl der als mandatory deklarierten Speicherorte.
2	Die Archivierung gilt als erfolgreich, wenn an beiden als mandatory deklarierten Speicherorten die Speicherung erfolgreich war. Die als optional deklarierten Speicherorte werden bei der Überprüfung ignoriert.
3	Es müssen beide als mandatory und mindestens einer der als optional deklarierten Speicherorte erfolgreich gewesen sein, bevor die Archivierung als erfolgreich gilt.
4	Die Archivierung ist erst dann erfolgreich, wenn an allen Speicherorten die Archived Logs gespeichert werden konnten.
5	Es tritt ein Fehler auf, da die Anzahl der Speicherorte geringer ist, als der Wert für <code>log_archive_min_succeed_dest</code> .

²mandatory = engl. obligatorisch, zwingend

Für das Zusammenspiel zwischen den beiden als mandatory deklarierten Speicherorten und dem Parameter `log_archive_min_succeed_dest` gelten die folgenden Regeln:

- Wird ein Speicherort nicht explizit als mandatory deklariert, wird er als optional angesehen.
- Wird für den Parameter `log_archive_min_succeed_dest` ein Wert angegeben, wird mindestens ein Speicherort als mandatory betrachtet.
- Der Wert für `log_archive_min_succeed_dest` kann nicht größer sein, als die Anzahl der konfigurierten Speicherorte.

Das folgende Beispiel zeigt, wie ein Speicherort als mandatory deklariert wird:

Listing 17.58: `log_archive_dest_n` als mandatory deklarieren

```
SQL> ALTER SYSTEM
  2  SET log_archive_dest_1='LOCATION=/u02/backup/archive_logs MANDATORY';
SQL> ALTER SYSTEM
  2  SET log_archive_dest_2='LOCATION=/u03/backup/archive_logs OPTIONAL';
```

17.8.3 Eine Datenbank in den Archivelog-Modus versetzen

Eine Datenbank kann bereits bei ihrer Erstellung oder auch nachträglich in den Archivelog-Modus versetzt werden. Dazu sind folgende Schritte notwendig:

1. Anpassen der Initialisierungsparameter, die für das Archiving notwendig sind
2. Instanz konsistent herunterfahren
3. Die Datenbank in die MOUNT-Phase versetzen.
4. Den Archivierungsmodus einschalten.
5. Datenbank öffnen
6. (Herunterfahren der Datenbank und durchführen eines Backups)
7. (Datenbank hochfahren)

Listing 17.59: Archivelog-Modus aktivieren

```
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
```

```
SQL> shutdown immediate
Database closed.
Database dismounted.
ORACLE instance shut down.
```

Listing 17.60: Archivelog-Modus aktivieren - Fortsetzung

```
SQL> startup mount
ORACLE instance started.

Total System Global Area  764121088 bytes
Fixed Size                  2217264  bytes
Variable Size                503319248  bytes
Database Buffers            251658240  bytes
Redo Buffers                 6926336  bytes
Database mounted.

SQL> ALTER DATABASE ARCHIVELOG;

Database altered.

SQL> ALTER DATABASE OPEN;

Database altered.
```

17.9 Das Data Dictionary

Der wichtigste Teil einer Oracle-Datenbank ist das „Data Dictionary“. Es besteht aus einer Menge von Tabellen mit wichtigen Metainformationen, auf die alle Nutzer Lesezugriff haben. Darunter fallen:

- Die Definitionen aller Schemaobjekte der Datenbank (Tabellen, Views, Indizes, usw.)
- Wie viel Speicherplatz für Schemaobjekte reserviert und aktuell genutzt wird
- Standardwerte für Tabellenspalten
- Integritäts Constraints
- Benutzernamen aller Oracle-Nutzer
- Privilegien und Rollen mit Zuordnung zu den Nutzern
- Auditing Informationen
- Weitere generelle Metainformationen

Die Tabellen des Data Dictionary sind auf die gleiche Art und Weise organisiert, wie andere Tabellen auch. Sie sind jedoch im SYSTEM-Tablespace gespeichert und gehören dem Nutzer SYS.

Nicht nur, dass das Data Dictionary ein zentraler Punkt in der Datebank ist, es stellt auch ein wichtiges Hilfsmittel für alle Nutzer, vom Endnutzer bis zum Administrator dar. Es kann mit Hilfe von SQL-Befehlen abgefragt, nicht aber geändert werden.

Der Oracle Nutzer SYS ist Eigentümer der Basistabellen und Nutzer-Views des Data Dictionary. Kein anderer Oracle Nutzer außer ihm sollte Zugriff auf diese Tabellen haben, da dies die Datenbankintegrität verletzen könnte.

17.9.1 Benutzung und Struktur des Data Dictionary

Das Data Dictionary wird auf drei unterschiedliche Arten genutzt:

- Oracle selbst greift lesend auf das Data Dictionary zu, um Informationen über Nutzer, Privilegien und Schemaobjekte zu erhalten.
- Immer wenn ein DDL-Statement abgesetzt wird, wird das Data Dictionary durch Oracle modifiziert
- Jeder Oracle Nutzer kann das Data Dictionary als Nachschlagewerk nutzen.

Das Data Dictionary besteht aus den folgenden Bestandteilen:

Dynamic Performance Views (X\$-Views)

Oracle verwaltet eine Menge von Pseudo-Views, die als Dynamic Performance Views bezeichnet werden. Es handelt sich hierbei nicht um echte Views. Sie zeigen Informationen über die Instanz und die Datenbank an und werden dynamisch im laufenden Betrieb durch den Kern von Oracle selbst aktualisiert.

Diese View tragen das Präfix „x\$“ in ihrem Namen. Sie werden bei der Erstellung der Datenbank automatisch mit erstellt. Um den Inhalt der X\$-Views für Administratoren nutzbar zu machen, hat Oracle zusätzlich sogenannte V\$-Views geschaffen.

Dynamic Performance Views (V\$-Views)

V\$-Views benutzen als Informationsgrundlage die eben beschriebenen X\$-VIEWS. Während die Namensgebung bei X\$-Views sehr undurchschaubar ist (z. B. X\$KSMFS, X\$KSMSS oder X\$KCBWAIT) ist die der V\$-Views klar verständlich (z. B. V\$SESSION, V\$LOG oder V\$LOGFILE). Auch die Struktur der V\$-Views ist für Administratoren besser zu durchschauen, außerdem sind V\$-Views dokumentiert, im Gegensatz zu den X\$-Views.

Basis Tabellen

Dies sind die zugrundeliegenden Tabellen, die die Metainformationen enthalten. Sie werden nur von der Datenbank selbst genutzt, da sie durch Normalisierung und kryptischen Inhalt für normale Nutzer wenig durchschaubar sind. Einige Beispiele für Basistabellen sind USER\$, TAB\$, OBJ\$ oder AUD\$.

Nutzer-Views

Diese Views stellen eine Zusammenfassung des Inhalts der Basistabellen dar. Sie dekodieren den unübersichtlichen Inhalt der Basistabellen und machen ihn für Nutzer lesbar. Nicht jeder Nutzer hat auf alle Views Zugriff.

17.9.2 View-Klassen im Data Dictionary

Es gibt unterschiedliche Arten von Views im Data Dictionary. Einige sind für alle Nutzer zugänglich, andere nur für Administratoren. Die einzelnen Klassen können an dem Präfix in ihrem Namen erkannt werden. Folgende Klassen gibt es:

- **USER:** User-Views zeigen, welche Objekte im Schema des Nutzers existieren
- **ALL:** Erweiterte User-Views zeigen, auf welche Objekte der Nutzer Zugriff hat
- **DBA:** Administrative Views zeigen den gesamten Datenbankinhalt

Nicht von allen Views existieren immer alle drei Klassen, z. B. DBA_LOCKS.



Views mit dem Präfix USER

Diese Views sind die interessantesten für normale Nutzer. Sie haben folgende Eigenschaften:

- Sie zeigen das private Umfeld eines Nutzers, mit all seinen Objekten in der Datenbank.
- Sie zeigen nur Tabellenspalten die für den Nutzer relevant sind.
- Sie sind eine Untermenge der Views mit dem Präfix ALL.

Views mit dem Präfix ALL

ALL-Views zeigen das für den Nutzer sichtbare Umfeld in der Datenbank. Sie liefern Informationen über alle Datenbankobjekte, auf die der Nutzer Zugriff hat, inklusive der Informationen die durch USER-Views angezeigt werden. Im Gegensatz zu den USER-Views haben die ALL-Views eine Spalte OWNER, die den Eigentümer eines Objekts anzeigt.

Views mit dem Präfix DBA

Views mit dem Präfix DBA zeigen eine umfassende Ansicht der Datenbank mit allen Objekten, Privilegien und Nutzern. Auf diese Views hat nur administratives Personal Zugriff.

Die Tabelle DUAL

Die Tabelle DUAL ist eine Tabelle des Data Dictionary mit einer einzigen Spalte names DUMMY, die den Wert „X“ enthält. Sie ist für die Durchführung von Berechnungen gedacht, wie das folgende Beispiel zeigt.

Listing 17.61: Die Tabelle DUAL

```
SQL> SELECT SYSDATE AS Datum
  2  FROM    dual;

  DATUM
  -----
29.08.13
```



- [i125539]
- [i2112]

17.10 Informationen

17.10.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10019]
- [REFRN10021]
- [REFRN10033]
- [REFRN10075]
- [REFRN10079]
- [REFRN10086]
- [REFRN10089]
- [REFRN10090]
- [REFRN10091]
- [REFRN10094]
- [REFRN10284]
- [REFRN10285]
- [REFRN10123]
- [REFRN10165]
- [REFRN10202]
- [REFRN10198]
- [REFRN10256]
- [REFRN10243]
- [REFRN10214]

17.10.2 Verzeichnis der relevanten Data Dictionary Views



- [REFRN29014]
- [REFRN30007]
- [sthref3187]
- [REFRN30089]
- [sthref3267]
- [sthref3423]
- [REFRN30129]
- [REFRN30128]
- [REFRN30159]
- [REFRN30314]
- [REFRN30275]

17.11 Übungen - Verwalten einer Oracle-Instanz

1. Starten Sie SQL*Plus und melden Sie sich als Nutzer SYS an der Datenbank an.
2. Starten Sie nur die Instanz Ihrer Datenbank.
3. Bringen Sie Ihre Datenbank in die Mount-Phase.
4. Öffnen Sie Ihre Datenbank.
5. Fahren Sie Ihre Datenbank herunter, so das automatisch alle Nutzer abgemeldet und alle laufenden Transaktionen zurückgerollt werden.
6. Starten und öffnen Sie Ihre Datenbank.
7. Lassen Sie sich den Wert des Parameters `memory_target` anzeigen.
8. Ändern Sie den Wert des Initialisierungsparameters `memory_target` auf 800M.
9. Ändern Sie die zwei folgenden Initialisierungsparameter im SPFile.
 - `sga_target`: 400M
 - `pga_aggregate_target`: 100M
10. Starten Sie die Instanz neu.
11. Wie wirken sich die Einstellungen der beiden Initialisierungsparameter `sga_target` und `pga_aggregate_target` auf die Speicherverwaltung der Instanz aus?
12. Wie groß sind die folgenden SGA-Strukturen:

Parameter	Wert
Database Buffer Cache:	
Shared Pool	
Large Pool	
Maximale Größe der SGA	

13. Ändern Sie die Sprache Ihrer Session auf Deutsch.

14. Erstellen Sie aus Ihrem aktuellen SPFile die Datei /home/oracle/initorcl.ora.
15. Erstellen Sie aus den aktuellen Einstellungen Ihrer Instanz ein neues SPFile names /home/oracle/spfileorcl.ora
16. Erstellen Sie eine weitere Kontrolldateikopie /u02/oradata/orcl/control03.ctl und binden Sie sie in Ihre Instanz ein.

17. Um die weiteren Übungsaufgaben bearbeiten zu können, führen Sie bitte das Skript lab_java.sql als Nutzer SYS aus. Das SQL-Skript befindet sich im Verzeichnis /home/oracle/labs.

```
SQL> @/home/oracle/labs/lab_java.sql
```

18. Ermitteln Sie die Redo Log-Konfiguration Ihres Systems (Gruppen und Member). Die erste Zeile in der folgenden Tabelle ist ein Beispieleintrag.

Gruppe	/u01	/u02	/u03	/u04	/u05
1	redo01a.log	redo01b.log	redo01c.log	-	-

19. Passen Sie das Setup Ihrer Redo Log Gruppen so an, dass Sie drei Gruppen mit je 3 Membern haben. Verteilen Sie die Member sinnvoll über die vorhandenen Datenträger /u01 bis /u05 und notieren Sie sich die Änderungen in der obigen Tabelle.
20. Prüfen Sie, ob sich Ihre Datenbank im Archive-Log-Modus befindet. Falls nicht wechseln Sie in den Archive-Log-Modus.
21. Ändern Sie die Parameter der Archivierung wie folgt:

Optional	/u02	/u03	/u04	/u05
Ja	dest 1	-	-	-
Nein	-	dest 2	-	-
Ja	-	-	dest 3	-

- Die Archivierung muss an mindestens zwei Speicherorten erfolgreich sein.
- Das Dateinamensformat der Archive Log Files muss wie folgt aufgebaut sein: archive_%d_%t_%s_%r.arc

17.12 Lösungen - Verwalten einer Oracle Instanz

1. Starten Sie SQL*Plus und melden Sie sich als Nutzer SYS an der Datenbank an.

```
sqlplus / as sysdba
```

2. Starten Sie nur die Instanz Ihrer Datenbank.

```
SQL> startup nomount
```

3. Bringen Sie Ihre Datenbank in die Mount-Phase.

```
SQL> ALTER DATABASE MOUNT;
```

4. Öffnen Sie Ihre Datenbank.

```
SQL> ALTER DATABASE OPEN;
```

5. Fahren Sie Ihre Datenbank herunter, so das automatisch alle Nutzer abgemeldet und alle laufenden Transaktionen zurückgerollt werden.

```
SQL> shutdown immediate
```

6. Starten und öffnen Sie Ihre Datenbank.

```
SQL> startup
```

7. Lassen Sie sich den Wert des Parameters memory_target anzeigen.

```
SQL> show parameter memory_target
```

8. Ändern Sie den Wert des Initialisierungsparameters memory_target auf 800M.

```
SQL> ALTER SYSTEM  
2 SET memory_target = 800M;
```

9. Ändern Sie die zwei folgenden Initialisierungsparameter im SPFile.

- sga_target: 400M
- pga_aggregate_target: 100M

```
SQL> ALTER SYSTEM
  2  SET sga_target = 400M
  3  SCOPE=spfile;
SQL> ALTER SYSTEM
  2  SET pga_aggregate_target = 100M
  3  SCOPE=spfile;
```

10. Starten Sie die Instanz neu.

```
SQL> shutdown immediate
SQL> startup
```

11. Wie wirken sich die Einstellungen der beiden Initialisierungsparameter `sga_target` und `pga_aggregate_target` auf die Speicherverwaltung der Instanz aus?

Antwort: Beide werden als Mindestwerte für die Größe der SGA (`sga_target`) und der PGAs (`pga_aggregate_target`) herangezogen. Der restliche Speicherplatz von 300M (`memory_target - sga_target - pga_aggregate_target`) kann dynamisch auf beide Strukturen verteilt werden.

12. Wie groß sind die folgenden SGA-Strukturen:

Parameter	Wert
Database Buffer Cache:	
Shared Pool	
Large Pool	
Maximale Größe der SGA	

Beachten Sie: Die Zahlen in der folgenden Lösung können bei Ihnen anders sein.

```
SQL> SELECT name, bytes
  2  FROM v$sgainfo
  3  WHERE name IN ('Buffer Cache Size', 'Shared Pool Size',
  4                    'Large Pool Size', 'Maximum SGA Size');

NAME                      BYTES
-----
Buffer Cache Size          272629760
Shared Pool Size           125829120
Large Pool Size            4194304
Maximum SGA Size           835104768
```

13. Ändern Sie die Sprache Ihrer Session auf Deutsch.

```
SQL> ALTER SESSION
  2  SET NLS_LANGUAGE='GERMAN';
```

14. Erstellen Sie aus Ihrem aktuellen SPFILE die Datei `/home/oracle/initorcl.ora`.

```
SQL> CREATE PFILE = '/home/oracle/initorcl.ora'
  2  FROM SPFILE;
```

15. Erstellen Sie aus den aktuellen Einstellungen Ihrer Instanz ein neues SPFile names /home/oracle/spfileorcl.ora

```
SQL> CREATE SPFILE = '/home/oracle/spfileorcl.ora'  
2  FROM    MEMORY;
```

16. Erstellen Sie eine weitere Kontrolldateikopie /u02/oradata/orcl/control03.ctl und binden Sie sie in Ihre Instanz ein.

```

SQL> shutdown immediate
SQL> startup mount
SQL> col name format a50
SQL> SELECT name
  2  FROM v$controlfile;

NAME
-----
/u01/app/oracle/oradata/orcl/control01.ctl
/u05/fast_recovery_area/orcl/control02.ctl

SQL> shutdown immediate
SQL> host cp /u01/app/oracle/oradata/orcl/control01.ctl
      /u02/oradata/orcl/control03.ctl
SQL> startup nomount
SQL> ALTER SYSTEM
  2  SET control_files = '/u01/app/oracle/oradata/orcl/control01.ctl',
      '/u05/fast_recovery_area/orcl/control02.ctl',
      '/u02/oradata/orcl/control03.ctl'
  3  SCOPE=spfile;
SQL> shutdown immediate
SQL> startup

```

17. Um die weiteren Übungsaufgaben bearbeiten zu können, führen Sie bitte das Skript lab_java.sql als Nutzer SYS aus. Das SQL-Skript befindet sich im Verzeichnis /home/oracle/labs.

```
SQL> @/home/oracle/labs/lab_java.sql
```

18. Ermitteln Sie die Redo Log-Konfiguration Ihres Systems (Gruppen und Member). Die erste Zeile in der folgenden Tabelle ist ein Beispieleintrag.

```

SQL> col member format a50
SQL> SELECT group#, member
  2  FROM v$logfile
  3  ORDER BY 1;

GROUP# MEMBER
-----
1 /u01/app/oracle/oradata/orcl/redo01.log
2 /u01/app/oracle/oradata/orcl/redo02.log
3 /u01/app/oracle/oradata/orcl/redo03a.log
4 /u01/app/oracle/oradata/orcl/redo04a.log
5 /u02/oradata/orcl/redo05b.log
5 /u01/app/oracle/oradata/orcl/redo05a.log

```

19. Passen Sie das Setup Ihrer Redo Log Gruppen so an, dass Sie drei Gruppen mit je 3 Membern haben. Verteilen Sie die Member sinnvoll über die vorhandenen Datenträger /u01 bis /u05 und notieren Sie sich die Änderungen in der obigen Tabelle. Beachten Sie: Die im folgenden gezeigte Lösung muss für Ihr System u. U. angepasst werden!

```

SQL> shutdown immediate
SQL> host mv /u01/app/oracle/oradata/orcl/redo01.log
      /u01/app/oracle/oradata/orcl/redo01a.log
SQL> host mv /u01/app/oracle/oradata/orcl/redo02.log
      /u02/oradata/orcl/redo02a.log
SQL> host mv /u01/app/oracle/oradata/orcl/redo03.log
      /u03/oradata/orcl/redo03a.log
SQL> startup mount
SQL> ALTER DATABASE
  2  RENAME FILE '/u01/app/oracle/oradata/orcl/redo01.log'
  3    TO '/u01/app/oracle/oradata/orcl/redo01a.log';
SQL> ALTER DATABASE
  2  ADD LOGFILE MEMBER '/u02/oradata/orcl/redo01b.log'
  3  TO GROUP 1;
SQL> ALTER DATABASE
  2  ADD LOGFILE MEMBER '/u03/oradata/orcl/redo01c.log'
  3  TO GROUP 1;
SQL> ALTER DATABASE
  2  RENAME FILE '/u01/app/oracle/oradata/orcl/redo02.log'
      TO '/u02/oradata/orcl/redo02a.log';
SQL> ALTER DATABASE
  2  ADD LOGFILE MEMBER '/u03/oradata/orcl/redo02b.log'
  3  TO GROUP 2;
SQL> ALTER DATABASE
  2  ADD LOGFILE MEMBER '/u01/app/oracle/oradata/orcl/redo02c.log'
  3  TO GROUP 2;
SQL> ALTER DATABASE
  2  RENAME FILE '/u01/app/oracle/oradata/orcl/redo03a.log'
      TO '/u03/oradata/orcl/redo03a.log';
SQL> ALTER DATABASE
  2  ADD LOGFILE MEMBER '/u01/app/oracle/oradata/orcl/redo03b.log'
  3  TO GROUP 3;
SQL> ALTER DATABASE
  2  ADD LOGFILE MEMBER '/u02/oradata/orcl/redo03c.log'
  3  TO GROUP 3;

```

20. Prüfen Sie, ob sich Ihre Datenbank im Archive-Log-Modus befindet. Falls nicht wechseln Sie in den Archive-Log-Modus.

```
SQL> SELECT log_mode
  2  FROM v$database;

LOG_MODE
-----
ARCHIVELOG

-- Nur falls notwendig ausfuehren
SQL> shutdown immediate
SQL> startup mount
SQL> ALTER DATABASE ARCHIVELOG;
```

21. Ändern Sie die Parameter der Archivierung wie folgt:

Optional	/u02	/u03	/u04	/u05
Ja	dest 1	-	-	-
Nein	-	dest 2	-	-
Ja	-	-	dest 3	-

- Die Archivierung muss an mindestens zwei Speicherorten erfolgreich sein.
- Das Dateinamensformat der Archive Log Files muss wie folgt aufgebaut sein: archive_%d_%t_%s_%r.arc

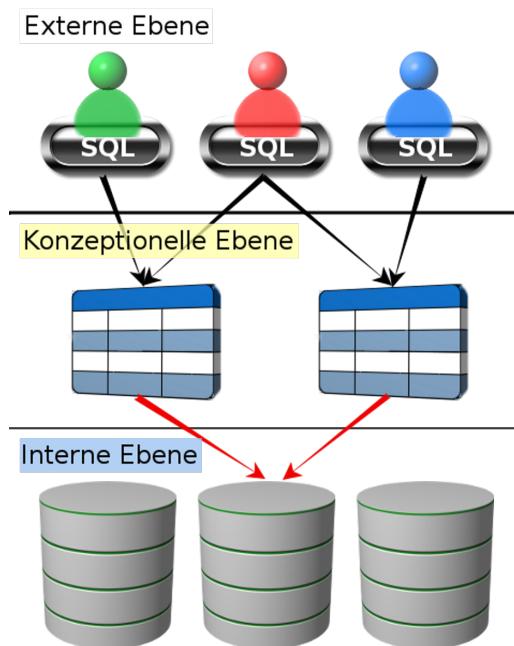
```
SQL> ALTER SYSTEM
  2  SET log_archive_dest_1 = 'LOCATION=/u02/archive OPTIONAL';
SQL> ALTER SYSTEM
  2  SET log_archive_dest_2 = 'LOCATION=/u03/archive MANDATORY';
SQL> ALTER SYSTEM
  2  SET log_archive_dest_3 = 'LOCATION=/u04/archive OPTIONAL';
SQL> ALTER SYSTEM
  2  SET log_archive_min_succeed_dest=2;
SQL> ALTER SYSTEM
  2  SET log_archive_format='archive_%d_%t_%s_%r.arc' SCOPE=spfile;
SQL> shutdown immediate
SQL> startup
```

18 Datenbank Storage-Strukturen verwalten

Inhaltsangabe

Die Hauptaufgabe einer Datenbank ist es, große Datenmengen effizient zu verwalten und diese dem Nutzer über eine einheitliche Schnittstelle zur Verfügung zu stellen. Dieses Ziel wird von Oracle dadurch erreicht, dass der Datenzugriff und die Datenverwaltung in einem mehrschichtigen System gekapselt sind. Als Vorlage für dieses System dient die ANSI-SPARC-Architekton.

Abb. 18.1:
Die
ANSI-SPARC
Architektur



Diese Architektur beschreibt, dass Benutzer mittels eines vereinheitlichten Zugriffsmechanismus (hier die Sprache SQL) auf eine konzeptionelle Ebene zugreifen (hier dargestellt durch Tabellen), wobei die Daten physikalisch in Dateien, auf einem Dateisystem abgelegt werden.

Für den Benutzer bedeutet das, dass er Tabellen sieht, obwohl in den Dateien tatsächlich nur „Bits und Bytes“ existieren, keine Spur von Tabellen. Die Tabellen dienen nur als Darstellungsmittel für den Menschen, zur leichteren Verarbeitung der Daten.

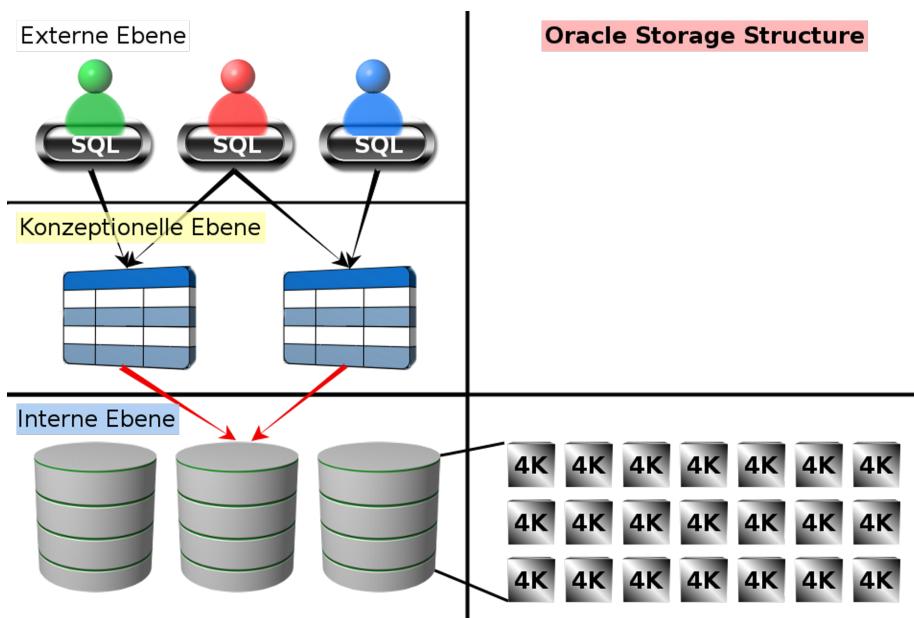
Der Vorteil dieses Modells ist, dass sich beide Ebenen, die konzeptionelle und die interne Ebene, getrennt von einander verändern können, ohne den jeweils anderen zu beeinflussen. Diesen Vorteil, der durch die Gliederung in drei Ebenen zustande kommt, hat Oracle sich zu Nutze gemacht und seine Storage-Struktur ebenfalls auf einer mehrschichtigen Architektur aufgebaut.

Die Oracle-Storage-Struktur gliedert sich in zwei Ebenen, die logische und die physikalische. Während die physikalische mit Dateien, Blöcken und Dateisystemen zu tun hat, beschreibt die logische den internen Aufbau der Datendateien.

Unter der physikalischen Storage-Struktur versteht man den Zusammenhang zwischen Dateisystem, Betriebssystemblöcken und Dateien. Der physikalische Aufbau einer Datei wird durch das Dateisystem gestaltet.

Im Regelfall besteht eine Datei aus mehreren „Betriebssystemblöcken“, der kleinsten Speichereinheit, die ein Dateisystem verwalten kann.

Abb. 18.2:
Die physikalische Storage-Struktur



Die eigentliche Dateiallage geschieht im Dateisystem. Welche Mechanismen, Buffer und Caches hier genutzt werden und welche Effekte dadurch zum Tragen kommen, ist für die Datenbank selbst unwesentlich. Sie kümmert sich nur um die interne Struktur der Datendateien. Der Datenbankadministrator jedoch sollte auch über das benutzte Dateisystem Bescheid wissen, da es durch Fehler in diesem Bereich zu Performanceeinbußen und sogar zu Datenverlust kommen kann!

Die logische Storage-Struktur ist, genau wie die konzeptionelle Ebene des ANSI-SPARC-Modells, eine Abstrahierung der physikalischen Struktur. Das heißt, dass die Datenbank keine BS-Blöcke allokiert, sondern Datenblöcke, Extents und Segmente. Diese drei Größen sind der logischen Struktur zugeordnet, da sie nicht physikalisch auf dem Datenträger existieren, sondern lediglich als Verwaltungseinheiten innerhalb der Oracle-Datenbank.

18.1 Die Komponenten der Oracle-Storage-Struktur

18.1.1 Datenblöcke

Die kleinste Einheit in der Oracle Speicherplatz verwaltet, ist der Oracle-Block oder auch Datenblock genannt. Die Standardgröße dieser Blöcke beträgt 8 K und kann zwischen 2 K und 32 K variieren.



Die Größe eines Oracle-Blocks muss zwingend ein Vielfaches der Größe eines Betriebssystemblocks entsprechen.

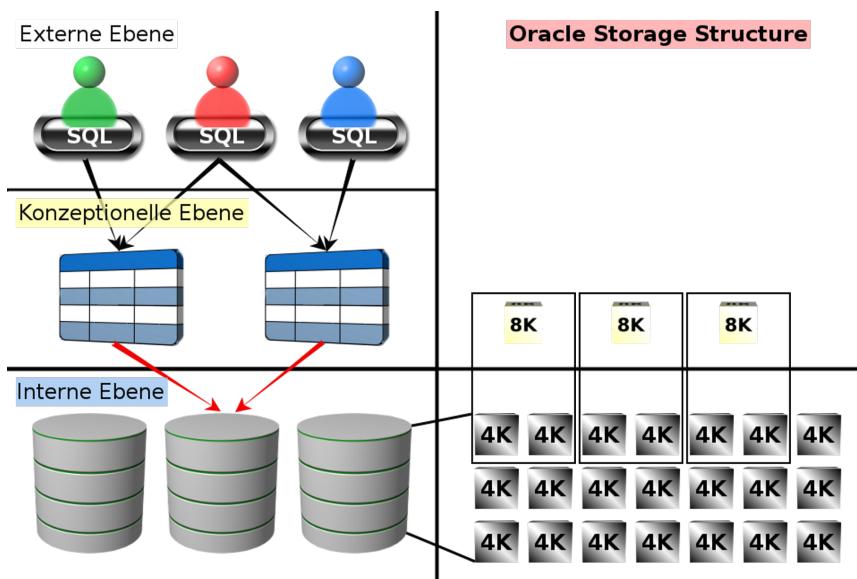
Welche Blockgröße für eine Datenbank benutzt wird, muss bei der Datenbankerstellung mit Hilfe des Parameters `db_block_size` festgelegt werden und kann später nicht mehr verändert werden. Meist werden Werte wie 4 K oder 8 K genutzt.

Listing 18.1: Der Parameter `db_block_size`

```
SQL> col name format a30 SQL> col value format a10
SQL> SELECT name, value
  2  FROM v$system_parameter
  3 WHERE name LIKE 'db_block_size';

NAME                      VALUE
-----                    -----
db_block_size              8192
```

Abb. 18.3:
Die logische
Storage-Struktur:
Oracle-Blöcke



Das Format eines Oracle-Blocks

Das Format eines Datenblocks ist unabhängig von seinem Inhalt. Es gibt vier Bereiche in einem Oracle-Block:

- Block header
- Table directory
- Row directory
- Row data

Die drei Teile Block header, Table directory und Row directory werden unter dem Begriff „Data Block Overhead“ zusammengefasst und stehen nicht für die Speicherung von Nutzdaten zur Verfügung.

Der Block Header enthält allgemeine Informationen über den Block, wie z. B. die Blockadresse und die Art des Blocks. Er ist durchschnittlich 48 bis 107 Byte groß. Das Table Directory ist eine Auflistung aller Tabellen, die Zeilen in diesem Block besitzen und das Row Directory speichert die Positionen aller Tabellenzeilen im Block.



- | | |
|---------------------------------------|------------------------|
| ■ | Block header |
| ■ | Table directory |
| ■ | Row directory |
| ■ | Row data |
| ■ | Free space |

Abb. 18.4:
Format eines
Oracle-Blocks

Eine festgelegte Speichermenge wird in jedem Datenblock als Puffer für **UPDATE**-Operationen freigehalten. Dies ermöglicht das Wachstum bestehender Zeilen. In dem als „Row data“ bezeichneten Teil eines Oracle-Blocks werden die Nutzdaten in Form von Zeilen abgelegt. Eine Tabellenzeile hat ein festgelegtes Format.

Aufbau einer Tabellenzeile

Oracle speichert jede Zeile einer Tabelle, nach Möglichkeit in einem Stück, das als „Row Piece“ bezeichnet wird. Ein Row Piece setzt sich aus verschiedenen Feldern zusammen, wie sie in Abbildung ?? zu sehen sind.

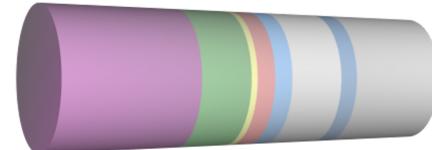
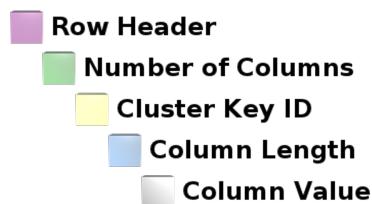


Abb. 18.5:
Format einer
Tabellenzeile



Der Row Header ist der Verwaltungsteil einer Zeile. Hier werden Informationen gespeichert, wie z. B. die Anzahl der Spalten innerhalb der Zeile. Nach dem Header, werden in jeder Zeile deren Länge und die Nutzdaten gespeichert.

Die Anordnung der Spalten einer Tabelle ist für alle Zeilen der Tabelle gleich. Üblicherweise werden die Spalten einer Tabelle in der Reihenfolge, in der sie im `CREATE TABLE`-Statement angegeben wurden, gespeichert. Dies muss jedoch nicht so sein.

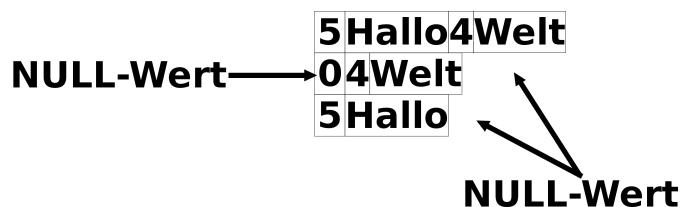
Speicherung von NULL-Werten

Oracle speichert einen NULL-Wert in einer Tabellenspalte, in dem es die Länge der Spalte mit dem Wert 0 angibt (benötigt 1 Byte). Enthält eine Spalte am Ende der Tabelle einen NULL-Wert, wird für diese Spalte kein Wert und keine Länge gespeichert, wodurch Speicherplatz gespart wird.



Aus diesem Grund, sollten Spalten die sehr oft einen NULL-Wert enthalten immer an das Ende einer Tabelle gestellt werden.

Abb. 18.6:
Null-Werte in
Zeilen



Die RowID

Die „RowID“ identifiziert eine Zeile eindeutig. Sie ist die 64-Bit-Kodierung des Zeilenspeicherorts, was bedeutet, dass sie alle Angaben enthält, die Oracle braucht, um eine Tabellenzeile in der Datenbank zu finden. Für die Kodierung werden die Zeichen A-Z, a-z, 0-9, + und / verwendet.

Das folgende Beispiel zeigt den Aufbau einer RowID:

AAAAaoAATAAABrXAAA

- Stelle 1 - 6 **Data object number** AAAAao: Sie identifiziert das Segment (siehe Abschnitt ??), in dem die Zeile gespeichert ist.
- Stelle 7 - 9 **Relativ datafile number** AAT: Dieser Teil der RowID kodiert die Datendatei, in der das Segment gespeichert ist.
- Stelle 10 - 15 **Data block number** AAABrX: Die Data block number ist die Nummer des Datenblocks, der die Zeile enthält.
- Stelle 16 - 18 **Row number** AAA: Die Nummer der Zeile.

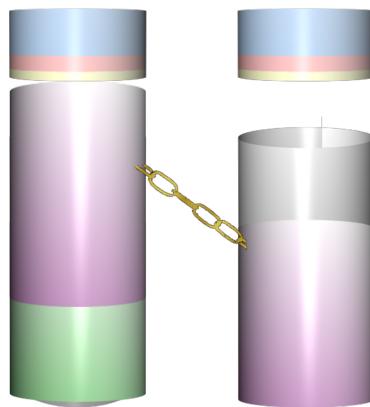
Rowchaining and Migration

Grundsätzlich speichert Oracle alle Tabellenzeilen immer in einem Stück. Es gibt jedoch Situationen, in denen dies nicht geht. Ist eine Zeile z. B. von Anfang an zu groß für einen Oracle-Block oder umfasst sie mehr als 255 Spalten, so muss sie über mehrere Datenblöcke verteilt werden.



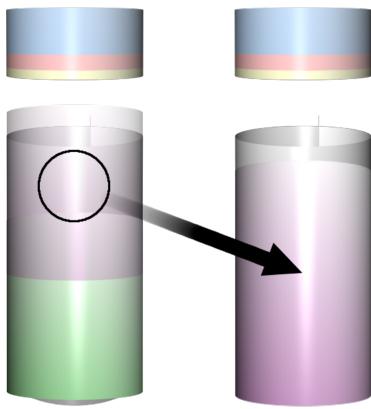
Eine Zeile die aus mehreren verketteten Row Pieces besteht wird als „Chained Row“ bezeichnet.

Abb. 18.7:
Eine verkettete
Tabellenzeile



Ein anderes Problem ist, dass Zeilen wachsen können. Wird eine Tabellenzeile, z. B. aufgrund eines **UPDATE**-Statements länger, kann es vorkommen, dass sie über den Oracle-Block hinaus wächst. In so einem Fall wird die Zeile in einen neuen Block migrieren und an ihrer Originalposition verbleibt nur ein Pointer, ein Wegweiser zum neuen Speicherort. Man spricht von „Row migration“.

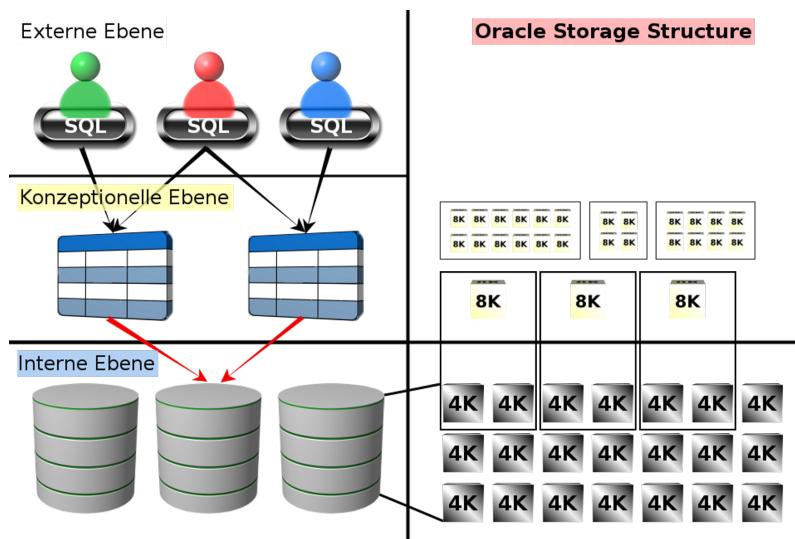
Abb. 18.8:
Eine migrierte
Tabellenzeile



18.1.2 Extents

Extents sind eine weitere logische Speicherverwaltungseinheit. Sie bestehen aus einer ununterbrochenen Kette von Datenblöcken. Wenn z. B. für eine Tabelle Speicher benötigt wird, sind Extents die kleinsten Einheiten die angefordert werden.

Abb. 18.9:
Die logische
Storage-Struktur:
Extents



Wann werden Extents angefordert?

Wird ein neues Objekt, wie z. B. eine Tabelle erstellt, wird ein erstes Extent, das sogenannte *Initial Extent* angefordert. Auch wenn das Objekt zu diesem Zeitpunkt noch keine Daten enthält, wird der Speicherplatz den das Initial Extent belegt als verbraucht angezeigt. Ein weiteres Extent wird erst dann angefordert, wenn das aktuelle Extent zu 100 % gefüllt ist.

Wann werden Extents wieder freigegeben?

Im Allgemeinen werden Extents nur dann freigegeben, wenn das Objekt, dem sie zugewiesen wurden wieder gelöscht wird. Zusätzlich hat der DBA die Möglichkeit, die Datenbank oder Teile von ihr zu Reorganisieren und somit nicht mehr benötigte Extents freizugeben.

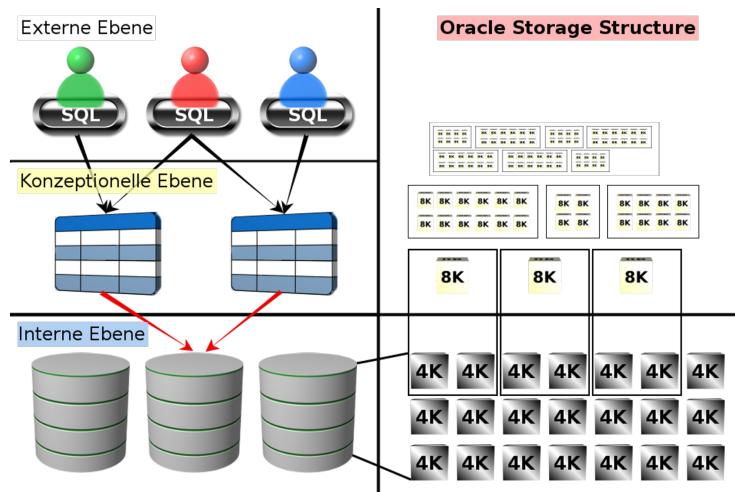
18.1.3 Segmente

Segmente sind das dritte Glied in der Kette der logischen Speicherstrukturen. Sie bestehen aus einer Menge von Extents und enthalten alle Daten eines Datenbankobjekts, wie z. B. einer Tabelle oder eines Indexes. Jedes Segment besteht aus mindestens einem Extent, dass bei der Erstellung des Segments angefordert wird (Initial Extent).

Seit Oracle 11g wird das Verfahren der „Deferred Segment Creation“ angewandt. Dies bedeutet, dass bei der Erstellung eines Datenbankobjektes nur Metadaten in das Data Dictionary eingetragen werden und ein Segment erst beim Einfügen von Daten angelegt wird. So wird die Verschwendungen von Speicherplatz vermieden, falls bei der Installation einer Anwendung viele Tabellen angelegt werden, die u. U. nie gebraucht werden.

Oracle fordert den Speicher für Segmente, Extent für Extent, an. Ist das Segment komplett gefüllt, wird ein weiteres Extent angefordert. Da die Extents für ein Segment nur bei Bedarf angefordert werden, sind diese meist nicht in einer fortlaufenden Reihenfolge.

Abb. 18.10:
Die logische
Storage-Struktur,
Segmente



18.1.4 Tablespaces und Datendateien

Die größte existierende, logische Datenstruktur in Oracle sind die Tablespaces. Ein Tablespace besteht physikalisch aus einer oder mehreren Datendateien.

- Eine Oracle Datenbank besteht aus einem oder mehreren Tablespaces, die die gesamten Daten enthalten.
- Ein Tablespace besteht aus einer oder mehreren Datendateien.

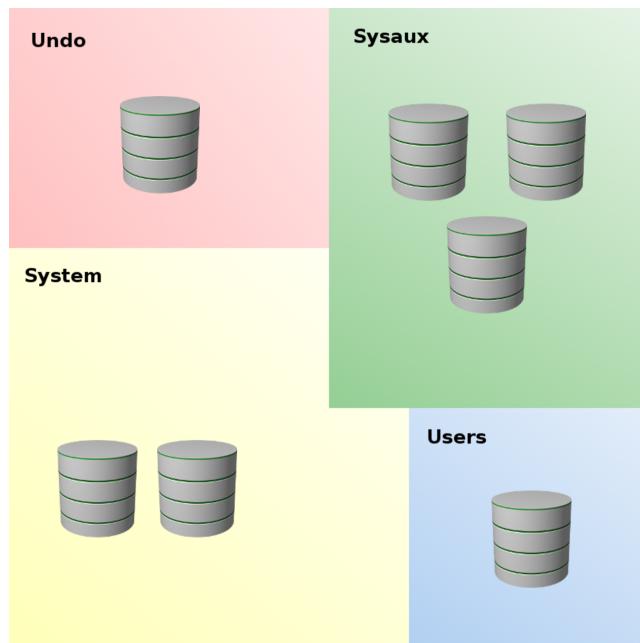


Abb. 18.11:
Tablespaces und
Datendateien

Verwaltung der Extents im Tablespace

Seit Oracle 9i gibt es zwei Möglichkeiten, wie die Verwaltung der freien Extents eines Tablespaces geschehen kann: Im Data Dictionary oder im Tablespace selbst.

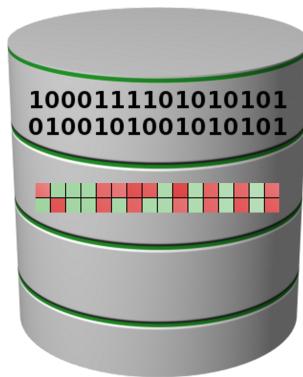
- **Locally Managed Tablespaces**: Bei Locally Managed Tablespaces geschieht die Verwaltung der Extents direkt im Tablespace.
- **Dictionary Managed Tablespaces**: Dictionary Managed Tablespaces verwalten ihre Extents im Data Dictionary

Wurde ein Tablespace als Dictionary Managed Tablespace erstellt, kann später die Verwaltung auf Locally Managed umgestellt werden. Umgekehrt jedoch nicht. Ist der SYSTEM-Tablespace ein Locally

Managed Tablespace, können keine Dictionary Managed Tablespaces in der Datenbank erstellt werden. Im Normalfall werden alle Tablespaces als Locally Managed Tablespaces angelegt (erst seit Oracle 10g).

Locally Managed Tablespaces verwalten, im Gegensatz zu Dictionary Managed Tablespaces, ihren Speicherplatz selbstständig. Dies erfolgt mit Hilfe von Bitmaps. Jede Datendatei eines Tablespaces enthält eine eigene Bitmap, um den Füllstatus ihrer Extents zu speichern.

Abb. 18.12:
Verwaltung freier
Extents mittels
Bitmap



Da die Nutzung von Dictionary Managed Tablespaces nicht mehr zeitgemäß ist und von Oracle auch nicht mehr empfohlen wird, wird diese Art der Speicherverwaltung hier nicht näher erläutert.

Die wichtigsten Tablespaces im Überblick

Bei der Erzeugung einer Datenbank werden einige Tablespaces automatisch erstellt. Im Folgenden werden die vier Standard tablespaces einer Oracle-Datenbank erläutert.

Der System-Tablespace

Jede Oracle Datenbank enthält einen Tablespace mit dem Namen **SYSTEM**. Dieser wird bei der Datenbankerstellung erzeugt und muss für den Betrieb der Datenbank immer verfügbar sein. Er enthält das Herzstück einer Oracle Datenbank, das Data Dictionary.

Der Sysaux-Tablespace

Der SYSAUX-Tablespace stellt eine Erweiterung zum SYSTEM-Tablespace dar. Er enthält wichtige Komponenten für einige Datenbankanwendungen wie z. B. den Enterprise Manager. Auch dieser Tablespace wird automatisch bei der Datenbankerstellung erzeugt und kann im laufenden Betrieb nicht gelöscht werden. Ist er nicht verfügbar, kann es zu unerwarteten Datenbankfehlern kommen.

Undo-Tablespace

Undo-Tablespace sind spezielle Tablespaces, die nur für die Aufnahme von Undo-Informationen genutzt werden. Es können keinerlei Segmente, wie z. B. Tabellen oder Cluster in einem Undo-Tablespace erstellt werden.

Temporäre Tablespaces

Temporäre Tablespaces enthalten temporäre Segmente und werden für große Sortieroperationen benötigt, die in der PGA eines Nutzers nicht durchgeführt werden können. Auch in dieser Art von Tablespace können manuell keine Objekte angelegt werden.

18.2 Automatic Segment Space Management (ASSM)

Die Verwaltung des Speicherplatzes umfasst nicht nur die Verwaltung der freien Extents, sondern auch die der freien Oracle-Blöcke in den einzelnen Segmenten. Hierfür gibt es zwei Verfahren. Das eine arbeitet mit „Freelists“ und das andere mit Bitmaps. Beim Erstellen eines Tablespaces kann die Art der Speicherverwaltung gewählt werden.

- **Automatic Segment Space Management:** Der Speicherplatz in den Segmenten wird durch eine Bitmap verwaltet.
- **Manual Segment Space Management:** Der Speicherplatz in den Segmenten wird durch Freelists verwaltet.

Die Standardvorgabe ist immer das Automatic Segment Space Management. Oracle empfiehlt bei dieser Einstellung zu bleiben.



18.2.1 Die Auxiliary Map

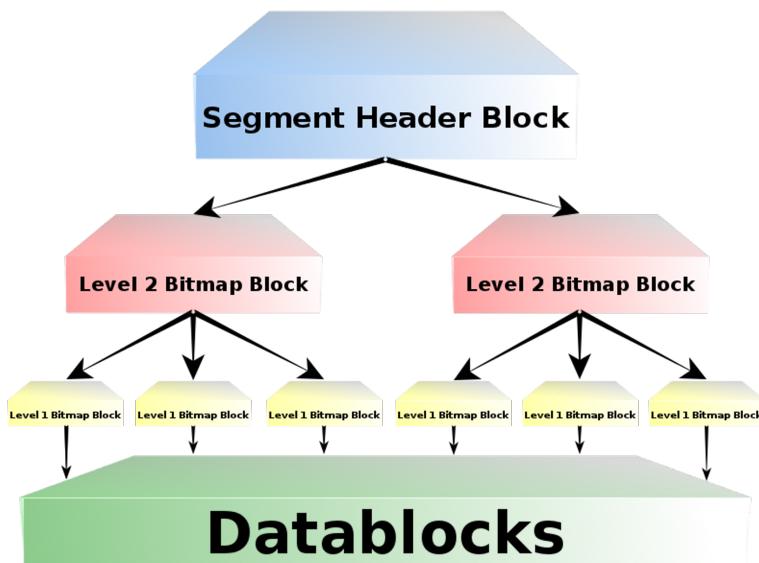
Beim Automatic Segment Space Management werden die Blöcke eines Segments unter zu Hilfenahme einer Bitmap, der „Auxiliary Map“ verwaltet. Sie untergliedert sich in bis zu vier Ebenen. Abbildung ?? zeigt den Aufbau einer dreistufigen Auxiliary Map.

Level 1 Bitmap-Blöcke

Die Verwaltung des freien Speichers in den Oracle-Blöcken, geschieht in den Level 1 Bitmaps (L1B). Jeder L1B verwaltet mehrere Oracle-Blöcke (zwischen 16 und 1024 Blöcke). Für jeden Oracle-Block wird sein Füllgrad im L1B festgehalten.

Die folgende Abbildung zeigt einen gekürzten Ausschnitt aus einem L1B.

Abb. 18.13:
Architektur des
Automatic
Segment Space
Management



Listing 18.2: Der Inhalt eines Level 1 Bitmap-Blocks

```

Freeness Status: nf1 0      nf2 1      nf3 0      nf4 3
Extent Map Block Offset: 4294967295
First free datablock : 2
-----
DBA Ranges :
-----
0x01c00029  Length: 8      Offset : 0
0x01c00031  Length: 8      Offset : 8

0:Metadata   1:FULL     2:26-50% free   3:FULL
4:FULL      5:FULL     6:FULL     7:FULL
8:FULL      9:FULL     10:76-100% free   11:76-100% free
12:FULL     13:76-100% free   14:FULL     15:FULL
  
```

Die Zeile „Freeness Status“ gibt an, wieviele Blöcke mit welchem Belegungsgrad existieren. Dabei gilt:

Tabelle 18.1: Belegungsgrad von Oracle-Blöcken

Freeness Status	Belegungsgrad
nf1	Belegungsgrad zwischen 0 % und 25 %
nf2	Belegungsgrad zwischen 26 % und 50 %
nf3	Belegungsgrad zwischen 51 % und 75 %
nf4	Belegungsgrad zwischen 76 % und 100 %

Somit existieren in Abbildung ?? 1 Block mit einem Füllstand von 26 % bis 50 % und 3 Blöcke mit einem Füllgrad zwischen 76 % und 100 %. Blöcke die noch leer oder zu 100 % gefüllt sind, werden in dieser Zeile nicht berücksichtigt.

Die Zeile „First free block“ gibt, bezogen auf das Segment, die Blocknummer des ersten freien Blocks an.

In den letzten vier Zeilen des Blockabbildes sind die exakten Füllgrade der einzelnen Blöcke zu sehen.

Level 2 Bitmap-Blöcke

Die Level 2 Bitmap-Blöcke stehen eine Ebene über den L1B in der hierarchischen Ordnung der Auxiliary Map. Sie enthalten die Adressen der Datenblöcke, in denen sich die L1B befinden, die sie verwalten.

Level 3 Bitmap Blöcke

Falls die Auxiliary Map stark anwachsen sollte, kann Oracle eine weitere Ebene in die Hierarchie einfügen, die Level 3 Bitmap-Blöcke. Sie stehen eine Stufe über den Level 2 Bitmap-Blöcken und enthalten demzufolge eine Liste der Blockadressen, in denen sich die Level 2 Bitmaps befinden.

18.2.2 Freie und Belegte Blöcke - PCTFREE

Ob ein Oracle-Block als frei oder als belegt gilt, hängt stark davon ab wie viel Speicher als „Puffer für wachsende Zeilen“ definiert wurde. Beim Erstellen einer Tabelle kann diese Puffergröße mittels des Storageparameters `pctfree` definiert werden. Wie die Abkürzung „PCT“ in seinem Namen sagt, nimmt er einen Prozentwert als Angabe entgegen. Beispiel ?? zeigt seine Anwendung.

Listing 18.3: Der Storageparameter `pctfree`

```
SQL> CREATE TABLE bank.bankfiliale
  2  (
  3    bankfiliale_id          NUMBER ,
  4    strasse                  VARCHAR2(50 CHAR) ,
  5    hausnummer               VARCHAR2(15 CHAR) ,
  6    plz                      CHAR(5 CHAR) ,
  7    ort                      VARCHAR2(20 CHAR)
  8  )
  9  PCTFREE 10;
```

`PCTFREE` wird in Beispiel ?? auf 10 % gesetzt, was bedeutet, dass 10 % des Speichers in jedem Datenblock als Puffer für wachsende Zeilen frei bleiben.

PCTFREE richtig setzen

Welcher Wert für **PCTFREE** gewählt werden sollte, hängt davon ab, ob häufige Änderungen an einer Tabelle vorgenommen werden (viele **UPDATE**-Statements) oder ob die Tabelle lediglich mit neuen Daten befüllt wird (viele **INSERT**-Statements). Was aber passiert, wenn **PCTFREE** falsch gesetzt wurde:

Tabelle 18.2: PCTFREE und seine Auswirkungen

	PCTFREE zu klein	PCTFREE zu groß
Viele Updates	Die Tabellenzeilen haben nicht genug Platz zum Wachsen und migrieren (Migrated Rows!)	Es wird Speicherplatz verschwendet, da mehr Puffer zur Verfügung gestellt wird, als nötig.
Viele Insert	Keine negativen Auswirkungen!	Es wird Speicherplatz verschwendet, da die Zeilen kaum wachsen.

Blöcke werden wieder frei, wenn . . .

Der Parameter **PCTFREE** ist für die Tabelle **BANKFILIALE** auf den Wert 10 gesetzt. Daraus folgt, dass ein Datenblock als belegt gilt, wenn sein Füllgrad die 90 % Marke überschreitet. Diese Marke liegt in der Sektion von 76 % bis 100 %. Der Block wird wieder frei, wenn er die nächste untere Sektionsgrenze unterschreitet. Sein Füllgrad muss somit unter die Marke von 51 % fallen, um diese Bedingung zu erfüllen, da die nächste untere Sektion von 51 % bis 75 % reicht.

Mit dieser Vorgehensweise soll verhindert werden, dass der Zustand eines Blockes in kurzen Intervallen von belegt auf frei und wieder zurück wechselt.

18.2.3 Vor- und Nachteile von ASSM

Die Verwendung von Locally Managed Tablespaces mit Automatic Segment Space Management hat folgende Vorteile gegenüber der Verwendung von Dictionary Managed Tablespaces oder Locally Managed Tablespaces mit Manual Segment Space Management:

- **Rekursive Speicherverwaltungsoperationen werden vermieden.**

In Dictionary Managed Tablespaces werden die Datenblöcke in Tabellen im Data Dictionary verwaltet. Dabei kann es vorkommen, dass das Anfordern eines Extents für eine Tabelle mit Nutzdaten, die Vergrößerung einer der beiden Tabellen UET\$ oder FET\$, die für die Verwaltung der Extents zuständig sind, nach sich zieht.

- Änderungen an der Extent-Bitmap erzeugen keine Undoinformationen.

- Der Füllgrad der einzelnen Blöcke ist schneller verfügbar.

Bei Automatic Segment Space Management kann der Füllgrad eines Block direkt aus der L1B entnommen werden, während beim Manual Segment Space Management der Füllgrad dem Blockheader des betreffenden Blockes entnommen werden muss.

- Die freien Blöcke müssen nicht mehr in einer bestimmten Reihenfolge genutzt werden.

18.3 Tablespaces verwalten

18.3.1 Das CREATE TABLESPACE-Kommando

Anlegen eines Tablespaces mit Standardwerten

Tablespaces werden mit dem Kommando `CREATE TABLESPACE` angelegt. In seiner einfachsten Form benötigt dieses Kommando nur drei Angaben:

- den Namen des Tablespaces in der `TABLESPACE`-Klausel,
- einen Pfad und einen Dateinamen für den Tablespace in der `DATAFILE`-Klausel
- und eine Größe für die Datendatei in der `SIZE`-Klausel.

Listing 18.4: Das `CREATE TABLESPACE`-Kommando

```
SQL> CREATE TABLESPACE bank
  2  DATAFILE '/u02/oradata/orcl/bank.dbf' SIZE 50M;
```



In der `DATAFILE`-Klausel können eine oder mehrere Datendateien angegeben werden, aus denen der Tablespace bestehen wird. Mit dem Schlüsselwort `SIZE` wird die Größe jeder einzelnen Datendatei festgelegt.

Listing 18.5: Ein Tablespace mit mehreren Datendateien

```
SQL> CREATE TABLESPACE bank
  2  DATAFILE '/u02/oradata/orcl/bank01.dbf' SIZE 50M,
  3        '/u03/oradata/orcl/bank02.dbf' SIZE 200M;
```

Gestaltung der Extents beeinflussen

Seit Oracle 10g ist jeder Tablespace von Haus aus lokal verwaltet. Locally Managed Tablespaces können ihre Extents auf zwei unterschiedliche Arten gestalten:

- **wachsend**: Die Extents im Tablespace wachsen automatisch. Das erste hat eine Größe von 64 K und das zweite Extent ebenso. Ab dem dritten Extent steuert Oracle das Wachstum. Je mehr Extents angefordert werden, desto größer werden diese. Sie können Größen von 1 M, 8 M und sogar 64 M annehmen.
- **gleichförmig**: Gleichförmig bedeutet, dass alle Extents mit einer einheitlichen Größe angelegt werden. `UNIFORM SIZE 1M` ist der Standard, falls nichts näher definiert wird.

Für beide Verfahren muss das `CREATE TABLESPACE`-Kommando um die `EXTENT MANAGEMENT LOCAL`-Klausel erweitert werden. Sollen automatisch wachsende Extents erzeugt werden, wird zusätzlich das Schlüsselwort `AUTOALLOCATE` benötigt.

Listing 18.6: Ein Tablespace mit automatisch wachsenden Extents

```
SQL> CREATE TABLESPACE bank
  2  DATAFILE '/u02/oradata/orcl/bank01.dbf' SIZE 50M,
  3          '/u03/oradata/orcl/bank02.dbf' SIZE 200M
  4  EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

Die Angabe von `EXTENT MANAGEMENT LOCAL AUTOALLOCATE` ist immer dann vorteilhaft, wenn in einem Tablespace Objekte mit sehr unterschiedlichem Volumen gespeichert werden. Durch das Anwachsen der Extents wird der Verwaltungsaufwand für das Anfordern verringert, da mit zunehmender Extentgröße weniger Extents angefordert werden müssen.



`EXTENT MANAGEMENT LOCAL AUTOALLOCATE` ist der Standard!

Uniform-Sized-Extents sind immer dann die bessere Wahl, wenn die Größe der Daten, die in die Tabelle eingefügt werden, in etwa vorausgesagt werden kann. Dies kann z. B. dann der Fall sein, wenn PDF oder DOC-Dateien in den Tabellen gespeichert werden sollen.

Listing 18.7: Uniform-Sized-Extents

```
SQL> CREATE TABLESPACE bank
  2  DATAFILE '/u02/oradata/orcl/bank01.dbf' SIZE 50M,
  3          '/u03/oradata/orcl/bank02.dbf' SIZE 200M
  4  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

Beispiel ?? zeigt wie der Tablespace BANK mit gleichförmigen 128 K Extents angelegt wird.



Die Standardgröße für Uniform-Sized-Extents ist 1M!

Bigfile Tablespaces

Bigfile Tablespaces haben die Besonderheit, dass sie nur aus einer einzigen Datendatei bestehen. Somit entfällt der dreistellige Anteil der Relative File Number aus der RowID, wodurch die Datendatei bis zu 2^{32} -Datenblöcke (ca. 4 Milliarden) haben kann. Bei einer maximalen Blockgröße von 32 K ermöglicht dies eine Kapazität von 128 TB pro Datendatei, bei einer Blockgröße von 8 K, sind es 32 TB pro Datendatei.

Da eine Oracle Datenbank bis zu 65.536 Datendateien haben darf, wird eine Datenbankgröße von bis zu 8 Exabyte möglich.

Im Vergleich dazu, kann ein Smallfile Tablespace nur bis zu 2^{22} -Datenblöcke enthalten, was bei einer Blockgröße von 32 K nur 128 GB gesamtgröße ausmacht. Die Gesamtgröße der Datenbank, bei 65.536 Datendateien, wäre dann nur 8 Petabyte, statt 8 Exabyte.



Seit dem Bigfile Tablespaces existieren, werden „normale Tablespaces“ als Smallfile Tablespaces bezeichnet.



- [REFRN0042]
- [doctbtthtm]

Beispiel ?? zeigt, wie ein Bigfile Tablespace angelegt wird.

Listing 18.8: Einen Bigfile Tablespace anlegen

```
SQL> CREATE BIGFILE TABLESPACE big_mac_ts
  2  DATAFILE '/u02/oradata/orcl/big_mac_ts.dbf' SIZE 100G;
```

Ob es sich bei einem Tablespace um einen Bigfile Tablespace handelt, kann mittels der Spalte BIGFILE, der View DBA_TABLESPACES ermittelt werden.

Listing 18.9: Die View DBA_TABLESPACES

```
SQL> SELECT tablespace_name, bigfile
  2  FROM    dba_tablespaces;
```

TABLESPACE_NAME	BIG
SYSTEM	NO
SYSAUX	NO
UNDOTBS1	NO
TEMP	NO
EXAMPLE	NO
BIG_MAC_TS	YES



Ein Bigfile Tablespace wird automatisch als Locally Managed Tablespace mit Automatic Segment Space Management angelegt. Durch Dateigrößenbeschränkungen in Dateisystemen kann die Erstellung von Bigfile Tablespaces auf eine bestimmte Größe limitiert sein.

Temporäre Tablespaces

Temporäre Tablespaces werden immer dann benötigt, wenn Sortier- oder Hashingoperationen nicht in der PGA eines Serverprozesses durchgeführt werden können, weil mehr Speicherplatz benötigt wird, als zur Verfügung steht. Ihre Funktion ist vergleichbar mit der der Windows Auslagerungsdatei.

Durch Hinzufügen des Schlüsselwortes `TEMPORARY` zum `CREATE TABLESPACE`-Statement wird ein temporärer Tablespace angelegt. Die `DATAFILE`-Klausel wird bei temporären Tablespace durch die `TEMPFILE`-Klausel ersetzt.

Listing 18.10: Einen temporären Tablespace anlegen

```
SQL> CREATE TEMPORARY TABLESPACE bank_temp
  2  TEMPFILE '/u02/oradata/orcl/bank_temp_01.dbf' SIZE 20 M;
```



Temporäre Tablespace können nur mit Uniform-Sized-Extents angelegt werden. Die `EXTENT MANAGEMENT LOCAL UNIFORM SIZE`-Klausel kann wahlweise dazu benutzt werden, um die Größe der Extents zu beeinflussen. Das Schlüsselwort `AUTOALLOCATE` darf bei temporären Tablespace nicht verwendet werden.

Ein temporärer Tablespace kann auch als Big-File Tablespace angelegt werden.

Listing 18.11: Einen temporären Big-File Tablespace anlegen

```
SQL> CREATE BIGFILE TEMPORARY TABLESPACE bank_temp
  2  TEMPFILE '/u02/oradata/orcl/bank_temp.dbf' SIZE 20G;
```

18.3.2 Das ALTER TABLESPACE-Kommando

Mit Hilfe des `ALTER TABLESPACE`-Kommandos ist es möglich, die Definition eines Tablespaces zu verändern. Im Folgenden werden einige Anwendungsfälle für dieses Kommando gezeigt.

Tablespaces On- und Offline setzen

Ein DBA kann alle Tablespaces, mit Ausnahme des SYSTEM-Tablespace, online (verfügbar) und offline (nicht verfügbar) setzen. Dies kann im laufenden Betrieb geschehen und ist für Wartungstätigkeiten, wie z. B. das Recovery eines einzelnen Tablespaces notwendig.

Wird ein Tablespace offline gesetzt, verbietet Oracle den Zugriff auf alle Objekte in diesem Tablespace. Laufende Transaktionen werden durch das Offlinesetzen nicht automatisch beendet, ein abschließen der Transaktion mit `COMMIT` ist nach wie vor möglich.

In manchen Fällen kann es vorkommen, dass Oracle einen Tablespace automatisch offline setzt, z. B. wenn der Tablespace durch einen Medienfehler beschädigt wurde.

Listing 18.12: Einen Tablespace offline setzen

```
SQL> ALTER TABLESPACE bank OFFLINE;
```

Ein Tablespace kann auf drei verschiedene Arten offline gesetzt werden:

- **OFFLINE NORMAL:** Der Tablespace wird so offline gesetzt, dass kein Recovery benötigt wird, wenn der Tablespace wieder online gesetzt werden soll. Dies funktioniert, jedoch nur dann, wenn alle Datendateien des Tablespaces fehlerfrei sind.
- **OFFLINE TEMPORARY:** Mit diesem Modus können alle noch verbliebenen, fehlerfreien Datendateien eines Tablespaces konsistent offline gesetzt werden. Fehlerhafte Datendateien werden ignoriert.
- **OFFLINE IMMEDIATE:** Der Tablespace wird sofort, ohne Checkpoint offline gesetzt. Beim Onlinesetzen wird in jedem Falle Recovery benötigt.

Beispiel ?? verdeutlicht den Unterschied zwischen **OFFLINE NORMAL** und **OFFLINE TEMPORARY**.

Listing 18.13: Der Unterschied zwischen NORMAL und TEMPORARY

```
SQL> CREATE TABLESPACE defekt_ts
  2  DATAFILE '/u02/oradata/orcl/defekt_ts_01.dbf' SIZE 5M,
  3          '/u03/oradata/orcl/defekt_ts_02.dbf' SIZE 10M;

-- Die Datei defekt_ts_01.dbf wird zerstoert!

SQL> SELECT file_name, online_status
  2  FROM dba_data_files
  3  WHERE file_name LIKE 'DEFEKT_TS_01.DBF';

FILE_NAME          ONLINE_STATUS
-----  -----
/u02/oradata/orcl/defekt_ts_01.dbf    RECOVER
```

```
SQL> ALTER TABLESPACE defekt_ts OFFLINE NORMAL;

ERROR at line 1:
ORA-01191: file 6 is already offline - cannot do a normal offline
ORA-01110: data file 6: '/u02/defekt_ts_01.dbf'

SQL> ALTER TABLESPACE defekt_ts OFFLINE TEMPORARY;

Tablespace altered.
```

Soll der Tablespace wieder Online gehen, wird das Schlüsselwort **ONLINE** verwendet.

Listing 18.14: Einen Tablespace online setzen

```
SQL> ALTER TABLESPACE defekt_ts ONLINE;
```

Read Only Tablespaces

Wird ein Tablespace mit der **READ ONLY**-Klausel schreibgeschützt, hat das zwei Effekte:

- Unbeabsichtigte Änderungen an den Daten werden verhindert.
- Der Oracle Recovery Manager erkennt Read Only Tablespaces. Nachdem ein solcher Tablespace einmal gesichert wurde, wird er bei allen folgenden Backups ausgelassen.

Listing 18.15: Einen Tablespace Read Only setzen

```
SQL> ALTER TABLESPACE bank READ ONLY;
```

Um den Schreibzugriff auf den Tablespace wieder zu ermöglichen, wird die Klausel **READ ONLY** durch die Klausel **READ WRITE** ersetzt.

Listing 18.16: Einen Tablespace Read Write setzen

```
SQL> ALTER TABLESPACE bank READ WRITE;
```

Zur Durchführung dieser Tätigkeiten, muss der Nutzer eines der beiden Systemprivilegien **ALTER TABLESPACE** oder **MANAGE TABLESPACE** besitzen.

Tablespaces umbenennen

Mit der **RENAME TO**-Klausel, des **ALTER TABLESPACE**-Kommandos, können Tablespaces umbenannt werden. Die Umbenennung ist sowohl für permanente, als auch für temporäre Tablespace möglich.

Listing 18.17: Einen Tablespace umbenennen

```
SQL> ALTER TABLESPACE bank RENAME TO bank_ts;
```



Soll ein UNDO-Tablespace umbenannt werden, ist es unbedingt notwendig, dass zum Betrieb der Instanz ein SPFile verwendet wird.

Big-File Tablespaces vergrößern

Ein Big-File Tablespace kann mit Hilfe der `RESIZE`-Klausel der `ALTER TABLESPACE`-Anweisung vergrößert werden. Da er nur eine Datendatei hat, wirkt sich die Änderung direkt auf diese aus.

Listing 18.18: Einen Big-File Tablespace vergrößern

```
SQL> ALTER TABLESPACE big_mac_ts RESIZE 200G;
```

18.3.3 Das `DROP TABLESPACE`-Kommando

Ein Tablespace kann mitsamt seinem Inhalt gelöscht werden, wenn er nicht mehr benötigt wird. Um einen Tablespace löschen zu können, benötigt der Nutzer das `DROP TABLESPACE`-System Privileg. Vor dem Löschen sollte unbedingt ein Backup der Datenbank gemacht werden, so dass der Tablespace im Zweifelsfalle wieder hergestellt werden kann.

Ein leerer Tablespace kann einfach mit dem `DROP TABLESPACE`-Kommando gelöscht werden. Um Probleme oder Verzögerungen beim Löschen zu vermeiden, sollte der Tablespace immer zuerst offline gesetzt werden.

Listing 18.19: Einen leeren Tablespace löschen

```
SQL> ALTER TABLESPACE big_mac_ts OFFLINE IMMEDIATE;
SQL> DROP TABLESPACE big_mac_ts;
```

Enthält der Tablespace Segmente, muss an das `DROP TABLESPACE`-Kommando die Klausel `INCLUDING CONTENTS` angehängt werden.

Listing 18.20: Einen Tablespace mit Inhalt löschen

```
SQL> CREATE TABLE empty_table
  2  (
  3    empty_id NUMBER
  4  )
 5 TABLESPACE big_mac_ts;
```

```
SQL> DROP TABLESPACE big_mac_ts;

ERROR at line 1:
ORA-01549: tablespace not empty, use INCLUDING CONTENTS option

SQL> DROP TABLESPACE big_mac_ts INCLUDING CONTENTS;

Tablespace dropped.
```



Beide Male werden die Datendateien der betreffenden Tablespaces nicht mit gelöscht. Diese können entweder manuell mit Betriebssystemmitteln gelöscht werden oder es kann die `AND DATAFILES`-Klausel des `DROP TABLESPACE`-Kommandos verwendet werden.

Listing 18.21: Einen Tablespace mit Inhalt und Datendateien löschen

```
SQL> DROP TABLESPACE big_mac_ts
  2 INCLUDING CONTENTS AND DATAFILES;
```

Enthalten die Segmente im Tablespace Fremdschlüssel-Constraints, die auf Segmente in einem anderen Tablespace verweisen, muss zusätzlich die `CASCADE CONSTRAINTS`-Klausel angehängt werden.

Listing 18.22: Einen Tablespace mit Inhalt, Datendateien und Constraints löschen

```
SQL> DROP TABLESPACE big_mac_ts
  2 INCLUDING CONTENTS AND DATAFILES
  3 CASCADE CONSTRAINTS;
```

18.4 Datendateien verwalten

18.4.1 Hinzufügen einer Datendatei zu einem Tablespace

Datendateien werden mit Hilfe der `ADD DATAFILE`-Klausel an einen Tablespace angefügt.

Listing 18.23: Hinzufügen einer Datendatei

```
SQL> ALTER TABLESPACE bank
  2 ADD DATAFILE '/u03/oradata/orcl/bank02.dbf' SIZE 100M;
```



Bei einem Bigfile Tablespace kann keine weitere Datendatei hinzugefügt werden.

18.4.2 Das Wachstum von Datendateien kontrollieren

Automatisches Wachstum erlauben

Eine Datendatei kann so erstellt werden, bzw. ihr Verhalten kann so geändert werden, dass sie bei Bedarf automatisch wächst. Dies hat folgende Vorteile:

- Der Administrator wird entlastet, da er nicht sofort eingreifen muss, wenn die Datendatei zu klein ist.
- Anwendungen bleiben nicht im Betrieb stehen, weil zu wenig Platz in einem Tablespace zur Verfügung steht.

Ob für eine Datendatei das automatische Wachstum bereits aktiviert wurde, kann mit Hilfe der View DBA_DATA_FILES ermittelt werden. Das folgende Beispiel zeigt die Erstellung eines Tablespace mit einer Datendatei, für die das automatische Wachstum aktiviert wird.

Listing 18.24: Erstellen eines Tablespace mit automatisch wachsender Datendatei

```
SQL> CREATE TABLESPACE auto_growing_ts
  2  DATAFILE '/u02/oradata/orcl/auto_growing_ts01.dbf' SIZE 100M
  3  AUTOEXTEND ON MAXSIZE 250M;
```

In Beispiel ?? wird das Wachstum der Datendatei auf 250 Megabyte begrenzt. Durch die Angabe von **MAXSIZE UNLIMITED** ist es möglich, ein unbegrenztes Wachstum der Datendatei einzurichten.

Auch im Nachhinein kann einem Tablespace eine Datendatei hinzugefügt werden, die automatisch wächst.

Listing 18.25: Hinzufügen einer Datendatei mit automatischem Wachstum

```
SQL> ALTER TABLESPACE auto_growing_ts
  2  ADD DATAFILE '/u03/oradata/orcl/auto_growing_ts02.dbf' SIZE 100M
  3  AUTOEXTEND ON MAXSIZE 250M;
```

Um das automatische Wachstum für eine Datendatei abzuschalten, muss die betroffene Datendatei mit dem **ALTER DATABASE**-Kommando angefasst werden.

Listing 18.26: Automatisches Wachstum für eine Datendatei abschalten

```
SQL> ALTER DATABASE
  2  DATAFILE '/u02/oradata/orcl/auto_growing_ts02.dbf'
  3  AUTOEXTEND OFF;
```

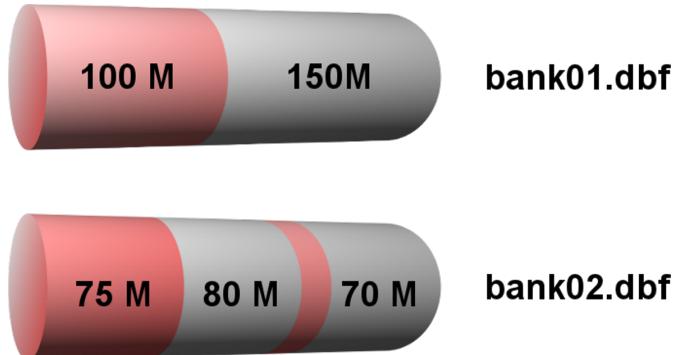
Eine Datendatei manuell vergrößern

Eine Datendatei kann mit Hilfe des **ALTER DATABASE**-Kommandos manuell vergrößert werden. Dadurch wird ermöglicht, einen Tablespace zu vergrößern, ohne neue Datendateien zur Datenbank hinzuzufügen. Dies ist dann vorteilhaft, wenn die Maximalanzahl an Datendateien für eine Datenbank fast erreicht ist. Das folgende Beispiel zeigt, wie die 100 M große Datendatei BANK02.DBF auf 350 M vergrößert wird.

Listing 18.27: Eine Datendatei manuell vergrößern

```
SQL> ALTER DATABASE
  2  DATAFILE '/u03/oradata/orcl/bank02.dbf'
  3  RESIZE 350M;
```

Mit Hilfe des gleichen Statements kann eine Datendatei auch wieder verkleinert werden. Bedingung dafür ist, dass der hintere Bereich der Datendatei leer ist. Die folgende Grafik veranschaulicht dies.



Die Abbildung ?? zeigt zwei Datendateien. Die Datei BANK01.DBF kann um 150 M reduziert werden, aber BANK02.DBF lediglich um 70 M, da belegte Datenblöcke in ihr eine weitere Verkleinerung verhindern.

18.4.3 Datendateien umbenennen und verschieben

Datendateien können umbenannt werden, um ihren Namen oder ihren Speicherort zu ändern.

Wird eine Datendatei in Oracle umbenannt, werden nur ihre Einträge in der Kontrolldatei und im Data Dictionary geändert. Auf dem Datenträger geschieht keine Veränderung. Diese muss manuell nachgeholt werden.

Um eine Datendatei umzubenennen gehen Sie wie folgt vor:

1. Den betreffenden Tablespace Offline setzen.

Listing 18.28: Tablespace Offline setzen

```
SQL> ALTER TABLESPACE bank OFFLINE NORMAL;
```

2. Umbenennen und evtl. auch verschieben der Datendatei auf dem Datenträger.

Listing 18.29: Tablespace Offline setzen

```
SQL> host mv /u02/oradata/orcl/bank02.dbf /u03/oradata/orcl/bank02.dbf
```

3. Die Datendatei in der Datenbank umbenennen.

Listing 18.30: Datendatei umbenennen

```
SQL> ALTER TABLESPACE bank
  2  RENAME DATAFILE '/u02/oradata/orcl/bank02.dbf'
  3  TO                 '/u03/oradata/orcl/bank02.dbf';
```

18.4.4 Datendateien löschen

Datendateien können unter der Voraussetzung gelöscht werden, dass sie noch unbenutzt sind. Hierfür existiert die `DROP DATAFILE`-Klausur des `ALTER TABLESPACE`-Kommandos. Zur Angabe der Datendatei, kann sowohl der vollständige Dateiname (Pfad + Dateiname), als auch die Dateinummer verwendet werden.



Zum Löschen einer Datendatei, muss der Tablespace online sein!

Listing 18.31: Eine Datendatei löschen

```
SQL> ALTER TABLESPACE bank ONLINE;
SQL> ALTER TABLESPACE bank
  2  DROP DATAFILE '/u02/oradata/orcl/bank02.dbf';
```

Listing 18.32: Benutzen der Dateinummer zum Löschen einer Datendatei

```
SQL> col tablespace_name format a15
SQL> col file_name format a50
SQL> col file_id format 999999
SQL> set linesize 200

SQL> SELECT    tablespace_name, file_name, file_id
  2  FROM      dba_data_files
```

```

3 WHERE      LOWER(tablespace_name) LIKE 'bank',
3 ORDER BY file_id;

TABLESPACE_NAME FILE_NAME                                FILE_ID
-----
BANK           /u02/oradata/orcl/bank01.dbf            6
BANK           /u03/oradata/orcl/bank02.dbf            7

SQL> ALTER TABLESPACE bank
2  DROP DATAFILE 6;

```

18.5 Temporäre Datendateien

Temporäre Tablespace verwenden keine gewöhnlichen Datendateien, sondern Tempfiles. Diese unterscheiden sich wie folgt von permanenten Datendateien:

- Sie können nicht Read Only gesetzt werden
- Es wird kein Recovery für Tempfiles durchgeführt
- Anders als bei permanenten Datendateien hat ein Tempfile bei seiner Erstellung noch nicht seine volle Größe.

18.5.1 Tempfiles on- und offline setzen

Um die Tempfiles eines temporären Tablespace offline zu setzen, wird das Schlüsselwort **DATAFILE** durch das Schlüsselwort **TEMPFILE** ersetzt. Auch für Tempfiles kann die Dateinummer verwendet werden. Diese findet man in der View **DBA_TEMP_FILES**.

Listing 18.33: Ein Tempfile mit Hilfe des Dateinamens offline setzen

```

SQL> ALTER DATABASE
2  TEMPFILE '/u02/oradata/orcl/temp01.dbf' OFFLINE;

```

Listing 18.34: Offline-/Onlinesetzen eines Tempfiles mittels der Dateinummer

```

SQL> col tablespace_name format a15
SQL> col file_name format a50
SQL> col file_id format 999999
SQL> set linesize 200
SQL> SELECT    tablespace_name, file_name, file_id
2  FROM        dba_temp_files;

```

TABLESPACE_NAME FILE_NAME	FILE_ID
TEMP /u02/oradata/orcl/temp01.dbf	1

```

SQL> ALTER DATABASE
  2  TEMPFILE 1 OFFLINE;

Database altered.

SQL> ALTER DATABASE
  2  TEMPFILE 1 ONLINE;

Database altered.
```

18.5.2 Tempfiles zu einem temporären Tablespace hinzufügen

Beim temporären Tablespace wird die `ADD DATAFILE`-Klausel durch `ADD TEMPFILE` ersetzt.

Listing 18.35: Hinzufügen eines Tempfiles

```

SQL> ALTER TABLESPACE temp
  2  ADD TEMPFILE '/u02/oradata/orcl/temp02.dbf' SIZE 100M;
```

18.5.3 Tempfiles löschen

Ein unbenutztes Tempfile wird mit der `DROP TEMPFILE`-Klausel des `ALTER TABLESPACE`-Kommandos gelöscht.

Listing 18.36: Löschen eines Tempfiles

```

SQL> ALTER TABLESPACE temp
  2  DROP TEMPFILE '/u02/oradata/orcl/temp02.dbf';
```

Auch bei Tempfiles kann statt dem Dateinamen, die Dateinummer genutzt werden.

Listing 18.37: Löschen eines Tempfiles mit Hilfe der Dateinummer

```

SQL> ALTER TABLESPACE temp
  2  DROP TEMPFILE 2;
```

18.6 Informationen

18.6.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10029]

18.6.2 Verzeichnis der relevanten Data Dictionary Views



- [sthref1938]
- [sthref1988]
- [REFRN23076]
- [sthref2435]
- [sthref2550]
- [sthref2555]
- [sthref2563]
- [sthref2579]
- [sthref3281]
- [sthref3785]
- [sthref3790]

18.7 Übungen - Datenbank Storage Strukturen verwalten

1. Ermitteln Sie die Datenblockgröße in Ihrer Datenbank!

2. Wie lauten die Namen der Tablespace, aus denen Ihre Datenbank aktuell besteht?

3. Gibt es in Ihrer Datenbank einen Tablespace, der aus Uniform-Sized Extents besteht?

4. Aus wie vielen Extents besteht das Segment BANK.BUCHUNG (DBA_SEGMENTS)?

5. Erstellen Sie den smallfile Tablespace UEBUNG_TS mit den beiden Datendateien uebung_ts01.dbf (Größe 100 M, Maximalgröße 500 M) und uebung_ts02.dbf (Größe 200 M). Verteilen Sie die Datendateien auf die Ihnen zur Verfügung stehenden Datenträger /u02 und /u03.

6. Erstellen Sie den Bigfile Tablespace BIG_DATA mit einer Größe von 1G und gleichförmigen, 512 Kb großen Extents. Legen Sie die Datendatei BIG_DATA.DBF auf /u02 ab.

7. Welche Tablespace in Ihrer Datenbank sind Bigfile Tablespace?

8. Wie heißt der temporäre Tablespace Ihrer Datenbank und wie groß ist sein Tempfile in Megabyte?

9. Legen Sie den temporären smallfile Tablespace TEMP_TS an. Sein Tempfile soll 500 M groß sein, auf bis zu 4 G anwachsen können, temp_ts01.dbf heißen und auf Laufwerk /u02 liegen. Benutzen Sie 1 M große Uniform-Sized Extents.

10. Verändern Sie die Datendatei uebung_ts02.dbf so, dass sie auf bis zu 1 G Größe anwachsen kann.
11. Führen Sie das Skript /home/oracle/labs/lab_fullts.sql aus. Nach der Ausführung des Skripts existiert ein neuer Tablespace FULLTS und ein neuer Nutzer ALICE.

12. Melden Sie sich als Nutzer ALICE (Passwort: Welcome01#) an Ihrer Datenbank an und versuchen Sie das folgende SQL-Statement abzusetzen:

```
SQL> INSERT INTO full  
2 VALUES (1,1,1);
```

Die Ausführung des Statements wird scheitern. Prüfen Sie im Alert.log, warum das Statement gescheitert ist.

13. Versuchen Sie das Problem aus der vorangegangenen Aufgabe auf zwei unterschiedliche Arten zu lösen.

14. Setzen Sie den Tablespace fullts offline. Mit welcher Option müssen Sie den Tablespace offline setzen, so das kein Recovery des Tablespace notwendig ist?
-

15. Löschen Sie den Tablespace fullts mit all seinen Datendateien.

16. Verschieben Sie die Datendatei uebungs_ts02.dbf nach /u02/oradata/orcl/.

17. Führen Sie zum Abschluss das Skript labs/lab_dbadmin05_cleardb.sql aus, um die Datenbank aufzuräumen. Es werden alle Tablespaces gelöscht, die im Rahmen dieser Übung angelegt werden sollten.

18.8 Lösungen - Datenbank Storage Strukturen verwalten

1. Ermitteln Sie die Datenblockgröße in Ihrer Datenbank!

```
SQL> col name format a15
SQL> col value format a10
SQL> SELECT name, value
  2  FROM v$system_parameter
  3  WHERE name LIKE 'db_block_size';

NAME          VALUE
-----
db_block_size    8192
```

2. Wie lauten die Namen der Tablespace, aus denen Ihre Datenbank aktuell besteht?

```
SQL> SELECT tablespace_name
  2  FROM dba tablespaces;

TABLESPACE_NAME
-----
SYSTEM
SYSAUX
UNDOTBS1
TEMP
USERS
EXAMPLE
```

3. Gibt es in Ihrer Datenbank einen Tablespace, der aus Uniform-Sized Extents besteht?

```
SQL> col allocation_type format a15
SQL> SELECT tablespace_name, allocation_type
  2  FROM dba tablespaces
  3  WHERE allocation_type LIKE 'UNIFORM';

TABLESPACE_NAME           ALLOCATION_TYPE
-----
TEMP                      UNIFORM
```

4. Aus wie vielen Extents besteht das Segment BANK.BUCHUNG (DBA_SEGMENTS)?

```
SQL> col segment_name format a30
SQL> SELECT segment_name, extents
  2  FROM dba segments
  3  WHERE LOWER(segment_name) LIKE 'buchung';

SEGMENT_NAME           EXTENTS
-----
BUCHUNG
```

5. Erstellen Sie den smallfile Tablespace UEBUNG_TS mit den beiden Datendateien uebung_ts01.dbf (Größe 100 M, Maximalgröße 500 M) und uebung_ts02.dbf (Größe 200 M). Verteilen Sie die Datendateien auf die Ihnen zur Verfügung stehenden Datenträger /u02 und /u03.

```
SQL> CREATE TABLESPACE uebung_ts
  2  DATAFILE '/u02/oradata/orcl/uebung_ts01.dbf' SIZE 100M
  3          AUTOEXTEND ON MAXSIZE 500M,
  4          '/u03/oradata/orcl/uebung_ts02.dbf' SIZE 200M;
```

6. Erstellen Sie den Bigfile Tablespace BIG_DATA mit einer Größe von 1G und gleichförmigen, 512 Kb großen Extents. Legen Sie die Datendatei BIG_DATA.DBF auf /u02 ab.

```
SQL> CREATE BIGFILE TABLESPACE big_data
  2  DATAFILE '/u02/oradata/orcl/big_data.dbf' SIZE 1G
  3  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 512K;
```

7. Welche Tablespaces in Ihrer Datenbank sind Bigfile Tablespaces?

```
SQL> SELECT tablespace_name, bigfile
  2  FROM dba_tablespaces
  3  WHERE bigfile LIKE 'YES'

TABLESPACE_NAME          BIG
-----
BIG_DATA                 YES
```

8. Wie heißt der temporäre Tablespace Ihrer Datenbank und wie groß ist sein Tempfile in Megabyte?

```
SQL> SELECT tablespace_name, bytes / POWER(1024, 2) AS MB
  2  FROM dba_temp_files;

TABLESPACE_NAME          MB
-----
TEMP                     20
```

9. Legen Sie den temporären smallfile Tablespace TEMP_TS an. Sein Tempfile soll 500 M groß sein, auf bis zu 4 G anwachsen können, temp_ts01.dbf heißen und auf Laufwerk /u02 liegen. Benutzen Sie 1 M große Uniform-Sized Extents.

```
SQL> CREATE TEMPORARY TABLESPACE temp_ts
  2  TEMPFILE '/u02/oradata/orcl/temp_ts01.dbf' SIZE 500M
  3          AUTOEXTEND ON MAXSIZE 4G
  4  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 1M;
```

10. Verändern Sie die Datendatei uebung_ts02.dbf so, dass sie auf bis zu 1 G Größe anwachsen kann.

```
SQL> ALTER DATABASE
  2  DATAFILE '/u03/oradata/orcl/uebung_ts02.dbf'
  3    AUTOEXTEND ON MAXSIZE 1G
```

11. Führen Sie das Skript /home/oracle/labs/lab_fullts.sql aus. Nach der Ausführung des Skripts existiert ein neuer Tablespace FULLTS und ein neuer Nutzer ALICE.

12. Melden Sie sich als Nutzer ALICE (Passwort: Welcome01#) an Ihrer Datenbank an und versuchen Sie das folgende SQL-Statement abzusetzen:

```
SQL> INSERT INTO full
  2  VALUES (1,1,1);
```

Die Ausführung des Statements wird scheitern. Prüfen Sie im Alert.log, warum das Statement gescheitert ist.

```
ERROR at line 1:
ORA-01653: unable to extend table ALICE.FULL by 128 in tablespace FULLTS
```

13. Versuchen Sie das Problem aus der vorangegangenen Aufgabe auf zwei unterschiedliche Arten zu lösen.

```
SQL> SELECT file_name
  2  FROM dba_data_files
  3  WHERE tablespace_name LIKE 'FULLTS';

FILE_NAME
-----
/u02/oradata/orcl/fullts01.dbf

-- Loesung 1
SQL> ALTER DATABASE
  2  DATAFILE '/u02/oradata/orcl/fullts01.dbf' RESIZE 20M;

-- Loesung 2
SQL> ALTER TABLESPACE fullts
  2  ADD DATAFILE '/u03/oradata/orcl/fullts02.dbf' SIZE 20M;
```

14. Setzen Sie den Tablespace fullts offline. Mit welcher Option müssen Sie den Tablespace offline setzen, so das kein Recovery des Tablespaces notwendig ist?

```
SQL> ALTER TABLESPACE fullts OFFLINE NORMAL;
```

15. Löschen Sie den Tablespace fullts mit all seinen Datendateien.

```
SQL> DROP TABLESPACE fullts
2   INCLUDING CONTENTS AND DATAFILES;
```

16. Verschieben Sie die Datendatei uebung_ts02.dbf nach /u02/oradata/orcl/.

```
SQL> col file_name format a50
SQL> SELECT file_name, file_id
2  FROM dba_data_files
3  WHERE tablespace_name LIKE 'UEBUNG_TS';

FILE_NAME                                FILE_ID
-----                                 -----
/u02/oradata/orcl/uebung_ts01.dbf          8
/u03/oradata/orcl/uebung_ts02.dbf          9

SQL> ALTER TABLESPACE uebung_ts OFFLINE NORMAL;

SQL> host mv /u03/oradata/orcl/uebung_ts02.dbf /u02/oradata/orcl

SQL> ALTER DATABASE
2  RENAME FILE '/u03/oradata/orcl/uebung_ts02.dbf'
3    TO '/u02/oradata/orcl/uebung_ts02.dbf';

SQL> ALTER TABLESPACE uebung_ts ONLINE;
```

17. Führen Sie zum Abschluss das Skript labs/lab_dbadmin05_cleardb.sql aus, um die Datenbank aufzuräumen. Es werden alle Tablespaces gelöscht, die im Rahmen dieser Übung angelegt werden sollten.

19 Lokale Benutzerverwaltung

Inhaltsangabe

19.1 Lokale Nutzer erstellen und verwalten

Lokale Nutzerkonten werden direkt in der Datenbank gespeichert und die Datenbank hält auch den Authentifizierungsmechanismus bereit. Diese Form der Nutzerverwaltung wird sehr häufig genutzt, ist jedoch nicht die Sicherste, da z. B. erst seit Oracle 11g die Passwörter case-sensitiv gespeichert werden.

Ein Datenbanknutzerkonto wird in SQL mit dem Kommando `CREATE USER` erstellt. Für diesen Vorgang wird das Systemprivileg `CREATE USER` benötigt.

19.1.1 Nutzer in SQL erstellen

Beispiel ?? zeigt, wie ein Nutzer in SQL erstellt wird.

Listing 19.1: Das CREATE USER Statement

```
SQL> CREATE USER oracle
  2 IDENTIFIED BY      oracle
  3 DEFAULT TABLESPACE users
  4 QUOTA              100 M ON users
  5 TEMPORARY TABLESPACE temp
  6 PASSWORD            EXPIRE
  7 PROFILE             employee;

SQL> GRANT create session TO oracle;
```



Ein neuer Nutzer wird ohne jegliche Privilegien erstellt, was bedeutet, dass er sich noch nicht einmal an der Datenbank anmelden kann. Es muss ihm erst das `CREATE SESSION` Privileg zugewiesen werden.

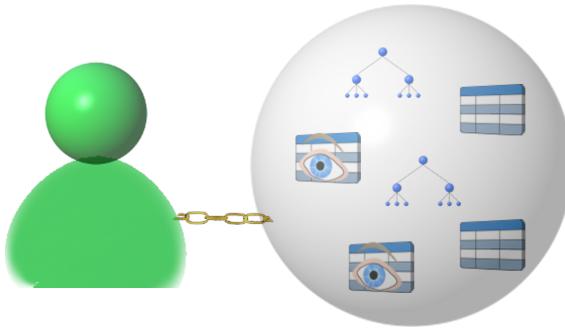
Nutzernamen vergeben

Der Name eines Datenbanknutzers muss innerhalb der Datenbank eindeutig sein. Der Name des Nutzerkontos wird in der `CREATE USER`-Klausel festgelegt. Zu jedem Nutzer wird auch ein gleichnamiges Schema angelegt, welches die Objekte des Nutzers enthält.



Ein Schema stellt einen Namensraum dar, innerhalb dem die Objekte eines Nutzers existieren. Jedes Nutzerkonto ist Besitzer eines gleichnamigen Schemas und somit Besitzer der Objekte im Schema.

Abb. 19.1:
Nutzer und
Schema



Nutzer authentifizieren

In Beispiel ?? wird der Nutzer „oracle“ durch sein Passwort authentifiziert. Die Passwortzuweisung geschieht in der **IDENTIFIED BY**-Klausel. Für Passwörter gelten folgende Regeln:

- Passwörter sollten zwischen 12 und 30 Zeichen lang sein.
- Es sollten mindestens ein Großbuchstabe, ein Kleinbuchstabe, eine Ziffer und ein Sonderzeichen darin vorkommen.
- Passwörter die mit einer Ziffer oder einem Sonderzeichen beginnen, bzw. die ein Sonderzeichen enthalten, müssen in Anführungszeichen eingeschlossen werden.

Da in Beispiel ?? die Klausel **PASSWORD EXPIRE** angegeben wurde, muss der Nutzer bei seiner ersten Anmeldung das abgelaufene Passwort ändern.

Der Default Tablespace

Wenn ein Nutzer neue Datenbankobjekte, wie z. B. Tabellen und Indizes erstellt, werden diese im Default Tablespace des Nutzers angelegt. Die Zuweisung eines Default Tablespace geschieht mit Hilfe der Klausel **DEFAULT TABLESPACE**.



Ein Tablespace der einem Nutzer als Default Tablespace zugewiesen wurde, kann erst gelöscht werden, wenn den betroffenen Nutzern ein anderer Default Tablespace zugewiesen wurde.

Wurde einem Nutzer kein Default Tablespace zugewiesen, gilt für ihn der Default Permanent Tablespace. Die View **DATABASE_PROPERTIES** enthält den Namen dieses Tablespace.

Listing 19.2: Der Default Permanent Tablespace

```
SQL> SELECT property_name , property_value
```

```
2 FROM database_properties  
3 WHERE property_name = 'DEFAULT_PERMANENT_TABLESPACE';
```



Wird bei der Erstellung der Datenbank kein Standard festgelegt, wird automatisch der SYSTEM-Tablespace zum Default Tablespace eines jeden Nutzers.

Die Begriffe Schema und Tablespaces sollten nicht verwechselt werden. Während Schemas die Besitzverhältnisse innerhalb der Datenbank regeln, regeln Tablespaces die physische Ablage der Daten in den Datendateien. Die Daten eines Nutzers gehören immer zu genau einem Schema, können aber über mehrere Tablespaces verteilt gespeichert sein.

Speicherbegrenzung - Tablespace Quotas

Quotas ermöglichen es, den Nutzern einer Oracle-Datenbank Speicher zuzuweisen. Während in Systemen, wie z. B. Windows Server 2003 Quotas den Platz einschränken, den ein Nutzerkonto zur Verfügung hat, stellen Quotas in Oracle eine Zuweisung von Ressourcen dar. Ohne Quotas hat ein Nutzerkonto in einer Oracle-Datenbank keinen Speicherplatz.

Quotas werden Nutzern auf Tablespace-Ebene zugewiesen. D. h. einem Nutzerkonto kann für jeden Tablespace eine eigene Quota eingerichtet werden, so dass die Speicherplatzmengen, die dem Nutzer zur Verfügung stehen, in jedem Tablespace anders sind.

In Beispiel ?? wurde dem Nutzer „oracle“ eine Quota von 100 Megabyte auf dem Tablespace USERS zugewiesen.



Es ist auch möglich, einem Nutzer die volle Menge Speicherplatz eines Tablespaces zuzuweisen. Dies geschieht mit der Anweisung: **QUOTA UNLIMITED ON users**

Quotas können bei der Erstellung eines Nutzers oder später zugewiesen werden. Bei einer Änderung der Quotas im Nachhinein auf einen kleineren Wert passiert folgendes:

- Überschreiten die Objekte des Nutzers die neue Speichermenge bereits, können sie nicht mehr wachsen.

- Ist die neu zugewiesene Speichermenge noch nicht überschritten, ist ein Wachstum bis zu dieser neuen Grenze möglich.

Besitzt ein Nutzer Objekte in einem Tablespace und seine Quota wird auf 0 MB herabgesetzt, bleiben seine Objekte erhalten, können jedoch nicht mehr wachsen. Es können dann auch keine neuen Objekte durch diesen Nutzer in diesem Tablespace angelegt werden.

Soll einem Nutzer aus einem bestimmten Grund die unbegrenzte Speicherplatzausnutzung in der Datenbank erlaubt werden, geschieht dies mit Hilfe des Privileges UNLIMITED TABLESPACE.



Dieses Privileg überschreibt alle expliziten Quotas eines Nutzers. Wird dem Nutzer dieses Privileg wieder entzogen, greifen wieder die expliziten Zuweisungen.

Vor der Zuweisung dieses Privileges an einen Nutzer sollte folgendes bedacht werden:

- **Vorteil:** Durch ein einziges Statement, hat ein Nutzer unbegrenzten Speicherplatz in der gesamten Datenbank zur Verfügung.
- **Nachteil:** Es werden alle expliziten Quota-Zuweisungen an den Nutzer überschrieben und es kann nicht selektiert werden, auf welche Tablespace sich das Privileg UNLIMITED TABLESPACE beziehen soll.

Zuweisung eines temporären Tablespace

Wird einem Nutzer nicht explizit ein temporärer Tablespace zugewiesen, geschieht folgendes:

- Wurde bei der Datenbankerstellung ein temporärer Tablespace als Standard festgelegt wird dieser benutzt.
- Wurde bei der Datenbankerstellung kein temporärer Tablespace als Standard festgelegt, wird der SYSTEM-Tablespace als temporärer Tablespace benutzt.

Um einen temporären Tablespace zuzuweisen, kennt das SQL-Kommando `CREATE USER` die Klausel `TEMPORARY TABLESPACE`. Auch für den Default temporary Tablespace gibt es einen Datenbankstandard.

Listing 19.3: Der Default Temporary Tablespace

```
SQL> SELECT property_name , property_value
  2  FROM database_properties
```

```
3 WHERE property_name = 'DEFAULT_TEMP_TABLESPACE';
```

Zuweisung eines Profils

Ein Profil ist eine Sammlung von Ressourcenlimits und Passwort-Policies. Wird einem Nutzer kein Profil zugewiesen, bekommt er ein Standardprofil. Die Zuteilung eines Profils geschieht mit der **PROFILE**-Klausel des **CREATE USER**-Kommandos.



- [BABGIFFE]

19.1.2 Nutzerkonten verändern

Ein Nutzer kann an seinem Nutzerkonto keinerlei Veränderungen, mit Ausnahme des Passworts vornehmen. Um andere Parameter eines Nutzerkontos verändern zu können bedarf der Nutzer des **ALTER USER** Privileges. Dieses Privileg lässt jedoch Veränderungen an jedem Nutzerkonto zu.



Veränderungen, die an einem Nutzerkonto vorgenommen werden, betreffen nur zukünftige Sessions, aber nicht die aktuelle.

Im folgenden Beispiel wird der Nutzer oracle verändert.

Listing 19.4: Das **ALTER USER** Statement

```
SQL> ALTER USER          oracle
  2 IDENTIFIED BY        "dr0wss@P"
  3 DEFAULT TABLESPACE   example
  4 QUOTA                100 M ON example
  5 QUOTA                0 M ON users
  6 PROFILE               entwickler;
```

Die vorgenommenen Änderungen sind:

- Das Passwort wird auf „dr0wss@P“ geändert.
- Als Default Tablespace wird der Tablespace EXAMPLE festgelegt.
- Der Nutzer oracle erhält 100 M Speicherplatz im Tablespace EXAMPLE.

- Der Speicherplatz im Tablespace USERS wird auf 0 M reduziert.
- Er bekommt ein neues Profil zugewiesen.

Das Passwort eines Nutzers ändern

Das Passwort eines Nutzerkontos kann durch den Datenbankadministrator, ohne Kenntnis des aktuellen geändert werden.

Listing 19.5: Das Passwort eines Nutzers ändern

```
SQL> ALTER USER      hr
  2 IDENTIFIED BY hr;
```

Nutzerkonten sperren und entsperren

Der DBA kann Nutzerkonten jederzeit manuell sperren und wieder entsperren. Hierfür wird die **ACCOUNT LOCK**- bzw. **ACCOUNT UNLOCK**-Klausel des **ALTER USER**-Kommandos benutzt.



Die Beispiel-Nutzerkonten, wie z. B. HR oder SH sind nach der Datenbankinstallation gesperrt und müssen manuell entsperrt werden!

Listing 19.6: Ein Nutzerkonto manuell entsperren

```
SQL> ALTER USER hr
  ACCOUNT UNLOCK;
```

Soll das Konto HR aus Sicherheitsgründen wieder gesperrt werden, wird die **ACCOUNT LOCK**-Klausel benutzt.

Listing 19.7: Ein Nutzerkonto manuell sperren

```
SQL> ALTER USER hr
  ACCOUNT LOCK;
```

Falls jemand nun versucht sich mit dem Nutzerkonto HR an der Datenbank anzumelden, wird dies mit dem Oracle Fehler ORA-28000 belohnt.

Listing 19.8: Ein Nutzerkonto manuell sperren

```
SQL> connect / as sysdba
Connected.
```

```
SQL> ALTER USER hr
      ACCOUNT LOCK;
SQL> connect hr/hr
ERROR:
ORA-28000: the account is locked

Warning: You are not longer connected to ORACLE.
```



- [BABHDBBE]

19.1.3 Nutzerkonten löschen

Beim Löschen eines Nutzerkontos wird das Schema aus dem Data Dictionary entfernt und alle Objekte des Nutzers gelöscht. Um einen Nutzer löschen zu können, müssen zwei Bedingungen erfüllt werden:

- Es wird das `DROP USER` Privileg benötigt.
- Der zu löschenende Nutzer darf nicht mehr angemeldet sein.

Ist der betreffende Nutzer noch an der Datenbank angemeldet, muss erst seine Session beendet werden.

Beenden einer Nutzersession

Folgende Schritte sind notwendig, um die Session eines Nutzers zu beenden:

1. Die beiden Angaben `sid` und `serial#` aus der View `V$SESSION` herausfinden.

Listing 19.9: Abfragen von sid und serial#

```
SQL> SELECT sid, serial#, username
  2  FROM v$session
  3 WHERE LOWER(username) LIKE 'oracle';

      SID      SERIAL# USERNAME
----- -----
     141        259 oracle
```

2. Session beenden

Listing 19.10: Beenden einer Nutzersession

```
-- Warten bis die Session die aktuelle Transaktion beendet
SQL> ALTER SYSTEM DISCONNECT SESSION '141,259' POST_TRANSACTION;

-- Die Session sofort beenden
SQL> ALTER SYSTEM DISCONNECT SESSION '141,259' IMMEDIATE;
```

- [sthref938]



Das Benutzerkonto löschen

Ein Benutzerkonto wird mit dem Kommando `DROP USER` gelöscht. Enthält das Schema des zu löschenen Nutzers noch Objekte, muss zusätzlich die `CASCADE`-Klausel angegeben werden, um diese Objekte ebenfalls zu löschen.

Listing 19.11: Das DROP USER Statement

```
SQL> DROP USER oracle CASCADE;
```



- [DBSEG99791]

19.2 Authentifizierung

Unter dem Begriff „Authentifizierung“ versteht man, die Identität einer Person, einer Anwendung oder eines Gerätes festzustellen. Es wird damit festgelegt, ob die Person, die Anwendung oder das Objekt vertrauenswürdig ist oder nicht.

Oracle kennt verschiedene Authentifizierungsformen und es ist jeder Instanz erlaubt alle Arten gemischt zu verwenden. Für Datenbankadministratoren wird eine eigene Form der Authentifizierung verwendet, da dieser Personenkreis über besondere Berechtigungen in der Datenbank verfügt.

Folgende Formen der Authentifizierung kennt Oracle: Authentifizierung durch...

- die Datenbank (Lokale Authentifizierung)
- das Betriebssystem
- das Netzwerk (LDAP)
- Multi-Tier Systeme (Mehrschichtige Systeme mit Anwendungsservern)
- SSL (Secure Socket Layer)
- Kerberos

19.2.1 Authentifizierung durch die Datenbank

Eine Oracle-Datenbank kann Nutzer authentifizieren, in dem sie Informationen verwendet, die in der Datenbank selbst gespeichert sind. Dies ist die einfachste Form der Authentifizierung, da kein zusätzlicher Authentifizierungsdienst konfiguriert werden muss.

Passwort-Hashes in Oracle 10g und 11g

Die Abfrage in Beispiel ?? zeigt, welche Informationen über die Nutzerkonten in der Datenbank gespeichert sind.

Die Spalte AUTHENTICATION_TYPE zeigt an, wie der Nutzer authentifiziert wird:

- **PASSWORD:** Authentifizierung durch ein in der Datenbank gespeichertes Passwort.
- **EXTERNAL:** Authentifizierung durch einen externen Mechanismus (SSL, Kerberos, Betriebssystem, uvm.)
- **GLOBAL:** Authentifizierung durch das „Oracle Internet Directory“.

Listing 19.12: In der Datenbank gespeicherte Nutzerdaten abrufen

USERNAME	PASSWORD	AUTHENTICATION_TYPE
SYS	10G 11G	PASSWORD
SYSTEM	10G 11G	PASSWORD
SCOTT	10G 11G	PASSWORD
SH	10G 11G	PASSWORD
BANK	10G 11G	PASSWORD

Die beiden Werte „10G“ und „11G“ in der Spalte PASSWORD geben an, mit welchem Hashing-Verfahren die Passwörter der Nutzer verschlüsselt wurden. Oracle 10g verwendet das case-insensitive 3DES (Triple-DES) Verfahren, während ab Oracle 11g R1 Passwörter case-sensitive mit dem SHA-1 Algorithmus verschlüsselt werden können.

Werden beide Werte, „10G“ und „11G“ angezeigt, bedeutet das, dass für jeden Nutzer zwei Passwort-Hashes gespeichert sind, die Oracle 10g konforme Variante und der neue SHA-1 Hash von Oracle 11g.

Listing 19.13: Einblicke in das Data Dictionary - Passwordhashes

SQL> col name format a6
SQL> col "10G 3DES" format a16
SQL> col "11G SHA-1" format a70
SQL> set linesize 150
SQL> SELECT name, password AS "10G 3DES", spare4 AS "11G SHA-1"
2 FROM sys.user\$
3 WHERE LOWER(name) IN ('sys', 'system', 'hr');

```

NAME    10G 3DES
-----
11G SHA-1
-----
SYS     268915ED849BEC9 S : BAEAB7196AC176EF8A42927DA297CE10B784982522420C31035E5
SYSTEM   2D594E86F93B17A1 S : E35D71785B8AC9596335215AF88BAF660ABD72F257EE1BBBAEE39
HR      6399F3B38EDF3288 S : 33567EF031F4A8BF5C9F79CB1CC910D1DB86E4C02E0C9E0DBB18A

```

Beispiel ?? zeigt die beiden Passworthashes direkt nebeneinander (der SHA-1 Hash wurde aus Platzgründen um sieben Stellen gekürzt).



Oracle sorgt automatisch dafür, dass Passwörter transparent für die Übertragung im Netzwerk verschlüsselt werden. Hierfür wird der AES-Standard (Advanced Encryption Standard) verwendet.

19.2.2 Authentifizierung der Datenbankadministratoren mit Passworddatei

Oracle bietet für DBAs die Nutzung zweier Authentifizierungsformen an. Eine Betriebssystemauthentifizierung und die Nutzung einer Passworddatei. Die Passworddatei ist eine kleine, verschlüsselte Datei, die Angaben zu allen Datenbankadministratorkonten speichert. DBAs benötigen sie, um sich remote an der Datenbank anmelden zu können. Da die Passworddatei außerhalb der Datenbank liegt, können sich DBAs auch dann anmelden, wenn die Instanz heruntergefahren ist. Dies ist zum Hochfahren der Instanz notwendig.

SYSDBA und SYSOPER

Die Privilegien sysdba und sysoper sind die beiden umfassensten Privilegien, welche die Oracle-Datenbank kennt. Ein Nutzer, der das sysdba-Privileg besitzt, kann alles mit der Datenbank machen. Ohne zusätzliche Software (Oracle Database Vault) kann er in seiner Handlungsweise nicht eingeschränkt werden. sysoper dagegen ist nur ein minimal abgestuftes Privileg, das in der Praxis faktisch keine Anwendung findet, da für alle wesentlichen administrativen Tätigkeiten, wie z. B. Backup and Recovery das sysdba-Privileg benötigt wird.



Da die Kontrolle über diese beiden Privilegien außerhalb der Datenbank liegt, muss ein Administrator bei der Anmeldung angeben, das er das sysdba-Privileg nutzen möchte.

Das SYSASM-Privileg

Mit Oracle 11g kam ein neues sys-Privileg hinzu: `sysasm`. Dieses Privileg ist ausschließlich für die Administration einer Oracle „Automatic Storage Management“-Instanz zuständig.

Nutzung einer Passworddatei vorbereiten

Folgende Schritte sind notwendig, um eine Passworddatei einzurichten:

- Erstellen einer Passworddatei, falls nicht bereits eine existiert.
- Den Parameter `remote_login_passwordfile` auf den Wert *EXCLUSIVE* setzen (Vorsicht! Dieser Parameter ist statisch).
- Als Nutzer `sys` an der Datenbank anmelden
- Den neuen Nutzer erstellen
- Eines der beiden Privilegien `sysdba` oder `sysoper` an den neuen Nutzer vergeben.



Durch die Vergabe des `sysdba` oder des `sysoper` Privilegs, wird der Nutzer in die Passworddatei aufgenommen und kann sich als Administrator an der Datenbank anmelden.

Für die Erstellung einer Passworddatei hält Oracle das Tool `orapwd` bereit. Wird dieses Tool ohne die Angabe von Parametern aufgerufen, erscheint folgende Bildschirmmeldung:

Listing 19.14: Das Tool ORAPWD

```
[oracle@FEA11-119SRV ~]$ orapwd Usage: orapwd file=<fname> entries=<users> force=<y/n>
      ignorecase=<y/n> nosysdba=<y/n>

      where
      file - name of password file (required),
      password - password for SYS will be prompted if not specified at command line,
      entries - maximum number of distinct DBA (optional),
      force - whether to overwrite existing file (optional),
      ignorecase - passwords are case-insensitive (optional),
      nosysdba - whether to shut out the SYSDBA logon (optional Database Vault only)

      There must be no spaces around the equal-to (=) character.
```

In Beispiel ?? wird eine neue Passworddatei namens ORAPWORCL erstellt, die bis zu fünf Einträge aufnehmen kann. Das Passwort des Nutzers SYS muss seit Oracle 11g nicht mehr im Klartext eingegeben werden, das orapwd-Tool fragt interaktiv nach dem Passwort.

Listing 19.15: Erstellen einer Passworddatei mit ORAPWD

```
orapwd file=$ORACLE_HOME/dbs/orapworcl entries=5 ignorecase=n
```

Die Parameter des Kommandos orapwd haben folgende Bedeutung:

- **FILE:** Dieser Parameter gibt den Namen der Passworddatei an. Es muss eine vollständige Angabe aus Pfad und Dateiname gemacht werden. Die Angabe dieses Parameters ist zwingend Vorgeschrieben.
- **ENTRIES:** Gibt die maximale Anzahl der möglichen Einträge in der Passworddatei vor. Ist eine Passworddatei zu klein, muss eine neue, größere erstellt werden und die Zuweisungen der sysdba/sysoper Privilegien an alle betroffenen Nutzer muss neu gemacht werden.
- **IGNORECASE:** Mit ignorecase=n wird dafür gesorgt, dass die Passwörter der Administratoren case-sensitiv gespeichert werden.



Damit eine Oracle-Datenbank ihre Passworddatei findet, muss der Name sich aus orapw und der SID der Datenbank zusammensetzen, z. B. orapworcl.

Die Passworddatei zur Authentifizierung benutzen

Wenn sich ein Administrator mit Hilfe von SQL*Plus an seiner lokalen oder einer Remotedatenbank anmelden möchte, tut er dies mit seinem Nutzernamen und dem Zusatz as sysdba oder as sysoper. Wurde beispielsweise dem Nutzer HR das Privileg sysdba zugewiesen, kann er sich wie folgt anmelden:

Listing 19.16: Anmelden mit dem sysdba-Privileg

```
[oracle@FEA11-119SRV ~]$ sqlplus hr/hr as sysdba
```

Die Anmeldung mit dem Zusatz as sysoper schlägt jedoch fehl, da dem Nutzer HR nur das sysdba, aber nicht das sysoper Privileg erteilt wurde.



Ein Nutzer, der sich mit sysdba oder sysoper Privilegien anmeldet, wird nicht mit seinem normalen Schema, sondern mit einem Standardschema verbunden. Für das sysdba Privileg ist dies das Schema des Nutzers SYS. Für sysoper ist es das Schema PUBLIC.

Das folgende Beispiel zeigt was passiert, wenn sich ein Nutzer normal bzw. mit administrativen Privilegien anmeldet. Dem Benutzer HR wird das hr-Privileg zugeteilt. Anschließend meldet sich dieser einmal mit dem Zusatz as sysdba und einmal ohne diesen an.

Listing 19.17: Anmeldung mit und ohne administrativen Privilegien

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba SQL*Plus:
Release 11.2.0.1.0 Production on Mon Sep 9 11:32:49 2013
Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> GRANT sysdba TO hr;
SQL> connect hr/hr
SQL> show user
USER is "HR"
SQL> disconnect
SQL> connect hr/hr as sysdba
SQL> show user
USER is "SYS"
```

Bei seiner ersten Anmeldung benutzt der Nutzer HR den Zusatz as sysdba nicht, weshalb das `show user`-Kommando das Schema HR anzeigt. Bei der zweiten Anmeldung, mit as sysdba wird statt dessen das SYS-Schema angezeigt. Beim Ermitteln, welche Nutzer in der Passworddatei enthalten sind, hilft die V\$-View `V$PWFILE_USERS` weiter.

Listing 19.18: Die View `V$PWFILE_USERS`

```
SQL> col sysdba format a6
SQL> col sysoper format a7
SQL> col sysasm format a6
SQL> SELECT *
2  FROM v$pwfile_users;

USERNAME          SYSDBA SYSOPER SYSASM
-----
SYS              TRUE   TRUE    FALSE
HR               TRUE   FALSE   FALSE
```

Um einen Nutzer wieder aus der Passworddatei zu entfernen, müssen ihm nur die sys-Privilegien sysdba, sysoper oder sysasm entzogen werden.

Listing 19.19: Einen Nutzer aus der Passworddatei entfernen

```
SQL> connect / as sysdba
SQL> REVOKE sysdba FROM hr;
SQL> SELECT *
2  FROM v$pwfile_users;
```

USERNAME	SYSDBA	SYSOPER	SYSASM
SYS	TRUE	TRUE	FALSE



- [ADMIN10241]

Der Parameter REMOTE_LOGIN_PASSWORDFILE

Zusätzlich zur Erstellung der Passworddatei, muss der Initialisierungsparameter `remote_login_passwordfile` richtig gesetzt werden. Folgende Werte sind möglich:

- **NONE**: Dieser Wert bringt Oracle dazu eine Passworddatei zu ignorieren. Das heißt, es ist keine Remoteanmeldung mit administrativen Privilegien möglich.
- **EXCLUSIVE**: Eine als exklusiv gekennzeichnete Passworddatei kann nur von einer Instanz genutzt werden. Um eine Passworddatei modifizieren zu können, muss der Parameter `remote_login_passwordfile` auf *EXCLUSIVE* gestellt werden.
- **SHARED**: Steht der Parameter auf *SHARED*, kann die Passworddatei von mehreren Instanzen auf dem gleichen Rechner genutzt werden. Eine als shared gekennzeichnete Passworddatei kann nicht verändert werden. D. h. es kann keinem weiteren Nutzer eines der Privilegien sysdba oder sysoper zugewiesen werden und es kann auch kein Nutzer mit einem dieser beiden Privilegien sein Passwort ändern.

19.2.3 Betriebssystemauthentifizierung für Administratoren

Die externe Authentifizierung durch das Betriebssystem wird für Administratoren durch eine spezielle Betriebssystemgruppe realisiert. Die Zugehörigkeit des Betriebssystemnutzerkontos zu dieser Betriebssystemgruppe verleiht dem Administrator automatisch sysdba-Privilegien in der Datenbank. Die Bezeichnung dieser Gruppe ist von Betriebssystem zu Betriebssystem unterschiedlich (Linux: „dba“, Windows: „ora_dba“).



Die externe Authentifizierung mittels Betriebssystemgruppe „dba“ hat Vorrang vor der Authentifizierung durch eine Passworddatei. Wenn jemand ein Betriebssystemnutzerkonto hat, das Mitglied in der Betriebssystemgruppe dba oder sysoper ist, kann er sich mit einem der beiden Zusätze as sysdba oder as sysoper anmelden, auch wenn er nicht in der Passworddatei aufgeführt ist. Die Authentifizierung mit administrativen Privilegien schlägt nur dann fehl, wenn jemand nicht in der Passworddatei aufgeführt ist und nicht Mitglied in den betreffenden Betriebssystemgruppen ist.

Mittels der Betriebssystemauthentifizierung kann sich ein Administrator ohne die Angabe von Nutzernamen und Passwort an der Datenbank anmelden, da diese darauf „vertraut“, dass der DBA durch das Betriebssystem authentifiziert wurde. In Beispiel ?? ist der Nutzer oracle an seinem Rechner angemeldet. Er ist Mitglied in der Betriebssystemgruppe „dba“, weshalb die Authentifizierung mit dem Zusatz as sysdba bei jedem beliebigen Datenbankaccount funktioniert .

Listing 19.20: Die Betriebssystemauthentifizierung für Administratoren

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba

SQL*Plus: Release 11.2.0.1.0 Production on Mon Sep 9 11:32:49 2013

Copyright (c) 1982, 2009, Oracle. All rights reserved.

Connected to:
Oracle Database 11g Enterprise Edition Release 11.2.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

SQL> show user
USER is "SYS"
```

- [i1006534]



19.3 Benutzerprofile

Ein Benutzerprofil ist ein Satz von Parametern für Ressourcenverwaltung und Passwortmanagement. Seit der Einführung des „Resource Manager“ in Oracle 10g, sollten jedoch die Parameter für die Ressourcenverwaltung nicht mehr genutzt werden, weshalb diese hier auch nicht näher beschrieben werden. Das Anlegen eines Benutzerprofils geschieht mittels des SQL-Kommandos `CREATE PROFILE`, gefolgt von einem Profilnamen und dem Schlüsselwort `LIMIT` werden dann die gewünschten Parameter aufgelistet.

Listing 19.21: Anzahl fehlerhafter Anmeldeversuche konfigurieren

```
SQL> CREATE PROFILE <name>
  2  LIMIT
  3    FAILED_LOGIN_ATTEMPTS  n
  4    PASSWORD_LOCK_TIME    n
  5    PASSWORD_LIFE_TIME    n
  6    PASSWORD_GRACE_TIME   n
  7    PASSWORD_REUSE_TIME   n
  8    PASSWORD_REUSE_MAX    n
  9    PASSWORD_VERIFY_FUNCTION <plsql_function>;
```

Beispiel ?? zeigt alle vorhandenen Passwortmanagement-Parameter. „n“ steht jeweils für eine Zahl. In den folgenden Abschnitten wird gezeigt, welche Passwortmanagement-Parameter es gibt und wie sie eingesetzt werden.

19.3.1 Sperrung von Nutzerkonten

Oracle kann Nutzerkonten nach einer festgelegten Anzahl fehlerhafter Anmeldeversuche sperren. Die Entsperrung kann nach einer bestimmten Zeit automatisch geschehen oder aber der Administrator muss dies übernehmen. Die manuelle Sperrung eines Accounts durch den Administrator ist ebenfalls möglich.

Das folgende Beispielprofil wird so erstellt, dass die Anzahl fehlerhafter Anmeldeversuche bei 4 und die Anzahl der Tage, die das Nutzerkonto gesperrt bleibt bei 30 liegt.

Listing 19.22: Anzahl fehlerhafter Anmeldeversuche konfigurieren und das Passwort 30 Tage sperren

```
SQL> CREATE PROFILE employee
  2  LIMIT
  3    FAILED_LOGIN_ATTEMPTS  4
  4    PASSWORD_LOCK_TIME    30;
```

Der Parameter `PASSWORD_LOCK_TIME` erwartet eine Zahl, die er als „Anzahl von Tagen“ interpretiert. Trotzdem ist es möglich die Sperrdauer des Passwortes auch auf Stunden oder Minuten zu begrenzen. Soll das Passwort nur für 45 Minuten gesperrt werden, muss ein Dezimalbruch angegeben werden: 1 Tag / 24 Stunden / 60 Minuten * 45 Minuten = (1 * 45) Tage / 24 Stunden / 60 Minuten.

Listing 19.23: Anzahl fehlerhafter Anmeldeversuche konfigurieren und das Passwort 45 Minuten sperren

```
SQL> CREATE PROFILE employee
  2  LIMIT
  3    FAILED_LOGIN_ATTEMPTS  4
  4    PASSWORD_LOCK_TIME    45 / 24 / 60;
```



Wird für den Parameter `PASSWORD_LOCK_TIME` kein Wert angegeben, wird der Standardwert „1“ herangezogen. Wird der Wert „unlimited“ für `PASSWORD_LOCK_TIME` gesetzt, muss der Administrator das Nutzerkonto manuell entsperren.

Nach einem erfolgreichen Anmeldeversuch, wird die Anzahl der fehlerhaften Anmeldeversuche auf 0 zurückgesetzt.

19.3.2 Passwortalterung und Ablauf des Passworts

Mit Hilfe des Profilparameters `PASSWORD_LIFE_TIME` kann eine maximale Lebensdauer für Passwörter festgelegt werden. Wenn die angegebene Frist verstrichen ist, muss der DBA oder der Nutzer das Passwort ändern, um sich wieder anmelden zu können. Im folgenden Beispiel wird im Profile `EMPLOYEE` eine maximale Lebensdauer von 90 Tagen für die Passwörter festgelegt.

Listing 19.24: Passwortlebensdauer

```
SQL> CREATE PROFILE employee
  2  LIMIT
  3    FAILED_LOGIN_ATTEMPTS 4
  4    PASSWORD_LOCK_TIME     30 / 24 / 60
  5    PASSWORD_LIFE_TIME     90;
```

Zusätzlich zur Lebensdauer kann auch noch eine „Gnadenfrist“ eingerichtet werden, innerhalb derer der Nutzer vor jeder Anmeldung dazu aufgefordert wird, sein Passwort zu ändern. Läuft auch diese zusätzliche Frist ab, wird das Nutzerkonto gesperrt und der Nutzer ist gezwungen sein Passwort auf ein neues zu ändern, um wieder Zugang zu seinem Konto zu erhalten.

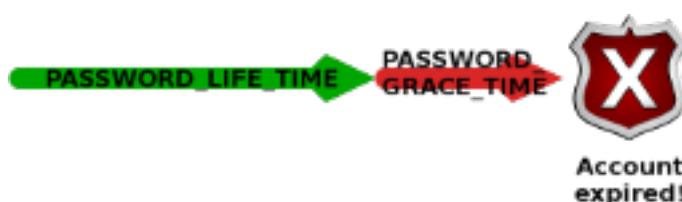


Abb. 19.2:
Lebensdauer und
Gnadenfrist eines
Passworts

Im folgenden Beispiel wird das Benutzerprofil `EMPLOYEE` angelegt, das eine Gnadenfrist von 3 Tagen vorsieht. Diese beginnt zu laufen, nachdem sich der Benutzer nach Ablauf der `PASSWORD_LIFE_TIME` versucht anzumelden. Wird das Profil wie in Beispiel ?? eingerichtet, läuft die Gnadenfrist frühestens nach 90 Tagen. Meldet sich ein Nutzer erst nach 100 Tage wieder am System an, beginnt die Gnadenfrist erst dann zu laufen.

Listing 19.25: Passwortlebensdauer und Gnadenfrist

```
SQL> CREATE PROFILE employee
```

```

2 LIMIT
3 FAILED_LOGIN_ATTEMPTS 4
4 PASSWORD_LOCK_TIME    30 / 24 / 60
5 PASSWORD_LIFE_TIME    90
6 PASSWORD_GRACE_TIME   3;

```

19.3.3 Passwort History

Eine Passwort History sorgt dafür, dass jeder Nutzer eine bestimmte Menge Passwörter benutzen muss und nicht immer nur ein einziges oder einige wenige. Mit den beiden Parametern `PASSWORD_REUSE_TIME` und `PASSWORD_REUSE_MAX` kann eine solche History aufgebaut werden.



Es sollte immer mindestens einer der beiden Parameter angegeben werden, da Passwörter sonst beliebig oft und beliebig lang benutzt werden können. Wird für einen der beiden Parameter der Wert „unlimited“ gesetzt, kann der Nutzer kein Passwort wieder verwenden. Werden beide Parameter auf den Wert „unlimited“ gesetzt ignoriert Oracle diese Einstellungen.

Das folgende Profil sorgt mit Hilfe des Parameters `PASSWORD_REUSE_TIME` dafür, dass ein Zeitraum von 30 Tagen verstreichen muss, ehe ein Nutzer ein bereits benutztes Passwort erneut verwenden kann.

Listing 19.26: Wiederverwendung eines Passworts

```

SQL> CREATE PROFILE employee
2 LIMIT
3 FAILED_LOGIN_ATTEMPTS 4
4 PASSWORD_LOCK_TIME    30 / 24 / 60
5 PASSWORD_LIFE_TIME    90
6 PASSWORD_GRACE_TIME   3
7 PASSWORD_REUSE_TIME   30;

```

Eine Alternative hierzu bietet der `PASSWORD_REUSE_MAX` Parameter. Mit ihm wird es möglich, dem Nutzer die Verwendung der n letzten Passwörter zu verbieten.

Listing 19.27: Wiederverwendung eines Passworts

```

SQL> CREATE PROFILE employee
2 LIMIT
3 FAILED_LOGIN_ATTEMPTS 4
4 PASSWORD_LOCK_TIME    30 / 24 / 60
5 PASSWORD_LIFE_TIME    90
6 PASSWORD_GRACE_TIME   3
7 PASSWORD_REUSE_MAX    13;

```

In Beispiel ?? wird das Profil so konfiguriert, dass die letzten 13 Passwörter nicht erneut verwendet werden können. Der Benutzer muss also mindestens 14 verschiedene Passwörter benutzen.

Listing 19.28: Wiederverwendung eines Passworts

```
SQL> CREATE PROFILE employee
  2  LIMIT
  3    FAILED_LOGIN_ATTEMPTS 4
  4    PASSWORD_LOCK_TIME     30 / 24 / 60
  5    PASSWORD_LIFE_TIME     90
  6    PASSWORD_GRACE_TIME   3
  7    PASSWORD_REUSE_MAX    13
  8    PASSWORD_REUSE_TIME   30;
```

Werden beide Parameter kombiniert, müssen auch beide Bedingungen erfüllt sein, bevor ein Passwort wiederverwendet werden kann. Gemäß Beispiel ?? muss ein Passwort älter als 30 Tage sein und es darf nicht in der Liste der letzten 13 Passwörter vorkommen.

19.3.4 Komplexitätsprüfung von Passwörtern

Oracle hat die Fähigkeit, ein Passwort auf seine Komplexität hin zu überprüfen. Für diese Aufgabe wird eine PL/SQL Routine verwendet. Es existiert bereits eine Standardroutine, die dem Skript `UTLPWDMG.SQL` entnommen werden kann. Der Administrator hat die Möglichkeit, diese Routine beizubehalten, sie zu verändern oder eine eigene zu entwerfen.

Falls eine eigene Routine entworfen werden soll, muss sie die folgende Signatur aufweisen:

Listing 19.29: Format der Passwortroutine

```
routinen_name
(
  username      IN VARCHAR2 ,
  password      IN VARCHAR2 ,
  old_password  IN VARCHAR2
) RETURN BOOLEAN
```

Das folgende Statement zeigt ein Beispiel für eine Komplexitätsprüfung.

Listing 19.30: Beispielkomplexitätsroutine

```
SQL> CREATE OR REPLACE FUNCTION password_verify_fkt (
  2    username      VARCHAR2 ,
  3    password      VARCHAR2 ,
  4    old_password  VARCHAR2
  5  ) RETURN BOOLEAN
  6  IS
  7  BEGIN
```

```

8   IF(LOWER(password) != 'hallo') THEN
9     RAISE_APPLICATION_ERROR(-20001,
10    'Die Komplexitaetspruefung Ihres Passworts ist fehlgeschlagen!');
11  END IF;
12  RETURN (TRUE);
13 END;

```

Eine neu erstellte Routine kann dann mittels eines Nutzerprofils zugewiesen werden.

Listing 19.31: Passwortroutine zuweisen

```

SQL> ALTER PROFILE angestellter
  2  LIMIT
  3    PASSWORD_VERIFY_FUNCTION password_verify_fkt;

```



Der Nutzer SYS muss der Besitzer der Passwortroutine sein.

Damit ein normaler Nutzer sein Passwort ändern kann, muss er das `ALTER USER`-Statement verwenden.

Listing 19.32: Ein Nutzer ändert sein Passwort

```

SQL> ALTER USER oracle IDENTIFIED BY hallo REPLACE oracle;

```

Durch die `REPLACE`-Klausel wird sichergestellt, dass der Nutzer auch das alte Passwort wissen muss, um eine Änderung durchführen zu können.

19.3.5 Benutzerprofile löschen

Ein Benutzerprofil wird mit dem Kommando `DROP PROFILE` gelöscht. Ist das betreffende Profil zum Zeitpunkt des Löschens noch einem Nutzerkonto zugewiesen, muss zusätzlich die Klausel `CASCADE` verwendet werden. Den Nutzerkonten wird dann automatisch das Profil `DEFAULT` zugewiesen.



- [i1006575]

19.4 Privilegien



Ein Privileg ist in einer Oracle-Datenbank das Recht, eine bestimmte Aktion auszuführen oder Zugriff auf ein Objekt eines anderen Nutzers zu erhalten.

Einige Beispiele hierfür sind:

- Anmelden an der Datenbank
- Erstellen einer Tabelle
- Erstellen von Views
- Verwalten von Benutzern
- Zeilen aus einer Tabelle eines anderen Nutzers selektieren
- Gespeicherte Prozeduren eines anderen Nutzers ausführen

Nutzern werden Privilegien erteilt, damit sie ihre Arbeit verrichten können. Sie sollten jedoch nur dann vergeben werden, wenn ein Nutzer sie auch wirklich benötigt, da die Vergabe nicht benötigter Privilegien die Sicherheit der Datenbank gefährden kann.

Nutzer können Privilegien auf zwei verschiedenen Wegen erhalten:

- Ein Privileg kann einem Nutzer direkt erteilt werden
- Privilegien können zu „Rollen“ zusammengefasst werden, die dann einem Nutzer zugewiesen werden.

Rollen werden an späterer Stelle in dieser Unterlage behandelt.

19.4.1 Systemprivilegien

Ein Systemprivileg ist das Recht, eine bestimmte Aktion innerhalb der Datenbank ausführen zu dürfen. Beispielsweise sind die Rechte, sich an der Datenbank anzumelden oder eine Tabelle zu erstellen beides Systemprivilegien. Es gibt insgesamt mehr als 100 einzelne Systemprivilegien.

Diese Kategorie von Privilegien betrifft meist nur Datenbankadministratoren oder Anwendungsentwickler. Die einzige Ausnahme bildet das Privileg sich an der Datenbank anmelden zu dürfen, da dies jeder Nutzer benötigt.

Systemprivilegien vergeben

Nur drei Nutzerarten können Systemprivilegien erteilen und auch wieder entziehen:

- Nutzer die das Privileg sysdba besitzen.
- Nutzer die das Privileg GRANT ANY PRIVILEGE besitzen.
- Nutzer die ein Privileg mit der „ADMIN OPTION“ übergeben bekommen haben.

Das Beispiel ?? zeigt, wie dem Nutzer HR das Privileg create session, mit Hilfe des SQL-Kommandos GRANT erteilt wird.

Listing 19.33: Zuweisen des create session-Privilegs

```
SQL> GRANT create session
  2 TO hr;
```

Da sich die Vergabe eines Privilegs augenblicklich auswirkt, kann sich der Nutzer direkt nach der Vergabe an der Datenbank anmelden.



Die Vergabe eines System Privilegs wirkt sich augenblicklich aus.

Es ist auch möglich, mehrere Privilegien auf einmal zu erteilen. Soll dem Nutzer HR gleichzeitig auch die Erstellung von Tabellen ermöglicht werden, geschieht dies wie folgt:

Listing 19.34: Zuweisen mehrerer Privilegien gleichzeitig

```
SQL> GRANT create session, create table
  2 TO hr;
```



Die Admin Option

Nutzer welche ein Systemprivileg mit der Admin Option übergeben bekommen haben, haben die Möglichkeit dieses Privileg an andere Nutzer weiterzugeben oder es den Anderen wieder zu entziehen. Daher wird dringend empfohlen, diese Option nur an äußerst sorgfältig ausgewähltes Administrationspersonal auszugeben.

In Beispiel ?? wird gezeigt, wie dem Benutzerkonto HR das `create session` Privileg mit der Admin Option übergeben wird und welche Folgen ein Missbrauch dieser Option haben kann.

Listing 19.35: Zuweisen eines System Privileges mit ADMIN OPTION

```
SQL> connect / as sysdba
Connected.
SQL> GRANT create session
  2  TO      hr WITH ADMIN OPTION;
Grant succeeded.

SQL> connect hr/hr
Connected.
SQL> GRANT create session
  2  TO      sh;
Grant succeeded.

SQL> connect sh/sh
Connected.
SQL> GRANT create session
  2  TO      oe;
ERROR at line 1:
ORA-01031: insufficient privileges

SQL> connect hr/hr
Connected.
SQL> REVOKE create session
  2  FROM    sh;

Revoke succeeded.
```

Man sieht wie der Nutzer HR das `create session`-Privileg mit der Admin Option übergeben bekommt und dieses an den Nutzer SH weiter gibt. Da er dies ohne Admin Option tut, kann SH seinerseits das Privileg nicht weitergeben. HR ist jedoch in der Lage, dem Nutzer SH das Privileg wieder zu entziehen.

Die Nutzung von Systemprivilegien einschränken

Einige Systemprivilegien tragen das Wort „ANY“ in ihrem Namen, wie z. B. das Privileg `create any table`. Der Unterschied zwischen `create table` und `create any table` ist, dass das `create table`-Privileg den Nutzer dazu berechtigt, in seinem eigenen Schema Tabellen anzulegen, während das `create any table`-Privileg ihm die Möglichkeit gibt, in jedem beliebigen Schema Tabellen anzulegen.

Ein anderes Beispiel ist das `select any table`-Privileg. Es ermöglicht einem Nutzer das Lesen aller Tabellen in allen Schemata der gesamten Datenbank. Die einzige Ausnahme bildet hier das Data Dictionary. Dies wird durch den Initialisierungsparameter `o7_dictionary_accessibility` (das erste Zeichen ist ein großes O) vor solchen Zugriffen geschützt.

Listing 19.36: Die Auswirkungen des `select any table`-Privileges

```
connect hr/hr
Connected.
SQL> SELECT *
  2  FROM sh.sales;
ERROR at line 2:
ORA-00942: table or view does not exists
SQL> connect / as sysdba
Connected.
SQL> GRANT select any table
  2  TO    hr;
SQL> connect hr/hr
Connected.
SQL> SELECT *
  2  FROM sh.sales;

  PROD_ID CUST_ID TIME_ID CHANNEL_ID PROMO_ID QUANTITY SOLD AMOUNT SOLD
-----  -----  -----  -----  -----  -----  -----  -----
      22    23226 10.02.98        4      999          1     26 ,56
      22      141 12.02.98        3      999          1     26 ,19
      22      141 12.02.98        4      999          1     26 ,19
      22      7626 13.02.98        4      999          1     26 ,56
      22     4433 16.02.98        2      999          1     26 ,61
      22     5027 16.02.98        2      999          1     26 ,61
      22     1519 16.02.98        3      999          1     26 ,24
      22     5027 16.02.98        3      999          1     26 ,24
      22     4433 16.02.98        4      999          1     26 ,61
      22      893 17.02.98        3      999          1     26 ,61
      22     1574 17.02.98        3      999          1     26 ,61
...

```



- [DBSEG99869]

Systemprivilegien entziehen

Privilegien werden in SQL mit dem Kommando `REVOKE` entzogen. Um dem Nutzer HR das Systemprivileg `CREATE ANY TABLE` wieder zu entziehen, wird folgendes Statement benutzt:

Listing 19.37: Entziehen eines Privileges

```
SQL> REVOKE create any table  
  2  FROM    hr;
```

Hiermit wird auch eine evtl. vergebene Admin Option entzogen. Ein explizites Entziehen der Admin Option, ohne das dazugehörige Privileg ist nicht möglich. Direkt nach diesem `REVOKE`-Statement kann der Nutzer HR keine Tabellen mehr in fremden Schemata erstellen.



Das Entziehen eines Systemprivilegs wirkt sich augenblicklich aus.

Es können auch mehrere Privilegien auf einmal entzogen werden:

Listing 19.38: Entziehen mehrerer Privilegien

```
SQL> REVOKE create table, create session  
  2  FROM    hr;
```

Auswirkungen des Entzuges von Systemprivilegien

1. Dem Nutzer HR wird das Privileg `create any table` mit der Admin Option übergeben.
2. HR erstellt eine Tabelle in seinem eigenen Schema.
3. Er gibt das `create any table`-Privileg an den Nutzer SH weiter.
4. SH erstellt eine Tabelle in seinem eigenen Schema.
5. Dem Nutzer HR wird das `create any table`-Privileg vom Administrator entzogen.

Die nach diesem Szenario verbleibenden Auswirkungen sind:

- Die von HR erstellte Tabelle existiert weiter.
- SH besitzt weiterhin das Privileg `create any table` und seine Tabelle bleibt ebenfalls bestehen.

Für den DBA bedeutet dies, dass der Entzug des `create any table`-Privileges sich nicht kaskadierend auswirkt. Das heißt, dass er allen Nutzern einzeln (hier HR und SH) das `create any table`-Privileg entziehen muss. Dies nur am Nutzer HR zu vollziehen genügt nicht.

19.4.2 Objektprivilegien



Objektprivilegien ermöglichen einem Nutzer den Zugriff auf Objekt in einem anderen Schema. Für fast jede Art von Schemaobjekt sind eigene Objektprivilegien vorhanden.

Objektprivilegien vergeben

Objektprivilegien werden auf die gleiche Art und Weise vergeben, wie Systemprivilegien. Der einzige Unterschied ist, dass man angeben muss, auf welches Objekt sich das Privileg beziehen soll. Muss beispielsweise dem Nutzer HR die Möglichkeit eingeräumt werden, die Tabelle SALES des Nutzers SH abzufragen, geschieht dies wie folgt:

Listing 19.39: Zuweisen eines Objekt Privilegs

```
SQL> GRANT select ON sh.sales  
2 TO hr;
```



Genau wie bei Systemprivilegien wirkt sich auch hier die Vergabe sofort nach dem `GRANT`-Statement aus.

Um einem Nutzer alle Objektprivilegien an einem Objekt zu übergeben wird das Schlüsselwort `ALL` verwendet.

Listing 19.40: Zuweisen aller Objektprivilegien

```
SQL> GRANT all ON sh.sales  
2 TO oracle;
```

Wer kann Objektprivilegien vergeben?

Jeder Nutzer hat automatisch alle Objektprivilegien für alle Objekte, die sich in seinem eigenen Schema befinden. Er kann somit die Objektprivilegien für alle eigenen Objekte an andere Nutzer weitergeben.

Eine weitere Möglichkeit Objektprivilegien verteilen zu können, ist mit dem Systemprivileg `grant any object privilege` gegeben. Besitzt ein Nutzer dieses Systemprivileg, kann er auf jedes Objekt eines beliebigen Schemas Objektprivilegien vergeben.

So wie bei den Systemprivilegien die Admin Option existiert, gibt es für Objektprivilegien die „GRANT OPTION“. Hat ein Nutzer ein Objektprivileg zusammen mit der Grant Option übergeben bekommen, kann er es an jeden anderen Nutzer weitergeben.

Listing 19.41: Zuweisen von Objektprivilegien mit GRANT OPTION

```
SQL> GRANT insert, update, delete, select ON sh.sales
  2 TO hr WITH GRANT OPTION;
```

Objektprivilegien anstelle des Objekteigentümers vergeben

Der Benutzer HR besitzt das `grant any object privilege`. Er besitzt keine Objektprivilegien auf die Tabelle SALES des Nutzers SH.

HR setzt folgendes SQL-Statement ab:

Listing 19.42: HR erteilt OE das SELECT-Privileg auf SH.SALES

```
SQL> GRANT select ON sh.sales TO oe
  2 WITH GRANT OPTION;
```

Eine Abfrage der Data Dictionary View DBA_TAB_PRIVS zeigt, dass der Nutzer SH als *grantor*¹ eingetragen wurde.

Listing 19.43: Abfrage der View DBA_TAB_PRIVS - 1

```
SQL> SELECT grantee, owner, grantor, privilege, grantable
  2 FROM dba_tab_privs
  3 WHERE table_name LIKE 'SALES' AND owner LIKE 'SH';

GRANTEE    OWNER    GRANTOR    PRIVILEGE    GRANTABLE
-----  -----  -----  -----  -----
OE          SH        SH        SELECT      YES
```

Diese Situation entsteht, da sich Oracle wie folgt verhält:

Der Nutzer, der als grantor eines Objekt Privilegs gespeichert wird, kann ein Nutzer mit dem „`grant any object privilege`“-Privileg sein oder der Eigentümer des betreffenden Objekts. Tritt ein Nutzer mit dem „`grant any object privilege`“-Privileg als grantor auf und hat dabei kein „`select`“-Privileg auf die

¹grantor = Nutzer der das Privileg vergeben hat

entsprechende Tabelle, wird der Eigentümer als grantor gespeichert. Andernfalls wird der tatsächliche grantor gespeichert.

Der Benutzer OE hat vom Nutzer HR das select-Privileg auf die Tabelle SALES des Nutzers SH mit Grant Option übergeben bekommen.

OE setzt folgendes SQL-Statement ab:

Listing 19.44: OE erteilt BI das SELECT-Privileg auf SH.SALES

```
SQL> GRANT select ON sh.sales TO bi;
```

Eine Abfrage der Data Dictionary View DBA_TAB_PRIVS zeigt, dass der Nutzer OE als grantor eingetragen wurde.

Listing 19.45: Abfrage der View DBA_TAB_PRIVS - 2

GRANTEE	OWNER	GRANTOR	PRIVILEGE	GRANTABLE
OE	SH	SH	SELECT	YES
BI	SH	OE	SELECT	NO

Da der Nutzer OE das select Privileg auf die Tabelle SALES besitzt, wird er als grantor eingetragen.

Privilegien auf Spalten vergeben

Die beiden Objektprivilegien `insert` und `update` können auch auf Spalten ebene vergeben werden. Dabei muss beachtet werden, dass alle Spalten, die ein NOT NULL-Constraint haben in die Privilegienvorgabe mit einbezogen werden oder diese Spalten müssen einen Standardwert aufweisen. Andernfalls ist es dem Nutzer nicht möglich eine Zeile in die Tabelle einzufügen. Beispiel ?? zeigt wie dem Nutzer SH das `insert`-Privileg auf die Spalten DEPARTMENT_ID und DEPARTMENT_NAME der Tabelle DEPARTMENTS des Nutzers HR übergeben wird.

Listing 19.46: Zuweisen von Objektprivilegien auf Spaltenebene

```
SQL> GRANT insert (department_id, department_name) ON hr.departments TO sh;
```

Der Benutzer SH kann diese Privilegien nun nutzen, um Datensätze in die Tabelle DEPARTMENTS einzufügen. Da er aber nur auf zwei der vier Spalten das `insert`-Privileg erhalten hat, kann er auch nur diese beiden befüllen.

Listing 19.47: Zuweisen von Objektprivilegien auf Spaltenebene

```
SQL> INSERT INTO hr.departments (department_id, department_name)
  2  VALUES ('999', 'My own department');

1 row created.

SQL> INSERT INTO hr.departments (department_id, department_name, location_id)
  2  VALUES ('888', 'my second department', 1400);

ERROR at linie 1:
ORA-01031: insufficient privileges
```

Der erste Einfügevorgang ist erfolgreich, da SH die notwendigen Privilegien besitzt. Der zweite scheitert, da der Nutzer SH kein insert-Privileg auf die Spalte LOCATION_ID hat.



Da sich in einer größeren Datenbank mit vielen Tabellen und Nutzern die Privilegienvergabe auf Spaltenebene extrem aufwendig gestalten kann, empfiehlt es sich stattdessen Views zum Einsatz zu bringen. Diese ermöglichen ebenfalls Nutzer auf bestimmte Spalten einer Tabelle einzuschränken.

Objektprivilegien entziehen

Objektprivilegien werden einem Nutzer auf die gleiche Art und Weise wie Systemprivilegien entzogen und auch die Auswirkungen diesbezüglich sind sofort zu spüren.



Ein Nutzer kann nur die Objektprivilegien entziehen, für die er als grantor eingetragen wurde.

Listing 19.48: Entziehen von Objektprivilegien

```
-- Entzieht ein einzelnes Privileg
REVOKE select ON sh.sales FROM oe;

Revoke succeeded.

-- Entzieht alle Privilegien
REVOKE ALL ON sh.sales FROM bi;

Revoke succeeded.
```



Objektprivilegien die auf einzelne Spalten vergeben wurden, können nicht spaltenweise entzogen werden. Das betreffende Privileg muss immer von der gesamten Tabelle entzogen und evtl. neu vergeben werden.

Kaskadierende Effekte beim Entzug von Objektprivilegien

Während bei den Systemprivilegien keine kaskadierenden Effekte beim Entzug auftreten, ist dies bei Objektprivilegien durchaus der Fall.

Beispiel ?? zeigt den soeben beschriebenen kaskadierenden Effekt, da mit einem einzigen `REVOKE`-Statement den beiden Nutzern OE und BI das select-Privileg auf die Tabelle SH.SALES entzogen wird.



Genauso wie die Admin Option bei Systemprivilegien, kann auch die Grant Option bei den Objektprivilegien nicht einzeln entzogen werden. Es muss erst das ganze Privileg entzogen und anschließend ohne Grant Option neu zugewiesen werden.

Listing 19.49: Entziehen von Objektprivilegien

```

SQL> connect / as sysdba
Connected.

SQL> SELECT grantee, owner, grantor, privilege, grantable
  2  FROM    dba_tab_privs
  3 WHERE   table_name LIKE 'SALES' AND owner LIKE 'SH';

no rows selected

SQL> GRANT select ON sh.sales TO oe WITH GRANT OPTION;
Grant succeeded.

SQL> connect oe/oe
Connected.

SQL> GRANT select ON sh.sales TO bi;
Grant succeeded.

SQL> connect / as sysdba
Connected.

SQL> SELECT grantee, owner, grantor, privilege, grantable
  2  FROM    dba_tab_privs
  3 WHERE   table_name LIKE 'SALES' AND owner LIKE 'SH';

GRANTEE    OWNER    GRANTOR    PRIVILEGE    GRANTABLE
-----  -----  -----  -----  -----
OE          SH        SH        SELECT      YES
BI          SH        OE        SELECT      NO

SQL> REVOKE select ON sh.sales FROM oe;

Revoke succeeded.

SQL> SELECT grantee, owner, grantor, privilege, grantable
  2  FROM    dba_tab_privs
  3 WHERE   table_name LIKE 'SALES' AND owner LIKE 'SH';

no rows selected

```



- [BABCIGHGB]

19.4.3 Rollen

Rollen sind benannte Gruppierungen von Privilegien, die Nutzern oder anderen Rollen zugewiesen werden können. Benutzerkonten und Rollen teilen sich den gleichen Namensraum, was bedeutet, dass der Name einer Rolle nicht gleichzeitig auch der Name eines Benutzerkontos sein kann.

Listing 19.50: Rollen und Benutzerkonten teilen sich den gleichen Namensraum

```
SQL> CREATE USER nameconflict
  2 IDENTIFIED BY password;

User created.

SQL> CREATE ROLE nameconflict;

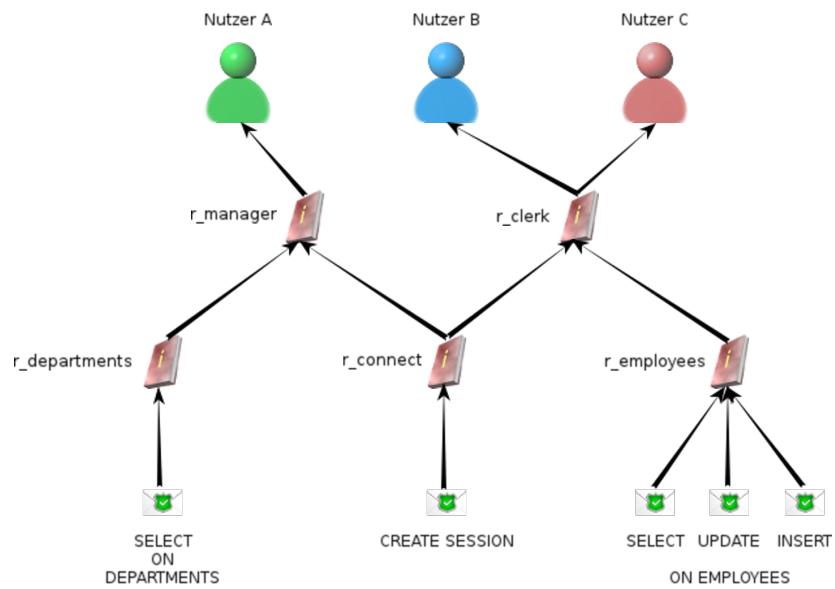
ERROR at line 1:
ORA-01921: role name 'NAMECONFLICT' conflicts with another user or role name
```

Sie sind keine Schemaobjekte, was bedeutet, dass Rollen nicht zu einem bestimmten Schema gehören. Einmal erstellt, sind sie in der gesamten Datenbank verfügbar.

Vorteile des Rollenkonzepts

- **Einfache Administration von Privilegien:** Anstatt die gleichen Privilegien mehreren Nutzern zu geben, werden diese zu einer Rolle zusammengefasst und dann den Nutzern erteilt.
- **Dynamische Verwaltung von Privilegien:** Benötigt eine Gruppe von Datenbankbenutzern andere Privilegien, müssen lediglich die betreffenden Rollen verändert werden.
- **Selektive Privilegien Verwaltung:** Rollen können ein- und ausgeschaltet werden. Eine ausgeschaltete Rolle stellt keine Privilegien mehr zur Verfügung. So können Privilegien situationsabhängig erteilt und entzogen werden.
- **Sicherheit:** Rollen können mit Passwörtern versehen werden, um sie vor unbefugter Benutzung zu sichern.

Rollen werden oft für bestimmte Datenbankanwendungen erstellt, um auf diese Weise den Nutzern der Anwendung alle benötigten Privilegien zuzuweisen.

Abb. 19.3:
Rollenkonzept

Rollen erstellen

Eine Rolle wird in SQL mit dem Statement `CREATE ROLE` erstellt, wofür das gleichnamige Privileg `CREATE ROLE` benötigt wird. Direkt nach der Erstellung besitzt eine Rolle keine Privilegien. Diese müssen später erteilt werden. Das folgende SQL-Statement erstellt die Rolle `r_employee`.

Listing 19.51: Erstellen einer Rolle

```
SQL> CREATE ROLE r_employee;
```

Wahlweise kann eine Rolle auch mit einem Passwort versehen werden.

Listing 19.52: Erstellen einer Rolle mit Passwort

```
SQL> CREATE ROLE r_manager
  2 IDENTIFIED BY password;
```

Rollen mit Privilegien ausstatten

Privilegien werden einer Rolle auf die gleiche Art und Weise zugewiesen oder entzogen, wie einem Benutzerkonto.

Listing 19.53: Eine Rolle mit Privilegien ausstatten

```
SQL> GRANT select, update, insert, delete ON hr.employees TO r_employee;
Grant succeeded.

SQL> REVOKE delete ON hr.employees FROM r_employee;
Revoke succeeded.
```

Rollen vergeben und entziehen

Jeder Nutzer, der das Systemprivileg GRANT ANY ROLE besitzt, kann anderen Nutzern oder Rollen, Rollen entziehen oder zuweisen. Für Rollen existiert ebenfalls die Admin Option. Das Verhalten dieser Option ist bei Rollen und Systemprivilegien gleich.



Eine Rolle, die einer anderen Rolle zugewiesen wurde, wird als „indirekt zugewiesene Rolle“ bezeichnet.

Listing 19.54: Zuweisen einer Rolle

```
SQL> GRANT r_employees TO r_clerk;  
  
Grant succeeded.  
  
SQL> GRANT r_clerk TO hr, sh;
```

Rollen und DDL

Um Objekte erstellen oder verändern zu können, benötigt ein Nutzer mindestens das entsprechende Systemprivileg, z. B. `create table`, `create view` oder `create procedure` und in manchen Situationen auch ein Objektprivileg. Wenn beispielsweise eine View erzeugt werden soll, muss der Ersteller folgende Privilegien besitzen:

- `create view`
- `select` auf die Basistabelle

Wird mittels einer Rolle ein Privileg zugewiesen, das zur Ausführung einer DML-Operation benötigt wird, so wird sich Oracle gegen die Ausführung des DDL-Statements sperren. Auf das Beispiel bezogen bedeutet dies konkret:

- Wird einem Nutzer das `select`-Privileg per Rolle zugewiesen, kann er die View nicht erstellen. Die Operation scheitert, da das `select`-Privileg ein Privileg ist, dass die Ausführung eines DML-Statements (im weitesten Sinne) erlaubt.
- Wird einem Nutzer das `create view`-Privileg per Rolle zugewiesen, kann er die View erstellen.

Der Benutzer HR will die View V_MARRIED_CUSTOMERS erstellen. Diese basiert auf der Tabelle CUSTOMERS des Nutzers SH. Der DBA erstellt die Rolle R_VIEWCREATOR und weist ihr das Privileg select on sh.customers zu. Das create view vergibt er direkt an HR. HR versucht nun mittels dieser Rolle die View zu erstellen.

Listing 19.55: Erstellen der View V_MARRIED_CUSTOMERS

```
SQL> CREATE ROLE r_viewcreator;

SQL> GRANT select ON sh.customers
  2  TO      r_viewcreator;

SQL> GRANT create view
  2  TO      hr;

SQL> GRANT r_viewcreator TO hr;

Grant succeeded.

SQL> connect hr/hr
Connected.
SQL> CREATE VIEW v_married_customers
  2 AS
  3   SELECT cust_id, cust_first_name, cust_last_name, cust_marital_status
  4   FROM   sh.customers
  5   WHERE  cust_marital_status LIKE 'married';

ERROR at line 4:
ORA-01031: insufficient privileges
```

Da HR das DML-Privileg aus der Rolle R_VIEWCREATOR erhält, kann er die View nicht erstellen. Damit die View erfolgreich erstellt werden kann, muss ihm das DML-Privileg direkt zugewiesen werden.



Oracle kennt eine Liste von Rollen, die direkt bei der Datenbankerstellung kreiert werden.

Die Gruppe PUBLIC

Die Gruppe PUBLIC ist vergleichbar mit der MS-Windows Nutzergruppe JEDER. Privilegien und Rollen, die dieser Gruppe zugewiesen werden, sind für alle Nutzer der Datenbank zugänglich.

19.5 Informationen

19.5.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10133]
- [REFRN10184]

19.5.2 Verzeichnis der relevanten Data Dictionary Views



- [sthref1100]
- [sthref1102]
- [sthref1608]
- [sthref1610]
- [sthref1894]
- [sthref2538]
- [sthref2385]
- [sthref2382]
- [sthref2523]
- [sthref2579]
- [sthref2723]
- [sthref2725]
- [sthref2727]
- [sthref2736]
- [sthref2738]
- [sthref3518]
- [sthref3655]

19.5.3 Verzeichnis der relevanten vordefinierten Rollen



- [DBSEG99887]

19.6 Übungen - Lokale Benutzerverwaltung

1. Erstellen Sie den Nutzer ALICE mit dem Passwort „Tqbfjotld1“. Der Default Tablespace für Alice soll EXAMPLE sein.

2. Welchen temporären Tablespace nutzt alice jetzt und welcher View können Sie diese Angabe entnehmen?

3. Erstellen Sie den Nutzer BOB mit dem Passwort „Pass1/gH,3word“.

4. Welchen Default Tablespace benutzt BOB und aus welcher View können Sie diese Angabe entnehmen?

5. Ändern Sie für den Nutzer ALICE den Default Tablespace auf USERS!

6. Verändern Sie den Nutzer BOB wie folgt:

- Default Tablespace: EXAMPLE
- Temporary Tablespace: TEMP
- Quota auf USERS: 25 M

7. Sperren Sie den Account des Nutzers ALICE.

8. Richten Sie den Nutzer CHLOE so ein, dass dieser sich mit sysdba-Privilegien an der Datenbank anmelden kann (Passwort: password). Für CHLOE gelten die Standardeinstellungen.

9. Welche View zeigt Ihnen, dass CHLOE tatsächlich sysdba-Privilegien hat?

Sollten Sie noch Zeit haben, können Sie sich jetzt mit den folgenden Aufgaben befassen!



10. Erstellen Sie das Nutzerprofil P_CLERK. In diesem Profil müssen die folgenden Angaben festgelegt werden:
 - Anzahl fehlerhafter Anmeldungen: 3 Versuche
 - Gültigkeitsdauer des Passworts: 15 Tage
 - Passworthistorie: 3 Stück
 - Dauer der Passwortsperre: 10 Minuten
 11. Weisen Sie den Nutzern ALICE, BOB und CHLOE das Profil P_CLERK zu und testen Sie die Auswirkungen!
 12. Fügen Sie dem Profil P_CLERK den Parameter password_grace_time mit dem Wert 5 Tage hinzu!
 13. Ändern Sie die Sperrdauer des Accounts im Profil P_CLERK auf unbegrenzt!
 14. Löschen Sie das Nutzerprofil P_CLERK in einem Arbeitsschritt!
 15. Ermöglichen Sie es dem Nutzer ALICE Tabellen im Tablespace EXAMPLE anzulegen. Sie darf nur innerhalb ihres eigenen Schemas Tabellen anlegen.
-
-
16. Richten Sie das in Abbildung ?? gezeigte Rollenkonzept ein. Ersetzen Sie dabei NUTZER A durch ALICE, NUTZER B durch BOB und NUTZER C durch CHLOE.
 17. Unternehmen Sie alle notwendigen Schritte, um dem Nutzer ALICE die Erstellung der View V_DEPARTMENTS_IN_USA, die auf der Tabelle DEPARTMENTS des Nutzers HR basiert zu ermöglichen.
- Listing 19.56: Die View V_DEPARTMENTS_IN_USA

```
SELECT department_name, street_address, postal_code, city, state_province
FROM   hr.departments d INNER JOIN hr.locations l
      ON (d.location_id = l.location_id)
WHERE  l.country_id = 'US';
```
18. Melden Sie sich als Nutzer CHLOE an. Welche Privilegien gelten für CHLOE in Ihrer aktuellen Session?

19.7 Lösungen - Lokale Benutzerverwaltung

1. Erstellen Sie den Nutzer ALICE mit dem Passwort „Tqbfjotld1“. Der Default Tablespace für Alice soll EXAMPLE sein.

```
SQL> CREATE USER      alice
  2 IDENTIFIED BY      Tqbfjotld1
  3 DEFAULT TABLESPACE example;
```

2. Welchen temporären Tablespace nutzt alice jetzt und welcher View können Sie diese Angabe entnehmen?

```
SQL> SELECT username , temporary_tablespace
  2 FROM    dba_users
  3 WHERE   username LIKE 'ALICE';

USERNAME          TEMPORARY_TABLESPACE
-----
ALICE             TEMP
```

3. Erstellen Sie den Nutzer BOB mit dem Passwort „Pass1/gH,3word“.

```
SQL> CREATE USER      bob
  2 IDENTIFIED BY "Pass1/gH,3word";
```

4. Welchen Default Tablespace benutzt BOB und aus welcher View können Sie diese Angabe entnehmen?

```
SQL> SELECT username , default_tablespace
  2 FROM    dba_users
  3 WHERE   username LIKE 'BOB';

USERNAME          DEFAULT_TABLESPACE
-----
BOB              USERS
```

5. Ändern Sie für den Nutzer ALICE den Default Tablespace auf USERS!

```
SQL> ALTER USER      alice
  2 DEFAULT TABLESPACE users;
```

6. Verändern Sie den Nutzer BOB wie folgt:

- Default Tablespace: EXAMPLE

- Temporary Tablespace: TEMP
- Quota auf USERS: 25 M

```
SQL> ALTER USER bob
  2  DEFAULT TABLESPACE example
  3  TEMPORARY TABLESPACE temp
  4  QUOTA 25 M ON users;
```

7. Sperren Sie den Account des Nutzers ALICE.

```
SQL> ALTER USER alice
  2  ACCOUNT LOCK;
```

8. Richten Sie den Nutzer CHLOE so ein, dass dieser sich mit sysdba-Privilegien an der Datenbank anmelden kann (Passwort: password). Für CHLOE gelten die Standardeinstellungen.

```
SQL> CREATE USER chloe
  2 IDENTIFIED BY password;

SQL> GRANT sysdba TO chloe;
```

9. Welche View zeigt Ihnen, dass CHLOE tatsächlich sysdba-Privilegien hat?

```
SQL> SELECT *
  2  FROM v$pwfile_users;

USERNAME          SYSDB  SYSOP  SYSAS
-----
SYS              TRUE   TRUE   FALSE
CHLOE            TRUE   FALSE  FALSE
```

10. Erstellen Sie das Nutzerprofil P_CLERK. In diesem Profil müssen die folgenden Angaben festgelegt werden:

- Anzahl fehlerhafter Anmeldungen: 3 Versuche
- Gültigkeitsdauer des Passworts: 15 Tage
- Passworthistorie: 3 Stück
- Dauer der Passwortsperre: 10 Minuten

```
SQL> CREATE PROFILE p_clerk
  2  LIMIT
  3    FAILED_LOGIN_ATTEMPTS      3
  4    PASSWORD_LIFE_TIME        15
  5    PASSWORD_REUSE_MAX        3
  6    PASSWORD_LOCK_TIME       10 / 1440;
```

11. Weisen Sie den Nutzern ALICE, BOB und CHLOE das Profil P_CLERK zu und testen Sie die Auswirkungen!

```
SQL> ALTER USER alice
  2  PROFILE      p_clerk;

SQL> ALTER USER bob
  2  PROFILE      p_clerk;

SQL> ALTER USER chloe
  2  PROFILE      p_clerk

SQL> connect bob/wrong_password
ERROR:
ORA-01017: invalid username/password; logon denied

SQL> connect bob/wrong_password
ERROR:
ORA-01017: invalid username/password; logon denied

SQL> connect bob/wrong_password
ERROR:
ORA-01017: invalid username/password; logon denied

SQL> connect bob/wrong_password
ERROR:
ORA-28000: the account is locked
```

12. Fügen Sie dem Profil P_CLERK den Parameter password_grace_time mit dem Wert 5 Tage hinzu!

```
SQL> ALTER PROFILE p_clerk
  2  LIMIT
  3    PASSWORD_GRACE_TIME 5;
```

13. Ändern Sie die Sperrdauer des Accounts im Profil P_CLERK auf unbegrenzt!

```
SQL> ALTER PROFILE p_clerk
  2  LIMIT
  3    PASSWORD_LOCK_TIME unlimited;
```

14. Löschen Sie das Nutzerprofil P_CLERK in einem Arbeitsschritt!

```
SQL> DROP PROFILE p_clerk CASCADE;
```

15. Ermöglichen Sie es dem Nutzer ALICE Tabellen im Tablespace EXAMPLE anzulegen. Sie darf nur innerhalb ihres eigenen Schemas Tabellen anlegen.

```
SQL> ALTER USER alice
  2 QUOTA      10 M ON example;

SQL> GRANT create table, create session
  2 TO alice;
```

16. Richten Sie das in Abbildung ?? gezeigte Rollenkonzept ein. Ersetzen Sie dabei NUTZER A durch ALICE, NUTZER B durch BOB und NUTZER C durch CHLOE.

```
SQL> CREATE ROLE r_manager;
SQL> CREATE ROLE r_clerk;
SQL> CREATE ROLE r_departments;
SQL> CREATE ROLE r_connect;
SQL> CREATE ROLE r_employees;
SQL> GRANT select ON hr.departments
  2 TO r_departments;
SQL> GRANT create session
  2 TO r_connect;
SQL> GRANT select, update, insert ON hr.employees
  2 TO r_employees;
SQL> GRANT r_departments
  2 TO r_manager;
SQL> GRANT r_connect
  2 TO r_manager, r_clerk;
SQL> GRANT r_employees
  2 TO r_clerk;
SQL> GRANT r_manager
  2 TO alice;
SQL> GRANT r_clerk
  2 TO bob, chloe
```

17. Unternehmen Sie alle notwendigen Schritte, um dem Nutzer ALICE die Erstellung der View V_DEPARTMENTS_IN_USA, die auf der Tabelle DEPARTMENTS des Nutzers HR basiert zu ermöglichen.

Listing 19.57: Die View V_DEPARTMENTS_IN_USA

```
SELECT department_name, street_address, postal_code, city, state_province
FROM   hr.departments d INNER JOIN hr.locations l
       ON (d.location_id = l.location_id)
WHERE  l.country_id = 'US';
```

```
SQL> GRANT create view
  2 TO alice;

SQL> GRANT select ON hr.departments
```

```
2 TO      alice;
SQL> GRANT select ON hr.locations
2 TO      alice;
```

18. Melden Sie sich als Nutzer CHLOE an. Welche Privilegien gelten für CHLOE in Ihrer aktuellen Session?

```
SQL> SELECT *
  2  FROM    session_privs;

PRIVILEGE
-----
CREATE SESSION
```


20 Schemaobjekte verwalten

Inhaltsangabe

20.1 Tabellen

20.1.1 Einführung

Oracle kennt mehrere verschiedene Arten von Tabellen.

- Heap-Organized Tables (Standardtabellen)
- Index-Organized Tables
- Partitioned Tables
- Temporary Tables

Heap-Organized Tables

Heap-Organized Tables sind die Basisdatenstruktur einer Oracle 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. Dies ist vergleichbar mit einem „Haufen“ (engl. Heap) bei dem alle Elemente einfach aufeinander geworfen werden.

Abb. 20.1:
Heap-Organized
Table

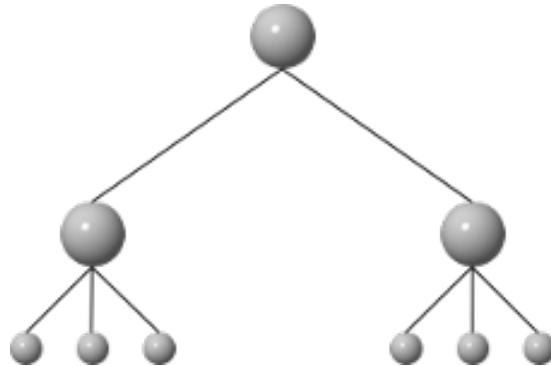


Abbildung ?? zeigt, dass die Tabellenzeilen, hier durch rote, blaue, grüne und gelbe Kästchen dargestellt, einfach nacheinander abgelegt werden.

Index-Organized-Tables

In einer Index-Organized Table dagegen, werden die Daten als sortierte Baumstruktur abgelegt.

Abb. 20.2:
Index-Organized-Table



Die unterste Ebene dieses Baumes, auch „Leaf Level“ genannt, dient zur Ablage der Daten. Die oberen Beiden können als „Inhaltsverzeichnis“ oder „Stichwortverzeichnis“ verstanden werden. Dort werden Metadaten gehalten, mit deren Hilfe, die Daten in der Leaf Ebene aufgefunden werden können. Nachteilig ist, dass der Baum bei jedem Änderungsvorgang gepflegt werden muss. Daraus folgt, dass eine solche Tabelle besonders da geeignet ist, wo hohe Lesegeschwindigkeit gefordert und ein geringes Änderungsvolumen vorhanden ist.

Partitioned Tables

Wie bereits bekannt, werden Tabellen in Form von Segmenten in einer Datendatei abgelegt. Wächst eine Tabelle stark an, vergrößert sich folglich auch ihr Segment. Dadurch wird mit der Zeit das Arbeiten mit den betroffenen Daten sehr langsam. Um bei solch großen Tabellen Abhilfe zu schaffen, kann eine Tabelle in mehrere Segmente, so genannte Partitionen, zerteilt werden. Dies ist 1:1 vergleichbar mit einer Festplatte, die in mehrere Partitionen aufgeteilt wird.

Ein großer Vorteil der Partitionierung ist, dass sie für Anwendungen völlig transparent ist. Das bedeutet, dass SQL-Statements in keiner Weise geändert werden müssen. Es sieht nach wie vor alles so aus, als wäre die Tabelle in „einem Stück“ gespeichert. Ein weiterer Vorteil ist, dass die Datenbank nicht mehr die komplette Tabelle verarbeiten muss, wenn ein Nutzer nur einen Teil der Daten abrufen muss (höhere Verarbeitungsgeschwindigkeit, aufgrund des niedrigeren Datenvolumens). Die Datenbank entscheidet selbstständig, auf welchen Partitionen gearbeitet werden muss.

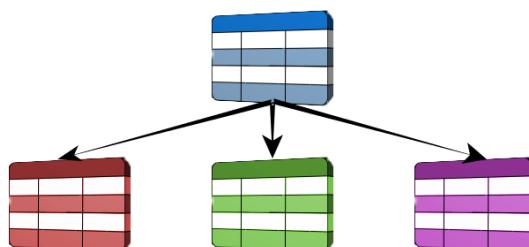


Abb. 20.3:
Partitionierte
Tabelle

Temporäre Tabellen

Wie der Name dieser Tabellenart bereits sagt, dienen sie zur temporären Aufnahme von Daten. Ihr Vorteil besteht darin, dass sie dynamisch wachsen und schrumpfen und keinerlei Redo-Informationen erzeugen. Dadurch sind sie ideal für Daten geeignet, die nur für kurze Zeit existieren.

20.1.2 Oracle 11g Advanced Compression

Oracle 11g Advanced Compression ist die Weiterentwicklung der Oracle Basic Compression Technologie, die seit Oracle 9i im Einsatz ist. Es ist nun möglich:

- DIRECT-Load-Operationen,
- DML-Operationen,
- Unstrukturierte Daten (LOBs),
- Backups und
- Networktraffic

zu komprimieren.

Die unter Oracle 9i eingeführte Basic Compression ermöglichte nur die transparente Komprimierung von Daten, die mittels eines Direct Path Load Vorganges (mittels SQL*Loader) in die Datenbank geschrieben wurden. Da aber die Masse der Daten mittels SQL oder des imp-tools geladen wurden, konnte die Kompression nur sehr selten zum Einsatz gebracht werden, was ihre Akzeptanz stark verringerte.



Die Advanced Compression ist eine lizenzi- und kostenpflichtige Zusatzoption für die Enterprise Edition der Datenbank!

Wie funktioniert die Kompression?

Die Kompression von Tabellendaten erfolgt, indem mehrfach auftretende Werte eliminiert werden. Oracle speichert komprimierte Daten in einem Block als „selbst erhaltende Daten“, d. h. alles was zur Rekonstruk-

tion der Daten benötigt wird, ist im gleichen Block gespeichert, in dem auch die Daten selbst abgelegt sind. Die Datenkompression ist transparent, was bedeutet, dass alle Features, die auf unkomprimierte Blöcke angewendet werden können, auch auf Komprimierte anwendbar sind.

Blockheader		
Tim	Peters	Uhlandstraße
Lina	Timme	Uhlandstraße
Udo	Krause	Gartenheimweg
Udo	Simon	Groperstraße

Abb. 20.4:
Ein
unkomprimierter
Datenblock

Der Aufbau eines unkomprimierten Datenblockes ist bereits bekannt. Nach dem Header, der alle Metainformationen zum Block enthält, werden die Nutzdaten eingetragen.

In einem komprimierten Block kommt eine neue Struktur hinzu, die „Symbol table“, die in Abbildung ?? blau dargestellt wird. Alle Werte, die vorher im unkomprimierten Block redundant waren, werden in ihr gespeichert. Im Datenbereich werden nur noch Zeiger (dargestellt durch rote X) abgelegt, die auf die jeweiligen Werte verweisen.

Blockheader		
Udo		Uhlandstraße
Tim	Peters	X
Lina	Timme	X
X	Krause	Gartenheimweg
X	Simon	Groperstraße

Abb. 20.5:
Ein
komprimierter
Datenblock



Zu beachten ist, dass sich die Kompression immer nur auf neue Datensätze auswirkt. Daten die schon vor dem Aktivieren der Kompression gespeichert waren, bleiben davon unberührt. Das bedeutet, dass ein Block sowohl komprimierte als auch unkomprimierte Datensätze enthalten kann.

Konfigurieren der Kompression

Die Komprimierung für Tabellen kann auf Ebene der Tablespace oder der, der Tabellen selbst konfiguriert werden. Wird sie auf Tablespace ebene konfiguriert, werden automatisch alle Tabellen in diesem Tablespace komprimiert.

Listing 20.1: Kompression auf Tablespace ebene aktivieren

```
-- Basic Compression
SQL> CREATE TABLESPACE data
  2  DATAFILE '/u02/oradata/orcl/data01.dbf' SIZE 100M
```

```
3  DEFAULT  COMPRESS BASIC;

-- Advanced Compression
SQL> CREATE BIGFILE TABLESPACE hr_data
2  DATAFILE '/u02/oradata/orcl/hr_data01.dbf' SIZE 500M
3  DEFAULT  COMPRESS FOR OLTP;
```



Die Advanced Compression kann in smallfile und in bigfile Tablespaces zum Einsatz kommen!

Auf Tabellenebene wird die Kompression ebenfalls mit der **COMPRESS**-Klausel aktiviert. Dies kann schon bei der Tabellenerstellung oder auch noch im Nachhinein erfolgen.

Listing 20.2: Eine Tabelle mit OLTP Compression erstellen

```
SQL> CREATE TABLE buchung_hist (
  2  Buchungs_ID          NUMBER,
  3  Betrag                NUMBER(12, 2),
  4  Buchungsdatum        DATE,
  5  Konto_ID              NUMBER NOT NULL,
  6  Transaktions_ID      NUMBER NOT NULL,
  7  Archivierungsdatum   DATE NOT NULL,
  8  CONSTRAINT buchung_hist_pk PRIMARY KEY (Buchungs_ID)
  9 )
10 COMPRESS FOR OLTP;
```

Die Tabelle BUCHUNG_HIST ist so konfiguriert, dass von Anfang an, alle Datensätze komprimiert werden. Da es sich hier um eine komprimierte Tabelle handelt, kann der View DBA_TABLES entnommen werden.

Listing 20.3: Eine Tabelle mit OLTP Compression erstellen

```
SQL> SELECT table_name, compression, compress_for
  2  FROM dba_tables
  3  WHERE table_name LIKE 'BUCHUNG_HIST';

TABLE_NAME           COMPRESS COMPRESS_FOR
-----  -----
BUCHUNG_HIST          ENABLED    OLTP
```

Existiert die Tabelle bereits und die Kompression soll nachträglich genutzt werden, gibt es zwei Möglichkeiten:

- Aktivieren der Kompression (nur neue Datensätze sind betroffen)
- Die gesamte Tabelle komprimieren (alte Datensätze werden ebenfalls komprimiert)

Listing 20.4: Nachträgliches aktivieren der OLTP Kompression

```
SQL> ALTER TABLE buchung COMPRESS FOR OLTP;
```

Soll die gesamte Tabelle komprimiert werden, muss sie neu aufgebaut werden. Dies geschieht mit der **MOVE**-Klausel. Diese sorgt, in Verbindung mit der **COMPRESS**-Klausel dafür, dass die Tabelle reorganisiert/defragmentiert und komprimiert wird.

Listing 20.5: Komprimieren einer Tabelle

```
SQL> ALTER TABLE buchung MOVE COMPRESS FOR OLTP;
```



Die Reorganisation einer Tabelle ist sehr ressourcenintensiv und sollte daher nur bei niedriger Grundlast im System durchgeführt werden.

Muss die Kompression aus irgendeinem Grund wieder deaktiviert werden, geschieht dies mit der **NOCOMPRESS**-Klausel.

Listing 20.6: Deaktivieren der Kompression

```
SQL> ALTER TABLE buchung NOCOMPRESS;
```

Listing 20.7: Dekomprimieren einer Tabelle

```
SQL> ALTER TABLE buchung MOVE NOCOMPRESS;
```

Ein Experiment

Im nachfolgenden Experiment soll gezeigt werden, wie sich die Kompression auf den Platzbedarf einer Tabelle auswirkt.

Im ersten Schritt wird die Anzahl der Datenblöcke und die Anzahl der Tabellenzeilen pro Datenblock, für die unkomprimierte Tabelle BUCHUNG ermittelt.

Listing 20.8: Anzahl der Datenblöcke + durchschnittliche Anzahl der Zeilen pro Block

```
col segment_name format a20
SQL> SELECT segment_name, blocks
  2  FROM dba_segments
  3 WHERE segment_name LIKE 'BUCHUNG';

SEGMENT_NAME          BLOCKS
-----
BUCHUNG                2176

SQL> SELECT AVG(Anzahl) As Mittelwert
  2  FROM (SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) AS Block_ID,
  3             COUNT(*) AS Anzahl
  4           FROM Buchung
  5          GROUP BY DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid));

MITTELWERT
-----
405,120428
```

Gemäß der Abfrage aus Beispiel ?? besteht die Tabelle BUCHUNG aus 2176 Datenblöcken mit durchschnittlich 405 Zeilen pro Datenblock.

Der zweite Schritt besteht darin, die Tabelle BUCHUNG zu komprimieren.

Listing 20.9: Komprimieren der Tabelle BUCHUNG

```
SQL> ALTER TABLE buchung MOVE COMPRESS FOR OLTP;
```

Zu guter Letzt, wird nun die gleiche Statistik, wie in Beispiel ?? erhoben.

Listing 20.10: Anzahl der Datenblöcke + durchschnittliche Anzahl der Zeilen pro Block nach der Kompression

```
col segment_name format a20
SQL> SELECT segment_name, blocks
  2  FROM dba_segments
  3 WHERE segment_name LIKE 'BUCHUNG';

SEGMENT_NAME          BLOCKS
-----
BUCHUNG                2048

SQL> SELECT AVG(Anzahl) AS Mittelwert
  2  FROM (SELECT DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid) AS Block_ID,
  3                  COUNT(*) AS Anzahl
  4            FROM Buchung
  5           GROUP BY DBMS_ROWID.ROWID_BLOCK_NUMBER(rowid));

MITTELWERT
-----
445,672228
```

Die neue Statistik beweist es. Durch die Kompression besteht die Tabelle BUCHUNG aus ca. 6 % weniger Datenblöcken und es werden im arithmetischen Mittel 10 % mehr Tabellenzeilen pro Datenblock gespeichert.



Durch die Nutzung von komprimierten Tabellen wird der Platzbedarf an Arbeitsspeicher und Speicher auf dem Datenträger reduziert. Daraus resultiert oft eine bessere Performance für Leseoperationen, die Kosten für Schreiboperationen steigen aber.

20.1.3 Tabellen reorganisieren

Eine Tabelle wird reorganisiert, indem sie in ein neues Segment verschoben wird. Dies kann auf zwei unterschiedliche Arten geschehen:

- Die Tabelle wird in ein neues Segment, im gleichen Tablespace, verschoben.

- Die Tabelle wird in ein neues Segment, in einem anderen Tablespace, verschoben.

Hierzu einige Beispiele:

Listing 20.11: Reorganisieren der Tabelle BUCHUNG

```
SQL> ALTER TABLE buchung MOVE;
```

Listing 20.12: Verschieben einer Tabelle in einen anderen Tablespace

```
SQL> ALTER TABLE buchung MOVE  
2 TABLESPACE users;
```

Das Verschieben einer Tabelle hat zur Konsequenz, dass die RowIDs aller Tabellenzeilen verändert werden. Daraus resultiert:

- Das alle Indizes, die auf der Tabelle liegen, ungültig werden und neu erstellt oder reorganisiert werden müssen.
- Während des Verschiebevorganges sind keine DML-Operationen auf der Tabelle möglich.
- Es werden alle Statistiken für das automatische Performance Tuning ungültig und müssen neu gesammelt werden.

Dies zeigt, dass das Verschieben einer Tabelle eine sehr teure Angelegenheit ist und gut überlegt sein sollte.

20.1.4 Informationen über Tabellen sammeln



- [ADMIN015]

20.2 Indizes

Indizes sind optionale Strukturen, die mit einer Tabelle verbunden sind. 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 Spaltenkombinationen für die Indizes unterscheiden. Dabei kann eine Spalte in mehreren Kombinationen vorkommen und ein Index kann sich über mehrere Spalten erstrecken.

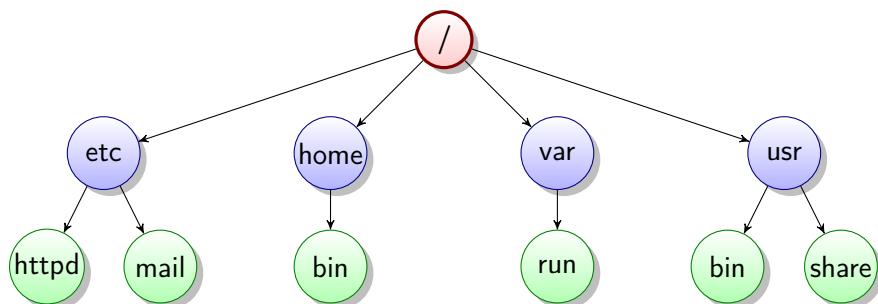
Oracle stellt die unterschiedlichsten Arten von Indizes bereit:

- **B-Baum** Index: Die meist genutzte Variante (Standard)
- **Reverse key** Index: Werden hauptsächlich im Oracle Real Application Cluster genutzt
- **Bitmap** Index: Sehr kompakte Variante, die am besten auf Spalten mit sehr wenigen Werten funktioniert
- **Function-based** Index: Enthalten den berechneten Ergebniswert einer Funktion

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.

20.2.1 Bäume in der Informatik

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.



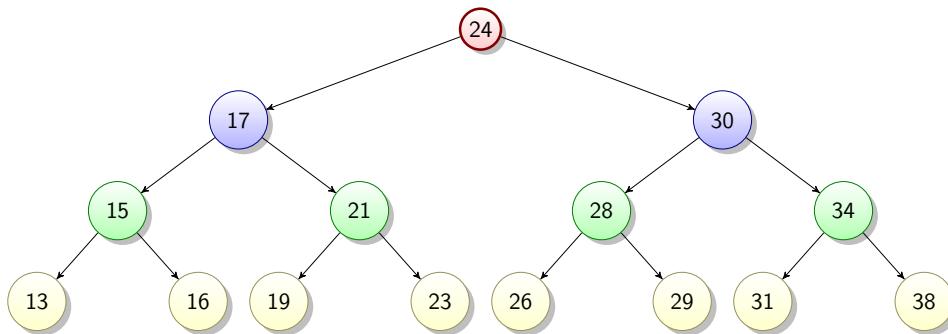
Die Abbildung zeigt den Unix-Verzeichnisbaum, anhand dessen einige Merkmale eines Baumes erkennbar sind.

- Jedes Element eines Baumes wird als „Knoten“ bezeichnet.
- Die Verbindungslinien/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).

Wichtige Eigenschaften von Bäumen am Beispiel des Binärbaumes

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.



Wichtige Eigenschaften eines Baumes sind:

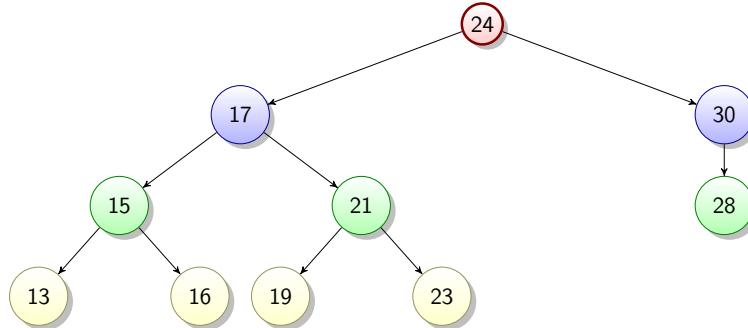
- **Tiefe:** Die Tiefe eines Knotens ist sein Abstand zur Wurzel (Anzahl der Kanten zwischen ihm und der Wurzel).
- **Ebene/Level:** Die Menge aller Knoten der gleichen Tiefe wird als Ebene bzw. Level bezeichnet.
- **Baumhöhe:** Die Höhe eines Baumes wird durch die maximale Tiefe, die ein Knoten erreichen 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: $n = 2^{h+1} - 1$.

$$2^{h+1} - 1 = n \quad | + 1 \quad (20.1)$$

$$2^{h+1} = n + 1 \quad | \log_2 \quad (20.2)$$

$$h + 1 = \log_2(n + 1) \quad | - 1 \quad (20.3)$$

$$h = \log_2(n + 1) - 1 \quad = \log_2(n) \text{ abgerundet} \quad (20.4)$$

Das $\log_2(n + 1) - 1 = \log_2(n) \text{ abgerundet}$ gilt lässt sich beweisen:

$$n = 15 \quad (20.5)$$

$$\log_2(15 + 1) - 1 = 3 \quad (20.6)$$

$$\log_2(15) = 3.91 (\text{Abrunden!}) \quad (20.7)$$



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.

20.2.2 B*-Baum Indizes

Aufbau und Funktionsweise

B*-Bäume (gesprochen B-Stern Baum) sind eine vielfach weiterentwickelte Variante der Binäräbäume. Urheber dieser Baumart ist Donald Ervin Knuth¹. Einer der wesentlichsten Unterschiede zu den Binäräbäumen 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.

B*-Baum Indizes sind die häufigste Index-Variante in einer Datenbank. Sie bestehen aus einer Reihe von Oracle-Blöcken und enthalten die indizierten Werte in sortierter Reihenfolge. Jedem 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 Oracle die RowID als Wert und den Namen der jeweiligen Bank als Schlüssel ab. Da mittels der RowID eine Tabellenzeile direkt aus einer Datendatei gelesen werden kann, bedeutet dies, dass nach dem Auffinden des Schlüsselwertes direkt die gewünschte Zeile gelesen wird.



Die RowID stellt in einem B*-Baum-Index das Hilfsmittel zum Auffinden der Tabellenzeilen dar. Die indizierten Tabellenspalten werden als Schlüssel benutzt.

Abbildung ?? zeigt beispielhaft, wie ein B*-Baum Index aussehen könnte.

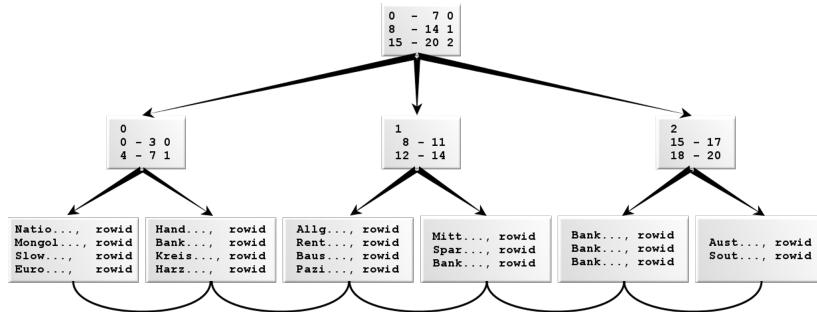


Abb. 20.6:
Ein B*-Baum
Index

Der Rootnode und die Branchnodes enthalten die Schlüsselwerte, in der Form: von - bis und die Adresse des dazugehörigen Indexblockes. 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).

¹Donald Ervin Knuth: US-Amerikanischer Informatiker und Professor an der Stanford University. Autor der Buchserie: „The Art of Computer Programming“ und Erfinder des Textsatzsystems „TeX“

In den Leafnodes stehen dann die indizierten Schlüsselwerte und die jeweilige RowID. 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.

B*-Tree Indizes erstellen

B*-Tree Indizes werden in Oracle mit dem Kommando `CREATE INDEX` erstellt.

Listing 20.13: Einen B*-Tree Index erstellen

```
SQL> CREATE INDEX idx_transactions
  2  ON bank.buchung(buchungs_ID, transaktions_ID)
  3  TABLESPACE bank;
```

Das SQL-Statement aus Beispiel ?? erstellt den Index `IDX_TRANSACTIONS` auf den beiden Spalten `BUCHUNGS_ID` und `TRANSAKTIONS_ID`. Die `TABLESPACE`-Klausel ist optional und beeinflusst wo der Index abgelegt wird.



Ein Index der auf eine Kombination von zwei oder mehr Spalten gelegt wird, wird als „Composite Index“ bezeichnet.

Um sich den soeben erstellten Index näher betrachten zu können, stellt Oracle die View `DBA_INDEXES` bereit.

Listing 20.14: Der Index unter der Lupe - Die View `dba_indexes`

```
SQL> col index_name format a20
SQL> col index_type format a10
SQL> col uniqueness format a10
SQL> col blevel format 99
SQL> col leaf_blocks format 9999999
SQL> SELECT index_name, index_type, blevel, leaf_blocks, uniqueness
  2  FROM dba_indexes
  3  WHERE index_name LIKE 'IDX_TRANSACTIONS';

INDEX_NAME      INDEX_TYPE    BLEVEL LEAF_BLOCKS UNIQUENESS
-----          -----        -
IDX_TRANSACTIONS      NORMAL           2          1325 NONUNIQUE
```

Die Spalte `INDEX_TYPE` gibt Auskunft darüber, welche Art von Index erstellt wurde. Die Angabe „normal“ besagt, dass es sich um einen B*-Tree Index handelt. Aus den Spalten `BLEVEL` (B*-Tree-Level) und `LEAF_BLOCKS` geht hervor, dass der Baum eine Höhe von 2 hat und 1325 Leafnodes besitzt.

Die interessanteste Spalte, dürfte die Spalte **UNIQUENESS** sein. Sie zeigt den Wert „**nonunique**“, der besagt, dass die Werte im Index nicht eindeutig sind. Da ein B*-Tree nur mit eindeutigen Schlüsselwerten funktionieren kann, muss Oracle an dieser Stelle einen Trick anwenden.

Der Trick besteht darin, die RowID als zusätzliche Indexspalte mit abzulegen. Das heißt, obwohl der Index auf nur zwei Spalten gelegt wurde, wird intern die RowID als dritte Indexspalte geführt. Dadurch werden zum einen alle Einträge eindeutig und können zum anderen nach der RowID sortiert werden.

Bei einem Unique Index ist ein solcher Trick nicht nötig. Hier werden nur die Schlüsselspalten gespeichert. Die RowID wird als eigenständiges Attribut zu den Schlüsselspalten abgelegt.



Letztlich ist der einzige Unterschied zwischen einem Unique Index und einen Nonunique Index das interne Format, in dem die Daten abgespeichert werden.

Um einen Unique Index zu erstellen, muss dem **CREATE INDEX**-Kommando noch das Schlüsselwort **UNIQUE** hinzugefügt werden.

Listing 20.15: Einen B*-Tree Index erstellen

```
SQL> CREATE UNIQUE INDEX idx_sozversnr
  2  ON bank.mitarbeiter(sozversnr)
  3  TABLESPACE bank;
```

Eine Abfrage auf **DBA_INDEXES** beweist, dass dieser Index Unique ist.

Listing 20.16: Ein Unique Index unter der Lupe - Die View dba_indexes - 2

```
SQL> col index_name format a20
SQL> col index_type format a10
SQL> col uniqueness format a10
SQL> col blevel format 99
SQL> col leaf_blocks format 999999
SQL> SELECT index_name, index_type, blevel, uniqueness
  2  FROM    dba_indexes
  3  WHERE   index_name LIKE 'IDX_SOZVERSNR';

INDEX_NAME      INDEX_TYPE     BLEVEL UNIQUENESS
-----          -----        -----
IDX_SOZVERSNR      NORMAL          0      UNIQUE
```

Einen Index für ein Constraint erstellen

Primary Key und Unique Constraints benötigen einen eigenen Unique Index, um arbeitsfähig zu sein. Dieser Index wird, beim Anlegen des Constraints, automatisch mit erstellt. Der DBA hat jedoch die Möglichkeit, einen Index selbst anzulegen und diesen mit dem Constraint zu verbinden. Dies ist beispielsweise dann sehr nützlich, wenn der Index, getrennt von der Tabelle, in einem anderen Tablespace liegen soll.

Listing 20.17: Ein Unique Constraint mit einem Index verbinden

```
SQL> ALTER TABLE bank.mitarbeiter
  2 ADD CONSTRAINT sozversnr_uk UNIQUE (sozversnr)
  3 USING INDEX IDX_SOZVERSNR;
```

Der Quellcode aus den Zeilen eins und zwei ist bereits bekannt. Es wird der Tabelle MITARBEITER ein Unique-Constraint, auf der Spalte SOZVERSNR, hinzugefügt. Die `USING INDEX`-Klausel in Zeile drei, ist nun dafür verantwortlich, dass das Constraint SOZVERSNR_UK mit dem Unique Index IDX_SOZVERSNR verbunden wird.

Mit Hilfe dieser Klausel kann ein Index aber auch direkt, zusammen mit dem Constraint angelegt werden.

Listing 20.18: Ein Unique Constraint zusammen mit einem Index erstellen

```
SQL> ALTER TABLE bank.bank
  2 ADD CONSTRAINT bank_name_uk UNIQUE (name)
  3 USING INDEX (
  4   CREATE UNIQUE INDEX bank.bank_name_uk
  5   ON bank.bank(name));
```

Da die beiden Constraints tatsächlich mit den gewünschten Indizes verbunden sind, kann mittels der View DBA_CONSTRAINTS geprüft werden.

Listing 20.19: Die View DBA_CONSTRAINTS

```
SQL> SELECT constraint_name, constraint_type, index_name
  2 FROM dba_constraints
  3 WHERE constraint_name IN ('SOZVERSNR_UK', 'BANK_NAME_UK');

CONSTRAINT_NAME          C INDEX_NAME
-----
SOZVERSNR_UK             U IDX_SOZVERSNR
BANK_NAME_UK              U IDX_BANK
```

Index Clustering Factor

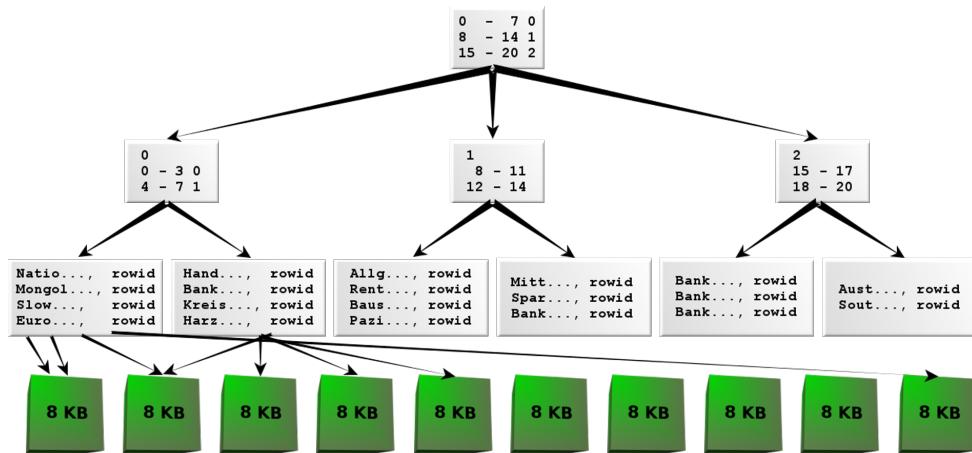
Der Index Clustering Factor ist ein Wert, der Auskunft darüber gibt, wie viele Leseoperationen benötigt werden, um die gesuchten Tabellenzeilen zu holen. Im Idealfall zeigen alle RowIDs eines Index-Leafnodes auf genau einen Tabellen block. Dies hätte einen Index Clustering Factor von 1 zur Folge. Im schlechtesten Fall, zeigt jede RowID eines Index-Leafnodes auf einen anderen Tabellen block. Dies würde dann mit einem sehr hohen Index Clustering Factor ausgedrückt werden.

Je höher der Index Clustering Factor, desto schlechter ist die Ausnutzung des Indizes möglich.



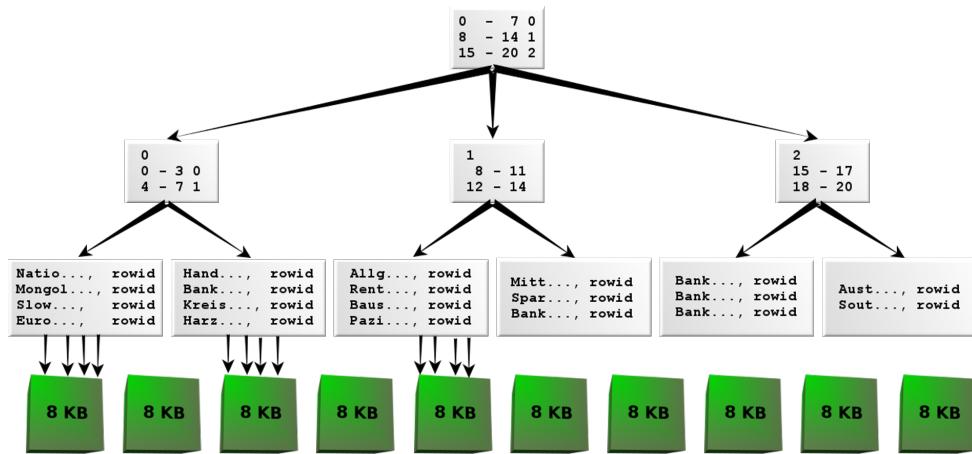
In Abbildung ?? ist, anhand der ersten beiden Indexblöcke, ein hoher Index Clustering Factor dargestellt. Nahezu jede RowID zeigt auf einen anderen Tabellen block. Sollten in diesem Zustand die vier Banken aus Index block Nummer 2 (zweiter von links) geholt werden, müsste Oracle vier Tabellenblöcke lesen, obwohl alle vier Zeilen auch in einem einzigen gespeichert werden könnten. Der Leseaufwand ist also viermal so hoch, wie nötig.

Abb. 20.7:
Ein hoher Index
Clustering Factor



In Abbildung ?? wird nun ein niedriger Clustering Factor gezeigt. Die ersten drei Indexblöcke benötigen jeweils nur einen Tabellen block, für alle Tabellenzeilen. Dies ist der theoretische Idealfall.

Abb. 20.8:
Ein sehr niedriger
Index Clustering
Factor



Der Clustering Factor kann mit Hilfe der View DBA_INDEXES abgefragt werden.

Listing 20.20: Den Index Clustering Factor anzeigen

INDEX_NAME	CLUSTERING_FACTOR
<hr/>	
BANK_PK	1
BANKFILIALE_PK	1
BANKKUNDE_PK	1
BUCHUNGS_PK	7857
DEPOT_PK	1

EIGENKUNDEN_PK	337
EIGENKUNDEKONTO_PK	832
EIGENKUNDEMitarbeiter_PK	1
FREMDKUNDEN_PK	1
FREMDKUNDEKONTO_PK	1
GIROKONTO_PK	2
INDEX_NAME	CLUSTERING_FACTOR
-----	-----
KONTO_PK	11
KUNDEN_PK	4
MITARBEITER_PK	2
SPARBUCH_PK	1

Beispiel ?? zeigt das drei der 15 Indizes aus dem BANK-Schema einen sehr hohen Clustering Factor haben (BUCHUNGS_PK, EIGENKUNDEN_PK und EIGENKUNDEKONTO_PK). Diese drei Indizes sollten neu aufgebaut werden, um den Clustering Factor zu verringern.

20.2.3 Reverse Key Indizes

Reverse Key Indizes sind eine Subvariante der B*-Tree Indizes. Ihre Besonderheit ist, dass sie die Bytes der Indexschlüsselspalten in umgekehrter Reihenfolge speichern. Durch das Speichern der Bytes in umgekehrter Reihenfolge wird erreicht, dass bei Einfügevorgängen die neuen Werte über den gesamten Index verteilt werden. Dadurch wird verhindert, dass viele Serverprozesse gleichzeitig auf ein und den selben Index block zugreifen, was bei einem normalen B*-Tree Index der Fall wäre.

Das folgende Beispiel soll die Funktionsweise, die Vorteile, aber auch die Nachteile von Reverse Key Indizes erläutern.

Für die Tabelle BUCHUNG wird auf der Spalte BUCHUNGS_ID ein neuer Reverse Key Index angelegt. Der bestehende B*-Tree Index wurde vorher gelöscht.

Listing 20.21: Einen Reverse Key Index anlegen

```
SQL> CREATE UNIQUE INDEX IDX_Buchung  
2   ON Buchung(Buchungs_ID)  
3   REVERSE;
```

Das Schlüsselwort **REVERSE**, in Zeile 3, macht aus einem normalen B*-Tree Index einen Reverse Key Index.



Nun werden 100 neue Datensätze, mit den Buchungs-IDs 469197 bis 469297 in die Tabelle eingefügt. Da es sich bei den Buchungs-IDs um fortlaufende Werte handelt, werden alle in den gleichen Index block eingefügt. Dieses Verhalten kann kritisch sein, wenn sehr viele Nutzer gleichzeitig Datensätze in eine Tabelle einfügen. Es entsteht eine große Konzentration von Schreibzugriffen auf einen Index block, was dazu führen kann, dass die Nutzer aufeinander warten müssen.

Um ein solches Warten zu verhindern, „dreht Oracle die Werte einfach um“. Das bedeutet, aus 469197 wird 791964 und aus 469198 wird 891964, usw. Somit werden Werte produziert, die nicht mehr fortlaufend sind und in unterschiedliche Indexblöcke eingetragen werden.

Der Vorteil ist, eine höhere Geschwindigkeit bei Schreibzugriffen, da die Nutzer nicht aufeinander warten müssen. Der Nachteil ist ein extrem hoher Index Clustering Factor, da keine Synchronisation zwischen Index block und Tabellen block erreicht werden kann.

20.2.4 Bitmap Indizes

Bitmap Indizes unterscheiden sich von B*-Tree und Reverse Key Indizes dahingehend, dass sie nicht als Baumstruktur, sondern als Bitmap angelegt werden. Ein solcher Index ist immer dann sinnvoll, wenn eine Tabellenspalte eine niedrige Kardinalität hat (wenig unterschiedliche Werte, z. B. „M“ für männlich und „W“ für weiblich) und nur wenige Änderungen an der Tabelle vorgenommen werden.

Eine Tabellenspalte, die diese Kriterien erfüllt, befindet sich in der Tabelle EIGENKUNDE. Es handelt sich um die Spalte STAATSANGEHOERIGKEIT. Sie kennt nur vier Werte: „Belgisch“, „Türkisch“, „Dänisch“ und „Deutsch“.

Um Suchvorgänge auf dieser Spalte zu beschleunigen, wird ein Bitmap Index angelegt.

Listing 20.22: Einen Bitmap Index anlegen

```
SQL> CREATE BITMAP INDEX idx_bm_staaatsangehoerigkeit
  2  ON eigenkunde(staatsangehoerigkeit);
```

Die View DBA_INDEXES zeigt, dass es sich bei IDX_BM_STAATSANGEHOERIGKEIT tatsächlich um einen Bitmap Index handelt.

Listing 20.23: Die Eigenschaften eines Bitmap Indexes

INDEX_NAME	INDEX_TYPE	BLEVEL	LEAF_BLOCKS
idx_bm_staaatsangehoerigkeit	BITMAP	1	1

```
SQL> col index_name format a30
SQL> col index_type format a15
SQL> SELECT index_name, index_type, blevel, leaf_blocks
  2  FROM    dba_indexes
  3  WHERE   index_name LIKE 'IDX_BM_STAATS%'
```

-----	-----	-----	-----	-----
IDX_BM_STAATSANGEHOERIGKEIT	BITMAP	0	1	-----

Die Angabe LEAF_BLOCKS = 1 sagt aus, dass der Bitmap Index aktuell aus nur einem Index block besteht.



Auch bei Bitmap Indizes werden die Indexblöcke als Leaf blocks bezeichnet!

Nützlich wird dieser Index, wenn z. B. eine Anfrage nach allen belgischen Kunden gestellt wird. In diesem Fall benutzt Oracle den erstellten Bitmap Index.

Belgisch	0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0
Dänisch	1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
Deutsch	0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0
Türkisch	0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0
<hr/>	
Ergebnis 0 0 1 0 0 0 0 0 1 1 0 1 0 1 1 0	

Abb. 20.9:
Benutzung eines
Bitmap Index

Abbildung ?? zeigt die schematische Darstellung eines Bitmap Index. Für jeden Wert der Spalte STAATS-ANGEHOERIGKEIT wird eine Bitmap erstellt, die von links nach rechts gelesen werden muss. Jedes Bit stellt eine Tabellenzeile dar. Eine 0 bedeutet, dass in dieser Zeile der betreffende Wert nicht steht. Laut Abbildung ?? steht also in Zeile 1 der Wert „Dänisch“ (eine 1 hinter dänisch). In der zweiten Spalte ist ein türkischer Staatsbürger erfasst und in Spalte drei ein Belgier. Oracle kann so sehr schnell und effizient die gewünschten Zeilen heraussuchen.

20.2.5 Visibility und Usability

Visibility

Seit Oracle 11g ist es möglich einen Index als sichtbar bzw. unsichtbar zu deklarieren. Sichtbare Indizes werden:

- vom SQL-Optimizer bei der Erstellung von Ausführungsplänen berücksichtigt,
- bei allen DML-Operationen mit gepflegt.

Dies gibt dem Admin die Möglichkeit, die Auswirkung eines Indizes auf einen Ausführungsplan zu testen, indem er ihn mit `ALTER INDEX` sichtbar bzw. unsichtbar macht.

Listing 20.24: Einen Index unsichtbar werden lassen

```
SQL> ALTER INDEX IDX_SOZVERSNR INVISIBLE;
```

Listing 20.25: Und so wird er wieder sichtbar

```
SQL> ALTER INDEX IDX_SOZVERSNR VISIBLE;
```

Die View DBA_INDEXES hilft dem Admin dabei, zu überprüfen, ob ein Index sichtbar oder unsichtbar ist.

Listing 20.26: Ist der Index sichtbar oder unsichtbar?

```
SQL> SELECT index_name, visibility
  2  FROM dba_indexes
  3  WHERE index_name LIKE 'IDX_SOZVERSNR'

INDEX_NAME          VISIBILITY
-----
IDX_SOZVERSNR      VISIBLE
```

Das folgende Beispiel zeigt, was passiert, wenn der Admin einen Index unsichtbar werden lässt.

Während der Index IDX_SOZVERSNR sichtbar ist, wird die folgende Abfrage ausgeführt.

Listing 20.27: Abfrage mit sichtbarem Index

```
SQL> SELECT vorname, nachname
  2  FROM bank.mitarbeiter
  3  WHERE sozversNr IN ('5679B983-694-22FA34D', '17211682-BA6-D9C0B30');
```

Das Ergebnis besteht aus zwei Zeilen. Mit Hilfe des `set autotrace traceonly`-Kommandos kann der Ausführungsplan, für dieses Statement, angezeigt werden.

Listing 20.28: Ausführungsplan für die Abfrage mit sichtbarem Index

```
SQL> set autotrace traceonly
SQL> SELECT vorname, nachname
  2  FROM Mitarbeiter
  3  WHERE sozversNr IN ('5679B983-694-22FA34D', '17211682-BA6-D9C0B30');

Execution Plan
-----
Plan hash value: 2743979585

-----
| Id | Operation           | Name        | Rows | Bytes | Cost (%CPU) | Time     |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT   |             | 2    | 72   | 2   (0) | 00:00:01 |
| 1 |  INLIST ITERATOR   |             |       |       |           |           |
| 2 |   TABLE ACCESS BY INDEX ROWID | MITARBEITER | 2    | 72   | 2   (0) | 00:00:01 |
```

```
| 3 | INDEX UNIQUE SCAN           | IDX_SOZVERSNR | 2 |      | 1   (0)|00:00:01|  
-----  
Predicate Information (identified by operation id):  
-----  
 3 - access("SOZVERSNR"='17211682-BA6-D9C0B30' OR  
           "SOZVERSNR"='5679B983-694-22FA34D')
```

Listing 20.29: Und so wird er wieder sichtbar - Fortsetzung

```
Statistics
-----
1 recursive calls
0 db block gets
4 consistent gets
0 physical reads
0 redo size
666 bytes sent via SQL*Net to client
524 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
2 rows processed
```

Interessant am Ausführungsplan aus Beispiel ?? sind die rot markierten Stellen. Die Angaben „TABLE ACCESS BY INDEX ROWID“ und „INDEX UNIQUE SCAN“ zeigen, dass der Index IDX_SOZVERSNR, für die Suche der gewünschten Ergebnissezeilen benutzt wurde. Die Angabe „4 consistent gets“ sagt aus, dass für die Suche nur vier Lesevorgänge benötigt wurden.

Nun wird der Index unsichtbar geschaltet und die Abfrage erneut ausgeführt.

Listing 20.30: Der Index wird unsichtbar und das Statement wird wiederholt

```
SQL> ALTER INDEX IDX_SOZVERSNR INVISIBLE;
SQL> SELECT vorname, nachname
  2  FROM Mitarbeiter
  3 WHERE sozversNr IN ('5679B983-694-22FA34D', '17211682-BA6-D9C0B30');

Execution Plan
-----
Plan hash value: 414804864

-----| Id | Operation          | Name      | Rows | Bytes | Cost (%CPU)| Time       |
-----| 0 | SELECT STATEMENT   |           |  2   |   72  |     3  (0) | 00:00:01  |
| 1 |  TABLE ACCESS FULL | MITARBEITER|  2   |   72  |     3  (0) | 00:00:01  |

-----| Predicate Information (identified by operation id): |
-----| 1 - filter("SOZVERSNR"='17211682-BA6-D9C0B30' OR
             "SOZVERSNR"='5679B983-694-22FA34D')|
```

Listing 20.31: Der Index wird unsichtbar und das Statement wird wiederholt - Fortsetzung

```
Statistics
-----
238 recursive calls
  0 db block gets
  56 consistent gets
  0 physical reads
  0 redo size
666 bytes sent via SQL*Net to client
524 bytes received via SQL*Net from client
  2 SQL*Net roundtrips to/from client
  6 sorts (memory)
  0 sorts (disk)
  2 rows processed
```

Der Ausführungsplan aus Beispiel ?? beweist, dass der unsichtbare Index nicht mehr benutzt wird. Als Suchmethode wurde ein „Full Table Scan“ genutzt, für den 56 Lesevorgänge benötigt wurden.

Um die Anzeige von Ausführungsplänen wieder zu deaktivieren, muss in SQL*Plus das Kommando `set autotrace off` eingegeben werden.

Usability

Es ist möglich, einen Index in Oracle nicht nur unsichtbar, sondern auch unbenutzbar werden zu lassen. Der Unterschied ist, dass der unsichtbare Index durch DML-Statements mit gepflegt wird, während der Unbenutzbare nicht gepflegt wird. Beide werden bei der Ausführung von `SELECT`-Anfragen nicht berücksichtigt.

Ein Index kann durch den Administrator oder die Datenbank selbst als unbenutzbar markiert werden. Häufig werden Indizes als unbenutzbar markiert, um große Data-Load Vorgänge zu beschleunigen.

Listing 20.32: Einen Index als unbenutzbar markieren

```
SQL> ALTER INDEX IDX_SOZVERSNR UNUSABLE;
```

Um zu überprüfen, in welchem Zustand sich ein Index befindet, kann wiederum die View DBA_INDEXES herangezogen werden.

Listing 20.33: Prüfen, ob ein Index nutzbar ist

```
SQL> SELECT index_name, visibility, status
  2  FROM dba_indexes
  3 WHERE index_name LIKE 'IDX_SOZVERSNR'
```

INDEX_NAME	VISIBILIT	STATUS
IDX_SOZVERSNR	INVISIBLE	UNUSABLE

Die Spalte STATUS zeigt, dass sich der Index in einem unbenutzbaren Zustand befindet. Dieser Zustand kann auf zwei Arten korrigiert werden:

- Den Index löschen und neu aufbauen
- Den Index reparieren (Index rebuild)

Den Index zu löschen und neu aufzubauen, würde bedeuten, dass evtl. Constraints (Unique oder Primary Key) mitgelöscht und neu erstellt werden müssten. Außerdem müsste der Admin die genauen Parameter des Indexes kennen, um ihn korrekt neu zu erstellen.

Bei der Methode „Index rebuild“ wird der noch vorhandene, aber unbenutzbare Index, mit den gespeicherten Parametern neu aufgebaut und Constraints bleiben erhalten.

Listing 20.34: Einen Index reparieren

```
SQL> ALTER INDEX IDX_SOZVERSNR REBUILD ONLINE;
```

Mit der Angabe `REBUILD ONLINE` wird der Index aufgebaut und die Tabelle bleibt für SQL-Abfragen offen. Eine erneute Abfrage der View DBA_INDEXES zeigt, dass der Index wieder benutzbar ist.

Listing 20.35: Der Index ist wieder benutzbar

INDEX_NAME	VISIBILIT	STATUS
IDX_SOZVERSNR	INVISIBLE	VALID

20.2.6 Indizes löschen

Beim Löschen eines Indizes werden alle durch den Index belegten Extents frei und können wieder verwendet werden.

Wie ein Index gelöscht werden kann, hängt davon ab, wie er angelegt wurde. Ein Index der explizit angelegt wurde, kann durch das Kommando `DROP INDEX` gelöscht werden. Bei einem Index der mit einem Primary

Key oder Unique Constraint mit erstellt wurde, muss das Constraint deaktiviert oder gelöscht werden, bevor der Index mit `DROP INDEX` gelöscht werden kann.

Listing 20.36: Einen Index Löschen

```
DROP INDEX idx_sozversnr;
```

20.3 Informationen

20.3.1 Verzeichnis der relevanten Initialisierungsparameter

In diesem Kapitel wurden keine Initialisierungsparameter angesprochen!

20.3.2 Verzeichnis der relevanten Data Dictionary Views

- [sthref2545]
- [sthref2528]
- [sthref2202]
- [sthref2191]



21 Transaktionen

Inhaltsangabe

Eine Transaktion ist eine logische Arbeitseinheit, die eines oder mehrere SQL-Statements enthält. Transaktionen sind in sich geschlossene Einheiten. Die Ergebnisse aller SQL-Statements einer Transaktionen können entweder in die Datenbank übernommen (committed) oder rückgängig gemacht (rolled back) werden.

Eine Transaktion beginnt implizit mit der ersten DML- oder DDL-Anweisung und endet mit einer der Anweisungen **COMMIT** (übernahme der Daten) oder **ROLLBACK** (rückgängig machen), bzw. implizit wenn eine DDL-Anweisung abgesetzt wird (auto commit).

Um das Konzept einer Transaktion bildlich darzustellen, stelle man sich die Datenbank eines Kreditinstitutes vor. Wenn ein Kunde Geld von seinem eigenen Konto auf ein anderes überweist, geschehen drei verschiedene Dinge:

1. Kontostand des Zahlenden herabsetzen
2. Kontostand des Zahlungsempfängers anpassen
3. Die Transaktion in einem Journal dokumentieren

Die Datenbank muss zwei verschiedene Fälle abdecken können:

1. Alle drei SQL-Statements können erfolgreich abgesetzt werden
2. Durch ein Problem kann mindestens eines der drei Statements nicht korrekt abgesetzt werden (falsche Kontonummer, Hardwarefehler, usw.)

Im ersten Fall muss die Datenbank die Änderungen der Transaktion in der Datenbank speichern, damit die Bankkonten der Kunden korrekt verwaltet werden. Tritt jedoch wie in Fall zwei ein Fehler auf, muss die gesamte Transaktion zurückgerollt werden.

21.1 Eigenschaften einer Transaktion (ACID)

Damit ein transaktionsbasiertes System funktionieren kann, müssen alle Transaktionen grundlegende Eigenschaften aufweisen. Diese können durch das Akronym „ACID“ beschrieben werden. ACID steht für „atomicity“, „consistency“, „isolation“ und „durability“. Im Deutschen wird statt ACID auch häufig AKID verwendet.

- **atomicity** (Atomarität): Eine Transaktion gilt als atomar, wenn Sie ganz oder gar nicht ausgeführt wird. Für den Benutzer muss es so aussehen, als wäre eine Transaktion eine einzelne elementare

Anweisung, die nicht unterbrochen werden kann. Da die einzelnen Anweisungen, aus denen sich eine Transaktion zusammensetzt, tatsächlich aber nacheinander ausgeführt werden müssen, muss im Falle dessen, dass die Transaktion nicht vollständig ausgeführt werden kann, jede einzelne Anweisung wieder rückgängig gemacht werden.

- **consistency** (Konsistenz): Konsistenz bedeutet, dass sich die Datenbank nach der Ausführung einer Transaktion in einem konsistenten Zustand befinden muss, davon ausgehend, dass die Datenbank auch vor der Transaktion schon konsistent war. Für die Konsistenz einer Datenbank sorgen die Integritäts Constraints, die bei der Ausführung einer Transaktion nicht verletzt werden dürfen.
- **isolation** (Isolation): Das Prinzip der Isolation bedeutet, dass parallel ausgeführte Transaktionen nicht sich nicht gegenseitig beeinflussen dürfen. Umgesetzt wird dies durch verschiedene Mechanismen, wie z. B. Sperren, Zeitstempelverfahren oder, im Falle von Oracle, das Multiversioning.
- **durability** (Dauerhaftigkeit): Das Ergebnis einer abgeschlossenen Transaktion muss dauerhaft in der Datenbank verfügbar sein, auch nach Systemabstürzen.

21.2 Transaktionen und ihre Phänomene

Die Isolation von Transaktionen ist eine der wesentlichen ACID-Eigenschaften, die an eine Transaktion gestellt werden. Fehlt die Isolation vollständig oder ist diese nur mangelhaft umgesetzt, können Probleme bei der Bearbeitung und dem Abfragen von Datensätzen auftreten. Diese Phänomene sind im ANSI/ISO SQL-Standard (SQL92) definiert.

21.2.1 Dirty Reads

Gründe für Dirty Reads sind:

- Das DBMS implementiert keine oder nur mangelhafte Isolation für Transaktionen.
- Konkurrierende Lese- und Schreibzugriffe

Tabelle 21.1: Dirty Reads

Transaktion 1	Transaktion 2
SELECT * FROM employees; t1	t2 UPDATE employees SET salary= 1000;

```
Anzeigen der Datensätze  
mit einem Gehalt von 1000    t3
                                |
                                t4  ROLLBACK;
```

Das vorangegangene Beispiel zeigt zwei konkurrierende Transaktionen. Transaktion 1 liest die Daten der Tabelle EMPLOYEES zum Zeitpunkt t1. Während Transaktion 1 noch liest, beginnt Transaktion 2 zum Zeitpunkt t2 diese Daten zu verändern. Zum Zeitpunkt t3 erfolgt für Transaktion 1 die Ausgabe der Daten.

Aufgrund nicht vorhandener Isolation werden auch die noch nicht bestätigten Änderungen von Transaktion 2 mit ausgegeben. Zum Zeitpunkt t4 macht die Transaktion 2 ihr **UPDATE**-Statement wieder rückgängig.

Die Schlussfolgerung aus diesem Szenario zeigt, dass Transaktion 1 zum Zeitpunkt t3 nicht committete Daten ausgegeben hat.



Dieses Szenario kann in Oracle nicht durchgeführt werden!

21.2.2 Non-Repeatable Reads

Non-Repeatable Reads sind ein Phänomen, dass immer dann auftritt, wenn:

- das DBMS keine oder nur mangelhafte Isolation für Transaktionen implementiert
- zwei gleiche Lesevorgänge *eines Datensatzes* in einer Transaktion unterschiedliche Ergebnisse liefern.

Tabelle 21.2: Non-Repeatable Reads

Transaktion 1	Transaktion 2
<pre>SELECT salary FROM employees WHERE employee_id = 100; t1 Anzeigen des Datensatzes (24000) t2</pre>	<pre>t3 UPDATE employees SET salary = 25000 WHERE employee_id = 100; COMMIT;</pre>
<pre>SELECT salary FROM employees</pre>	

WHERE employee_id = 100;	t4
Anzeigen des Datensatzes (25000)	t5

Transaktion 1 führt in diesem Beispiel zweimal den gleichen Lesevorgang durch. Dieser liefert zum Zeitpunkt t2 ein Gehalt von 24.000 \$. Bei Zeitpunkt t3 verändert Transaktion 2 die Basistabelle. Der erneute Lesevorgang liefert zum Zeitpunkt t5 ein Gehalt von 25.000 \$. Der gleiche Lesevorgang *ein und des selben Datensatzes* konnte also nicht zweimal mit dem gleichen Ergebnis durchgeführt werden, was als Non-Repeatable Read bezeichnet wird.

21.2.3 Phantom Reads

Phantom Reads treten immer dann auf, wenn:

- das DBMS nicht die höchst mögliche Isolation für Transaktionen implementiert und
- zwei gleiche Lesevorgänge in einer Transaktion eine *unterschiedliche Menge an Ergebniszellen* liefern. Dies bedeutet, dass zu den bereits gelesenen Zeilen neue hinzugekommen oder bestehende weggefallen sind, sich also die Menge der gelesenen Zeilen verändert hat.

Tabelle 21.3: Phantom Reads

Transaktion 1	Transaktion 2
SELECT * FROM employees	
WHERE department_id = 30; t1	
Anzeigen der Datensätze: 6 Stück t2	
	INSERT INTO employees
	t3 VALUES (...);
	COMMIT;
SELECT * FROM employees	
WHERE department_id = 30; t4	
Anzeigen der Datensätze: 7 Stück t5	

Transaktion 1 führt in diesem Beispiel zweimal den gleichen Lesevorgang aus. Lesevorgang 1 liefert zum Zeitpunkt t2 6 Zeilen. Zum Zeitpunkt t3 verändert Transaktion 2 diese Tabelle. Der erneute Lesevorgang von Transaktion 1 liefert jetzt, zum Zeitpunkt t5 ein verändertes Ergebnis. Es sind nun 7 Zeilen. Der gleiche Lesevorgang konnte also *nicht zweimal mit der gleichen Ergebnismenge* durchgeführt werden.



Der Unterschied zwischen Non-Repeatable Reads und Phantom Reads ist der, dass bei den Non-Repeatable Reads bestehende Datensätze verändert werden. Dadurch kann sich auch die Ergebnismenge ändern. Bei den Phantom Reads werden neue Datensätze hinzugefügt oder bestehende gelöscht. Auch hier wird die Ergebnismenge geändert, aber auf eine andere Art.

21.3 Transaktionslevel

Um die beschriebenen Transaktionsphänomene zu umgehen, wurden im ANSI/ISO SQL-Standard (SQL92) vier verschiedene Transaktionslevel festgelegt. Diese Level legen unterschiedliche Einschränkungen fest, um die genannten Phänomene zu unterdrücken.

Tabelle 21.4: Transaktionslevel gemäß SQL92-Standard

Isolationslevel	Dirty Reads	Non-Repeatable Reads	Phantom Reads
Read Uncommitted	Ja	Ja	Ja
Read Committed	Nein	Ja	Ja
Repeatable Read	Nein	Nein	Ja
Serializable	Nein	Nein	Nein

21.3.1 Read Uncommitted

Dies ist der Transaktionslevel mit den geringsten Einschränkungen (es gibt nämlich keine). Es findet keinerlei Isolation statt, so dass eine Transaktion unbestätigte Informationen einer anderen Transaktion lesen kann. Dieser Level ist in Oracle nicht implementiert!

21.3.2 Read Committed

Diese Stufe bringt die ersten Einschränkungen mit sich. Es können nur noch bestätigte Informationen anderer Transaktionen gelesen werden. Eine Transaktion wird also lediglich vor fehlerhaften Daten einer anderen Transaktion geschützt, da fehlerhafte Daten meist zurückgerollt werden. Dieser Level ist der Standard in Oracle.

21.3.3 Repeatable Read

Durch eine verbesserte Isolation der Transaktionen wird in diesem Level sichergestellt, dass auch das Phänomen der Non-Repeatable Reads verhindert werden kann. Dieser Level ist in Oracle nicht implementiert!

21.3.4 Serializable

Serializable ist der strengste Transaktionslevel. Er verhindert jegliche Transaktionsphänomene. Er tut dies allerdings auf Kosten der Performance, da, wie sein Name sagt, eine Serialisierung der einzelnen Transaktionen durchgeführt wird, d. h. Oracle versucht zwei parallel ausgeführte Transaktionen so auszuführen, als würden sie hintereinander ausgeführt.

Zusätzlich zu den Transaktionsleveln die der SQL92-Standard festgelegt hat, kennt Oracle noch einen weiteren Level, den Read Only Level. Wird eine Transaktion in diesem Level gestartet, kann sie nur Abfragen, aber kein DML Statement durchführen. Er hat die gleichen Auswirkungen wie der Level Serializable und ist für sehr lange laufende Abfragen gedacht, die ein hohes Maß an Leseconsistenz benötigen.

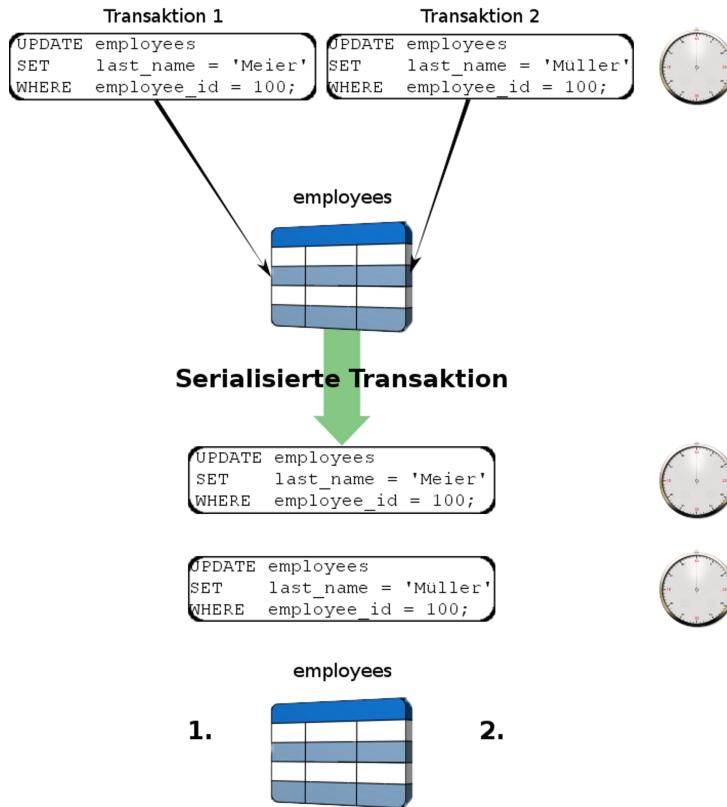


Abb. 21.1:
Serialisierung von
Transaktionen

21.4 Transaktionssteuerung

21.4.1 Eine Transaktion starten

Eine Transaktion kann auf zwei verschiedene Arten gestartet werden: implizit oder explizit. Implizit wird eine Transaktion durch ein beliebiges DML-Kommando gestartet. D. h. eine Transaktion beginnt implizit, sobald der Nutzer ein DML-Statement abgesetzt hat.

Um das Transaktionslevel einer Transaktion zu setzen, kann das `SET TRANSACTION ISOLATION LEVEL`-Kommando benutzt werden. Die drei möglichen Isolationslevel werden wie folgt gesetzt:

Listing 21.1: Isolationslevel einer Transaktion wählen

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;  
SET TRANSACTION READ ONLY;
```

Soll das Transaktionslevel für eine gesamte Session geändert werden, geschieht dies mit **ALTER SESSION**.

Listing 21.2: Isolationslevel einer Session wählen

```
ALTER SESSION SET isolation_level = READ COMMITTED;

ALTER SESSION SET isolation_level = SERIALIZABLE;
```

21.4.2 Eine Transaktion beenden

Eine Transaktion kann auf unterschiedliche Art und Weise beendet werden:

- Durch das Kommando **COMMIT**.

Ein **COMMIT** sorgt dafür, dass eine Transaktion beendet und ihre Änderungen in der Datenbank dauerhaft gemacht werden. Ein **COMMIT** kann nur dann erfolgreich sein, wenn keine Verletzung der Datenkonsistenz vorliegt.

- Durch das Kommando **ROLLBACK**.

Mit **ROLLBACK** werden alle Änderungen, die eine Transaktion an einer Datenbank vorgenommen hat, rückgängig gemacht. Die Datenbank wird in den letzten konsistenten Zustand zurückversetzt.

- Durch das Abbrechen einer Session.

Wird eine Session unerwartet abgebrochen, werden alle offenen Transaktionen der Session beendet. Es erfolgt kein **COMMIT**.

21.4.3 Transaktionsteile zurückrollen

Innerhalb einer Transaktion können Marken gesetzt werden, die als Savepoints bezeichnet werden. Dadurch wird eine Transaktion in einzelne Teile zerlegt. Der Nutzer hat so die Möglichkeit eine Transaktion nur teilweise, bis zu einem bestimmten Savepoint, zurückzurollen. Dies kann bei langen und komplexen Transaktion sehr nützlich sein.

Wird ein Rollback zu einem Savepoint durchgeführt, hebt Oracle nur die Sperren auf, die für die zurückgerollten Statements notwendig waren. Die Transaktion bleibt, trotz der teilweisen Rollbacks, erhalten. Andere Transaktionen, die Zugriff auf die bisher gesperrten Daten benötigen, können dann mit ihrer Arbeit fortfahren.

Listing 21.3: Einen Savepoint setzen

```
UPDATE departments
SET     department_ID = 11
WHERE   department_ID = 10;

SAVEPOINT dept;
```

Listing 21.4: Rollback zu einem Savepoint

```
UPDATE departments
SET     department_ID = 21
WHERE   department_ID = 20;

SAVEPOINT dept2;

ROLLBACK TO SAVEPOINT dept;
```



- [dataconcurrencyandconsistency].
- [transactionmanagement].

21.4.4 Deadlocks

Ein Deadlock tritt immer dann auf, wenn zwei oder mehr Nutzer Sperren auf ein und die selbe Ressource legen möchten. Deadlocks verhindern, dass die betreffenden Transaktionen weiterarbeiten können. Diese Situation wird deshalb als Deadlock bezeichnet, weil es egal ist, wie lange jede Transaktion warten würde, da jeder auf den anderen wartet.

Oracle kann automatisch Deadlock-Situationen erkennen und auflösen. Dies geschieht, in dem die Transaktion, die den Deadlock bemerkt auf Statementebene zurückgerollt wird (Statement-level rollback). So werden die betreffenden Sperren freigegeben und die andere Transaktion kann weiter arbeiten.



- [howoraclelocksdata]

21.5 Multiversion Concurrency Control

Multiversion Concurrency Control ist ein Mechanismus, den Oracle zur Gewährleistung der Lesekonsistenz nutzt. Dabei werden verschiedene Versionen von Datenbankobjekten aufbewahrt, so dass jeder Nutzer

die benötigte Sicht seiner Daten bekommt. Die Umsetzung dieses Verfahrens wird durch verschiedene Methoden, wie z. B. Zeitstempelverfahren oder Snapshots erreicht. Oracle benutzt hierfür sogenannte Before Images seiner Datenblöcke.

21.5.1 Lesekonsistenz

Lesekonsistenz auf Statementebene (Statementlevel Read Consistency)



Unter dem Begriff „Statementlevel Read Consistency“ versteht man, dass eine Abfrage nur die Daten sieht, die zum Startzeitpunkt der Abfrage gültig (committed) waren. Die Länge der Laufzeit der Abfrage darf dabei keine Rolle spielen.

Tabelle 21.5: Statementlevel Read Consistency

Transaktion 1	Transaktion 2
SELECT * FROM employees WHERE department_id = 30; t1	
	DELETE employees t2 WHERE employee_id = 117; COMMIT;
Anzeigen der Datensätze: 6 Stück t3	
SELECT * FROM employees WHERE department_id = 30; t4	
Anzeigen der Datensätze: 5 Stück t5	

Das Beispiel ?? zeigt die beiden Transaktionen 1 und 2. Transaktion 1 startet zum Zeitpunkt t1. Unmittelbar nach dem Start von Transaktion 1 startet Transaktion 2 zum Zeitpunkt t2. Der Delete-Vorgang von Transaktion 2 darf die Abfrage von Transaktion 1 nicht beeinflussen und die Ausgabe der Abfrageergebnisse zum Zeitpunkt t3 zeigt auch korrekte 6 Datensätze an. Erst die zum Zeitpunkt t4 gestartete Abfrage in Transaktion 1 registriert die durch Transaktion 2 vorgenommenen Änderungen. Die Lesekonsistenz auf Statementebene ist gewährleistet.

Lesekonsistenz auf Transaktionsebene (Transactionlevel Read Consistency)



Unter dem Begriff „Transactionlevel Read Consistency“ versteht man, dass eine Abfrage nur die Daten sieht, die zum Startzeitpunkt der Transaktion gültig (committed) waren. Die Länge der Laufzeit der Transaktion darf dabei keine Rolle spielen. Um Transactionlevel Read Consistency zu erwirken, muss das Isolationslevel Serializable verwendet werden.

Das Beispiel ?? zeigt die beiden Transaktionen 1 und 2. Transaktion 1 startet zum Zeitpunkt t1. Unmittelbar nach dem Start von Transaktion 1 startet Transaktion 2 zum Zeitpunkt t2. Der Delete-Vorgang von Transaktion 2 darf die gesamte Transaktion 1 nicht beeinflussen und die beiden Ausgaben der Abfrageergebnisse, zu den Zeitpunkten t4 und t6, zeigen auch korrekte 6 Datensätze an. Die Lesekonsistenz auf Transaktionsebene ist gewährleistet.

Tabelle 21.6: Transactionlevel Read Consistency

Transaktion 1	Transaktion 2	
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;	t1	
SELECT * FROM employees WHERE department_id = 30;	t2	
		DELETE employees t3 WHERE employee_id = 117; COMMIT;
Anzeigen der Datensätze: 6 Stück	t4	
SELECT * FROM employees WHERE department_id = 30;	t5	
Anzeigen der Datensätze: 6 Stück	t6	

21.5.2 Undo-Segmente

Um die Lesekonsistenz für Abfragen zu gewährleisten, benutzt Oracle Undo-Segmente. Undo-Segmente sind spezielle Segmente, die anders als z. B. Tabellensegmente oder Clustersegmente, nicht direkt durch den Nutzer bearbeitet werden können. Seit Oracle 9i sind alle Undo-Segmente in einem Undo-Tablespace zusammengefasst, der nur minimale Administration benötigt.

Undo-Segmente werden von der Datenbank genutzt, um Before Images von Datenblöcken zu speichern.

Before Images

Unter dem Begriff Before Image versteht Oracle eine Kopie eines Oracle blocks, bevor dieser geändert wird. Ein Before Image kann im weitesten Sinne als eine „Kopie der Originalwerte“ verstanden werden.

Before Images werden für verschiedene Zwecke benötigt. Dies sind im wesentlichen:

- Zurückrollen einer Transaktion, wenn ein **ROLLBACK**-Kommando ausgeführt werden soll
- Recovery der Datenbank
- Leseconsistenz
- Oracle Flashback Query
- Oracle Flashback Table

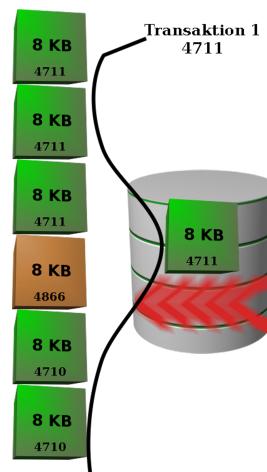
Wird ein **ROLLBACK**-Kommando am Ende einer Transaktion abgesetzt, müssen alle durch die Transaktion verursachten Änderungen rückgängig gemacht werden. Hierzu werden die Originalwerte aus den Before Images, die in den Undo-Segmenten liegen, benutzt.

Zu beachten ist, dass es mehrere Before Images eines Oracle blocks, mit unterschiedlichen Versionsständen geben kann. Da jeder Oracle block im Database Buffer Cache mit einer SCN versehen wird, kann anhand dieser das Alter des Blocks (und somit seine Version) bestimmt werden. Je höher die SCN, desto neuer ist der Block.

Multiversion Concurrency Control durch Before Images

Abbildung ?? zeigt die Nutzung der Before Images für das Erzeugen von Leseconsistenz.

Abb. 21.2:
Lesekonsistenz
durch
Multiversion
Concurrency
Control



Eine Transaktion wird bei SCN 4711 gestartet. Um Leseconsistenz zu gewährleisten, muss die Datenbank dafür sorgen, dass diese Transaktion nur solche Datenblöcke liest, deren SCN kleiner gleich 4711 lautet. Ein Oracle block trägt bereits die SCN 4866. Für diesen muss Oracle ein Before Image, mit einer SCN kleiner oder gleich 4711, aus den Undo-Segmenten holen.

22 Undo-Daten verwalten

Inhaltsangabe

22.1 Undo-Tabelespaces verwalten

22.1.1 Automatic Undo Management aktivieren

Oracle ist in der Lage, Undo-Daten voll automatisiert zu verwalten. Der dazu verwendete Mechanismus heißt „Automatic Undo Management“. Wird er benutzt, muss nur ein Undo-Tabelspace erstellt werden und Oracle kümmert sich um den Rest.

Zur Aktivierung dieses Features, muss der Parameter `undo_management` auf den Wert „auto“ gesetzt werden. Dadurch wird bei der Erstellung der Datenbank automatisch ein „Default Undo-Tabelspace“ angelegt. Ebenfalls werden auch alle Initialisierungsparameter, die mit manuellem Undo-Management zu tun haben ignoriert.



Manuelles Undo Management ist zwar noch immer möglich, wird aber von Oracle keinesfalls mehr empfohlen!

Ist der Undo-Tabelspace nicht verfügbar, kann die Datenbank nicht geöffnet werden. Tritt dieser Fall ein, wird ein Eintrag in der Alert log Datei gemacht.

Listing 22.1: Fehlermeldung bei nicht vorhandenem Undo-Tabelspace

```
Mon Sep 23 15:52:19 2007
Errors in file /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_544.trc:
ORA-30012: Undo Tablespace 'UNDOTBS01' ist nicht vorhanden oder hat den falschen
Typ
Mon Sep 23 15:52:19 2007
Error 30012 happened during db open, shutting down database
user: terminating instance due to error 30012
Mon Sep 23 15:52:19 2007
Error in file /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_pmon_3000.trc
ORA-30012: undo tablespace '' does not exist or of wrong type
```

Der Parameter `undo_tablespace` legt den Undo-Tabelspace der Instanz fest.

Listing 22.2: Der Parameter `undo_management`

```
UNDO_TABLESPACE = undotbs_01
```

22.1.2 Einen Undo-Tabelspace erstellen

Es gibt zwei Möglichkeiten einen Undo-Tabelspace zu erstellen:

- Durch das `CREATE DATABASE`-Kommando, beim Anlegen der Datenbank.
- Mit dem `CREATE UNDO TABLESPACE`-Statement

Wird ein Undo-Tablespace mit dem `CREATE UNDO TABLESPACE`-Statement erstellt, sind die beiden einzigen Parameter die dabei geändert werden können:

- die Datafile-Klausel
- die Extent-Management-Klausel

Im folgenden Beispiel wird ein automatisch wachsender Undo-Tablespace erstellt.

Listing 22.3: Undo-Tablespace erstellen

```
SQL> CREATE UNDO TABLESPACE undotbs02
  2  DATAFILE '/u02/oradata/orcl/undotbs02_01.dbf' SIZE 10M
  3  AUTOEXTEND ON;
```

 Es können mehrere Undo-Tabelspaces erstellt werden, jedoch ist immer nur einer aktiv.

22.1.3 Einen Undo-Tablespace verändern

Undo-Tabelspaces werden mit dem Kommando `ALTER TABLESPACE` verändert. Da die meisten Parameter für einen Undo-Tablespace direkt vom System verwaltet werden, muss sich der DBA nur um folgende Dinge kümmern:

- Datendateien zum Tablespace hinzufügen
- Datendateien des Tablespaces umbenennen
- Datendateien des Tablespaces on-/offline setzen
- (De-)Aktivieren der Retention Guarantee

Andere Parameter können nicht durch den Administrator verändert werden.

Um zu verhindern, dass der Platz in einem Undo-Tablespace nicht mehr ausreicht, kann eine Datendatei hinzugefügt werden.

Listing 22.4: Datendatei zum Undo-Tablespace hinzufügen

```
SQL> ALTER TABLESPACE undotbs02
  2 ADD DATAFILE '/u01/app/oracle/oradata/orcl/undotbs02_02.dbf' SIZE 100M
  3 AUTOEXTEND ON MAXSIZE 5G;
```

Eine andere Möglichkeit besteht darin, eine bestehende Datendatei mit dem `ALTER DATABASE`-Kommando in ihrer Größe zu verändern.

Listing 22.5: Datendatei in ihrer Größe verändern

```
SQL> ALTER DATABASE
  2 DATAFILE '/u01/app/oracle/oradata/orcl/undotbs02_02.dbf'
  3 RESIZE 250M;
```

22.1.4 Einen Undo-Tablespace löschen

Um einen Undo-Tablespace zu löschen, wird das `DROP TABLESPACE`-Kommando verwendet.

Listing 22.6: Undo-Tablespace löschen

```
SQL> DROP TABLESPACE undotbs02;
```



Ein Undo-Tablespace kann nur gelöscht werden, wenn keine offenen Transaktionen ihn mehr verwenden. Zu beachten ist dabei, dass ein Undo-Tablespace, trotz der Tatsache, dass als unexpired markierte Undo-Records existieren, gelöscht werden kann. Daher ist beim Löschen von Undo-Tablespaces größte Vorsicht geboten.

22.1.5 Den Undo-Tablespace wechseln

Da der `undo_tablespace`-Initialisierungsparameter dynamisch ist, ist es möglich den Undo-Tablespace im laufenden Betrieb zu wechseln.

Listing 22.7: Undo-Tablespace wechseln

```
SQL> ALTER SYSTEM SET undo_tablespace = undotbs02;
```

Auswirkungen des Wechsels

Hat der Wechsel funktioniert, wird der neue Undo-Tablespace, ohne für die Nutzer spürbare Änderungen, verwendet.

Ein Fehlschlagen des Wechsels kann aus folgenden Gründen geschehen:

- Der neue Tablespace existiert nicht
- Der neue Tablespace ist kein Undo-Tablespace
- Der neue Undo-Tablespace wird bereits von einer anderen Instanz benutzt (RAC)

Wenn nach dem Wechsel neue Transaktionen durch Nutzer gestartet werden, werden die Undo-Daten hierfür im neuen Undo-Tablespace verwaltet. Der alte Undo-Tablespace bekommt den Status „pending offline“. Transaktionen, die vor dem Wechsel bereits bestanden, werden noch aus dem alten Undo-Tablespace bedient, es können jedoch keine neuen mehr hinzukommen.

Während ein Undo-Tablespace im pending offline-Modus ist, kann er nicht gelöscht werden. Erst wenn alle diesen Undo-Tablespace betreffenden Transaktionen beendet wurden, wird der Status des Tablespaces von pending offline auf offline gesetzt. Jetzt kann dieser Undo-Tablespace entweder einer anderen Instanz zur Verfügung gestellt oder gelöscht werden.

22.2 Undo Retention

Durch das Ausführen von Transaktionen sammeln sich Undo-Daten an, die für das Zurückrollen der Transaktionen oder für Recoveryzwecke gebraucht werden. Auch nach dem Abschluss einer Transaktion ist es notwendig, diese Daten weiterhin vorzuhalten, da sie zur Aufrechterhaltung der Lesekonsistenz anderer Transaktionen wichtig sind.

Die Undo Retention bestimmt, wie lange Undo-Daten im Undo-Tablespace verweilen. Je nach dem, wie die Datenbank genutzt wird (viele Schreibvorgänge, lange Lesevorgänge, usw.) ist es wichtig diesen Wert höher oder niedriger einzustellen. Mit der Aktivierung der automatischen Undo-Verwaltung wird auch ein Standardwert von 900 Sekunden für die Undo Retention gesetzt. Abhängig von der Undo Retention und davon, wie lange ein Datenblock bereits im Undo-Tablespace verweilt, kann er zwei verschiedene Markierungen erhalten:

- **expired:** Alte Undo-Daten, deren Transaktionen bereits bestätigt wurden und deren Alter größer ist, als die aktuelle Undo Retention, werden als expired¹ markiert.
- **unexpired:** Undo-Daten, deren Transaktionen bereits bestätigt wurden, die aber noch jünger sind, als die aktuelle Undo Retention, werden als unexpired² markiert.

¹expired = engl. abgelaufen

²unexpired = engl. noch nicht abgelaufen

Die Undo Retention wird durch Oracle automatisch, basierend auf der Größe des Undo-Tablespace und der aktuellen Aktivitäten im System, richtig gesetzt. Eine manuelle Einstellung der Undo Retention kann über den Parameter `undo_retention` geschehen. Die Angabe erfolgt in Sekunden. Vorausgesetzt der Undo-Tablespace ist groß genug, hält die Datenbank diese Vorgabe ein.

Wird der Platz für neue Transaktionen zu knapp, werden als expired markierte Undo-Daten überschrieben. Sollte dann immer noch nicht genug Platz vorhanden sein, beginnt die Datenbank, die als unexpired markierten Daten zu überschreiben. Dies kann dazu führen, dass die überschriebenen Daten, noch von einer lang laufenden Abfrage benötigt würden, weshalb die Datenbank die Abfrage mit der Fehlermeldung „Snapshot too old“ abbricht.

Die beiden folgenden Punkte beschreiben noch einmal kurz, wie sich die Undo Retention auf die Lesekonsistenz der Datenbank auswirkt.

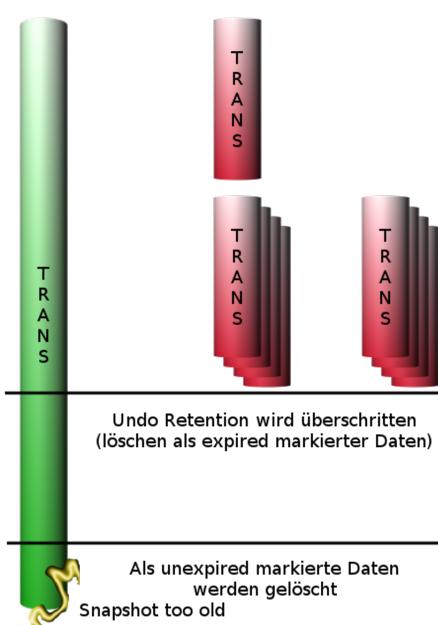


Abb. 22.1:
Die
Fehlermeldung
Snapshot too old

- Hat der Undo-Tablespace eine vordefinierte Größe und kann nicht wachsen (autoextend wurde nicht benutzt), wird der Parameter `undo_retention` ignoriert, sobald der Platz im Undo-Tablespace zu knapp wird. In dieser Situation werden als unexpired markierte Undo-Daten überschrieben.
- Kann der Undo-Tablespace wachsen, versucht die Datenbank sich an den Vorgabewert für die Undo Retention zu halten. Wird der Platz im Undo-Tablespace zu knapp wird der Tablespace vergrößert, bis eine evtl. definierte Maximalgröße erreicht wird. Danach werden unter Umständen wieder als unexpired markierte Undo-Records gelöscht.

22.2.1 Automatisches Tuning der Undo Retention

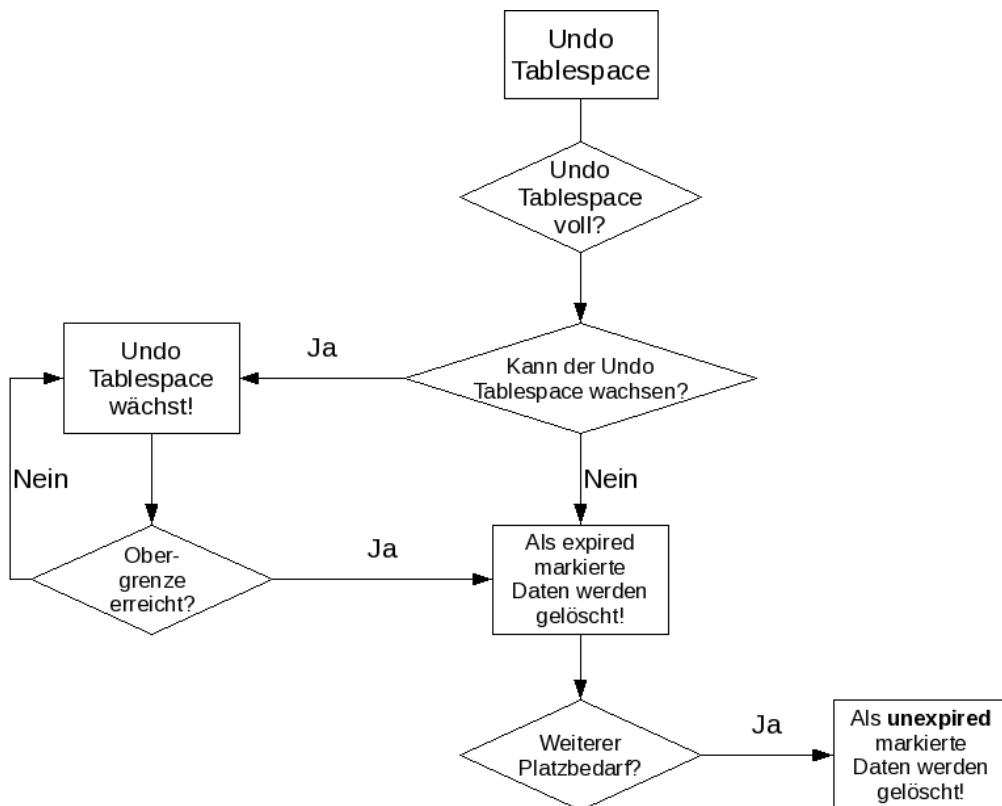
Wie bereits erwähnt, versucht Oracle den Wert für die Undo Retention automatisch zu tunen. Dies geschieht nach den folgenden Vorgaben:

- Hat der Undo-Tablespace eine feste Größe, tuned Oracle die Undo Retention so, das sie für die Größe des Undo-Tablespaces und die aktuelle Transaktionslast optimal ist.
- Wurde der Undo-Tablespace so konfiguriert, das er wachsen kann, versucht Oracle die Undo Retention so zu tunen, das sie groß genug für die bisher größte gelaufene Abfrage ist.

Die aktuelle *Tuned Undo Retention* kann in der dynamischen Performance View V\$UNDOSTAT in 10-Minuten-Intervallen über die letzten 4 Tage hinweg abgefragt werden. Der Wert wird dort in Sekunden angegeben. Soll die Tuned Undo Retention über einen längeren Zeitraum, als 4 Tage, hinweg beobachtet werden, kann hierzu die View DBA_HIST_UNDOSTAT verwendet werden.

Listing 22.8: Die Tuned Undo Retention abfragen

```
SQL> SELECT TO_CHAR(begin_time , 'DD.MM.YY HH24:MI') AS "Begin Time",
  2      TO_CHAR(end_time , 'DD.MM.YY HH24:MI') AS "End Time",
  3      tuned_undoretention
  4  FROM v$undostat
  5 ORDER BY end_time;
```

Abb. 22.2:
Undo Retention

22.2.2 Retention Guarantee

Um den Erfolg von besonders lang laufenden Abfragen garantieren zu können, bietet Oracle die „Retention Guarantee“. Sie wird beim Anlegen des Undo-Tablespaces durch die Klausel `RETENTION GUARANTEE` aktiviert. Ist dies der Fall, werden die als unexpired markierten Undo-Records solange gespeichert, wie sie benötigt werden, um die Leseconsistenz für alle laufenden Abfragen aufrecht zu erhalten.

Die Aufrechterhaltung der Lesekonsistenz geschieht jedoch nicht ohne Kosten. Durch die Retention Guarantee ist es möglich, dass für DML-Statements bzw. deren Transaktionen nicht mehr genügend Speicher im Undo-Tablespace vorhanden ist. Eine solche Transaktion muss dann abgebrochen und zurückgerollt werden. Aus diesem Grund wird von Oracle empfohlen, diesen Mechanismus mit Bedacht zu benutzen.

Um die Retention Guarantee zu deaktivieren gibt es das `ALTER TABLESPACE`-Statement zusammen mit der `RETENTION NOGUARANTEE`-Klausel.

Listing 22.9: (De-)aktivieren der Retention Guarantee

```
SQL> ALTER TABLESPACE undotbs02 RETENTION GUARANTEE;
SQL> ALTER TABLESPACE undotbs02 RETENTION NOGUARANTEE;
```

Ob für den Undo-Tablespace die Retention Guarantee aktiviert wurde, kann der View `DBA_TABLESPACES` entnommen werden.

Listing 22.10: Den Status der Retention Guarantee abfragen

```
SQL> SELECT tablespace_name , retention
  2  FROM   dba_tablespaces
  3 WHERE  contents = 'UNDO';

TABLESPACE_NAME          RETENTION
-----  -----
UNDOTBS1                NOGUARANTEE
```

22.3 Informationen

22.3.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10224]
- [REFRN10225]
- [REFRN10227]

22.3.2 Verzeichnis der relevanten Data Dictionary Views



- [sthref3583]
- [sthref3800]
- [sthref2575]
- [sthref3804]
- [REFRN23460]

22.4 Übungen - Undo-Daten verwalten

1. Erstellen Sie einen neuen Undo-Tablespace names UNDOTBS03. Die Datendatei soll /u02/oradata/orcl/undotbs03 heißen und 20 M groß sein. Das automatische Wachstum für diesen Tablespace muss deaktiviert werden!
2. Die folgenden Aufgaben müssen in Kombination bearbeitet werden.
 - a) Vergrößern Sie UNDOTBS1 auf 200 M.
 - b) Öffnen Sie sich ein zweites Terminalfenster und starten Sie in diesem Fenster SQL*Plus als sysdba.
 - c) Führen Sie das Skript lab_transaktionen.sh aus. Geben Sie dazu in der Shell folgendes ein:

```
[oracle@FEA11-119SRV ~]$ cd ~/labs  
[oracle@FEA11-119SRV ~]$ ./lab_transaktionen.sh 5
```

Dieses Skript befindet sich im Verzeichnis /home/oracle/labs und wird für ca. 5 Minuten mehrere Transaktionen starten. Nach Ablauf der 5 Minutenfrist werden diese Transaktionen committed.

- d) Warten Sie, bis für alle fünf Transaktionen die Meldung *Connected.* angezeigt wird. Dies kann ein paar Sekunden (ca. 20) dauern.
- e) Wechseln Sie den Undo-Tablespace, so dass UNDOTBS03 ihr aktueller Undo-Tablespace wird.
- f) Prüfen Sie den Status des Original-Undo-Tablespace UNDOTBS1. Wann findet der Wechsel von UNDOTBS1 zu UNDOTBS03 tatsächlich statt? Führen Sie hierzu die folgende Abfrage mehrfach in ca. 30 Sekunden Abständen aus:

```
SQL> SELECT usn, status  
2 FROM v$rollstat;
```

3. Wechseln Sie den Undo-Tablespace zurück zum Original Undo-Tablespace und löschen Sie UNDOTBS02 und UNDOTBS03.

22.5 Lösungen - Undo-Daten verwalten

- Erstellen Sie einen neuen Undo-Tablespace names UNDOTBS03. Die Datendatei soll /u02/oradata/orcl/undotbs03_01.dbf heißen und 20 M groß sein. Das automatische Wachstum für diesen Tablespace muss deaktiviert werden!

```
SQL> CREATE UNDO TABLESPACE undotbs03
  2  DATAFILE '/u02/oradata/orcl/undotbs03_01.dbf' SIZE 20M
  3  AUTOEXTEND OFF;
```

- Die folgenden Aufgaben müssen in Kombination bearbeitet werden.

- Vergrößern Sie UNDOTBS1 auf 200 M.

```
SQL> ALTER DATABASE
  2  DATAFILE '/u01/app/oracle/oradata/orcl/undotbs01.dbf' RESIZE 200M;
```

- Öffnen Sie sich ein zweites Terminalfenster und starten Sie in diesem Fenster SQL*Plus als sysdba.

```
[oracle@FEA11-119SRV ~]$ . oraenv
ORACLE_SID = [orcl] ? orcl
The Oracle base for ORACLE_HOME=/u01/app/oracle/product/11.2.0/ORCL
is /u01/app/oracle
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba
```

- Führen Sie das Skript lab_transaktionen.sh aus. Geben Sie dazu in der Shell folgendes ein:

```
[oracle@FEA11-119SRV ~]$ cd ~/labs
[oracle@FEA11-119SRV ~]$ ./lab_transaktionen.sh 5
```

Dieses Skript befindet sich im Verzeichnis /home/oracle/labs und wird für ca. 5 Minuten mehrere Transaktionen starten. Nach Ablauf der 5 Minutenfrist werden diese Transaktionen committed.

- Warten Sie, bis für alle fünf Transaktionen die Meldung *Connected.* angezeigt wird. Dies kann ein paar Sekunden (ca. 20) dauern.
- Wechseln Sie den Undo-Tablespace, so dass UNDOTBS03 ihr aktueller Undo-Tablespace wird.

```
SQL> ALTER SYSTEM
  2  SET undo_tablespace='UNDOTBS03' SCOPE=both;
```

- f) Prüfen Sie den Status des Original-Undo-Tablespace UNDOTBS1. Wann findet der Wechsel von UNDOTBS1 zu UNDOTBS03 tatsächlich statt? Führen Sie hierzu die folgende Abfrage mehrfach in ca. 30 Sekunden Abständen aus:

```
SQL> SELECT usn, status  
2 FROM v$rollstat;
```

3. Wechseln Sie den Undo-Tablespace zurück zum Original Undo-Tablespace und löschen Sie UNDOTBS02 und UNDOTBS03.

```
SQL> ALTER SYSTEM  
2 SET undo_tablespace='UNDOTBS1' SCOPE=both;  
SQL> DROP TABLESPACE undotbs02  
2 INCLUDING CONTENTS AND DATAFILES;  
SQL> DROP TABLESPACE undotbs03  
2 INCLUDING CONTENTS AND DATAFILES;
```


23 Konfigurieren der Oracle-Netzwerkumgebung

Inhaltsangabe

23.1 Der Listener

Der Listener ist ein eigenständiger, auf dem Datenbankserver laufender Prozess. Er empfängt eingehende Verbindungsanforderungen der Clients und leitet den Verbindungsaufbau zur Datenbank ein. Er speichert seine Konfiguration in der Datei `listener.ora`.

Da es für alle verbindungsspezifischen Parameter Standardwerte gibt, ist es möglich einen Listener ohne eine Konfiguration zu starten. Dieser Default-Listener trägt den Namen „LISTENER“ und hat keine Informationen über vorhandene Instanzen zur Verfügung. Die Datenbankinstanzen müssen sich dynamisch beim Listener registrieren (PMON, Service registration). Daraus ergeben sich folgende Vorteile:

- **Vereinfachte Konfiguration:** Es sind keine zusätzlichen Konfigurationseinstellungen für dieses Feature notwendig.
- **Connect-time failover:** Da bei dynamischer Registrierung einer Instanz dem Listener auch der Status der Instanz (started, shutdown) mitgegeben wird, kann er im Falle dessen, dass eine Instanz ausfällt, den Nutzerprozess an eine andere Instanz der gleichen Datenbank (RAC) verweisen. Dies geschieht transparent für den Nutzer. Bei der statischen Registrierung einer Instanz ist kein Connect-time failover möglich.
- **Runtime connection load balancing:** Service registration ermöglicht dem Listener die Verbindungsanforderungen der Clients immer an den am schwächsten ausgelasteten Service handler (Dispatcher, Dedicated Server Prozess) weiterzuleiten.

23.1.1 Kommunikation zwischen Client und Server

Für die Kommunikation zwischen Client und Datenbankserver verwendet Oracle eine Architektur, die auf dem ISO/OSI Modell basiert. In Abbildung ?? wird sowohl der client- als auch der serverseitige Aufbau der Kommunikationsarchitektur gezeigt.

Die Gliederung der Fähigkeiten in einzelne Schichten bzw. Ebenen hat den Vorteil, dass eine Änderung an einer Ebene sich nicht auf die anderen auswirkt.

Clientanwendung

Anwendungen die mit einer Oracle-Datenbank kommunizieren wollen, müssen das Oracle Call Interface (OCI) oder das Oracle C++ Call Interface (OCCI) benutzen. Diese Interfaces stellen der Clientanwendung alle Methoden zur Verfügung, um z. B. eine Session aufzubauen, SQL-Statements an einen Serverprozess zu senden und vieles mehr.

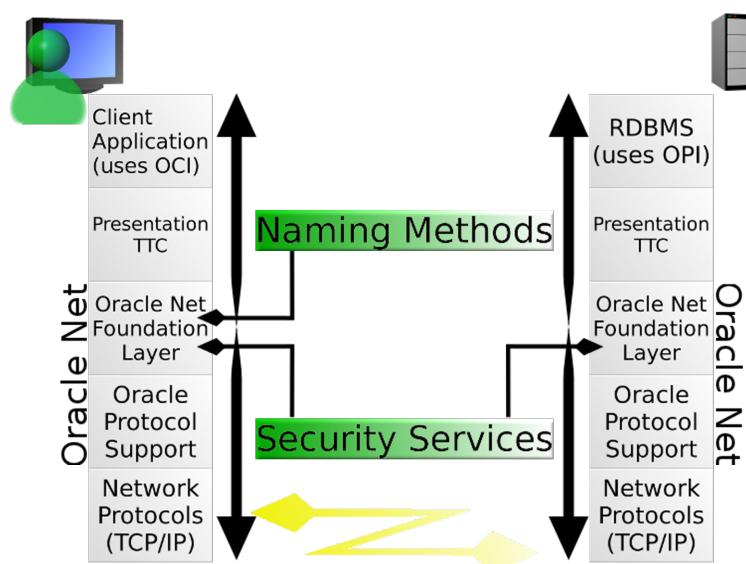
Presentation - TTC

Wenn der Datenbankserver und die Clientanwendung auf unterschiedlichen Betriebssystemen laufen, kann es zu Schwierigkeiten kommen, da der Client und der Server unterschiedliche Zeichensätze zur Darstellung der Informationen verwenden. Der Two-Task-Common-Presentation Layer, kurz TTC, beseitigt diese Probleme durch Konvertierung der Zeichensätze.



Der TTC konvertiert nur Zeichensätze. Er ist kein „Sprachübersetzer“.

Abb. 23.1:
Die Oracle Netzwerkarchitektur



Oracle Net Foundation Layer

Der *Oracle Net Foundation Layer (ONFL)* ist dafür Verantwortlich, die Kommunikation zwischen der Clientanwendung und dem Datenbankserver zu etablieren und aufrecht zu erhalten, sowie den Nachrichtenaustausch zwischen beiden Seiten zu ermöglichen. Diese Anforderungen kann der ONFL erfüllen, da er

eine Technik verwendet, die sich „Transparent Network Substrate (TNS)“ nennt. TNS stellt die Möglichkeit für eine Peer-To-Peer Kommunikation zwischen Client und Server bereit.

Eine weitere Fähigkeit des ONFL ist es, Benennungsmethoden für die Kommunikation zwischen Client und Server bereitzustellen.

Oracle Protocol Support

Auf der Position zwischen dem Oracle Network Foundation Layer und den Netzwerkprotokollen ist es seine Aufgabe als Gateway zwischen TNS, TCP/IP und den anderen Protokollen zu dienen. Er sorgt dafür, dass die TNS-Fähigkeiten des ONFL mit Hilfe der Netzwerkprotokolle über das Netzwerk übertragen werden können.

23.1.2 Service Registration

Ob eine Datenbank im Netzwerk verfügbar ist, bestimmt der Listener durch einen Mechanismus, der „Service registration“ genannt wird. Es gibt zwei unterschiedliche Arten von Service Registration:

- Static Service Registration
- Dynamic Service Registration

Dynamic Service Registration

Die Dynamic Service Registration wird durch den Oracle Hintergrundprozess PMON durchgeführt. Er übergibt folgende Informationen an den Listener:

- Servicenamen der verbundenen Datenbank
- Name der Instanz(en) die zu diesem Servicenamen gehören
- Service handler die für diese Instanz verfügbar sind (siehe ??)

Mit Hilfe dieser Informationen ist der Listener in der Lage, die Clientanwendung mit dem Datenbankserver zu verbinden.



Der Listener muss vor dem Hochfahren der Instanz gestartet werden, da der PMON sonst die dynamic service registration nicht ordnungsgemäß durchführen kann.

Der PMON versucht in Intervallen von ca. 60 Sekunden eine Verbindung zu einem Listener herzustellen, um die Registrierung nachzuholen. Dieser Mechanismus kann auch manuell, mit dem Kommando `ALTER SYSTEM REGISTER` durchgeführt werden.

Erhält ein Listener eine Verbindungsanforderung bevor die angeforderte Instanz registriert wurde, wird er den Client abweisen.

Abb. 23.2:
Ablauf des
Service
Registration
Mechanismus



Um zu erfahren, ob eine Datenbank dynamisch bei ihrem Listener registriert wurde, kann das Listener control utility (siehe ??) genutzt werden. Die Anzeige von „status READY“ sagt aus, dass die Datenbank dynamisch registriert wurde.

Listing 23.1: Wurde die Datenbank registriert?

```
Service "orcl" has 1 instance(s).
Instance "orcl", status READY, has 1 handler(s) for this service...
```

Static service registration

Bei der Static Service Registration müssen die Informationen, die der Listener benötigt, in die Konfigurationsdatei des Listeners eingetragen werden. Die Ausgabe des Listener control utilities sieht dann wie folgt aus:

Listing 23.2: Statische Registrierung

```
Service "orcl" has 1 instance(s).
Instance "orcl", status UNKNOWN, has 1 handler(s) for this service...
```

Die Anzeige von „status UNKNOWN“ zeigt an, dass die Datenbank statisch registriert wurde, da der Listener nichts über den Zustand der Verbindung zur Datenbank weiß.

23.1.3 Die Datei listener.ora

Die Datei `listener.ora` dient als Konfigurationsdatei für den Listener. Sie befindet sich im Verzeichnis `$ORACLE_HOME/network/admin`. Sie enthält folgende Angaben:

- Namen der Listener
- Hostname, Port und Protokoll für Verbindungen
- Namen der Datenbankservices
- Listenerkontroll-Parameter



Mit Hilfe der Umgebungsvariable TNS_ADMIN kann ein alternativer Speicherort für die Datei `listener.ora` angegeben werden.

Beispiel ?? zeigt den Inhalt einer Listenerkonfigurationsdatei.

In Zeile 1 steht der Name des Listeners: „LISTENER“. Die Zeilen zwei und drei definieren die „Protokolladresse“ des Listeners. Sie enthält Angaben über das zu verwendende Netzwerkprotokoll, den Namen des Hosts, auf dem sich der Listener befindet und den Port, auf dem er lauscht.

Listing 23.3: Die Datei `listener.ora`

```

LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = FEA11-119SRV.oracle.com)(PORT = 1521))
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = orcl.local)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/orcl)
      (SID_NAME = orcl)
    )
  )

ADR_BASE_LISTENER = /u01/app/oracle

```

In Zeile 6 beginnt mit dem Parameter SID_LIST_<listenername> die statische Registrierung eines Datenbankservices. Die Angaben GLOBAL_DBNAME, ORACLE_HOME und SID_NAME sind notwendig, um Clients mit einer Datenbank verbinden zu können.

In der letzten Zeile wird durch den Parameter ADR_BASE_<listenername> dem Listener mitgeteilt, wo sich das ADR (Automatic Diagnostic Repository) befindet.

23.1.4 Der Oracle Net Manager

Der Oracle Net Manager ist eine in Java geschriebene Software, welche seit den Tagen von Oracle 8i existiert. Er ermöglicht die einfache und schnelle Konfiguration eines Listeners. Gestartet wird er durch das Kommando `netmgr` auf der Shell.



Genau wie für das SQL*Plus-Tool muss auch für den Net Manager zuerst das oraenv-Skript ausgeführt werden.

Mit Hilfe des Net Managers können verschiedene Einstellungen an der Oracle Netzwerkkonfiguration vorgenommen werden.

- Listener konfigurieren
- Benennungsmethoden konfigurieren
- Net Service Names erstellen

Einen Listener erstellen

1. In der Baumansicht auf das Pluszeichen neben „Lokal“ klicken, um den Rest der Baumansicht aufzuklappen.



Abb. 23.3:
Startbildschirm
des Net Managers

2. Auf den Knotenpunkt „Listener“ klicken.



Abb. 23.4:
Das Register
Listener erweitern

3. Links in der Symbolleiste auf das grüne Pluszeichen klicken.



Abb. 23.5:
Neuen Listener
hinzufügen

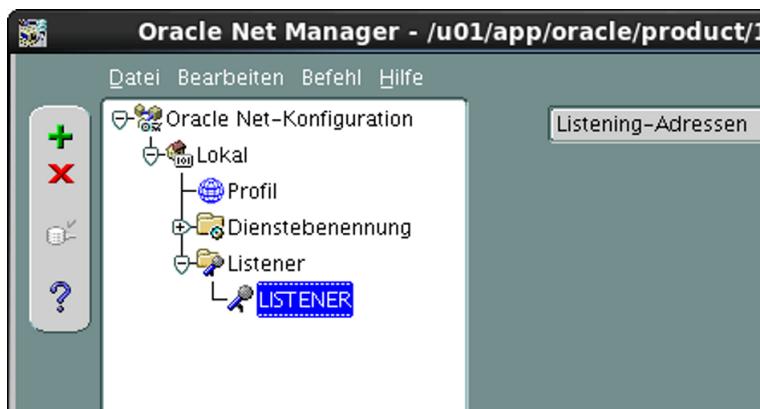
4. Geben Sie eine Bezeichnung für den Listener ein und klicken Sie auf „OK“.

Abb. 23.6:
Den neuen
Listener benennen



5. Der neue Listener ist fertig.

Abb. 23.7:
Der neue Listener
ist da



Der erste Listener den Sie erstellen sollte immer „LISTENER“ heißen, da dies der Standardname für einen Listener ist.

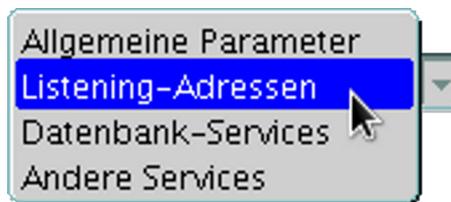
Eine weitere Protokolladresse konfigurieren

Die Möglichkeit eine weitere Protokolladresse zu konfigurieren erlaubt es dem Administrator, den einen Listener auf einem alternativen Port lauschen zu lassen oder aber einen weiteren Listening-Port hinzuzufügen. Laut Aussage von Oracle genügt jedoch in den meisten Umgebungen ein einziger Listener mit einem Port. Interessant ist diese Option daher nur dann, falls der Listener auf zwei unterschiedliche Netzwerkprotokolle konfiguriert werden soll.

Das Konfigurieren einer weiteren Protokolladresse geschieht im Net Manager im Bereich „Listening-Adressen“.

1. Klicken Sie auf die Bezeichnung des Listener, den Sie konfigurieren möchten.
2. Wählen Sie aus dem Dropdownfeld rechts oben, die Option „Listening-Adressen“ aus.

Abb. 23.8:
Die Option
Listening-
Adressen
auswählen



3. Klicken Sie rechts unten auf die Schaltfläche „Adresse hinzufügen“.



Abb. 23.9:
Eine neue
Listening-Adresse
hinzufügen

4. Geben Sie den Hostnamen und die Listener Portnummer im jeweiligen Textfeld ein.

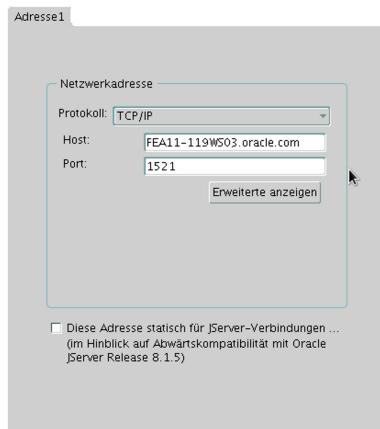


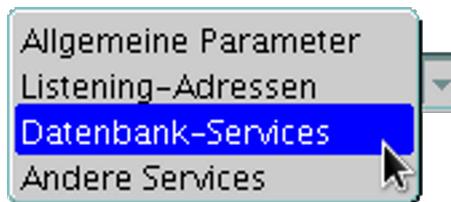
Abb. 23.10:
Fehlende
Angaben
ergänzen

Statische Registrierung konfigurieren

1. In der Baumansicht auf das Pluszeichen neben „Lokal“ klicken, um den Rest der Baumansicht aufzuklappen.
2. Auf den Knotenpunkt „Listener“ klicken.
3. Klicken Sie auf die Bezeichnung des Listener, den Sie konfigurieren möchten.

4. Wählen Sie aus dem Dropdownfeld rechts oben, die Option „Datenbank-Services“ aus.

Abb. 23.11:
Die Option
„Datenbank-
Services“
auswählen



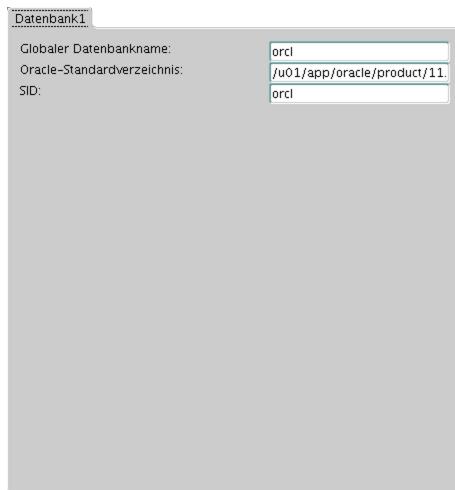
5. Klicken Sie rechts unten auf die Schaltfläche „Datenbank hinzufügen“.

Abb. 23.12:
Die Schaltfläche
„Datenbank
hinzufügen“



6. Füllen Sie die drei Textfelder aus, um eine Instanz statisch zu registrieren. Dies geht auch dann, wenn die Instanz noch gar nicht existiert.

Abb. 23.13:
Angaben für die
statische
Registrierung



Logging konfigurieren

1. In der Baumsicht auf das Pluszeichen neben *Lokal* klicken, um den Rest aufzuklappen.
2. Auf den Knotenpunkt *Listener* klicken.
3. Klicken Sie auf die Bezeichnung des Listener, den Sie konfigurieren möchten.

4. Wählen Sie aus dem Dropdownfeld rechts oben, die Option *Allgemeine Parameter* aus.

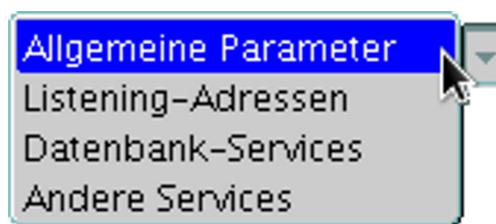


Abb. 23.14:
Die Option
,Allgemeine
Parameter“
auswählen

5. Wählen Sie die Registerkarte *Logging und Tracing* aus.

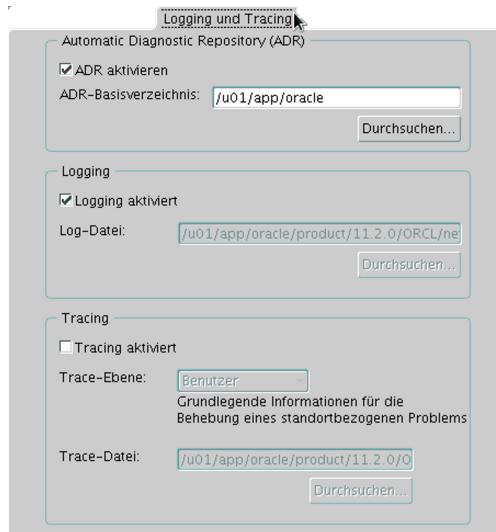


Abb. 23.15:
Die Registerkarte
,Logging und
Tracing“

6. Klicken Sie auf *Logging aktiviert* um das Logging zu aktivieren.

7. Passen Sie evtl. Pfad und Dateiname der Log-Datei an (zuerst muss der Haken bei *ADR aktivieren* entfernt werden).

23.1.5 Das Tool lsnrctl

Das Listener Control Utility, kurz `lsnrctl` steht dem Administrator als Konfigurationswerkzeug für den Listener zur Verfügung. Mit seiner Hilfe kann der Listener zur Laufzeit beeinflusst und seine aktuelle Konfiguration abgefragt werden. Es ist im Wesentlichen für das Starten und Stoppen des Listeners, sowie das Abfragen von Verbindungsdaten gedacht. Es befindet sich im Verzeichnis: `$ORACLE_HOME/bin`

Aufgerufen wird es mit dem Kommando `lsnrctl`. Die vier wichtigsten Befehle dieses Tools sind:

- **start** [listener]: Startet den angegebenen Listener. Wird kein Listener angegeben, wird der Standard-listener LISTENER gestartet.
- **stop** [listener]: Stoppt den angegebenen Listener. Wird kein Listener angegeben, wird der Standard-listener LISTENER gestoppt.
- **status** [listener]: Zeigt eine Statusmeldung über den angegebenen Listener. Wird kein Listener angegeben, wird eine Statusmeldung zum Standardlistener LISTENER gezeigt.
- **service** [listener]: Zeigt an, welche Services ein Listener unterstützt und um welche Services es sich handelt. Dieses Kommando unterscheidet sich vom Kommando `status` darin, dass es mehr Informationen zu den einzelnen Services ausgibt.



- [i486171]

23.2 Benennungsmethoden

Um eine Verbindung zu einer Datenbank aufbauen zu können, müssen Nutzer einen sogenannten „connect string“ angeben. Dieser setzt sich zusammen aus:

- einem Nutzernamen,
- einem Passwort und
- einem „Connect Identifier“

Der Connect Identifier ist eine Zeichenkette, die alle Angaben zu der Oracle-Instanz enthält, mit der die Verbindung hergestellt werden soll. Er wird mittels einer Benennungsmethode in einen Connect Descriptor aufgelöst. Der Listener erhält dann die Verbindungsanforderung des Clients und übernimmt den Verbindungsaufbau. Oracle kennt vier verschiedene Benennungsmethoden:

- Hostnaming

Hostnaming ist die einfachste Benennungsmethode. Wenn auf einem Server nur eine einzige Datenbank läuft, genügt die IP-Adresse/der DNS-Name des Servers, um die Datenbank zu kontaktieren. Es wird dafür keine weitere Konfiguration auf dem Clientrechner benötigt.

- Easy Naming

Easy Naming ermöglicht es, einen TCP-Connect String, der aus einer IP-Adresse, einem Port und einem Servicename besteht, zum Verbindungsaufbau zu verwenden. Diese Methode arbeitet völlig konfigurationslos.

- Local Naming

Hier wird der Net Service Name in einer lokalen Datei mit dem Namen: `tnsnames.ora` gespeichert.

- Directory Naming

Beim Directory Naming werden die Connect Identifier in einem zentralen LDAP-Service, dem „Oracle Internet Directory (OID)“ gespeichert.

- External Naming

Bei dieser Methode, werden Net Service Names in einem externen, von Oracle unterstützten, Verzeichnis (NIS, CDS) gespeichert.

- [CIHGGHEE]
- [i1047762]



23.2.1 Die Datei `sqlnet.ora`

Die Datei `sqlnet.ora` ist für die Konfiguration der Oracle Netzwerkeinstellungen da. Hier kann unter anderem auch festgelegt werden, welche Benennungsmethoden aktiviert sind. Beispiel ?? zeigt beispielsweise den Parameter `names.directory_path`.

Listing 23.4: Der Parameter `NAMES.Directory_path` in der Datei `sqlnet.ora`

```
NAMES.DIRECTORY_PATH=(HOSTNAME, TNSNAMES, EZCONNECT)
```

Zwischen den beiden Klammern werden die Namen der Benennungsmethoden in der Reihenfolge angegeben, in der sie genutzt werden sollen. Damit eine Benennungsmethode funktioniert, muss sie an dieser Stelle aufgelistet sein. Folgende Werte sind für `names.directory_path` zulässig:

- HOSTNAME : Hostnaming
- TNSNAMES : Local Naming
- EZCONNECT : Easy Connect Naming
- LDAP : Directory Naming
- NIS : Network Information Server (auch als Yellow Page bekannt)

23.2.2 Die Benennungsmethode Easy Naming

In einer TCP/IP-Umgebung ist das Easy Naming die einfachste Methode eine Verbindung zu einer Datenbank aufzubauen. Es besteht dabei keine Notwendigkeit, Net Service Names in der Datei `tnsnames.ora` zu speichern. Es wird einfach die TCP/IP-Methode des Hostnamings erweitert. Die Syntax für die Nutzung des Easy Namings ist folgende:

Listing 23.5: Easy Naming

```
connect username/password@[:] host [ :port ] [/services_name]
```

Einige Beispiele für das Easy Naming:

Listing 23.6: Beispiele für Easy Naming

```
connect hr/hr@FEA11-119SRV.oracle.com:1521/orcl.it-training-alt.fus
connect sys/oracle@//FEA11-119SRV.oracle.com/orcl.it-training-alt.fus as sysdba
```



Diese Verbindungsmethode kann nur dann verwendet werden, wenn keine Advanced Features (Verschlüsselung u. ä.) für die Verbindung zur Datenbank gefordert sind.



- [i507143]

23.2.3 Die Benennungsmethode Hostnaming

Die einfachste Benennungsmethode ist das Hostnaming. Hier ist keine Konfigurationsarbeit auf dem Clientrechner notwendig. Alle notwendigen Einstellungen werden nur einmal auf dem Server vorgenommen.

Funktionsweise

Damit das Hostnaming funktioniert, müssen folgende Bedingungen erfüllt sein:

- Das Hostnaming muss in der Datei `sqlnet.ora` aktiviert sein.
- Auf dem Datenbankserver muss ein Listener gestartet sein, der TCP/IP benutzt.

- Der Listeningport muss 1521 (Standardport) sein.
- Der Listener muss einen Service kennen, dessen Name gleichlautend mit dem Hostnamen des DB-Servers ist. Das bedeutet, wenn der Hostname des Datenbankservers FEA11-119SRV.oracle.com lautet, dann muss folglich auch der Service Name FEA11-119SRV.oracle.com sein.

Die ersten beiden Bedingungen sind standardmäßig immer erfüllt, die Dritte muss überprüft werden. Die vierte Bedingung kann auf zwei unterschiedliche Arten erfüllt werden.

- Eine Instanz, deren Initialisierungsparameter db_name gleichlautend mit dem Hostnamen ist, hat sich dynamisch beim Listener registriert.
- Es wurde eine statische Registrierung beim Listener vorgenommen und der Listenerparameter GLOBAL_DBNAME wurde passend gesetzt:

Listing 23.7: Eine statische Registrierung und der Parameter GLOBAL_DBNAME

```
LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = FEA11-119SRV.oracle.com)(PORT = 1521))
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = FEA11-119SRV.oracle.com)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0)
      (SID_NAME = orcl)))
```

Fehler im System - Bug.6374523

In der Oracle Version 11.2.0.1 liegt ein Bug vor, der die Benutzung des Hostnamings erschwert. Der Fehler besteht darin, dass der Client, bei seiner Verbindungsanfrage, den Service Name der gewünschten Datenbank nicht mitsendet. Dies kann durch die Anwendung des Tools tnsping sichtbar gemacht werden.

Listing 23.8: Der Parameter DEFAULT_SERVICE_listener_name

```
TNS Ping Utility for Linux: Version 11.2.0.1.0

Copyright (c) 1997, 2011, Oracle. All rights reserved.

Used parameter files:
/u01/app/oracle/product/11.2.0/orcl/network/admin/sqlnet.ora

Used HOSTNAME adapter to resolve the alias
```

```
Attempting to contact (DESCRIPTION=(CONNECT_DATA=(SERVICE_NAME=))
(ADDRESS=(PROTOCOL=TCP)(HOST=192.168.20.100)(PORT=1521)))
OK (20 msec)
```

Die Lösung hierfür ist im Oracle MySupport Forum beschrieben. Es muss der Listener-Parameter `DEFAULT_SERVICE_listener_name` in die Datei `listener.ora` eingefügt werden. Heißt der Listener „LISTENER“, so muss folgende Eintragung gemacht werden.

Listing 23.9: Der Parameter DEFAULT_SERVICE_listener_name

```
LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = FEA11-119SRV.oracle.com)(PORT = 1521))
  )

SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = FEA11-119SRV.oracle.com)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0)
      (SID_NAME = orcl)))
  )

DEFAULT_SERVICE_LISTENER=FEA11-119SRV.oracle.com
```

Durch die Angabe dieses Parameters, zusammen mit dem Hostnamen, wird jeder Client, dessen Anfrage keinen Service Name enthält, an den statisch registrierten Service „FEA11-119SRV.oracle.com“ verwiesen.

Benutzung des Hostnamings

Um diese Benennungsmethode zu nutzen, muss der Nutzer an seinem Clientrechner einen Nutzernamen, ein Passwort und den Hostnamen des Datenbankservers angeben. Dies könnte z. B. so aussehen:

Listing 23.10: Ein Beispiel für Hostnaming

```
connect hr/hr@FEA11-119SRV.oracle.com
```

Der Hostname wird durch ein @ Zeichen vom Passwort getrennt. Die Namensauflösung erfolgt, wie gewohnt durch einen DNS-Server oder evtl. auch durch eine Hosts-Datei.

23.2.4 Die Benennungsmethode Local Naming

Beim Local Naming werden Net Service Names in einer lokalen Datei mit dem Namen `tnsnames.ora` gespeichert. Jeder Net Service Name gehört zu einem Connect Descriptor. Wie Net Service Names und Connect Descriptors in der Datei `tnsnames.ora` gespeichert werden, ist im folgenden Beispiel zu sehen.

Connect Descriptoren

Ein Connect Descriptor wird zusammengesetzt aus:

- einer oder mehreren Protokolladressen eines Listeners
- zusätzlichen Verbindungsdaten für die Zieldatenbank

Listing 23.11: Ein Connect Descriptor

```
(DESCRIPTION=
  (ADDRESS_LIST=
    (ADDRESS= (PROTOCOL=tcp)(HOST=FEA11-119SRV.oracle.com)(PORT=1521))
  )
  (CONNECT_DATA=
    (SERVICE_NAME=orcl)))
```

Der ADDRESS-Abschnitt enthält die Protokolladresse des Listeners. Im zweiten Abschnitt, der die Bezeichnung „CONNECT_DATA“ trägt, sind die zusätzlichen Verbindungsdaten für die Zieldatenbank enthalten. In diesem Fall wird die Zieldatenbank durch ihren Service Name „orcl“ identifiziert.

Um die Nutzung solcher Connect Descriptoren zu vereinfachen, können Sie mit einem „Net Service Name“ versehen werden. Im folgenden Beispiel, wird der Net Service Name „sales“ für einen Connect Descriptor vergeben. Dieser Net Service Name kann dann anstatt des kompletten Connect Descriptors verwendet werden.

Listing 23.12: Aufbau eines Net Service Names

```
sales=
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS= (PROTOCOL=tcp)(HOST=FEA11-119SRV.oracle.com)(PORT=1521))
    )
    (CONNECT_DATA=
      (SERVICE_NAME=orcl)))
```

Um sich bei einer Datenbank anzumelden, kann, wie in den folgenden Beispielen, entweder der komplette Connect Descriptor oder aber der Net Service Name genutzt werden.

Listing 23.13: Verwendung eines Connect Descriptors als Connect Identifier

```
connect scott/tiger@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)
(HOST=FEA11-119SRV.oracle.com)(PORT=1521)) (CONNECT_DATA=(SERVICE_NAME=orcl))) \
```

Listing 23.14: Verwendung eines Net Service Name als Connect Identifier

```
connect scott/tiger@orcl
```

Konfiguration des Local Naming im Net Manager

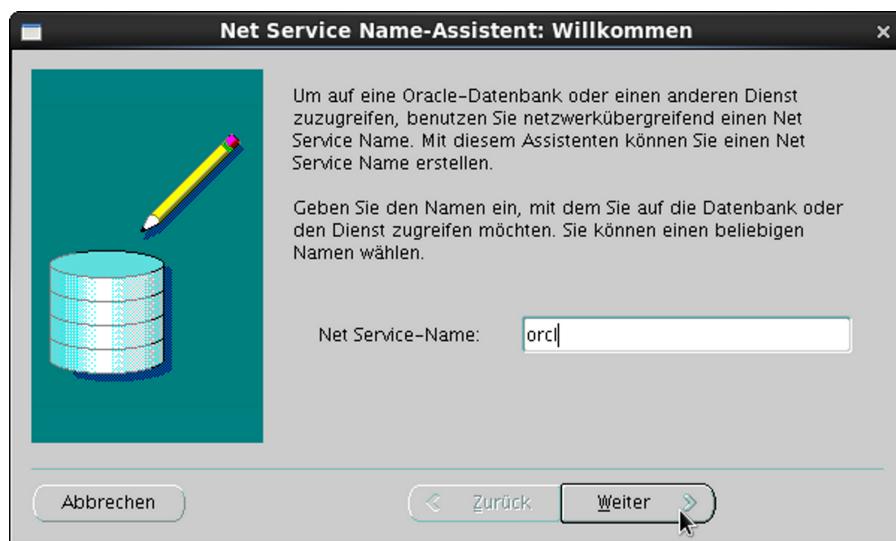
1. In der Baumansicht auf das Pluszeichen neben *Lokal* klicken, um den Rest der Baumansicht aufzuklappen.
2. Auf den Knotenpunkt „Dienstebenennung“ klicken.

Abb. 23.16:
Der Knotenpunkt
„Dienstebenennung“



3. Links in der Symbolleiste auf das grüne Pluszeichen klicken.
4. Geben Sie den Net Service Namen ein und klicken Sie auf „Weiter“.

Abb. 23.17:
Eingabe des Net
Service Name



5. Wählen Sie die gewünschten Netzwerkprotokolle aus, mit deren Hilfe eine Verbindung zur Datenbank erstellt werden soll und klicken Sie dann auf „Weiter“

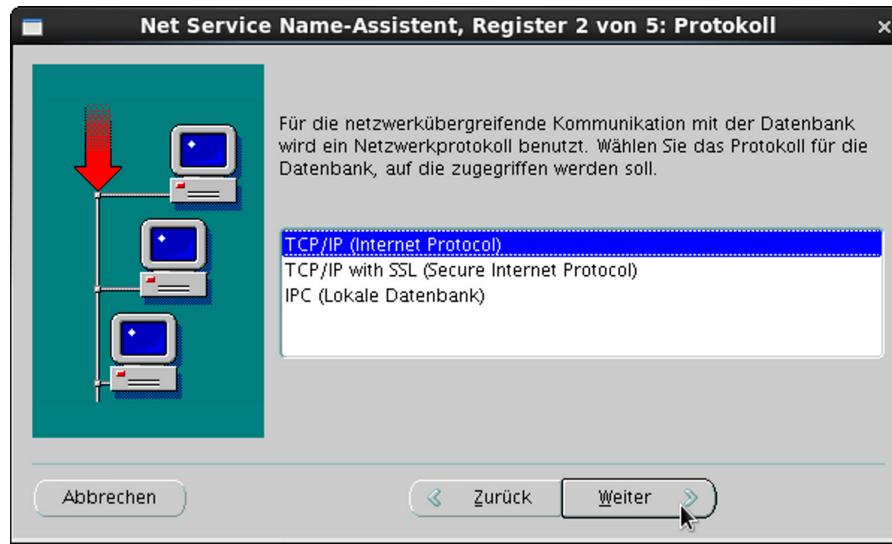


Abb. 23.18:
Auswahl des
Netzwerkproto-
kolls

6. Geben Sie die IP-Adresse oder den Hostnamen des Datenbankservers ein. Der Listenerport ist standardmäßig immer 1521. Falls Sie auf dem Server einen anderen Listenerport konfiguriert haben, tragen Sie diesen jetzt ein. Klicken Sie anschließend auf „Weiter“.

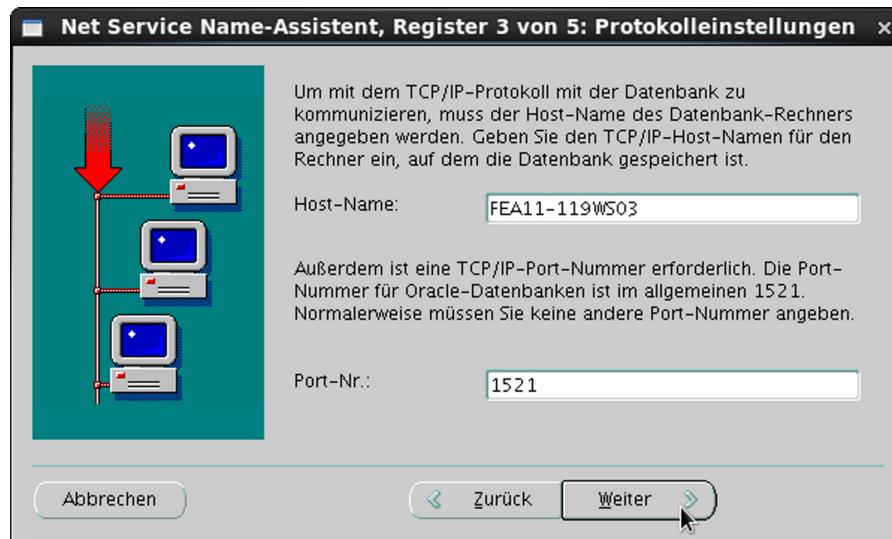


Abb. 23.19:
IP-Adresse und
Port des Listeners
festlegen

7. Geben Sie zur Identifikation der Datenbankinstanz den Service Name der Instanz an und klicken Sie auf „Weiter“.

Abb. 23.20:
Der Service
Name der
Datenbank



8. Um die Verbindung zu testen, klicken Sie auf „Testen...“. Klicken Sie anschließend auf „Beenden“.

Abb. 23.21:
Der
abschließende
Verbindungstest

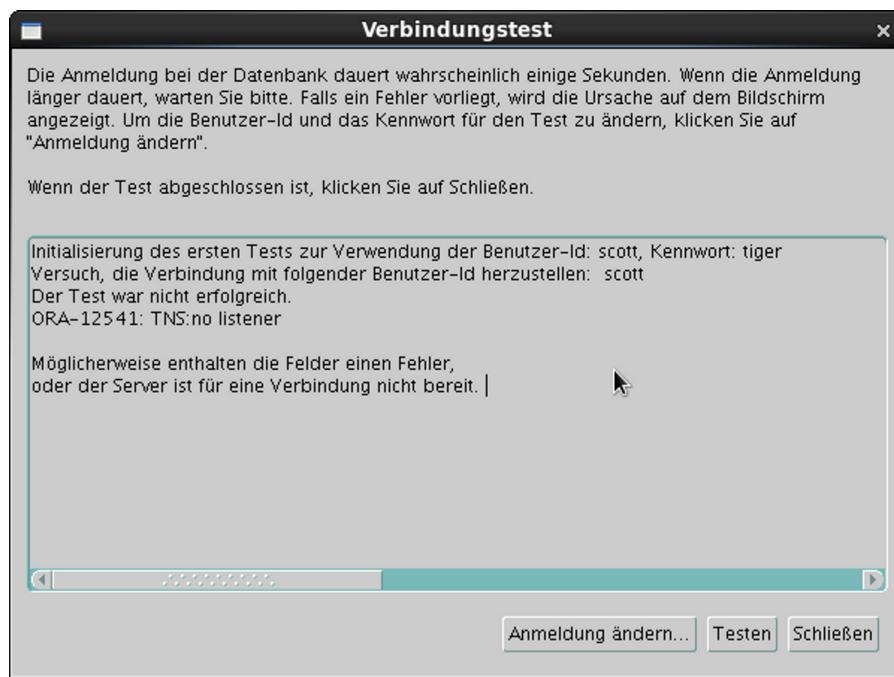




Abb. 23.22:
Den Assistenten
beenden

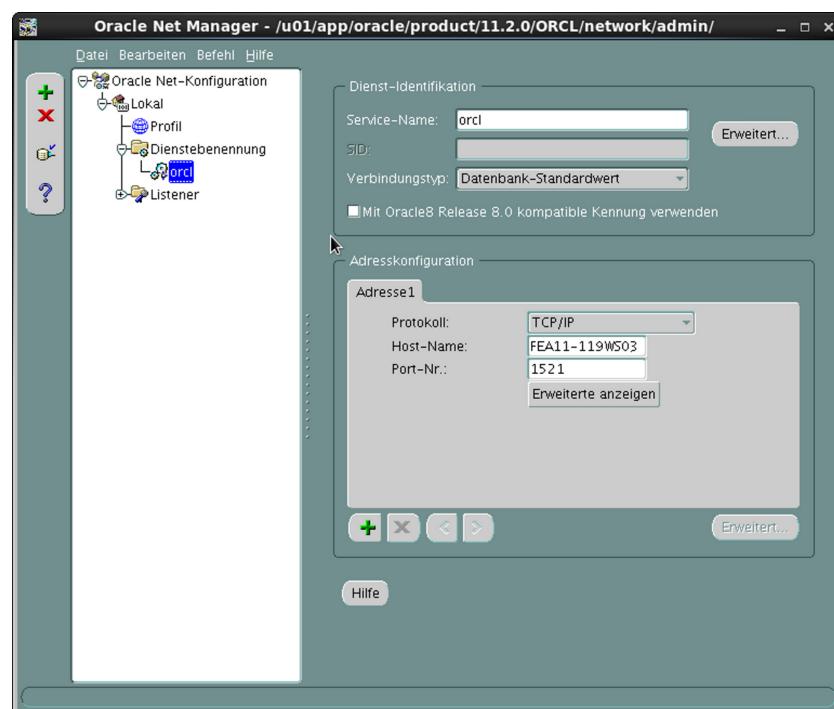


Abb. 23.23:
Der fertig
konfigurierte Net
Service Name

23.2.5 Die Benennungsmethode Directory Naming

Beim Directory Naming wird die gleiche Technik wie beim Local Naming verwendet: TNS. Der Unterschied zum Local Naming besteht jedoch darin, dass die TNS-Einträge nicht mehr in einer Textdatei gespeichert werden, die lokal auf jedem Clientrechner verfügbar sein muss, sondern in einem LDAP-Verzeichnisdienst, dem Oracle Internet Directory.

Um das Directory Naming nutzen zu können, müssen die folgenden Bedingungen erfüllt werden:

- Das Directory Naming muss in der Datei `sqlnet.ora` aktiviert sein.
- Die Datei `ldap.ora` muss auf den Clientrechnern konfiguriert werden.

Eine neue Konfigurationsdatei die hier ins Spiel kommt, ist die Datei `ldap.ora`. Sie steuert den Zugriff auf einen LDAP-Dienst und sieht folgendermaßen aus:

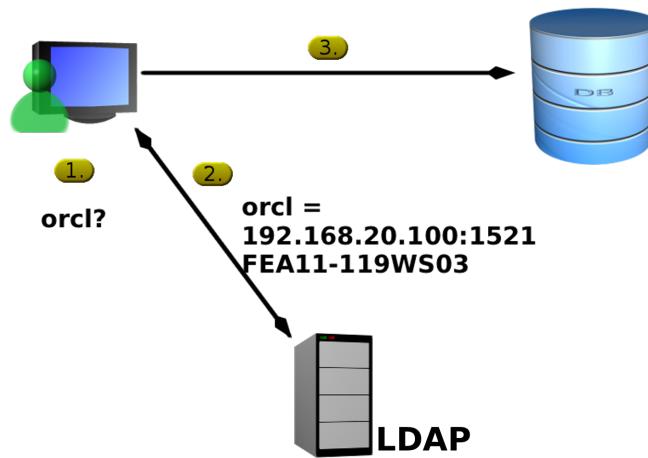
Listing 23.15: Die Datei `ldap.ora`

```
DEFAULT_ADMIN_CONTEXT = "ou=OracleContext"
DIRECTORY_SERVERS = (FEA11-1190ID20.oracle.com:389:636)
DIRECTORY_SERVER_TYPE = OID
```

Die drei gezeigten Parameter haben folgende Bedeutung:

- `default_admin_context`: Gibt an, wo in der Verzeichnisstruktur des LDAP die TNS-Daten gespeichert werden.
- `directory_servers`: Rechnername/IP-Adresse und Port/SSL-Port des Directoryservers.
- `directory_server_type`: Der Typ des Directoryserver (OID = Oracle Internetdirectory oder AD = Active Directory)

Die Auflösung eines in einem Directory gespeicherten Net Service Name läuft wie folgt ab:

Abb. 23.24:
Directory Naming

- Auf dem Clientrechner wird mittels der Datei `sqlnet.ora` ermittelt, dass eine Namensauflösung durch einen LDAP-Dienst gemacht werden soll. Die Datei `ldap.ora` gibt vor, welcher LDAP-Dienst zuständig ist.
- Der LDAP-Dienst beantwortet die Anfrage nach dem Net Service Name „orcl“ mit den Verbindungsdaten für den Listener des Datenbankservers (Protokoll, IP-Adresse, Port, Service Name).

- Der Client kann im dritten Schritt eine Verbindung zum Datenbankserver aufbauen. Wie der Verbindungsauflauf genau abläuft, ist davon abhängig, ob eine Dedicated Server Umgebung oder eine Shared Server Umgebung aufgebaut wurde.

23.3 Netzwerk Verbindungsmodelle

Unter dem Begriff „Verbindungsmodell“ versteht man die Art und Weise, wie ein Clientprozess mit einer Oracle-Instanz verbunden wird. Seit Oracle 11g gibt es drei verschiedene Verbindungsmodelle:

- Dedicated Server Architektur
- Shared Server Architektur
- Database Resident Connection Pooling (Oracle 11g New Feature)



Bei Dedicated Serverprozessen bzw. Dispatchern handelt es sich um sogenannte Service Handler. Ein Service Handler ist ein serverseitiger Prozess, der Anfragen eines Clients entgegen nimmt und bearbeitet (Serverprozess) oder nur zur Verarbeitung weiterleitet (Dispatcher).

23.3.1 Die Dedicated Server Architektur

In einer Dedicated Server Architektur startet der Listener für jeden Client einen eigenen Serverprozess. Nach dem die Session des Clients beendet wurde, wird der Serverprozess ebenfalls beendet. Diese Konfiguration benötigt sehr viele Ressourcen, da jeder Client seinen eigenen Serverprozess bekommt. Grundsätzlich ist dies aber die zu bevorzugende Variante, da sie die beste Performance erzielt.

1. Der Client schickt eine Verbindungsanforderung an den Datenbankserver. Der Listener nimmt diese entgegen.
2. Der Listener erstellt einen Serverprozess.
3. Der Serverprozess führt die Authentifizierung des Clients durch.
4. Es erfolgt ein *Connection-Redirect*. D. h. der Listener vermittelt die Verbindung zwischen dem Clientprozess und dem Serverprozess.

Im Anschluss an diese vier Schritte, ist der Client mit dem Datenbankserver verbunden und kann seine Arbeit aufnehmen.

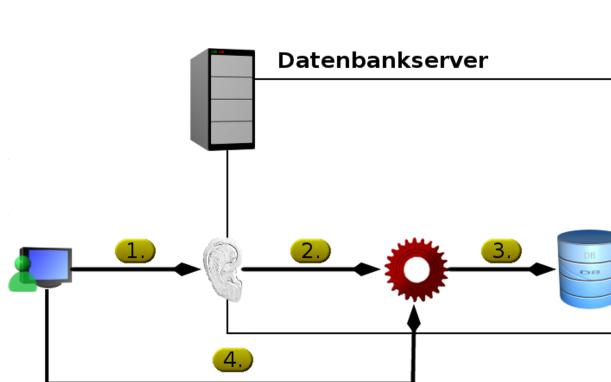
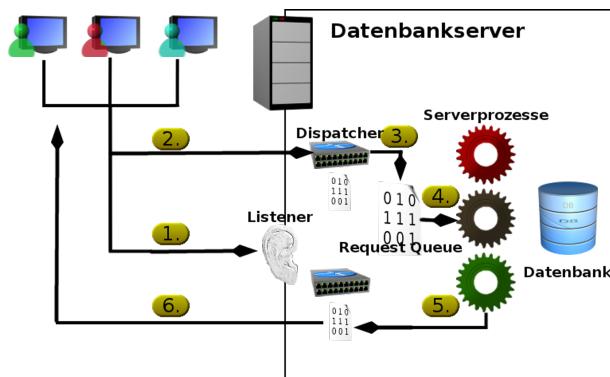


Abb. 23.25:
Aufbau der
Verbindung in
einer Dedicated
Server
Architektur

23.3.2 Die Shared Server Architektur

In einer Shared Server Architektur werden Clients nicht mit einem Serverprozess, sondern mit einem Dispatcher verbunden. Ein Dispatcher funktioniert als Verwalter und Verteiler von Arbeitsaufträgen. Er nimmt Anforderungen von Clients entgegen und leitet sie an einen Serverprozess weiter, der dann die eigentliche Arbeit verrichtet.

Abb. 23.26:
Aufbau der
Verbindung in
einer Shared
Server
Architektur



Jeder Dispatcher kann mehrere Clientverbindungen annehmen. Jede Clientanforderung wird von den Dispatchern in eine Warteschlange (Request Queue) aufgenommen. Ein Shared Serverprozess, der eine Anforderung abgearbeitet hat, nimmt sich aus dieser Warteschlange die nächste Anforderung, um sie abzuarbeiten und legt das Ergebnis der Anforderung in die Response Queue des Dispatchers, von dem die Anforderung kam. Die Dispatchers, lesen ihre Response Queue aus, um die Ergebnisse an ihre Clients auszuliefern. Auf diese Weise kann eine kleine Menge Shared Server Prozesse eine große Anzahl Clients bedienen.

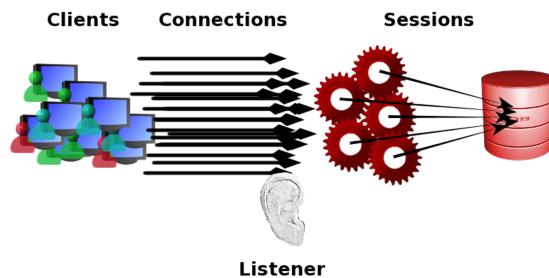
Der in Abbildung ?? gezeigte Verbindungsaufbau zwischen einem Client und der Datenbank läuft wie folgt ab:

1. Ein Client baut eine Verbindung zum Listener auf und fordert eine Verbindung zur Datenbank an.
2. Der Listener leitet den Client an einen der bestehenden Dispatcher-Prozesse weiter.
3. Der Dispatcher platziert die Anforderung in der Request Queue.
4. Ein Serverprozess nimmt die Anforderung zur Verarbeitung aus der Request Queue.
5. Der Serverprozess legt die Anforderung in die Response Queue des Dispatchers, von dem die Anforderung kam.
6. Der Dispatcher leitet das Ergebnis aus seiner Response Queue an den Client weiter, der die Anforderung gestellt hat.

23.3.3 Database Resident Connection Pooling (DRCP)

DRCP ist eine neue Technologie, die den Versuch darstellt, die Vorteile des Dedicated Server- und des Shared Server Modells zu vereinen. Beim DRCP erstellt die Datenbank einen Pool von Dedicated Serverprozessen, die als „Pooled Server“ bezeichnet werden. Dieser Pool kann von einer sehr großen Anzahl Clients genutzt werden, sofern die Anwendungen die gleichen Credentials (Username, Passwort) für den Aufbau der Session nutzen. Im Vergleich zum Shared Server Modell werden hier nicht nur die Serverprozesse geteilt, sondern auch die Sessions. Besonders vorteilhaft ist dies bei Webanwendungen, wo in kurzer Zeit sehr viele und zeitlich sehr begrenzte Connections aufgebaut werden.

Abb. 23.27:
Database
Resident
Connection
Pooling



Der Connection Broker

Der Connection Broker ist ein in Oracle 11g neu hinzugekommener Hintergrundprozess, der die Verwaltung der Pooled Server und der Sessions übernimmt. Muss ein Client an der Datenbank arbeiten, hat der Connection Broker die Aufgabe, ihm einen Serverprozess zuzuweisen. Der Serverprozess verhält sich wie

ein Dedicated Serverprozess für den Client. Hat der Client seine Arbeit beendet, übernimmt der Connection Broker wieder die Verwaltung von Serverprozess und Session.

Die Ressourcenersparnis

Die durch die Nutzung von DRCP entstehende Ressourcenersparnis, kann an einem einfachen Rechenbeispiel dargestellt werden:

In einem Firmenintranet soll eine Datenbankinfrastruktur für bis zu 3.000 Clients geschaffen werden. Bei der Nutzung des Dedicated Server Modells werden für jeden Serverprozess ca. 4 MB RAM benötigt, plus 400 KB für die Session. Dies ergibt in Summe: $3000 * (4MB + 400KB) \approx 12,86GB$

Wird die gleiche Infrastruktur als Shared Serverumgebung realisiert, werden weniger Serverprozesse benötigt, da sich mehrere Clients einen Serverprozess teilen. Werden für 3.000 Clients insgesamt 75 Serverprozesse zur Verfügung gestellt, ergeben sich folgende Zahlen: $(3000 * 400KB) + (75 * 4MB) \approx 1,44GB$ Arbeitsspeicher.

Das Database Resident Connection Pooling reduziert den Ressourcenverbrauch noch einmal, da mehrere Clients sich einen Serverprozess und eine Session teilen. Hinzu kommt jedoch ein Verwaltungsoverhead, von ca. 35 KB pro Client. Das Ergebnis sieht so aus: $(75 * (400KB + 4MB)) + (3000 * 35KB) \approx 0,42GB \hat{=} 431MB$ Arbeitsspeicher.

DRCP kann genauso wie die Shared Server Architektur nicht für Administratoren genutzt werden!



23.3.4 Dedicated Server, Shared Server und DRCP im Vergleich

Tabelle 23.1: Einsatz von Dedicated und Shared Server Modell

Dedicated Server	Shared Server	DRCP
Alle Clients haben einen eigenen Serverprozess und eine eigene Session.	Clients geben ihre Anforderungen über Dispatcher an einen Pool von Serverprozessen weiter. Sie haben nur noch eine eigene Session, keinen eigenen Serverprozess mehr.	Clients werden vom Connection Broker Prozess an einen Serverprozess mit einer bestehenden Session vermittelt.
Soll die Client-Connection beendet werden, müssen der Serverprozess und die Session zerstört werden.	Es muss nur die Session des Clients beendet werden, der Serverprozess bleibt erhalten.	Beim Verbindungsabbau übernimmt der Connection Broker wieder die Verwaltung des Serverprozesses und der Session.

23.4 Informationen

23.4.1 Verzeichnis der Konfigurationsdateien



- [NETRF011]
- [NETRF008]
- [NETRF007]
- [NETRF006]

23.5 Übungen - Konfigurieren der Oracle-Netzwerkumgebung

1. Konfigurieren Sie Ihre Namensauflösung so, dass Sie auf die Datenbank Ihres Banknachbarn zugreifen können. Verwenden Sie die folgenden Einstellungen zur Konfiguration der Namensauflösung:

Objekt	Einstellung
Net Service Name	orclneighbour
Protokol	TCP/IP
Host	IP/Hostname Ihres Nachbarn
Port	1521
Service-Name	SID Ihres Nachbarn
Verbindungstype	Datenbank-Standardwert

2. Testen Sie den Zugriff auf die soeben konfigurierte Datenbank, in dem Sie versuchen sich mit SQL*Plus zu verbinden.
 3. Lassen Sie sich den Inhalt der View V\$INSTANCE anzeigen, um sicher zu gehen, dass Sie mit der richtigen Datenbank verbunden sind.
 4. Konfigurieren Sie Static Service Registration für Ihre Datenbank.
 5. Stoppen und starten Sie Ihren Listener mittels des lsnrctl-Tools und prüfen Sie, ob Ihre Instanz auch wirklich statisch registriert wurde.
 6. Konfigurieren Sie Ihren Host so, dass Ihr Banknachbar sich per Hostnaming auf Ihre Datenbank verbinden kann.
 7. Konfigurieren Sie die Benennungsmethode Directory Naming für Ihre Datenbank. Der Directory Server ist ein OID und wird unter der IP-Adresse 192.168.111.250 zufinden sein (Ports: 389 und 636). Der LDAP-Context lautet: „cn=OracleContext“. Nutzen Sie für diese Aufgabe den Oracle Net Configuration Assistant (netca - Konfigurieren von Directoy-Verwendung).
 8. Recherchieren Sie, wie Sie beim Local Naming eine Client-Connection explizit als Dedicated Server Verbindung oder als DRCP Verbindung konfigurieren können.
-
-
-

9. Konfigurieren Sie zwei Net Service Names, einen für eine Dedicated Server Verbindung und einen für eine DRCP-Verbindung. Testen Sie beide Verbindungen (Hinweis: Um die DRCP-Verbindung testen zu können, muss vorher das Connection Pooling aktiviert werden, siehe: [Database Resident Connection Pooling, Database Concepts 11g Release 2, Kapitel 9](#))

23.6 Lösung - Konfigurieren der Oracle-Netzwerkumgebung

- Konfigurieren Sie Ihre Namensauflösung so, dass Sie auf die Datenbank Ihres Banknachbarn zugreifen können. Verwenden Sie die folgenden Einstellungen zur Konfiguration der Namensauflösung:

Objekt	Einstellung
Net Service Name	orclneighbour
Protokol	TCP/IP
Host	IP/Hostname Ihres Nachbarn
Port	1521
Service-Name	SID Ihres Nachbarn
Verbindungstype	Datenbank-Standardwert

Listing 23.16: tnsnames.ora

```
ORCLNEIGHBOUR =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = FEA11-119SRV.oracle.com)
       (PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = orcl)
    )
  )
```

- Testen Sie den Zugriff auf die soeben konfigurierte Datenbank, in dem Sie versuchen sich mit SQL*Plus zu verbinden.

```
[oracle@FEA11-119SRV ~]$ sqlplus sys/oracle@ORCLNEIGHBOUR as sysdba
```

- Lassen Sie sich den Inhalt der View V\$INSTANCE anzeigen, um sicher zu gehen, dass Sie mit der richtigen Datenbank verbunden sind.

```
SQL> SELECT *
  2  FROM v$instance;
```

4. Konfigurieren Sie Static Service Registration für Ihre Datenbank.

Listing 23.17: listener.ora

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = orcl)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/ORCL)
      (SID_NAME = ORCL)
    )
  )
LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = FEA11-119SRV.oracle.com)(PORT = 1521))
  )

ADR_BASE_LISTENER = /u01/app/oracle
```

5. Stoppen und starten Sie Ihren Listener mittels des lsnrctl-Tools und prüfen Sie, ob Ihre Instanz auch wirklich statisch registriert wurde.

```
[oracle@FEA11-119SRV ~]$ lsnrctl

LSNRCTL> stop
Connecting to ...
The command completed successfully
LSNRCTL> start
Starting /u01/app/oracle/product/11.2.0/ORCL/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 11.2.0.1.0 - Production
System parameter file is $ORACLE_HOME/network/admin/listener.ora
Log messages written to /u01/app/oracle/diag/tnslsnr/.../log.xml
Listening on: ...
Connecting to ...
STATUS of the LISTENER
-----
Alias                      LISTENER
Version                    TNSLSNR for Linux: Version 11.2.0.1.0 - Production
Start Date                 10-OCT-2013 11:00:25
Uptime                     0 days 0 hr. 0 min. 0 sec
Trace Level                off
Security                   ON: Local OS Authentication
SNMP                       OFF
Listener Parameter File   $ORACLE_HOME/network/admin/listener.ora
Listener Log File          /u01/app/oracle/diag/tnslsnr/.../log.xml
Listening Endpoints Summary...
...
Services Summary...
Service "orcl" has 1 instance(s).
  Instance "ORCL", status UNKNOWN, has 1 handler(s) for this service...
```

The command completed successfully

6. Konfigurieren Sie Ihren Host so, dass Ihr Banknachbar sich per Hostnaming auf Ihre Datenbank verbinden kann.

Listing 23.18: listener.ora

```
SID_LIST_LISTENER =
  (SID_LIST =
    (SID_DESC =
      (GLOBAL_DBNAME = FEA11-119SRV.oracle.com)
      (ORACLE_HOME = /u01/app/oracle/product/11.2.0/ORCL)
      (SID_NAME = ORCL)
    )
  )

LISTENER =
  (DESCRIPTION =
    (ADDRESS = (PROTOCOL = TCP)(HOST = FEA11-119SRV.oracle.com)(PORT = 1521))
  )

ADR_BASE_LISTENER = /u01/app/oracle
```

7. Konfigurieren Sie die Benennungsmethode Directory Naming für Ihre Datenbank. Der Directory Server ist ein OID und wird unter der IP-Adresse 192.168.111.250 zufinden sein (Ports: 389 und 636). Der LDAP-Context lautet: „cn=OracleContext“. Nutzen Sie für diese Aufgabe den Oracle Net Configuration Assistant (netca - Konfigurieren von Directoy-Verwendung).

Listing 23.19: ldap.ora

```
DEFAULT_ADMIN_CONTEXT = "cn=OracleContext"
DIRECTORY_SERVERS = (192.168.20.100:38911:63611)
DIRECTORY_SERVER_TYPE = OID
```

8. Recherchieren Sie, wie Sie beim Local Naming eine Client-Connection explizit als Dedicated Server Verbindung oder als DRCP Verbindung konfigurieren können.

9. Konfigurieren Sie zwei Net Service Names, einen für eine Dedicated Server Verbindung und einen für eine DRCP-Verbindung. Testen Sie beide Verbindungen (Hinweis: Um die DRCP-Verbindung testen zu können, muss vorher das Connection Pooling aktiviert werden, siehe: [Database Resident Connection Pooling, Database Concepts 11g Release 2, Kapitel 9](#))

Listing 23.20: tnsnames.ora

```
ADMIN_ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)
        (HOST = FEA11-119SRV.oracle.com)
        (PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = orcl)
    )
  )

ORCL =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)
        (HOST = FEA11-FEA11-119SRV.oracle.com)
        (PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = POOLED)
      (SERVICE_NAME = orcl)
    )
  )
```

Listing 23.21: Connection Pooling aktivieren in SQL*Plus

```
SQL> exec DBMS_CONNECTION_POOL.Start_Pool();
```


24 Verwalten des Result Caches

Inhaltsangabe

Zusammen mit der Version 11g seiner Datenbank hat Oracle ein neues Feature auf den Markt gebracht, dass unter dem Namen Result Cache firmiert. Es handelt sich dabei um eine SGA-Komponente, die im Shared Pool angesiedelt ist. Ihre Aufgabe ist es, die Arbeit mit häufig wiederkehrenden SQL-Statements bzw. PL/SQL-Funktionen zu beschleunigen, in dem die Ergebnisse von SQL-Abfragen zwischengespeichert und direkt von dort wieder ausgeliefert werden. Der Result Cache unterscheidet sich vom Library Cache, da er keine Ausführungspläne sondern ganze Result Sets (Ergebniszeilen) puffert.



Um dieses Feature sinnvoll nutzen zu können, muss der Datenbankserver über eine große Menge Arbeitsspeicher verfügen.

24.1 Bestandteile des Result Caches

24.1.1 SQL Query Result Cache

Der SQL Query Result Cache ist eine der beiden serverseitigen Komponenten, die den Result Cache bilden. Seine Aufgabe ist die sessionübergreifende Zwischenspeicherung von SQL-Result Sets. Mit seiner Hilfe reduziert sich die Ausführungszeit, eines wiederholten SQL-Statements, meist auf den Bruchteil einer Sekunde. Ändert sich jedoch etwas an den Basistabellen der Abfrage, wird das im Cache gespeicherte Result Set automatisch invalidiert und das Ergebnis muss neu berechnet werden.



Optimal einsetzbar ist der SQL Query Result Cache nur bei Tabellen mit geringer Volatilität^a.

^aVolatilität = Änderungshäufigkeit

Einmal konfiguriert, steht der SQL Query Result Cache allen Anwendungen transparent zur Verfügung. Das heißt, dass sich durch seine Nutzung nichts an den SQL-Statements der Anwendungen ändert.

24.1.2 Der PL/SQL Function Cache

Der PL/SQL-Funktion Cache ist der Bruder des SQL Query Result Caches. Er speichert Ausführungsergebnisse von PL/SQL-Funktionen. Damit eine Funktion diesen Cache nutzt, muss sie in ihrer Definition die Klausel `RESULT_CACHE RELIES_ON (tabellenliste)` enthalten.

24.2 Arbeiten mit dem Result Cache

24.2.1 Aktivieren des Result Caches

Es existieren insgesamt 4 Initialisierungsparameter, zur Konfiguration des Result Caches.

Listing 24.1: Die Result Cache Parameter

```
SQL> col name format a30
SQL> SELECT name, issys_modifiable
  2  FROM v$system_parameter
  3 WHERE LOWER(name) LIKE 'result%';

NAME                      ISSYS_MOD
-----
result_cache_mode          IMMEDIATE
result_cache_max_size      IMMEDIATE
result_cache_max_result    IMMEDIATE
result_cache_remote_expiration IMMEDIATE
```

Aktiviert wird der Result Cache durch das Setzen des Parameters `result_cache_max_size`, auf einen Wert größer 0.

Listing 24.2: Den Result Cache aktivieren

```
SQL> col value format a10
SQL> col name format a30
SQL> SELECT name, value
  2  FROM v$system_parameter
  3 WHERE LOWER(name) LIKE 'result_cache_max_size';

NAME                  VALUE
-----
result_cache_max_size 2097152

SQL> ALTER SYSTEM
  2  SET result_cache_max_size = 300M;

System altered.
```

Um nun zu prüfen, ob der Result Cache tatsächlich aktiviert wurde, kann die Funktion `STATUS()` aus dem PL/SQL-Paket `DBMS_RESULT_CACHE` ausgeführt werden.

Listing 24.3: Welchen Status hat der Result Cache?

```
SQL> col status format a10
SQL> SELECT DBMS_RESULT_CACHE.STATUS() AS status
  2  FROM dual;
```

STATUS

ENABLED

Der `result_cache_max_size` Parameter beeinflusst die Größe des Result Caches. Hat er den Wert 0, ist der Result Cache deaktiviert. Wird das Automatic Memory Management genutzt (Parameter `memory_target`), aktiviert Oracle den Result Cache automatisch und es werden bis zu 25 % von `memory_target` für ihn reserviert. Ist der Parameter `sga_target` in Benutzung, wird Oracle bis zu 0,5 % von `sga_target` reservieren. Wird der Shared Pool direkt durch `shared_pool_size` dimensioniert, allokiert Oracle bis zu 1 % von `shared_pool_size` für den Result Cache.

Insgesamt werden aber nie mehr als 75 % des Shared Pools für den Result Cache freigegeben.

24.2.2 Konfigurieren des Result Caches

Result_Cache_Mode

Neben `result_cache_max_size` existieren noch drei weitere Parameter, die zur Konfiguration des Result Caches dienen. `result_cache_mode` legt fest, ob der Optimizer automatisch versuchen soll, alle Result Sets im Cache zu speichern (Force) oder ob dies bei jedem Statement einzeln festgelegt werden muss (Manual). Der Standardwert für diesen Parameter ist „manual“.

Um ein Result Set manuell im Result Cache abzulegen, muss der Result Cache Hint `/*+ result_cache */`, im SQL-Statement angegeben werden.

Listing 24.4: Den Result Cache manuell benutzen

```
SQL> set autotrace traceonly explain
SQL> SELECT /*+ result_cache */ vorname, nachname
  2  FROM bank.mitarbeiter;

Execution Plan
-----
Plan hash value: 414804864
-----
| Id | Operation          | Name           | Rows | Bytes | Cost (%CPU) | Time      |
| 0  | SELECT STATEMENT   |                | 100  | 1500  |    3  (0)  | 00:00:01  |
| 1  | RESULT CACHE       | c4y4n5s0cg4tbpabs63n3u2k5sal |       |       |            |           |
| 2  | TABLE ACCESS FULL  | MITARBEITER   | 100  | 1500  |    3  (0)  | 00:00:01  |

Result Cache Information (identified by operation id):
```

```

1 - column-count=2; dependencies=(BANK.MITARBEITER);
  name="SELECT /*+ result_cache */ vorname, nachname
FROM   bank.mitarbeiter"

```

Dass das Result Set des Statements, aus Beispiel ??, tatsächlich im SQL Query Result Cache gespeichert wurde, ist aus der zweiten Zeile des Ausführungsplanes ersichtlich. Die Angabe: „RESULT CACHE cy4n5s0cg4tbpabs63n3u2k5sa“ zeigt an, dass unter der ID „cy4n5s0cg4tbpabs63n3u2k5sa“, ein Statement im Result Cache abgelegt wurde. Mit Hilfe dieser ID kann in der View V\$RESULT_CACHE_OBJECTS nach dem dazugehörigen SQL-Statement gesucht werden.

Listing 24.5: Objekte im Result Cache suchen

```

SQL> col name format a35
SQL> col namespace format a9
SQL> col cache_id format a40
SQL> set linesize 100
SQL> SELECT type, name, namespace, row_count, cache_id
  2  FROM v$result_cache_objects
  3 WHERE cache_id = 'cy4n5s0cg4tbpabs63n3u2k5sa';

TYPE          NAME                      NAMESPACE  ROW_COUNT
-----        -----
CACHE_ID

-----          -----
Result        SELECT /*+ result_cache */ vorname, SQL           100
              nachname
              FROM   bank.mitarbeiter
cy4n5s0cg4tbpabs63n3u2k5sa

```

Soll der Query Optimizer versuchen alle SQL Result Sets im Result Cache abzulegen, muss der Parameter result_cache_mode auf den Wert „force“ eingestellt werden. Ein solches Vorgehen dürfte jedoch in nur ganz wenigen Fällen sinnvoll sein.

Result_Cache_Max_Result

Mit result_cache_max_result wird die maximale Größe eines Result Sets, prozentual von result_cache_max_size festgelegt. Statements deren Result Sets mehr Speicherplatz benötigen, als result_cache_max_result freigibt, werden nicht im Result Cache gespeichert.

Mit result_cache_max_result = 5 wurde festgelegt, dass ein einzelnes Result Set nicht mehr als 5 % der Gesamtgröße des Result Caches (hier 5 % von 300M = 15M) verwenden darf. Der Ausführungsplan aus

Beispiel ?? zeigt, dass das Statement 268 MB Arbeitsspeicher benötigen würde. Daher wird es nicht im Result Cache aufgenommen.

Listing 24.6: Maximalgröße von Objekten im Result Cache

```

SQL> ALTER SYSTEM
2  SET result_cache_max_result = 5;

SQL> set autotrace traceonly explain
SQL> SELECT *
2  FROM bank.mitarbeiter a, bank.mitarbeiter b, bank.mitarbeiter c;

Execution Plan
-----
Plan hash value: 2203449472

-----
| Id | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
|---|---|---|---|---|---|---|
| 0 | SELECT STATEMENT   |           | 1000K | 268M | 13707  (1)| 00:02:45 |
| 1 |  MERGE JOIN CARTESIAN |          | 1000K | 268M | 13707  (1)| 00:02:45 |
| 2 |    MERGE JOIN CARTESIAN |          | 10000 | 1835K | 140   (0)| 00:00:02 |
| 3 |      TABLE ACCESS FULL | MITARBEITER | 100   | 9400  |       3  (0)| 00:00:01 |
| 4 |      BUFFER SORT     |           | 100   | 9400  |     137  (0)| 00:00:02 |
| 5 |      TABLE ACCESS FULL | MITARBEITER | 100   | 9400  |       1  (0)| 00:00:01 |
| 6 |      BUFFER SORT     |           | 100   | 9400  | 13705  (1)| 00:02:45 |
| 7 |      TABLE ACCESS FULL | MITARBEITER | 100   | 9400  |       1  (0)| 00:00:01 |
-----
```

DBMS_RESULT_CACHE

Das PL/SQL-Paket DBMS_RESULT_CACHE enthält eine Reihe von Funktionen zur Administration des Result Caches.

- FLUSH(): Löscht den kompletten Inhalt des Result Caches
- STATUS(): Zeigt den Status (enabled/disabled) des Result Caches an
- MEMORY_REPORT(): Zeigt einen Nutzungsbericht zum Result Cache an

24.2.3 Einschränkungen bei der Nutzung des Result Caches

Ob der Result Cache für das Result Set eines SQL-Statements genutzt werden kann, ist von verschiedenen Faktoren abhängig. Er kann nicht genutzt werden wenn:

- die Abfrage Tabellen des Data Dictionary oder temporäre Tabellen enthält,

- eine Sequenz im SQL-Statement genutzt wird,
- nicht deterministische SQL- oder PL/SQL-Funktionen genutzt werden.



Der Begriff „deterministisch“ bedeutet, dass das Ergebnis einer Funktion, bei gleichbleibenden Eingabeparametern, unveränderlich ist. Beispielsweise ist das Ergebnis von `SQRT(9)` immer 3, da die Quadratwurzel von 9 unveränderlich ist. Das Ergebnis der Funktion `SYSTIMESTAMP` ändert sich jedoch mit jeder Nanosekunde. Daher gilt sie als nichtdeterministisch.

24.2.4 Monitoring des SQL Query Result Caches

Für die Überwachung des Result Caches gibt es zwei Quellen:

- Die View `V$RESULT_CACHE_STATISTICS`
- Die Funktion `DBMS_RESULT_CACHEMEMORY_REPORT()`

Die View `V$RESULT_CACHE_STATISTICS` zeigt eine Kurzform der Ausgabe der Funktion `DBMS_RESULT_CACHEMEMORY_REPORT()`.

Listing 24.7: `V$RESULT_CACHE_STATISTICS`

ID	NAME	VALUE
1	Block Size (Bytes)	1024
2	Block Count Maximum	307200
3	Block Count Current	1952
4	Result Size Maximum (Blocks)	15360
5	Create Count Success	4
6	Create Count Failure	0
7	Find Count	0
8	Invalidation Count	0
9	Delete Count Invalid	1
10	Delete Count Valid	0
11	Hash Chain Length	1

Das Ergebnis aus Beispiel ?? sagt aus, dass der Result Cache aus 1 KB großen Blöcken besteht (Block Size (Bytes)) und das er 307200 Blöcke umfasst (Block Count Maximum). Von diesen 307200 Blöcken sind derzeit 1952 allokiert (Block Count Current). Ein Statement darf maximal 15360 Blöcke umfassen (Result Size Maximum (Blocks)). Der Wert „Create Count Success“ sagt aus, dass sich aktuell 4 Statements im

Cache befinden. Einen etwas ausführlicheren Überblick bekommt man, wenn man die PL/SQL-Funktion DBMS_RESULT_CACHEMEMORY_REPORT() ausführt.

Listing 24.8: Der Result Cache Report - DBMS_RESULT_CACHE.MEMORY_REPORT()

```

SQL> set serveroutput on
SQL> exec DBMS_RESULT_CACHE.MEMORY_REPORT();
R e s u l t   C a c h e   M e m o r y   R e p o r t
[Parameters]
Block Size          = 1K bytes
Maximum Cache Size = 300M bytes (300K blocks)
Maximum Result Size = 15M bytes (15K blocks)
[Memory]
Total Memory = 2194328 bytes [0.872% of the Shared Pool]
... Fixed Memory = 10696 bytes [0.004% of the Shared Pool]
... Dynamic Memory = 2183632 bytes [0.868% of the Shared Pool]
..... Overhead = 184784 bytes
..... Cache Memory = 1952K bytes (1952 blocks)
..... Unused Memory = 715 blocks
..... Used Memory = 1237 blocks
..... Dependencies = 1 blocks (1 count)
..... Results = 1236 blocks
..... SQL = 2 blocks (1 count)
..... Invalid = 1234 blocks (2 count)

PL/SQL procedure successfully completed.

```

Tabelle 24.1: MEMORY_REPORT und V\$RESULT_CACHE_STATISTICS im Vergleich

V\$RESULT_CACHE_STATISTICS	MEMORY_REPORT
Block Size (Bytes) 1024	Block Size = 1K bytes
Block Count Maximum 307200	Maximum Cache Size = 300M bytes (300K blocks)
Block Count Current 1952	Cache Memory = 1952K bytes (1952 blocks) Unused Memory = 715 blocks Used Memory = 1237 blocks
Result Size Maximum (Blocks) 15360	Maximum Result Size = 15M bytes (15K blocks)
Create Count Success 4	Dependencies = 1 blocks (1 count) SQL = 2 blocks (1 count) Invalid = 1234 blocks (2 count)
Delete Count Invalid 1	N. a.

Tabelle ?? zeigt, dass im Bericht aus Beispiel ?? die gleichen Werte wiederzufinden sind, wie in der View V\$RESULT_CACHE_STATISTICS.

24.3 Informationen

24.3.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10285]
- [REFRN10256]
- [REFRN10202]
- [REFRN10298]
- [REFRN10272]
- [REFRN10270]
- [REFRN10294]

24.3.2 Verzeichnis der relevanten Data Dictionary Views



- [REFRN30438]
- [REFRN30439]

24.4 Übungen - Verwalten des Result Caches

1. Ermitteln Sie, ob der serverseitige Result Cache aktiviert ist!
-

2. Bestimmen Sie die aktuelle Größe des serverseitigen Result Caches!
-

3. Konfigurieren Sie, für den serverseitigen Result Cache, eine Größe von 1G! Sofern dies nicht möglich sein sollte, begründen Sie warum nicht!

4. Prüfen Sie in welchem Modus der Result Cache läuft (Manual oder Force)
-

5. Schalten Sie das SQL*Plus-Timing ein: `set timing on`

6. Schalten Sie das SQL-Autotracing ein: `set autotrace traceonly explain`

7. Setzen Sie das folgende SQL-Statement, unter Benutzung des Result Caches ab. Da Sie dieses Statement mehrfach ausführen müssen, sollten Sie es in einer SQL-Datei speichern.

```
SQL> SELECT /*+ result_cache */ *
  2  FROM   (SELECT TO_CHAR(Buchungsdatum, 'Q') AS Quartal,
  3                  TO_CHAR(Buchungsdatum, 'YYYY') AS Datum, Betrag
  4                FROM bank.Buchung)
  5  PIVOT (SUM(Betrag) FOR Datum IN ('1985', '1986', '1987', '1988',
  6                                    '1989'));
```

8. Welche Kosten hat das Statement verursacht und wie viel Zeit wurde für die Ausführung benötigt?
-
-

9. Führen Sie das gleiche Statement nochmals aus und ermitteln Sie erneut die verursachten Kosten und die benötigte Ausführungszeit! Haben sich die Kosten verändert?
-
-

-
10. Schalten Sie das SQL-Autotracing und das Timing wieder aus: `set autotrace off` und `set timing off`

11. Führen Sie das SQL-Statement erneut aus und prüfen Sie:

- Wie viele Result Sets sich aktuell im Cache befinden
-

- Wie groß ein Result Set maximal sein darf
-

24.5 Lösungen - Verwalten des Result Caches

1. Ermitteln Sie, ob der serverseitige Result Cache aktiviert ist!

```
SQL> SELECT DBMS_RESULT_CACHE.STATUS() AS cache_status
  2  FROM dual;

CACHE_STATUS
-----
ENABLED
```

2. Bestimmen Sie die aktuelle Größe des serverseitigen Result Caches!

```
SQL> col name format a30
SQL> col value format a20
SQL> SELECT *
  2  FROM v$result_cache_statistics
  3  WHERE id IN (1, 3);
```

3. Konfigurieren Sie, für den serverseitigen Result Cache, eine Größe von 1G! Sofern dies nicht möglich sein sollte, begründen Sie warum nicht!

```
SQL> ALTER SYSTEM
  2  SET result_cache_max_size = 1G;
```

4. Prüfen Sie in welchem Modus der Result Cache läuft (Manual oder Force)

```
SQL> col name format a30
SQL> col modus format a30
SQL> SELECT name, value as Modus
  2  FROM v$system_parameter
  3  WHERE LOWER(name) LIKE 'result_cache_mode';

NAME                MODUS
-----
result_cache_mode      MANUAL
```

5. Schalten Sie das SQL*Plus-Timing ein: `set timing on`

6. Schalten Sie das SQL-Autotracing ein: `set autotrace traceonly explain`

7. Setzen Sie das folgende SQL-Statement, unter Benutzung des Result Caches ab. Da Sie dieses Statement mehrfach werden ausführen müssen, sollten Sie es in einer SQL-Datei speichern.

```
SQL> SELECT /*+ result_cache */ *
```

```

2   FROM   (SELECT TO_CHAR(Buchungsdatum, 'Q') AS Quartal,
3                  TO_CHAR(Buchungsdatum, 'YYYY') AS Datum, Betrag
4            FROM bank.Buchung)
5   PIVOT (SUM(Betrag) FOR Datum IN ('1985', '1986', '1987',
6                                    '1988', '1989'));

```

8. Welche Kosten hat das Statement verursacht und wie viel Zeit wurde für die Ausführung benötigt?

Listing 24.9: Der Ausführungsplan wurde gekürzt

Id	Operation	Name	Cost (%)CPU)
0	SELECT STATEMENT		1404 (2)
1	RESULT CACHE	4ygxpvhbjy9q7ggm3jz5p8vh50	
2	HASH GROUP BY PIVOT		1404 (2)
3	TABLE ACCESS FULL	BUCHUNG	566 (1)

Elapsed: 00:00:00.20

9. Führen Sie das gleiche Statement nochmals aus und ermitteln Sie erneut die verursachten Kosten und die benötigte Ausführungszeit! Haben sich die Kosten verändert?

Listing 24.10: Der Ausführungsplan wurde gekürzt

Id	Operation	Name	Cost (%)CPU)
0	SELECT STATEMENT		1404 (2)
1	RESULT CACHE	4ygxpvhbjy9q7ggm3jz5p8vh50	
2	HASH GROUP BY PIVOT		1404 (2)
3	TABLE ACCESS FULL	BUCHUNG	566 (1)

Elapsed: 00:00:00.03

10. Schalten Sie das SQL-Autotracing und das Timing wieder aus: `set autotrace off` und `set timing off`

11. Führen Sie das SQL-Statement erneut aus und prüfen Sie:

- Wie viele Result Sets sich aktuell im Cache befinden
-
- Wie groß ein Result Set maximal sein darf


```
SQL> col name format a30
SQL> col value format a30
SQL> SELECT *
  2  FROM    v$result_cache_statistics
  3  WHERE   id <= 5;

          ID NAME                  VALUE
----- -----
  1 Block Size (Bytes)           1024
  2 Block Count Maximum        327680
  3 Block Count Current         32
  4 Result Size Maximum (Blocks) 16384
  5 Create Count Success          1
```

25 Implementing Security

Inhaltsangabe

25.1 Database Auditing

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 ist 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 Oracle gibt es fünf Arten des Auditings:

- **Statementauditing:** Die Nutzung bestimmter SQL-Statements wird überwacht.
- **Privilegeauditing:** Die Nutzung von Privilegien wird überwacht.
- **Objectauditing:** Der Zugriff auf bestimmte Objekte wird überwacht.
- **Networkauditing:** Überwachung von Fehlern im Netzwerk
- **Standardauditing:** Standardmäßige Überwachung, die immer aktiv ist.



Jeder Datenbankadministrator kann das Auditing jederzeit aktivieren.

25.1.1 Grundsätze und Vorgehensweisen beim Auditing

Oracle 11g ermöglicht es Auditinginformationen in den Auditingtrail der Datenbank oder den des Betriebssystems (Windowsereignisanzeige oder SysLog Daemon) zu schreiben. Je nachdem welche Nutzergruppe überwacht werden muss, normale Nutzer oder Administratoren, empfiehlt es sich einen adäquaten Auditingtrail auszuwählen, so dass dieser nicht durch den betroffenen Nutzerkreis verändert/gefalscht werden

kann. Datenbankadministratoren z. B. haben in einer Oracledatenbank unbeschränkten Zugriff, weshalb ein Sammeln der Auditingdaten innerhalb der Datenbank nicht sinnvoll ist.

Auditinginformationen überschaubar halten

Auch wenn Auditing keine großen Kosten erzeugt, sollte die Menge der Auditinginformationen auf das Notwendigste beschränkt werden. Auf diese Weise soll der Auditingtrail überschaubar gehalten und eine einfache Auswertung der Informationen ermöglicht werden.

Das Wichtigste für die Entwicklung einer Auditingstrategie ist: Der genaue Zweck des Auditings muss bestimmt werden. Erst wenn festgelegt wurde, warum die Datenbank überwacht werden muss, kann eine zielführende Überwachungsstrategie entwickelt werden.

Wenn das Ziel des Auditings das Sammeln von historischen Informationen über die Datenbank ist, sollten folgende Regeln eingehalten werden:

- Auditinginformationen archivieren und den Auditingtrail leeren

Wurden alle benötigten Informationen gesammelt, sollten diese archiviert und der Auditingtrail geleert werden, damit dieser später für neue Überwachungen zur Verfügung steht.

- Datenschutzrichtlinien beachten

Das Bundesdatenschutzgesetz (BDSG) regelt sehr genau, auf welche Weise Daten erhoben werden dürfen wie lange diese gespeichert werden dürfen und wer in welchem Falle Zugriff auf die gesammelten Daten hat.

Überwachen von verdächtigen Aktivitäten

Beim Überwachen verdächtiger Aktivitäten sollte wie folgt vorgegangen werden:

- Erst umfassend und dann immer spezieller auditieren

Wenn verdächtige Aktivitäten in einem System, wie z. B. einer Datenbank, festgestellt werden, liegen meist nicht viele Informationen über diese vor. Daher muss am Anfang ein umfassenderes Auditing stattfinden, bis alle notwendigen Informationen gesammelt wurden, um die Überwachung auf bestimmte Objekte, Nutzer oder Aktivitäten einzchränken zu können.

Die Auditinginformationen sollten immer wieder dahingehend ausgewertet werden, ob nicht eine weitere Spezialisierung des Auditings stattfinden kann.

- Absichern des Auditingtrails vor unauthorisierten Zugriffen

Der Auditingtrail sollte vor unerwünschten Veränderungen von außerhalb geschützt werden, z. B. durch eine Überwachung des Auditingtrails.

Überwachen administrativer Nutzer

Sessions des Nutzers SYS oder anderer Nutzer, die sich mit den Zusätzen `as sysdba` oder `as sysoper` anmelden, werden standardmäßig nicht auditiert. Ist es notwendig, eine Überwachung für diese Nutzer zu aktivieren, geschieht dies mit Hilfe des Initialisierungsparameters `audit_sys_operations` (statischer Parameter). Der Standardwert für diesen Parameter ist „false“. Wird dieser Wert auf „true“ geändert, aktiviert sich dadurch das Auditing für administratives Personal.



Alle Auditinginformationen über den Nutzer SYS werden im Auditingtrail des Betriebssystems und nicht in der Datenbank aufbewahrt.

25.1.2 Der Audittrail

Informationen im Auditingtrail

Ein Audittrail enthält unterschiedliche Informationen, abhängig davon, welche Ereignisse überwacht werden und wie das Auditing konfiguriert wurde. Der folgende Literaturhinweis zeigt eine Auflistung der Informationen, die immer im Auditingtrail, sowohl in der Datenbank, also auch im Betriebssystem, gespeichert werden.



- [BCGIDBFI]

Ist es dem Audittrail nicht mehr möglich neue Einträge aufzunehmen, kann eine überwachte Aktion nicht durchgeführt werden und bricht mit einer Fehlermeldung ab.

Der Auditingtrail enthält keine Informationen über Werte, die in einem SQL-Statement verwendet wurden. Wird beispielsweise ein **UPDATE**-Statement überwacht, werden die alten und neuen Werte der Tabelle nicht mit aufgezeichnet. Dies kann jedoch durch das sogenannte „Fine-Grained-Auditing“ erreicht werden.

Die Wahl des Audittrails

Welcher Auditingtrail verwendet werden sollte, ist von verschiedenen Faktoren abhängig:

- Welche Aktivitäten sollen überwacht werden?
- Kann Oracle den Auditingtrail des verwendeten Betriebssystems benutzen?
- Welcher Auditingtrail kann besser gegen unerwünschte Zugriffe gesichert werden?

Beide Audittrails haben Vor- und Nachteile, die bei der Wahl berücksichtigt werden sollten. Die folgende Liste ist eine Gegenüberstellung der Vorteile beider Trails.

Datenbankauditingtrail:

- Die Ergebnisse des Auditings können, mit Hilfe von SQL oder PL/SQL und den vordefinierten Data Dictionary Views, schnell und einfach, in der Datenbank, analysiert werden.
- Es kann sehr einfach ein Report über die Auditinformationen aus der Datenbank erzeugt werden.

Betriebssystemauditingtrail:

- Auditinformationen die im Auditingtrail des Betriebssystems geschrieben wurden, sind unter Umständen sicherer als im Datenbankauditingtrail, da für den Zugriff auf den Trail des Betriebssystems Berechtigungen benötigt werden, die meist nur ein Systemadministrator hat.
- Die Informationen haben eine größere Verfügbarkeit, da sie auch dann noch erreichbar sind, wenn die Datenbank heruntergefahren wurde.
- Die Auditinformationen können als XML-Dateien an einen sicheren Ort im Netzwerk geschrieben werden. Mit Hilfe der View `V$XML_AUDIT_TRAIL` können diese Dateien ganz einfach durch SQL-Kommandos abgefragt werden.

- Die Nutzung des Betriebssystemauditingtrails konsolidiert alle Auditinginformationen. Es können auf diese Art und Weise alle Auditinginformationen, aller Anwendungen an einer Quelle zusammengeführt werden, was eine zentrale Auswertung ermöglicht.

Konfiguration des Auditingtrails

Der statische Parameter `audit_trail` legt fest, welcher Audittrail genutzt wird. Er kann folgende Werte annehmen:

- **NONE**: Das Auditing ist deaktiviert, da kein Auditingtrail verwendet wird.
- **DB**: Standardverhalten. Es wird der Auditingtrail der Datenbank benutzt, mit Ausnahme der Informationen, die immer in den Betriebssystemauditingtrail geschrieben werden.
- **DB, EXTENDED**: Es wird der Auditingtrail der Datenbank benutzt und es werden zusätzliche Informationen gespeichert.
- **OS**: Alle Auditinginformationen werden in den Betriebssystemauditingtrail geschrieben.
- **XML**: Wie OS, nur das alle Informationen in XML-Dateien geschrieben werden.
- **XML, EXTENDED** Wie XML, aber es werden zusätzliche Informationen gespeichert.



Der Hauptunterschied zwischen den beiden Einstellungen „DB“ und „XML“ zu ihren „EXTENDED“ Varianten ist, dass bei „EXTENDED“ der genaue Wortlaut des überwachten SQL-Statements mitgespeichert wird.

Listing 25.1: Der Parameter `audit_trail`

```
SQL> ALTER SYSTEM
  2  SET audit_trail=XML , EXTENDED SCOPE=spfile;
```

Einen Speicherort für den Betriebssystemauditingtrail festlegen

Wenn der Parameter `audit_trail` einen der beiden Werte „OS“ oder „XML“ hat, legt der Parameter `audit_file_dest` ein Verzeichnis auf dem Datenträger fest, in dem die Datenbank ihre Auditinginformationen

ablegt. Der audit_file_dest-Parameter ist dynamisch und kann mit Hilfe des Kommando `ALTER SYSTEM` wie folgt geändert werden:

Listing 25.2: Der Parameter audit_file_dest

```
SQL> ALTER SYSTEM
2 SET audit_file_dest='/u01/app/oracle' DEFERRED;
```



- [i2282157]

Der Standardwert für den Parameter audit_file_dest ist:

- Windows: %ORACLE_BASE%\admin\%ORACLE_SID%\adump
- Linux/Unix: \$ORACLE_BASE/admin/\$ORACLE_SID/adump



- [085271]

25.1.3 Standardauditing

Unabhängig davon, ob das Datenbankauditing aktiviert wurde oder nicht, führt Oracle standardmäßig ein Auditing für bestimmte Aktivitäten durch und schreibt die Informationen in den Auditingtrail des Betriebssystems. Dies wird als „Standardauditing“ bezeichnet. Es handelt sich dabei um die folgenden Aktivitäten:

- Verbindungen zur Instanz mit administrativen Privilegien

Es wird ein Eintrag im Betriebssystemauditingtrail erzeugt, der den Nutzernamen des Betriebssystemnutzers enthält, der sich mit einem der Zusätze `as sysdba` oder `as sysoper` angemeldet hat.

- Hochfahren der Instanz

Es wird ein Eintrag im Betriebssystemauditingtrail erzeugt, der Informationen über den Nutzer enthält, der das Hochfahren der Instanz veranlasst hat.

- Herunterfahren der Instanz

Es wird ein Eintrag im Betriebssystemauditingtrail erzeugt, der Informationen über den Nutzer enthält, der das Herunterfahren der Instanz veranlasst hat.



Als Auditingtrail wird das System bezeichnet, welches die Auditinginformationen aufnimmt. Dies kann z. B. eine Datenbanktabelle, eine XML-Datei, die Windowsereignisanzeige oder der Linux SysLog Daemon sein.

25.1.4 Statementauditing aktivieren

Um Statementauditing zu konfigurieren, muss das **AUDIT**-Kommando zusammen mit einem Bezeichner aufgerufen werden, der vorgibt, welche Art von Statements zu auditieren sind.



Zum Überwachen von Systemprivilegien wird das Privileg `audit system` benötigt. Für Objektauditing wird das Privileg `audit any` benötigt.

Ein einfaches Beispiel für Statementauditing ist:

Listing 25.3: Statementauditing aktivieren

```
SQL> AUDIT TABLE;
```

Dieses Kommando sorgt dafür, dass alle `CREATE TABLE`-, `DROP TABLE`- und `TRUNCATE TABLE`-Statements audiert werden.



- [i2059073]

Mit Hilfe der `AUDITING BY`-Klausel kann das Auditing nach Nutzern eingeschränkt werden. Um beispielsweise alle `ALTER TABLE`-Statements des Nutzers BANK zu auditieren wird das **AUDIT**-Statement wie folgt abgewandelt:

Listing 25.4: Statementauditing auf Nutzer einschränken

```
SQL> AUDIT ALTER TABLE
  2  BY bank;
```

Um nur erfolgreiche `ALTER TABLE`-Statements des Nutzers BANK zu überwachen, wird zusätzlich die Option `WHENEVER SUCCESSFUL` verwendet.

Listing 25.5: Nur erfolgreiche Statements auditieren

```
SQL> AUDIT ALTER TABLE
  2  BY bank
  3  WHENEVER SUCCESSFUL;
```



Das Gegenstück zur Option `WHENEVER SUCCESSFUL` ist die Option `WHENEVER NOT SUCCESSFUL` (Standardwert).

Es ist möglich Auditingrichtlinien für mehrere Nutzer und mehrere Ereignisse zu kombinieren. In einem solchen Fall, wird dann eine Liste mit Benutzernamen in der `BY`-Klausel angegeben.

Listing 25.6: Statementauditingoptionen kombinieren

```
SQL> AUDIT TABLE , ALTER TABLE
  2  BY bank , hr
  3  WHENEVER SUCCESSFUL;
```



Eine neue Session bezieht ihre Auditingeinstellungen aus dem Data Dictionary. Diese Einstellungen bleiben während der gesamten Lebensdauer der Session erhalten. Änderungen an den Auditingeinstellungen werden für eine Session erst nach einem Neustart wirksam.

Um eine Übersicht darüber zu erhalten, welche Statementauditingoptionen derzeit aktiviert sind, kann die View `DBA_STMT_AUDIT_OPTS` abgefragt werden.

Listing 25.7: Aktivierte Statementauditingoptionen

USER_NAME	AUDIT_OPTION	SUCCESS	FAILURE
BANK	TABLE	BY ACCESS	BY ACCESS
HR	ALTER TABLE	BY ACCESS	BY ACCESS
BANK	ALTER TABLE	BY ACCESS	NOT SET
HR	TABLE	BY ACCESS	NOT SET
	TABLE	BY ACCESS	NOT SET

Beispiel ?? zeigt, dass die Auditingstatements aus den Beispielen Beispiel ??, Beispiel ??, Beispiel ?? und Beispiel ?? kumuliert werden. Für die Einträge drei, vier und fünf wurde jeweils die Klausel `WHENEVER SUCCESSFUL` angegeben, was am Wert „NOT SET“ in der Spalte FAILURE erkennbar ist.

25.1.5 Privilegeauditing aktivieren

Das Auditieren von Systemprivilegien funktioniert auf die gleiche Art und Weise wie das aktivieren von Statementauditing. Einziger Unterschied ist, dass dem `AUDIT`-Statement ein oder mehrere Systemprivilegien zur Überwachung übergeben werden müssen.

Listing 25.8: Ein einfaches Privilegiauditing konfigurieren

```
SQL> AUDIT connect;
```

Wie beim Statementauditing kann auch hier nach Nutzerkonten eingeschränkt werden.

Listing 25.9: Privilegiauditing nach Nutzern einschränken

```
SQL> AUDIT connect
  2 BY bank;
```

Die Angabe der Klauseln `WHENEVER SUCCESSFUL` und `WHENEVER NOT SUCCESSFUL` ist ebenfalls möglich.

Listing 25.10: Nur erfolglose Anmeldungen überwachen

```
SQL> AUDIT connect
  2 BY bank
  3 WHENEVER NOT SUCCESSFUL;
```

Die View `DBA_PRIV_AUDIT_OPTS` dient dazu, die gesetzten Privilegeauditingoptionen auszuwerten.

Listing 25.11: Aktivierte Statementauditingoptionen

USER_NAME	PRIVILEGE	SUCCESS	FAILURE
BANK	CREATE SESSION	NOT SET	BY ACCESS
	CREATE SESSION	BY ACCESS	BY ACCESS

25.1.6 Objectauditing aktivieren

Das Objectauditing ist dem Privilegeauditing sehr ähnlich. Statt Systemprivilegien werden Objektprivilegien verwendet und es müssen eines oder mehrere Objekte angegeben werden, auf die sich das Auditing bezieht.

Listing 25.12: Ein einfaches Objectauditing konfigurieren

```
SQL> AUDIT SELECT, INSERT, UPDATE, DELETE ON bank.mitarbeiter
  2 BY ACCESS
  3 WHENEVER NOT SUCCESSFUL;
```

Ein weiterer Unterschied zum Statement- oder Privilegeauditing ist, dass beim Objectauditing keine **AUDITING BY**-Klausel verwendet werden kann, was bedeutet, dass nicht nach Nutzern eingeschränkt werden kann. Neu beim Objectauditing ist, dass mit Hilfe der Angaben **BY SESSION** und **BY ACCESS** gesteuert werden kann, ob:

- **BY ACCESS**: Es wird bei jedem Auftreten eines Ereignisses ein Eintrag im Audittrail erzeugt.
- **BY SESSION**: Nur beim ersten Auftreten eines Ereignisses ein Eintrag im Audittrail erzeugt wird.

Listing 25.13: Nur eine Warnung pro Session für ein Ereignis

```
SQL> AUDIT select ON bank.Buchung
  2 BY SESSION
  3 WHENEVER SUCCESSFUL;
```

Auch für diese Art des Auditings existiert eine View, die eine Auswertung der Auditingoptionen erlaubt, **DBA_OBJ_AUDIT_OPTS**. Sie gestaltet sich jedoch gänzlich anders, als **DBA_STMT_AUDIT_OPTS** oder **DBA_PRIV_AUDIT_OPTS**.

In der View **DBA_OBJ_AUDIT_OPTS** existiert für jedes Objektprivileg eine eigene Spalte. Die Spaltenbezeichnungen sind Abkürzungen der betroffenen Privilegien. Beispielsweise beinhaltet die Spalte **ALT** die Objectauditingoptions für das Objektprivileg **alter** oder die Spalte **SEL** die Optionen für das **select**-Privileg.

Listing 25.14: Die View DBA_OBJ_AUDIT_OPTS

Name	Null?	Type
OWNER		VARCHAR2(30)
OBJECT_NAME		VARCHAR2(30)
OBJECT_TYPE		VARCHAR2(23)
ALT		VARCHAR2(3)
AUD		VARCHAR2(3)
COM		VARCHAR2(3)
DEL		VARCHAR2(3)
GRA		VARCHAR2(3)
IND		VARCHAR2(3)
INS		VARCHAR2(3)
LOC		VARCHAR2(3)
REN		VARCHAR2(3)
SEL		VARCHAR2(3)
UPD		VARCHAR2(3)
REF		CHAR(3)
EXE		VARCHAR2(3)
CRE		VARCHAR2(3)
REA		VARCHAR2(3)
WRI		VARCHAR2(3)
FBK		VARCHAR2(3)

Um die Auditingoptionen für die Objektprivilegien select, insert, update und delete abzufragen zu können, müssen die Spalten SEL, INS, UPD und DEL in die Abfrage einbezogen werden.

Listing 25.15: Objectauditingoptions abfragen

```
SQL> SELECT sel, ins, upd, del
  2  FROM    dba_obj_audit_opts;

SEL  INS  UPD  DEL
---  ---  ---  ---
S/S  A/-  -/S  A/A
```

Diese Angaben sind wie folgt zu interpretieren:

Der linke Buchstabe, steht für die Überwachung erfolgreich verlaufener Ereignisse (Klausel WHENEVER SUCCESSFUL)
Hier gibt es drei Möglichkeiten:

- S: Es ist eine Überwachung erfolgreicher Ereignisse im Modus BY SESSION konfiguriert.
- A: Es ist eine Überwachung erfolgreicher Ereignisse im Modus BY ACCESS konfiguriert.
- -: Es wurde keine Überwachung erfolgreicher Ereignisse konfiguriert.

Der rechte Buchstabe, steht für eine Überwachung erfolglos verlaufener Ereignisse(Klausel **WHENEVER NOT SUCCESSFUL**) Hier gibt es wiederum drei Möglichkeiten:

- **S**: Es ist eine Überwachung erfolgloser Ereignisse im Modus **BY SESSION** konfiguriert.
- **A**: Es ist eine Überwachung erfolgloser Ereignisse im Modus **BY ACCESS** konfiguriert.
- -: Es wurde keine Überwachung erfolgloser Ereignisse konfiguriert.

Das Ergebnis aus Beispiel ?? zeigt vier verschiedene Varianten:

- **sel (S/S)**: Für die Nutzung des **select**-Privileges sollen sowohl erfolgreiche, als auch erfolglose Ereignisse, im Modus **BY SESSION** überwacht werden.
- **ins (A/-)**: Für die Nutzung des **insert**-Privileges werden nur erfolgreiche Ereignisse, im Modus **BY ACCESS** überwacht. Es erfolgt keine Überwachung erfolgloser Ereignisse.
- **upd (-/S)**: Für das **update**-Privileg werden nur erfolglose Ereignisse, im Modus **BY SESSION**, überwacht. Es erfolgt keine Überwachung erfolgreicher Ereignisse.
- **del (A/A)**: Für die Nutzung des **delete**-Privileges sollen sowohl erfolgreiche, als auch erfolglose Ereignisse, im Modus **BY ACCESS** überwacht werden.

25.1.7 Audittrails auswerten

Genauso wichtig oder sogar noch wichtiger als die Auditingoptionen sind die Auditingergebnisse, die in den Audittrails stehen. Wie für viele andere Dinge, stellt auch hier Oracle entsprechende Views zur Verfügung. Die beiden wichtigsten sind **DBA_AUDIT_TRAIL** und **DBA_COMMON_AUDIT_TRAIL**.

DBA_AUDIT_TRAIL zeigt den Inhalt des Datenbankaudittrails für alle Auditingarten an. Um nach den unterschiedlichen Auditingarten zu untergliedern gibt es noch die folgenden Views:

- **DBA_AUDIT_STATEMENT**: Enthält alle Statementauditing-Einträge im Datenbankaudittrail.
- **DBA_AUDIT_OBJECT**: Enthält alle Objectauditing-Einträge im Datenbankaudittrail.
- **DBA_AUDIT_SESSION**: Enthält alle connect und disconnect Einträge im Datenbankaudittrail.



Statt DBA_AUDIT_TRAIL direkt abzufragen, sollten die spezialisierten Views genutzt werden, da diese die Informationen des Datenbankauditingtrails übersichtlicher darstellen.

Listing 25.16: Den Datenbankauditingtrail nach connects auswerten

```
SQL> col userhost format a30
SQL> SELECT username , userhost ,
       TO_CHAR(timestamp , 'DD.MM.YYYY HH24:MI') AS Time
  2 FROM dba_audit_session
  3 ORDER BY timestamp;

USERNAME          USERHOST                TIME
-----            -----
ALICE             FEA11-119WS03.oracle.com  11.09.2013 12:09
CHLOE             FEA11-119WS03.oracle.com  11.09.2013 12:12
BANK              FEA11-119WS03.oracle.com  18.09.2013 10:59
SYSTEM            FEA11-119WS03.oracle.com  18.09.2013 11:35
SYSTEM            FEA11-119WS03.oracle.com  18.09.2013 11:44
SYSTEM            FEA11-119WS03.oracle.com  18.09.2013 11:55
SYSTEM            FEA11-119WS03.oracle.com  18.09.2013 12:00
BANK              FEA11-119CL.oracle.com   14.10.2013 10:47
BANK              FEA11-119CL.oracle.com   14.10.2013 10:49
BANK              FEA11-119CL.oracle.com   14.10.2013 10:57
```

Beispiel ?? zeigt eine Auswertung des Audittrails bezüglich connect-Ereignissen.

Der Betriebssystemaudittrail, der XML-Audittrail und das Fine Grained Auditing werden nur in der View DBA_COMMON_AUDIT_TRAIL angezeigt, was bedeutet, dass die drei oben genannten Views DBA_AUDIT_STATEMENT, DBA_AUDIT_OBJECT und auch DBA_AUDIT_SESSION nicht funktionieren. Um die gleiche Auswertung zu erreichen, wie in Beispiel ??, müsste dann die Spalte ACTION der View DBA_COMMON_AUDIT_TRAIL genutzt werden, um nach connect und disconnect Ereignissen zu filtern.

Listing 25.17: Einen externen Audittrail auswerten

```
SQL> col db_user format a10
SQL> col userhost format a29
SQL> col action format 999999
SQL> col extended_timestamp format a31
SQL> SELECT db_user , userhost , action , extended_timestamp
  2 FROM dba_common_audit_trail
  3 WHERE action IN (100,101)
  4 ORDER BY extended_timestamp;
```

Listing 25.18: Einen externen Audittrail auswerten - Fortsetzung

DB_USER	USERHOST	ACTION	EXTENDED_TIMESTAMP
BANK	FEA11-119CL.oracle.com	101	14.10.13 10:57:52,477400 +02:00
BANK	FEA11-119CL.oracle.com	100	14.10.13 11:03:32,648556 +02:00
BANK	FEA11-119SRV.oracle.com	100	16.10.13 10:54:51,029779 +02:00
BANK	FEA11-119SRV.oracle.com	101	16.10.13 10:54:53,737176 +02:00
BANK	FEA11-119SRV.oracle.com	100	16.10.13 10:54:54,059818 +02:00
BANK	FEA11-119SRV.oracle.com	100	16.10.13 10:54:57,221702 +02:00
BANK	FEA11-119SRV.oracle.com	100	16.10.13 10:54:59,997771 +02:00
BANK	FEA11-119SRV.oracle.com	101	16.10.13 10:55:02,779921 +02:00

Die Werte 100 und 101 der Spalte ACTION stehen für connect (100) und disconnect (101). Sollte es so sein, dass der OS-Audittrail oder der XML-Audittrail genutzt werden, empfiehlt es sich für den Administrator, die drei Views DBA_AUDIT_STATEMENT, DBA_AUDIT_OBJECT und DBA_AUDIT_SESSION nach zu bauen. Als Beispiel wird hier die View COMMON_AUDIT_SESSION gezeigt, die alle connect und disconnect Ereignisse, auf Basis der View DBA_COMMON_AUDIT_TRAIL anzeigen.

Listing 25.19: Eine eigene View für dba_common_audit_trail

```
SQL> CREATE OR REPLACE VIEW common_audit_session
(DB_USER, USERHOST, ACTION, EXTENDED_TIMESTAMP, EXTENDED_ACTION)
  2 AS
  3   SELECT db_user, userhost, action, extended_timestamp,
  4         DECODE(action, 100, 'CONNECT',
  5                101, 'DISCONNECT',
  6                0, 'CONNECT AS SYSDBA')
  7   FROM dba_common_audit_trail
  8  WHERE action IN (0, 100, 101)
  9  ORDER BY extended_timestamp;

View created.
```

25.1.8 Auditing deaktivieren

Das Kommando **NOAUDIT** deaktiviert eine konfigurierte Auditingeinstellung wieder. Es können damit alle Auditingarten deaktiviert werden.



Das Kommando **NOAUDIT** kennt die beiden Optionen **BY ACCESS** und **BY SESSION** nicht. Deshalb können diese nicht verwendet werden.

Im Folgenden werden einige Beispiele für das **NOAUDIT**-Kommando gezeigt.

Listing 25.20: Statement- und Privilegeauditing deaktivieren

```
SQL> NOAUDIT TABLE;
SQL> NOAUDIT ALTER TABLE BY bank;
SQL> NOAUDIT ALTER TABLE BY bank WHENEVER SUCCESSFUL;
SQL> NOAUDIT connect;
```

Listing 25.21: Alle AUDIT ALL-Statements deaktivieren

```
SQL> NOAUDIT ALL;
SQL> NOAUDIT ALL BY bank;
SQL> NOAUDIT ALL BY bank WHENEVER SUCCESSFUL;
```



Um alle Privilegeauditings zu deaktivieren gibt es das Schlüsselwort **ALL PRIVILEGES**.

Listing 25.22: Alle Privilegeauditings deaktivieren

```
SQL> NOAUDIT ALL PRIVILEGES;
SQL> NOAUDIT ALL PRIVILEGES BY bank;
SQL> NOAUDIT ALL PRIVILEGES BY bank
  2 WHENEVER SUCCESSFUL;
```

Auch beim Objectauditing gibt es die Möglichkeit, alle Auditings direkt zu deaktivieren. Dies geschieht mit dem Statement **NOAUDIT ALL ON DEFAULT**.

Listing 25.23: Objectauditing deaktivieren

```
SQL> NOAUDIT DELETE ON bank.mitarbeiter;
SQL> NOAUDIT SELECT, UPDATE, INSERT ON bank.mitarbeiter;
SQL> NOAUDIT SELECT ON bank.mitarbeiter
  2 WHENEVER SUCCESSFUL;

SQL> NOAUDIT ALL ON bank.mitarbeiter;
SQL> NOAUDIT ALL ON DEFAULT;
```

25.2 Die Nutzerauthentifizierung sicherer gestalten

25.2.1 Sichere Authentifizierung

Brute Force Protection

Mit Oracle 11g kommt ein altes/neues Feature, welches als Schutz vor Brute Force Attacken fungiert. Oracle verzögert bei fehlerhaften Anmeldeversuchen den nächsten Anmeldevorgang. Es gilt:

- Während der ersten drei Anmeldeversuche existiert noch keine Verzögerung
- Ab dem vierten Versuch wird eine stetig ansteigende Verzögerung (max. 10 Sekunden) eingebaut.

Ein einfacher Test mit einem Java-Programm zeigt die Auswirkungen dieses neuen Features.

Listing 25.24: Logon Delay Test

```
[oracle@FEA11-119CL ~]$ /usr/java/jdk1.7.0_45/bin/java AuthDelayTest
Connecting with URL=jdbc:oracle:oci8:@ORCL as bank/wrong_password
Versuch: 0
Delay: 0 Sek.
Versuch: 1
Delay: 0 Sek.
Versuch: 2
Delay: 0 Sek.
Versuch: 3
Delay: 1 Sek.
Versuch: 4
Delay: 2 Sek.
Versuch: 5
Delay: 3 Sek.
Versuch: 6
Delay: 4 Sek.
Versuch: 7
Delay: 5 Sek.
Versuch: 8
Delay: 6 Sek.
Versuch: 9
Delay: 7 Sek.
Versuch: 10
Delay: 0 Sek.
Versuch: 11
Delay: 0 Sek.
Versuch: 12
Delay: 0 Sek.
Versuch: 13
Delay: 0 Sek.
```

Das das Delay ab Versuch Nummer 10 wieder auf 0 zurückgeht, hängt damit zusammen, dass der Account dann gesperrt ist.

max_failed_login_attempts

Mit dem Profilparameter MAX_FAILED_LOGIN_ATTEMPTS ist es möglich, für valide Nutzerkonten eine maximale Anzahl fehlerhafter Anmeldeversuche festzulegen. Ein Konto wird gesperrt, sobald das Passwort zu oft falsch eingegeben wurde. Dieser Mechanismus schützt vor Brute Force Attacken, nicht aber vor dem Ausspähen von Nutzernamen.

Mit dem neuen sec_max_failed_login_attempts-Parameter kann die maximale Anzahl fehlerhafter Anmeldeversuche für einen Client konfiguriert werden. Sendet der Client zu häufig eine falsche Kombination aus Nutzernamen und Passwort, wird seine Netzwerkverbindung zur Datenbank getrennt. Dieser Initialisierungsparameter bezieht sich auch auf Anmeldeversuche mit nicht existenten Nutzerkonten, im Unterschied zum Profilparameter MAX_FAILED_LOGIN_ATTEMPTS, der nur für existente Konten gilt.

Wenn beispielsweise ein Hacker einen Serverprozess startet um mittels Brute Force unterschiedliche Kombinationen aus Nutzernamen und Passwort zu testen, kann mittels dieses neuen Initialisierungsparameters seine Verbindung zum Serverprozess nach n Versuchen getrennt werden. Ohne diesen Parameter könnte der Hacker tausende von Versuchen starten, ohne dabei unterbrochen zu werden.

Listing 25.25: Der Parameter sec_max_failed_login_attempts

```
SQL> ALTER SYSTEM
  2  SET sec_max_failed_login_attempts = 3
  3  SCOPE=spfile;
```

Wird der Parameter wie in Beispiel ?? konfiguriert, bedeutet dies für alle Clients, dass nach drei falschen Anmeldeversuchen die Connection zum Datenbankserver getrennt wird.

Dieser Parameter ist statisch!

25.2.2 Sichere Passwörter

Bis zur Einführung von Oracle 11g waren alle Nutzernammpasswörter, inklusive der Administratorenpasswörter case insensitiv, „HalloWelt“ war gleich „hallowelt“. Durch den neuen Initialisierungsparameter sec_case_sensitive_logon ist es nun möglich, Passwörter case sensitiv in der Datenbank zu speichern. Der Kern dieses neuen Verfahrens ist der SHA1 Hashing Algorithmus. Dieser ist im Gegensatz zum 3DES-Algorithmus in der Lage die Passwörter case sensitiv zum hashen. Zusätzlich dazu bringt der SHA1-Algorithmus ein Salt mit sich, der die Passwort-Hashes noch sicherer macht.

Listing 25.26: sec_case_sensitive_logon

```
SQL> show parameter sec_case_sensitive_logon
```

NAME	TYPE	VALUE
sec_case_sensitive_logon	boolean	TRUE

Eine Stolperfalle, die es zu beachten gilt ist, dass die case-sensitivität von Administratorenpasswörtern nicht nur von sec_case_sensitive_logon abhängig ist, sondern zusätzlich auch davon, wie die Passwortdatei erstellt wurde. Wird die Passwortdatei mit dem Parameter ignorecase=y erstellt, bleibt sec_case_sensitive_logon für die Administratoren wirkungslos. Nur wenn ignorecase=n ist, haben auch Admins case-sensitive Passwörter.



In Oracle 12c ist gilt dieser Parameter als Deprecated (veraltet), da Passwörter in der Version 12c nur noch case sensitiv gespeichert werden.

25.3 Verschlüsselte Tablespaces

Seit Oracle 11g ist es möglich, statt der gesamten Datenbank, nur einzelne Tablespace zu verschlüsseln. Dies hat den Vorteil, dass der Administrator gezielt, sensitive Daten verschlüsseln kann. Der von Oracle genutzte Verschlüsselungsmechanismus wird als Transparent Data Encryption (TDE) bezeichnet. Das Schlüsselwort „Transparent“ bedeutet, dass alle Anwendungen, ohne Veränderung, die verschlüsselten Daten nutzen können, als seien sie unverschlüsselt. Die Daten werden sogar automatisch verschlüsselt im Undo Tablespace und in den Redo Logs abgelegt, so dass auch dort Datendiebstahl nicht ohne weiteres erfolgen kann.

Oracle benutzt zur Verschlüsselung Industriestandards, wie AES256, AES192, AES128 und 3DES168. Welcher Standard genutzt werden soll, muss bei der Erstellung des Tablespace angegeben werden. Unterschiedliche Tablespace können unterschiedliche Verschlüsselungsverfahren nutzen.



Das Standardverfahren innerhalb der Oracle-Datenbank ist AES128.

Grundlage für die Kryptierung ist ein Wallet.

25.3.1 Vorbereiten der Tablespaceverschlüsselung

Wallets

Ein Wallet¹ ist ein verschlüsselter Speicher für Zertifikate und Schlüssel. Es kann für unterschiedliche Zwecke eingesetzt werden, wie z. B. Authentifizierung oder Verschlüsselung. Wird das Wallet zur Verschlüsselung benutzt, spricht man von einem „Encryptionwallet“.

Erstellen eines Walletspeicherortes

Um ein Wallet für die Verschlüsselung eines Tablespaces nutzen zu können, muss vorher ein „Walletspeicher“ definiert werden. Hierbei handelt es sich um ein Verzeichnis, welches der Datenbank bekannt gemacht werden muss. Dazu muss der Walletspeicher in der Datei `sqlnet.ora` eingetragen werden.

Listing 25.27: Ein Encryptionwallet registrieren

```
ENCRYPTION_WALLET_LOCATION =
  (SOURCE =
    (METHOD = FILE)
    (METHOD_DATA =
      (DIRECTORY = /u01/app/oracle/wallets/))
  )
```

Der `sqlnet.ora`-Parameter `ENCRYPTION_WALLET_LOCATION` gibt an, wo das Wallet gespeichert werden soll. `METHOD = FILE` weisst darauf hin, dass das Wallet als Datei auf dem Dateisystem abgelegt wird. `DIRECTORY` gibt den eigentlichen Speicherort an.

Erstellen des Encryptionwallets

Das Encryptionwallet wird direkt mit SQL erzeugt.

Listing 25.28: Ein Encryptionwallet erzeugen

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY
  2 IDENTIFIED BY "P@ssw0rd";
```

¹engl. Brieftasche

25.3.2 Einen verschlüsselten Tablespace erstellen

Öffnen des Wallets

Um ein Encryptionwallet nutzen zu können, muss es zuerst geöffnet werden.

Listing 25.29: Ein Encryptionwallet öffnen

```
SQL> ALTER SYSTEM
  2  SET ENCRYPTION WALLET OPEN
  3  IDENTIFIED BY "P@ssw0rd";
```

Nach dem Öffnen des Wallets kann der Tablespace kreiert werden.

Erstellen des Tablespaces

Die Erstellung des verschlüsselten Tablespaces erfolgt mit Hilfe des SQL-Kommandos `CREATE TABLESPACE`.

Listing 25.30: Einen kryptierten Tablespace mit AES128-Verschlüsselung erstellen

```
SQL> CREATE TABLESPACE crypto_ts
  2  DATAFILE '/u02/oradata/orcl/crypto_ts01.dbf' SIZE 50M
  3  ENCRYPTION
  4  DEFAULT STORAGE(ENCRYPT);

Tablespace created.
```

Die Zeilen 3 und 4 `ENCRYPTION` und `DEFAULT STORAGE(ENCRYPT)` sorgen für die Verschlüsselung. Die `ENCRYPTION`-Klausel kann variiert werden, um einen anderen Verschlüsselungsalgorithmus zu verwenden.

Listing 25.31: Einen kryptierten Tablespace mit alternativer Verschlüsselungsmethode erzeugen

```
SQL> CREATE TABLESPACE crypto_ts_aes256
  2  DATAFILE '/u02/oradata/orcl/crypto_ts_aes25601.dbf' SIZE 50M
  3  ENCRYPTION USING 'AES256'
  4  DEFAULT STORAGE(ENCRYPT);

Tablespace created.
```

So wohl das Erstellen, als auch der Zugriff auf kryptierte Tablespaces funktioniert nur, wenn das Encryptionwallet geöffnet wurde.

Listing 25.32: Einen kryptierten Tablespace mit AES128-Verschlüsselung erstellen

```
SQL> CREATE TABLESPACE crypto_ts_false
  2  DATAFILE '/u02/oradata/orcl/crypto_ts_false01.dbf' SIZE 50M
  3  ENCRYPTION
```

```

4  DEFAULT STORAGE(ENCRYPT);
ORA-28365: wallet is not open

```

Oracle quittiert den Zugriff ohne geöffnetes Wallet mit dem Fehler ORA-28365.

Schließen des Wallets

Sollte das Wallet für keinen weiteren Vorgang gebraucht werden, ist es ratsam, das Wallet wieder zu schließen.

Listing 25.33: Das Encryptionwallet schließen

```

SQL> ALTER SYSTEM
2  SET ENCRYPTION WALLET CLOSE
3  IDENTIFIED BY "P@ssw0rd";

```

25.3.3 Umgang mit verschlüsselten Tablespaces

Informationen sammeln

Informationen über die Verschlüsselung und den benutzten Algorithmus liefert die View V\$ENCRYPTED_TABLESPACES.

```

SQL> SELECT t.ts#, t.name, e.encryptionalg, e.encryptedts
  2  FROM    v$tablespace t INNER JOIN v$encrypted tablespaces e
  3  ON  (t.ts# = e.ts#);

   TS#  NAME          ENCRYPT  ENC
   ---  --  -----
     16 CRYPTO_TS      AES128   YES
     17 CRYPTO_TS_AES256 AES256   YES

```

Einschränkungen

Im Umgang mit kryptierten Tablespaces existieren drei wesentliche Einschränkungen:

- Ein unverschlüsselter Tablespace kann nicht im Nachhinein verschlüsselt werden. Dies kann nur über Umwege, mittels der Oracle Data Pump erfolgen.

- Verschlüsselte Tablespaces können nicht ohne Weiteres in eine andere Datenbank umgezogen werden.
- Beim Wiederherstellen eines kryptierten Tablespace muss vor der Recovery-Phase das Wallet geöffnet werden, da sonst der Oracle Recovery Manager die Datenblöcke nicht verarbeiten kann.



- [ADMIN12327]

25.4 Informationen

25.4.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10019]
- [REFRN10274]
- [REFRN10299]

25.4.2 Verzeichnis der relevanten Data Dictionary Views



- [sthref1850]
- [sthref1854]
- [sthref1856]
- [sthref1858]
- [sthref2293]
- [sthref2337]
- [sthref2472]
- [sthref2579]
- [REFRN30496]
- [REFRN30405]

25.4.3 Verzeichnis der Konfigurationsdateien und ihrer Parameter



- [NETRF006]

25.5 Übungen - Implementing Security

1. Stellen Sie den Initialisierungsparameter audit_trail auf den Wert *xml, extended* ein, um das Datenbankauditing zu aktivieren!
 2. Konfigurieren Sie das Auditing für die Tabelle BANK.MITARBEITER so, dass erfolgreiche **INSERT**- und **UPDATE**-Statements auditiert werden! Es soll jeweils nur ein Auditeintrag pro Session für jedes **INSERT/UPDATE** gemacht werden.
 3. Prüfen Sie, ob die Auditingeinstellungen richtig sind! Welche View muss hierfür benutzt werden?
-

4. Führen Sie das Skript lab_auditing.sql aus! Dieses Skript wird Auditinginformationen für die Tabelle BANK.MITARBEITER erzeugen.

```
SQL> @/home/oracle/labs/lab_auditing.sql
```

5. Prüfen Sie den Audittrail auf neue Informationen! Welche Tätigkeiten wurden an der auditierten Tabelle ausgeführt?

6. Machen Sie alle Auditingeinstellungen rückgängig und löschen Sie den Inhalt des Auditingtrails!
7. Schalten Sie das Auditing für alle erfolgreichen Anmeldeversuche ein! Lassen Sie diese Auditin-geinträge im *DB, Extended* Auditingtrail speichern!
8. Melden Sie sich jeweils mit den Nutzerkonten HR, SH und OE an der Datenbank an! Die beiden Nutzer SH und OE müssen evtl. erst noch freigeschaltet und mit einem Passwort versehen werden. Führen Sie auch absichtlich einige Anmeldeversuche durch, die fehlschlagen!
9. Fragen Sie den Auditingtrail der Datenbank ab und suchen Sie nach ihren Login-Versuchen von gerade eben! Welche View bietet sich für derartige Abfragen an?

10. Schalten Sie das Auditing wieder ab!

11. Der Tablespace BANK liegt derzeit in unverschlüsselter Form vor. Dies muss zukünftig anders sein. Bereiten Sie einen Tablespace BANK_ENCRYPTED vor, der mittels AES256 verschlüsselung gesichert ist und der die gleichen Dimensionen hat, wie der Originaltablespace BANK! Rufen Sie alle notwendigen Informationen über den Tablespace BANK aus dem Data Dictionary ab! Welche Views helfen Ihnen dabei?

25.6 Lösungen - Implementing Security

1. Stellen Sie den Initialisierungsparameter audit_trail auf den Wert *xml, extended* ein, um das Datenbankauditing zu aktivieren!

```
SQL> ALTER SYSTEM
  2  SET audit_trail=xml, extended SCOPE=spfile;
System altered.

SQL> shutdown immediate
SQL> startup
```

2. Konfigurieren Sie das Auditing für die Tabelle BANK.MITARBEITER so, dass erfolgreiche **INSERT**- und **UPDATE**-Statements auditiert werden! Es soll jeweils nur ein Auditeintrag pro Session für jedes **INSERT/UPDATE** gemacht werden.

```
SQL> AUDIT UPDATE, INSERT ON bank.mitarbeiter
  2  BY SESSION
  3  WHENEVER SUCCESSFUL;

Audit succeeded.
```

3. Prüfen Sie, ob die Auditingeinstellungen richtig sind! Welche View muss hierfür benutzt werden?

```
SQL> SELECT object_name, ins, upd
  2  FROM dba_obj_audit_OPTS;

OBJECT_NAME           INS  UPD
-----
MITARBEITER          S/-  S/-
```

4. Führen Sie das Skript lab_auditing.sql aus! Dieses Skript wird Auditinginformationen für die Tabelle BANK.MITARBEITER erzeugen.

```
SQL> @/home/oracle/labs/lab_auditing.sql
```

5. Prüfen Sie den Audittrail auf neue Informationen! Welche Tätigkeiten wurden an der auditierten Tabelle ausgeführt?

```
SQL> col db_user format a10
SQL> col sql_text format a50
SQL> set linesize 100
SQL> SELECT db_user,
  2        TO_CHAR(extended_timestamp, 'DD.MM.YYYY HH24:MI:SS') AS time,
  3        sql_text
```

```
4  FROM    dba_common_audit_trail  
5  WHERE   object_schema LIKE 'BANK'
```

DB_USER	TIME	SQL_TEXT
T_JONES	21.10.2013 17:08:39	UPDATE bank.mitarbeiter SET gehalt = gehalt * 10
D_HARRY	21.10.2013 17:08:40	UPDATE bank.mitarbeiter SET provision = provision * 1.1
T_JONES	21.10.2013 17:09:11	UPDATE bank.mitarbeiter SET gehalt = gehalt * 10
D_HARRY	21.10.2013 17:09:11	UPDATE bank.mitarbeiter SET provision = provision * 1.1
T_JONES	21.10.2013 17:09:58	UPDATE bank.mitarbeiter SET gehalt = gehalt * 10
D_HARRY	21.10.2013 17:09:58	UPDATE bank.mitarbeiter SET provision = provision * 1.1

6. Machen Sie alle Auditingeinstellungen rückgängig und löschen Sie den Inhalt des Auditingtrails!

```
SQL> NOAUDIT ALL ON bank.mitarbeiter;
```

7. Schalten Sie das Auditing für alle erfolgreichen Anmeldeversuche ein! Lassen Sie diese Auditin-geinträge im *DB*, *Extended* Auditingtrail speichern!

```
SQL> ALTER SYSTEM
  2  SET audit_trail=db, extended
  3  SCOPE=spfile;

System altered.

SQL> shutdown immediate
SQL> startup
SQL> AUDIT connect
  2  WHENEVER SUCCESSFUL;

Audit succeeded.
```

8. Melden Sie sich jeweils mit den Nutzerkonten HR, SH und OE an der Datenbank an! Die beiden Nutzer SH und OE müssen evtl. erst noch freigeschaltet und mit einem Passwort versehen werden. Führen Sie auch absichtlich einige Anmeldeversuche durch, die fehlschlagen!

9. Fragen Sie den Auditingtrail der Datenbank ab und suchen Sie nach ihren Login-Versuchen von gerade eben! Welche View bietet sich für derartige Abfragen an?

```
SQL> SELECT username, userhost,
  2          TO_CHAR(extended_timestamp, 'DD.MM.YYYY HH24:MI:SS') AS TIME
  3    FROM dba_audit_session
 4* ORDER BY timestamp;

USERNAME      USERHOST          TIME
-----  -----
HR            FEA11-119SRV.oracle.com 22.10.2013 07:38:25
HR            FEA11-119SRV.oracle.com 22.10.2013 07:38:28
SH            FEA11-119SRV.oracle.com 22.10.2013 07:38:28
SH            FEA11-119SRV.oracle.com 22.10.2013 07:38:32
```

10. Schalten Sie das Auditing wieder ab!

```
SQL> ALTER SYSTEM
  2  SET audit_trail=none
  3  SCOPE=spfile;
```

11. Der Tablespace BANK liegt derzeit in unverschlüsselter Form vor. Dies muss zukünftig anders sein. Bereiten Sie einen Tablespace BANK_ENCRYPTED vor, der mittels AES256 verschlüsselung gesichert ist und der die gleichen Dimensionen hat, wie der Originaltablespace BANK! Rufen Sie alle notwendigen Informationen über den Tablespace BANK aus dem Data Dictionary ab! Welche Views helfen Ihnen dabei?

```
#sqlnet.ora
ENCRYPTION_WALLET_LOCATION =
(SOURCE =
(METHOD = FILE)
(METHOD_DATA =
(DIRECTORY = /u01/app/oracle/product/11.2.0/orcl/network/admin)
)
)
```

```
SQL> ALTER SYSTEM SET ENCRYPTION KEY
  2  IDENTIFIED BY "P@ssw0rd";

SQL> ALTER SYSTEM
  2  SET ENCRYPTION WALLET OPEN
  3  IDENTIFIED BY "P@ssw0rd";
```

```
SQL> col tablespace_name format a10
SQL> col file_name format a45
SQL> col mbytes format 9999
SQL> col autoextensible format a3

SQL> set linesize 100
SQL> SELECT tablespace_name, file_name, bytes / POWER(1024, 2) AS mbytes,
  2       autoextensible
  3   FROM dba_data_files
  4 WHERE tablespace_name LIKE 'BANK';

TABLESPACE      FILE_NAME          MBYTES AUT
-----          -----
BANK           /u01/app/oracle/oradata/orcl/bank01.dbf      100 NO
BANK           /u01/app/oracle/oradata/orcl/bank02.dbf      100 NO

SQL> CREATE TABLESPACE bank_encrypted
  2   DATAFILE '/u01/app/oracle/oradata/orcl/bank_encrypted01.dbf' SIZE 100M,
        '/u01/app/oracle/oradata/orcl/bank_encrypted02.dbf' SIZE 100M
  3   ENCRYPTION USING 'AES256'
  4   DEFAULT STORAGE(ENCRYPT);
```


26 Verschieben von Daten - Extract Transform and Load (ETL)

Inhaltsangabe

26.1 Oracle Data Pump

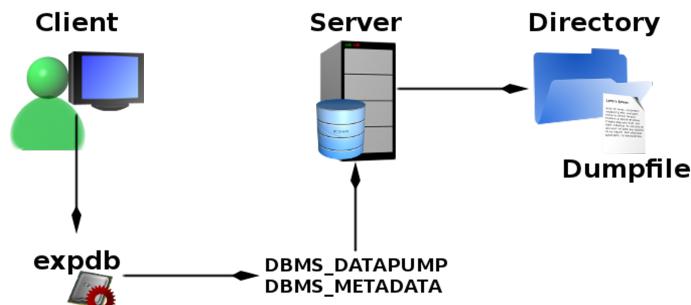
Oracle Data Pump ist eine serverseitige Technologie, die es ermöglicht, Daten von einer Oracle-Datenbank, in eine andere Oracle-Datenbank, zu verschieben. Auf ihr basieren die beiden Tools „Data Pump Export“ (expdb) und „Data Pump Import“ (impdp). Mit dem Export-Utility werden die Daten in eine Binärdatei, mit proprietärem Format geschrieben, die als „Dumpfile“ bezeichnet wird. Diese Datei kann dann zu einer anderen Datenbank transportiert und dort importiert werden.

Eine andere Möglichkeit des Datenaustausches besteht darin, Quell- und Zieldatenbank direkt, mittels Import-Utility, über das Netzwerk zu verbinden. Diese Methode wird als „Direct Import“ bezeichnet. Hierbei wird kein Dumpfile erzeugt.

Die Data Pump besteht aus drei Teilen:

- Den Kommandozeilen-Programmen expdp und impdp
- Dem PL/SQL-Paket DBMS_DATAPUMP (Auch Data Pump API genannt)
- Dem PL/SQL-Paket DBMS_METADATA (Auch Metadata API genannt)

Abb. 26.1:
Ein Export mit
expdb



26.1.1 Data Pump konfigurieren

Da Data Pump ein serverbasiertes Tools ist, werden Dumpfilesets und Logdateien auf dem Server geschrieben. Um der Datenbank den Speicherort der Dumpfile Sets und Logdateien bekannt zu geben, muss in der Datenbank ein Directory-Objekt erstellt werden.



Ein Directory-Objekt verbindet einen Namen in der Datenbank, mit einem Verzeichnis auf dem Datenträger. Dadurch wird es möglich, den Nutzerzugriff auf dem Datenbankserver zu begrenzen.

Directory-Objekte in SQL erstellen

Das folgende Beispiel erstellt ein Directory-Objekt namens `dpump_dir` für das Verzeichnis `/u02/dpdump`.

Listing 26.1: Beispiel für `CREATE DIRECTORY`

```
SQL> CREATE DIRECTORY dpump_dir
  2 AS '/u02/dpdump';
```

Um einem Nutzer Lese- oder Schreibrechte auf ein Directory zu geben, werden die beiden Privilegien `read` und `write` verwendet.

Listing 26.2: Zugriff auf ein Directory gewähren

```
SQL> GRANT read, write ON DIRECTORY dpump_dir
  2 TO bank;
```

Diese beiden Privilegien haben nur innerhalb der Datenbank Auswirkungen. Der DBA muss sicherstellen, dass die Datenbank auf dem Dateisystem Lese- und Schreibrechte hat.

- [sthref4351]



26.1.2 Mit Data Pump einen Export durchführen

Die wichtigste Eigenschaft eines Data Pump-Exports ist, dass festgelegt werden kann, welche Teile der Datenbank exportiert werden sollen. Dies geschieht auf der Kommandozeile durch Angabe eines Parameters. Es gibt folgende Exportarten:

- **Full Export Mode:** Es wird die gesamte Datenbank exportiert. Hierfür benötigt der Nutzer die Rolle `datapump_exp_full_database`.

Listing 26.3: Full Database Export

```
[oracle@FEA11-119SRV admin]$ expdp system/oracle DIRECTORY=dpump_dir \
> DUMPFILE=expdat.dmp FULL=y
```

Der Benutzer `SYSTEM` ist im Besitz der Rolle `datapump_exp_full_database`!



Für das Kommando aus Beispiel ?? gibt es noch eine Kurzschreibweise. Die beiden Parameter `DUMPFILE` und `DIRECTORY` können im Parameter `DUMPFILE` zusammengefasst werden zu: `DUMPFILE=dpump_dir:expdat.dmp`.

- **Schema Mode** (Standardmodus): Befindet sich ein Benutzer im Besitz der Rolle `datapump_exp_full_database` kann er eine Liste zu exportierender Schemata angeben. Optional hat er die Möglichkeit, auch andere Schemadefinitionen (Nutzerkonten, Grants von Rollen und Systemprivilegien, Standardrollen und Tablespace-Quotas) mit zu exportieren.

Besitzt der Nutzer die Rolle `datapump_exp_full_database` nicht, kann er nur sein eigenes Schema exportieren.

Listing 26.4: Schema Export

```
[oracle@FEA11-119SRV admin]$ expdp system/oracle DIRECTORY=dpump_dir \
> DUMPFILE=expdat.dmp SCHEMAS=bank
```

- **Table Mode:** In diesem Modus werden nur die Nutzdaten der angegebenen Tabellen, aber keine Spaltendefinitionen exportiert. Die Tabelle muss in der Zieldatenbank bereits bestehen und muß aus dem gleichen Schema stammen.

Hat ein Nutzer die Rolle `datapump_exp_full_database` nicht, kann er nur Tabellen aus seinem eigenen Schema exportieren.

Listing 26.5: Table Export

```
[oracle@FEA11-119SRV admin]$ expdp system/oracle DIRECTORY=dpump_dir \
> DUMPFILE=expdat.dmp TABLES=bank.mitarbeiter, bank.bankfiliale
```

- **Tablespace Mode:** Im Tablespace Modus werden alle Objekte der angegebenen Tablespace (Nutz- und Metadaten, keine Grants von Objektprivilegien), plus alle von den Tabellen abhängigen Objekte exportiert.

Listing 26.6: Tablespace Export

```
[oracle@FEA11-119SRV admin]$ expdp system/oracle DIRECTORY=dpump_dir \
> DUMPFILE=expdat.dmp TABLESPACES=bank
```

- **Transportable Tablespace Mode:** In diesem Modus werden nur die Metadaten der Tabellen und der von ihnen abhängigen Objekte (inklusive Grants von Objektprivilegien) exportiert. Dies ermöglicht es den Tablespace in eine andere Datenbank aufzunehmen, ohne die Grants von Objektprivilegien neu machen zu müssen. Die zu diesem Tablespace gehörenden Datendateien müssen manuell der importierenden Datenbank zur Verfügung gestellt werden.

Damit ein Tablespace in diesem Modus exportiert werden kann, muss er im Read Only Modus sein und alle Objekte in diesem Tablespace müssen „self-contained“ (autark) sein. D. h. alle Partitionen einer partitionierten Tabelle müssen im Set der angegebenen Tablespace sein.

- Der Tablespace sollte daraufhin überprüft werden, ob er als transportable tablespace exportiert werden kann. Dies kann mittels der PL/SQL-Prozedur TRANSPORT_SET_CHECK aus dem PL/SQL-Paket DBMS_TTS geschehen.

Listing 26.7: Ausführen von DBMS_TTS

```
SQL> EXEC DBMS_TTS.TRANSPORT_SET_CHECK('BANK', TRUE);
```

Auf diese Weise kann festgestellt werden, ob der Tablespace, samt seiner Constraints als self-contained (in sich geschlossen) bezeichnet werden kann.

- Die Ergebnisse der Prozedur TRANSPORT_SET_CHECK können mit Hilfe der View TRANSPORT_SET_VIOLATIONS abgefragt werden.

Listing 26.8: Die Ergebnisse von DBMS_TTS.transport_set_check

```
SQL> SELECT *
  2  FROM   transport_setViolations;
```

Nur wenn diese View keine Ergebnisse zeigt, kann der Tablespace exportiert werden. Andernfalls müssen alle angezeigten Probleme erst behoben werden.

- Der betreffende Tablespace muss „read only“ geschaltet werden.

Listing 26.9: Tablespace Read Only schalten

```
SQL> ALTER TABLESPACE bank READ ONLY;
```

- Durchführen des Exports.

Listing 26.10: Transportable Tablespace Export

```
[oracle@FEA11-119SRV admin]$ expdp system/oracle DIRECTORY=dpump_dir \
> DUMPFILE=expdat.dmp TRANSPORT_TABLESPACES=bank
```

- Der Tablespace kann nach dem Export wieder „read write“ geschaltet werden.

Listing 26.11: Tablespace wieder Read Write schalten.label

```
SQL> ALTER TABLESPACE bank READ WRITE;
```

- Kopieren der Datendateien auf das Zielsystem

Listing 26.12: Prüfen, welche Datendateien kopiert werden müssen

```
SQL> SELECT file_name
  2  FROM   dba_data_files
  3  WHERE   tablespace_name LIKE 'BANK';

FILE_NAME
```

```
-----  
/u01/app/oracle/oradata/orcl/bank01.dbf  
  
SQL> host cp /u01/app/oracle/oradata/orcl/bank01.dbf ...
```



- [e22490toc]

26.1.3 Mit Data Pump einen Import durchführen

Das Importieren von Daten in eine Datenbank funktioniert analog zum Exportieren. Es existieren die gleichen Modi und es gelten auch die gleichen Einschränkungen. Der einzige Unterschied ist, dass für den Datenimport die Rolle `datapump_imp_full_database` existiert, die analog zur Rolle `datapump_exp_full_database` verwendet werden kann.

Dumpfile basierte Imports

Listing 26.13: Full Import

```
[oracle@FEA11-119SRV admin]$ impdp system/oracle DIRECTORY=dpump_dir \  
> DUMPFILE=expdat.dmp FULL=y
```

Listing 26.14: Schema Import

```
[oracle@FEA11-119SRV admin]$ impdp system/oracle DIRECTORY=dpump_dir \  
> DUMPFILE=expdat.dmp SCHEMAS=bank
```

Listing 26.15: Table Import

```
[oracle@FEA11-119SRV admin]$ impdp system/oracle DIRECTORY=dpump_dir \  
> DUMPFILE=expdat.dmp TABLES=bank.mitarbeiter, bank.bankfiliale
```

Listing 26.16: Tablespace Import

```
[oracle@FEA11-119SRV admin]$ impdp system/oracle DIRECTORY=dpump_dir \  
> DUMPFILE=expdat.dmp TABLESPACES=bank
```

Listing 26.17: Transportable Tablespace Import

```
[oracle@FEA11-119SRV admin]$ impdp system/oracle DIRECTORY=dpump_dir \  
> DUMPFILE=expdat.dmp TRANSPORT_DATAFILES='/u02/bank01.dbf'
```

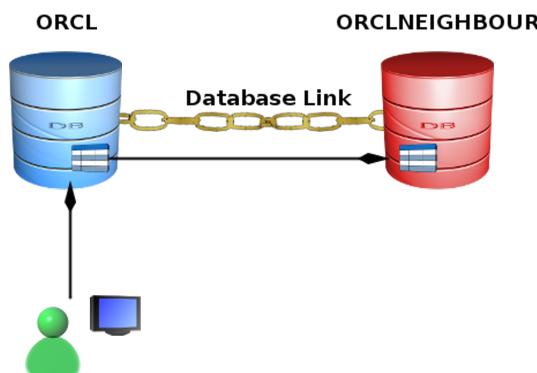
Import über das Netzwerk

Das Tool impdp ist in der Lage, einen Datenimport aus einer anderen Datenbank, unter Umgehung von Dumpfiles, zu machen. Es wird dabei ein Database Link benutzt, der eine benannte Verknüpfung zwischen zwei Datenbanken darstellt. Für die Verknüpfung wird die Oracle TNS-Technologie verwendet.

Folgendes Beispiel zeigt einen Netzwerkimport mit impdp:

Es existieren zwei Datenbanken: ORCL und ORCLNEIGHBOUR. Es wird ein Datenimport von ORCL nach ORCLNEIGHBOUR erfolgen. Der Connect Identifier für die Datenbank ORCLNEIGHBOUR wird in der Datei tnsnames.ora, der Datenbank ORCL unter dem Namen ORCLNEIGHBOUR einge tragen.

Abb. 26.2:
Import mit impdp
übers Netzwerk



Listing 26.18: Der Connect Identifier für ORCLNEIGHBOUR

```
ORCLNEIGHBOUR =
  (DESCRIPTION =
    (ADDRESS =
      (PROTOCOL = TCP)
      (HOST = FEA11-119SRV.oracle.com)
      (PORT = 1521)
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = ORCLNEIGHBOUR)
    )
  )
```

Als nächstes wird ein Database Link von ORCL nach ORCLNEIGHBOUR eingerichtet.

Listing 26.19: Einen Database Link erstellen

```
SQL> CREATE PUBLIC DATABASE LINK orcl_to_neighbour
  2  USING 'ORCLNEIGHBOUR';
```

Nach dem Erstellen des Database Link kann der Importvorgang beginnen.

Listing 26.20: Den Netzwerkimport starten

```
[oracle@FEA11-119SRV admin]$ impdp system/oracle SCHEMAS=bank \
> DIRECTORY=dpump_dir NETWORK_LINK='ORCL_TO_NEIGHBOUR',
```

Zu beachten ist, dass der Parameter DUMPFILE entfällt und statt dessen der Parameter NETWORK_LINK verwendet wird.



Der Name des Database Links sollte in Hochkommas ' ' gesetzt werden.

26.2 Der SQL*Loader - Import von CSV-Dateien

Der SQL*Loader ermöglicht es, Daten aus externen Quellen in die Datenbank zu laden. Er besitzt eine mächtige Parser-Engine, die dem Eingabeformat der Daten kaum Grenzen setzt.

Eine typische SQL*Loader-Session bezieht ihre Parameter aus einer Kontrolldatei. Diese Kontrolldateien sind nicht mit den Kontrolldateien der Datenbank zu verwechseln. Für jeden SQL*Loader-Job kann eine eigene Kontrolldatei, mit SQL*Loader-Parametern erstellt werden.

Die zu importierenden Daten werden direkt in die Datenbank geschrieben. Während eines Importvorgangs kann der SQL*Loader die folgenden Dateien erstellen:

- Meldungen zum Importvorgang: Log-Datei
- Zeilen die aufgrund eines Fehlers nicht eingefügt werden können (z. B. falscher Datentyp in der Zieltabelle): Bad-File
- Zeilen die einer Filterbedingung nicht entsprechen: Discard-File

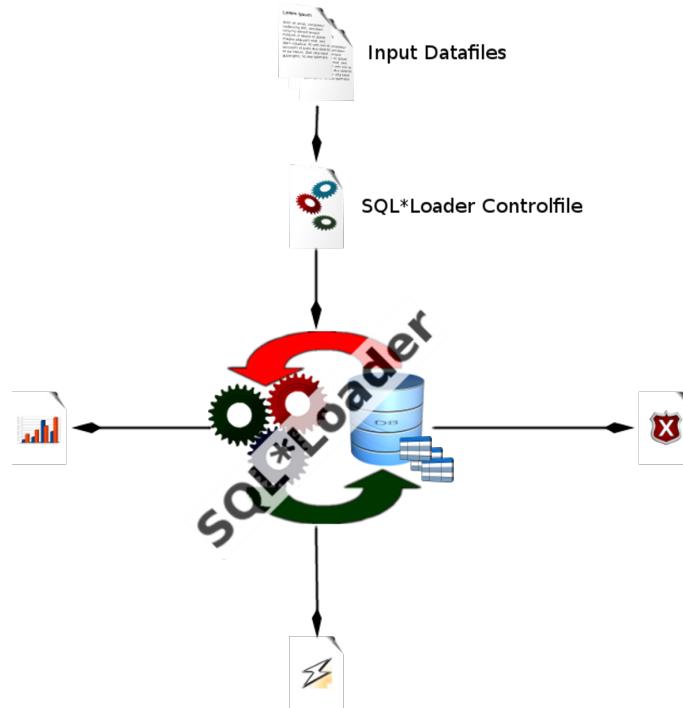


Abb. 26.3:
Übersicht über
den SQL*Loader

26.2.1 SQL*Loader aufrufen

Der SQL*Loader wird auf der Kommandozeile mit dem Kommando `sqlldr` aufgerufen. Hinzukommen Parameter, die den Verlauf der SQL*Loader-Session beeinflussen.

Listing 26.21: Den SQL*Loader benutzen

```
[oracle@FEA11-119SRV admin]$ sqlldr CONTROL=bank.ldr, LOG=bank.log, \
> BAD=bank.bad, DATA=buchungen.dat USERID=bank/bank, \
> DISCARD=bank.dsc
```

Werden immer wieder die gleichen Parameter benötigt, können diese in einer Kontrolldatei zusammengefasst werden. Eine vollständige Auflistung aller SQL*Loader Parameter finden Sie im folgenden Literaturhinweis.



- [SUTIL004]

26.2.2 SQL*Loader Kontrolldateien

Die Kontrolldatei ist eine Textdatei, die SQL*Loader Parameter enthält. Sie teilt dem Loader mit, wo die Eingangsdaten zu finden sind, wie sie zu parsen sind, wo sie eingefügt werden sollen und vieles mehr.

Eine Kontrolldatei besteht aus drei Sektionen:

- In der ersten Sektion werden Session-Informationen angegeben, beispielsweise die `infile`-Klausel, die festlegt, wo sich die Eingabedatei befindet und wie sie heißt.
- Sektion Nummer zwei besteht aus einem oder mehreren „`Into Table`“ -Blöcken. Jeder Block beschreibt eine Zieltabelle, in die die Eingabedaten geladen werden sollen.
- Die dritte Sektion ist optional und kann Informationen zu den Eingabedaten enthalten.

Einige Hinweise zur Syntax der Kontrolldatei:

- Die Syntax ist frei formatierbar (Statements können über mehrere Zeilen hinweg geschrieben werden)
- Sie ist nicht Case-Sensitiv

- Kommentare beginnen mit zwei Bindestrichen -
- Die beiden Begriffe CONSTANT und ZONE sind reservierte Schlüsselwörter.

• [SUTIL004]



26.2.3 Daten Laden

Die Beispieldateien

Für die folgenden SQL-Loader-Beispiele werden die beiden Dateien `buchungen_fix.csv` und `buchungen_var.csv` als Datenquellen herangezogen.

Listing 26.22: Die Datei `buchungen_fix.csv`

```
500001 -2875,92 30.04.2013 1 600000
500002 2875,92 30.04.2013 5 600000
500003 4687,36 08.05.2013 18 600001
500004 -4687,36 08.05.2013 4 600001
500005 46,45 25.05.2013 71 600002
500006 -46,45 25.05.2013 2 600002
500007 752,20 28.05.2013 83 600003
500008 -752,20 28.05.2013 26 600004
500009 -196,20 07.06.2013 10 600005
500010 196,20 07.06.2013 11 600005
```

Listing 26.23: Die Datei `buchungen_var.csv`

```
500011;-2394,32;08.06.2013;21;600011
500012;2394,32;08.06.2013;63;600011
500013;1234,56;17.06.2013;45;600012
500014;-1234,56;17.06.2013;16;600012
500015;66,45;23.06.2013;29;600013
500016;-66,45;23.06.2013;49;600013
500017;852,20;02.07.2013;41;600014
500018;-852,20;02.07.2013;3;600014
500019;-296,20;07.07.2013;8;600015
500020;296,20;07.07.2013;9;600015
500021;832,17;16.07.2013;13;600016
500022;-832,17;16.07.2013;43;600016
500023;-171,00;29.07.2013;27;600017
500024;171,00;29.07.2013;61;600017
500025;45,30;01.08.2013;52;600018
500026;45,30;01.08.2013;11;600018
```

Datensätze fester Länge

Datensätze fester Länge haben die Eigenschaft, dass alle Spalten in der Quelldatei eine genau definierte Länge haben. Durch die Angabe von POSITION(X:Y) in der SQL*-Loader Kontrolldatei, werden die Positionen der einzelnen Spalten in der Quelldatei festgelegt. POSITION(1:6) besagt, dass die Spalte BUCHUNGS_ID von Zeichen eins bis Zeichen sechs reicht. Gleiches gilt für die restlichen Spalten.

Listing 26.24: Datensätze fester Länge- Die Kontrolldatei buchungen_fix.ldr

```

LOAD DATA
INFILE  'labs/buchungen_fix.csv',
BADFILE 'buchungen_fix.bad',
APPEND INTO TABLE Buchung (
  buchungs_id      POSITION(1:6)      INTEGER EXTERNAL ,
  betrag           POSITION(8:15)     DECIMAL EXTERNAL ,
  buchungsdatum    POSITION(17:26)    DATE "DD.MM.YYYY",
  konto_id         POSITION(28:29)    INTEGER EXTERNAL ,
  transaktions_id  POSITION(31:36)    INTEGER EXTERNAL
)

```

Vor dem Aufruf des SQL*Loader muss zwingend die Umgebungsvariable NLS_LANG gesetzt werden, da der Loader sonst nicht mit dem Komma, als Dezimaltrennzeichen zurecht kommt.

Listing 26.25: Den SQL*Loader benutzen

```
[oracle@FEA11-119SRV admin]$ export NLS_LANG=german_germany.UTF8
```

Die Angabe german_germany.UTF8 bewirkt drei Dinge:

- Der SQL*Loader erstellt seine Log-Datei in deutsch (german).
- Die Ländereinstellungen (Dezimaltrennzeichen, Währungssymbol, usw.) werden auf deutsch eingesetzt (germany)
- Als Zeichensatz wird UTF8 genutzt (.UTF8).

Der Aufruf für den SQL*Loader sieht wie folgt aus:

Listing 26.26: Den SQL*Loader benutzen

```
[oracle@FEA11-119SRV admin]$ sqlldr USERID=bank/bank \
> CONTROL=buchungen_fix.ldr, LOG=buchungen_fix.log
```

Datensätze variabler Länge

Bei Datensätzen variabler Länge existiert keine fixe Spaltenlänge. Vielmehr werden die einzelnen Spalten der Quelldatei durch ein Trennzeichen von einander getrennt. In der Datei buchungen_var.csv ist dies das Semikolon (;). Die Kontrolldatei für den SQL*-Loader ändert sich dahingehend, dass das Trennzeichen festgelegt werden muss.

Listing 26.27: Datensätze variabler Länge - Die Kontrolldatei buchungen_var.ldr

```

LOAD DATA
INFILE  'labs/buchungen_var.csv'
BADFILE 'buchungen_var.bad'
APPEND INTO TABLE Buchung
FIELDS TERMINATED BY ";" OPTIONALLY ENCLOSED BY '"' (
    buchungs_id,
    betrag,
    buchungsdatum,
    konto_id,
    transaktions_id
)

```

Die Klausel FIELDS TERMINATED BY legt das Spaltentrennzeichen fest. Mit der zusätzlichen Angabe OPTIONALLY ENCLOSED BY kann ein Zeichen bestimmt werden, das als „Zeichenkettenbegrenzer“ funktioniert. Zeichenketten, wie z. B. „Hallo Welt“ können Leer- oder Sonderzeichen enthalten. Deshalb müssen sie in Begrenzungszeichen, z. B. das „ eingeschlossen werden. Nur so kann der Loader Start und Ende der Zeichenkette zuverlässig erkennen. Das Begrenzungszeichen wird beim Import nicht mitgenommen. Nur die Zeichen dazwischen werden importiert.

Der Aufruf für den SQL*Loader sieht wie folgt aus:

Listing 26.28: Den SQL*Loader benutzen

```
[oracle@FEA11-119SRV admin]$ sqlldr USERID=bank/bank \
> CONTROL=buchungen_var.ldr, LOG=buchungen_var.log
```

Spaltenüberschriften beim Laden auslassen

Mit Hilfe der skip n-Klausel können beliebig viele Zeilen am Dateianfang ausgelassen werden. n steht dabei für eine ganze Zahl. Dies ist beispielsweise dann nützlich, wenn die erste Zeile der Quelldatei Spaltenüberschriften enthält.

Listing 26.29: Datensätze auslassen

```

OPTIONS (SKIP=1)
LOAD DATA
INFILE  'labs/buchungen_var.csv'
BADFILE 'buchungen_var.bad'
APPEND INTO TABLE Buchung
FIELDS TERMINATED BY ";" OPTIONALLY ENCLOSED BY '"' (
    buchungs_id,
    betrag,
    buchungsdatum,
    konto_id,
    transaktions_id
)

```

)

26.3 Externe Tabellen

Externe Tabellen stellen eine Erweiterung des SQL*Loaders bzw. der Oracle Datapump dar. Sie ermöglichen es, externe Datenstrukturen so zu nutzen, als wären sie in der Datenbank gespeichert. Prinzipiell können externe Tabellen immer dann genutzt werden, wenn Daten in die Datenbank importiert werden sollen. Meist werden die externen Tabellen für Staging-Zwecke verwendet.



Staging bedeutet, die zu importierenden Daten auf Gültigkeit zu prüfen und in die für den Import nötige Form zu transformieren.

26.3.1 Spaltendefinitionen und Treiberwahl

Die Definition externer Tabellen besteht aus zwei Teilen:

- Spaltendefinitionen
- Treibereinstellungen

Erstellen der Spaltendefinitionen

Die Syntax für die Spaltendefinitionen ist die Gleiche, wie bei „normalen“ Tabellen.

Listing 26.30: Die Spaltendefinition einer externen Tabelle

```
CREATE TABLE kunden_ext (
    Vorname          VARCHAR2(30),
    Nachname         VARCHAR2(30),
    Geburtsdatum     DATE,
    Adresse          VARCHAR2(200),
    PersAuswNr       VARCHAR2(9),
    AusstDatum       DATE,
    Ablaufdatum      DATE,
    Staatsangeh      VARCHAR2(2),
    Geburtsort        VARCHAR2(50),
    Telefon          VARCHAR2(15),
    Email            VARCHAR2(35),
```

```

    IBAN           VARCHAR2(22),
    KtoEroeffDat  DATE,
    Freistellung   NUMBER(12, 2)
)
...

```

Wie wird die Tabelle zur externen Tabelle?

Um eine externe Tabelle erstellen zu können, muss die Klausel `ORGANIZATION EXTERNAL` an die Spaltendefinitionen angehängt werden.

Listing 26.31: Die Tabelle zur externen Tabelle machen

```

CREATE TABLE kunden_ext (
    Vorname           VARCHAR2(30),
    Nachname          VARCHAR2(30),
    Geburtsdatum      DATE,
    Adresse            VARCHAR2(200),
    PersAuswNr        VARCHAR2(9),
    AusstDatum        DATE,
    Ablaufdatum       DATE,
    Staatsangeh        VARCHAR2(2),
    Geburtsort         VARCHAR2(50),
    Telefon            VARCHAR2(15),
    Email              VARCHAR2(35),
    IBAN               VARCHAR2(22),
    KtoEroeffDat      DATE,
    Freistellung       NUMBER(12, 2)
)
ORGANIZATION EXTERNAL
(
...
    LOCATION('bankkunden.csv')
);
...

```

Mit der `LOCATION`-Klausel wird der Name der Quelldatei angegeben.

Die Treibereinstellungen

Die Daten für eine externe Tabelle können aus zwei verschieben Quellen stammen:

- Textdateien (CSV)

- Datapump Dumpfiles

Das Laden von Daten aus CSV-Textdateien erfolgt mit Hilfe des SQL*Loaders. Datapump Dumpfiles werden unter zu Hilfenahme der Oracle Datapump „angezapft“. Welcher dieser beiden Treiber genutzt werden soll, wird in der **TYPE**-Klausel angegeben.

Listing 26.32: Den Treiber auswählen

```
CREATE TABLE kunden_ext (
...
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
...
  LOCATION('bankkunden.csv')
);
```

In Beispiel ?? wird durch die Angabe von `TYPE ORACLE_LOADER` der SQL*Loader-Treiber ausgewählt. Mit `TYPE ORACLE_DATAPUMP` kann stattdessen die Datapump gewählt werden. Abhängig vom gewählten Treiber können Treibereinstellungen vorgenommen werden.



- [b28319et_concepts]

26.3.2 Der ORACLE_LOADER Treiber

Wo kommen die Daten her?

Die Quelldatei einer externen Tabelle muss sich in einem Verzeichnis auf dem Dateisystem befinden, das im Zugriff der Datenbank steht. Das bedeutet, dass ein Directory-Objekt für das betroffene Verzeichnis existieren muss.



In Oracle existiert von Haus aus das Directory-Objekt `DATA_PUMP_DIR` das für Ladevorgänge geschaffen wurde.

Mit der Klausel `DEFAULT DIRECTORY` wird der externen Tabelle das Directory-Objekt, in dem sich die Quelldatei befindet mitgeteilt.

Listing 26.33: Ein Directory-Objekt für die Datenquelle wählen

```
CREATE TABLE kunden_ext (
...
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY data_pump_dir
```

```
...
  LOCATION('bankkunden.csv')
);
```

Interpretation der Quelldaten – Die Access parameter

Wie bereits in Beispiel ?? gezeigt benötigt der SQL*Loader verschiedene Angaben, um die Quelldatei richtig interpretieren zu können. Zum Beispiel:

- **FIELDS TERMINATED BY ',';**: Legt das Trennzeichen zwischen den einzelnen Spalten fest
- **OPTIONALLY ENCLOSED BY ''''**: Gibt an, wie Zeichenfolgen umschlossen werden.
- **RECORDS DELIMITED BY NEWLINE**: Sagt aus, wie die einzelnen Zeilen in der Quelldatei getrennt werden. **NEWLINE** ist das Schlüsselwort für einen Zeilenumbruch.
- **MISSING FIELD VALUES ARE NULL**: Werte die in der Quelldatei nicht vorhanden sind, sollen in der Datenbank als NULL-Werte gelten.
- **SKIP n**: n Zeilen zu Beginn der Quelldatei auslassen.

Listing 26.34: Festlegen der Access parameter

```
CREATE TABLE kunden_ext (
  ...
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY data_pump_dir
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY NEWLINE
    SKIP 1
    FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY ''
    MISSING FIELD VALUES ARE NULL
  )
  LOCATION('bankkunden.csv')
);
```

Interpretation der Quelldaten - Die Spalten der Quelldatei

Damit Oracle die Daten der Quelldatei korrekt interpretieren kann, muss im `CREATE TABLE`-Statement der externen Tabelle, zusätzlich zur Definition der Tabellenspalten, auch eine Definition der „Quellspalten“ erfolgen. Besonders wichtig ist dies, sobald Datums- oder numerische Werte importiert werden sollen.

Datumsspalten werden in der Quellspaltendefinition immer mit dem Datentyp `CHAR` und der Formatdefinition `date_format DATE mask "DD.MM.RR"`. Dies entspricht der Verwendung der SQL-Funktion `TO_CHAR` mit Datumswerten. Das angegebene Datumsformat „DD.MM.RR“ muss das Datumsformat der Quelldaten wiederspiegeln.

Bei numerischen Werten empfiehlt sich die Nutzung des Datentyps `DECIMAL`, ohne Größenbeschränkung, da mit ihm die besten Ergebnisse erzielt werden können.

Listing 26.35: Festlegen der Access parameter

```

CREATE TABLE kunden_ext (
  ...
)
ORGANIZATION EXTERNAL
(
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY data_pump_dir
  ACCESS PARAMETERS
  (
    RECORDS DELIMITED BY NEWLINE
    SKIP 1
    FIELDS TERMINATED BY ';'
    OPTIONALLY ENCLOSED BY ""
    MISSING FIELD VALUES ARE NULL (
      Vorname          CHAR(30),
      Nachname         CHAR(30),
      Geburtsdatum     CHAR date_format DATE mask "DD.MM.RR",
      Adresse           CHAR(200),
      PersAuswNr       CHAR(9),
      AusstDatum       CHAR date_format DATE mask "DD.MM.RR",
      Ablaufdatum      CHAR date_format DATE mask "DD.MM.RR",
      Staatsangeh      CHAR(2),
      Geburtsort        CHAR(50),
      Telefon          CHAR(15),
      Email            CHAR(35),
      IBAN             CHAR(22),
      KtoEroeffDat     CHAR date_format DATE mask "DD.MM.RR",
      Freistellung     DECIMAL
    )
  )
)
LOCATION('bankkunden.csv')

```

```
) ;
```

LogFile und Badfile

Bei der Nutzung von externen Tabellen können ebenfalls Logfiles und Badfiles angelegt werden, so wie dies im Beispiel ?? gezeigt wurde. Beispiel ?? zeigt das komplette Statement zur Erstellung der externen Tabelle KUNDEN_EXT.

Listing 26.36: Logfile und Badfile

```

SQL> CREATE TABLE kunden_ext (
  2  Vorname          VARCHAR2(30),
  3  Nachname         VARCHAR2(30),
  4  Geburtsdatum     DATE,
  5  Adresse           VARCHAR2(200),
  6  PersAuswNr       VARCHAR2(9),
  7  AusstDatum       DATE,
  8  Ablaufdatum      DATE,
  9  Staatsangeh      VARCHAR2(2),
 10 Geburtsort        VARCHAR2(50),
 11 Telefon          VARCHAR2(15),
 12 Email            VARCHAR2(35),
 13 IBAN             VARCHAR2(22),
 14 KtoEroeffDat     DATE,
 15 Freistellung     NUMBER(12, 2)
 16 )
 17 ORGANIZATION EXTERNAL
 18 (
 19   TYPE ORACLE_LOADER
 20   DEFAULT DIRECTORY data_pump_dir
 21   ACCESS PARAMETERS
 22   (
 23     RECORDS DELIMITED BY NEWLINE
 24     SKIP 1
 25     BADFILE 'bankkunden.bad'
 26     LOGFILE 'bankkunden.log'
 27     FIELDS TERMINATED BY ','
 28     OPTIONALY ENCLOSED BY ''
 29     MISSING FIELD VALUES ARE NULL (
 30       Vorname          CHAR(30),
 31       Nachname         CHAR(30),
 32       Geburtsdatum     CHAR date_format DATE mask "DD.MM.RR",
 33       Adresse           CHAR(200),
 34       PersAuswNr       CHAR(9),
 35       AusstDatum       CHAR date_format DATE mask "DD.MM.RR",
 36       Ablaufdatum      CHAR date_format DATE mask "DD.MM.RR",
 37       Staatsangeh      CHAR(2),
 38       Geburtsort        CHAR(50),
 39       Telefon          CHAR(15),
 40       Email            CHAR(35),
 41       IBAN             CHAR(22),
 42       KtoEroeffDat     CHAR date_format DATE mask "DD.MM.RR",
 43       Freistellung     DECIMAL
 44     )
 45   )
 46   LOCATION('bankkunden.csv')
 47 );

```



- [b28319et_params]

Laden von Binärdaten

Mit Hilfe des ORACLE_LOADER Treibers ist es nicht nur möglich ASCII-Daten, sondern auch Binärdaten zu laden. In Beispiel ?? ff. soll für jeden Mitarbeiter der Bank ein Foto in die Datenbank geladen werden. Dazu wird der Tabelle MITARBEITER ein Spalte FOTO, vom Datentyp BLOB hinzugefügt. Im Anschluss daran werden dann die Daten geladen.



Das Akronym „BLOB“ steht für „Binary Large Object“. Ein BLOB kann ein beliebiges Binärobject sein (PDF-Datei, PNG- oder JPG-Dateien, Worddokumente, usw.)

Listing 26.37: Anfügen einer BLOB-Spalte

```
SQL> ALTER TABLE mitarbeiter
  2 ADD Foto BLOB;
```

Listing 26.38: Die Datei mitarbeiter.csv

```
Mitarbeiter_ID;JPG_filename
1;foto_1.jpg
2;foto_2.jpg
```

Listing 26.39: Erstellen der externen Tabelle MITARBEITER_EXT

```
SQL> CREATE TABLE mitarbeiter_ext (
  2 Mitarbeiter_ID      NUMBER ,
  3 Foto                  BLOB
  4 )
  5 ORGANIZATION EXTERNAL
  6 (
  7   TYPE ORACLE_LOADER
  8   DEFAULT DIRECTORY data_pump_dir
  9   ACCESS PARAMETERS
 10  (
 11    RECORDS DELIMITED BY NEWLINE
 12    SKIP 1
 13    BADFILE 'mitarbeiter.bad'
 14    LOGFILE 'mitarbeiter.log'
 15    FIELDS TERMINATED BY ';'
 16    MISSING FIELD VALUES ARE NULL
 17  (
 18    Mitarbeiter_ID      CHAR(1),
 19    JPG_filename        CHAR(10)
 20  )
```

Listing 26.40: Erstellen der externen Tabelle MITARBEITER_EXT - Fortsetzung

```

21   COLUMN TRANSFORMS
22   (
23     Foto FROM LOBFILE (JPG_filename) FROM (data_pump_dir) BLOB
24   )
25   )
26   LOCATION('mitarbeiter.csv')
27 );

```

Neu an dieser externen Tabelle ist die `COLUMN TRANSFORMS`-Klausel. Sie hat die Aufgabe, die Dateinamen aus der CSV-Datei, Spalte `JPG_FILENAME`, auszulesen, die Bilddateien zu öffnen und deren Inhalt in die Spalte `Foto` der externen Tabelle zu laden.

26.3.3 Der ORACLE _ DATAPUMP Treiber

Der Oracle Datapump Treiber ermöglicht es, Daten aus Dumpfiles zu lesen und Daten in ein Dumpfile zu schreiben. Die Syntax betreffend ändern sich nur zwei Dinge:

- Die `TYPE`-Klausel
- Die Access parameter

Das folgende Beispiel zeigt, wie die Daten der Bankkunden aus einem Dumpfile gelesen werden, anstatt aus einer CSV-Datei.

Listing 26.41: Eine externe Tabelle, basierend auf einem Dumpfile

```

SQL> CREATE TABLE kunden_ext (
  2  Vorname          VARCHAR2(30),
  3  Nachname         VARCHAR2(30),
  4  Geburtsdatum     DATE,
  5  Adresse           VARCHAR2(200),
  6  PersAuswNr       VARCHAR2(9),
  7  AusstDatum       DATE,
  8  Ablaufdatum      DATE,
  9  Staatsangeh      VARCHAR2(2),
 10 Geburtsort        VARCHAR2(50),
 11 Telefon           VARCHAR2(15),
 12 Email             VARCHAR2(35),
 13 IBAN              VARCHAR2(22),
 14 KtoEroeffDat     DATE,
 15 Freistellung      NUMBER(12, 2)
 16 )
17 ORGANIZATION EXTERNAL
 18 (

```

```
19  TYPE ORACLE_DATAPUMP  
20  DEFAULT DIRECTORY data_pump_dir
```

Listing 26.42: Eine externe Tabelle, basierend auf einem Dumpfile - Fortsetzung

```

21  ACCESS PARAMETERS
22  (
23    LOGFILE 'bankkunden.log',
24    COMPRESSION enabled
25  )
26  LOCATION('bankkunden.dmp')
27 );

```

Die Accessparameter, die zur Beschreibung des Aufbaus der Quelldatei dienten entfallen und hinzukommt die Möglichkeit, den Inhalt eines Dumpfiles zu komprimieren.



Der **COMPRESSION**-Access parameter wirkt sich nur aus, wenn die Daten aus der Datenbank in das Dumpfile entladen werden.



- [b28319et_dp_driver]

26.4 Informationen

26.4.1 Verzeichnis der relevanten Data Dictionary Views



- [sthref1938]
- [REFRN23339]
- [i1576965]
- [i1577333]
- [i1577411]

27 Grundlagen des Backup und Recovery

Inhaltsangabe

27.1 Was ist Backup and Recovery?

Im Allgemeinen stehen die Begriffe Backup und Recovery für verschiedenste Strategien und Arbeitsabläufe, mit deren Hilfe versucht wird, eine Datenbank gegen Datenverlust zu schützen.

27.1.1 Physische und logische Backups

Ein Backup ist eine Kopie der Daten der Datenbank, das dazu genutzt werden kann, die Datenbank wiederherzustellen. Backups werden in physische und logische Backups unterteilt.

- **Physische Backups:** Dies sind Kopien von Datenbankdateien, d. h. sie enthalten sowohl die Datenbankstruktur, als auch die Daten selbst. Sie können in den verschiedensten Formen vorliegen, z. B. komprimiert oder auch als inkrementelles Backup.
- **Logische Backups:** Solche Backups enthalten Nutz- und Metadaten von Datenbankobjekten, die mit Hilfe von Oracle Utilities aus der Datenbank exportiert wurden. Hierbei handelt es sich nur um Datenexporte, nicht aber um vollständige Backups.

Physische Backups sind die Grundlage für jede Backup and Recovery Strategie. Logische Backups stellen in einigen Situationen eine hilfreiche Ergänzung zu den physischen Backups dar, sind jedoch kein ernstzunehmender Schutz für eine Datenbank gegen Datenverlust.

Dieser Teil der Unterrichtsunterlage kümmert sich im Wesentlichen nur um physische Backups. Daher wird mit dem Begriff Backup, wenn nicht anders angegeben, auch immer auf ein physisches Backup verwiesen.

27.1.2 Ursachen die ein Recovery notwendig machen

Obwohl es viele Ursachen gibt, die die normale Funktion einer Oracle Datenbank beeinträchtigen können, gibt es im Wesentlichen nur zwei, die das Eingreifen des Administrators erfordern:

- Medienfehler
- Bedienerfehler

Bedienerfehler

Bedienerfehler liegen immer dann vor, wenn es entweder durch eine fehlerhafte Anwendungslogik oder durch eine direkte Fehleingabe eines Nutzers zu Datenverlust kommt. Dies äußert sich meist in fälschlicherweise geänderten oder gelöschten Daten. Obwohl durch Nutzerschulungen, sowie dem vorsichtigen und umsichtigen Umgang mit Privilegien, die meisten Bedienerfehler vermieden werden können, bestimmt die Backup und Recovery Strategie, wie einfach und unkompliziert der Administrator die verlorenen Daten wiederherstellen kann.

Medienfehler

Unter dem Begriff Medienfehler versteht man den Verlust von Daten, aufgrund der Fehlfunktion eines Datenträgers. Keine Datenbankdatei ist gegen diese Art von Fehler geschützt. Die passende Recovery Strategie für solche Fälle hängt davon ab, welche Datenbankdateien betroffen sind.

27.2 Unterschiedliche Arten physischer Backups mit RMAN

Physische Backups können nach den unterschiedlichsten Gesichtspunkten unterschieden werden, z. B. in welchem Zustand (geöffnet oder geschlossen) war die Datenbank zum Zeitpunkt des Backups, welche Teile der Datenbank wurden gesichert und in welcher Form wird das Backup gespeichert.

27.2.1 Image Copies, Backup Sets und Backup Pieces

Image Copy

RMAN (Recovery Manager) kann ein Backup entweder als Image Copy oder als Backup Set speichern. Eine Image Copy ist eine 1:1 Kopie einer Datei, die auch mit Hilfe von Betriebssystemmitteln erstellt werden könnte. Der Vorteil der Nutzung von RMAN bei der Erstellung von Image Copies ist, dass er bei diesem Vorgang den Inhalt der Image Copy auf korrupte Blöcke prüfen kann. Image Copies werden im RMAN Repository registriert.

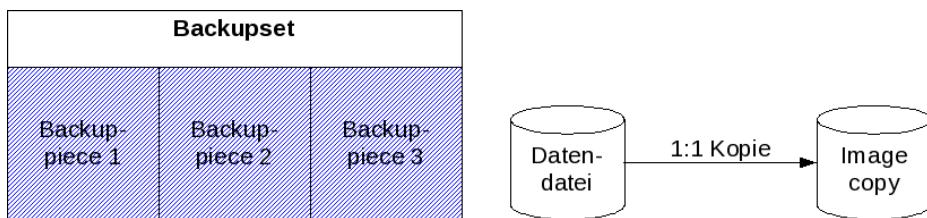
Backup Set

Eine andere Form der Speicherung von Backups sind Backup Sets. Ein Backup Set besteht aus mehreren Backup Pieces. Ein Backup Piece ist eine binäre Archivdatei, welche mit einer ZIP-Datei verglichen werden kann. Ein mit RMAN durchgeföhrter Backup-Job kann eines oder mehrere Backup Sets erzeugen. Backup Sets können auch nur durch RMAN wieder zurückgeschrieben werden.



RMAN kann nur Backup Sets auf Bandlaufwerke übertragen.

Abb. 27.1:
Backup Set und
Image Copy



27.2.2 Konsistente und inkonsistente Backups

Physische Backups werden in konsistente Backups (Cold Backup) und inkonsistente Backups (Hot Backup) unterschieden. Ein Backup wird als konsistent bezeichnet, wenn die Datenbank zum Zeitpunkt des Backups in einem konsistenten Zustand war. Das ist dann der Fall, wenn alle Änderungen der Redo Logs in die Datendateien übertragen wurden. Dies geschieht nur, wenn die Datenbank ordnungsgemäß heruntergefahren wurde. Eine Datenbank, die aus einem konsistenten Backup wiederhergestellt wurde, kann sofort ohne weiteres Recovery geöffnet werden.

Es können aber auch Backups einer geöffneten Datenbank durchgeführt werden. Dabei handelt es sich dann um inkonsistente Backups, sogenannte Hot Backups. Wurde eine Datenbank aus einem inkonsistenten Backup wiederhergestellt, muss in jedem Falle ein Media Recovery durchgeführt werden, um alle Änderungen der Redo Logs in die Datendateien zu übernehmen. Da hier auch archivierte Redo Logs benötigt werden, muss sich die Datenbank im Archivelog-Modus befinden.

27.2.3 Vollständige und inkrementelle Backups

Als vollständig werden Backups dann bezeichnet, wenn sie komplett Datendateien enthalten. Solche Backups können mit dem RMAN oder mit Betriebssystemmitteln erzeugt werden. Inkrementelle Backups basieren auf der Idee, nur die Datenblöcke einer Datendatei zu sichern, die sich seit dem letzten vollständigen Backup geändert haben. Der Vorteil dieser Methode ist, dass durch das Zurückschreiben kompletter Datenblöcke der Zeitaufwand für das Zurückspielen der Redo Logs erheblich reduziert und somit die

gesamte Recovery Phase verringert wird. Inkrementelle Backups können nur mit RMAN durchgeführt werden.

27.3 Oracle Backup and Recovery Lösungen

Oracle kennt zwei verschiedene Möglichkeiten physische Backups zu erstellen.

- **Recovery Manager (RMAN):** Der RMAN ist ein Programm, das sowohl über die Kommandozeile als auch über den Enterprise Manager bedient werden kann. Er erstellt eigene Sessions auf dem Datenbankserver, um seine Backup- oder Recovery-Operationen durchzuführen.
- **User-Managed Backup and Recovery:** Bei dieser Methode wird mit Hilfe von SQL*Plus und Betriebssystemkommandos ein Backup der Datenbank erstellt bzw. zurückgespielt. Hierbei ist der Administrator selbst für die Verwaltung der Backups zuständig.

Beide Methoden werden von Oracle unterstützt und sind vollständig dokumentiert. Die bevorzugte Variante stellt jedoch der RMAN dar. Er liefert eine einheitliche, betriebssystemunabhängige Bedienoberfläche und bietet zu den Möglichkeiten die das User-Managed Backup and Recovery nicht kennt.

27.3.1 Backup and Recovery Features des RMAN

Wie bereits erwähnt, besitzt der RMAN einige wesentliche Vorteile gegenüber dem User-Managed Backup and Recovery. Die Wissenswertesten davon sind:

- **Inkrementelle Backups:** Inkrementelle Backups sind kompakter als vollständige Backups und können schneller durchgeführt werden. Sie verkürzen auch die Recovery Phase, da durch inkrementelle Backups weniger Redo Log-Informationen zurückgespielt werden müssen.
- **Block media recovery:** Eine Datendatei welche nur einige wenige zerstörte Blöcke enthält, kann online repariert werden.
- **Unused block compression:** RMAN kann in bestimmten Fällen unbenutzte Blöcke bei einem Backup auslassen.
- **Binary compression:** Durch einen integrierten Kompressionsalgorithmus werden Backups komprimiert.
- **Encrypted backups:** Backups können verschlüsselt werden.

Der RMAN reduziert den administrativen Aufwand bei der Verwaltung von Backups, da er ein eigenes Verzeichnis, das *RMAN Repository* führt, in dem Informationen über Backups gespeichert werden. RMAN kann mit Hilfe dieses Verzeichnisses genau bestimmen, welches das für diese Situation optimalste Backup ist, das zum Restore benutzt werden soll. Des Weiteren hat der Administrator die Möglichkeit, sich Berichte aus dem RMAN Repository ausgeben zu lassen.

Primär speichert der RMAN seine benötigten Informationen in der Kontrolldatei der Datenbank. Es kann jedoch auch ein *Recovery Catalog* erstellt werden, um das RMAN Repository zu erweitern. Dies ist ein Datenbankschema, in das der RMAN Informationen über eine oder mehrere Datenbanken speichern kann. Für den RMAN Catalog sollte eine eigenständige Datenbank verwendet werden.

27.3.2 Welche Dateien kann RMAN sichern?

RMAN kann alle für ein Recovery der Datenbank benötigten Dateien sichern. Im Einzelnen sind dies folgende:

- Datendateien und Image Copies von Datendateien
- Kontrolldateien und Image Copies von Kontrolldateien
- Archivierte Redo Logs
- Das aktuelle SPFILE
- Backup pieces anderer RMAN Backups

Andere Dateien die für den Betrieb der Datenbank benötigt werden, wie z. B. Netzwerkkonfigurationsdateien (tnsnames.ora, listener.ora sqlnet.ora) oder die Passworddatei, können nicht durch den RMAN gesichert werden. Für diese Dateien muss ein anderer Backupmechanismus eingesetzt werden.

Der RMAN kann Backups sowohl auf einem Storage-Laufwerk (Festplatte, RAID usw.), als auch auf Tapes erstellen. Bandlaufwerke werden oft auch als SBT¹-Geräte bezeichnet. RMAN kommuniziert mit SBT-Geräten über einen sogenannten *Media Management Layer*.

¹SBT = System Backup to Tape

27.4 Unterschiedliche Formen des Recovery

27.4.1 Data File Media Recovery

Das Data File Media Recovery oder kurz Media Recovery ist die am häufigsten benötigte Form des Recovery. Es wird durchgeführt, um verlorene Datendateien, SPPfiles oder Kontrolldateien wiederherzustellen. In den folgenden Situation ist ein Media Recovery notwendig:

- Es musste das Backup einer Datendatei wieder hergestellt werden.
- Es wurde eine Backup-Kontrolldatei wieder hergestellt.
- Eine Datendatei wurde ohne die Option `OFFLINE NORMAL` offline geschaltet

Damit das Recovery einer Datendatei durchgeführt werden kann, muss mindestens eine der folgenden Bedingungen erfüllt sein:

- Die betreffende Datenbank ist heruntergefahren
- Die betreffende Datendatei ist offline, falls die Datenbank nicht heruntergefahren werden kann.

Eine Datendatei die ein Recovery benötigt, kann erst in den Online-Status gebracht werden, wenn das Recovery abgeschlossen wurde. Eine Datenbank kann nicht geöffnet werden, wenn eine ihrer Datendateien ein Recovery benötigt.

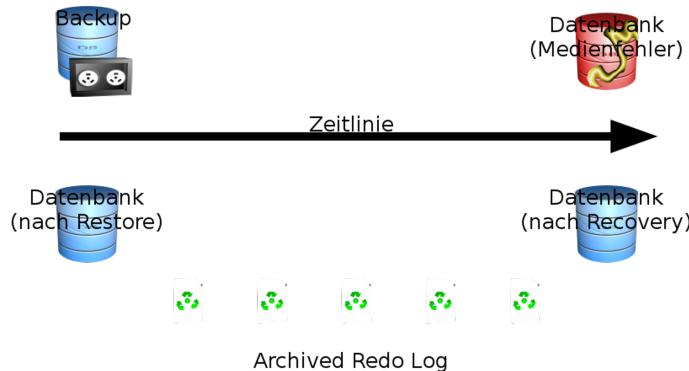
Die Phasen eines Recovery-Prozesses

Das Wiederherstellen von Teilen einer Datenbank oder einer kompletten Datenbank wird in zwei Phasen unterteilt:

- **Restore:** Unter dem Begriff Restore versteht man das Zurückspielen eines Backups auf den Datenträger.
- **Recovery:** Als Recovery bezeichnet man den Prozess des Aktualisierens der zurückgespielten Datenbankdateien, mit den Informationen aus den (archivierten) Redo Log Dateien.

Abbildung ?? zeigt das Grundprinzip von Backup, Restore und Recovery.

Abb. 27.2:
Grundprinzipien
des Backup and
Recovery



In diesem Beispiel wird bei SCN² 75 ein vollständiges Backup der Datenbank gemacht. Die von der Datenbank erzeugten Redo Logs enthalten alle Änderungen von SCN 75 bis SCN 666. Gefüllte Redo Logs werden archiviert. Bei SCN 666 gehen Datendateien der Datenbank durch einen Medienfehler verloren. Durch ein Restore wird die Datenbank dann auf den Stand von SCN 75 zurückgebracht. Beim anschließenden Recovery, mit Hilfe der Redo Logs und der archivierten Redo Logs, wird die Datenbank wieder auf den Stand von SCN 666 überführt.

Vollständiges und unvollständiges Recovery

Beim Data File Media Recovery unterscheidet man zwei verschiedene Arten:

- Vollständiges Recovery
- Unvollständiges Recovery



Unter einem vollständigen Recovery versteht man das Recovern der gesamten Datenbank oder auch nur von Teilen der Datenbank, bis auf den neuesten Stand. D. h. die Datenbank wird ohne den Verlust von abgeschlossenen Transaktionen wiederhergestellt.

In einigen Fällen kann es jedoch notwendig sein, die Datenbank bis zu einem ganz bestimmten Zeitpunkt zurückzusetzen. Wenn beispielsweise durch einen Bedienerfehler eine Tabelle gelöscht wurde, die noch benötigte Informationen enthält, muss die Datenbank bis zu dem Zeitpunkt direkt vor dem Bedienerfehler zurückgesetzt werden, so dass die betreffende Tabelle wieder existiert.

²SCN = System Change Number

Diese Form des Recovery wird als unvollständiges Recovery oder als Point-In-Time-Recovery bezeichnet. Ziel dieser Recoveryform ist es, die Datenbank auf eine ganz bestimmte SCN oder einen bestimmten Zeitpunkt zurückzusetzen, um die Folgen von Bedienerfehlern zu beheben.

Point-In-Time-Recovery stellt auch die einzige Reaktionsmöglichkeit dar, wenn der Administrator feststellt, das eines oder mehrere archivierte Redo Logs verloren gegangen sind, die für ein vollständiges Recovery benötigt würden.



Bemerkt der Administrator im laufenden Betrieb der Datenbank, dass Redo Log Dateien verloren gegangen sind, sollte augenblicklich ein vollständiges Backup der Datenbank durchgeführt werden.

27.4.2 Instance Recovery / Crash Recovery

Wird eine Oracle Datenbank neu gestartet, prüft der SMON, ob ein automatisches Recovery der Datendateien notwendig ist. Ziel dieser Form des Recovery ist es, die Datenbank vor dem Öffnen auf einen konsistenten Stand zu bringen, ohne abgeschlossene Transaktionen zu verlieren.

Instance Recovery und Media Recovery haben ähnliche Ziele. Es existieren jedoch auch einige Unterschiede zwischen den beiden Formen.

- Media Recovery muss durch einen Nutzer eingeleitet werden, es wird niemals automatisch durchgeführt.
- Media Recovery behandelt nur die durch ein Restore wiederhergestellten Datendateien, jedoch keine Datendateien die während des Crashes online waren.
- Media Recovery benötigt die Online Redo Logs und die archivierten Redo Logs um die Datenbank vollständig wiederherstellen zu können.

Im Gegensatz zum Media Recovery benötigt das Instance Recovery nur die Online Redo Logs und es wirkt sich nur auf die Datendateien aus, die während des Neustarts der Datenbank den Status online hatten. Es werden keine archivierten Redo Logs benötigt und es wird auch kein Restore durchgeführt.

Die Datenbank rollt beim Instance Recovery alle offenen Transaktionen zurück (Rollback-Phase) und wendet anschließend den Inhalt der Online Redo Logs auf die Datendateien an (Rollforward-Phase). So wird die Datenbank auf den neuesten Stand vor dem Crash gebracht.

28 Konfigurieren des Recovery Manager

Inhaltsangabe

28.1 Die Architektur des Recovery Managers

Tabelle 28.1: Die Komponenten der RMAN-Umgebung

Komponente	Beschreibung	Benötigt?
Zieldatenbank	Hierbei handelt es sich um die Datenbank (Kontrolldateien, Datendateien und optional auch Archive Log Dateien), die durch RMAN gesichert werden soll. RMAN verwendet die Kontrolldateien der Zieldatenbank, um Informationen über diese zu gewinnen und um eigene Informationen zu dieser Datenbank zu speichern. Die Backup- bzw. Recoveryjobs werden durch Serverprozesse der Zieldatenbank durchgeführt.	Ja
RMAN Client	Dies ist die Clientanwendung, die die Backup and Recovery Operationen ausführt. Da der RMAN Oracle Net-fähig ist, kann er sich von jedem beliebigen Rechner aus zur Zieldatenbank verbinden.	Ja
Recovery Katalog Datenbank	Als Recovery Catalog wird eine Datenbank bezeichnet, welche die von RMAN benötigten Metainformationen speichert.	Nein
Recovery Katalog Schema	Dieses Schema wird in der Recovery Katalog Datenbank angelegt und enthält die Metainformationen des RMAN. Es wird regelmäßig durch RMAN mit der Kontrolldatei der Zieldatenbank synchronisiert.	Nein
Media Management Anwendung	Dies sind herstellerspezifische Anwendungen, die es dem RMAN erlauben, Backup Sets auf Streamertapes zu kopieren.	Nein
Media Management Katalog	Der Media Management Katalog wird in Zusammenhang mit einer Media Management Anwendung verwendet und speichert Angaben, die RMAN benötigt, um auf ein Tapelaufwerk zugreifen zu können.	Nein

28.1.1 Der Kommandozeilen Client

Globalization Support Variablen für RMAN setzen

Vor dem Aufruf des Recovery Managers ist es notwendig, dass die beiden Umgebungsvariablen NLS_DATE_FORMAT und NLS_LANG gesetzt werden. Sie beeinflussen die Formatierung von Datums- und Zeitangaben im RMAN. Das folgende Beispiel zeigt mögliche Einstellungen für beide:

Listing 28.1: Beispiel für NLS_DATE_FORMAT und NLS_LANG

```
-- Ohne Zeichensatz
[oracle@FEA11-119SRV ~]$ export NLS_LANG=german_germany

-- Mit Zeichensatz
[oracle@FEA11-119SRV ~]$ export NLS_LANG=german_germany.UTF8

[oracle@FEA11-119SRV ~]$ export NLS_DATE_FORMAT='DD.MM.YYYY HH24:MI:SS'
```



Bei der Angabe eines Zeichensatzes in der Variablen NLS_LANG ist immer der Zeichensatz zu verwenden, den das Betriebssystem aktuell in Nutzung hat. Unter Linux wird standardmäßig der WE8ISO8859P1 Zeichensatz genutzt, unter Windows der WE8MSWIN1252.

Der folgende Literaturhinweis liefert weiterführende Informationen zur Konfiguration der National Language Support Umgebung!



- [NLSPG189]

Starten von RMAN

RMAN wird auf der Kommandozeile durch die Eingabe von rman gestartet.

Listing 28.2: Starten des RMAN

```
[oracle@FEA11-119SRV ~]$ rman target /
[oracle@FEA11-119SRV ~]$ rman target sys/oracle@ORCL NOCATALOG
[oracle@FEA11-119SRV ~]$ rman target / catalog rman/cat@CATDB
[oracle@FEA11-119SRV ~]$ rman target / catalog rman/cat@CATDB \
> auxiliary sys/oracle@AUXDB as sysdba
```



Das Schlüsselwort target gibt an, dass sich RMAN mit der Zieldatenbank verbinden soll. Um sinnvoll arbeiten zu können, muss sich RMAN mit einer Zieldatenbank verbunden haben.

In den beiden ersten Zeilen wird darauf verzichtet, sich mit einem Recovery Katalog zu verbinden. Das Schlüsselwort NOCATALOG drückt dies explizit aus, es kann jedoch auch weggelassen werden. Zeile Nummer drei verbindet RMAN mit einer Zieldatenbank und zusätzlich mit einem Recovery Katalog.

Für spezielle Arbeiten, wie z. B. das Duplizieren einer Datenbank oder das Aufbauen einer Standby Datenbank, kann es erforderlich sein, sich zusätzlich mit einer Hilfsdatenbank¹ zu verbinden. Die meisten Tätigkeiten erfordern jedoch nur, dass RMAN sich mit einer Zieldatenbank verbindet. Die Verbindung zu einem Recovery Katalog ist grundsätzlich optional.

Um den RMAN zu verlassen wird das Kommando exit verwendet.

¹Auxiliary Database = engl. Hilfsdatenbank

Verbindungen und Authentifizierung



Um sich im RMAN mit einer Zieldatenbank oder einer Hilfsdatenbank verbinden zu können, benötigt der Nutzer das SYSDBA-Privileg. Die Authentifizierung kann dabei über eine Passworddatei oder das Betriebssystem erfolgen.

Da das Verbinden mit einer Ziel- oder Hilfsdatenbank immer das SYSDBA-Privileg voraussetzt, ist es im RMAN, anders als bei SQL*Plus, nicht notwendig die Klausel `as sysdba` mit anzugeben.

Verwendet die Zieldatenbank eine Passworddatei, kann zur Authentifizierung im RMAN ein Passwort verwendet werden. Soll die Betriebssystemauthentifizierung innerhalb von RMAN genutzt werden, kann die Umgebungsvariable `ORACLE_SID` gesetzt werden. Sie muss den Instanznamen der Zieldatenbank enthalten.

Eingeben von Kommandos

Wenn der RMAN gestartet wurde, wird folgendes Commandprompt angezeigt:

Listing 28.3: Das RMAN-Commandprompt

```
RMAN >
```

Ab diesem Zeitpunkt können beliebige RMAN-Kommandos eingegeben werden:

Listing 28.4: Beispiel für einige RMAN-Kommandos

```
RMAN > CONNECT target /
RMAN > CONNECT catalog rman/cat@CATDB

RMAN > BACKUP database;
```

Fast alle RMAN-Kommandos akzeptieren einen oder mehrere Parameter und müssen mit einem Semikolon ';' abgeschlossen werden. Wenige Ausnahmen, wie z. B. `startup`, `shutdown` oder `connect` können ohne Semikolon benutzt werden.

Ein RMAN-Kommando kann auf mehrere Zeilen verteilt werden.

Listing 28.5: Ein mehrzeiliges RMAN-Kommando

```
RMAN > BACKUP database
2>   INCLUDE current controlfile;
```

RMAN zum Starten und Herunterfahren der Datenbank benutzen

Wenn ein Vorgang es erfordert, dass die Zieldatenbank gestartet, heruntergefahren, in den MOUNT- oder NOMOUNT-Status versetzt wird, kann der RMAN die entsprechende Änderung an der Datenbank herbeiführen. Im folgenden Beispiel werden ein Shutdown, ein Startup und einige andere SQL-Statements durchgeführt:

Listing 28.6: Startup und Shutdown im RMAN

```
[oracle@FEA11-119SRV ~]$ rman target /  
  
RMAN> shutdown immediate  
RMAN> startup nomount  
RMAN> SQL 'ALTER DATABASE NOMOUNT';  
RMAN> SQL 'ALTER DATABASE MOUNT';  
RMAN> SQL 'ALTER DATABASE OPEN';
```

28.1.2 Das RMAN Repository

Als RMAN Repository wird eine Sammlung von Metadaten bezeichnet, die der RMAN für Backup, Recovery und Wartungsarbeiten benötigt. RMAN speichert dieses Repository immer in der Kontrolldatei der Zieldatenbank. Von der Korrektheit der Kontrolldatei hängt es ab, welchen Stand der Datenbank der RMAN kennt und welche Backups er wiederherstellen kann. Dies ist einer der Gründe, warum die Kontrolldatei eine besondere Rolle bei der Erstellung einer Backupstrategie spielt. RMAN kann, allein mit Hilfe der Kontrolldatei alle notwendigen Backup und Recovery Operationen ausführen.

Optional kann ein Recovery Katalog erstellt werden. Dies ist ein Schema in einer Oracle-Datenbank, welches die gleichen Informationen speichert wie die Kontrolldatei. Im Unterschied zu einer Kontrolldatei, deren Einträge im RMAN Repository nur eine begrenzte Lebensdauer haben, ehe sie überschrieben werden, kann der Recovery Katalog diese Informationen unbegrenzt speichern. Die erhöhte Komplexität die durch die Verwaltung eines Recovery Katalogs entsteht, kann schnell durch die Bequemlichkeit ersetzt werden, die er bietet.

Einige Features des RMAN funktionieren nur mit Hilfe eines Recovery Katalogs, z. B. „RMAN Stored Scripts“ und alle Kommandos, die mit solchen Skripten in Verbindung stehen. Andere RMAN Kommandos sind speziell für die Verwaltung eines Recovery Katalogs und werden nicht benötigt, wenn keiner verwendet wird.

Der Recovery Katalog wird durch den RMAN selbst verwaltet. Die Zieldatenbank greift niemals in ihn ein. RMAN gibt automatisch alle notwendigen Informationen, aus der Kontrolldatei der Zieldatenbank an den Recovery Katalog weiter, falls eine Änderung eintritt.

Das RMAN Repository in der Kontrolldatei

Das RMAN Repository kennt zwei Eintragsarten: Circular Reuse Records und Noncircular Reuse Records.

Circular Reuse Records enthalten Daten, die einer hohen Änderungshäufigkeit unterliegen und bei Bedarf überschrieben werden können. Sie sind als „logischer Ring organisiert“, was bedeutet, dass sobald alle Speicherplätze belegt sind, entweder neuer Platz durch eine Erweiterung der Kontrolldatei geschaffen wird oder das bestehende Speicherplätze überschrieben werden. `control_file_record_keep_time` ist der Initialisierungsparameter, der vorgibt, nach wie vielen Tagen ein Eintrag überschrieben werden kann.

Non Circular Records enthalten Informationen, die für den Betrieb der Datenbank wichtig sind. Deshalb können diese nicht durch den RMAN überschrieben werden. Dazu zählt beispielsweise, welche Daten- und Redo Log Dateien eine Datenbank enthält.

Das RMAN Repository im Recovery Katalog

Wird ein RMAN Katalog benutzt, sollte er sich nicht in der Zieldatenbank befinden. Wird er dennoch dort gespeichert, ist er verloren, wenn die Zieldatenbank verloren geht, was das Recovery der Zieldatenbank deutlich erschwert.

Um den Recovery Katalog mit einer Zieldatenbank zu verknüpfen, muss diese bei ihm registriert werden. Es können beliebig viele Datenbanken registriert werden. RMAN unterscheidet die einzelnen Datenbanken anhand einer DBID.



Jede Oracle-Datenbank besitzt eine eigene, eindeutige DBID. Diese kann der View `V$DATABASE` entnommen werden.

Im Einzelnen beinhaltet der Recovery Katalog Informationen über folgende Dinge:

- Backup Sets und Pieces von Datendateien, Redo Logs und Archive Logs
- Kopien von Datendateien
- Kopien von Archive Logs

- Tablespaces und Datendateien der Zieldatenbank(en)
- RMAN stored scripts
- Dauerhafte Konfigurationseinstellungen des RMAN

Resynchronisieren des Recovery Katalogs

Damit der Recovery Katalog aktuell bleibt, muss er immer wieder mit dem RMAN Repository resynchronisiert werden. Dies geschieht mit Hilfe eines „Snapshot Controlfile“, dass immer dann durch den RMAN automatisch erzeugt wird, wenn eine Resynchronisation notwendig ist. Es stellt eine konsistente Momentaufnahme der Kontrolldatei dar. Da solche Snapshots nur kurzzeitig verwendet werden, werden sie nicht im Recovery Katalog registriert. RMAN speichert die Checkpoint SCN aus dem Snapshot Controlfile im Recovery Katalog, um die Aktualität des Katalogs sicherzustellen.

Die Datenbank garantiert, dass immer nur ein RMAN-Prozess auf ein Snapshot Controlfile zugreift. Dies ermöglicht, dass sich unterschiedliche RMAN-Prozesse nicht gegenseitig stören.

28.2 Konfigurieren des RMAN

Die Konfiguration des RMAN ist einfach und unkompliziert, da für fast alle Parameter Standardwerte existieren. Dadurch ist es möglich, ohne manuelle Konfiguration direkt mit dem RMAN zu arbeiten.

Um den RMAN jedoch effizienter nutzen zu können, ist es notwendig die wichtigsten Optionen des RMAN zu kennen. RMAN-Parameter können mit dem Kommando `CONFIGURE` gesetzt und gespeichert werden, so dass sie nicht bei jedem Backup erneut geändert werden müssen. Das Kommando `SHOW` ermöglicht die Ausgabe aller Optionen.

28.2.1 Die aktuelle RMAN-Konfiguration anzeigen

Mit Hilfe des Kommandos `SHOW` können die aktuellen Werte der RMAN-Parameter angezeigt werden.

Listing 28.7: Das SHOW-Kommando

```
RMAN> SHOW retention policy;
RMAN> SHOW device type;
```

```
RMAN> SHOW default device type;
RMAN> SHOW channel;

RMAN> SHOW all;
```

Das Kommando `SHOW all` zeigt alle Optionen als Serie von `CONFIGURE`-Kommandos. Sie kann in eine Textdatei gespeichert werden, die dann dazu dienen kann, die Konfiguration für eine andere Datenbank einzurichten. Das Spooling in eine Textdatei wird mit Hilfe des `SPOOL LOG TO Dateiname`-Kommandos aktiviert und mit `SPOOL LOG OFF` wieder deaktiviert.

Listing 28.8: Ausgabe des Kommandos SHOW-ALL

```
RMAN> SPOOL LOG TO '/home/oracle/rman.cfg';
RMAN> SHOW ALL;
RMAN> SPOOL LOG OFF;
```

28.2.2 Anpassen der RMAN-Konfiguration

Das Backup-Standardgerät festlegen

Standardmäßig legt der RMAN alle Backups in einem Verzeichnis auf der Festplatte ab. Er kann aber auch so konfiguriert werden, dass er seine Backups auf ein SBT-Gerät speichert.

Nach der Konfiguration des SBT-Gerätes, kann es im RMAN zum Backup-Standardgerät gemacht werden:

Listing 28.9: Konfigurieren des Backup-Standardgeräts

```
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
```

Anschließend werden alle Backups, bei denen kein Backupgerät angegeben wurde, auf das Backup-Standardgerät gespeichert. Um eine Festplatte als Backup-Standardgerät einzurichten, muss dem `DEFAULT DEVICE TYPE`-Parameter der Wert „disk“ gegeben werden:

Listing 28.10: Konfigurieren des Backup-Standardgeräts auf Festplatte

```
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO disk;
```

Auch ohne das Einrichten eines Backup-Standardgeräts, können Backups auf SBT-Gerät oder Festplatte gespeichert werden. Das folgende Beispiel zeigt zwei unterschiedliche `BACkUP`-Kommandos:

Listing 28.11: Backup-Beispiele

```
RMAN> BACKUP DEVICE TYPE SBT database;
RMAN> BACKUP DEVICE TYPE DISK database;
RMAN> BACKUP database;
```

Das erste Kommando `BACKUP DEVICE TYPE sbt DATABASE` speichert das Backup der Zieldatenbank auf ein SBT-Gerät. Das zweite sichert das Backup auf Festplatte, da als Device Type der Wert „disk“ vorgegeben wurde. Das dritte `BACKUP database`-Kommando benutzt das Backup-Standardgerät.

Parameter, die einem Kommando, wie z. B. `BACKUP`, `RESTORE` oder `RECOVER` mitgegeben werden, haben immer Vorrang vor einer Einstellung, die mit `CONFIGURE` vorkonfiguriert wurde.

Listing 28.12: Überschreiben einer Konfigurationseinstellungen

```
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO sbt;
RMAN> BACKUP database
2>     DEVICE TYPE disk;
```

Konfigurieren des Standard-Backuptyps

Der Standard-Backuptyp legt fest, welche Art von Backup gemacht wird, wenn nichts spezielles definiert wurde. Zur Auswahl stehen:

- Unkomprimierte Backup Sets
- Komprimierte Backup Sets
- Image Copies

Wurde hier nichts abweichendes konfiguriert, erstellt RMAN unkomprimierte Backup Sets. Es existiert keine Möglichkeit Image Copies für ein SBT-Gerät zu konfigurieren, da RMAN nur Backup Sets auf SBT-Geräte schreiben kann.

Listing 28.13: Standard-Backuptyp festlegen

```
RMAN> CONFIGURE DEVICE TYPE disk
2>     BACKUP TYPE TO copy;
RMAN> CONFIGURE DEVICE TYPE disk
2>     BACKUP TYPE TO backupset;

RMAN> CONFIGURE DEVICE TYPE disk
2>     BACKUP TYPE TO compressed backupset;
RMAN> CONFIGURE DEVICE TYPE sbt
2>     BACKUP TYPE TO compressed backupset;
```

Komprimierte Backup Sets können sowohl auf Festplatte, als auch auf einem SBT-Gerät gespeichert werden.

28.2.3 RMAN auf seine Standardkonfiguration zurücksetzen

Jeder RMAN-Parameter kann auf seinen Standardwert zurückgesetzt werden. Dies geschieht mit dem RMAN-Kommando `CONFIGURE ... CLEAR`.

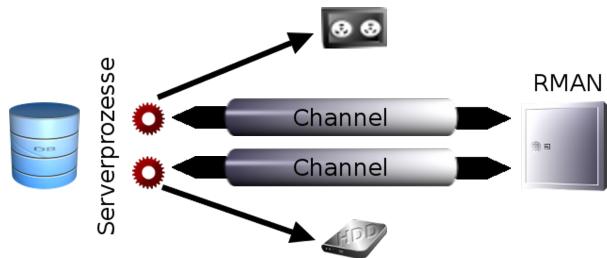
Listing 28.14: CONFIGURE ... CLEAR

```
RMAN> CONFIGURE BACKUP OPTIMIZATION CLEAR;
RMAN> CONFIGURE RETENTION POLICY CLEAR;
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP FORMAT FOR DEVICE TYPE DISK CLEAR;
```

28.3 Kanäle für RMAN konfigurieren

Ein Kanal im Sinne von RMAN ist ein Datenstrom zwischen einem Speichergerät und einem Serverprozess. Die meisten RMAN-Kommandos werden mit Hilfe eines Serverprozesses ausgeführt. Wie in Abbildung ?? zu sehen ist, erstellt jeder Kanal eine Verbindung zu einer Zieldatenbank, in dem er dort einen Serverprozess startet. Der Serverprozess führt dann die Backup-, Restore- oder Recovery-Tätigkeiten durch.

Abb. 28.1:
RMAN und
Channels



Kanäle können in RMAN auf zwei unterschiedliche Arten erstellt werden: automatisch oder manuell.

28.3.1 Die manuelle Kanalanforderung

In RMAN ist es möglich Kanäle manuell, für ganz bestimmte Zwecke anzufordern. Dies geschieht durch das `ALLOCATE`-Kommando, wie in Beispiel ?? gezeigt wird.

Listing 28.15: Manuelle Kanalanforderung

```
RMAN> RUN
```

```

2> {
3>   ALLOCATE CHANNEL c1 DEVICE TYPE disk;
4>   BACKUP database PLUS ARCHIVELOG;
5> }
```

Die Zeile `ALLOCATE CHANNEL c1 DEVICE TYPE disk` fordert einen Kanal namens `c1` an. Dieser liest seine Daten von einem Disk-Gerät. Zu beachten ist an dieser Stelle der RUN-Block.



RUN-Blöcke sind vergleichbar mit Prozeduren/Methoden aus der Programmierung. Sie stellen eine in sich geschlossene Einheit dar, die eine Abfolge von Befehlen enthält, welche sequenziell abgearbeitet wird. Für die manuelle Kanalanforderung ist es zwingend notwendig, einen RUN-Block zu benutzen.

RMAN benutzt den angeforderten Kanal für alle Operationen, die innerhalb des RUN-Blocks definiert sind. Nach dem der RUN-Block abgearbeitet worden ist, wird der Kanal automatisch wieder geschlossen.

28.3.2 Die automatische Kanalanforderung

Seit Oracle 10g ist es nicht mehr zwingend erforderlich, Kanäle manuell anzufordern. Der RMAN hält standardmäßig immer einen generischen Kanal für alle Operationen bereit.



Ein Kanal ist generisch, wenn er nur auf den Standardeinstellungen von RMAN beruht. Das heißt, er wird z. B. durch das Standardbackupgerät beeinflusst.

Kanäle vordefinieren

Mit Hilfe des Kommandos `CONFIGURE CHANNEL` können die Konfigurationseinstellungen für Kanäle geändert werden. Das folgende Beispiel zeigt, wie für einen Kanal der Speicherort (das Format) konfiguriert wird.

Listing 28.16: Vordefinieren eines Channels mit Speicherort

```

RMAN> CONFIGURE CHANNEL DEVICE TYPE disk
2> FORMAT '/u04/backup/ora_df%t_%s%p.bkp';
```

Dieses Kommando legt als Speicherort für den Channel das Verzeichnis /u04/backup/ fest. Das Format für den Dateinamen der Backups gliedert sich wie folgt:

- Der Dateiname beginnt mit den Buchstaben ora_df.
- Der Platzhalter %t wird durch einen Zeitstempel ersetzt.
- Der Platzhalter %s wird mit der Nummer des Backup Sets ersetzt.
- Der Platzhalter %p wird mit der Nummer des Backup Piece ersetzt.



Zu beachten bei der Konfiguration eines expliziten Speicherorts für Backup Sets ist: Wird ein expliziter Speicherort für Backup Sets konfiguriert, werden die Backup Sets außerhalb der Fast Recovery Area (siehe ??) gespeichert. Damit geht die gesamte Funktionalität der Fast Recovery Area verloren.

Ein anderes Beispiel für die Konfiguration eines Kanals zeigt Beispiel ??.

Listing 28.17: Ein vordefinierter Channel mit Backup Piece Size

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE disk
2> MAXPIECESIZE = 500M;
```

In diesem Beispiel wird die Maximalgröße für die einzelnen Backup Pieces auf 500 Megabyte festgelegt. Jedes über diesen Kanal angefertigte Backup Set, wird automatisch in 500 Megabyte Stücke, sogenannte Backup Pieces, zerlegt.

Abb. 28.2:
Ein Backup Set,
das aus drei
Backup Pieces
besteht

Backupset		
Backup-piece 1	Backup-piece 2	Backup-piece 3
500 MB	500 MB	500 MB

Optimierung automatisch angeforderter Kanäle

RMAN optimiert die Nutzung von automatisch angeforderten Kanälen dahin gehend, dass ein solcher Kanal nur so lange offen bleibt, wie er benötigt wird.



Im folgenden Beispiel sind drei Backup-Operationen zu sehen. Die Erste öffnet den vordefinierten Kanal. Die zweite und die dritte Operation benötigen einen exakt genauso konfigurierten Kanal, wie die erste Operation. Deshalb wird der Kanal nicht jedes mal neu geöffnet, sondern bleibt offen.

Erst nach der Benutzung eines der beiden Kommandos `ALLOCATE` oder `CONFIGURE` wird der vordefinierte Kanal wieder geschlossen.

Listing 28.18: Optimierung der automatischen Kanalanforderung

```
RMAN> BACKUP datafile 1; --Kanal wird geoeffnet
RMAN> BACKUP current controlfile; -- Kanal wird erneut genutzt
RMAN> BACKUP archivelog all; -- Kanal wird weiter genutzt und bleibt offen
RMAN> CONFIGURE DEFAULT DEVICE TYPE TO disk; -- Der Kanal wird geschlossen
```

28.3.3 Parallelisierung bei der Kanalanforderung

Mit Hilfe der Parallelisierung können, in einem Arbeitsschritt, mehrere gleichartige Kanäle synchron angefordert werden.

Parallelisierung manuell angeforderter Kanäle

Bei der manuellen Kanalanforderung ist die Parallelisierung sehr einfach einzurichten. Im folgenden Beispiel werden zwei Kanäle angefordert, c1 und c2. Da mehrere Dateien gesichert werden müssen, kann RMAN die Parallelisierung nutzen und die Arbeit wird auf beide Kanäle verteilt.

Listing 28.19: Parallelisierung und die manuelle Kanalanforderung

```
RMAN> RUN {
2>   ALLOCATE CHANNEL c1 DEVICE TYPE disk;
3>   ALLOCATE CHANNEL c2 DEVICE TYPE disk;
4>   BACKUP database PLUS ARCHIVELOG;
5> }
```



RMAN nutzt immer so viele Kanäle, wie er für seine aktuelle Operation benötigt. Sind zwei Kanäle angefordert und RMAN kann nur einen für seine Arbeit nutzen (z. B. weil nur eine Datei gesichert werden soll), nutzt er auch nur den ersten angeforderten Kanal.

Parallelisierung automatisch angeforderter Kanäle

In seiner Standardeinstellung hat RMAN nur einen einzigen Kanal zur Verfügung. Um mehrere Kanäle zu nutzen, muss die Parallelisierung aktiviert werden. Beispiel ?? zeigt das Kommando zur parallelen Anforderung von 3 Festplatten-Kanälen.

Listing 28.20: Parallelisierung von Kanälen

```
RMAN> CONFIGURE DEVICE TYPE disk PARALLELISM 3;
```

Hier gilt das gleiche Prinzip, wie bei der manuellen Kanalanforderung: Es werden immer nur so viele Kanäle genutzt, wie zur Verfügung stehen und benötigt werden. Jeder einzelne dieser Kanäle kann mit Hilfe des **CONFIGURE CHANNEL**-Kommandos konfiguriert werden.

Listing 28.21: Unterschiedliche Kanäle vordefinieren

```
RMAN> CONFIGURE CHANNEL 0 DEVICE TYPE disk
2>   MAXPIECESIZE = 500M;
RMAN> CONFIGURE CHANNEL 1 DEVICE TYPE sbt
2>   PARMS 'SBT_LIBRARY=oracle.disksbt,ENV=(BACKUP_DIR=/u04)';
RMAN> CONFIGURE CHANNEL 2 DEVICE TYPE disk
2>   FORMAT '/u02/backup/backupset_%u.bkp';
```



Mit **CONFIGURE CHANNEL n** werden Kanäle durch Nummern von 0 bis n angesprochen. Somit ist es möglich, getrennte Konfigurationen für Kanäle anzulegen. Die Kommandos **CONFIGURE CHANNEL** und **CONFIGURE CHANNEL 0** sind identisch.

Wenn alle parallel angeforderten Kanäle die gleichen Einstellungen aufweisen sollen, kann das Kommando **CONFIGURE CHANNEL** genutzt werden, um Kanal Nummer 0 zu konfigurieren. Wurde nur dieser eine Kanal eingerichtet, besitzen alle geöffneten Kanäle die gleichen Vorgaben.

Listing 28.22: Parallele Anforderung von Kanälen mit gleicher Konfiguration

```
RMAN> CONFIGURE DEVICE TYPE disk
2> PARALLELISM 3;
RMAN> CONFIGURE CHANNEL DEVICE TYPE disk
2> MAXPIECESIZE = 500M
3> FORMAT '/u02/backup/%d_%D-%M-%Y_%u.bkp';
```

Im vorangegangenen Beispiel werden drei Kanäle parallel genutzt. Die Konfiguration für alle drei Kanäle wird von der Konfiguration des Kanals Nummer 0 abgeleitet.

Namenskonventionen der automatischen Kanalanforderung

RMAN benutzt die Konvention „ORA_gerät_n“ für die Benennung automatisch angeforderter Kanäle. Der Platzhalter „gerät“ steht dabei für das Gerät, auf das der Kanal zeigt, beispielsweise „disk“ oder „sbt_tape“. n steht für die laufende Nummer des Kanals. RMAN benennt den ersten Festplatten-Kanal „ORA_DISK_1“, den zweiten „ORA_DISK_2“ usw. Für Streamergeräte lautet der Name des ersten automatisch angeforderten Kanals „ORA_SBT_TAPE_1“.

Das folgende Beispiel zeigt den Zusammenhang zwischen automatischer Kanalanforderung, Parallelisierung und deren Namenskonventionen.

Der Nutzer setzt folgende Kommandos ab:

Listing 28.23: Namenskonventionen für RMAN-Kanäle

```
RMAN> CONFIGURE DEVICE TYPE disk PARALLELISM 3;
RMAN> BACKUP database PLUS ARCHIVELOG;
```

Effektiv tut RMAN folgendes:

Listing 28.24: Namenskonventionen für vordefinierte RMAN-Kanäle 2

```
RMAN> RUN {
2>     ALLOCATE CHANNEL ORA_DISK_1 DEVICE TYPE disk;
3>     ALLOCATE CHANNEL ORA_DISK_2 DEVICE TYPE disk;
4>     ALLOCATE CHANNEL ORA_DISK_3 DEVICE TYPE disk;
5>
6>     BACKUP database PLUS ARCHIVELOG;
7> }
```



Beachtet werden muss, das Kanäle die mit dem Präfix *ORA_* beginnen, nur für die interne Nutzung durch RMAN angefordert werden können. Wird ein Kanal mit einem solchen Namen manuell angefordert, quittiert RMAN dies mit der folgenden Fehlermeldung:

```
RMAN-00571: =====
RMAN-00569: ====== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-03002: Fehler bei allocate Befehl auf 07/22/2013 14:06:01
RMAN-06472: Kanal-ID ORA_DISK_1 wird automatisch zugewiesen
```

Automatisches Channel-Failover

Sind für eine Operation mehrere Kanäle angeforderte, kann RMAN im Falle dessen, dass ein Kanal ausfällt, auf einen anderen Kanal wechseln. Somit kann die Operation unter Umständen trotz des Ausfalls eines Kanals fortgesetzt werden. Dieser Mechanismus wird als „Channel-Failover“ bezeichnet.

28.4 Autobackup für Kontrolldateien und SPFiles konfigurieren

In vielen Recoverysituationen ist es sehr wichtig, ein Backup der aktuellen Kontrolldatei zu besitzen. Um die Wahrscheinlichkeit zu erhöhen, dass ein solches Backup existiert, bietet Oracle die Möglichkeit, das „Controlfile Autobackup“ einzurichten. Es wird ein automatisches Backup der Kontrolldatei angefertigt, sobald Änderungen am RMAN Repository gemacht wurden.

Mit Hilfe eines Controlfile Autobackups kann RMAN die Datenbank auch dann wiederherstellen, wenn die aktuelle Kontrolldatei, der Recovery Katalog und das SPFile nicht mehr verfügbar sind. Um das automatische Controlfile Autobackup zu konfigurieren, werden die beiden folgenden Kommandos verwendet:

Listing 28.25: Controlfile Autobackup konfigurieren

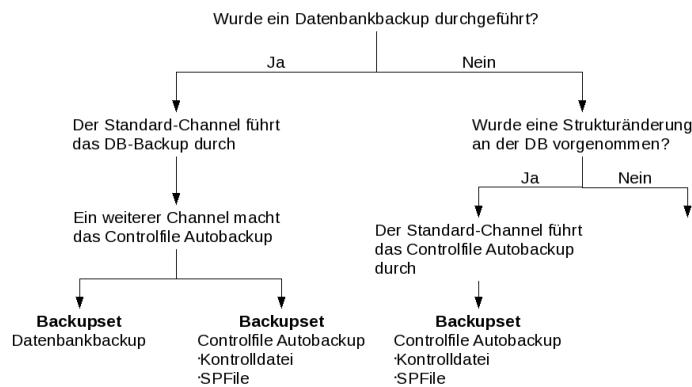
```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP OFF;
```



Oracle empfiehlt, dass das Controlfile Autobackup immer aktiviert sein sollte.

Der Ablauf eines Controlfile Autobackup gliedert sich wie folgt:

Abb. 28.3:
Ablauf eines
Controlfile
Autobackups



Nachdem das Controlfile Autobackup abgeschlossen wurde, wird der komplette Pfad und der Dateiname des Backup Sets in die Alert.log Datei eingetragen.

Wie in Abbildung ?? zu sehen, wird nur in zwei Situationen ein Controlfile Autobackup durchgef\u00fchrt:

1. Wenn ein Backup erfolgreich verlaufen ist.
2. Wenn eine strukturelle \u00e4nderung an der Datenbank vorgenommen wurde (hinzuf\u00f6gen oder entfernen von Tablespace, \u00e4ndern von Redo Log Gruppen, u. \u00e4.).

Im ersten Fall wird das Autobackup durch einen RMAN Channel durchgef\u00fchrt. Im zweiten Fall f\u00fchrt der Serverprozess, der die Struktur\u00e4nderung an der Datenbank vorgenommen hat das Autobackup durch.



Sobald Datendatei Nummer 1, der System-Tablespace in einem Backup enthalten ist, wird immer die Kontrolldatei mitgesichert, auch wenn das Controlfile Autobackup deaktiviert ist.



- [cfandspfileautobackups]

28.5 Die Backup Retention Policy

Die Backup Retention Policy² legt fest, welche Backups erhalten bleiben müssen, um ein Recovery der Datenbank durchführen zu können. Diese Regel kann auf einem Zeitfenster (Recovery Window) basieren oder auf Redundanz.

28.5.1 Zeitfensterbasierte Retention Policy einrichten

Als Recovery Window wird ein Zeitraum bezeichnet, der vom aktuellen Datum ausgehend x Tage in die Vergangenheit reicht. Innerhalb dieses Zeitfensters kann die Datenbank mit einem Point-In-Time-Recovery in jeden beliebigen Zustand zurückversetzt werden. Der frühest mögliche Zeitpunkt, an den eine Datenbank zurückversetzt werden kann, wird als *Point Of Recoverability* bezeichnet.

Listing 28.26: RECOVERY WINDOW setzen

```
RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
```

Diese Regel stellt sicher, dass für jede Datendatei ein Backup existiert, das älter ist, als der Point of Recoverability, in diesem Falle also älter als sieben Tage.

Alle Backups, die älter sind, als das aktuellste Backup, das die genannte Bedingung erfüllt gelten als obsolet (nicht mehr benötigt).

Das folgende Beispiel soll die Nutzung eines Recovery Window verdeutlichen. Folgendes gilt:

- Es wurden Backups am 01.06.20XX und am 13.06.20XX gemacht.
- Die Datenbank befindet sich im Archivelog Modus und alle notwendigen Archive sind vorhanden.

In Abbildung ?? ist das aktuelle Datum der 30.06.20XX. Der Point of Recoverability liegt am 23.06.20XX, sieben Tage vom aktuellen Datum zurück. Um ein Recovery gemäß der gültigen Retention Policy machen zu können, wird das Backup vom 13.06.20X, sowie die Archive Logs von Sequenz 550 bis 800, benötigt. Das Backup vom 01.06.20XX und alle Archive Logs vor Sequenz 550 sind obsolet und können gelöscht werden.

²Retention Policy (wörtliche Übersetzung): Erhaltungsregel

Abb. 28.4:
Nutzung eines
Recovery
Window

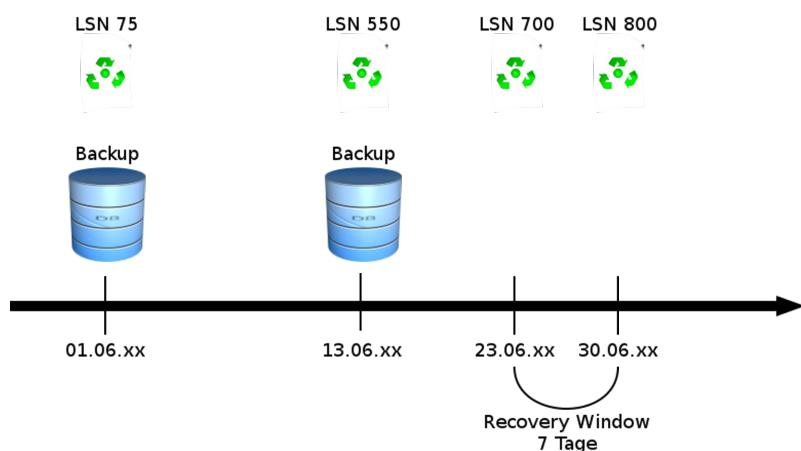
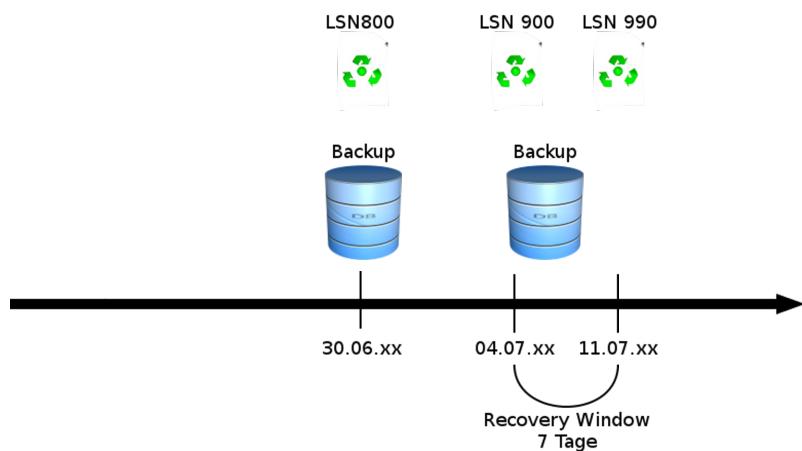


Abbildung ?? zeigt ein verändertes Szenario:

Abb. 28.5:
Nutzung eines
Recovery
Window



Das aktuelle Datum ist der 11.07.20XX und der Point of Recoverability ist der 04.07.20XX. Um die Retention Policy zu erfüllen, werden nun das Backup vom 30.06.20XX und alle Archive von Logs von Sequenz 800 an benötigt. Das Backup vom 06.07.20X ist zwar aktueller als das vom 30.06.20XX, es kann aber die Retention Policy nicht erfüllen, da es jünger ist als der Point of Recoverability.

28.5.2 Eine redundanzbasierte Retention Policy einrichten

Eine redundanzbasierte Retention Policy legt fest, wie viele Backups von einer Datendatei vorhanden sein müssen, um die Policy zu erfüllen.

Listing 28.27: REDUNDANCY setzen

```
RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 2;
```

Dieses Kommando legt fest, dass von jeder Datendatei mindestens zwei Backups vorhanden sein müssen. Hierbei wird kein Unterschied zwischen Image Copy und Backup Set gemacht.

Welche der beiden Verwaltungsmethoden vorzuziehen ist, hängt davon ab, ob ein Point-In-Time-Recovery der Datenbank in Frage kommt oder nicht. Falls nicht, erweist sich die redundanzbasierte Methode als platzsparender. Im anderen Falle ist die zeitfensterbasierte Variante besser, da mittels des Point Of Recoverability genau bestimmt werden kann, wie weit ein Point-In-Time-Recovery in die Vergangenheit zurückreichen kann.



Die Standardeinstellung für die Retention Policy ist REDUNDANCY = 1.

28.5.3 Die Retention Policy anzeigen lassen

Mit Hilfe des `SHOW`-Kommandos kann die aktuelle Retention Policy angezeigt werden.

Listing 28.28: Anzeigen der Retention Policy

```
RMAN> SHOW RETENTION POLICY;
```

28.5.4 Die Retention Policy abschalten

Mit dem folgenden Kommando kann die Retention Policy abgeschaltet werden:

Listing 28.29: Abschalten der Retention Policy

```
RMAN> CONFIGURE RETENTION POLICY TO NONE;
```

28.5.5 Backups aus der Retention Policy ausschließen

Es kann vorkommen, dass man bestimmte Backups für besondere Zwecke länger aufbewahren muss, als die Retention Policy dies zulässt. Solche Langzeit Backups können im RMAN Repository verzeichnet sein, müssen aber von der Retention Policy ausgenommen werden.

Ein Backup kann von der Retention Policy ausgeschlossen werden, indem:

- Die `KEEP`-Option des `BACKUP`-Kommandos verwendet wird

Listing 28.30: Ein neues Backup aus der Retention Policy ausschließen

```
RMAN> BACKUP database KEEP UNTIL TIME "31.12.2014" NOLOGS  
2> FORMAT "/u03/backup/0RCL/backup_%s_%p.bkp";
```

- Das `CHANGE`-Kommando mit der `KEEP`-Option verwendet wird.

Listing 28.31: Ein bestehendes Backup aus der Retention Policy ausschließen

```
RMAN> CHANGE BACKUPSET 2 KEEP UNTIL TIME "31.12.2014"  
2> NOLOGS;
```

Backups die von der Retention Policy ausgeschlossen sind, sind nach wie vor vollständig gültig und nutzbar. Um ein Backup wieder in die Retention Policy einzuschließen, wird die `NOKEEP`-Option des `CHANGE`-Kommandos verwendet.

LOGS und NOLOGS

Die Optionen `LOGS` und `NOLOGS` legen fest, ob die zu einem Backup benötigten Archive Logs (alle Archive Logs die neuer sind als das Backup) mit von der Retention Policy ausgeschlossen werden (`LOGS`) oder ob die Archive Logs nicht berücksichtigt werden sollen (`NOLOGS`).

Es gibt die Möglichkeit, ein Backup für immer aus der Retention Policy auszuschließen. Dies funktioniert mit der Option `KEEP FOREVER`. Hierbei sollte jedoch immer die Option `NOLOGS` verwendet werden, da sonst alle Archive Logs, die jünger als das Backup sind für immer aufbewahrt werden müssen.

Einschränkungen der KEEP-Option

Für die Nutzung der `KEEP`-Option gibt es die folgenden Einschränkungen.

- Werden Backup Sets mit einem Keep-Attribut versehen, können diese nicht in der Fast Recovery Area (siehe ??) gespeichert werden. Aus diesem Grund muss hier zwingend ein alternativer Speicherort mit dem `FORMAT`-Schlüsselwort gesetzt werden. Andernfalls bricht RMAN den Backupjob mit der Fehlermeldung ab:

Listing 28.32: Keine Dateien mit KEEP-Attribut in der FRA aufbewahren!

```
% ORA-19811: Dateien in DB_RECOVERY_FILE_DEST mit Keep-Attribut  
sind nicht moeglich.
```

- Backups, die Archive Logs beinhalten, können nicht aus der Retention Policy ausgeschlossen werden. Oracle bricht einen solchen Versuch mit der folgenden Fehlermeldung ab:

Listing 28.33: Backups mit Archive Logs dürfen kein KEEP-Attribut haben!

```
RMAN-06530: CHANGE ... KEEP not supported for backup set which contains  
archive logs.
```

28.6 Die Fast Recovery Area



Mit Oracle 10g wurde die „Flash Recovery Area“ eingeführt. Da es immer wieder zu Verwechslungen mit der Technologie „Flashback Database“ kam, wurde die Flash Recovery Area, in Oracle 11g in „Fast Recovery Area“ umbenannt.

Wie bereits erwähnt, ist die Fast Recovery Area ein Speicherbereich auf einem Datenträger, der verschiedene, für das Recovery der Datenbank benötigte Dateien zwischenspeichert (Cache-Funktion).

Die Fast Recovery Area erleichtert die Administration der Datenbank dahingehend, dass Backupdateien automatisch benannt werden, dass sie automatisch solange vorgehalten werden, wie sie für ein Restore and Recovery benötigt werden und dass sie automatisch gelöscht werden, wenn sie nicht mehr benötigt werden.

28.6.1 Planen der Fast Recovery Area

Die in der Fast Recovery Area gespeicherten Dateien werden in zwei Kategorien eingeteilt.

Permanente Dateien

Die einzigen permanenten Dateien sind gespiegelte Kopien der Kontrolldatei und der Redo Log Dateien. Diese Dateien können nicht gelöscht werden, ohne einen Absturz der Instanz zu bewirken.

Flüchtige Dateien

Alle anderen Dateien gelten als flüchtig und werden von Oracle automatisch gelöscht, wenn Sie aufgrund der Backup Retention Policy als veraltet gelten oder wenn sie durch ein Backup erfasst wurden.

Dies schließt folgende Dateien ein:

- Archivierte Redo Logs
- Image Copies von Datendateien
- Image Copies von Kontrolldateien
- Kontrolldatei-Autobackups
- Backup Pieces

Den geeigneten Speicherplatz für die Recovery Area finden

Bevor eine Fast Recovery Area erstellt werden kann, muss der geeignete Speicherort für sie gefunden werden (Verzeichnis oder ASM Disk Group).



Eine Fast Recovery Area kann nicht auf einem Raw-Datenträger gespeichert werden.

Des Weiteren muss für die Fast Recovery Area eine Disk Quota festgelegt werden, die bestimmt, wie groß die Area werden darf.

Die Fast Recovery Area sollte auf einem anderen Datenträger liegen, als die Datenbank, da sonst die Gefahr besteht, das Datenbank und Fast Recovery Area, verloren gehen.

Die Größe der Fast Recovery Area planen

Die Fast Recovery Area sollte groß genug sein, um folgende Dateien aufzunehmen:

- Kopien aller Datendateien
- Inkrementelle Backups
- Online Redo Logs
- Archivierte Redo Logs, die in noch keinem Backup gesichert wurden
- Kontrolldateien
- Kontrolldatei-Autobackups

Sollte so viel Speicherplatz nicht zur Verfügung stehen, ist es das Beste, die Fast Recovery Area groß genug zu machen, um Backups der wichtigsten Tablespace zu speichern und alle archivierten Redo Logs, die in noch keinem Backup gesichert wurden. Im Minimum muss die Fast Recovery Area jedoch so groß sein, dass sie die noch nicht durch ein Backup gesicherten archivierten Redo Logs aufnehmen kann.



Wie groß man seine Fast Recovery Area planen sollte hängt von verschiedenen Faktoren ab. Diese sind unter anderem:

- Finden viele oder nur sehr wenige Änderungen an der Datenbank statt?
- Werden Backups nur auf Festplatte oder auch auf SBT-Geräte gespeichert?
- Wie viele Backups müssen ständig verfügbar sein?
- Soll der Flashback Database Mechanismus genutzt werden oder nicht?

28.6.2 Konfigurieren der Fast Recovery Area

Um die Fast Recovery Area zu aktivieren, müssen die beiden Initialisierungsparameter `db_recovery_file_dest_size` und `db_recovery_file_dest` gesetzt werden.



Der Parameter `db_recovery_file_dest_size` muss zuerst gesetzt werden. Er legt den maximal zur Verfügung stehenden Speicherplatz für die Fast Recovery Area fest. Der Wert für diesen Parameter sollte immer so geplant werden, dass ein Verwaltungsoverhead von ca. 10% auf dem betreffenden Datenträger übrig bleibt.

Listing 28.34: DB_RECOVERY_FILE_DEST_SIZE setzen

```
SQL> ALTER SYSTEM SET db_recovery_file_dest_size=4G;
```



`db_recovery_file_dest` legt ein Verzeichnis oder eine ASM Disk Group als Speicherort für die Fast Recovery Area fest.

Listing 28.35: DB_RECOVERY_FILE_DEST_SIZE setzen

```
SQL> ALTER SYSTEM SET db_recovery_file_dest='/u05/fast_recovery_area';
```

Die Views `v$recovery_file_dest` und `v$recovery_area_usage`

Diese beiden Views enthalten nützliche Informationen darüber, ob genügend Speicherplatz für die Fast Recovery Area zur Verfügung gestellt wird.

- V\$RECOVERY_FILE_DEST: Enthält Informationen über den Speicherort, die Disk Quota, den genutzten Speicher und den noch zur Verfügung stehenden Speicher in der Fast Recovery Area.

Listing 28.36: V\$RECOVERY_FILE_DEST

```
SQL> col name format a23
SQL> SELECT name, space_limit / POWER(1024, 2) AS Space_Limit,
2      ROUND(space_used / POWER(1024, 2),2) AS space_used,
3      number_of_files
4  FROM v$recovery_file_dest;

NAME          SPACE_LIMIT SPACE_USED NUMBER_OF_FILES
-----        -----       -----      -----
/u05/fast_recovery_area           4096      640,09            4
```

- V\$RECOVERY_AREA_USAGE: Zeigt Informationen darüber an, wie viel Speicherplatz jede Dateiart in der Fast Recovery Area belegt und wie viel Speicherplatz durch das Löschen als veraltet markierter Dateien gewonnen werden kann.

Listing 28.37: V\$RECOVERY_AREA_USAGE

```
SQL> SELECT file_type,
2      TO_CHAR(percent_space_used, '90.00') AS pct_space_used,
3      TO_CHAR(percent_space_reclaimable, '90.00') AS pct_space_recl,
4      number_of_files
5  FROM v$recovery_area_usage;

FILE_TYPE          PCT_SP PCT_SP NUMBER_OF_FILES
-----        -----       -----      -----
CONTROL FILE        0.00    0.00            0
REDO LOG           0.00    0.00            0
ARCHIVED LOG        0.00    0.00            0
BACKUP PIECE       15.63    0.23            4
IMAGE COPY          0.00    0.00            0
FLASHBACK LOG        0.00    0.00            0
FOREIGN ARCHIVED LOG        0.00    0.00            0
```

- [sthref3529]
- [sthref3528]



Die Fast Recovery Area abschalten

Um die Fast Recovery Area abzuschalten, muss lediglich der Initialisierungsparameter db_recovery_file_dest auf einen Null-String ('') zurückgesetzt werden.

Listing 28.38: Die Fast Recovery Area abschalten

```
SQL> ALTER SYSTEM
  2  SET db_recovery_file_dest='';
```

Sowohl die in der Fast Recovery Area gespeicherten Dateien als auch die Einträge im RMAN Repository bleiben dabei erhalten.

Verhalten der Fast Recovery Area bei einem Instanz-Crash

Grundsätzlich verwaltet sich die Fast Recovery Area selbst. Im Falle eines Instanz-Crashes kann es jedoch vorkommen, dass aktuell geöffnete Dateien unvollständig in der Fast Recovery Area zurückbleiben. Oracle zeigt dies mit der folgenden Fehlermeldung im Alert Log:

Listing 28.39: Beschädigte Dateien in der Fast Recovery Area

```
ORA-19816: WARNING: Files may exist in location that are not known to database.
```

location wird ersetzt durch den aktuellen Speicherort der Fast Recovery Area.

Liegt ein solcher Fall vor, gibt es zwei Möglichkeiten, um die Situation zu bereinigen:

- Die betreffende Datei kann, falls der Dateiheader unbeschädigt ist, einfach erneut durch RMAN erfasst und im Repository gespeichert werden.

Listing 28.40: Beschädigte Dateien in der Fast Recovery Area neu katalogisieren

```
RMAN> CATALOG RECOVERY AREA;
```

- Die Datei muss manuell gelöscht werden, da der Dateiheader zerstört/ungültig ist.

28.6.3 Speicherplatzverwaltung in der Fast Recovery Area

Oracle löscht keine Dateien in der Fast Recovery Area, bevor nicht weiterer Speicherplatz benötigt wird. Der daraus resultierende Effekt ist, dass Dateien die bereits auf ein SBT-Gerät gesichert wurden, meist noch einige Zeit danach in der Fast Recovery Area vorhanden sind. Somit fungiert die Area als Cache.

Welche Dateien werden wann gelöscht

Es sind vier einfache Regeln, von denen drei hier genannt werden, wann welche Dateien aus der Fast Recovery Area entfernt werden:

- Permanente Dateien werden nie gelöscht.
- Dateien, die durch die Retention Policy als veraltet markiert wurden werden gelöscht, wenn neuer Speicherplatz benötigt wird.
- Flüchtige Dateien, die von einem Backup erfasst wurden, werden gelöscht, wenn weiterer Speicherplatz benötigt wird.



Welche Dateien in der Fast Recovery Area zuerst gelöscht werden, kann nicht genau bestimmt werden.

Wenn nicht genug Speicher zur Verfügung steht...

Ist die Retention Policy des RMAN so eingestellt, dass mehr Speicherplatz für Backups benötigt wird, als in der Fast Recovery Area vorhanden ist oder wurde die Retention Policy abgeschaltet, kann es passieren, dass die Fast Recovery Area sich zu 100% füllt.

Sind nur noch weniger als 15% freier Speicher in der Fast Recovery Area vorhanden, wird eine Warnung durch die Datenbank ausgegeben. Sind es nur noch 3%, wird eine kritische Warnung angezeigt. Die Datenbank wird solange neuen Speicher anfordern, bis keiner mehr vorhanden ist.

Eine komplett gefüllte Fast Recovery Area löst folgende Fehlermeldung aus:

Listing 28.41: 100% gefüllte Fast Recovery Area

```
ORA-19809: limit exceeded for recovery files
ORA-19804: cannot reclaim nnnnn bytes disk space from mmmmm limit
```

nnnnn stellt dabei die benötigte Speichermenge und *mmmmmm* die Disk Quota der Fast Recovery Area dar.

Oft reagiert die Datenbank auf so einen Fehler, in dem sie stehen bleibt. Wenn zum Beispiel ein als zwingend markierter Speicherort der Archived Logs in der Fast Recovery Area liegt, kann die Datenbank nicht mit ihrem Betrieb fortfahren, bis das Problem behoben ist.

Bereinigen einer vollen Fast Recovery Area

Es gibt verschiedene Möglichkeiten, eine zu 100 % gefüllte FRA zu bereinigen:

- Mehr Speicherplatz verfügbar machen, in dem der Wert des Initialisierungsparameters `db_recovery_file_dest_size` erhöht wird.
- Backups von den Dateien in der Fast Recovery Area auf Band anfertigen(siehe ??) und anschließend, mit Hilfe des RMAN Kommandos `DELETE` die Fast Recovery Area entleeren.



Manuelle Eingriffe in die Fast Recovery Area mit Betriebssystemmitteln sollten vermieden werden, da die Datenbank derartige Änderungen nicht registriert und der Speicherplatz nach wie vor als belegt gilt. Solche Fehler können anschließend nur mittels des RMAN Kommandos `CROSCHECK` behoben werden.

Die Fast Recovery Area an einen neuen Platz verschieben

Muss die Fast Recovery Area an einen neuen Platz verschoben werden, genügt es, das SQL*Plus-Tool zu öffnen und den Parameter `db_recovery_file_dest` anzupassen:

Listing 28.42: Den Speicherort der Fast Recovery Area ändern

```
SQL> ALTER SYSTEM
  2  SET db_recovery_file_dest='/u04/fast_recovery_area'
  3  SCOPE=both;
```



Anschließend an die Änderung dieses Parameters, werden alle Dateien am neuen Speicherort erstellt. Alle bereits bestehenden Dateien verbleiben am Alten und verbrauchen weiterhin Speicherplatz in der Fast Recovery Area, bis sie durch die automatische Verwaltung der Fast Recovery Area gelöscht werden.

Die nächsten Schritte sind nur dann notwendig, wenn auch die bestehenden Dateien an den neuen Speicherort überführt werden müssen. Mit Hilfe des `BACKUP`-Kommandos können einzelne Dateiarten in die neue Fast Recovery Area gesichert werden.

Listing 28.43: Verschieben von Dateien in eine neue Fast Recovery Area

```
RMAN> BACKUP AS COPY ARCHIVELOG ALL DELETE INPUT;

RMAN> BACKUP DEVICE TYPE DISK BACKUPSET ALL DELETE INPUT;
```

```
RMAN> BACKUP AS COPY DATAFILECOPY ALL DELETE INPUT;
```

Hiermit werden alle Archive Logs, Backup Sets und Image Copies in die neue Fast Recovery Area gesichert und am alten Speicherort gelöscht. Bei allen anderen Dateitypen sorgt die automatische Verwaltung für das Löschen, wenn die Dateien nicht mehr benötigt werden.

Zusammenspiel zwischen Retention Policy und der Fast Recovery Area

Wird ein Backup als obsolet gekennzeichnet, ist dies immer auf eine im RMAN konfigurierte Retention Policy zurückzuführen. Wird die Fast Recovery Area konfiguriert, benutzt die Datenbank einen internen Algorithmus, um Dateien bestimmen zu können, die von der Festplatte gelöscht werden können. Es gibt zwei Fälle, wann Dateien durch die Verwaltung der Fast Recovery Area gelöscht werden:

- Dateien tragen den Status obsolete, da sie gegen die konfigurierte Retention Policy verstößen.
- Dateien sind noch nicht obsolete, wurden aber anderweitig, z. B. auf Tape, gesichert.

28.7 Übungen - Konfigurieren des Recovery Manager

1. Konfigurieren Sie die unten angegebenen Werte für die beiden Umgebungsvariablen NLS_LANG und NLS_DATE_FORMAT.
 - NLS_LANG=GERMAN_GERMANY
 - NLS_DATE_FORMAT="dd.mm.yyyy hh24:mi:ss"
2. Geben Sie eine Liste aller RMAN-Einstellungen aus und speichern Sie diese in eine Datei.
3. Konfigurieren Sie im RMAN das automatische Backup von Kontrolldatei und SPFile.
4. Konfigurieren Sie die Retention Policy auf ein Zeitfenster von 7 Tagen.
5. Legen Sie als Standardbackuptyp komprimierte Backup Sets fest.
6. Setzen Sie nur die Retention Policy auf ihren Standardwert zurück.

28.8 Lösungen - Konfigurieren des Recovery Manager

1. Konfigurieren Sie die unten angegebenen Werte für die beiden Umgebungsvariablen NLS_LANG und NLS_DATE_FORMAT.

- NLS_LANG=GERMAN_GERMANY
- NLS_DATE_FORMAT=,,dd.mm.yyyy hh24:mi:ss“

```
[oracle@FEA11-119SRV ~]$ export NLS_LANG=german_germany
[oracle@FEA11-119SRV ~]$ export NLS_DATE_FORMAT="dd.mm.yyyy hh24:mi:ss"
```

2. Geben Sie eine Liste aller RMAN-Einstellungen aus und speichern Sie diese in eine Datei.

```
RMAN> SPOOL LOG TO '/home/oracle/rman.cfg';
RMAN> SHOW ALL;
RMAN> SPOOL LOG OFF;
```

3. Konfigurieren Sie im RMAN das automatische Backup von Kontrolldatei und SPFile.

```
RMAN> CONFIGURE CONTROLFILE AUTOBACKUP ON;
```

4. Konfigurieren Sie die Retention Policy auf ein Zeitfenster von 7 Tagen.

```
RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
```

5. Legen Sie als Standardbackuptyp komprimierte Backup Sets fest.

```
RMAN> CONFIGURE DEVICE TYPE DISK
2> PARALLELISM 1
3> BACKUP TYPE TO COMPRESSED BACKUPSET;
```

6. Setzen Sie nur die Retention Policy auf ihren Standardwert zurück.

```
RMAN> CONFIGURE RETENTION POLICY CLEAR;
```


29 Verwalten des Recovery Katalogs

Inhaltsangabe

29.1 Einrichten eines Recovery Katalogs

Das Einrichten eines Recovery Katalogs umfasst vier Schritte:

- Erstellen/Konfigurieren der Datenbank, die den Recovery Katalog aufnimmt
- Schaffen einer Netzwerkverbindung (TNS) zwischen der Zieldatenbank und der Recovery Katalog Datenbank.
- Den Eigentümer des Recovery Katalogs erstellen
- Den Recovery Katalog erstellen

Bevor der Katalog genutzt werden kann, muss noch die Zieldatenbank registriert werden.

29.1.1 Erstellen der Katalogdatenbank

Für die Planung des Recovery Katalogs ist es wichtig zu wissen, wie viel Speicherplatz zur Verfügung gestellt werden muss. Dies ist davon abhängig, wie viele Datenbanken durch den Katalog verwaltet werden oder wie groß die Anzahl der Archive Logs und der Backupdateien einer jeden Datenbank ist. Als drittes werden auch noch RMAN stored Scripts im Recovery Katalog gespeichert, für die ebenfalls geringe Platzreserven benötigt werden.



Laut Oracle sind ca. $15MB * \text{Anzahl reg DB} = \text{Wachstum/Jahr}$ ein guter Ansatz.

Der typische Speicherplatzverbrauch für einen Katalog teilt sich wie folgt auf:

Tabelle 29.1: Speicherplatzbedarf einer Katalogdatenbank

Art des Speicherplatzes	Verbrauch/Jahr
System-Tablespace	90 MB
Temp-Tablespace	5 MB
Undo-Tablespace	5 MB
Recovery Katalog Tablespace	15 MB pro registrierter Datenbank
Online Redo Logs	10 MB pro Member (3 Gruppen mit je 2 Membern)

29.1.2 Die TNS-Netzwerkverbindung erstellen

Um eine Netzwerkverbindung mittels TNS zwischen der Zieldatenbank und dem Recovery Katalog zu ermöglichen, sollte das Local Naming oder das Directory Naming genutzt werden. Im Folgenden wird eine Beispielkonfiguration für das Local Naming gezeigt.

Es werden folgende Annahmen getroffen:

- Der Rechner auf dem sich der Recovery Katalog befindet benutzt TCP/IP.
- Der Name des Rechners, der den Recovery Katalog enthält ist: FEA11-119CAT.
- Der Listener dieses Rechners läuft auf Port 1521.
- Der Servicename der Katalogdatenbank ist: CATDB.

Die Datei `tnsnames.ora` der Zieldatenbank sollte folgenden Abschnitt beinhalten:

Listing 29.1: Der Net Service Name der CATDB

```
CATDB =
  (DESCRIPTION=
    (ADDRESS_LIST=
      (ADDRESS= (PROTOCOL=tcp)(HOST=FEA11 -119CAT)(PORT=1521))
    )
  (CONNECT_DATA=
    (SERVICE_NAME=CATDB)))
```

29.1.3 Den Katalogeigentümer erstellen

Ist die Katalogdatenbank erst einmal erstellt und die TNS-Netzwerkverbindung geschaffen, kann der dritte Schritt, das Erstellen des Katalogeigentümers vorgenommen werden.

1. Als Nutzer sys mit der Datenbank verbinden.

Listing 29.2: Mit der Katalogdatenbank verbinden

```
[oracle@FEA11 -119SRV ~]$ sqlplus sys/oracle@CATDB as sysdba
```

2. Erstellen des Katalogtablespaces

Listing 29.3: Den Katalogtablespace erstellen

```
SQL> CREATE TABLESPACE catts
```

```
2  DATAFILE '/u02/oradata/catdb/catts01.dbf' SIZE 15 M  
3  AUTOEXTEND ON MAXSIZE 30 M;
```

3. Erstellen des Katalogeigentümers.

Listing 29.4: Den Katalogeigentümer erstellen

```
SQL> CREATE USER catowner
  2 IDENTIFIED BY catpass
  3 DEFAULT TABLESPACE catts
  4 QUOTA unlimited ON catts;
```

4. Dem Katalogeigentümer die Rolle recovery_catalog_owner geben.

Listing 29.5: Die Rolle recovery_catalog_owner übergeben

```
SQL> GRANT create session, recovery_catalog_owner TO catowner;
```



Damit ein Nutzer Eigentümer des Recovery Katalogschemas sein kann, muss er zwingend die Rolle recovery_catalog_owner besitzen.

29.1.4 Den Recovery Katalog erstellen

Das Erstellen des Kataloges geschieht im RMAN.

1. RMAN starten und mit der Katalogdatenbank verbinden

Listing 29.6: Mit der Katalogdatenbank verbinden

```
[oracle@FEA11-119SRV ~]$ rman catalog catowner/catpass@CATDB
```

2. Den Katalog erstellen

Listing 29.7: Katalog erstellen

```
RMAN> CREATE CATALOG;
```

Das Kommando `CREATE CATALOG` erstellt den Recovery Katalog im Default Tablespace des Katalogeigentümers. Wahlweise kann dieses Kommando auch um die `TABLESPACE`-Klausel erweitert werden.



29.1.5 Registrieren einer Datenbank

Der letzte Schritt, bevor eine Datenbank mit dem Recovery Katalog genutzt werden kann, ist die Datenbank zu registrieren.

- Bei Zieldatenbank und Recovery Katalog anmelden

Listing 29.8: Mit Ziel- und Katalogdatenbank verbinden

```
[oracle@FEA11-119SRV ~]$ rman target / catalog catowner/catpass@CATDB
```

- Die Zieldatenbank in den Mount-Status versetzen

- Registrieren der Zieldatenbank

Listing 29.9: Zieldatenbank registrieren

```
RMAN> REGISTER DATABASE;
```

Während dieses Vorgangs erstellt RMAN Zeilen in den Katalogtabellen, die alle notwendigen Informationen aus der Kontrolldatei der Zieldatenbank enthalten. So wird der Recovery Katalog mit der Zieldatenbank synchronisiert. Um zu überprüfen, ob die Synchronisation erfolgreich war, kann das Kommando `REPORT SCHEMA` genutzt werden.

Listing 29.10: Schemainformationen anzeigen

```
RMAN> REPORT SCHEMA;

File  Size (MB)    Tablespace          RB  segs Datafile Name
-----  -----
1      307200   SYSTEM            NO
2      303500   SYSAUX           NO
3      20480    UNDOTBS        YES
4      10240    USERS            NO
5      10240    EXAMPLE           NO
```

29.2 Datenbanken verwalten

29.2.1 Eine weitere Datenbank registrieren

Es können mehrere Zieldatenbank in einem Recovery Katalog verwaltet werden, wenn sie unterschiedliche Datenbank-IDs aufweisen. Beim Vorgang des Duplizierens einer neuen Datenbank, mit dem RMAN-Kommando `DUPPLICATE` oder bei der Erstellung mit Hilfe des SQL-Statements `CREATE DATABASE`, wird automatisch eine neue DBID generiert. Lediglich bei Datenbankkopien, die auf anderem Wege erstellt wurden, kann es zu Problemen kommen. Hierzu muss zuerst mit dem RMAN-Kommando `DBNEWID` eine neue DatenbankID erstellt werden.



Eine Datenbank kann auch in mehreren Recovery Katalogen registriert werden.

29.2.2 Aufheben einer Datenbankregistrierung

Zum Aufheben einer Datenbankregistrierung im Recovery Katalog gibt es das Kommando `UNREGISTER DATABASE`. Dabei werden alle Einträge zu einer Datenbank aus dem Katalog gelöscht.



Informationen die zwar im Recovery Katalog gespeichert waren, aber nicht in der Kontroldatei (`control_file_record_keep_time`) gespeichert sind, gehen bei diesem Vorgang verloren.

Die Registrierung einer Datenbank wird wie folgt aufgehoben:

1. RMAN starten und eine Verbindung zur betreffenden Zieldatenbank und dem Recovery Katalog herstellen.

Listing 29.11: Starten des RMAN

```
[oracle@FEA11-119SRV ~]$ rman target / catalog catowner/catpass@CATDB
```

Es ist nicht zwingend notwendig, sich bei der Zieldatenbank anzumelden. Sind jedoch mehrere Datenbanken im Recovery Katalog registriert, muss die Zieldatenbank durch ihre DBID identifiziert werden.

2. Auflisten aller Backup Sets und Image Copies der Zieldatenbank. Dies sollte aus Sicherheitsgründen erfolgen, damit Backups wieder katalogisiert werden können.

Listing 29.12: Backup Sets und Image Copies auflisten

```
RMAN> LIST BACKUP SUMMARY;
RMAN> LIST COPY;
```

3. Soll die Datenbank dauerhaft aus dem Recovery Katalog gelöscht werden, müssen auch alle Backups der Zieldatenbank gelöscht werden.

Listing 29.13: Löschen aller Backups

```
RMAN> DELETE BACKUP DEVICE TYPE sbt;
RMAN> DELETE BACKUP DEVICE TYPE DISK;
RMAN> DELETE COPY;
```



Dieser Schritt darf nur dann ausgeführt werden, wenn auch die Zieldatenbank selbst gelöscht werden soll!!!

4. Aufheben der Datenbankregistrierung

Listing 29.14: Aufheben der Datenbankregistrierung

```
RMAN> UNREGISTER DATABASE;
```

Wird eine Datenbankregistrierung aufgehoben, ohne dass eine Verbindung zur Zieldatenbank möglich ist, muss Schritt 4 wie folgt verändert werden.

Listing 29.15: Aufheben der Datenbankregistrierung ohne Verbindung zur Zieldatenbank

```
RMAN> RUN {
2>   SET DBID=93457265485;
3>   UNREGISTER DATABASE;
4> }
```

29.3 Synchronisation des Recovery Katalogs

Wenn RMAN eine Synchronisation des Recovery Katalogs durchführt, vergleicht er den Inhalt der Kontrolldatei der Zieldatenbank, mit dem Inhalt des Recovery Katalogs und gleicht beide einander an. RMAN führt dabei folgende Schritte durch:

1. Erstellen eines Snapshot Controlfiles
2. Snapshot Controlfile und Recovery Katalog vergleichen
3. Angleichen von Snapshot Controlfile und Recovery Katalog

RMAN führt automatisch die Synchronisation des Recovery Katalogs durch. Dies geschieht in verschiedenen Situationen, wie z. B. bei Ausführung des [BACKUP](#)-Kommandos. Die Synchronisation kann jedoch auch im Bedarfsfall manuell durchgeführt werden. Dies geschieht mit dem [RESYNC CATALOG](#)-Kommando.

29.3.1 Vollständige und teilweise Resynchronisation

Die Synchronisation des Recovery Katalogs kann vollständig oder nur teilweise erfolgen. In einer teilweisen Resynchronisation liest RMAN die Kontrolldatei der Zieldatenbank und wertet diese nach neuen Backups, neuen Archive Logs und der Redo Log Historie aus. Es werden keine Schemadaten der Zieldatenbank synchronisiert. Diese werden nur bei einer vollständigen Synchronisation abgeglichen.

29.3.2 Wann sollte manuell synchronisiert werden?

Da RMAN die Synchronisation des Recovery Katalogs automatisch durchführt, ist ein manuelles Synchronisieren meist nicht notwendig. Es gibt jedoch einige Ausnahmesituationen, die im Folgenden beschrieben werden.



Grundsatz: Manuelle Synchronisation ist immer dann notwendig, wenn über einen längeren Zeitraum keine Verbindung zwischen dem Katalog und der Zieldatenbank bestanden hat.

Resynchronisation wenn der Recovery Katalog nicht verfügbar war

Es kann vorkommen, dass Backup oder Recovery Arbeiten notwendig sind, ohne dass der Recovery Katalog verfügbar ist. In so einem Fall kann keine automatische Synchronisation erfolgen.

Resynchronisation bei unregelmäßigen Backups

Folgendes Beispiel:

- Die Zieldatenbank läuft im Archive Logs Modus.
- Die Datenbank wird in unregelmäßigen Abständen gesichert.
- Zwischen den einzelnen Backups wird eine hohe Anzahl Redo Log Switches erzeugt.

Bezugnehmend auf das obige Beispiel, kann es sinnvoll sein, den Recovery Katalog manuell zu synchronisieren, da er nicht automatisch bei jedem Redo Log Switch aktualisiert wird. Informationen über Log Switches werden nur im Controlfile der Zieldatenbank gesichert. Wie aktuell der Recovery Katalog gehalten wird, hängt dann davon ab, wie oft er manuell synchronisiert wird.

Resynchronisation nach einer Änderung der Datenbankstruktur

Wie schon im vorhergehenden Fall, wird der Recovery Katalog auch bei einer Strukturänderung der Datenbank nicht automatisch resynchronisiert. Nach einer Strukturänderung an der Datenbank sollte wie folgt vorgegangen werden:

1. RMAN starten und eine Verbindung zur betreffenden Zieldatenbank und dem Recovery Katalog herstellen.

Listing 29.16: Starten des RMAN

```
[oracle@FEA11-119SRV ~]$ rman target / catalog catowner/catpass@CATDB
```

2. Resynchronisieren des Recovery Katalogs

Listing 29.17: Resynchronisieren des Recovery Katalogs

```
RMAN> RESYNC CATALOG;
```

29.3.3 CONTROL_FILE_RECORD_KEEP_TIME

Der Initialisierungsparameter `control_file_record_keep_time` legt fest, wie lange die Einträge des RMAN Repositories in der Kontrolldatei der Zieldatenbank erhalten bleiben. Überschreitet ein Eintrag diesen Wert, so wird er aus der Kontrolldatei gelöscht. Deshalb sollte darauf geachtet werden, dass die Synchronisationsvorgänge innerhalb der angegebenen Zeitspanne bleiben. Das heißt, dass der Wert für den Initialisierungsparameter `control_file_record_keep_time` so gewählt werden sollte, dass das RMAN Repository ohne Verlust in den Recovery Katalog synchronisiert werden kann.

30 Backups mit dem RMAN

Inhaltsangabe

30.1 Backups anfertigen

Backups werden im RMAN mit dem Kommando `BACKUP` durchgeführt. Dabei sind unterschiedliche Arten von Backups, wie z. B. Hot- oder Coldbackups, Full oder Incremental Backups möglich.

30.1.1 Hot- oder Coldbackup

Ob ein Backup ein Hot- oder ein Coldbackup wird, hängt vom Status der Datenbank ab. Ist die Datenbank geöffnet, wird das Backup als Hotbackup (inkonsistentes Backup) bezeichnet. Ist die Datenbank im Mount-Status, ist es ein Coldbackup (konsistentes Backup). Die beiden folgenden Beispiele zeigen, wie Hot- bzw. Coldbackups durchgeführt werden können.

Listing 30.1: Ein Coldbackup durchführen

```
RMAN> shutdown immediate  
  
RMAN> startup mount  
  
RMAN> BACKUP database;
```

Listing 30.2: Ein Hotbackup durchführen

```
RMAN> SQL 'ALTER DATABASE OPEN';  
  
RMAN> BACKUP database;
```

30.1.2 Das BACKUP-Kommando

Das `BACKUP`-Kommando verwendet die vorkonfigurierten Einstellungen und Standardeinstellungen des RMAN. Soll jedoch von diesen Einstellungen abgewichen werden, kann dies durch die Angabe von Parametern geschehen.

Das Backup-Ausgabegerät festlegen

Das `BACKUP`-Kommando kennt die Klausel `DEVICE TYPE`. Sie gibt an, ob das Backup auf einen Datenträger oder einem SBT-Gerät gespeichert werden soll. In den beiden folgenden Beispielen wird die `SHOW`-Anweisung dazu benutzt, um die aktuelle Channel-Einstellung vor dem Backup anzuzeigen.

Listing 30.3: Ein Backup to Disk

```
RMAN> SHOW CHANNEL;

RMAN configuration parameters for database with db_unique_name ORCL are:
RMAN configuration has no stored or default parameters

RMAN> BACKUP DEVICE TYPE disk database;
```

Listing 30.4: Ein Backup to Tape

```
RMAN> SHOW CHANNEL;

using target database control file instead of recovery catalog
RMAN configuration parameters for database with db_unique_name ORCL are:
CONFIGURE CHANNEL DEVICE TYPE 'SBT_TAPE'
PARMS   'SBT_LIBRARY=oracle.disksbt,ENV=(BACKUP_DIR=/u04)';

RMAN> BACKUP DEVICE TYPE sbt database;
```

Image Copy oder Backup Set?

Um ein Backup als Image Copy anzufertigen gibt es die `AS COPY`-Klausel. Image Copies können nur auf Disk-, aber nicht auf SBT-Geräte gespeichert werden.

Listing 30.5: Eine Image Copy der Datenbank erstellen

```
RMAN> BACKUP AS COPY database;
```

Das Gegenstück zur `AS COPY`-Klausel ist die `AS BACKUPSET`-Klausel.

Listing 30.6: Ein Backup Set der Datenbank erstellen

```
RMAN> BACKUP AS BACKUPSET database;
```

Kompression verwenden

RMAN ist in der Lage Backup Sets zu komprimieren, um Speicherplatz zu sparen. Das Erstellen komprimierter Backup Sets benötigt jedoch mehr Zeit als die Erstellung normaler Backup Sets.

Kompression kann für Image Copies nicht angewendet werden.



Listing 30.7: Ein komprimiertes Backup Set der Datenbank erstellen

```
RMAN> BACKUP AS COMPRESSED BACKUPSET database;
```

Den Speicherort des Backups angeben

Mit Hilfe der **FORMAT**-Klausel des **BACKUP**-Kommandos kann angegeben werden, wo das Backup gespeichert werden soll. Standardmäßig werden Backups in der Fast Recovery Area gespeichert.

Listing 30.8: Den Speicherort des Backups ändern

```
RMAN> BACKUP database FORMAT = '/u04/backup/%d_%D-%M-%Y_%t_%s_%p.bkp';
```

Der folgenden Literaturhinweis erläutert die Bedeutung der einzelnen Platzhalter.



- [formatSpec].

Backups mit Tags versehen

RMAN hängt an jedes Backup einen kurzen Kommentar an, der als „Tag“¹ bezeichnet wird und zur besseren Identifizierung dient. Um manuell ein Tag an ein Backup anzuhängen, gibt es die **TAG**-Klausel im **BACKUP**-Kommando.

Listing 30.9: Ein Backup mit manuellem Tag versehen

```
RMAN> BACKUP database TAG = 'Full-Backup 28.10.13';
```

30.1.3 Full-Backups

Unter einem Full-Backup (Vollsicherung) wird die vollständige Sicherung

- einer Datendatei,
- eines Tablespaces,
- oder einer Datenbank verstanden.

Das Gegenstück zur Vollsicherung ist die inkrementelle Sicherung.

¹tag = engl. Schild

Backup einer Datenbank

Beispiel ?? zeigt die einfachste Variante eines kompletten Backups einer Datenbank.

Listing 30.10: Backup einer ganzen Datenbank

```
RMAN> BACKUP database;
RMAN> SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
```

Das Kommando `BACKUP database` fertigt das Backup der Datenbank an. Im zweiten Schritt werden mit dem Kommando `ALTER SYSTEM ARCHIVE LOG CURRENT` die aktiven Redo Logs gewechselt (Log Switch) und archiviert. Somit wird sichergestellt, dass alle für das Recovery mit diesem Backup benötigten Redo Logs archiviert wurden. Wird dieses Backup als Hotbackup durchgeführt, sollten anschliessend noch die Archive Logs gesichert werden.

Backups einzelner Tablespace

Um ein Backup eines oder mehrerer Tablespace durchzuführen, gibt es das Kommando `BACKUP tablespace`. Im folgenden Beispiel werden die beiden Tablespace `USERS` und `BANK` auf ein SBT-Gerät gesichert.

Listing 30.11: Backup eines Tablespace

```
RMAN> BACKUP DEVICE TYPE sbt tablespace users, bank;
RMAN> SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
```

Backups einzelner Datendateien

Um einzelne Datendateien sichern zu können, gibt es das Kommando `BACKUP datafile`. Wenn beispielsweise die Datendateien mit den Nummern 1, 2 und 4 gesichert werden sollen, könnte dies wie folgt geschehen:

Listing 30.12: Backup einzelner Datendateien

```
RMAN> BACKUP datafile 1, 2, 4;
```



Ist das Controlfile Autobackup aktiviert, wird anschliessend an dieses Backup, automatisch ein Backup Set mit der aktuellen Kontrolldatei und der Serverparameterdatei angelegt. Ist das Controlfile Autobackup deaktiviert, werden die aktuelle Kontrolldatei und die Serverparameterdatei automatisch in das Backupset der Datendateien integriert, da Tablespace Nummer 1 gesichert wurde.

Um die Nummern der Datendateien herauszufinden kann die View DBA_DATA_FILES wie folgt benutzt werden:

Listing 30.13: Herausfinden der Datendateinummern

```
SQL> SELECT file_id, file_name, tablespace_name
  2  FROM dba_data_files
  3  ORDER BY tablespace_name, file_id;
```

30.1.4 Inkrementelle Backups

Führt der RMAN ein inkrementelles Backup durch, sichert er nur die Datenblöcke, die sich seit einem bestimmten Backup, welches als Referenz angegeben wurde, geändert haben. Inkrementelle Backups können sowohl von ganzen Datenbanken als auch von Tablespace und Datendateien durchgeführt werden.

Es gibt verschiedene Begründungen, warum inkrementelle Backups in eine Backup-Strategie mit einbezogen werden sollten:

- Reduzieren des Zeitaufwands für tägliche Backups
- Netzwerkbandbreite durch kleinere Datenmengen einsparen

Wie funktionieren inkrementelle Backups?

Jeder Datenblock in einer Oracle Datenbank führt in seinem Header eine System Change Number (SCN). Dies ist immer die SCN, die während der letzten Veränderung am Datenblock aktuell war. Während eines inkrementellen Backups liest RMAN die SCN jedes Datenblocks und vergleicht diese mit der SCN des Referenzbackups. Ist die SCN im Datenblock größer, als die SCN im Referenzbackup, wird der Datenblock gesichert.

Inkrementelle Backups können als Level 0 oder als Level 1 Backups angelegt werden. Ein Level 0 Backup sichert alle Datenblöcke genau wie ein normales Full-Backup. Der einzige Unterschied ist, dass ein normales Full-Backup nicht in eine inkrementelle Backup-Strategie integriert werden kann.

Für Level 1 Backups gibt es zwei verschiedene Arten:

- **Inkrementelles Backup:** Es werden alle Datenblöcke gesichert, die sich seit dem letzten Level 1 oder Level 0 Backup geändert haben.

- **Kumulatives Backup:** Es werden alle Datenblöcke gesichert, die sich seit dem letzten Level 0 Backup geändert haben.

Inkrementelle Backups sparen Speicherplatz und Zeit während des Backupvorgangs, da unter Umständen weniger Datenblöcke gesichert werden müssen, als bei einem kumulativen Backup. 

Der Vorteil der kumulativen Backups liegt darin, dass die Recovery-Zeit geringer ist, als bei den Inkrementellen. Bei inkrementellen Backups müssen alle Backups, seit dem letzten Level 0 Backup in der richtigen Reihenfolge wiederhergestellt werden. Bei kumulativen Backups muss nur das aktuellste Backup seit dem letzten Level 0 Backup aufgespielt werden.

Level 0 Backups durchführen

Die Grundlage einer jeden inkrementellen Backupstrategie sind Level 0 Backups. Egal welche Art von Level 1 Backups gemacht wird, die Art und Weise wie Level 0 Backups erstellt werden, bleibt davon unberührt. Beispiel ?? zeigt verschiedene Varianten eines Level 0 Backups.

Listing 30.14: Inkrementelles Level 0 Backup

```
-- Level 0 Backup der Datenbank
RMAN> BACKUP INCREMENTAL LEVEL 0 database;
-- Level 0 Backup eines Tablespaces
RMAN> BACKUP INCREMENTAL LEVEL 0 tablespace bank;
-- Level 0 Backup einer Datendatei
RMAN> BACKUP INCREMENTAL LEVEL 0 datafile 1;
```

Level 1 Backups durchführen

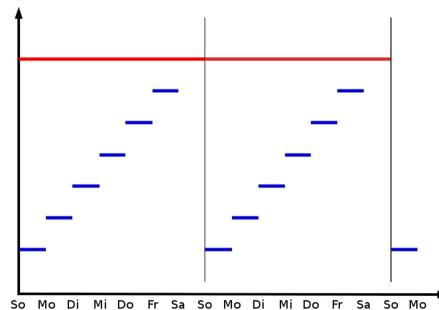
Zur Durchführung eines Level 1 Backups wird lediglich die Klausel `LEVEL 0` durch `LEVEL 1` ersetzt. Hierbei handelt es sich dann um ein inkrementelles Backup.

Listing 30.15: Inkrementelles Level 1 Backup der Datenbank

```
RMAN> BACKUP INCREMENTAL LEVEL 1 database;
```

Abbildung ?? zeigt ein Beispiel, in dem an jedem Sonntag ein inkrementelles Level 0 Backup durchgeführt wird. Dieses dient dann als Referenzbackup für das Level 1 Backup, welches am darauf folgenden Montag erstellt wird. Das Level 1 Backup vom Montag gilt dann wiederum als Referenzbackup für das Level 1 Backup am Dienstag, bis am nächsten Sonntag wieder ein Level 0 Backup erstellt wird und die Kette von vorne beginnt.

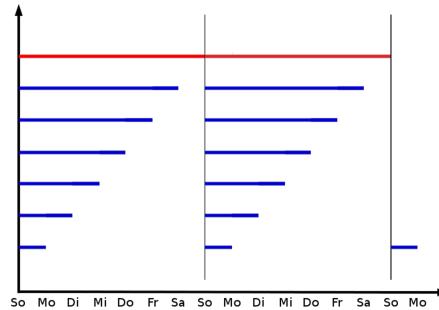
Abb. 30.1:
Eine
inkrementelle
Backupstrategie



Kumulative Backups durchführen

Um kumulative Backups durchzuführen, muss zusätzlich zur `INCREMENTAL LEVEL N`-Klausel noch das Schlüsselwort `CUMULATIVE` angegeben werden.

Abb. 30.2:
Eine kumulative
Backupstrategie



Listing 30.16: Kumulatives Level 1 Backup der Datenbank

```
RMAN> BACKUP INCREMENTAL LEVEL 1 CUMULATIVE database;
```

30.1.5 Backups der Archive Logs durchführen

Separate Sicherung der Archive Logs

Archive Logs sind der Schlüssel zu einer erfolgreichen Backup und Recoverystrategie, weshalb sie regelmäßig gesichert werden sollten. Ähnlich wie Kontrolldateien können Archive Logs als eigenes Backup Set gesichert oder in einem anderen Backup Set mitgesichert werden. Um ein manuelles Backup der Archive Logs durchzuführen gibt es die `ARCHIVELOG-ALL`-Klausel des `BACKUP`-Kommandos.

Listing 30.17: Manuelles Backup aller Archive Logs

```
RMAN> BACKUP archivelog ALL;
```

Anders als bei den Datendateien, muss bei der Sicherung der Archive Logs immer angegeben werden, welche Archive Logs zu sichern sind. In Beispiel ?? wird das Schlüsselwort `ALL` benutzt, um alle Archive Logs zu sichern. Es gibt jedoch noch andere Möglichkeiten, um die Menge der zu sichernden Archive Logs einzuschränken.

Listing 30.18: Alle Archive Logs ab Sequenz Nummer 4711 sichern

```
RMAN> BACKUP archivelog  
2>   FROM SEQUENCE 4711;
```

Listing 30.19: Alle Archive Logs bis Sequenz Nummer 4711 sichern

```
RMAN> BACKUP archivelog  
2>   UNTIL SEQUENCE 4711;
```

Listing 30.20: Alle Archive Logs zwischen Sequenz Nummer 666 und 4711 sichern

```
RMAN> BACKUP archivelog  
2>   SEQUENCE BETWEEN 666 AND 4711;
```

Wie in ?? bereits erwähnt, sollte aus Sicherheitsgründen die Archivierung der Log Dateien immer an mehreren Speicherorten erfolgen. Wenn nun mehrere identische Kopien einer Archive Log Datei existieren, wird der RMAN immer nur die erste verfügbare Kopie sichern. Archive Logs gelten als identische wenn folgende Attribute gleich sind:

- die Datenbank ID (DBID),
- die Log Thread Number (Laufende Nummer des LGWr-Prozesses, nur in RAC-Umgebungen relevant)
- die Log Sequence Number
- die Reset Logs SCN

Archive Logs mitsichern

Eine andere Variante ist die Archive Logs beim Backup der Datenbank oder eines Teils der Datenbank mitzusichern. Dies geschieht durch die Klausel `PLUS archivelog`, die dem `BACKUP`-Kommando angehängt werden kann.

Listing 30.21: Manuelles Backup des Archive Logs

```
RMAN> BACKUP database PLUS archivelog;
```

Nach dem Absetzen dieses Kommandos führt der RMAN folgende Schritte durch:

1. Es wird ein Log Switch herbeigeführt.
2. RMAN führt das Kommando `BACKUP archivelog ALL` aus.
3. Das Backup der Datenbank wird angelegt.
4. Es wird erneut ein Log Switch herbeigeführt
5. Das Kommando `BACKUP archivelog ALL` wird ein zweites mal ausgeführt.

Damit wird sichergestellt, dass alle Archive Logs, die vor und während des Backups entstanden sind mitgesichert werden.

Gesicherte Archive Logs automatisch löschen

RMAN bietet die Möglichkeit, Archive Logs automatisch, nach ihrer Sicherung zu löschen. Dies erspart ein manuelles Eingreifen des Administrators. Um die Archive Logs automatisch zu löschen, muss eine der beiden Klauseln `DELETE INPUT` oder `DELETE ALL INPUT` an das `BACKUP ARCHIVELOG`-Kommando angehängt werden. Der Unterschied zwischen diesen Klauseln liegt darin, welche Archive Logs gelöscht werden:

- `DELETE INPUT`: Es werden nur die Kopien einer Archive Log Datei gelöscht, die vom RMAN zur Sicherung benutzt wurden. Alle anderen Kopien bleiben erhalten.
- `DELETE ALL INPUT`: Es werden alle Kopien einer Archive Log Datei gelöscht, auch die, welche nicht zur Sicherung benutzt wurden.

Beispiel ?? zeigt die Anwendung der `DELETE ALL INPUT`-Klausel.

Listing 30.22: Manuelles Backup und gleichzeitiges Löschen der Archive Logs

```
RMAN> BACKUP archivelog ALL  
2>     DELETE ALL INPUT;
```

30.1.6 Backups der Fast Recovery Area anfertigen

In verschiedenen Fällen kann es sinnvoll sein, den Inhalt der Fast Recovery Area auf Band zu sichern. Eine einfache Vorgehensweise beim Sichern der Fast Recovery Area könnte sein:

Abb. 30.3:
Automatisches
löschen der
Archive Logs

DELETE INPUT



DELETE ALL INPUT



1. Festlegen einer passenden Backup Retention Policy
2. Backup der Datenbank durchführen
3. Sichern der Fast Recovery Area auf Band
4. Obsolete Backups der Fast Recovery Area von den Bändern löschen
5. Obsolete Datenbankbackups aus der Fast Recovery Area löschen

Listing 30.23: Sichern der Fast Recovery Area

```
RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 2;

RMAN> RUN {
2>   ALLOCATE CHANNEL c_disk
3>   DEVICE TYPE disk;
4>   ALLOCATE CHANNEL c_sbt
5>   DEVICE TYPE sbt
6>   PARMS  'SBT_LIBRARY=oracle.disksbt ,ENV=(BACKUP_DIR=/u04)';
7>
8>   BACKUP CHANNEL c_disk database;
9>   BACKUP CHANNEL c_sbt recovery area;
10> }

RMAN> DELETE obsolete;
```



Der Inhalt der Fast Recovery Area kann mittels `BACKUP RECOVERY AREA` auf SBT gesichert werden. Eine Sicherung auf Festplatte ist nicht möglich.

30.1.7 Manuelle Kontrolldatei-Backups

Backup mit dem RMAN

RMAN benutzt ein „Snapshot Controlfile“, um konsistente Backups der Kontrolldatei anzufertigen. Ist das Controlfile Autobackup-Feature des RMAN aktiviert, macht RMAN automatisch Backups der Kontrolldatei und der Serverparameterdatei nach jedem Backup und jeder Strukturänderung der Datenbank. Ist dieses Feature nicht aktiviert, sollten regelmässig manuelle Backups der Kontrolldatei erfolgen. Dies kann auf unterschiedliche Arten geschehen:

- Durch das RMAN-Kommando `BACKUP current controlfile`

- In dem eine Kopie der Kontrolldatei in das aktuelle Backup aufgenommen wird
- Durch das Sichern von Datendatei Nummer 1, da dabei die Kontrolldatei immer automatisch mitgesichert wird. Dies geschieht zum Beispiel im Rahmen eines Datenbank-Backups.

Damit die Kontrolldatei bei einem beliebigen Backup mitgesichert werden kann, gibt es die Klausel `INCLUDE current controlfile`.

Listing 30.24: Kontrolldatei in ein Backup mit einschließen

```
RMAN> BACKUP tablespace bank
2>   INCLUDE current controlfile;
```

Backup to Trace

Oracle stellt eine zweite Variante zur Sicherung des Controlfiles zur Verfügung: Das Backup to Trace. Hierbei wird keine Binär-Kopie, sondern ein SQL-Skript erstellt, mit dessen Hilfe die Kontrolldatei neu kreiert werden kann. Erzeugt wird dieses Skript mit Hilfe des SQL-Kommandos `ALTER DATABASE BACKUP`.

Listing 30.25: Kontrolldatei in ein Backup mit einschließen

```
SQL> ALTER DATABASE
2  BACKUP CONTROLFILE TO TRACE
3  AS '/home/oracle/control.bkp';
```

Und so sieht es aus:

Listing 30.26: Das `CREATE CONTROLFILE`-Skript

```
startup nomount
CREATE CONTROLFILE REUSE DATABASE "ORCL" NORESETLOGS ARCHIVELOG
  MAXLOGFILES 16
  MAXLOGMEMBERS 3
  MAXDATAFILES 100
  MAXINSTANCES 8
  MAXLOGHISTORY 292
LOGFILE
  GROUP 1 '/u01/app/oracle/oradata/orcl redo01.log', SIZE 50M BLOCKSIZE 512,
  GROUP 2 '/u01/app/oracle/oradata/orcl redo02.log', SIZE 50M BLOCKSIZE 512,
  GROUP 3 '/u01/app/oracle/oradata/orcl redo03a.log', SIZE 50M BLOCKSIZE 512,
  GROUP 4 '/u01/app/oracle/oradata/orcl redo04a.log', SIZE 50M BLOCKSIZE 512,
  GROUP 5 (
    '/u01/app/oracle/oradata/orcl redo05a.log',
    '/u02/oradata/orcl redo05b.log'
  ) SIZE 50M BLOCKSIZE 512

DATAFILE
  '/u01/app/oracle/oradata/orcl/system01.dbf',
```

```
'/u01/app/oracle/oradata/orcl/sysaux01.dbf',
'/u01/app/oracle/oradata/orcl/undotbs01.dbf',
'/u01/app/oracle/oradata/orcl/users01.dbf',
'/u01/app/oracle/oradata/orcl/example01.dbf',
'/u01/app/oracle/oradata/orcl/bank01.dbf'
CHARACTER SET WE8MSWIN1252;
```

Beispiel ?? zeigt nur einen Ausschnitt aus dem `CREATE CONTROLFILE`-Skript.

Interessant ist ein solches Skript immer dann, wenn Probleme mit der Datenbank auftreten, die sich auf dem „offiziellen“ Weg nicht lösen lassen. Dies ist unter anderem dann der Fall, wenn alle Dateien der Current Redo Log Group beschädigt sind.

30.1.8 Backup des SPFile

Das SPFile wird in verschiedenen Situationen automatisch mitgesichert (siehe ??). Um ein manuelles Backup des SPFile durchzuführen gibt es das folgende Kommando:

Listing 30.27: Manuelles Backup des SPFile

```
RMAN> BACKUP spfile;
```

30.1.9 Backupduplexing konfigurieren

Es ist möglich RMAN so zu konfigurieren, dass von jedem Backuppiece mehrere identische Kopien angefertigt werden. Dieses Features ist als „duplexing“ bekannt und bezieht sich nur auf Backup Sets, nicht aber auf Image Copies.

Listing 30.28: Konfigurieren des Backupduplexing

```
RMAN> CONFIGURE DATAFILE BACKUP COPIES
2>      FOR DEVICE TYPE disk TO 2;
RMAN> CONFIGURE ARCHIVELOG BACKUP COPIES
2>      FOR DEVICE TYPE sbt TO 2;

RMAN> BACKUP database PLUS ARCHIVELOG
2>      FORMAT '/u02/backups/%U', '/u03/backups/%U';
```

Beispiel ?? zeigt, wie sowohl für Datendateien, als auch für Archive Log Dateien, das Backup Duplexing auf den Wert zwei konfiguriert wird. RMAN erstellt zwei identische Kopien der Datendateien



und verteilt diese auf die beiden Speicherorte /u02/backups und /u03/backups. Ein solcher Formatstring kann mit Hilfe eines der Kommandos `BACKUP`, `CONFIGURE CHANNEL` oder `ALLOCATE CHANNEL` angegeben werden.

Zu beachten ist dabei, dass das Backup Duplexing nicht zusammen mit der Fast Recovery Area funktioniert. Dies wird durch die folgende Fehlermeldung angezeigt:

```
ORA-19806: cannot make duplex backups in recovery area.
```

Aus diesem Grund muss immer ein Format-String verwendet werden, der die Duplexkopien ausserhalb der Fast Recovery Area speichert. Es können maximal 4 identische Kopien eines Backups erzeugt werden.

30.2 Backups verwalten

30.2.1 Backups katalogisieren

In bestimmten Situationen kann es notwendig sein, dass Image Copies, Backupsets oder Archive Logs im RMAN Repository oder im RMAN Katalog neu erfasst werden müssen. Ein solcher Fall tritt ein, wenn:

- Kopien von Datendateien ohne Zuhilfenahme des RMAN angefertigt wurden. Diese können dann als Datafile Copies im Repository/Katalog registriert werden.
- Backup Pieces ohne RMAN auf dem Datenträger verschoben wurden.
- Backup Pieces aus dem Repository gelöscht wurden, die Dateien selbst aber noch existieren und wiederverwendet werden sollen.
- die Kontrolldatei verloren geht und kein Recovery Katalog genutzt wird.

Listing 30.29: Backup Pieces katalogisieren

```
RMAN> CATALOG BACKUPPIECE  '/u03/backup/backup_820.bkp',
2>                      '/u04/backup/backup_821.bkp';
```

In Beispiel ?? wird RMAN versuchen beide Backup Pieces zu katalogisieren. Er wird seine Arbeit auch dann fortsetzen, wenn eines der beiden Pieces defekt ist. Es wird dann nur das Funktionsfähige ins

Repository aufgenommen. Soll ein aus mehreren Pieces bestehendes Backup Set katalogisiert werden, gelingt dies nur, wenn alle Pieces fehlerfrei sind.

Listing 30.30: Datendatei-Kopien katalogisieren

```
RMAN> CATALOG DATAFILECOPY '/u03/backup/bank01.dbf';
```

Listing 30.31: Archive Logs katalogisieren

```
RMAN> CATALOG ARCHIVELOG '/u03/backup/archive1_731.dbf',  
2> '/u03/backup/archive1_732.dbf';
```

Es ist möglich, den Inhalt eines gesamten Verzeichnisses, in einem Arbeitsschritt zu katalogisieren.

Listing 30.32: Verzeichnisinhalt katalogisieren

```
RMAN> CATALOG START WITH '/disk1/backups/';
```

Bei dieser Art des Katalogisierens, muss beachtet werden, dass es sich bei der Angabe von `/disk1/backups/` nicht um einen Verzeichnisnamen handelt, sondern nur um ein Prefix. Wird `/disk1/backups` angegeben, werden alle Verzeichnisse, die mit dieser Zeichenkette beginnen, z. B. `/disk1/backupssets` oder `/disk1/backups-jahr-2013` ebenfalls katalogisiert. Dies kann dazu führen, dass unbeabsichtigt Dateien in den Recovery Katalog aufgenommen werden.



Um solche Probleme zu vermeiden, sollte das Prefix immer mit einem / abgeschlossen werden, also z. B. `/disk1/backups/`.

30.2.2 Backup-Crosschecks durchführen

Ein Crosscheck vergleicht den Recovery Katalog mit dem Inhalt des Dateisystems, um Unterschiede festzustellen. Wenn z. B. ein Nutzer ein Backup Set mit Betriebssystemmitteln von der Festplatte löscht, existiert trotzdem noch der Eintrag für dieses Backup Set im Recovery Katalog. Mit Hilfe des `CROSSCHECK`-Kommandos können solche Inkonsistenzen bereinigt werden. Nach einem Crosscheck bekommt jeder Eintrag im RMAN einen Status:

- **EXPIRED:** Das Backup ist nicht auf dem Dateisystem verfügbar.
- **AVAILABLE:** Das Backup ist verfügbar und darf genutzt werden.
- **UNAVAILABLE:** Das Backup ist verfügbar, darf aber nicht durch den RMAN genutzt werden.



Der Status expired sollte nicht mit dem Status obsolete verwechselt werden.

Die Ergebnisse eines Crosschecks können direkt im RMAN, mit Hilfe des [LIST](#)-Kommandos oder in SQL*PLUS, mit Hilfe der View V\$BACKUP_FILES betrachtet werden. Während eines Crosschecks werden keine Dateien von der Festplatte und auch keine RMAN-Einträge gelöscht. Lediglich der Status eines Eintrags wird verändert. Um Backups zu löschen, muss das [DELETE](#)-Kommando des RMAN verwendet werden (siehe [??](#)). Einfache Beispiele für die Nutzung des [CROSSCHECK](#)-Kommandos könnten so aussehen:

Listing 30.33: [CROSSCHECK](#) aller Backups

```
RMAN> CROSSCHECK backup;
```

Bei der Angabe von [BACKUP](#) werden alle Arten von Backups überprüft. Es gibt verschiedene Möglichkeiten, um die Menge der zu überprüfenden Backups einzuschränken.

Listing 30.34: **CROSSCHECK** auf Backup Sets beschränken

```
RMAN> CROSSCHECK backupset;

RMAN> CROSSCHECK backupset 666, 815, 4711;

RMAN> CROSSCHECK backupset TAG = 'nightly_backup';
```

Das erste Kommando prüft alle vorhandenen Backup Sets. Das zweite nur die Backup Sets mit den IDs 666, 815 und 4711. Das dritte **CROSSCHECK**-Kommando checkt nur das Backup Piece mit dem Tag „nightly_backup“.

Listing 30.35: Nur Datafile Copies checken

```
RMAN> CROSSCHECK datafilecopy ALL;

RMAN> CROSSCHECK datafilecopy 113, 114, 115;
```

Wird der Crosscheck auf Datafilecopies, Archive Logs oder Controlfilecopies angewendet, muss in jedem Fall die Menge der zu prüfenden Dateien angegeben werden.

Listing 30.36: Alle Arten von Copies (Datafilecopy, Archive Logs, Controlfilecopy) checken

```
RMAN> CROSSCHECK copy;
```

30.2.3 Manuelle Statusänderungen an Backups

Mit Hilfe des **CHANGE**-Kommandos kann der Status eines Backups manuell auf „available“ oder „unavailable“ gesetzt werden. Da der RMAN sich selbst die für eine Wiederherstellung notwendigen Backups sucht, ist es immer dann sinnvoll, ein Backup Set als „unavailable“ zu markieren, wenn der RMAN es nicht für das Restore benutzen darf.



Zu beachten ist, dass Dateien die sich in der Fast Recovery Area befinden, nicht als *unavailable* markiert werden können.

Listing 30.37: Manuelle Statusänderung eines Backup Sets auf unavailable

```
RMAN> CHANGE BACKUPSET 12 UNAVAILABLE;
```

Listing 30.38: Das Backup Set wieder verfügbar machen

```
RMAN> CHANGE BACKUPSET 12 AVAILABLE;
```



Manuelle Statusänderungen sind auch für Datafilecopies, Controlfilecopies und alle anderen Arten von Backups möglich.

30.2.4 Backups löschen

Bestimmte Backups löschen

In regelmäßigen Abständen ist es notwendig, Backups zu löschen und Platz für Neue zu schaffen. Diese Arbeit sollte immer mit dem RMAN-Kommando `DELETE` verrichtet werden, da nicht nur die Dateien gelöscht, sondern auch das RMAN Repository gepflegt werden muss. Es bedient sich grundsätzlich der gleichen Syntax, wie der `CROSSCHECK`-Befehl.

Listing 30.39: Alle vorhandenen Backups löschen

```
RMAN> DELETE backup;
```

Listing 30.40: Nur Backup Sets löschen

```
RMAN> DELETE backupset;  
RMAN> DELETE backupset 666, 815, 4711;
```

`DELETE backupset` löscht alle Backup Sets, während durch die Angabe von Backup Set IDs nur bestimmte Backup Sets gelöscht werden.

Listing 30.41: Archive Logs löschen

```
RMAN> DELETE ARCHIVELOG ALL;  
  
RMAN> DELETE ARCHIVELOG  
2>     UNTIL SEQUENCE = 200;
```

Gerade beim Löschen von Archive Logs ist größte Vorsicht geboten!



RMAN fragt vor dem Löschen jeder Datei nach. Diese Nachfrage kann durch die Angabe von `DELETE NOPROMPT` unterbunden werden.

Obsolete Backups löschen

Die Backup Retention Policy legt fest, welche Backups für ein Recovery benötigt werden und welche nicht (siehe ??). Verstößt ein Backup gegen die Retention Policy, wird es als „obsolete“ markiert. Obsolete Backups können mittels `DELETE`-Befehl gelöscht werden.

Listing 30.42: Obsolete Backups löschen

```
RMAN> DELETE obsolete;
```

Expired-Backups löschen

Wird ein Crosscheck ausgeführt, um den Inhalt des Recovery Katalogs mit dem Dateisystem zu vergleichen, werden Backups die zwar noch im Katalog eingetragen sind, aber auf dem Dateisystem nicht mehr existieren auf den Status *expired* gesetzt. Mit dem Kommando `DELETE EXPIRED` können solche Einträge anschliessend entfernt werden. Sollte die zu dem Eintrag gehörende Datei noch existieren, wird sie ebenfalls gelöscht.

Folgender Vorgang löscht alle als *expired* markierten Backups:

1. Durchführen eines Crosschecks.

Listing 30.43: CROSSCHECK durchführen

```
CROSSCHECK backup;
```

2. Löschen der als *expired* markierten Backups.

Listing 30.44: Löschen der Backups

```
DELETE EXPIRED backup;
```

30.3 Block Change Tracking

Das „Block Change Tracking“-Feature des RMAN erhöht die Performance inkrementeller Backups dadurch, dass Informationen über geänderte Blöcke in einem „Trackingfile“ gespeichert werden. Wenn RMAN das Trackingfile benutzt, entfällt das Durchsuchen der Datenbank nach geänderten Blöcken.

Das erste Level 0 Backup, nach dem Einschalten des Block Change Tracking, zieht noch keinen Nutzen daraus, da das Trackingfile noch nicht den aktuellen Stand in der Datenbank wiedergeben kann. Aber bereits das nächste Level 1 Backup, kann die Informationen im Trackingfile nutzen.



Durch die Nutzung von Block Change Tracking ändert sich nichts an den Backup- oder Recovery-Kommandos.

Standardmäßig ist das Block Change Tracking deaktiviert, da es geringe Performanceeinbussen, im normalen Betrieb der Datenbank mit sich bringt. Der Performancegewinn bei der Durchführung inkrementeller Backups, mit großen Datenmengen, ist jedoch sehr hoch.

Das Trackingfile kann an einem beliebigen Ort auf dem Datenträger gespeichert werden. Oracle empfiehlt es in der Fast Recovery Area (Standardspeicherort) abzulegen.



Im Trackingfile werden für bis zu acht inkrementelle Backups (ein Level 0 plus sieben Level 1) die Trackinginformationen gespeichert. Sobald ein weiteres Level 1 Backup angelegt wird, werden die Informationen zum ersten Backup (Level 0) überschrieben.

Um dieses Feature zu aktivieren, muss ein Trackingfile angegeben werden. Dies kann nur geschehen, wenn die Datenbank geöffnet oder in der Mount-Phase ist.

Listing 30.45: Block Change Tracking aktivieren

```
SQL> ALTER DATABASE
  2  ENABLE BLOCK CHANGE TRACKING
  3  USING FILE '/u02/rman_change_track.f' REUSE;
```

Die **USING FILE**-Klausel gibt den Pfad und den Dateinamen des Trackingfiles an. Das Schlüsselwort **REUSE** sorgt dafür, dass eine evtl. bereits bestehende Datei überschrieben wird.

Verwenden Sie die Klausel **DISABLE BLOCK CHANGE TRACKING**, um das Block Change Tracking zu deaktivieren.

Listing 30.46: Block Change Tracking deaktivieren

```
SQL> ALTER DATABASE
  2  DISABLE BLOCK CHANGE TRACKING;
```

Mit Hilfe der View V\$BLOCK_CHANGE_TRACKING kann man ersehen, ob das Block Change Tracking aktiviert ist. Sie enthält eine Spalte STATUS, die den Zustand des Block Change Trackings angibt.

Listing 30.47: Den Status des Block Change Trackings überprüfen

```
SQL> col status format a8
SQL> col filename format a30
SQL> col bytes format 999,999.00
```



```
SQL> SELECT *
  2  FROM v$block_change_tracking;

STATUS      FILENAME                      BYTES
-----  -----
DISABLED
```

- [BRADV89537]

30.4 Das Kommando LIST

Das Kommando `LIST` benutzt das RMAN Repository, um verschiedene Informationen über Backups und Archive Logs anzuzeigen. Es ist nur dann funktionsfähig, wenn eine Verbindung zur Zieldatenbank besteht.

30.4.1 Auflistungen gruppieren

`LIST` kann Backups auf zwei Arten anzeigen:

- Gruppiert nach Backup: Zu jedem Backup werden die zugehörigen Dateien angezeigt.
- Gruppiert nach Datei: Zu jeder Datei wird das betreffende Backup angezeigt.

Nach Backup gruppieren

Listing 30.48: `LIST` - Alle Backups, gruppiert nach Backup anzeigen

```
RMAN> LIST backup;

using target database control file instead of recovery catalog

List of Backup Sets
=====

BS Key  Type LV Size      Device Type   Elapsed Time Completion Time
-----  --  --  --      -----  --       -----  -----
1       Full  591.36M    DISK          00:01:00   24-OCT-13
      BP Key: 1    Status: AVAILABLE  Compressed: NO    Tag: TAG20131024T141243
```

```

Piece Name: /u05/fast_recovery_area/ORCL/backupset/201...
List of Datafiles in backup set 1
File LV Type Ckp SCN      Ckp Time   Name
-----
1       Full  2054849    24-OCT-13  /u01/app/oracle/oradata/orcl/system01.dbf

BS Key  Type LV Size      Device Type Elapsed Time Completion Time
-----
2       Full   9.36M    DISK          00:00:04   24-OCT-13
BP Key: 2   Status: AVAILABLE  Compressed: NO  Tag: TAG20131024T141243
Piece Name: /u05/fast_recovery_area/ORCL/backupset/201...
SPFILE Included: Modification time: 23-OCT-13
SPFILE db_unique_name: ORCL
Control File Included: Ckp SCN: 2054873      Ckp time: 24-OCT-13

```

Listing 30.49: LIST - Bestimmte Backup Sets, gruppiert nach Backup anzeigen

```
RMAN> LIST backupset 666, 815, 4711;
```

Nach Datei gruppieren

Um in umgekehrter Folge zu Gruppieren, muss dem LIST-Kommando der Zusatz BY FILE mitgegeben werden.

Listing 30.50: LIST - Backups, gruppiert nach Backup Datei anzeigen

```
RMAN> LIST backup BY FILE;

List of Datafile Backups
=====

File Key     TY LV S Ckp SCN      Ckp Time   #Pieces #Copies Compressed Tag
----- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
1   1       B  F  A 2054849    24-OCT-13  1        1      NO    TAG2013...
2   7       B  F  A 2076573    27-OCT-13  1        1      NO    TAG2013...
6   5       B  F  A 2062383    25-OCT-13  1        1      NO    TAG2013...
      3       B  F  A 2054959    24-OCT-13  1        1      NO    TAG2013...
7   3       B  F  A 2054959    24-OCT-13  1        1      NO    TAG2013...

List of Control File Backups
=====

CF Ckp SCN Ckp Time   BS Key   S #Pieces #Copies Compressed Tag
----- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
2076573    27-OCT-13 8      A 1        1      NO    TAG20131028T093159
2062392    25-OCT-13 6      A 1        1      NO    TAG20131025T112454
2054967    24-OCT-13 4      A 1        1      NO    TAG20131024T141608
2054873    24-OCT-13 2      A 1        1      NO    TAG20131024T141243
```

List of SPFILE Backups								
=====								
Modification	Time	BS	Key	S	#Pieces	#Copies	Compressed	Tag
27-OCT-13		8	A	1	1	1	NO	TAG20131028T093159
23-OCT-13		6	A	1	1	1	NO	TAG20131025T112454
23-OCT-13		4	A	1	1	1	NO	TAG20131024T141608
23-OCT-13		2	A	1	1	1	NO	TAG20131024T141243

Anzeigen einer Zusammenfassung

Mit `LIST ... SUMMARY` kann für jeden beliebigen Backuptyp eine Zusammenfassung angezeigt werden. Diese enthält nur noch die wesentlichsten Informationen, in geraffter Form.

Listing 30.51: Anzeigen einer Zusammenfassung

30.4.2 Auflisten der Expired Backups

Es ist möglich sich solche Backups anzeigen zu lassen, die im Repository als „expired“ markiert sind.

Listing 30.52: Expired Backups auflisten

```

RMAN> LIST expired backup;

List of change, crosscheck, and delete Backup Sets
=====
BS Key  Size      Device Type Elapsed Time Completion Time
-----  -----  -----
7       136M     disk      00:00:20   04-NOV-12
        BP Key: 7    Status: available  Compressed: NO  Tag: TAG2012...
        Piece Name: /u05/fast_recovery_area/backup/2003_11_04/...
List of Archived Logs in backup set 7
Thrd Seq      Low scn    Low Time   Next scn   Next Time
-----  -----  -----
1     1        173832   21-OCT-12  174750    21-OCT-03
1     2        174750   21-OCT-12  174755    21-OCT-03
1     3        174755   21-OCT-12  174758    21-OCT-03
1     37       533321   01-NOV-12  575472    03-NOV-03
1     38       575472   03-NOV-12  617944    04-NOV-03
1     39       617944   04-NOV-12  631495    04-NOV-03
BS Key  Type LV Size      Device Type Elapsed Time Completion Time
-----  -----  -----  -----
8       Full  2M      disk      00:00:01   04-NOV-12
        BP Key: 8    Status: available  Compressed: NO  Tag: TAG2012...
        Piece Name: /u05/fast_recovery_area/c-774627068-20121104-01
Controlfile Included: Ckp scn: 631510          Ckp time: 04-NOV-12

```

```
spfile Included: Modification time: 21-OCT-12
```

30.4.3 Ausgewählte Backups anzeigen

Anbei noch einige Beispiele, die zeigen, wie verschiedenste Backups zur Auflistung ausgewählt werden können.

Listing 30.53: Auflistungen eingrenzen

```
# Nur Full-Backups der Datenbank anzeigen
RMAN> LIST backup OF database;

# Image Copies einer bestimmten Datendatei anzeigen
RMAN> LIST copy OF datafile '/u02/oradata/ORCL/system01.dbf';

# Ein bestimmtes Backup Set anzeigen
RMAN> LIST backupset 213;

# Ein Backup Set nach seinem Tag aussuchen
RMAN> LIST backupset TAG 'weekly_full_db_backup';

# Alle Image Copies einer Datendatei, die auf einem sbt-Gerät liegen anzeigen
RMAN> LIST copy OF datafile '/u02/oradata/ORCL/system01.dbf'
2>   DEVICE TYPE sbt;

# Alle Backups anzeigen, die in einem bestimmten Verzeichnis liegen
RMAN> LIST backup LIKE '/tmp/%';

# Image Copies anzeigen, die in einem bestimmten Zeitraum fertiggestellt wurden
RMAN> LIST copy OF datafile 2 COMPLETED BETWEEN '10-DEC-2002' AND '17-DEC-2002';

# Alle Archive Logs, die mindestens zweimal auf Band gesichert wurden anzeigen
RMAN> LIST archivelog ALL BACKED UP 2 TIMES TO DEVICE TYPE sbt;

# Auflisten eines bestimmten Backupsets nach Tag
RMAN> LIST backupset TAG 'weekly_full_db_backup';
```

30.5 Das Kommando REPORT

Das **REPORT**-Kommando analysiert das RMAN Repository der Zieldatenbank, um Antworten auf verschiedene Fragen zu liefern.

30.5.1 Welche Dateien wurden noch nicht gesichert?

Erfüllt eine Datendatei die aktuelle Backup Retention Policy nicht, weil noch nicht genügende Backups vorliegen, bzw. das älteste Backup noch nicht alt genug ist, um das Recovery Window zu gewährleisten, kann dies mit dem Kommando `REPORT NEED BACKUP` ermittelt werden.

Listing 30.54: Wer verstößt gegen die Retention Policy?

```
RMAN> REPORT NEED BACKUP;

RMAN retention policy will be applied to the command
RMAN retention policy is set to redundancy 1
Report of files with less than 1 redundant backups
File #bkps Name
-----
3    0    /u01/app/oracle/oradata/orcl/undotbs01.dbf
4    0    /u01/app/oracle/oradata/orcl/users01.dbf
5    0    /u01/app/oracle/oradata/orcl/example01.dbf
```

Die Ausgabe aus Beispiel ?? zeigt an, dass bei einer redundanzbasierenden Retention Policy mit dem Wert 1, noch insgesamt 3 Datendateien gesichert werden müssen, ehe die konfigurierte Policy eingehalten wird.

Es ist auch möglich, mit der `NEED BACKUP`-Klausel eine andere Retention Policy zu unterstellen. Soll zum Beispiel geprüft werden, welche Datendateien noch gesichert werden müssten, wenn eine redundanzbasierende Retention Policy mit dem Wert 2 Gültigkeit hätte, wird die `NEED BACKUP`-Klausel wie folgt ergänzt:

Listing 30.55: Datenbankdateien die gesichert werden müssen

```
RMAN> REPORT NEED BACKUP REDUNDANCY 2;

Report of files with less than 2 redundant backups
File #bkps Name
-----
1    1    /u01/app/oracle/oradata/orcl/system01.dbf
2    1    /u01/app/oracle/oradata/orcl/sysaux01.dbf
3    0    /u01/app/oracle/oradata/orcl/undotbs01.dbf
4    0    /u01/app/oracle/oradata/orcl/users01.dbf
5    0    /u01/app/oracle/oradata/orcl/example01.dbf
7    1    /u01/app/oracle/oradata/orcl/bank01.dbf
```

Der gleiche Vorgang ist auch für eine Zeitfensterbasierende Retention Policy möglich.

Listing 30.56: Datenbankdateien die gesichert werden müssen

```
RMAN> REPORT NEED BACKUP RECOVERY WINDOW OF 3 DAYS;

Report of files that must be backed up to satisfy 3 days recovery window
```

File	Days	Name
1	4	/u01/app/oracle/oradata/orcl/system01.dbf
3	1535	/u01/app/oracle/oradata/orcl/undotbs01.dbf
4	1535	/u01/app/oracle/oradata/orcl/users01.dbf
5	67	/u01/app/oracle/oradata/orcl/example01.dbf
7	4	/u01/app/oracle/oradata/orcl/bank01.dbf

Die Spalte TAGE gibt an, wie viele Archive Log Informationen, in Tagen gerechnet, eine Datendatei benötigt, um vollständig wiederhergestellt werden zu können.

30.5.2 Nicht wiederherstellbare Datendateien aufspüren

Nicht rekonstruierbare Operationen (unrecoverable operations) sind Schreibvorgänge in Datendateien, die unter Umgehung der SGA stattfinden. Dabei werden Informationen direkt in Datendateien geschrieben, ohne vorher durch den Database Buffer Cache geschleust zu werden. Ein solcher Schreibvorgang wird als „Direct Load“ bezeichnet und hat den Vorteil, dass er sehr viel schneller ist, als ein normaler Schreibvorgang. Der Nachteil an einer solchen Vorgehensweise ist jedoch, dass keine Redo-Informationen erzeugt werden.

Daraus resultiert, dass eine so behandelte Datendatei im Falle eines Crashes auch nicht mit Hilfe der Archive Logs wiederhergestellt werden kann. Die Empfehlung seitens Oracle ist deshalb, eine solche Datei umgehend in einem Backup zu sichern.

Listing 30.57: Datenbankdateien die nicht rekonstruierbare Operationen enthalten aufspüren

```
RMAN> REPORT UNRECOVERABLE;

Report of files that need backup due to unrecoverable operations
File Type of Backup Required Name
-----
5    full                  /u01/app/oracle/oradata/orcl/example01.dbf
```

30.5.3 Nicht mehr benötigte Backups

Wenn eine ausreichende Anzahl Backups vorhanden ist, um die geltende Retention Policy zu erfüllen, werden alle Backups, die nicht mehr benötigt werden, als „obsolete“ markiert. Eine Auflistung der obsoleten Backups kann mit `REPORT OBSOLETE` angezeigt werden.

Listing 30.58: Backups die nicht mehr benötigt werden anzeigen

```
RMAN> REPORT obsolete;
```

```
RMAN retention policy will be applied to the command
RMAN retention policy is set to redundancy 1
Report of obsolete backups and copies
Type          Key    Completion Time   Filename/Handle
-----
Backup Set    2      24-OCT-13
  Backup Piece 2      24-OCT-13           /u05/fast_recovery_area/0RCL/b...
Backup Set    4      24-OCT-13
  Backup Piece 4      24-OCT-13           /u05/fast_recovery_area/0RCL/...
```

Auch hierbei kann wieder eine veränderte Retention Policy unterstellt werden, so wie es auch bei der **NEED BACKUP**-Klausel möglich ist.

Listing 30.59: Backups die nicht mehr benötigt werden anzeigen

```
RMAN> REPORT obsolete RECOVERY WINDOW OF 3 DAYS;

Report of obsolete backups and copies
Type          Key    Completion Time   Filename/Handle
-----
Backup Set    2      24-OCT-13
  Backup Piece 2      24-OCT-13           /u05/fast_recovery_area/0RCL/b...
Backup Set    4      24-OCT-13
  Backup Piece 4      24-OCT-13           /u05/fast_recovery_area/0RCL/a...
```

30.6 Erarbeiten von Backup und Recovery Strategien

30.6.1 Redundancy Sets

Das Set der Dateien, das dazu benötigt wird, um eine Datenbank im Fehlerfall zu Recovern wird als „Redundancy Set“ bezeichnet. Ein solches Set sollte folgende Dateien enthalten:

- Das letzte Backup der Kontrolldatei und aller Datendateien
- Alle Archive Logs, die nach dem letzten Vollbackup entstanden sind
- Multiplexing Duplikate von Kontrolldateien und Redo Log Dateien
- Kopien von Konfigurationsdateien (SPFile, tnsnames.ora, listener.ora)

Aus Sicherheitsgründen ist es wichtig, das Redundancy Set nicht auf dem gleichen Datenträger aufzubewahrt, wie die Zieldatenbank selbst. Dies wird am Einfachsten dadurch erreicht, das eine Fast Recovery

Area auf einem von der Datenbank getrennten Datenträger eingerichtet wird. Unabhängig davon sollten folgende Empfehlungen beachtet werden:

- Kontroll- und Redo Log Dateien sollten auf Datenbankebene verteilt werden (Multiplexing).
- Wird die Datenbank im Archive Log Modus betrieben, sollten die Archive an unterschiedliche Stellen verteilt werden.
- Für die Speicherung der Datendateien sollte Mirroring auf Betriebssystem- oder Hardwareebene genutzt werden.
- Es sollte immer mindestens eine Kopie des Redundancy Sets verfügbar sein. Wird das Redundancy Set auf einem Band gespeichert, sollten aufgrund der hohen Ausfallwahrscheinlichkeit eines Bands immer mindestens zwei Kopien des Redundancy Sets vorhanden sein.

Die Nutzung einer Fast Recovery Area ist nicht verpflichtend, wird jedoch seitens Oracle empfohlen.

30.6.2 Planen einer Backupstrategie

Eine Backupstrategie legt fest, welche Teile der Datenbank gesichert werden müssen, welche Tools dafür herangezogen werden und wie die Datenbank möglichst robust konfiguriert werden kann, um Backup und Recovery Operationen zu erleichtern. Die Entwicklung einer Backupstrategie ist auch nicht zu letzt eine Frage des Budgets, da hochwertige Backupsysteme meist sehr kostspielig sind.

Archive Log oder Noarchive Log Modus

Die Datenbank kann in zwei verschiedenen Modi betrieben werden: Archivelog und Noarchivelog. Der Archivelog Modus erweitert die Möglichkeiten des DBA bei einem Recovery wesentlich, hat jedoch auch Nachteile, wie z. B. einen höheren Speicherplatzverbrauch. Im Folgenden werden die Vor- und Nachteile dieser beiden Modi gegenübergestellt. So kann eine Entscheidung getroffen werden, welcher Modus der Richtige für den Betrieb der Datenbank ist.

Die Datenbank im Noarchivelog Modus zu betreiben, hat folgende Auswirkungen:

- Es können keine Online-Backups gemacht werden. Die Datenbank muss vor einem Backup heruntergefahren werden.

- Erweiterte Recovery Techniken, wie z. B. Point-In-Time Recovery oder Flashback Database können nicht genutzt werden.

Im Gegensatz dazu, hat der Betrieb der Datenbank im Archivelog Modus diese Auswirkungen:

- Die gesamte Breite der Recovery Techniken steht dem Administrator zur Verfügung.
- Zusätzlicher Speicherplatz für die Speicherung der Archive Logs wird benötigt.
- Die Archive Logs müssen verwaltet werden (Bereitstellung von Speicherorten, Sicherung auf Band, usw.)
- Durch die Archivierung entsteht ein Performance Overhead.

Festlegen der Backupfrequenz

Abhängig von der Häufigkeit, mit der eine Datenbank geändert wird, sollte auch die Backupfrequenz bestimmt werden. Sie richtet sich nach den folgenden Gesichtspunkten:

- Wie häufig werden Schemaobjekte gelöscht und neu angelegt?
- Wie häufig werden Tabellenzeilen bearbeitet (eingefügt, gelöscht, geändert)?
- Wie häufig wird die Struktur der Datenbank geändert?

Von den Antworten auf diese Fragen, den vorhandenen Mitteln und der maximalen Down-Time, welche die Datenbank aufweisen darf, hängt ab, wie häufig ein Backup der Datenbank erfolgen muss. Bei strukturellen Änderungen der Datenbank sollte grundsätzlich immer vorher und nachher ein Backup erfolgen.

Wenn die Änderungshäufigkeit für bestimmte Objekte sehr hoch ist und im Rest der Datenbank eher niedrig, kann auch ein partielles Backup der Datenbank in Frage kommen, so dass häufig geänderte Objekte auch häufiger gesichert werden, als andere.

Wartungsfenster

Die Planung einer Backupstrategie hängt nicht zuletzt von evtl. vorhanden Wartungsfenstern ab.



Als Wartungsfenster wird ein Zeitraum bezeichnet, innerhalb dessen die Datenbank zu Wartungszwecken heruntergefahren werden kann.

Da jede Datenbank einen bestimmten Verfügbarkeitsgrad haben muss, also zu bestimmten Zeiten arbeitsbereit sein soll, werden die Möglichkeiten für Wartungsfenster automatisch eingeschränkt.

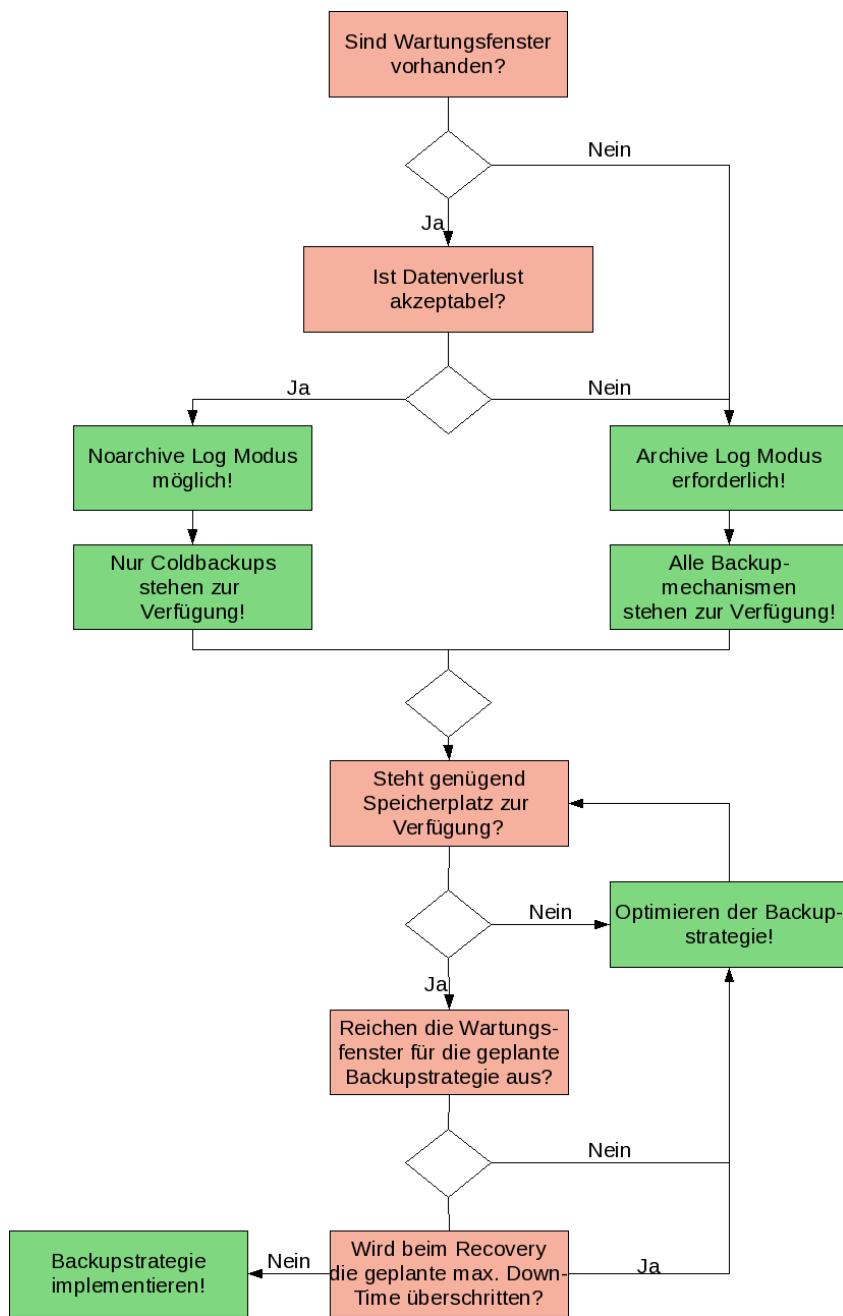
Maximale Down-Time



Als Down-Time wird der Zeitraum bezeichnet, in dem die Datenbank aufgrund eines Ausfalls nicht mehr verfügbar ist. Hier gilt es möglichst realistische Schätzungen, evtl. auch basierend auf Erfahrungswerten, zu machen, wie lange ein schnellst mögliches Recovery der Datenbank dauert und wie lange die Datenbank somit nicht verfügbar sein wird. Die max. Down-Time sollte so gering wie möglich gehalten werden.

Da eine Backupstrategie die Basis für jede Recoverystrategie darstellt, muss die max. Down-Time nicht erst beim Erstellen der Recoverystrategie(n) berücksichtigt werden, sondern bereits hier. Abbildung ?? zeigt eine mögliche Vorgehensweise bei der Planung einer Backupstrategie.

Abb. 30.4:
Planung einer
Backupstrategie



30.6.3 RMAN-Skripte benutzen

RMAN stellt die Möglichkeit bereit, Skripte für ständig wiederkehrende Aufgaben zu benutzen. Dabei handelt es sich um einfache Textdateien, die auf jeder Betriebssystemplattform erstellt und genutzt werden können. Aus diesem Grund stellen RMAN-Skripte das ideale Hilfsmittel zur Implementierung einer Backupstrategie dar.

RMAN mit Skript-Datei starten

Listing 30.60: Aufruf eines RMAN-Skripts

```
[oracle@FEA11-119SRV ~]$ rman target / cmdfile backup_sunday_full.cmd
```



Die Dateiendung .cmd wurde willkürlich gewählt.

Nach der Ausführung aller Kommandos im Skript, wird RMAN automatisch beendet. Um die Ausgaben des RMAN-Skripts sehen zu können, kann zusätzlich eine Logdatei angegeben werden.

Listing 30.61: RMAN-Skript mit Log-Datei benutzen

```
[oracle@FEA11-119SRV ~]$ rman target / cmdfile backup_sunday_full.cmd \
> log backup_sunday_full.log
```

Skript-Dateien in RMAN starten

RMAN-Skripte können auch innerhalb von RMAN aufgerufen werden:

Listing 30.62: Ein Skript-Datei in RMAN aufrufen

```
RMAN> @backup_sunday_full.cmd
```

Wurden die Kommandos im RMAN-Skript ausgeführt, zeigt RMAN die Meldung „**end-of-file**“ und wird nicht beendet.

Syntax Check

RMAN bietet die Möglichkeit, einen Syntax-Check an Skript-Dateien durchzuführen. Dies geschieht mit Hilfe des Parameters CHECKSYNTAX. Wird RMAN mit diesem Parameter aufgerufen, werden alle Kom-

mandos nur getestet und nicht ausgeführt. Ist ein Kommando fehlerhaft, wird die RMAN-Fehlermeldung „RMAN-00558“ ausgegeben.

Listing 30.63: Ein RMAN-Skript auf korrekte Syntax überprüfen

```
[oracle@FEA11-119SRV ~]$ rman checksyntax cmdfile backup_sunday_full.cmd

Recovery Manager: Release 11.2.0.1.0 - Production on Sun Oct 27 08:51:42 2013

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

RMAN> BACKUP database;
The cmdfile has no syntax errors
Recovery Manager complete.
```

30.6.4 Planen einer Recoverystrategie

Die Fehler, die in einer Datenbank entstehen können, decken die gesamte Skala von Nutzerfehlern, über Blockfehler in Datendateien bis hin zu Medienfehlern ab. Wie schnell ein Fehler behoben werden kann, hängt im Wesentlichen von der Recoverystrategie ab.

Vor der Erstellung einer Recoverystrategie sollten folgende Fragen beantwortet werden:

- Wie soll auf den Ausfall eines gesamten Speichermediums reagiert werden?
- Wie kann ein logischer Fehler, der durch eine Anwendung erzeugt wurde, aufgespürt und behoben werden?
 - Welche Auswirkungen entstehen dabei auf zwischenzeitlich durchgeführte Updates?
 - Wie kann ein erneutes Auftreten des gleichen Fehlers verhindert werden?
- Wie soll reagiert werden, wenn das Alert.log File einen oder mehrere defekte Datenblöcke in den Datendateien anzeigt und wie können diese Blöcke repariert werden?
- Welches Disaster Recovery ist im Falle der Zerstörung des kompletten Datenbankservers notwendig und wie lange würde ein solches Recovery dauern?
- Kann eine andere Person, die Datenbank Recovern, falls der DBA abwesend ist?

Wurden Antworten auf diese Fragen gefunden, kann geklärt werden, welche Techniken beim Recovery zum Einsatz kommen sollen. Hier stehen zur Verfügung:

- User Managed Backup and Recovery
- Der Recovery Manager
- Oracle Flashback Database
- Block Media Recovery

Dies sind jedoch nur einige wenige Überlegungen, die bei der Planung einer Recoverystrategie gemacht werden sollten. Zusätzlich kommt es immer auf Hardware, Personal und Budget an.

30.7 Ein Backupszenario

30.7.1 Backupstrategie bei hohem Änderungsvolumen

In diesem Szenario wird eine Datenbank beschrieben, für die eine wöchentliche Sicherung eingerichtet werden soll. Das Änderungsvolumen dieser Datenbank, innerhalb einer Woche, ist hoch. Da in einem solchen Szenario selbst inkrementelle Backups sehr groß werden können, wird hier ein wöchentliches Full-Backup mit zusätzlicher Sicherung der Archive Logs und einer Zeitfensterbasierten Retention Policy empfohlen.

Die Strategie setzt sich aus den folgenden einzelnen Elementen zusammen:

- Die Redo Logs werden mit Hilfe der Fast Recovery Area gesichert.
- Wöchentlich wird ein Full-Backup der Datenbank in die Fast Recovery Area gesichert.
- Täglich werden alle Backupdateien (einschließlich der Archive Logs), die sich in der Fast Recovery Area befinden auf ein SBT-Gerät gesichert. Obsolete Backups werden vom SBT gelöscht.

Wird für die Backupstrategie eine Recovery Window basierte Retention Policy verwendet, ist sichergestellt, dass alle Backups, solange sie benötigt werden, vorrätig sind.

Vorbereitungen

Vorbereitend für diese Backupstrategie muss eine Fast Recovery Area mit der richtigen Größe erstellt werden. Diese kann nach folgender Formel berechnet werden:

Speicherplatzbedarf = (Größe eines Full-Backups) + (Größe aller Archive Logs für Y+1 Tage)

Die Variable Y steht in dieser Formel für die Anzahl der Tage, die zwischen den Sicherungen der Fast Recovery Area vergehen. In diesem Szenario erfolgt eine tägliche Sicherung, d. h. es gilt $Y = 1$. Wichtig ist ebenfalls, dass die Fast Recovery Area als Speicherort für die Redo Logs angegeben wird.

Zur Umsetzung dieser Strategie werden zwei verschiedene RMAN-Skripte benutzt. Eines für das wöchentliche Full-Backup und eines für die täglichen Backups der Fast Recovery Area. Das wöchentliche Skript wird jeden Sonntag ausgeführt. Das tägliche Skript wird sechs Tage die Woche ausgeführt.

Das wöchentliche Skript

Listing 30.64: Das wöchentliche Skript

```
RMAN> RUN {  
2>     ALLOCATE CHANNEL c1 DEVICE TYPE disk;  
3>     ALLOCATE CHANNEL c2 DEVICE TYPE sbt  
4>     PARMS 'SBT_LIBRARY=oracle.disksbt ,ENV=(BACKUP_DIR=/u04)';  
5>  
6>     BACKUP AS COMPRESSED BACKUPSET database  
7>     CHANNEL c1;  
8>  
9>     BACKUP recovery area  
10>    CHANNEL c2;  
11>  
12>     DELETE obsolete DEVICE TYPE sbt;  
13> }
```

Das tägliche Skript

Listing 30.65: Das tägliche Skript

```
RMAN> RUN {  
3>     ALLOCATE CHANNEL c2 DEVICE TYPE sbt  
4>     PARMS 'SBT_LIBRARY=oracle.disksbt ,ENV=(BACKUP_DIR=/u04)';  
5>  
5>     BACKUP recovery area  
6>     CHANNEL c2;  
7>  
8>     DELETE obsolete DEVICE TYPE sbt;  
9> }
```

Da täglich die gesamte Fast Recovery Area gesichert wird, kann der RMAN, Dateien aus der Recovery Area löschen, sobald dies notwendig ist. Zwischen den einzelnen Backups ist es schwierig zu sagen, welche Dateien noch in der Fast Recovery Area sind.

30.8 Informationen

30.8.1 Verzeichnis der relevanten Data Dictionary Views



- [REFRN30022]
- [REFRN30030]

30.9 Übungen - Backups mit dem RMAN



Zur Vorbereitung auf diese Übung löschen Sie bitte alle vorhandenen Backups!

1. Erstellen Sie auf dem Server FEA11-119CAT einen Recovery Katalog! Benutzen Sie dazu folgende Angaben:
 - \$ORACLE_SID: CATDB
 - Service Name: XE
 - Listenerport: 1521
 - Benutzername: catownerXX (XX = Ihre Platznummer, z. B. 04)
 - Passwort: SeCuRe
 - Tablespace: cattsXX (XX = Ihre Platznummer, z. B. 04)
 - Datendatei: /u02/oradata/CATDB/cattsXX_01.dbf (10MB, Maxsize: 20MB)
2. Registrieren Sie Ihre Datenbank in Ihrem neuen Recovery Katalog!
3. Konfigurieren Sie eine Retention Policy auf Redundanzbasis, mit dem Wert 2!
4. Erstellen Sie ein unkomprimiertes Backup Set des Tablespaces BANK! Das Backup Set soll in der FRA (Fast Recovery Area) abgelegt werden!
5. Ermitteln Sie den Backup Set Key des gerade eben erstellten Backup Sets!

6. Löschen Sie das gerade erstellte Backup Set des Tablespaces BANK!
7. Erstellen Sie erneut ein Backup Set des Tablespaces BANK, aber dieses mal in komprimierter Form! Vergeben Sie die Beschriftung „ts_bank_DD.MM.YYYY-HH24:MI“ für dieses Backup Set! Ersetzen Sie die Buchstaben durch eine konkrete Datums/Zeit-Angabe!

8. Ermitteln Sie in der Oracle Online-Dokumentation, wie Sie das gerade erstellte Backup Set auf das SBT-Gerät verschieben können! Schlagen Sie in der „Oracle Database Backup and Recovery Reference“, unter dem Kapitel „RMAN Commands: @ (at sign) to Quit“ und dort unter „BACKUP“ nach! Das Beispiel „Example 2-21“ beschreibt einen solchen Vorgang.
 9. Verschieben Sie jetzt das Backup Set des BANK-Tablespaces auf das SBT-Gerät!
 10. Löschen Sie das Backup Set des BANK-Tablespaces von der Festplatte, so dass es nur noch auf SBT verfügbar ist!
 11. Erstellen Sie einen RUN-Block, der folgende Tätigkeiten durchführt:
 - Anfertigen eines komprimierten Backup Sets der Datenbank in der FRA, unter Ausnutzung der Paralellisierung
 - Verschieben des Backup Sets auf das SBT-Gerät
 12. Prüfen Sie, welche Datendateien, gemäß der aktuellen Retention Policy noch gesichert werden müssen!
 13. Ermitteln Sie, ob Sie obsolete Backups haben! Falls ja, löschen Sie sie!
 14. Ermitteln Sie, wie viele Backups Sie von Ihrer Kontrolldatei haben.
-

15. Schlagen Sie die aktuelle Größe und den belegten Speicherplatz Ihrer Fast Recovery Area nach! Welcher View können Sie diese Angabe entnehmen?

16. Wie groß ist die Auslastung Ihrer FRA durch Backup Pieces (prozentual)?

17. Vergrößern Sie Ihre FRA auf 6 Gigabyte!
18. Erstellen Sie RMAN-Skripte, um die im Folgenden beschriebene Backup Strategie zu implementieren! Benutzen Sie für die Umsetzung der Skripte, RUN-Blöcke und die manuelle Kanalanforderung!
 - Jeden Sonntag muss ein komprimiertes Level 0 Backup der Datenbank erstellt werden!

- Von Montag bis Freitag werden im Drei-Stunden-Rhythmus Backups der Archive Logs erzeugt. Aus Sicherheitsgründen müssen diese Backups, an den Speicherorten /u02/backup und /u03/backup dupliziert werden! Alle Kopien der gesicherten Archive Logs sollen automatisch aus der FRA entfernt werden!
 - Jeden Montag, Dienstag, Donnerstag und Freitag werden Level 1 Backups (inkrementell) der Datenbank erstellt und in der FRA gespeichert!
 - Jeden Mittwoch muss ein kumulatives Level 1 Backup der Datenbank erzeugt werden!
 - Jeden Samstag sind der komplette Inhalt der FRA und die Backups der Archive Logs auf das SBT-Gerät zu verschieben! Es ist sicherzustellen, dass im Anschluss daran, alle obsoleten Backups gelöscht werden!
 - Für diese Backup Strategie ist ein Recovery Window von 7 Tagen notwendig.
 - Aktivieren Sie das Block Change Tracking, um die Dauer der inkrementellen Backups zu beschleunigen!
19. Führen Sie das Skript labs/lab_delete_backups.sql aus. Dieses Skript wird eine willkürliche Anzahl Ihrer Backups löschen.
20. Prüfen Sie im RMAN welche Backups nun nicht mehr zur Verfügung stehen.

21. Löschen Sie die Einträge für alle nicht mehr verfügbaren Backups.

22. Durchsuchen Sie das Verzeichnis /tmp. Dort befinden sich Kopien aller Backups, die gelöscht wurden. Verschieben Sie diese Kopien an den Speicherort /u02/backup und registrieren Sie sie im Recovery Katalog.

30.10 Lösungen - Backups mit dem RMAN

1. Erstellen Sie auf dem Server FEA11-119CAT einen Recovery Katalog! Benutzen Sie dazu folgende Angaben:

- \$ORACLE_SID: CATDB
- Service Name: XE
- Listenerport: 1521
- Benutzername: catownerXX (XX = Ihre Platznummer, z. B. 04)
- Passwort: SeCuRe
- Tablespace: cattsXX (XX = Ihre Platznummer, z. B. 04)
- Datendatei: /u02/oradata/CATDB/cattsXX_01.dbf (10MB, Maxsize: 20MB)

Listing 30.66: TNS-Eintrag in der tnsnames.ora

```
CATDB =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = FEA11-119CAT)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = CATDB.local)
    )
  )
```

Listing 30.67: Erstellen des Katalogschemas

```
[oracle@FEA11-119SRV ~]$ sqlplus sys/oracle@CATDB as sysdba

SQL> CREATE TABLESPACE catts99
  2  DATAFILE '/u03/oradata/CATDB/catts99_01.dbf' SIZE 10M
  3  AUTOEXTEND ON MAXSIZE 20M;

Tablespace created.

SQL> CREATE USER catowner99
  2  IDENTIFIED BY SeCuRe
  3  DEFAULT TABLESPACE catts99
  4  QUOTA unlimited ON catts99;
```

```
User created.  
SQL> GRANT create session, recovery_catalog_owner  
  2  TO catowner99;
```

Listing 30.68: Recovery Katalog erstellen und Datenbank registrieren

```
[oracle@FEA11-119SRV ~]$ export NLS_DATE_FORMAT='DD.MM.YYYY HH24:MI'  
[oracle@FEA11-119SRV ~]$ rman catalog catowner/catpass@CATDB  
RMAN> CREATE CATALOG;
```

2. Registrieren Sie Ihre Datenbank in Ihrem neuen Recovery Katalog!

```
RMAN> connect target /  
RMAN> REGISTER DATABASE;
```

3. Konfigurieren Sie eine Retention Policy auf Redundanzbasis, mit dem Wert 2!

```
RMAN> CONFIGURE RETENTION POLICY TO REDUNDANCY 2;
```

4. Erstellen Sie ein unkomprimiertes Backup Set des Tablespaces BANK! Das Backup Set soll in der FRA (Fast Recovery Area) abgelegt werden!

```
RMAN> BACKUP AS BACKUPSET tablespace bank;
```

5. Ermitteln Sie den Backup Set Key des gerade eben erstellten Backup Sets!

```
RMAN> LIST backupset OF tablespace bank SUMMARY  
2>      COMPLETED AFTER '29.10.2013'  
  
List of Backups  
=====  
Key      TY LV S Device Type Completion Time #Pieces #Copies Compressed Tag  
-----  -- -- - -----  
9        B  F  A DISK          29.10.13 11:54   1       1      NO      TAG...
```

6. Löschen Sie das gerade erstellte Backup Set des Tablespaces BANK!

```
RMAN> DELETE backupset 9;
```

7. Erstellen Sie erneut ein Backup Set des Tablespaces BANK, aber dieses mal in komprimierter Form! Vergeben Sie die Beschriftung „ts_bank_DD.MM.YYYY-HH24:MI“ für dieses Backup Set! Ersetzen Sie die Buchstaben durch eine konkrete Datums/Zeit-Angabe!

```
RMAN> BACKUP AS COMPRESSED BACKUPSET tablespace bank  
2>      TAG='ts_bank_29.10.2013-13:53';
```

8. Verschieben Sie jetzt das Backup Set des BANK-Tablespaces auf das SBT-Gerät!

```
RMAN> CONFIGURE CHANNEL DEVICE TYPE sbt
2>     PARMS 'SBT_LIBRARY=oracle.disksbt,ENV=(BACKUP_DIR=/u04)';
RMAN> BACKUP DEVICE TYPE sbt
2>     backupset FROM TAG='ts_bank_29.10.2013-13:53';
```

9. Löschen Sie das Backup Set des BANK-Tablespaces von der Festplatte, so dass es nur noch auf SBT verfügbar ist!

```
RMAN> DELETE backuppiece 11;
```

10. Erstellen Sie einen RUN-Block, der folgende Tätigkeiten durchführt:

- Anfertigen eines komprimierten Backup Sets der Datenbank in der FRA, unter Ausnutzung der Paralellisierung
- Verschieben des Backup Sets auf das SBT-Gerät

```
RMAN> CONFIGURE DEVICE TYPE disk PARALLELISM 2
2>     BACKUP TYPE TO COMPRESSED BACKUPSET;
RMAN> RUN {
2>     BACKUP DEVICE TYPE disk database
3>     TAG='database_29.10.2013-14:37';
4>
5>     BACKUP DEVICE TYPE sbt
6>     backupset FROM TAG='database_29.10.2013-14:37',
7>     DELETE INPUT;
8> }
```

11. Prüfen Sie, welche Datendateien, gemäß der aktuellen Retention Policy noch gesichert werden müssen!

```
RMAN> REPORT NEED BACKUP;
```

12. Ermitteln Sie, ob Sie obsolete Backups haben! Falls ja, löschen Sie sie!

```
RMAN> REPORT obsolete;
```

13. Eruieren Sie, wie viele Backups Sie von Ihrer Kontrolldatei haben.

```
RMAN> LIST BACKUP OF controlfile SUMMARY;
```

14. Schlagen Sie die aktuelle Größe und den belegten Speicherplatz Ihrer Fast Recovery Area nach!
Welcher View können Sie diese Angabe entnehmen?

```
SQL> col name format a30
SQL> SELECT name, space_limit, space_used
  2  FROM v$recovery_file_dest;

NAME                      SPACE_LIMIT  SPACE_USED
-----  -----
/u05/fast_recovery_area      4294967296  1446789120
```

15. Wie groß ist die Auslastung Ihrer FRA durch Backup Pieces (prozentual)?

```
SQL> SELECT file_type, percent_space_used
  2  FROM v$recovery_area_usage
  3  WHERE file_type LIKE 'BACKUP PIECE';

FILE_TYPE          PERCENT_SPACE_USED
-----  -----
BACKUP PIECE          33,69
```

16. Vergrößern Sie Ihre FRA auf 6 Gigabyte!

```
SQL> ALTER SYSTEM
  2  SET db_recovery_file_dest_size=6G;
```

17. Erstellen Sie RMAN-Skripte, um die im Folgenden beschriebene Backup Strategie zu implementieren! Benutzen Sie für die Umsetzung der Skripte, RUN-Blöcke und die manuelle Kanalanforderung!

- Jeden Sonntag muss ein komprimiertes Level 0 Backup der Datenbank erstellt werden!
- Von Montag bis Freitag werden im Drei-Stunden-Rhythmus Backups der Archive Logs erzeugt. Aus Sicherheitsgründen müssen diese Backups, an den Speicherorten /u02/backup und /u03/backup dupliziert werden! Alle Kopien der gesicherten Archive Logs sollen automatisch aus der FRA entfernt werden!
- Jeden Montag, Dienstag, Donnerstag und Freitag werden Level 1 Backups (inkrementell) der Datenbank erstellt und in der FRA gespeichert!
- Jeden Mittwoch muss ein kumulatives Level 1 Backup der Datenbank erzeugt werden!
- Jeden Samstag sind der komplette Inhalt der FRA und die Backups der Archive Logs auf das SBT-Gerät zu verschieben! Es ist sicherzustellen, dass im Anschluss daran, alle obsoleten Backups gelöscht werden!
- Für diese Backup Strategie ist ein Recovery Window von 7 Tagen notwendig.

- Aktivieren Sie das Block Change Tracking, um die Dauer der inkrementellen Backups zu beschleunigen!

Listing 30.69: Block Change Tracking aktivieren

```
SQL> ALTER DATABASE
  2  ENABLE BLOCK CHANGE TRACKING
  3  USING FILE '/u02/rman_change_track.f' REUSE;
```

Listing 30.70: Retention Policy setzen

```
RMAN> CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 7 DAYS;
```

Listing 30.71: backup_sunday_level_0.cmd

```
RMAN> RUN {
  2>   ALLOCATE CHANNEL c_disk_1
  3>   DEVICE TYPE disk;
  4>
  5>   BACKUP AS COMPRESSED BACKUPSET INCREMENTAL LEVEL 0
  6>   CHANNEL c_disk_1
  7>   database;
  8> }
```

Listing 30.72: backup_archive_log.cmd

```
RMAN> RUN {
  2>   ALLOCATE CHANNEL c_disk_1
  3>   DEVICE TYPE disk
  4>   FORMAT '/u02/backup/archive_log_%u_%c.bkp';
  5>
  6>   ALLOCATE CHANNEL c_disk_2
  7>   DEVICE TYPE disk
  8>   FORMAT '/u03/backup/archive_log_%u_%c.bkp';
  9>
 10>  BACKUP COPIES 2 archivelog all
 11>  DELETE ALL INPUT;
 12> }
```

Listing 30.73: backup_mon_tue_thu_fri.cmd

```
RMAN> BACKUP INCREMENTAL LEVEL 1 database;
```

Listing 30.74: backup_wed.cmd

```
RMAN> BACKUP CUMULATIVE INCREMENTAL LEVEL 1 database;
```

Listing 30.75: backup_sat_fra.cmd

```
RMAN> RUN {
2>   ALLOCATE CHANNEL c_sbt_1
3>   DEVICE TYPE sbt
4>   PARMS 'SBT_LIBRARY=oracle.disksbt ,ENV=(BACKUP_DIR=/u04)';
5>
6>   BACKUP recovery area;
7>
8>   BACKUP archivelog all;
9> }
```

18. Führen Sie das Skript labs/lab_delete_backups.sql aus. Dieses Skript wird eine willkürliche Anzahl Ihrer Backups löschen.

```
[oracle@FEA11-119SRV ~]$ sqlplus / as sysdba
SQL> @labs/lab_delete_backups.sql
```

19. Prüfen Sie im RMAN welche Backups nun nicht mehr zur Verfügung stehen.

```
RMAN> CROSSCHECK backup;
```

20. Löschen Sie die Einträge für alle nicht mehr verfügbaren Backups.

```
RMAN> DELETE expired backup;
```

21. Durchsuchen Sie das Verzeichnis /tmp. Dort befinden sich Kopien aller Backups, die gelöscht wurden. Verschieben Sie diese Kopien an den Speicherort /u02/backup und registrieren Sie sie im Recovery Katalog.

```
RMAN> CATALOG START WITH '/u02/backup/';
```

31 Recovery mit dem RMAN

Inhaltsangabe

Obwohl der RMAN das Recovery einer Oracle-Datenbank wesentlich vereinfacht, sind im Vorfeld trotzdem noch einige Planungsarbeiten notwendig. Welche dies sind, ist im Wesentlichen von der Art des Datenverlustes abhängig.

31.1 Ein Recovery planen und vorbereiten

Die Durchführung eines Restore und Recovery Prozesses besteht aus 5 Schritten.

1. Ermitteln welche Datenbankdateien wiederhergestellt werden müssen und welche Backups dafür herangezogen werden können. Dies kann auch Archive Logs, das SPFile und die Kontrolldatei einschließen.
2. Die Datenbank in den benötigten Zustand versetzen. Für ein vollständiges Restore and Recovery der ganzen Datenbank ist dies meist der MOUNT-Status. Müssen nur ein einzelner Tabelspace oder einzelne Datendateien wiederhergestellt werden, genügt es, den betreffenden Tablespace in den Offline-Status zu versetzen.
3. Durchführen der Restore-Phase. Hierbei kann es notwendig sein, die Datenbankdateien an einem neuen Speicherort wiederherzustellen, weil der Alte nicht mehr verfügbar ist. Eventuelle Anpassungen am SPFile dürfen dabei nicht vergessen werden.
4. Durchführen der Recovery-Phase.
5. Durchführen von Tätigkeiten, die zum weiteren Betrieb der Datenbank notwendig sind, z. B. öffnen der Datenbank.

Nicht jeder Recovery-Prozess benötigt immer alle 5 Schritte. Muss beispielsweise nur das SPFile aus einem Backup wiederhergestellt werden, ist ein Recovery der Datenbank nicht notwendig. |

31.2 Pflege- und Erhaltungsmaßnahmen

31.2.1 Datenbankdateien auf Fehler prüfen

Mit dem `VALIDATE`-Kommando können Datenbankdateien vor einem Backup auf Funktionsfähigkeit geprüft werden. Dabei wird getestet, ob die Datenbankdateien existieren, ob sie sich am richtigen Speicherort

befinden und ob sie physische bzw. logische Beschädigungen aufweisen. RMAN verfährt dabei genauso wie bei einem Backup, nur wird beim Validieren kein Backup Set erzeugt.

Listing 31.1: Eine Datendatei validieren

```
RMAN> VALIDATE datafile 1;

Starting validate at 29-OCT-13
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=25 device type=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: SID=145 device type=DISK
channel ORA_DISK_1: starting validation of datafile
channel ORA_DISK_1: specifying datafile(s) for validation
input datafile file number=00001 name=/u01/app/oracle/oradata/orcl/system01.dbf
channel ORA_DISK_2: starting validation of datafile
channel ORA_DISK_2: specifying datafile(s) for validation
including current SPFILE in backup set
channel ORA_DISK_2: validation complete, elapsed time: 00:00:01
List of Control File and SPFILE
=====
File Type      Status Blocks Failing Blocks Examined
-----
SPFILE         OK      0          2
channel ORA_DISK_2: starting validation of datafile
channel ORA_DISK_2: specifying datafile(s) for validation
including current control file for validation
channel ORA_DISK_2: validation complete, elapsed time: 00:00:01
List of Control File and SPFILE
=====
File Type      Status Blocks Failing Blocks Examined
-----
Control File   OK      0          594
channel ORA_DISK_1: validation complete, elapsed time: 00:00:48
List of Datafiles
=====
File Status Marked Corrupt Empty Blocks Blocks Examined High SCN
-----
1    OK      0          13310     92201      2086674
File Name: /u01/app/oracle/oradata/orcl/system01.dbf
Block Type Blocks Failing Blocks Processed
-----
Data        0          59907
Index       0          12711
Other       0          6232

Finished validate at 29-OCT-13
```

Die Syntax des `VALIDATE`-Befehls ist der des `BACKUP`-Kommandos sehr ähnlich. Es können Archive Logs, Kontrolldateien, SPFiles, Datendateien, Tablespaces oder auch die gesamte Datenbank geprüft werden.

Listing 31.2: Eine ganze Datenbank validieren

```
RMAN> VALIDATE database;
```

Listing 31.3: Validieren der Archive Logs

```
RMAN> VALIDATE archivelog all;
Starting validate at 29-OCT-13
released channel: ORA_SBT_TAPE_1
using channel ORA_DISK_1
using channel ORA_DISK_2
channel ORA_DISK_1: starting validation of archived log
channel ORA_DISK_1: specifying archived log(s) for validation
input archived log thread=1 sequence=56 RECID=89 STAMP=830101540
channel ORA_DISK_1: validation complete, elapsed time: 00:00:01
List of Archived Logs
=====
Thrd Seq      Status Blocks Failing Blocks Examined Name
----- ----- ----- ----- -----
1    56      OK        0           130          /u02/backup/archive_logs/1...
Finished validate at 29-OCT-13
```

Wird während der Validierung ein Fehler in einer Datenbankdatei entdeckt, wird die View V\$DATABASE_BLOCK_CORRUPTION mit Informationen gefüllt. Diese Fehler können dann evtl. durch ein Block-Media Recovery behoben werden.

31.2.2 Funktionsfähigkeit eines Backup Sets ermitteln

Das Kommando **VALIDATE BACKUPSET** ermöglicht die Validierung eines Backup Sets. Dabei wird das Backup Set auf logische und physikalische Fehler überprüft.

Listing 31.4: Ein Backup Set validieren

```
RMAN> VALIDATE backupset 4711;

Starting validate at 29-OCT-13
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_SBT_TAPE_1
channel ORA_SBT_TAPE_1: starting validation of datafile backup set
channel ORA_SBT_TAPE_1: reading from backup piece 06on917u_1_1
channel ORA_SBT_TAPE_1: piece handle=06on917u_1_1 tag=TAG20131025T112446
channel ORA_SBT_TAPE_1: restored backup piece 1
channel ORA_SBT_TAPE_1: validation complete, elapsed time: 00:00:04
Finished validate at 29-OCT-13
```

31.2.3 Wiederherstellbarkeit einer Datenbankdatei prüfen

Neben den Möglichkeiten Datenbankdateien und Backups auf Fehlerfreiheit zu prüfen, kann noch ein weiterer Prüfschritt unternommen werden. Mit `RESTORE VALIDATE` kann festgestellt werden, ob eine Datenbankdatei, mit Hilfe der bestehenden Backups wiederhergestellt werden kann.

Listing 31.5: Kann der Tablespace BANK wiederhergestellt werden?

```
RMAN> RESTORE VALIDATE tablespace bank;

Starting restore at 29-OCT-13
using channel ORA_DISK_1
using channel ORA_DISK_2
allocated channel: ORA_SBT_TAPE_1
channel ORA_SBT_TAPE_1: SID=21 device type=SBT_TAPE
channel ORA_SBT_TAPE_1: WARNING: Oracle Test Disk API

channel ORA_DISK_1: starting validation of datafile backup set
channel ORA_DISK_2: starting validation of datafile backup set
channel ORA_DISK_1: reading from backup piece /u05/fast_recovery_area/0RCL/...
channel ORA_DISK_2: reading from backup piece /u05/fast_recovery_area/0RCL/...
channel ORA_DISK_2: piece handle=/u05/fast_recovery_area/0RCL/...
channel ORA_DISK_2: restored backup piece 1
channel ORA_DISK_2: validation complete, elapsed time: 00:00:03
channel ORA_DISK_1: piece handle=/u05/fast_recovery_area/0RCL/backupset/...
channel ORA_DISK_1: restored backup piece 1
channel ORA_DISK_1: validation complete, elapsed time: 00:00:15
Finished restore at 29-OCT-13
```

In Beispiel ?? wird der Restore-Prozess für den Tablespace bank validiert. Dabei wird geprüft, ob alle notwendigen Backup Sets, Image Copies und Archive Logs vorhanden und funktionsfähig sind.

31.2.4 Der Parameter DB_Ultra_Safe

Der Initialisierungsparameter `db_ultra_safe` wurde mit Oracle 11g neu eingeführt. Er soll helfen, Beschädigungen an Datenblöcken zu vermeiden. Er selbst stellt keine neuen Mechanismen zur Verfügung, sondern beeinflusst drei andere Initialisierungsparameter, die teilweise bereits seit Oracle 8i vorhanden sind. Dies sind:

- `db_block_checking`
- `db_lost_write_protect`
- `db_block_checksum`

DB_Block_Checking

Der Parameter db_block_checking legt fest, ob Oracleblöcke einer semantischen Prüfung unterzogen werden. Unter einer semantischen Prüfung versteht man den Abgleich der Blockstruktur mit seinem Inhalt. Wenn beispielsweise eine Spalte den Wert „Florian Weidinger“ enthält, aber den Datentyp DATE aufweist, muss dies ein Fehler sein. Ein anderes Beispiel sind Spalten des Typs VARCHAR2. Diese können in Oracle maximal 4.000 Byte lang sein. Taucht plötzlich eine Spalte auf, die größer als 4.000 Byte ist, so muss auch dies ein Fehler sein. Diese Art von Fehlern wird zumeist durch fehlerhafte Speichermedien hervorgerufen.

db_block_checking kennt folgende Werte:

- **OFF**: Das Blockchecking wird lediglich im SYSTEM-Tablespace durchgeführt.
- **FALSE**: Dieser Wert hat die gleiche Bedeutung wie „OFF“ und wird nur aus Kompatibilitätsgründen bereitgestellt.
- **LOW**: Es findet eine semantische Überprüfung der Blockheader, in allen Tablespace statt.
- **MEDIUM**: Es findet eine vollständige semantische Prüfung der Oracleblöcke, in allen Tablespace statt. Ausgenommen sind Indexstrukturen.
- **FULL**: Es findet eine vollständige semantische Prüfung der Oracleblöcke, in allen Tablespace statt. Auch Indexstrukturen werden geprüft.
- **TRUE**: Dieser Wert hat die gleiche Bedeutung wie „FULL“ und wird nur aus Kompatibilitätsgründen bereitgestellt.

DB_Lost_Write_Protect

Dieser Parameter ist nur in einer Oracle Data Guard Umgebung sinnvoll und hilft dort, Datenverlust durch Fehlfunktionen von Datenträgern zu vermeiden. Er kennt die Werte „NONE“, „TYPICAL“ und „FULL“.

DB_Block_Checksum

Mit Hilfe von db_block_checksum kann die Datenbank dazu veranlasst werden, jedem Block, der in eine Datendatei geschrieben wird, eine Checksumme zu geben. Mit Hilfe dieser Checksumme kann beim Lesen des Blockes sichergestellt werden, dass der Inhalt gelesen wird, der vorher geschrieben wurde (dass der Block unverändert geblieben ist).

Anders als mit db_block_checking kann mit diesem Parameter nicht verhindert werden, dass ein fehlerhafter Block gelesen wird. Wird ein Block beim Schreiben auf den Datenträger beschädigt, wird von diesem beschädigten Block eine Checksumme gebildet. Beim erneuten Lesen des Blockes wird nur die Checksumme geprüft und es wird festgestellt, dass der Block unverändert (fehlerhaft) geblieben ist.

db_block_checksum kennt folgende Werte:

- **OFF**: Nur im SYSTEM-Tablespace werden Blockchecksummen gebildet.
- **FALSE**: Dieser Wert hat die gleiche Bedeutung wie „OFF“ und wird nur aus Kompatibilitätsgründen bereitgestellt.
- **TYPICAL**: In allen Tablespaces werden Blockchecksummen gebildet und vor jedem Lesevorgang verglichen.
- **FULL**: Wie „TYPICAL“, aber zusätzlich wird vor jedem UPDATE oder DELETE, dass auf einen Block angewandt wird, die Checksumme überprüft und direkt nach der Änderung neu berechnet, nicht erst, wenn der Block auf den Datenträger geschrieben wird.
- **TRUE**: Dieser Wert hat die gleiche Bedeutung wie „TYPICAL“ und wird nur aus Kompatibilitätsgründen bereitgestellt.

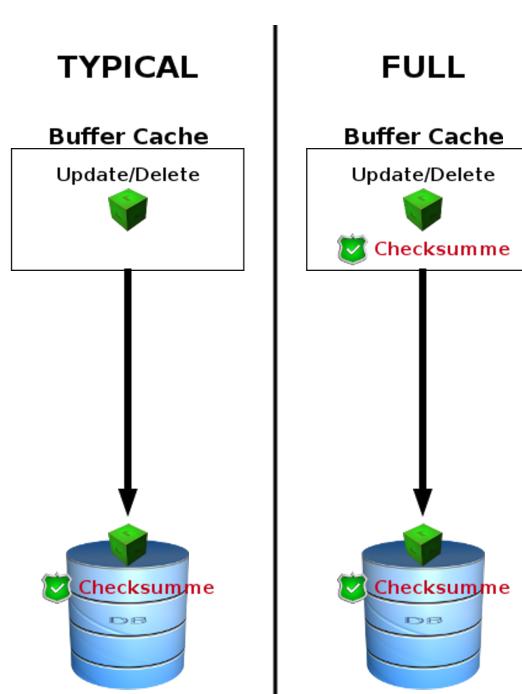


Abb. 31.1:
Der Unterschied
zwischen
TYPICAL und
FULL

Werte für DB_Ultra_Safe

Der Parameter db_ultra_safe kann folgende Werte annehmen:

- **OFF**: Es gelten die Einstellungen der Initialisierungsparameter db_block_checking, db_lost_write_protect und db_block_checksum.
- **DATA_ONLY**: Die Einstellung der Parameter werden verändert:
 - db_block_checking: MEDIUM
 - db_lost_write_protect: TYPICAL
 - db_block_checksum: FULL
- **DATA_AND_INDEX**: Die Einstellung der Parameter werden verändert:
 - db_block_checking: FULL
 - db_lost_write_protect: TYPICAL
 - db_block_checksum: FULL

Es wird von Seiten Oracle empfohlen, db_ultra_safe auf den Wert „DATA_ONLY“ oder höher einzustellen. Da dies kein dynamischer Parameter ist, muss ein Neustart der Datenbank erfolgen.

31.3 Recovery unkritischer Verluste

Bei unkritischen Verlusten handelt es sich um Schäden, welche die Datenbank nicht unmittelbar zum Stillstand bringen. Die Datenbank kann trotz Beschädigung weiterarbeiten, möglicherweise auch nur für begrenzte Zeit. Beispielsweise wird der Verlust eines SPFiles die Datenbank nicht beeinträchtigen, solange kein Neustart erfolgen soll. Auch eine Passworddatei ist unkritisch für den Betrieb der Datenbank. Lediglich die Anmeldung der DBAs wird dadurch beeinträchtigt.

31.3.1 Verlust eines SPFile beheben (mit Recovery Katalog)

Wiederherstellung aus einem Controlfile Autobackup

- Mit dem RMAN an der Zieldatenbank und am Recovery Katalog anmelden.

Listing 31.6: An der Zieldatenbank und am Recovery Katalog anmelden

```
[oracle@FEA11-119SRV ~]$ rman target / catalog catowner/catpass@CATDB
```

- Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.7: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```

Dieser Schritt ist in SQL*Plus nicht möglich!



- Wiederherstellen des SPFiles aus dem Controlfile Autobackup. Dieser Schritt unterscheidet sich, je nachdem, ob die Instanz in der NOMOUNT-Phase oder geöffnet ist.

- NOMOUNT-Phase**

Listing 31.8: Wiederherstellen des SPFiles aus dem Controlfile Autobackup

```
RMAN> RESTORE spfile FROM AUTOBACKUP;
```

- MOUNT-Phase oder geöffnete Instanz**

Bei geöffneter oder gemounteter Instanz muss im Anschluss an diesen Arbeitsschritt, dass wiederhergestellte SPFile in das \$ORACLE_HOME/dbs-Verzeichnis verschoben werden. Dies ist notwendig, da Oracle das (vermeindlich noch vorhandene) SPFile mit einer Schreibsperrre, im Dateisystem belegt.

Listing 31.9: Wiederherstellen des SPFiles aus dem Controlfile Autobackup

```
RMAN> RESTORE spfile
2>   TO '/u01/app/oracle/product/11.2.0/0RCL/spfileorcl.ora'
3>   FROM AUTOBACKUP;
```

- Durchstarten der Instanz

Listing 31.10: Neustart der Instanz

```
RMAN> startup force
```

Nach dem Neustart werden die Initialisierungsparameter gemäß dem wiederhergestellten SPFile gesetzt und müssen auf ihre Aktualität überprüft werden.

Wiederherstellung aus einem Backup Set

1. Mit dem RMAN an der Zieldatenbank und am Recovery Katalog anmelden.

Listing 31.11: An der Zieldatenbank und am Recovery Katalog anmelden

```
[oracle@FEA11 -119SRV ~]$ rman target / catalog catowner/catpass@CATDB
```

2. Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.12: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```



Dieser Schritt ist in SQL*Plus nicht möglich!

3. Wiederherstellen des SPFiles. Dieser Schritt unterscheidet sich, je nachdem, ob die Instanz in der NOMOUNT-Phase oder geöffnet ist.

- **NOMOUNT-Phase**

Listing 31.13: Wiederherstellen des SPFiles in der NOMOUNT-Phase

```
RMAN> RESTORE spfile;
```

- **MOUNT-Phase oder geöffnete Instanz**

Bei geöffneter oder gemounteter Instanz muss im Anschluss an diesen Arbeitsschritt das wiederhergestellte SPFile in das \$ORACLE_HOME/dbs-Verzeichnis verschoben werden. Dies ist notwendig, da Oracle das (vermeindlich noch vorhandene) SPFile mit einer Schreibsperrre, im Dateisystem belegt.

Listing 31.14: Wiederherstellen des SPFiles in der MOUNT-Phase

```
RMAN> RESTORE spfile
2>      TO '/u01/app/oracle/product/11.2.0/spfileorcl.ora';
```

4. Durchstarten der Instanz

Listing 31.15: Neustart der Instanz

```
RMAN> startup force
```

Nach dem Neustart werden die Initialisierungsparameter gemäß dem wiederhergestellten SPFile gesetzt und müssen auf ihre Aktualität überprüft werden.

31.3.2 Verlust eines SPFiles beheben (ohne Recovery Katalog)

Die Datenbank-ID (DBID)

Jede Oracle-Datenbank wird anhand einer zehnstelligen ID, der Datenbank-ID (DBID) identifiziert. Diese ist in der Kontrolldatei gespeichert und wird vom RMAN benötigt, damit dieser erkennen kann, welche Backup Sets zu welcher Datenbank gehören. Ist die Instanz nicht gemountet oder geöffnet, kann der RMAN die DBID nicht selbstständig ermitteln.

Es gibt drei Möglichkeiten, an die DBID zu gelangen:

1. Abfragen der View V\$DATABASE

Listing 31.16: Abfragen von V\$DATABASE

```
SQL> SELECT dbid
  2  FROM v$database;

          DBID
-
1351916467
```

2. Starten des RMAN: Beim Starten des RMAN wird die DBID der Zieldatenbank rechts unten in der Startmeldung angezeigt, sofern die Instanz hochgefahren ist.

Listing 31.17: Startmeldung des RMAN

```
Recovery Manager: Release 11.2.0.1.0 - Production on Tue Oct 29 17:34:34 2013

Copyright (c) 1982, 2009, Oracle and/or its affiliates. All rights reserved.

connected to target database: ORCL (DBID=1351916467)
```

3. Durchsuchen eines unkomprimierten Backups, welches die Datendatei Nummer 1 beinhaltet.

Listing 31.18: Die DBID in einem Backup Set suchen

```
[oracle@FEA11-119SRV ~]$ strings /u02/backup/ORCL_29-10-2013_4aklgp2d.bkp \
> | grep MAXVALUE,
1351916467, MAXVALUE,
```

Das Linux-Kommando `strings` durchsucht eine Binärdatei nach „lesbaren“ Zeichenketten. In einem unkomprimierten und unverschlüsselten Backup Set kann so die DBID sichtbar gemacht werden.

Wiederherstellung aus einem Controlfile Autobackup

- Mit dem RMAN an der Zieldatenbank anmelden.

Listing 31.19: An der Zieldatenbank anmelden

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle
```

- Setzen der DBID

Listing 31.20: Setzen der DBID

```
RMAN> SET DBID 1351916467;
```

- Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.21: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```

Dieser Schritt ist in SQL*Plus nicht möglich!

- Wiederherstellen des SPFiles aus dem Controlfile Autobackup. Dieser Schritt unterscheidet sich, je nachdem, ob die Instanz in der NOMOUNT-Phase oder geöffnet ist.

- NOMOUNT-Phase**

Listing 31.22: Wiederherstellen des SPFiles aus dem Controlfile Autobackup

```
RMAN> RESTORE spfile FROM AUTOBACKUP
2>   RECOVERY AREA '/u05/fast_recovery_area'
3>   DB_NAME      'orcl';
```

Wenn die Instanz nicht gemountet oder geöffnet ist, ist es dem RMAN unmöglich, den Speicherort der Controlfile Autobackups zu ermitteln. Mit Hilfe der beiden Parameter RECOVERY AREA und DB_NAME wird der Speicherort, die Fast Recovery Area, dem RMAN mitgeteilt.

- MOUNT-Phase oder geöffnete Instanz**

Listing 31.23: Wiederherstellen des SPFiles aus dem Controlfile Autobackup

```
RMAN> RESTORE spfile
2>   TO '/u01/app/oracle/product/11.2.0/ORCL/spfileorcl.ora',
3>   FROM AUTOBACKUP;
```

Bei geöffneter oder gemounteter Instanz muss im Anschluss an diesen Arbeitsschritt das wiederhergestellte SPFile in das \$ORACLE_HOME/dbs-Verzeichnis verschoben werden. Dies ist notwendig, da Oracle das (vermeindlich noch vorhandene) SPFile mit einer Schreibsperrre, im Dateisystem belegt.

5. Durchstarten der Instanz

Listing 31.24: Neustart der Instanz

```
RMAN> startup force
```

Nach dem Neustart werden die Initialisierungsparameter gemäß dem wiederhergestellten SPFile gesetzt und müssen auf ihre Aktualität überprüft werden.

Wiederherstellung aus einem Backup Set

1. Mit dem RMAN an der Zieldatenbank anmelden und die DBID setzen.

Listing 31.25: An der Zieldatenbank anmelden und die DBID setzen

```
[oracle@FEA11-119SRV ~]$ rman target /
RMAN> SET DBID 1351916467;
```

2. Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.26: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```



Dieser Schritt ist in SQL*Plus nicht möglich!

3. Wiederherstellen des SPFiles. Dieser Schritt unterscheidet sich, je nachdem, ob die Instanz in der NOMOUNT-Phase oder geöffnet ist.

- **NOMOUNT-Phase**

Listing 31.27: Wiederherstellen des SPFiles in der NOMOUNT-Phase

```
RMAN> RESTORE spfile
2>      FROM  '/u02/backup/3ukkpd6p.bkp';
```

- **MOUNT-Phase oder geöffnete Instanz**

Bei geöffneter oder gemounteter Instanz muss im Anschluss an diesen Arbeitsschritt das wiederhergestellte SPFile in das \$ORACLE_HOME/dbs-Verzeichnis verschoben werden. Dies ist notwendig, da Oracle das (vermeindlich noch vorhandene) SPFile mit einer Schreibsperrre, im Dateisystem belegt.

Listing 31.28: Wiederherstellen des SPFiles in der MOUNT-Phase

```
RMAN> RESTORE spfile
2>   TO '/u01/app/oracle/product/11.2.0/spfileorcl.ora'
3>   FROM '/u02/backup/3ukkpd6p.bkp';
```

4. Durchstarten der Instanz

Listing 31.29: Neustart der Instanz

```
RMAN> startup force
```

Nach dem Neustart werden die Initialisierungsparameter gemäß dem wiederhergestellten SPFile gesetzt und müssen auf ihre Aktualität überprüft werden.

Wiederherstellen eines PFiles

Da ein PFile nicht automatisch durch den Datenbankserver verwaltet wird, gibt es auch keinen datenbankeigenen Backupmechanismus. Für die Sicherung und Wiederherstellung eines PFiles muss manuell gesorgt werden. Es empfiehlt sich daher, bei Benutzung eines PFiles, dieses durch Betriebssystemmittel vor Verlust zu schützen.



Schon seit Oracle 9i wird davon abgeraten, ein PFile zu benutzen.

31.3.3 Verlust einer Passworddatei beheben

Die Passworddatei gehört zu den Dateien, die nicht durch RMAN gesichert werden können. Daher sollte auch diese auf anderem Wege gesichert werden. Geht die Passworddatei verloren und es existiert kein Backup, muss sie wie neu erstellt werden.

31.3.4 Verlust von Konfigurationsdateien beheben

Für Konfigurationsdateien wie, z. B. die `tnsnames.ora`, `listener.ora` oder die Datei `sqlnet.ora` gilt Gleichtes, wie für die Passworddatei. Bei einem Verlust müssen sie neu angelegt werden und sollten daher auf jeden Fall gesichert werden.

31.3.5 Verlust eines Tempfiles beheben

Geht im laufenden Betrieb der Datenbank ein Tempfile verloren, kann bei der Ausführung eines SQL-Statements folgende Fehlermeldung auftreten:

Listing 31.30: Verlust eines Tempfiles

```
ORA-01565: error in identifying file
'/u02/oradata/ORCL/temp01.dbf'
ORA-27037: unable to obtain file status
Linux Error: 2: No such file or directory
```

Es gibt zwei Möglichkeiten auf diese Situation zu reagieren:

- Neustarten der Instanz. Hierbei wird das Tempfile automatisch neu erstellt. In der Alert Log Datei wird eine Meldung ausgegeben, ähnlich dieser:

Listing 31.31: Automatisches Neuerstellen eines Tempfiles

```
Recreating tempfile
'/u02/oradata/ORCL/temp01.dbf'
```

- Erstellen eines neuen Tempfiles und löschen der verlorenen Datei aus dem Data Dictionary.

1. Ermitteln zu welchem Temp-Tablespace die beschädigte Datei gehört.

Listing 31.32: Ermitteln des richtigen Temp-Tablespace

```
SQL> SELECT tablespace_name
  2  FROM    dba_temp_files
  3  WHERE   file_name LIKE '/u02/oradata/ORCL/temp01.dbf'

TABLESPACE_NAME
-----
TEMP
```

2. Neues Tempfile erstellen

Listing 31.33: Neues Tempfile erstellen

```
SQL> ALTER TABLESPACE temp
  2 ADD TEMPFILE '/u02/oradata/0RCL/temp02.dbf'
  3 SIZE 20 M AUTOEXTEND ON MAXSIZE 500M;
```

3. Das beschädigte Tempfile löschen.

Listing 31.34: Beschädigtes Tempfile löschen

```
SQL> ALTER TABLESPACE temp
  2 DROP TEMPFILE '/u02/oradata/0RCL/temp01.dbf';
```

31.4 Datafile Media Recovery

Ein Media Recovery kümmert sich immer um die Behebung von Datenbankfehlern, die durch den Verlust einer der folgenden Dateiarten zustande gekommen sind:

- Datendateien
- Kontrolldatei
- Archive Log Dateien

Archive Log Dateien bilden dabei eine Ausnahme, da sie meist nur temporär für ein Recovery benötigt werden, müssen sie nicht zwingend sofort wiederhergestellt werden, sondern es genügt eine funktionsfähige Kopie dieser Dateien zu besitzen.

31.4.1 Identifizieren beschädigter Dateien

Um zu analysieren welche Dateien ein Recovery brauchen, ist Folgendes notwendig:

1. Starten von SQL*Plus
2. Status der Instanz abfragen

Listing 31.35: Status der Instanz abfragen

```
SQL> SELECT status FROM v$instance;

STATUS
-----
```

OPEN

Auch wenn der Status „OPEN“ angezeigt wird, ist es möglich, dass Teile der Datenbank ein Recovery benötigen.

3. Damit die folgenden Abfragen sinnvolle Ergebnisse liefern, muss zuerst ein Checkpoint stattfinden, da erst dann der Status der Datendateien aktualisiert wird.

Listing 31.36: Einen Checkpoint absetzen

SQL> ALTER SYSTEM CHECKPOINT;

4. Den Dateistatus abfragen. Hierfür gibt es mehrere Möglichkeiten:

- a) Abfragen der View V\$DATAFILE_HEADER

Jede zurückgegebene Zeile steht stellvertretend für eine Datendatei. Die Spalte RECOVER zeigt an, ob eine Datendatei ein Recovery benötigt oder nicht. In der Spalte ERROR wird angezeigt, ob ein Problem mit der Datendatei vorliegt (z. B. ob der Datendateiheader nicht gelesen werden konnte).

Listing 31.37: Status der Datendateien abfragen mit V\$DATAFILE_HEADER

SQL> col file# format 999 SQL> col status format A7 SQL> col error format A10 SQL> col tablespace_name format A10 SQL> col name format A30 SQL> SELECT file#, status, error, recover, tablespace_name, name 2 FROM v\$datafile_header 3 WHERE recover = 'YES' 4 OR (recover IS NULL 5 AND error IS NOT NULL);
FILE# STATUS ERROR RECOVER TABLESPACE NAME

6 OFFLINE FILE NOT FOUND

Zeigt die Spalte ERROR einen Wert an, sollte zuerst nach Hardware- oder Betriebssystemproblemen gesucht werden. Liegen keine derartigen Probleme vor, muss die betreffende Datendatei wiederhergestellt werden.

Zeigt die Spalte RECOVER den Wert YES und die Spalte ERROR ist leer, so ist meist nur ein Recovery der Datendatei notwendig. Da diese View aber nur die Header der Datendateien liest, kann sie nicht alle Probleme anzeigen, die ein Recovery der Datendatei notwendig machen!



Die Anzeige defekter Datendateien funktioniert nur dann zuverlässig, wenn der Parameter `filesystemio_options` den Wert `directIO` hat. Diese Einstellung sorgt dafür, dass Informationen direkt auf den Datenträger geschrieben und Caching-Mechanismen des Dateisystems umgangen werden.

b) Abfragen der View V\$RECOVER_FILE

Listing 31.38: Status der Datendateien abfragen mit V\$RECOVER_FILE

FILE#	ERROR	ONLINE_	CHANGE#	TIME
6	FILE NOT FOUND	OFFLINE		0

Zusätzlich zu V\$DATAFILE_HEADER zeigt sie an, dass eine Datendatei defekte Blöcke enthält. Im Gegensatz zu V\$DATAFILE_HEADER kann sie nur dann verwendet werden, wenn die Kontrolldatei nicht wiederhergestellt werden musste.

c) Abfragen der Views V\$DATAFILE und V\$TABLESPACE

Listing 31.39: Status der Datendateien abfragen

```

col df#      format 999
col df_name   format A35
col tbsp_name format A7
col status     format A7
col error      format A10
col change#    format 99999999

SELECT r.file# AS df#, d.name AS df_name, t.name AS tbsp_name,
       d.status, r.error
  FROM v$recover_file r, v$datafile d, v$tablespace t
 WHERE t.ts# = d.ts#
   AND d.file# = r.file#;

DF# DF_NAME                                TBSP_NA STATUS  ERROR
----- ----- ----- ----- -----
6 /u02/oradata/ORCL/example01.dbf          EXAMPLE RECOVER FILE NOT
                                            FOUND

```

31.4.2 Ermitteln der benötigten Backups

RMAN bietet mit dem Kommando `RESTORE PREVIEW` die Möglichkeit festzustellen, welche Backups für einen Restore-Prozess benötigt werden. Dies ist gerade dann wichtig, wenn Backup Sets auf SBT-Tapes liegen und erst wieder in die FRA zurückgeholt werden müssen. Im folgenden Beispiel wird angenommen, dass der Tablespace BANK vollständig wiederhergestellt werden muss.

Listing 31.40: Vorschau auf ein Restore

```

RMAN> RESTORE PREVIEW tablespace bank;

Starting restore at 30-OCT-13
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_SBT_TAPE_1

List of Backup Sets
=====
BS Key  Type LV Size
----- - - - -
16      Full   26.34M

```

```

List of Datafiles in backup set 16
File LV Type Ckp SCN      Ckp Time   Name
-----
6       Full 2084386    29-OCT-13 /u01/app/oracle/oradata/orcl/bank01.dbf

...
BS Key  Size      Device Type Elapsed Time Completion Time
-----
30      240.25M   SBT_TAPE 00:00:03   30-OCT-13
        BP Key: 48   Status: AVAILABLE Compressed: NO Tag: TAG20131030T111928
        Handle: 18onmqq1_1_1 Media: /u04,18onmqq1_1_1

List of Archived Logs in backup set 30
Thrd Seq      Low SCN     Low Time   Next SCN   Next Time
-----
1   56        2085711   29-OCT-13 2086381   29-OCT-13
1   57        2086381   29-OCT-13 2111298   30-OCT-13
1   58        2111298   30-OCT-13 2111915   30-OCT-13
1   59        2111915   30-OCT-13 2112126   30-OCT-13
validation succeeded for backup piece
Media recovery start SCN is 2084386
Recovery must be done beyond SCN 2084393 to clear datafile fuzziness
validation succeeded for backup piece
Finished restore at 30-OCT-13

```

Die Ausgabe des `RESTORE PREVIEW`-Befehls zeigt eine komplette Auflistung aller Backup Sets und Copies, die der RMAN für den angegebenen Restore-Prozess heranziehen würde. Besonderes Augenmerk muss auf die Backup Sets gelegt werden, die auf SBT-Tapes gespeichert sind, wie hier z. B. Backup Set Nummer 30. Dessen Inhalt sollte vor dem Restore des Tablespace in die FRA zurückgeholt werden.

31.4.3 Wiedherstellen der gesamten Datenbank



Sollte die Datenbank eines oder mehrere verschlüsselte Elemente (Tabellenspalten, Tablespace, o. ä.) enthalten, muss vor dem Recovery das Encryption Wallet geöffnet werden. ?? zeigt wie ein verschlüsselter Tablespace recovered wird.

Alle benötigten Dateien befinden sich in der FRA

Das Wiederherstellen der gesamten Datenbank ist der mit Sicherheit zeitaufwendigste Recovery-Fall der entstehen kann. Ob und wann die gesamte Datenbank wiederherzustellen ist, kann von verschiedenen Faktoren abgeleitet werden. Die folgende Liste zeigt nur einige dieser Gründe:

- Es wurden mehr als 70 % aller Datendateien zerstört.
- Die Datenbank muss auf einen anderen Server umgezogen werden.

Im Folgenden wird ein einfaches vollständiges Restore and Recovery einer ganzen Datenbank, ohne Recovery Katalog durchgeführt. Die Kontrolldatei, das SPFile und mindestens eine gespiegelte Kopie der Redo Logs der Datenbank sind noch erhalten. Des Weiteren befinden sich alle benötigten Archive Logs in der FRA.

1. Mit dem RMAN an der Zieldatenbank anmelden

Listing 31.41: An der Zieldatenbank anmelden

```
[oracle@FEA11-119SRV ~]$ rman target /
```

2. Die Datenbank in den Mount-Status bringen

Listing 31.42: Datenbank mounten

```
RMAN> startup force mount
```

3. Das Restore durchführen

Listing 31.43: Restore der Datendateien

```
RMAN> RESTORE database;
```

4. Recovern der Datenbank

Listing 31.44: Recovery der Datenbank

```
RMAN> RECOVER database;
```

5. Öffnen der Datenbank

Listing 31.45: Öffnen der Datenbank nach dem Recovery

```
RMAN> SQL 'ALTER DATABASE OPEN';
```



Da in diesem Fall die Kontrolldatei der Zieldatenbank erhalten blieb, gibt es keinen Unterschied zwischen einem Recovery mit Hilfe eines Recovery Katalogs und einem Recovery ohne Nutzung des Recovery Katalogs.

Archive Logs müssen von SBT zurückgeholt werden

Das vorangegangene Beispiel wird nun dahingehend abgewandelt, dass nicht alle benötigten Archive Logs in der FRA vorliegen.

1. Mit dem RMAN an der Zieldatenbank anmelden

Listing 31.46: An der Zieldatenbank anmelden

```
[oracle@FEA11 -119SRV ~]$ rman target /
```

2. Die Datenbank in den Mount-Status bringen

Listing 31.47: Datenbank mounten

```
RMAN> startup force mount
```

3. Ermitteln welche Backup Sets auf SBT-Tape liegen.

Listing 31.48: Ermitteln der benötigten Dateien

```
RMAN> RESTORE PREVIEW database;
```

4. Zurückholen der benötigten Archive Logs.

Listing 31.49: Restore der Archive Logs von SBT in die FRA

```
RMAN> RESTORE archive log UNTIL SEQUENCE 60;
```

5. Das Restore der Datenbank durchführen

Listing 31.50: Restore der Datendateien

```
RMAN> RESTORE database;
```

6. Recovern der Datenbank

Listing 31.51: Recovery der Datenbank

```
RMAN> RECOVER database;
```

7. Öffnen der Datenbank

Listing 31.52: Öffnen der Datenbank nach dem Recovery

```
RMAN> SQL 'ALTER DATABASE OPEN';
```

8. Löschen der nicht mehr benötigten Archive Logs.

Listing 31.53: Löschen der Archive Logs, die bereits auf SBT-Tape gesichert wurden

```
RMAN> DELETE archivelog all BACKED UP 1 TIMES  
2> TO DEVICE TYPE sbt;
```

Listing 31.54: Löschen der Archive Logs, die bereits auf Disk gesichert wurden

```
RMAN> DELETE archivelog all BACKED UP 1 TIMES  
2> TO DEVICE TYPE disk;
```

Durch die Klausel `BACKUPED UP 1 TIMES` wird beim Löschen sichergestellt, dass nur solche Archive Logs aus der FRA entfernt werden, die bereits mindestens 1x gesichert worden sind.

31.4.4 Wiederherstellung eines Tablespaces

Wenn der Tablespace unverschlüsselt ist

Die Wiederherstellung eines Tablespaces kann sowohl im MOUNT- als auch im OPEN-Status der Datenbank durchgeführt werden. Im folgenden Beispiel wird angenommen, dass der Tablespace BANK beschädigt wurde, während die Datenbank geöffnet ist. Wie schon beim Recovery der gesamten Datenbank, so muss auch hier erst geprüft werden, ob alle benötigten Dateien in der FRA vorliegen.

1. Mit dem RMAN an der Zieldatenbank anmelden

Listing 31.55: An der Zieldatenbank anmelden

```
[oracle@FEA11-119SRV ~]$ rman target /
```

2. Überprüfen ob alle benötigten Dateien in der FRA vorliegen

Listing 31.56: Voraussetzungen überprüfen

```
RMAN> RESTORE PREVIEW tablespace bank;
```

3. Nicht vorliegende Archive Logs von SBT-Tapes holen

Listing 31.57: Voraussetzungen überprüfen

```
RMAN> RESTORE archive log UNTIL SEQUENCE 60;
```

4. Offline setzen des betreffenden Tablespaces

Listing 31.58: Betreffenden Tablespace Offline setzen

```
RMAN> SQL 'ALTER TABLESPACE bank OFFLINE IMMEDIATE';
```

5. Das Restore des Tablespaces durchführen

Listing 31.59: Restore des betreffenden Tablespaces

```
RMAN> RESTORE tablespace bank;
```

6. Recovern des Tablespaces

Listing 31.60: Recovery des Tablespaces

```
RMAN> RECOVER tablespace bank DELETE archivelog;
```

Die Klausel `DELETE archivelog` sorgt dafür, dass alle nicht mehr benötigten Archive Logs sofort gelöscht werden. Dabei werden keine Logs gelöscht, die noch nicht gesichert wurden.

7. Online setzen des betreffenden Tablespaces nach dem Recovery

Listing 31.61: Betreffenden Tablespace Online setzen

```
RMAN> SQL 'ALTER TABLESPACE bank ONLINE';
```

Wenn der Tablespace verschlüsselt ist

Bei einem verschlüsselten Tablespace sind grundsätzlich die gleichen Schritte für ein Restore and Recovery notwendig, wie bei einem Unverschlüsselten. Der einzige Unterschied ist, dass ein kryptierter Tablespace nur dann recovered werden kann, wenn das Wallet vorher geöffnet wurde.

1. Mit dem RMAN an der Zieldatenbank anmelden

Listing 31.62: An der Zieldatenbank anmelden

```
[oracle@FEA11-119SRV ~]$ rman target /
```

2. Das Wallet öffnen

Listing 31.63: Öffnen des Wallets

```
RMAN> SQL 'ALTER SYSTEM SET ENCRYPTION WALLET OPEN IDENTIFIED BY "P@ssw0rd";'
```

3. Überprüfen ob alle benötigten Dateien in der FRA vorliegen

Listing 31.64: Voraussetzungen überprüfen

```
RMAN> RESTORE PREVIEW tablespace bank;
```

4. Nicht vorliegende Archive Logs von SBT-Tapes holen

Listing 31.65: Voraussetzungen überprüfen

```
RMAN> RESTORE archive log UNTIL SEQUENCE 60;
```

5. Offline setzen des betreffenden Tablespaces

Listing 31.66: Betreffenden Tablespace Offline setzen

```
RMAN> SQL 'ALTER TABLESPACE bank OFFLINE IMMEDIATE';
```

6. Das Restore des Tablespaces durchführen

Listing 31.67: Restore des betreffenden Tablespaces

```
RMAN> RESTORE tablespace bank;
```

7. Recovern des Tablespaces

Listing 31.68: Recovery des Tablespaces

```
RMAN> RECOVER tablespace bank DELETE archivelog;
```

Die Klausel `DELETE archivelog` sorgt dafür, dass alle nicht mehr benötigten Archive Logs sofort gelöscht werden. Dabei werden keine Logs gelöscht, die noch nicht gesichert wurden.

8. Online setzen des betreffenden Tablespaces nach dem Recovery

Listing 31.69: Betreffenden Tablespace Online setzen

```
RMAN> SQL 'ALTER TABLESPACE bank ONLINE';
```

9. Das Wallet schließen

Listing 31.70: Schließen des Wallets

```
RMAN> SQL 'ALTER SYSTEM SET ENCRYPTION WALLET CLOSE IDENTIFIED BY "P@ssw0rd"';
```

31.4.5 Wiederherstellen einzelner Datendateien

Sind nur einzelne Datendateien beschädigt, ist es meist wirtschaftlicher, explizit die beschädigten Dateien wiederherzustellen und nicht die gesamten Tablespaces.

Datendateien Offline setzen

Datendateien können genauso wie Tablespaces Offline gesetzt werden. Wenn dies geschieht, ist der Tablespace der sie enthält nicht verfügbar, bis die Datendatei wieder Online gesetzt wurde.

Um eine Datendatei Offline zu setzen, wird das `ALTER DATABASE`-Kommando und das `alter database`-System Privileg benötigt.

Listing 31.71: Eine Datendatei offline setzen

```
SQL> ALTER DATABASE
  2  DATAFILE '/u02/oradata/orcl/bank01.dbf' OFFLINE;
```

Eine Datendatei kann nicht nur mit ihrem Dateinamen, sondern auch mit ihrer internen Dateinummer ange- sprochen werden. Herauszufinden kann man die Dateinummer mit Hilfe der View `DBA_DATA_FILES`.

Listing 31.72: Herauszufinden der file_id einer Datendatei

```
SQL> SELECT file_id
  2  FROM dba_data_files
  3  WHERE file_name LIKE '%bank01%';

FILE_ID
-----
6
```

Im `ALTER DATABASE`-Kommando wird der Dateiname durch die Dateinummer ersetzt.

Listing 31.73: Eine Datendatei mit Hilfe der file_id Offline setzen

```
SQL> ALTER DATABASE
  2  DATAFILE 6 OFFLINE;
```

Soll die Datendatei wieder Online gebracht werden, wird das `ALTER DATABASE`-Statement zusammen mit dem Schlüsselwort `ONLINE` verwendet.

Listing 31.74: Eine Datendatei online setzen

```
SQL> ALTER DATABASE
  2  DATAFILE 6 ONLINE;
```

Restore and Recovery von Datendateien

Im folgenden Beispiel wird angenommen, dass die Datenbank geöffnet ist.

1. Mit dem RMAN an der Zieldatenbank anmelden

Listing 31.75: An der Zieldatenbank anmelden

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle
```

2. Offline setzen der betreffenden Datendatei

Listing 31.76: Betreffende Datendatei Offline setzen

```
RMAN> SQL 'ALTER DATABASE DATAFILE 6 OFFLINE';
```

3. Das Restore der Datendatei durchführen

Listing 31.77: Restore der betreffenden Datendatei

```
RMAN> RESTORE datafile 6;
```

4. Recovern der Datendatei

Listing 31.78: Recovery der Datendatei

```
RMAN> RECOVER datafile 6 DELETE archivelog;
```

5. Online setzen der Datendatei nach dem Recovery

Listing 31.79: Betreffende Datendatei Online setzen

```
RMAN> SQL 'ALTER DATABASE DATAFILE 6 ONLINE';
```

Restore and Recovery einer Datendatei an einem neuen Speicherort

Um eine Datendatei an einem neuen Speicherort wiederherzustellen, muss dieser in der Kontrolldatei angegeben werden. RMAN bietet für das Angeben des neuen Speicherorts das `SET NEWNAME`-Kommando und `SWITCH DATAFILE` zum Umschalten auf den neuen Speicherort.

Die Kombination der beiden RMAN-Kommandos `SET NEWNAME` und `SWITCH DATAFILE` ist vergleichbar mit dem SQL-Kommando `ALTER DATABASE RENAME FILE`.

Im folgenden Beispiel wird davon ausgegangen, dass der Tablespace `BANK` beschädigt wurde. Eine der beiden Datendateien dieses Tablespace, muss an einem anderen Speicherort wiederhergestellt werden, da der Originalspeicherort derzeit nicht verfügbar ist. Mit Hilfe der unter ?? beschriebenen Methode wurde herausgefunden, dass die Datendatei `bank02.dbf` die Nummer 7 hat.

1. Mit dem RMAN an der Zieldatenbank anmelden

Listing 31.80: An der Zieldatenbank anmelden

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle
```

2. Offline setzen der betreffenden Datendatei

Listing 31.81: Betreffende Datendatei Offline setzen

```
RMAN> SQL ,ALTER TABLESPACE bank OFFLINE TEMPORARY ;
```

3. Öffnen eines RUN-Blocks

Listing 31.82: Einen RUN-Block öffnen

```
RMAN> RUN {
```

4. Neuen Speicherort der Datendatei festlegen

Listing 31.83: Neuen Speicherort festlegen

```
2>      SET NEWNAME FOR datafile 7 TO '/u02/oradata/0RCL/bank02.dbf' ;
```

5. Durchführen des Restores

Listing 31.84: Restore durchführen

```
3>      RESTORE datafile 7 ;
```

6. Umschalten auf den neuen Speicherort der Datendatei

Listing 31.85: Auf den neuen Speicherort umschalten

```
4>      SWITCH DATAFILE ALL ;
```

7. Recovery der Datendatei durchführen und den RUN-Block schließen.

Listing 31.86: Recovery durchführen

```
5>      RECOVER datafile 7 ;  
6>    }
```

8. Datendatei in den ONLINE-Status bringen

Listing 31.87: Datendatei in den ONLINE-Status bringen

```
RMAN> SQL ,ALTER TABLESPACE bank ONLINE ;
```

31.4.6 Wiederherstellen von Archive Logs

Benötigt der RMAN Archive Logs für ein Recovery, stellt er diese selbstständig wieder her. In manchen Fällen ist es jedoch notwendig, Archive Logs manuell wiederherzustellen. Im Folgenden werden zwei Szenarios beschrieben, bei denen die Archive Logs manuell wiederherzustellen sind.

Die Archive Logs an einem neuen Speicherort wiederherstellen

Standardmäßig geben `log_archive_format` und `log_archive_dest_n` vor, unter welchem Namen und an welchem Speicherort die Archive Logs wiederhergestellt werden. Wenn es sinnvoll ist, kann mit dem `SET ARCHIVELOG DESTINATION`-Kommando des RMAN ein anderer Ort definiert werden.

1. Öffnen eines RUN-Blocks

Listing 31.88: Einen RUN-Block öffnen

```
RMAN> RUN {
```

2. Neuen Speicherort der Archive Logs festlegen

Listing 31.89: Neuen Speicherort festlegen

```
2>      SET ARCHIVELOG DESTINATION TO '/tmp';
```

3. Wiederherstellen der Archive Logs und schließen des RUN-Blockes.

Listing 31.90: Wiederherstellen der Archive Logs

```
3>      RESTORE archivelog ALL;
4> }
```

Archive Logs auf mehrere Speicherorte verteilen

Es ist möglich die Menge der Archive Logs auf mehrere Orte aufzuteilen, falls dies aus Speicherplatzgründen notwendig ist. Das folgende Beispiel stellt 150 Archive Logs wieder her und verteilt diese auf zwei Locations.

Listing 31.91: Wiederherstellen der Archive Logs an verschiedenen Speicherorten

```
RMAN> RUN {
2>      SET ARCHIVELOG DESTINATION TO '/u02/backup';
3>      RESTORE ARCHIVELOG FROM SEQUENCE 1 UNTIL SEQUENCE 75;
4>
5>      SET ARCHIVELOG DESTINATION TO '/u03/backup';
6>      RESTORE ARCHIVELOG FROM SEQUENCE 76 UNTIL SEQUENCE 150;
7> }
```

Wird innerhalb dieses RUN-Blocks ein Recovery durchgeführt, findet RMAN automatisch seine Archive Logs an den unterschiedlichen Speicherorten.



Durch den Befehl `SET` getätigte Einstellungen gelten immer nur innerhalb des RUN-Blockes, in dem Sie vorgenommen wurden.

31.5 Block Media Recovery

Es kann vorkommen, dass durch einen Fehler, einzelne Blöcke in einer Datendatei beschädigt werden. Um solche Schäden zu beheben, ist es zum einen möglich, die gesamte Datendatei wiederherzustellen oder aber nur die defekten Blöcke zu recovern. Werden nur einzelne Blöcke einer Datendatei repariert, spricht man von „Block Media Recovery“.

Block Media Recovery hat dem Datafile Media Recovery (Recovery einer ganzen Datendatei) gegenüber folgende Vorteile:

- Die Zeit für das Instance Recovery verringert sich, da nur die wiederhergestellten Blöcke dem Instance Recovery unterzogen werden müssen.
- Die betroffene Datendatei kann während eines Block Media Recovery Online bleiben.

Das Block Media Recovery unterliegt aber auch einer ganzen Reihe von Einschränkungen, die beachtet werden müssen:

- Block Media Recovery kann nur aus dem RMAN heraus gesteuert werden.
- Ein einmal begonnenes Block Media Recovery muss vollständig durchgeführt werden.
- Für Block Media Recovery können nur Full-Backups herangezogen werden, keine inkrementellen Backups.
- Es können nur solche Blöcke mit Block Media Recovery repariert werden, die als corrupt¹ markiert wurden. In `V$DATABASE_BLOCK_CORRUPTION` kann ersehen werden, welche Blöcke dies sind. Diese View wird aber nicht automatisch befüllt. Die betroffene Datendatei muss erst mit `VALIDATE` validiert werden.



Block Media Recovery ist nur in der Enterprise Version von Oracle 11g möglich!

¹corrupt = engl. fehlerhaft

31.5.1 Wann sollte Block Media Recovery angewendet werden?

Block Media Recovery stellt eine Erweiterung zum Datafile Media Recovery dar. In Fällen, in denen die Anzahl der defekten Blöcke sehr hoch ist, kann Block Media Recovery das Datafile Media Recovery nicht ersetzen.

Das ein defekter Block vorliegt, kann an der Fehlermeldung ORA-01578 festgestellt werden. Hier ein Beispiel für eine solche Fehlermeldung:

Listing 31.92: Der Fehler ORA-01578

```
ORA-01578: ORACLE data block corrupted (file # 7, block # 3)
ORA-01110: data file 7: '/u02/oradata/orcl/bank02.dbf'
ORA-01578: ORACLE data block corrupted (file # 7, block # 235)
ORA-01110: data file 2: '/u02/oradata/orcl/bank02.dbf'
```

31.5.2 Block Media Recovery und die Redo Logs

Für Datafile Media Recovery ist immer eine ununterbrochene Kette von Archived Logs notwendig, da das Recovery sonst nicht erfolgreich sein kann. Block Media Recovery kann unter Umständen auch dann noch erfolgreich sein, wenn einzelne Archived Logs verloren gegangen sind. Die einzige Bedingung für Block Media Recovery ist, dass alle Archived Logs, die Informationen über die wiederherzustellenden Blöcke enthalten, vorhanden sein müssen.

Wenn der RMAN das Fehlen eines Archived Logs feststellt, bricht er seine Arbeit nicht sofort ab. Es kann vorkommen, dass ein Datenblock in einem späteren Archive Log als „newed block“ geführt wird. Dies geschieht beispielsweise dann, wenn alle Zeilen einer Tabelle gelöscht wurden oder die gesamte Tabelle aus dem Block gelöscht wurde. In so einem Fall formatiert Oracle den Block neu, wodurch alle alten Redo Informationen irrelevant werden.

31.5.3 Backups defekter Datendateien anfertigen

Bevor ein Block Media Recovery durchgeführt wird, sollte zuerst ein Backup der betroffenen Datendatei(en) gemacht werden. Standardmäßig weigert sich der RMAN eine Datendatei mit beschädigten Blöcken zu sichern. Dies resultiert aus der Standardeinstellung des RMAN-Parameters `MAXCORRUPT` mit dem Wert 0. Dieser Parameter kann so konfiguriert werden, dass eine bestimmte Anzahl defekter Blöcke für eine Datendatei zulässig ist.

Listing 31.93: Hotbackup einer defekten Datendatei anfertigen

```
RMAN> RUN {
```

```
2>      SET MAXCORRUPT FOR datafile 6 TO 999;
3>      BACKUP AS BACKUPSET DATAFILE 6;
4>  }
```

31.5.4 Block Media Recovery durchführen

Das RMAN-Kommando `BLOCKRECOVER` kann einzelne Datenblöcke in der Datenbank wiederherstellen. Das folgende Szenario zeigt eine einfache Nutzung dieses Kommandos.

1. Zwei defekte Blöcke werden angezeigt.

Listing 31.94: Der Fehler ORA-01578

```
ORA-01578: ORACLE data block corrupted (file # 7, block # 3)
ORA-01110: data file 7: '/u02/oradata/orcl/bank02.dbf'
ORA-01578: ORACLE data block corrupted (file # 7, block # 235)
ORA-01110: data file 2: '/u02/oradata/orcl/bank02.dbf'
```

2. Die Blöcke können recovered werden, während die Datenbank geöffnet ist.

Listing 31.95: Das Kommando BLOCKRECOVER

```
RMAN> BLOCKRECOVER datafile 7 block 3
2>     datafile 2 block 235;
```

Ist die Liste der defekten Blöcke sehr groß, kann das folgende Kommando genutzt werden, um alle korrupten Blöcke in einem Arbeitsgang zu recovern:

Listing 31.96: Das Kommando BLOCKRECOVER

```
RMAN> BLOCKRECOVER CORRUPTION LIST;
```



- [i1016424]

31.6 Unvollständiges Recovery (Point-In-Time Recovery)

Point-In-Time Recovery versetzt die Datenbank in einen Stand zurück, der zeitlich vor dem Aktuellen liegt. Diese Art des Recovery wird auch *unvollständiges Recovery* genannt, da nicht alle vorhandenen Archive Logs und inkrementellen Backups zur Wiederherstellung der Datenbank benutzt werden. Ein Point-In-Time-Recovery kann gezielt erfolgen oder unfreiwillig notwendig werden, da bestimmte Teile der Datenbank verloren gegangen sind.

31.6.1 Die Datenbank im Verlauf der Zeit

Ist Datenverlust für eine Datenbank inakzeptabel, müssen zwei Dinge sichergestellt werden:

1. Die Datenbank muss sich im ARCHIVELOG Modus befinden.
2. Es müssen regelmässig Backups der Datenbank angefertigt werden.

Wenn diese beiden Bedingungen erfüllt sind, könnte ein Ausschnitt aus dem Lebenszyklus einer Datenbank wie folgt aussehen:

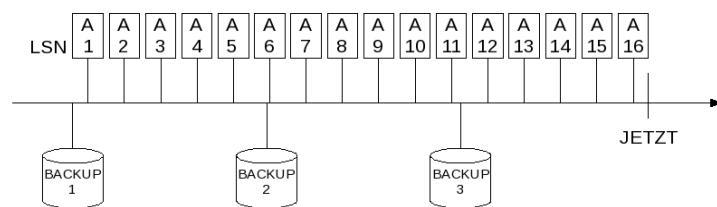


Abb. 31.2:
Lebenszyklus
einer Datenbank
erster Teil

Abbildung ?? zeigt eine Datenbank für die die drei Backups, Backup 1, Backup 2 und Backup 3 existieren. Desweitern sind insgesamt 16 Archive Logs mit der Logsequenz Nummern 1 bis 16 vorhanden. In diesem Szenario kann die Datenbank auf jeden beliebigen Zeitpunkt zwischen der Erstellung von Backup 1 und JETZT zurückgesetzt werden.

In der folgenden Abbildung ?? wird der Fall dargestellt, dass zum Zeitpunkt X fälschlicher Weise eine Tabelle gelöscht wurde, die unbedingt wiederhergestellt werden muss.

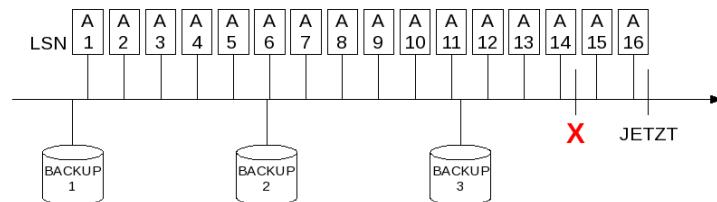


Abb. 31.3:
Lebenszyklus
einer Datenbank
zweiter Teil

Um den Verlust der Tabelle wieder auszugleichen, muss die Datenbank auf einen Zeitpunkt zurückgesetzt werden, der vor dem Zeitpunkt X liegt.

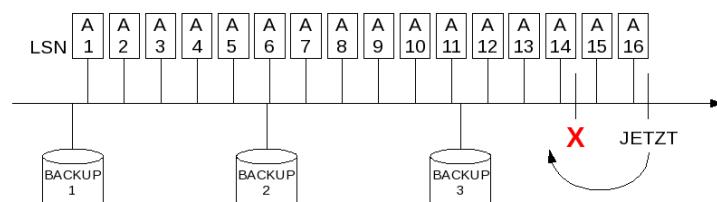


Abb. 31.4:
Lebenszyklus
einer Datenbank
dritter Teil

Um dieses Ziel zu erreichen, müssen folgende Schritte durchgeführt werden:

1. RESTORE: Wiederherstellen von Backup Nummer 3
2. RECOVER: Benutzen der Archive Logs Nummer 12, 13, 14 und von Teilen des Archive Log Nummer 15 um sich an den Zeitpunkt (X - n) heranzutasten.
3. Öffnen der Datenbank

Die Folgen dieses Szenarios wären:

- Die Datenbank wurde zurückgesetzt auf die Logsequenz Nummer 14.
- Der Inhalt der aktuellen Redo Log Datei, mit der Logsequenz Nummer 17 ist unbrauchbar.

Daraus ergibt sich Folgendes:

1. Die Redo Log Dateien müssen geleert/formatiert werden, da ihr Inhalt unbrauchbar ist. Die Redo Logs werden im weiteren Verlauf mit neuem Inhalt gefüllt.
2. Bei einem Logswitch würde erneut eine Archive Log Datei mit der Nummer 15 erzeugt werden, die es aber bereits gibt.

Um die Problematik der Erzeugung von Archive Log Dateien mit gleicher Log Sequenze Nummer (in diesem Beispiel LSN 15) zu vermeiden, hat Oracle den Begriff der „Inkarnation“ eingeführt.



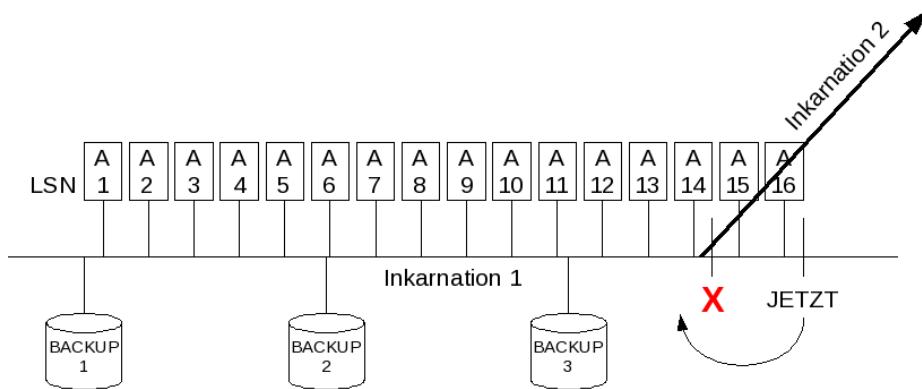
Unter einer Inkarnation versteht Oracle den Lebenszyklus einer Datenbank. Der Anfang eines Lebenszykluses wird durch den Zeitpunkt markiert, an dem die Logsequenz Nummer 1 vergeben wurde. Das Ende wird durch ein Point-In-Time-Recovery markiert.

Mit diesen neuen Erkenntnissen ergibt sich nach dem Recovery der Datenbank, aus dem obigen Beispiel, ein anderes Bild:

Durch das Öffnen der Datenbank und das Zurücksetzen der Logsequenz Nummer auf 1 wird ein neuer Lebenszyklus der Datenbank begonnen, also eine neue Inkarnation.



Abb. 31.5:
Lebenszyklus
einer Datenbank
vierter Teil



Eine neue Datenbankinkarnation wird immer dann erstellt, wenn beim Öffnen der Datenbank das Kommando `ALTER DATABASE OPEN RESETLOGS` verwendet wird. Die Zählung der Inkarnationen beginnt bei 1 und wird fortlaufend durchgeführt.

Obwohl in dieser Situation jetzt zwei Archive Log Dateien mit der Logsequenz Nummer 1 existieren, kann die Datenbank diese Beiden, anhand der Inkarnationsnummer 1 bzw. 2 auseinanderhalten.

Um sehen zu können, welche Datenbankinkarnationen existieren, kann im RMAN das Kommando `LIST INCARNATION` oder in SQL*Plus die View `V$DATABASE_INCARNATION` benutzt werden.

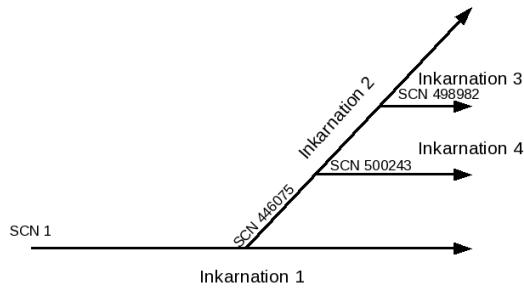
Listing 31.97: Die View v\$database_incarnation

```
SQL> SELECT incarnation#, resetlogs_change#, prior_incarnation#
  2  FROM v$database_incarnation

INCARNATION# RESETLOGS_CHANGE# PRIOR_INCARNATION#
-----
1          1                  0
2         446075                1
3         498982                2
4         500243                2
```

In Beispiel ?? zeigt die Spalte *RESETLOGS_CHANGE#* die SCN, des Startzeitpunktes der jeweiligen Inkarnation an. Wie zu sehen ist, beginnt Inkarnation Nummer 1 mit SCN 1 und jede weitere Inkarnation mit einer höheren SCN. Somit lässt sich folgendes Bild zeichnen:

Abb. 31.6:
Datenbankinkar-
nationen



Inkarnationen und die Logsequenz Nummer (LSN)

Eine neue Inkarnation wird mit dem Kommando `ALTER DATABASE OPEN RESETLOGS` erzeugt. Das Schlüsselwort `RESETLOGS` deutet darauf hin, dass die Logsequenz Nummer bei jeder neuen Inkarnation auf den Wert 1 zurückgesetzt wird. Tatsächlich geschehen folgende Schritte:

1. Die aktuellen Redo Logs werden archiviert,
2. die Log Sequenz Nummer wird auf den Wert 1 zurückgesetzt und
3. die Redo Logs erhalten einen neuen Zeitstempel, sowie eine neue SCN.

31.6.2 Voraussetzungen für Database Point-In-Time Recovery

Um ein Database-Point-In-Time-Recovery durchführen zu können, müssen die folgenden Voraussetzungen gegeben sein:

- Die betreffende Datenbank muss sich im Archivelog-Modus befinden.
- Es müssen Backups aller Datendateien bestehen, die vor dem gewünschten Recoveryzeitpunkt entstanden sind.
- Alle Archive Logs, die zum Roll-Forward des Backups bis zum gewünschten Recoveryzeitpunkt benötigt werden, müssen vorhanden sein.

31.6.3 Database Point-In-Time Recovery vorbereiten

Die folgenden Schritte sollten für ein Datenbank Point-In-Time Recovery vorbereitet werden.

- Festlegen des Zielzeitpunktes, der SCN, des Restore-Points oder der Log Sequence Number, bei der das Recovery stoppen soll. Dies kann mit Hilfe der Oracle Flashback Features geschehen. Auch die Alert.log-Datei kann hierbei dienlich sein.
- Soll als Abbruchkriterium für das Recovery ein Zeitpunkt verwendet werden, sollten die beiden Umgebungsvariablen NLS_LANG und NLS_DATE_FORMAT gesetzt sein.

31.6.4 Point-In-Time Recovery durchführen

1. Starten des RMAN und mit der Zieldatenbank verbinden.

Listing 31.98: Starten und Anmelden

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle
```

2. Überführen der Datenbank in die MOUNT-Phase.

Listing 31.99: Shutdown und Mounten

```
RMAN> shutdown immediate
RMAN> startup mount
```

3. Die folgenden Schritte sollten in einem RUN-Block ausgeführt werden:

- a) Zielzeitpunkt mit dem `SET UNTIL`-Kommando festlegen.
- b) Kanäle zum Zugriff auf die Datenbank konfigurieren, falls keine automatisch Vorkonfigurierten vorhanden sind.
- c) Restore and Recovery

Listing 31.100: Restore and Recovery

```
RMAN> RUN {
2>     SET UNTIL TIME '31.10.2013 10:30:00';
#Alternativen
# SET UNTIL SCN 487159;
# SET UNTIL SEQUENCE 62;
# SET UNTIL RESTORE POINT before_update;
3>
4>     RESTORE database;
```

```

5>      RECOVER database;
6>    }

```

4. Datenbank mit OPEN RESETLOGS öffnen

Listing 31.101: Datenbank mit open resetlogs öffnen

```
RMAN> SQL 'ALTER DATABASE OPEN RESETLOGS';
```

31.6.5 Recovery nach Verlust der Kontrolldatei (mit Recovery Katalog)

Wenn es vorkommt, dass alle Kopien der Kontrolldatei einer Datenbank zerstört werden oder verloren gehen, muss eine Kontrolldatei aus einem Backup wiederhergestellt werden. Diese wird dann als „Backup-controlfile“ bezeichnet. Da die Kontrolldatei aus einem Backup, nicht mehr den aktuellsten Stand der Datenbank wiederspiegelt, muss die gesamte Datenbank auf den Stand des Backupcontrolfile recovered werden.

Wiederherstellen eines Controlfiles aus einem Controlfile Autobackup

1. Mit dem RMAN an der Zieldatenbank und am Recovery Katalog anmelden.

Listing 31.102: An der Zieldatenbank und am Recovery Katalog anmelden

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle catalog catowner/catpass@CATDB
```

2. Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.103: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```



Dieser Schritt ist in SQL*Plus nicht möglich!

3. Wiederherstellen des Controlfiles aus dem Controlfile Autobackup

Listing 31.104: Wiederherstellen des Controlfiles

```
RMAN> RESTORE controlfile
2>   FROM AUTOBACKUP;
```

4. Die Zieldatenbank in den MOUNT-Status versetzen

Listing 31.105: Zieldatenbank mounten

```
RMAN> SQL 'ALTER DATABASE MOUNT';
```

5. Wiederherstellen aller Datendateien

Listing 31.106: Datendateien wiederherstellen

```
RMAN> RESTORE database;
```

6. Recovern der Datenbank

Listing 31.107: Recovern der Datenbank

```
RMAN> RECOVER database;
```

7. Öffnen der Datenbank mit der Option OPEN RESETLOGS

Listing 31.108: Datenbank mit open resetlogs öffnen

```
RMAN> SQL 'ALTER DATABASE OPEN RESETLOGS';
```

Wiederherstellen eines Controlfiles aus einem Backup Set

1. Mit dem RMAN an der Zieldatenbank und am Recovery Katalog anmelden.

Listing 31.109: An der Zieldatenbank und am Recovery Katalog anmelden

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle catalog catowner/catpass@CATDB
```

2. Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.110: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```

Dieser Schritt ist in SQL*Plus nicht möglich!



3. Wiederherstellen des Controlfiles.

Listing 31.111: Wiederherstellen des Controlfiles

```
RMAN> RESTORE controlfile;
```

4. Die Zieldatenbank in den MOUNT-Status versetzen

Listing 31.112: Zieldatenbank mounten

```
RMAN> SQL ,ALTER DATABASE MOUNT ;
```

5. Wiederherstellen aller Datendateien

Listing 31.113: Datendateien wiederherstellen

```
RMAN> RESTORE database ;
```

6. Recovern der Datenbank

Listing 31.114: Recovern der Datenbank

```
RMAN> RECOVER database ;
```

7. Öffnen der Datenbank mit der Option OPEN RESETLOGS

Listing 31.115: Datenbank mit open resetlogs öffnen

```
RMAN> SQL ,ALTER DATABASE OPEN RESETLOGS ;
```

31.6.6 Recovery nach Verlust der Kontrolldatei (ohne Recovery Katalog)

Wiederherstellen eines Controlfiles aus einem Controlfile Autobackup

1. Mit dem RMAN an der Zieldatenbank anmelden.

Listing 31.116: An der Zieldatenbank anmelden

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle
```

2. Setzen der DBID

Listing 31.117: Setzen der DBID

```
RMAN> SET DBID 1351916467;
```

3. Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.118: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```

 Dieser Schritt ist in SQL*Plus nicht möglich!

4. Wiederherstellen des Controlfiles aus dem Controlfile Autobackup

Listing 31.119: Wiederherstellen des Controlfiles

```
RMAN> RESTORE controlfile FROM AUTOBACKUP
2>   RECOVERY AREA '/u05/fast_recovery_area',
3>   DB_NAME      'orcl';
```

 Wenn die Instanz nicht gemountet oder geöffnet ist, ist es dem RMAN unmöglich, den Speicherort der Controlfile Autobackups zu ermitteln. Mit Hilfe der beiden Parameter `RECOVERY AREA` und `DB_NAME` wird der Speicherort, die Fast Recovery Area, dem RMAN mitgeteilt.

5. Die Zieldatenbank in den MOUNT-Status versetzen

Listing 31.120: Zieldatenbank mounten

```
RMAN> SQL ,ALTER DATABASE MOUNT ,
```

6. Wiederherstellen aller Datendateien

Listing 31.121: Datendateien wiederherstellen

```
RMAN> RESTORE database;
```

7. Recovern der Datenbank

Listing 31.122: Recovern der Datenbank

```
RMAN> RECOVER database;
```

8. Öffnen der Datenbank mit der Option OPEN RESETLOGS

Listing 31.123: Datenbank mit open resetlogs öffnen

```
RMAN> SQL 'ALTER DATABASE OPEN RESETLOGS';
```

Wiederherstellen eines Controlfiles aus einem Backup Set

1. Mit dem RMAN an der Zieldatenbank anmelden und die DBID setzen.

Listing 31.124: An der Zieldatenbank anmelden und die DBID setzen

```
[oracle@FEA11-119SRV ~]$ rman target sys/oracle
```

```
RMAN> SET DBID 1351916467;
```

2. Wurde die Zieldatenbank heruntergefahren, muss erst eine Instanz erzeugt werden. Der RMAN benutzt Standardeinstellungen, um eine Minimalinstanz zu erstellen. Diese kann für das weitere Recovery genutzt werden.

Listing 31.125: Zieldatenbank im RMAN in den NOMOUNT-Status bringen

```
RMAN> startup nomount
```



Dieser Schritt ist in SQL*Plus nicht möglich!

3. Wiederherstellen des Controlfiles.

Listing 31.126: Wiederherstellen des Controlfiles

```
RMAN> RESTORE controlfile
2>      FROM '/u02/3ukkpd6p.bkp';
```

4. Die Zieldatenbank in den MOUNT-Status versetzen

Listing 31.127: Zieldatenbank mounten

```
RMAN> SQL ,ALTER DATABASE MOUNT ,
```

5. Wiederherstellen aller Datendateien

Listing 31.128: Datendateien wiederherstellen

```
RMAN> RESTORE database ;
```

6. Recovern der Datenbank

Listing 31.129: Recovern der Datenbank

```
RMAN> RECOVER database ;
```

7. Öffnen der Datenbank mit der Option OPEN RESETLOGS

Listing 31.130: Datenbank mit open resetlogs öffnen

```
RMAN> SQL ,ALTER DATABASE OPEN RESETLOGS ;
```

31.6.7 Wiederherstellen einer Datenbank im NOARCHIVELOG Modus

Das Wiederherstellen einer Datenbank im NOARCHIVELOG Modus ist ähnlich dem Wiederherstellen einer Datenbank im ARCHIVELOG Modus. Die Hauptunterschiede dabei sind:

- Es können nur Coldbackups zur Wiederherstellung verwendet werden.
- Block Media Recovery ist nicht möglich, da keine Archive Logs existieren.

Mit Hilfe von inkrementellen Backups ist auch im NOARCHIVELOG Modus eine eingeschränkte Form des Recovery möglich.

Das folgende Szenario verdeutlicht das Recovery einer Datenbank im NOARCHIVELOG Modus.

- Die Datenbank läuft im NOARCHIVELOG Modus.
- Es wird ein Recovery Katalog benutzt.
- Die Datenbank wird konsistent heruntergefahren und es wird ein Level 0 Backup am Sonntagabend gemacht.

- Am darauf folgenden Mittwoch wird die Datenbank ebenfalls konsistent heruntergefahren und es wird ein Level 1 Backup angefertigt.
- Donnerstagabend crashed die Datenbank. Es gehen 50 % aller Datendateien und sämtliche Redo Logs verloren.

In solch einem Fall muss ein Media Recovery unter Nutzung des Level 0 und des Level 1 Backups durchgeführt werden. Weiterhin muss berücksichtigt werden, dass die Redo Logs komplett verloren gegangen sind.

1. Mit dem RMAN an der Zieldatenbank und am Recovery Katalog anmelden.

Listing 31.131: An der Zieldatenbank und am Recovery Katalog anmelden

```
[oracle@FEA11-119SRV ~]$ rman target / catalog catowner/catpass@CATDB
```

2. Herunterfahren der Datenbank und starten im NOMOUNT-Status.

Listing 31.132: Shutdown und Mounten

```
RMAN> startup force nomount;
```

Ein `startup force nomount` kann ohne Weiteres durchgeführt werden, da die Datenbank bereits irreparabel beschädigt ist. Ein konsistentes Herunterfahren ist nicht mehr nötig/möglich.

3. Wiederherstellen der Kontrolldatei auf einem Autobackup oder einem Backup Set.

Listing 31.133: Kontrolldatei wiederherstellen

```
RMAN> RESTORE controlfile;
```

4. Die Datenbank in den MOUNT-Status überführen

Listing 31.134: MOUNT-Status erreichen

```
RMAN> SQL 'ALTER DATABASE MOUNT';
```

5. Wiederherstellen der Datenbankdateien

Listing 31.135: Datenbankdateien wiederherstellen

```
RMAN> RESTORE database;
```

6. Recovern der Datenbank

Listing 31.136: Datenbankdateien wiederherstellen

```
RMAN> RECOVER database NOREDO;
```

7. Öffnen der Datenbank

Listing 31.137: Datenbank mit open resetlogs öffnen

```
RMAN> SQL 'ALTER DATABASE OPEN RESETLOGS';
```

Bei diesem Vorgang werden alle Änderungen, die bis einschließlich des Level 1 Backups vorgenommen wurden recovered. Die Angabe von `NOREDO` sorgt dafür, dass RMAN nicht versucht, die Redo Logs beim Recovery zu nutzen, da diese nicht mehr existieren.

Selbst wenn die Redo Logs noch existieren würden, müsste trotzdem die `NOREDO`-Klausel angegeben werden, da zwischen den letzten Änderung im Level 1 Backup und den Redo Logs eine große Lücke klafft.

31.7 Der Data Recovery Advisor

Der Data Recovery Advisor ist eine mit Oracle 11g neu eingeführte PL/SQL-Anwendung, die den Administrator bei einem Recovery-Szenario unterstützt. Er kann automatisch nach Fehlern und Lösungen suchen und, per Anweisung des Admins, eine Reparatur ausführen.

Ohne dieses Tool muss der Admin selbstständig eine Fehlerdiagnose durchführen und anschließend geeignete Maßnahmen ergreifen, um die Schäden zu beheben. Eine solches Prozedere ist sehr umständlich und fehleranfällig, weshalb es gute Kenntnisse der Materie und ein hohes Maß an Erfahrung erfordert.

31.7.1 Fehler

Fehler, im Sinne des Data Recovery Advisors sind persistente Störungen der Datenbank die durch einen „Data Integrity Check“ gefunden wurden. Sobald ein Fehler festgestellt wurde, kann der Data Recovery Advisor Informationen darüber liefern und ihn gegebenenfalls beheben. Gespeichert werden Fehler nicht innerhalb der Datenbank, sondern im Automatic Diagnostic Repository, kurz ADR, einer Verzeichnisstruktur im Dateisystem. Daher wird die Funktionsweise des Data Recovery Advisor nicht beeinträchtigt, falls sich die Datenbank in der NOMOUNT-Phase befindet.

Fehlerarten

Der Data Recovery Advisor ist in der Lage, verschiedenste Fehlerarten zu erkennen. Beispielsweise kann er feststellen, dass Datenbankdateien (Datendateien, Kontroll- und Redo Log Dateien) nicht geöffnet werden können, weil sie nicht existieren oder Zugriffsrechte fehlen. Des Weiteren kann er Beschädigungen von Dateien erkennen, wenn z. B. eine Datendatei nicht mehr synchron mit der Datenbank ist (Inkonsistenz) oder wenn einzelne Oracleblöcke physikalisch beschädigt sind. Teilweise kann er sogar Fehler erkennen, die nicht in der Datenbank, sondern im Betriebssystem auftreten (Disk I/O, Treiberfehler, usw.).

Status und Priorität

Jeder Fehler hat einen Status und eine Priorität. Wenn ein Fehler entsteht, erhält er automatisch den Status „Open“. Sobald der Fehler behoben wurde, wird der Status auf „Closed“ geändert. Diese Statusänderung geschieht entweder durch das RMAN-Kommando `LIST FAILURE` oder durch `CHANGE FAILURE`.

Wie schwerwiegend ein Fehler ist, wird in drei Prioritätsklassen ausgedrückt: „LOW“, „HIGH“ und „CRITICAL“.

- **CRITICAL:** Dies sind schwerwiegende Fehler, die die Verfügbarkeit der gesamten Datenbank beeinflussen können und die deshalb sofortiger Aufmerksamkeit bedürfen (Ausfall von Kontrolldateien oder des SYSTEM-Tablespaces).
- **HIGH:** Dies Fehlerklasse umfasst alle Ereignisse, welche die Verfügbarkeit von Teilen der Datenbank beeinflussen können (Korrupte Blöcke, Ausfall von Nicht-System-Tablespaces, fehlen von Archive Logs).
- **LOW:** Die Fehlerklasse „LOW“ wird nicht vom System, sondern nur vom Administrator vergeben, wenn er Fehler der Klasse „HIGH“ als „nicht besonders wichtig“ einstuft.

31.7.2 Die Fehlersuche - LIST FAILURE

Das Kommando `LIST FAILURE` dient dazu, die Ergebnisse von automatischen oder manuellen Fehlerdiagnosen anzuzeigen. `LIST FAILURE` selbst diagnostiziert keine Fehler.

Listing 31.138: Das Kommando `LIST FAILURE`

```
RMAN> LIST FAILURE;  
  
List of Database Failures
```

```
=====
Failure ID Priority Status      Time Detected Summary
-----
42        HIGH    OPEN       01-NOV-13      One or more non-system
datafiles are missing
1141      HIGH    OPEN       01-NOV-13      Datafile 7:
'/u01/app/oracle/oradata/orcl/bank02.dbf' contains one or more corrupt blocks
```

Beispiel ?? zeigt zwei Fehler der Kategorie „HIGH“, versehen mit den IDs 42 und 1141. In der Spalte „Summary“ wird eine kurze Erklärung zu diesen Fehlern angezeigt.

31.7.3 Der Ratschlag - ADVISE FAILURE

Mit dem [ADVISE FAILURE](#)-Befehl können durch den RMAN generierte Reparaturskripte angezeigt und bereits behobene Fehler geschlossen werden (Status „CLOSED“).



Es muss immer zuerst das Kommando [LIST FAILURE](#) ausgeführt werden, bevor der Befehl [ADVISE FAILURE](#) ausgeführt werden kann.

Listing 31.139: Das Kommando [ADVISE FAILURE](#)

```
RMAN> ADVISE FAILURE;

List of Database Failures
=====

Failure ID Priority Status      Time Detected Summary
-----
42        HIGH    OPEN       01-NOV-13      One or more non-system
datafiles are missing
1141      HIGH    OPEN       01-NOV-13      Datafile 7:
'/u01/app/oracle/oradata/orcl/bank02.dbf' contains one or more corrupt blocks

analyzing automatic repair options; this may take some time
using channel ORA_DISK_1
using channel ORA_DISK_2
allocated channel: ORA_SBT_TAPE_1
channel ORA_SBT_TAPE_1: SID=134 device type=SBT_TAPE
channel ORA_SBT_TAPE_1: WARNING: Oracle Test Disk API
analyzing automatic repair options complete

Mandatory Manual Actions
```

```
=====
no manual actions available

Optional Manual Actions
=====
1. If file /u01/app/oracle/oradata/orcl/bank01.dbf was unintentionally
renamed or moved, restore it

Automated Repair Options
=====
Option Repair Description
-----
1      Restore and recover datafile 6; Recover multiple corrupt blocks
          in datafile 7
Strategy: The repair includes complete media recovery with no data loss
Repair script: /u01/app/oracle/diag/rdbms/orcl/orcl/hm/reco_184824511.hm
```

Zuerst wird eine Liste der erkannten Fehler angezeigt, bevor dann eine Analyse durchgeführt wird, um automatische Reparaturmaßnahmen zu finden. Diese werden dann in einem Reparaturskript zusammengefasst, dessen Name ganz unten im Bericht angezeigt wird.

Listing 31.140: Das Reparaturskript

```
Strategy: The repair includes complete media recovery with no data loss
Repair script: /u01/app/oracle/diag/rdbms/orcl/orcl/hm/reco_184824511.hm
```



RMAN versucht immer seine Reparaturskripte zu konsolidieren, was bedeutet, dass möglichst viele Fehler mit möglichst wenigen Reparaturschritten behoben werden sollen. In manchen Fällen ist dies jedoch nicht möglich, weshalb RMAN dann eine Meldung anzeigt, dass aktuell einige Fehler nicht behoben werden können.

ADVISE FAILURE zeigt aber nicht nur automatische Reparaturmaßnahmen an, sondern wo immer es sich anbietet auch Manuelle. Diese werden in „optionale“ und „zwingend notwendige“ Maßnahmen unterteilt.

Beispiel ?? zeigt für Datendatei Nummer 6 einen optionalen Schritt an. Sollte die Datei nur aus Versehen umbenannt oder verschoben worden sein, kann dieser Schritt manuell rückgängig gemacht werden. Eine solche Vorgehensweise ist unter Umständen viel zeit- und ressourcensparender als ein Restore & Recovery der Datendatei.

Mandatory manual options werden immer dann angezeigt, wenn keine automatischen Reparaturschritte erzeugt werden können. Dies könnte z. B. dann der Fall sein, wenn ein Archive Log fehlt, dass für das Recovery einer Datendatei benötigt wird.

Listing 31.141: Mandatory manual options

```
RMAN> ADVISE FAILURE;

List of Database Failures
=====

Failure ID Priority Status      Time Detected Summary
----- ----- -----
42        HIGH    OPEN       01-NOV-13     One or more non-system
datafiles are missing
1141      HIGH    OPEN       01-NOV-13     Datafile 7:
'/u01/app/oracle/oradata/orcl/bank02.dbf' contains one or more corrupt blocks
analyzing automatic repair options; this may take some time
using channel ORA_DISK_1
using channel ORA_DISK_2
using channel ORA_SBT_TAPE_1
```

```
analyzing automatic repair options complete

Mandatory Manual Actions
=====
1. If file /u01/app/oracle/oradata/orcl/bank01.dbf was unintentionally
   renamed or moved, restore it
2. If you have an export of tablespace BANK, then drop and re-create
   the tablespace and import the data.
3. No backup of block 232 in file 7 was found. Drop and re-create the
   associated object (if possible), or use the DBMS_REPAIR package to
   repair the block corruption
4. No backup of block 233 in file 7 was found. Drop and re-create the
   associated object (if possible), or use the DBMS_REPAIR package to
   repair the block corruption
5. No backup of block 234 in file 7 was found. Drop and re-create the
   associated object (if possible), or use the DBMS_REPAIR package to
   repair the block corruption
6. No backup of block 235 in file 7 was found. Drop and re-create the
   associated object (if possible), or use the DBMS_REPAIR package to
   repair the block corruption
7. Contact Oracle Support Services if the preceding recommendations
   cannot be used, or if they do not fix the failures selected for repair

Optional Manual Actions
=====
no manual actions available

Automated Repair Options
=====
no automatic repair options available
```

31.7.4 Die Reparatur - REPAIR FAILURE

Der „Dritte im Bunde“ ist [REPAIR FAILURE](#). Dieses Kommando benutzt das von [ADVISE FAILURE](#) erstellte Reparaturskript, um die festgestellten Fehler zu beheben. Vor der eigentlichen Reparatur kann mit Hilfe von [REPAIR FAILURE PREVIEW](#) zuerst das komplette Repairscrip angzeigt werden.

Listing 31.142: Eine Vorschau auf das Repairscrip

```
RMAN> REPAIR FAILURE PREVIEW;

Strategy: The repair includes complete media recovery with no data loss
Repair script: /u01/app/oracle/diag/rdbms/orcl/orcl/hm/reco_1758777565.hm

contents of repair script:
# restore and recover datafile
sql 'alter database datafile 6 offline';
restore datafile 6;
recover datafile 6;
sql 'alter database datafile 6 online';
# block media recovery for multiple blocks
recover datafile 7 block 232 to 235;
```

Bevor der Befehl **REPAIR FAILURE** ausgeführt wird, sollte immer zuerst eine Endkontrolle des Reparaturscripts erfolgen.

Listing 31.143: Die Fehler reparieren

```
RMAN> REPAIR FAILURE;

Strategy: The repair includes complete media recovery with no data loss
Repair script: /u01/app/oracle/diag/rdbms/orcl/orcl/hm/reco_1758777565.hm

contents of repair script:
# restore and recover datafile
sql 'alter database datafile 6 offline';
restore datafile 6;
recover datafile 6;
sql 'alter database datafile 6 online';
# block media recovery for multiple blocks
recover datafile 7 block 232 to 235;

Do you really want to execute the above repair (enter YES or NO)? YES
executing repair script
```

31.8 Informationen

31.8.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10029]
- [REFRN10030]
- [REFRN10268]
- [REFRN10295]
- [REFRN10061]
- [REFRN10089]
- [REFRN10086]

31.8.2 Verzeichnis der relevanten Data Dictionary Views



- [REFRN30047]
- [REFRN30048]
- [REFRN30049]
- [sthref3281]
- [REFRN30052]
- [REFRN30196]
- [sthref3785]

31.9 Übungen - Recovery mit dem RMAN

1. Führen Sie ein Recovery bei Verlust einer Redo Log Datei durch!

- a) Starten Sie das Skript `lab_delete_redolog_member.sql`! Es löscht einen beliebigen Member einer Ihrer Redo Log Gruppen.

```
SQL> @/home/oracle/labs/lab_delete_redolog_member.sql
```

- b) Die Datenbank läuft weiterhin normal und es liegen keinerlei Probleme vor. Öffnen Sie die Alert Log Datei, um herauszufinden welcher Redo Log Member gelöscht wurde!

- c) Führen Sie geeignete Maßnahmen zur Problembehebung durch!

2. Führen Sie das Skript `lab_delete_tempfile.sql` im laufenden Betrieb der Datenbank aus! Dieses Skript wird ein Tempfile eines Ihrer temporären Tablespaces löschen.

```
SQL> @/home/oracle/labs/lab_delete_tempfile.sql
```

3. Prüfen Sie in der Alert Log Datei, welches Tempfile gelöscht wurde!

4. Überlegen Sie sich eine geeignete Maßnahme, um mit möglichst geringem Aufwand Ihren Temp-Tablespace wieder einsatzfähig zu machen!

5. Führen Sie ein Recovery bei Verlust einer Kontrolldatei durch.

- a) Starten Sie das Skript `lab_delete_ctlfl.sql`. Es löscht eine Ihrer Kontrolldateien.

```
SQL> @/home/oracle/labs/lab_delete_ctlfl.sql
```

- b) Analysieren Sie das Problem und führen Sie ein geeignetes Recovery durch.

6. Führen Sie ein Recovery bei Verlust einer Anwendungsdatendatei durch.

a) Starten Sie das Skript `lab_delete_application_datafile.sql`. Es löscht eine Datendatei Ihrer Datenbank.

```
SQL> @/home/oracle/labs/lab_delete_application_datafile.sql
```

b) Prüfen Sie welche Datendatei defekt ist.

c) Ergreifen Sie geeignete Maßnahmen, um die defekte Datendatei wieder herzustellen.

7. Führen Sie ein Recovery bei Verlust einer Systemdatendatei durch.

- a) Starten Sie das Skript `lab_delete_system_datafile.sql`. Es löscht die Datendatei Ihres SYSTEM-Tablespaces oder Ihres UNDO-Tablespaces.

```
SQL> @/home/oracle/labs/lab_delete_system_datafile.sql
```

b) Prüfen Sie welche Datendatei defekt ist.

c) Ergreifen Sie geeignete Maßnahmen, um die defekte Datendatei wieder herzustellen.

8. Führen Sie ein Recovery mit einem Backup Controlfile durch.

- a) Starten Sie das Skript lab_delete_all_controlfiles.sql. Es wird alle Kontrolldateien löschen.

- b) Ergreifen Sie geeignete Maßnahmen, um die Datenbank wieder lauffähig zu machen.

9. Führen Sie das Skript `lab_destroy_table.sql` aus. Dieses Skript wird einzelne Blöcke im Tablespace BANK zerstören.

```
SQL> @/home/oracle/labs/lab_destroy_table.sql
```

10. Finden Sie heraus, welche Blöcke im Tablespace BANK beschädigt wurden!

11. Fertigen Sie ein Backup des defekten BANK-Tablespaces an.

12. Führen Sie mit Hilfe von RMAN ein Block-Media-Recovery durch, um die beschädigten Blöcke zu reparieren.

31.10 Lösungen - Recovery mit dem RMAN

1. Führen Sie ein Recovery bei Verlust einer Redo Log Datei durch!

- a) Starten Sie das Skript `lab_delete_redolog_member.sql`! Es löscht einen beliebigen Member einer Ihrer Redo Log Gruppen.

```
SQL> @/home/oracle/labs/lab_delete_redolog_member.sql
```

- b) Die Datenbank läuft weiterhin normal und es liegen keinerlei Probleme vor. Öffnen Sie die Alert Log Datei, um herauszufinden welcher Redo Log Member gelöscht wurde!
- c) Führen Sie geeignete Maßnahmen zur Problembehebung durch!

Listing 31.144: Ermitteln des betroffenen Redo Log Members im Alert Log

```
Errors in file /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_arc0_3513.trc
ORA-00313: open failed for members of log group 5 of thread 1
ORA-00312: online log 5 thread 1: '/u01/app/oracle/oradata/orcl redo05a.log',
ORA-27037: unable to obtain file status
Linux-x86_64 Error: 2: No such file or directory
```

Listing 31.145: Recovern des Members

```
[oracle@FEA11-119SRV ~]$ cd /u01/app/oracle/oradata/orcl
[oracle@FEA11-119SRV ~]$ cp /u02/oradata/orcl/redo05b.log redo05a.log
```

2. Führen Sie das Skript `lab_delete_tempfile.sql` im laufenden Betrieb der Datenbank aus! Dieses Skript wird ein Tempfile eines Ihrer temporären Tablespaces löschen.

```
SQL> @/home/oracle/labs/lab_delete_tempfile.sql
```

3. Prüfen Sie in der Alert Log Datei, welches Tempfile gelöscht wurde!

4. Überlegen Sie sich eine geeignete Maßnahme, um mit möglichst geringem Aufwand Ihren Temp-Tablespace wieder einsatzfähig zu machen!

Listing 31.146: Ermitteln des betroffenen Tempfiles im Alert Log

```
Errors in file /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_m000_7480.trc
ORA-01116: error in opening database file 201
ORA-01110: data file 201: '/u01/app/oracle/oradata/orcl/temp01.dbf',
ORA-27041: unable to open file
Linux-x86_64 Error: 2: No such file or directory
Additional information: 3
```

Listing 31.147: Neues Tempfile erstellen

```
SQL> ALTER TABLESPACE temp
  2  ADD TEMPFILE '/u02/oradata/0RCL/temp02.dbf'
  3  SIZE 20 M AUTOEXTEND ON MAXSIZE 500M;
```

Listing 31.148: Beschädigtes Tempfile löschen

```
SQL> ALTER TABLESPACE temp
  2  DROP TEMPFILE '/u02/oradata/0RCL/temp01.dbf';
```

5. Führen Sie ein Recovery bei Verlust einer Kontrolldatei durch.

- a) Starten Sie das Skript lab_delete_ctlfl.sql. Es löscht eine Ihrer Kontrolldateien.

```
SQL> @/home/oracle/labs/lab_delete_ctlfl.sql
```

- b) Analysieren Sie das Problem und führen Sie ein geeignetes Recovery durch.

Listing 31.149: Ermitteln der betroffenen Kontrolldatei im Alert Log

```
Errors in file /u01/app/oracle/diag/rdbms/orcl/orcl/trace/orcl_ora_2765.trc:
ORA-00210: cannot open the specified control file
ORA-00202: control file: '/u01/app/oracle/oradata/orcl/control01.ctl'
ORA-27041: unable to open file
Linux-x86_64 Error: 2: No such file or directory
```

Listing 31.150: Recovern der Kontrolldatei

```
[oracle@FEA11-119SRV ~]$ cd /u01/app/oracle/oradata/orcl
[oracle@FEA11-119SRV ~]$ cp control02.ctl control01.ctl
```

6. Führen Sie ein Recovery bei Verlust einer Anwendungsdatendatei durch.

- a) Starten Sie das Skript lab_delete_application_datafile.sql. Es löscht eine Datendatei Ihrer Datenbank.

```
SQL> @/home/oracle/labs/lab_delete_application_datafile.sql
```

- b) Prüfen Sie welche Datendatei defekt ist.

- c) Ergreifen Sie geeignete Maßnahmen, um die defekte Datendatei wieder herzustellen.

Listing 31.151: Recovern der Datendatei

```
RMAN> VALIDATE database;
Starting validate at 03-NOV-13
using target database control file instead of recovery catalog
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=21 device type=DISK
allocated channel: ORA_DISK_2
channel ORA_DISK_2: SID=144 device type=DISK
RMAN-06169: could not read file header for datafile 6 error reason 5
RMAN-00571: ======
RMAN-00569: ====== ERROR MESSAGE STACK FOLLOWS ======
```

```
RMAN-00571: =====
RMAN-03002: failure of validate command at 11/03/2013 09:24:26
RMAN-06056: could not access datafile 6
```

```
RMAN> SQL 'ALTER DATABASE DATAFILE 6 OFFLINE';

RMAN> RESTORE datafile 6;

RMAN> RECOVER datafile 6;

RMAN> SQL 'ALTER DATABASE DATAFILE 6 ONLINE';
```

7. Führen Sie ein Recovery bei Verlust einer Systemdatendatei durch.

- a) Starten Sie das Skript lab_delete_system_datafile.sql. Es löscht die Datendatei Ihres SYSTEM-Tablespaces oder Ihres UNDO-Tablespaces.

```
SQL> @/home/oracle/labs/lab_delete_system_datafile.sql
```

- b) Prüfen Sie welche Datendatei defekt ist.
- c) Ergreifen Sie geeignete Maßnahmen, um die defekte Datendatei wieder herzustellen.

Listing 31.152: Recovern der Systemdatendatei

```
RMAN> VALIDATE database;
Starting validate at 03-NOV-13
released channel: ORA_SBT_TAPE_1
using channel ORA_DISK_1
using channel ORA_DISK_2
channel ORA_DISK_1: starting validation of datafile
channel ORA_DISK_1: specifying datafile(s) for validation
RMAN-00571: =====
RMAN-00569: ====== ERROR MESSAGE STACK FOLLOWS =====
RMAN-00571: =====
RMAN-03009: failure of validate command on ORA_DISK_1 channel at 11/03/2013
ORA-01122: database file 1 failed verification check
ORA-01110: data file 1: '/u01/app/oracle/oradata/orcl/system01.dbf'
ORA-01565: error in identifying file
          '/u01/app/oracle/oradata/orcl/system01.dbf'
ORA-27037: unable to obtain file status
Linux-x86_64 Error: 2: No such file or directory
Additional information: 3

RMAN> shutdown abort

RMAN> startup mount

RMAN> RESTORE datafile 1;

RMAN> RECOVER datafile 1;
```

RMAN> ALTER DATABASE OPEN;

8. Führen Sie ein Recovery mit einem Backup Controlfile durch.

- a) Starten Sie das Skript lab_delete_all_controlfiles.sql. Es wird alle Kontrolldateien löschen.

```
SQL> @/home/oracle/labs/lab_delete_all_controlfiles.sql
```

- b) Ergreifen Sie geeignete Maßnahmen, um die Datenbank wieder lauffähig zu machen.

Listing 31.153: Herunterfahren der Datenbank in SQL*Plus

```
SQL> shutdown abort
```

Listing 31.154: Recovern der Kontrolldateien

```
RMAN> SET DBID 1351916467;
RMAN> startup nomount
RMAN> RESTORE controlfile
2>   FROM AUTOBACKUP
3>   RECOVERY AREA '/u05/fast_recovery_area'
4>   DB_NAME = 'orcl';

Starting restore at 03-NOV-13
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=134 device type=DISK

recovery area destination: /u05/fast_recovery_area
database name (or database unique name) used for search: ORCL
channel ORA_DISK_1: AUTOBACKUP /u05/fast_recovery_area/ORCL/autobackup/
channel ORA_DISK_1: looking for AUTOBACKUP on day: 20131103
channel ORA_DISK_1: restoring control file from AUTOBACKUP
channel ORA_DISK_1: control file restore from AUTOBACKUP complete
output file name=/u01/app/oracle/oradata/orcl/control01.ctl
output file name=/u05/fast_recovery_area/orcl/control02.ctl
Finished restore at 03-NOV-13

RMAN> SQL 'ALTER DATABASE MOUNT';
RMAN> RECOVER database;
RMAN> SQL 'ALTER DATABASE OPEN RESETLOGS';
```

9. Führen Sie das Skript lab_destroy_table.sql aus. Dieses Skript wird einzelne Blöcke im Tablespace BANK zerstören.

```
SQL> @/home/oracle/labs/lab_destroy_table.sql
```

10. Finden Sie heraus, welche Blöcke im Tablespace BANK beschädigt wurden!

Listing 31.155: Validieren der Datenbank

```
RMAN> VALIDATE database;
```

11. Fertigen Sie ein Backup des defekten BANK-Tablespaces an.

Listing 31.156: Validieren der Datenbank

```
RMAN> RUN {
2>   SET MAXCORRUPT FOR datafile 7 TO 999;
3>   BACKUP AS BACKUPSET DATAFILE 7;
4> }
```

12. Führen Sie mit Hilfe von RMAN ein Block-Media-Recovery durch, um die beschädigten Blöcke zu reparieren.

Listing 31.157: Recovern der Datendatei

```
RMAN> BLOCKRECOVER CORRUPTION LIST;
```

32 Oracle Flashback

Inhaltsangabe

32.1 Die Oracle Flashback Technologie

Die Oracle Flashback Technologie stellt Möglichkeiten bereit, „in die Vergangenheit“ von Daten zu sehen, ohne ein Recovery durchführen zu müssen. Die meisten der Oracle Flashback Features arbeiten auf der Ebene logischer Backups:

- **Oracle Flashback Query:** Diese Technologie ermöglicht es, Daten einer Tabelle zu einem definierten Zeitpunkt zu sehen. Damit ist es möglich, fehlerhafte `UPDATE`- oder `DELETE`-Operationen rückgängig zu machen.
- **Oracle Flashback Version Query:** Mit diesem Feature können alle Versionen einer Tabellenzeile innerhalb eines spezifizierten Zeitintervalls betrachtet werden. Es werden nicht nur die Nutzdaten, sondern auch Metadaten (Startzeit, Endzeit und TransaktionsID der Transaktion) angezeigt. Damit ist die Möglichkeit gegeben, sowohl Datenverlust zu beheben, als auch Auditing zu betreiben.
- **Oracle Flashback Transaction Query:** Hiermit kann man sich alle Änderungen betrachten, die eine einzelne Transaktion oder alle Transaktionen innerhalb eines bestimmten Zeitraums durchgeführt haben.
- **Oracle Flashback Transaction Backout:** Mit dieser neuen Technologie ist es möglich, bereits committete Transaktionen rückgängig zu machen.
- **Oracle Flashback Table:** Dieser Mechanismus ermöglicht es, eine Tabelle online in einen Zustand zurück zuversetzen, der durch einen Zeitpunkt definiert wird.
- **Oracle Flashback Drop:** Der Flashback Drop macht es möglich, die Auswirkungen eines `DROP TABLE`-Statements rückgängig zu machen.
- **Oracle Flashback Database:** Diese Option ist die „Rewind-Taste“ an der Datenbank. Die gesamte DB kann innerhalb kürzester Zeit auf einen genau definierten Stand zurückversetzt werden.

Die Features Flashback Table, Flashback Query, Flashback Transaction Query und Flashback Version Query basieren alle auf Undo-Daten. Der Mechanismus Flashback Drop verwendet einen Speicherbereich der „Recycle Bin“ genannt wird. In diesem Speicherbereich werden gelöschte Tabellen gespeichert, bis der Speicher überläuft und Platz für neue Objekte geschaffen werden muss. Alle diese Features sind unabhängig von RMAN.

Auf einer anderen Ebene existiert das Feature „Oracle Flashback Database“. Es ist eine Alternative zum Point-In-Time-Recovery. Wenn in der Datenbank eine größere Anzahl fehlerhafter Änderungen

gespeichert ist, kann Flashback Database diese Datendateien in einen früheren Zustand zurückversetzen.

Die Auswirkungen des Oracle Flashback Database Mechanismus sind denen eines Point-In-Time-Recovery sehr ähnlich, jedoch funktioniert der Flashback Mechanismus deutlich schneller als ein Recovery, da das Restore der Datendateien wegfällt und nur wenige Redo Logs für das Flashback benötigt werden.

Flashback Database benutzt „Flashback logs“ und zusätzlich die archivierten Redo Logs, um frühere Versionen eines Datenblocks wieder herzustellen. Zur Speicherung von Flashback Logs muss eine „Fast Recovery Area“ erstellt werden. Flashback logging ist standardmäßig deaktiviert.

Flashback Database ist in den RMAN integriert. Er kann sich automatisch aus den vorhandenen Backups die benötigten archivierten Redo Logs und Flashback Logs holen, um das Flashback durchzuführen. Auch eine manuelle Nutzung von Flashback Database mit SQL*Plus ist möglich.

32.2 Oracle Flashback Query

Flashback Query macht es möglich, einen in der Vergangenheit liegenden Stand einer Tabelle abzufragen. Wenn beispielsweise um 12:30 festgestellt wird, dass aus der Tabelle MITARBEITER der Angestellte „Wolf“ versehentlich gelöscht wurde, er aber um 09:30 Uhr noch existierte, kann der Stand der Tabelle MITARBEITER von 09:30 abgefragt werden.

Das folgende Beispiel zeigt eine Flashback Query auf der Tabelle MITARBEITER mit Hilfe der **AS OF**-Klausel:

Listing 32.1: Flashback Query mit AS OF TIMESTAMP

```
SQL> SELECT Mitarbeiter_ID, Vorname, Nachname
  2  FROM   mitarbeiter
  3 WHERE  Nachname LIKE 'Wolf'
  4 AND   Mitarbeiter_ID = 98;

no rows selected.

SQL> SELECT Mitarbeiter_ID, Vorname, Nachname
  2  FROM   mitarbeiter AS OF TIMESTAMP
  3      TO_TIMESTAMP('03.11.2013 09:30:00', 'DD.MM.YYYY HH24:MI:SS')
  4 WHERE  Nachname LIKE 'Wolf'
  5 AND   Mitarbeiter_ID = 98;

MITARBEITER_ID VORNAME          NACHNAME
-----
```

In diesem Beispiel wird die **AS OF**-Klausel zusammen mit einem Zeitpunkt benutzt. Es sind aber noch andere Angaben, wie z. B. Restore Points und SCNs möglich.



Die **AS OF**-Klausel kann an jedes beliebige **SELECT**-Statement angehängt werden.



- [ADFNS01003]

32.3 Oracle Flashback Version Query

Mit der Oracle Flashback Version Query können verschiedene Versionen einer oder mehrerer Tabellenzeilen angezeigt werden. Diese Technologie eignet sich hervorragend, um Änderungen an einer Tabellenzeile mitzuverfolgen. In Beispiel ?? werden die Kontobewegungen am Konto 447 zurückverfolgt.

Listing 32.2: Verschiedene Versionen einer Zeile mit Flashback Version Query

```
SQL> SELECT Konto_ID, Guthaben
  2  FROM bank.girokonto VERSIONS BETWEEN TIMESTAMP
  3      TO_TIMESTAMP('03.11.2013 10:55:00', 'DD.MM.YYYY HH:MI:SS')
  4  AND  TO_TIMESTAMP('03.11.2013 11:10:40', 'DD.MM.YYYY HH:MI:SS')
  5 WHERE Konto_ID = 447;

KONTO_ID    GUTHABEN
-----  -----
  447        1231,4
  447        1182,5
```

Die Klausel **VERSIONS BETWEEN** macht aus einer normalen Abfrage eine Flashback Version Query. Das Schlüsselwort **TIMESTAMP** kann, genau wie bei der **AS OF**-Klausel, durch das Schlüsselwort **SCN** ersetzt werden.

Bei einer Flashback Version Query können mit Hilfe von Pseudospalten noch weitere Informationen, über eine Zeile abgefragt werden. Tabelle ?? zeigt die Pseudospalten, die bei einer Flashback Version Query zur Verfügung stehen.

Tabelle 32.1: Pseudospalten bei der Flashback Version Query

Pseudospalte	Beschreibung
VERSIONS_STARTSCN VERSIONS_STARTTIME	Diese beiden Spalten zeigen die SCN oder den Zeitpunkt, an dem der erste Datenblock der betreffenden Zeile durch eine DML-Operation verändert wurde. Diese Information kann für das Wiederherstellen einer Zeile mit Hilfe einer Flashback Query benutzt werden.
VERSIONS_ENDSCN VERSIONS_ENDTIME	Zeigt die SCN oder den Zeitpunkt, an dem die noch aktuelle Zeile durch die neue Version vollständig ersetzt worden ist.
VERSIONS_XID	Identifiziert die Transaktion, die die Zeile verändert hat
VERSIONS_OPERATION	Zeigt die Art der DML-Operation an, die auf der Zeile ausgeführt wurde (I für INSERT, U für UPDATE und D für DELETE)

Listing 32.3: Informationsgewinnung mit Flashback Version Query

```
SQL> col VERSIONS_STARTTIME format a18
SQL> col VERSIONS_ENDTIME format a18

SQL> SELECT Konto_ID, Guthaben, VERSIONS_STARTTIME, VERSIONS_ENDTIME,
2      VERSIONS_XID, VERSIONS_OPERATION
2  FROM bank.girokonto VERSIONS BETWEEN TIMESTAMP
3      TO_TIMESTAMP('03.11.2013 10:55:00', 'DD.MM.YYYY HH:MI:SS')
4  AND TO_TIMESTAMP('03.11.2013 11:10:40', 'DD.MM.YYYY HH:MI:SS')
5 WHERE Konto_ID = 447;

KONTO_ID    GUTHABEN VERSIONS_STARTTIME VERSIONS_ENDTIME    VERSIONS_XID      V
-----  -----
        447      1231,4 03.11.13 11:09:52                      0900190059040000  U
        447      1182,5          03.11.13 11:09:52
```



- [ADFNS01004]

32.4 Oracle Flashback Transaction

Während mit Flashback Query die Vergangenheit einer Tabellenzeile und mit Flashback Version Query die Bearbeitungshistorie einer Zeile angezeigt werden kann, ermöglicht das Flashback Transaction Feature:

- zu sehen, was eine spezifische Transaktion an einer Tabellenzeile gemacht hat.
- Bestehende Transaktionen wieder rückgängig zu machen.

32.4.1 Flashback Transaction Query

Voraussetzungen

Flashback Transaction Query basiert, im Gegensatz zu Flashback Query und Flashback Version Query nicht nur auf den Undo Daten, sondern auch auf Redo Records (Archive Logs). Damit diese Technologie einwandfrei funktionieren kann, muss das Supplemental Logging aktiviert werden. Ohne Supplemental Logging kann Flashback Transaction Query nicht funktionieren.

Supplemental Logging

Supplemental Logging sorgt dafür, dass bei DML-Operationen zusätzliche Informationen in den Redo Logs festgehalten werden. Normalerweise wird bei einem `UPDATE` auf eine Zeile nur deren RowID als identifizierendes Merkmal festgehalten. Bei aktiviertem Supplemental Logging werden alle Spalten aufgezeichnet, die zur Identifikation einer Tabellenzeile benutzt wurden. Beispiel ?? zeigt ein `UPDATE`-Statement auf die Tabelle GIROKONTO.

Listing 32.4: Supplemental Logging - Demo Schritt 1

```
SQL> UPDATE girokonto
  2   SET      Guthaben = 1256.37
  3 WHERE Konto_ID = 447;

SQL> COMMIT;
```

Ohne das Supplemental Logging wird zu dieser Transaktion nur das aufgezeichnet, was in Beispiel ?? gezeigt wird.

Listing 32.5: Supplemental Logging - Demo Schritt 2

```
update "BANK"."GIROKONTO"
  set
    "GUTHABEN" = 1256,37
  where
    "GUTHABEN" = 1231,40 and
    ROWID = 'AAASWaAAHAAAADUACG',
```

Mit eingeschaltetem Supplemental Logging wird zusätzlich zu diesen Informationen noch die Primärschlüsselspalte festgehalten, die zur Identifizierung der Tabellenzeile benötigt wird.

Listing 32.6: Supplemental Logging - Demo Schritt 3

```
update "BANK"."GIROKONTO"
  set
    "GUTHABEN" = 1256,37
  where
    "KONTO_ID" = 447 and
    "GUTHABEN" = 1231,40 and
    ROWID = 'AAASWaAAHAAAADUACG',
```

Supplemental Logging aktivieren

Listing 32.7: Supplemental Logging

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA;
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (FOREIGN KEY) COLUMNS;
```

Das erste Kommando aktiviert das „Minimal supplemental logging“. Dies ist die Grundlage für die beiden anderen Varianten. Mit dem zweiten Befehl wird das Supplemental Logging für Primary Key values aktiviert. Dies ist zwingend erforderlich, um Flashback Transaction nutzen zu können.

Zeile drei enthält einen Befehl, der das Supplemental Logging für Foreign Keys values aktiviert. Dieser ist nicht in allen Fällen notwendig und sollte auch nur mit Bedacht genutzt werden, da das Supplemental Logging für Foreign Key die Redo Logs immens aufblähen kann.

Funktionsweise

Oracle Flashback Transaction Query erlaubt es zu sehen, was eine spezifische Transaktion an einer Tabellenzeile gemacht hat. Realisiert wird dieses Feature mit Hilfe der Data Dictionary View FLASHBACK_TRANSACTION_QUERY.

Um zu erfahren, welche Transaktionen Veränderungen an der Tabelle GIROKONTO vorgenommen haben, muss nur die Tabelle FLASHBACK_TRANSACTION_QUERY abgefragt werden.

Das Interessante an diesem Feature ist, dass zu jedem SQL-Statement ein Undo-SQL-Statement generiert wird. Mit dessen Hilfe kann die bereits bestehende Änderung rückgängig gemacht werden.

Listing 32.8: Die Tabelle FLASHBACK_TRANSACTION_QUERY

Name	Null?	Type
XID		RAW(8)
START_SCN		NUMBER
START_TIMESTAMP		DATE
COMMIT_SCN		NUMBER
COMMIT_TIMESTAMP		DATE
LOGON_USER		VARCHAR2(30)
UNDO_CHANGE#		NUMBER
OPERATION		VARCHAR2(32)
TABLE_NAME		VARCHAR2(256)
TABLE_OWNER		VARCHAR2(32)
ROW_ID		VARCHAR2(19)
UNDO_SQL		VARCHAR2(4000)

Listing 32.9: Informationsgewinnung mit Flashback Transaction Query

```

col logon_user format a10
col undo_sql format a50

SQL> SELECT xid, start_timestamp, commit_scn, logon_user
  2      undo_sql
  3  FROM flashback_transaction_query
  4 WHERE LOWER(table_name) LIKE 'girokonto';

XID          START_TIMESTAMP      COMMIT_SCN LOGON_USER
-----  -----
UNDO_SQL
-----
0A00130036030000 03.11.2013 14:13:09      2278483 BANK
update "BANK"."GIROKONTO" set "GUTHABEN" = '1231,40'
1' where ROWID = 'AAASWaAAHAAAADUACG';

```



- [ADFNS01005]

32.4.2 Flashback Transaction Backout

Flashback Transaction Backout ist eine Weiterentwicklung der Flashback Transaction Query. Mit Flashback Transaction Backout können Transaktionen rückgängig gemacht werden, ähnlich wie mit Flashback Transaction Query. Der Unterschied zwischen beiden ist, dass Flashback Transaction Backout in Form einer PL/SQL-Prozedur existiert, die Abhängigkeiten zwischen Transaktionen erkennt.

Abhängigkeiten zwischen Transaktionen

Die größte Gefahr beim Zurückrollen bereits bestätigter Transaktionen ist, dass zwei Transaktionen von einander abhängig sein können. Beim Rückgängigmachen einer Transaktion kann eine Zweite in Mitleidenschaft gezogen werden. Es muss in jedem Falle geprüft werden, ob das Ergebnis noch korrekte Daten wiedergibt.

Eine Transaktion B kann von einer Transaktion A auf drei unterschiedlichen Wegen abhängig sein:

- **Write-After-Write:** Bei einer solchen Abhängigkeit fügt Transaktion A eine Zeile in eine Tabelle ein, die später von Transaktion B geändert wird.

Listing 32.10: Eine Write-After-Write Abhängigkeit

```
-- Transaktion A
SQL> UPDATE TABLE girokonto
  2  SET      Guthaben = 1256.80
  3  WHERE   Konto_ID = 447;
SQL> COMMIT;

-- Transaktion B
SQL> UPDATE TABLE girokonto
  2  SET      Guthaben = 1481.16
  3  WHERE   Konto_ID = 447;
SQL> COMMIT;
```

- **Primary Key Abhängigkeit:** Transaktion A löscht eine Zeile aus einer Tabelle. Der Wert des Primärschlüssels dieser Zeile war „n“. Anschließend fügt Transaktion B eine Zeile in die Tabelle ein. Der Wert des Primärschlüssels der neuen Zeile ist ebenfalls „n“. Beide Zeilen haben somit den gleichen Wert als Primärschlüssel.

Listing 32.11: Eine Primary Key Abhängigkeit

```
-- Transaktion A
SQL> DELETE girokonto
  2  WHERE   Konto_ID = 447;
SQL> COMMIT;

-- Transaktion B
SQL> INSERT INTO girokonto
  2  VALUES (447, 10.5, 1256.37, 10, 15000);
SQL> COMMIT;
```

- **Foreign Key Abhängigkeit:** Transaktion A erstellt eine Zeile in Tabelle A. Transaktion B erstellt eine von Tabelle A abhängige Zeile in Tabelle B.

Beispiel ?? zeigt, dass die in Tabelle GIROKONTO erstellte Zeile, durch ein Foreign Key Constraint zwischen den beiden Tabellen KONTO und GIROKONTO, von der Tabelle KONTO abhängig ist.

Listing 32.12: Eine Foreign Key Abhängigkeit

```
-- Transaktion A
SQL> INSERT INTO Konto
  2  VALUES (666, 'DE2355880228095276211', NULL);

SQL> COMMIT;

-- Transaktion B
SQL> INSERT INTO Girokonto
  2  VALUES (666, 10.5, 2400, 10, 15000);
```

```
SQL> COMMIT;
```

Flashback Transaction Backout kann alle drei Abhängigkeiten erkennen und den Nutzer mit einer Fehlermeldung warnen, bevor eine Transaktion rückgängig gemacht wird.

Die einzige Angabe, die benötigt wird, um eine Transaktion zurückzurollen, ist die XID der betroffenen Transaktion. Diese kann mit Flashback Version Query oder Flashback Transaction Query ermittelt werden.

Listing 32.13: Die XID einer Transaktion ermitteln

```
SQL> SELECT xid, start_timestamp, logon_user
  2  FROM flashback_transaction_query
  3 WHERE LOWER(table_name) LIKE 'girokonto';

XID           START_TIMESTAMP      LOGON_USER
-----  -----
02000D0056040000 03.11.2013 14:20:10 BANK
04000D0048030000 03.11.2013 14:00:59 BANK
05001400F6040000 03.11.2013 13:41:55 BANK
05000000F6040000 03.11.2013 13:20:49 BANK
0600020049040000 03.11.2013 14:05:25 BANK
0800020049040000 03.11.2013 14:00:41 BANK
0800210048040000 03.11.2013 13:21:22 BANK
090020005B040000 03.11.2013 11:09:52 SYS
0900190059040000 03.11.2013 10:56:50 SYS
0A00130036030000 03.11.2013 14:13:09 BANK

10 rows selected.
```

Flashback Transaction Backout kann eine oder mehrere Transaktionen gleichzeitig zurückrollen. Beispiel ?? zeigt die Anwendung der Prozedur TRANSACTION_BACKOUT.

Listing 32.14: Eine bestätigte Transaktion zurückrollen

```
SQL> BEGIN
  2  DBMS_FLASHBACK.TRANSACTION_BACKOUT (
  3    numberofxids => 1,
  4    xids          => xid_array('02000D0056040000'),
  5    options        => DBMS_FLASHBACK.NOCASCADE
  6  );
  7  END;
  8 /
```

Die Prozedur TRANSACTION_BACKOUT wird in Beispiel ?? mit drei Parametern aufgerufen:

- **numberofxids:** Die Anzahl der Transaktionen die zurückgerollt werden sollen.

- **xids:** Ein Array mit den XIDs aller zurückzurollenden Transaktionen
- **options:** Optionen die das Verhalten von TRANSACTION_BACKOUT steuern.

Es gibt folgende Optionen:

- **NOCASCADE**: Der Vorgang wird beim Auftreten der ersten Abhängigkeit sofort abgebrochen.
- **NOCASCADE_FORCE**: Trotz Abhängigkeiten wird die Transaktion rückgängig gemacht. Sollte dabei kein Constraint verletzt werden, kann der Vorgang erfolgreich abgeschlossen werden.
- **NONCONFLICT_ONLY**: Es werden nur die Transaktionen zurückgerollt, die keine Abhängigkeiten aufweisen. Die anderen bleiben bestehen. Dies löst kein Problem mit der Datenbankkonsistenz aus, jedoch muss sehr genau geprüft werden, ob das Ergebnis den eigenen Vorstellungen entspricht.
- **CASCADE**: Mit dieser Option werden alle betroffenen Transaktionen der Reihe nach zurückgerollt.

Einschränkungen

Flashback Transaction Backout hat folgende Einschränkungen:

- DDL Operationen können nicht rückgängig gemacht werden.
- Transaktionen mit LOB Typen (CLOB, BLOB, BFILE, NCLOB) können nicht rückgängig gemacht werden.
- Die Undo Segmente im Undo Tablespace müssen groß genug sein.



- [dflashbhtm]
- [ADFNS01009]

32.5 Oracle Flashback Table

Oracle Flashback Table ermöglicht es dem DBA eine Tabelle, ohne kompliziertes Recovery, in einen durch einen Zeitpunkt oder eine SCN bestimmten Zustand zurückzuversetzen. Flashback Table ersetzt dadurch meist ein wesentlich komplizierteres Point-In-Time Recovery.

Folgende Voraussetzungen müssen für den Einsatz von Flashback Table geschaffen werden:

- Einschalten von *Row Movement* für die betreffenden Tabellen.

Listing 32.15: Einschalten von Row Movement

```
SQL> ALTER TABLE mitarbeiter ENABLE ROW MOVEMENT
```

- Der Nutzer muss das Systemprivileg flashback any table haben.
- Der Nutzer muss die Objektpflegeprivilegien select, insert, delete und alter auf die betreffenden Tabellen haben.
- Es müssen genügend Undo-Informationen verfügbar sein.

Die beiden folgenden Beispiele zeigen, wie die Tabelle MITARBEITER, mit Hilfe einer SCN bzw. eines Zeitstempels in einen vergangenen Zustand zurückversetzt wird.

Listing 32.16: Flashback Table mit SCN

```
SQL> FLASHBACK TABLE mitarbeiter TO SCN 1000;
```

Listing 32.17: Flashback Table mit Zeitstempel

```
SQL> FLASHBACK TABLE mitarbeiter TO TIMESTAMP
2      TO_TIMESTAMP('03.11.2013 16:43:25', 'DD.MM.YYYY HH:MI:SS');
```

32.6 Oracle Flashback Drop

Mit Hilfe von Flashback Drop kann das Löschen einer Tabelle rückgängig gemacht werden. Flashback Drop ist schneller als jede Form von Point-In-Time Recovery, denn seit Oracle 10g wird eine Tabelle beim Löschen nicht mehr sofort entfernt, sondern:

1. Die Tabelle wird umbenannt und erhält einen Recycle Bin Namen, wie zum Beispiel `BIN$6kkdf0AiG1jgQGRkFBQQiQ==`
2. Sie wird in einen Speicherbereich verschoben, der sich „Recycle Bin“ nennt.

Der Flashback Drop macht diese beiden Schritte wieder rückgängig.

32.6.1 Der Recycle Bin

Der Recycle Bin ist ein logischer Kontainer für Tabellen und deren abhängige Objekte. Wird eine Tabelle gelöscht, speichert die Datenbank diese dann mit all ihren abhängigen Objekten im Recycle Bin, so dass die Tabelle später wiederhergestellt werden kann.

Funktionsweise des Recycle Bin

Tabellen erhalten einen neuen Namen und werden solange im Recycle Bin gespeichert, bis dieser bereinigt wird. Die Namensgebung im Recycle Bin folgt diesem Schema:

`BIN$globalUID$version`

„`globalUID`“ steht dabei für eine Zeichenkette aus 24 Zeichen, die in der ganzen Datenbank eindeutig ist. „`version`“ ist die Versionsnummer (laufende Nummer) der Tabelle. Die Namen im Recycle Bin sind immer 30 Zeichen lang.

Die Bereinigung des Recycle Bin geschieht automatisch, kann aber auch manuell durchgeführt werden. Der Recycle Bin wird in den folgenden Situationen automatisch bereinigt:

- wenn der Tablespace, in dem sich das betreffende Objekt befindet keinen freien Speicher mehr hat und wachsen müsste und
- wenn die Quota, die der Besitzer des Objekts auf den Tablespace hat, in dem sich das Objekt befindet, ausgereizt ist

Die Objekte werden dann nach dem First-In-First-Out Prinzip endgültig gelöscht.

Soll der Recycle Bin manuell entleert werden, muss das SQL-Kommando `PURGE` verwendet werden. Es wird auf Objekte angewandt, die sich bereits im Recycle Bin befinden. Für seine Ausführung kann der Objektname oder auch der Recycle Bin Name genutzt werden.

Listing 32.18: Das Kommando `PURGE`

```
-- Tabellen entfernen
SQL> PURGE TABLE mitarbeiter;

-- Oder
SQL> PURGE TABLE "BIN$6kkdf0AiG1jgQGRkFBQQiQ==$0";

-- Indizes entfernen
SQL> PURGE INDEX "BIN$4fkefoBjG1jgQGRkFBQQiQ==$0";

-- Tablespace entfernen
SQL> PURGE TABLESPACE example;
```

Will ein Nutzer alle Objekte, die er gelöscht hat, aus dem Recycle Bin entfernen, kann er dies mit dem folgenden Kommando tun:

Listing 32.19: Das Kommando `PURGE RECYCLEBIN`

```
SQL> PURGE RECYCLEBIN;
```

Für den DBA gibt es zusätzlich das Kommando `PURGE DBA_RECYCLEBIN`, mit dem alle Objekte aus dem Recycle Bin, unabhängig von deren Besitzer, gelöscht werden.

Listing 32.20: Das Kommando `PURGE DBA_RECYCLEBIN`

```
SQL> PURGE DBA_RECYCLEBIN;
```

Um eine Tabelle so zu löschen, dass sie erst gar nicht in den Recycle Bin verschoben wird, kann das `DROP TABLE`-Statement um das Schlüsselwort `PURGE` erweitert werden.

Listing 32.21: `DROP TABLE PURGE`

```
SQL> DROP TABLE employees PURGE;
```

Privilegien für die Nutzung von Flashback Drop

Für Flashback Drop und Purge Operationen gelten die folgenden Regeln:

- Ein Nutzer, der das drop-Privileg auf eine Tabelle hat, kann diese Löschen und somit im Recycle Bin platzieren.
- Will ein Nutzer eine Flashback Drop-Operation ausführen, benötigt er das drop-Privileg für das betreffende Objekt.
- Soll das Objekt endgültig mit `PURGE` entfernt werden, benötigt der Nutzer dazu ebenfalls nur das drop-Privileg für das betreffende Objekt.

Den Inhalt des Recycle Bin anzeigen

Um sich den Inhalt des Recycle Bin anzeigen zu lassen, können entweder die beiden Views `USER_RECYCLEBIN`, `DBA_RECYCLEBIN` oder das SQL*Plus-Kommando `show recyclebin` verwendet werden. Die Ausgabe des `show recyclebin`-Kommandos sieht so aus:

Listing 32.22: `show recyclebin`

ORIGINAL NAME	RECYCLEBIN NAME	TYPE	DROP TIME
MITARBEITER	BIN\$6kkdfoAiGljgQGRkFBQQiQ==\$0	TABLE	2013-03-11 17:16:59

Zusätzlich zu USER_RECYCLEBIN und DBA_RECYCLEBIN, können die Data Dictionary Views USER_TABLES, ALL_TABLES, DBA_TABLES, USER_INDEXES, ALL_INDEXES und DBA_INDEXES darüber Auskunft geben, welche Objekte gelöscht wurden. In allen diesen Views existiert eine Spalte DROPPED. Wurde ein Objekt gelöscht, taucht es nach wie vor in diesen Views auf, nur die Spalte DROPPED wird auf YES gesetzt.

Den Recycle Bin ein- und ausschalten

Standardmäßig ist der Recycle Bin eingeschaltet. Über den Initialisierungsparameter `recyclebin` kann er ein- und ausgeschaltet werden. Der Parameter nimmt die beiden Werte „ON“ und „OFF“ entgegen. `recyclebin` kann sowohl Sessionweit mit `ALTER SESSION` als auch Systemweit mit `ALTER SYSTEM` geändert werden.

Listing 32.23: Den Recycle Bin ausschalten

```
SQL> ALTER SYSTEM
  2  SET recyclebin=off;
```

Die Auswirkungen der Änderung werden erst nach einer Neuanmeldung für die Nutzer spürbar.

Regeln und Einschränkungen für die Nutzung des Recycle Bin

- Die Flashback Drop-Funktionalität ist nur für Objekte in einem Nicht-System und lokal verwalteten Tablespace verfügbar.
- Es kann nicht bestimmt werden, wie lange sich ein Objekt im Recycle Bin befinden soll.
- Abfragen auf Objekte im Recycle Bin sind erlaubt, DML und DDL Operationen aber nicht.
- Flashback Queries auf Objekte im Recycle Bin können nur mit deren Recycle-Bin-Namen ausgeführt werden.
- Wird eine Tabelle in den Recycle Bin verschoben, werden automatisch alle von ihr abhängigen Objekte mit verschoben.
- Fremdschlüssel gehen beim Verschieben eines Objekts in den Recycle Bin verloren und werden beim Wiederherstellen des Objekts nicht wiederhergestellt.

32.6.2 Einen Flashback Drop ausführen

Ein Flashback Drop kann sowohl mit dem Originalnamen, als auch mit dem Recycle Bin Namen des Objekts ausgeführt werden. Das folgende Beispiel zeigt diese beiden Möglichkeiten für die Tabelle `MITARBEITER`.

Listing 32.24: Ausführen eines Flashback Drop

```
SQL> DROP TABLE mitarbeiter;

SQL> FLASHBACK TABLE mitarbeiter TO BEFORE DROP;

SQL> FLASHBACK TABLE "BIN$gk3lsj/3akk5hg3j21k15j3d==$0" TO BEFORE DROP;
```

Ein im Recycle Bin befindliches Objekt kann bei seiner Wiederherstellung direkt umbenannt werden. Hierzu wird das `FLASHBACK TABLE ... TO BEFORE DROP`-Statement um die Klausel `RENAME TO` wie folgt erweitert:

Listing 32.25: Ausführen eines Flashback Drop mit gleichzeitiger Umbenennung

```
SQL> FLASHBACK TABLE mitarbeiter TO BEFORE DROP
  2 RENAME TO bank.angestellte;

SQL> FLASHBACK TABLE "BIN$gk3lsj/3akk5hg3j21k15j3d==$0" TO BEFORE DROP
  2 RENAME TO bank.angestellte;
```

32.7 Oracle Total Recall

Immer mehr Unternehmen haben das Problem, dass eine immer größer werdende Datenflut für sehr lange Zeit vorgehalten werden muss. Teils geschieht dies aufgrund von gesetzlichen Vorgaben (z. B. 10 Jahre im Bereich Gewerbesteuer), teils ist dies notwendig, um Projekte bis in ihre Anfänge zurückverfolgen zu können.

Je größer die Datenmengen, desto höher sind auch die Anforderungen an die Datenbank, Abfragen performant bearbeiten zu können. Um diesem Problem wirkungsvoll entgegentreten zu können, hat Oracle, in der Version 11g R2 seiner Datenbank, die Option „Total Recall“ eingeführt. Diese Option enthält das Feature „Flashback Data Archive“, mit dessen Hilfe historische Daten außerhalb der Datenbank gelagert werden können, aber mittels Flashback Query im direkten Zugriff der Anwendung bleiben.

32.7.1 Flashback Data Archive (FBDA)- Architektur

FBDA ist eine optionale und für Anwendungen völlig transparente Technologie. Es speichert Daten in „Historientabellen“, die in eigenen Tablespaces angelegt werden. Um Speicherplatz zu sparen und die Performance von Abfragen zu erhöhen, sind Historientabellen partitioniert und komprimiert.

Befüllt werden die Historientabellen durch einen neuen Hintergrundprozess, den „FBDA“ - Flashback Data Archiver. Dieser Prozess erwacht in bestimmten Zeitabständen und durchforstet den Undo Tablespace nach Beforeimages, die archiviert werden müssen. Seine „Schlafenszeit“ justiert der FBDA automatisch nach dem Transaktionsaufkommen (Standardzeitintervall = 5 Minuten).

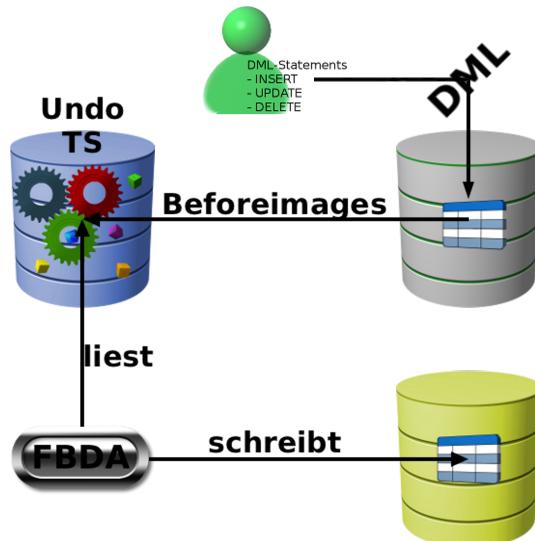


Abb. 32.1:
Flashback Data
Archive -
Architektur



Nur die beiden DML-Befehl **UPDATE** und **DELETE** beeinflussen das Flashback Data Archive, da **INSERT**-Kommandos keine Beforeimages erzeugen!

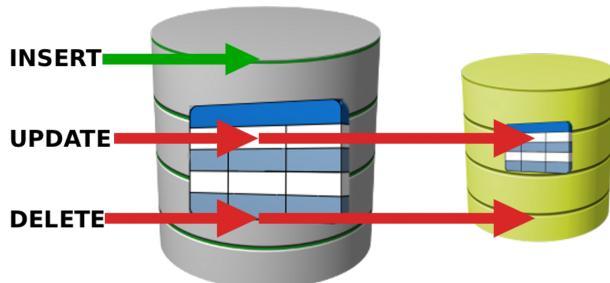


Abb. 32.2:
DML und
Flashback Data
Archive

32.7.2 Flashback Data Archive administrieren

Voraussetzungen

Um Flashback Data Archive einrichten zu können, müssen die folgenden Voraussetzungen gegeben sein:

- Der Tablespace für das Flashback Data Archive muss Automatic Segment Space Management nutzen
- Das Automatic Undo Management muss aktiviert sein.

Privilegien

Es gibt drei Privilegien, welche im Zusammenhang mit der Arbeit mit Flashback Data Archive stehen:

- `flashback archive administer`: Erlaubt das Erstellen und Verwalten von FBDAAs.
- `flashback archive`: Ermöglicht das Einschalten FBDA für eine Tabelle.
- `sysdba`: Beinhaltet `flashback archive administer` und `flashback archive`

Archive erstellen

Das Erstellen eines Archives geschieht mit dem Kommando `CREATE FLASHBACK ARCHIVE`. Für die Archive sollte ein eigenständiger Tablespace erstellt werden.

Listing 32.26: Ein Flashback Data Archive anlegen

```
SQL> CREATE FLASHBACK ARCHIVE fbda_bank_1
  2  TABLESPACE bank
  3  QUOTA      10G
  4  RETENTION  5 YEAR;
```

Die Klausel `TABLESPACE bank` legt fest, dass das Archiv FBDA_BANK_1 für den Tablespace BANK angelegt wird. Mit `QUOTA 10G` wird die Speicherplatzquota für das Archiv angegeben. Diese wirkt sich genauso aus, wie die `MAXSIZE`-Klausel bei Tablespaces. Mit `RETENTION 5 YEAR` wird die Verweildauer der Daten im FBDA bestimmt. Daten die älter sind, als die angegebene Retention werden automatisch aus dem Archiv entfernt.



Wird die `QUOTA`-Klausel nicht angegeben gilt automatisch `QUOTA unlimited`!

Beim Anlegen eines FBDA ist es möglich, mit Hilfe des Schlüsselwortes `DEFAULT` ein Standard Flashback Data Archive zu erzeugen. Wann dies von Nutzen ist, wird in ?? erläutert.

Listing 32.27: Ein Default Flashback Data Archive anlegen

```
SQL> CREATE FLASHBACK ARCHIVE DEFAULT fbda_bank_def
  2  TABLESPACE archive
  3  QUOTA      10G
  4  RETENTION   5 YEAR;
```

Das ein Flashback Data Archive angelegt wurde, kann mittels der Data Dictionary View DBA_FLASHBACK_ARCHIVE geprüft werden.

Listing 32.28: DBA_FLASHBACK_ARCHIVE abfragen

```
SQL> col flashback_archive_name format a30
SQL> SELECT flashback_archive_name , retention_in_days , status
  2  FROM    dba_flashback_archive

FLASHBACK_ARCHIVE_NAME          RETENTION_IN_DAYS STATUS
-----  -----
FBDA_BANK_1                      1825
FBDA_BANK_DEF                    1825 DEFAULT
```

Flashback Data Archive aktivieren

Das Aktivieren von FBDA erfolgt auf Tabellenebene.

Listing 32.29: Flashback Data Archive aktivieren

```
SQL> ALTER TABLE bank.buchung
  2  FLASHBACK ARCHIVE fbda_bank_1;
```

Die Tabelle BUCHUNG ist nun mit dem Flashback Data Archive FBDA_BANK_1 verknüpft. Für sie wird eine Historientabelle im Archiv angelegt.

Listing 32.30: Wo ist die Historientabelle?

```
SQL> col table_name format a20
SQL> col flashback_archive_name format a20
SQL> col archive_table_name format a20

SQL> SELECT table_name , flashback_archive_name , archive_table_name , status
  2  FROM    dba_flashback_archive_tables

TABLE_NAME          FLASHBACK_ARCHIVE_NA ARCHIVE_TABLE_NAME     STATUS
-----  -----
BUCHUNG              FBDA_BANK_1           SYS_FBA_HIST_75155  ENABLED
```

Die View DBA_FLASHBACK_ARCHIVE_TABLES enthält für jede Tabelle, für die FBDA aktiviert wurde, einen Eintrag. Die Historientabelle für BANK.BUCHUNG wurde automatisch, unter dem Namen SYS_FBA_HIST_75155 angelegt.

 Die Historientabelle wird erst dann erzeugt, wenn der FBDA-Hintergrundprozess anfängt, seine Arbeit zu machen!

Die interne Struktur der Historientabelle sieht so aus:

Listing 32.31: Die Struktur der Historientabelle

Name	Null?	Type
RID		VARCHAR2(4000)
STARTSCN		NUMBER
ENDSCN		NUMBER
XID		RAW(8)
OPERATION		VARCHAR2(1)
BUCHUNGS_ID		NUMBER
BETRAG		NUMBER(12,2)
BUCHUNGSDATUM		DATE
KONTO_ID		NUMBER
TRANSAKTIONS_ID		NUMBER

Zusätzlich zu den Spalten der Tabelle BUCHUNG werden noch weitere Spalten für Metadaten mitgeführt.

In Beispiel ?? wird Flashback Data Archive, unter Angabe eines Archivs aktiviert. Die Archivangabe kann aber auch entfallen.

Listing 32.32: Flashback Data Archive mit einem Default Archive aktivieren

```
SQL> ALTER TABLE bank.buchung
  2 FLASHBACK ARCHIVE;
```

Das Statement in Beispiel ?? aktiviert Flashback Data Archive für die Tabelle BUCHUNG, wenn vorher ein Default Flashback Data Archive angelegt wurde. Existiert kein Default Archive, schlägt dieses Statement fehl.

Listing 32.33: Welche Archive wurden benutzt?

```
SQL> col table_name format a20
SQL> col flashback_archive_name format a20
SQL> col archive_table_name format a20
```

```
SQL> SELECT table_name , flashback_archive_name , archive_table_name , status  
2  FROM    dba_flashback_archive_tables;
```

TABLE_NAME	FLASHBACK_ARCHIVE_NA	ARCHIVE_TABLE_NAME	STATUS
BUCHUNG	FBDA_BANK_DEF	SYS_FBA_HIST_75155	ENABLED

Abfragen des Flashback Archives

Die Benutzung der Flashback Archive geschieht mit Hilfe der **AS OF**- und **VERSIONS BETWEEN**-Klauseln.

Listing 32.34: Flashback Data Archive mit einem Default Archive aktivieren

```
SQL> SELECT *
  2  FROM    buchung AS OF TIMESTAMP
  3      TO_TIMESTAMP('03.11.2010 09:30:00', 'DD.MM.YYYY HH24:MI:SS')
```

Sofern das Flashback Archive bereits lange genug existiert, werden die Daten so angezeigt, wie sie zum 03.11.2010, um 09:30 vorlagen.

Flashback Archive bearbeiten

Flashback Archive können im laufenden Betrieb verändert werden. Folgende Veränderungen sind möglich:

- Verändern der Größe
- Ein Archive zum Standardarchiv machen
- Verändern des Parameters **RETENTION**
- Historische Daten löschen

Die Größe eines Archives kann auf zwei Arten geändert werden. Die Quota an einem Tablespace kann erhöht oder ein weiterer Tablespace hinzugefügt werden.

Listing 32.35: Die Quota eines Archives auf 20GB erhöhen

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_def
  2  MODIFY TABLESPACE archive QUOTA 20G;
```

Listing 32.36: Dem Archiv einen weiteren Tablespace hinzufügen

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_def
  2  ADD TABLESPACE archive2 QUOTA 10G;
```

Verkleinert wird ein Archiv, in dem ihm ein Tablespace entzogen oder die Quotas verringert werden.

Listing 32.37: Einen Tablespace aus dem Archiv entfernen

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_def
  2  REMOVE TABLESPACE archive2;
```

Listing 32.38: Die Quota eines Tablespaces verringern

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_def
  2 ADD TABLESPACE archive2 QUOTA 5G;
```



Das Entfernen eines Tablespaces aus einem Archiv funktioniert nur, wenn der Speicherplatz im betroffenen Tablespace noch nicht belegt ist.

Mit Hilfe der `SET DEFAULT`-Klausel kann ein Archiv zum Standardarchiv gemacht werden.

Listing 32.39: Ein Archiv zum Standardarchiv machen

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_1
  2 SET DEFAULT;
```

Um die Retention eines Archives zu verändern, wird die `MODIFY RETENTION`-Klausel verwendet.

Listing 32.40: Die Vorhaltezeit eines Archives verändern

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_1
  2 MODIFY RETENTION 3 YEAR;
```

Archive manuell bereinigen

Oracle bereinigt den Inhalt der Flashback Archives automatisch. Dennoch hat der Administrator die Möglichkeit, den Inhalt manuell zu löschen.

Listing 32.41: Den gesamten Inhalt eines Archives löschen

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_1
  2 PURGE ALL;
```

Listing 32.42: Alle Einträge löschen, die älter als ein Jahr und sechs Monate sind

```
SQL> ALTER FLASHBACK ARCHIVE fbda_bank_1
  2 PURGE BEFORE TIMESTAMP (SYSTIMESTAMP - INTERVAL '1-6' YEAR TO MONTH);
```

Flashback Data Archive deaktivieren

Flashback Data Archive kann jeder Zeit für eine Tabelle deaktiviert werden.



Beim Deaktivieren von Flashback Archive wird die Historientabelle gelöscht!

Listing 32.43: Flashback Data Archive deaktivieren

```
SQL> ALTER TABLE bank.buchung
  2 NO FLASHBACK ARCHIVE;
```

Flashback Archive löschen

Beim Löschen eines Flashback Archives wird die Historientabelle gelöscht, der Tablespace, in dem sie lag, bleibt erhalten.

Listing 32.44: Flashback Data Archive deaktivieren

```
SQL> DROP FLASHBACK ARCHIVE fbda_bank_1;
```

32.8 Informationen

32.8.1 Verzeichnis der relevanten Initialisierungsparameter



- [REFRN10264]

32.8.2 Verzeichnis der relevanten Data Dictionary Views



- [ADFNS01005]
- [REFRN23342]
- [sthref2545]
- [sthref2202]
- [REFRN23719]
- [sthref2010]

Teil IV

Appendix

Oracle Database 11g Release 2