SQL – Die Abfragesprache für Datenbanken

INHALTSVERZEICHNIS

Kapitel 1Was ist eine Datenbank?

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie wissen, was eine Datenbank ist
- Sie kennen den Unterschied zwischen strukturierten und unstrukturierten Daten

Eine Datenbank lässt sich als die dauerhafte Ablage von Informationen in einer festgelegten Struktur beschreiben, um diese auf einfache Weise anlegen, wieder zurückgewinnen und auswerten zu können.

Diese Beschreibung gilt nicht erst seit der Einführung von EDV-Systemen, sondern gilt für jegliche Form der Archivierung von Informationen. So kann man auch einen Karteikasten mit Karteikarten als Datenbank verstehen.

Auch bei der manuellen Verwaltung ist die strukturierte Ablage der Daten wichtig für Interpretation der Information.

Beispiel:

Manfred	Müller	Essen	Frankfurt
Erwin	Otto	Gladbeck	Köln

Aus dieser Tabelle lässt sich die enthaltene Information nicht eindeutig entnehmen, da uns die Bedeutung der Spalten, also die Struktur der Daten, nicht bekannt ist.

So ist es möglich:

Vorname	Nachname	Geburtsort	Wohnort
Manfred	Müller	Essen	Frankfurt
Erwin	Otto	Gladbeck	Köln

Gerade bei EDV-Systemen, bei denen ja die Informationen lediglich als „1“ oder „0“ vorliegen, ist ohne die Kenntnis der Datenstruktur die Rückgewinnung der Informationen unmöglich.

So kann die Bitfolge

0100110101010010

als eine Zahl (19794)
oder als zwei Buchstaben (MR)
interpretiert werden

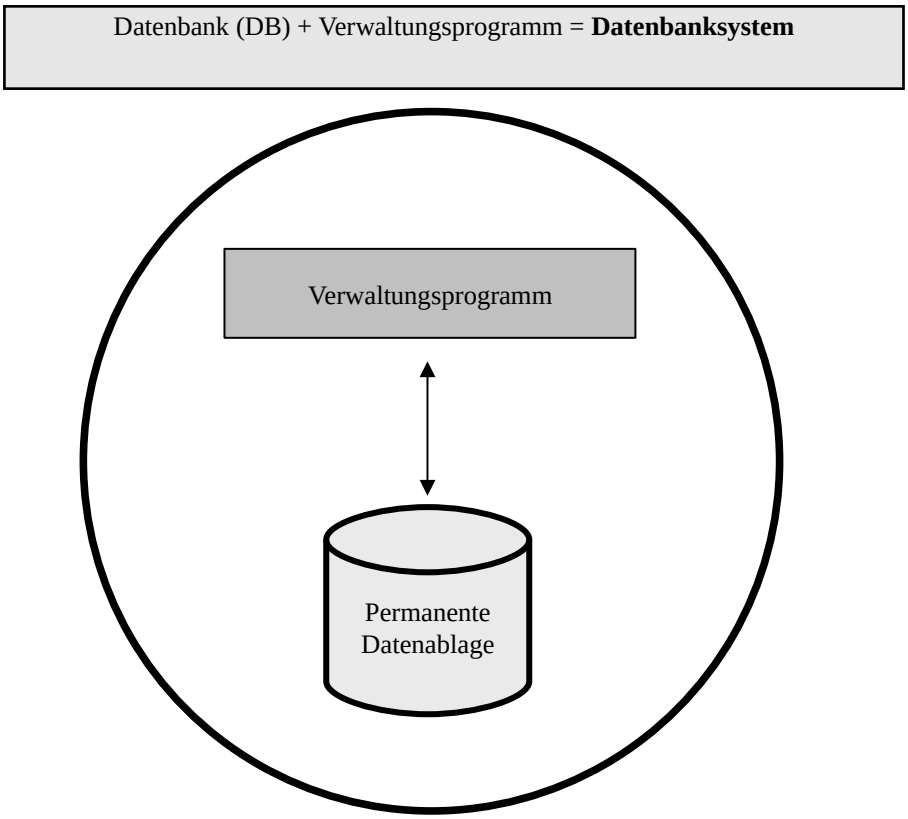
Kapitel 2 Was ist ein Datenbanksystem?

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie wissen, was ein Datenbanksystem kennzeichnet
- Sie können Stammdaten und Bewegungsdaten voneinander unterscheiden

Neben der Datenablage benötigt man, bei einem EDV-gestützten Datenbanksystem, noch ein Verwaltungsprogramm, mit dem die Datensammlung organisiert, kontrolliert und ausgewertet wird. Ohne das Verwaltungsprogramm ist die eigentliche Datenbank wertlos, da die Informationen nicht wiedergewonnen werden können.

Im einfachsten Fall besteht ein Datenbanksystem also aus zwei Komponenten.



Bei den Daten unterscheidet man zwischen Stammdaten und Bewegungsdaten.

- **Stammdaten** sind Daten, die in geringerer Anzahl vorhanden sind und deren Datenbestand sich selten oder gar nicht ändern. Beispiele hierzu sind Artikel- oder Kundendaten.
- **Bewegungsdaten** sind Daten, die in größerer Anzahl vorhanden sind und die häufig ergänzt werden. Beispiele hierzu sind Bestell- oder Rechnungsdaten.

Kapitel 3 Anforderungen an ein Datenbanksystem

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie kennen die einzelnen Anforderungen an ein Datenbanksystem
- Sie wissen, wie redundante Daten und Dateninkonsistenzen vermieden werden
- Sie kennen den Unterschied zwischen Datenschutz und Schutz vor Datenverlust
- Sie wissen, wie parallele Zugriffe/Mehrbenutzerbetrieb auf Datenbestände ablaufen

Infolge unkontrolliert gewachsener Datenbestände ist fast überall ein Datenchaos entstanden, das bewältigt werden muss. Nicht nur die Informationsmenge ist gewachsen, sondern die Informationen können zunehmend widersprüchlich sein, da die Datenaktualisierung (Datenpflege) kaum Schritt halten kann.

Es kommt also darauf an, dass sich

- die gesuchten Informationen schnell wieder finden lassen
- die abgelegten Daten komfortabel pflegen lassen

Um große Datenbestände effektiv verwalten zu können, benötigt man ein Datenbanksystem, das zusätzlich folgende Anforderungen erfüllt:

- Datenunabhängigkeit
- Gewährleistung des Datenschutzes
- Vermeidung von redundanten Daten
- Vermeidung von Dateninkonsistenzen
- Parallele Zugriffe auf den Datenbestand
- Schutz vor Datenverlust
- Möglichkeit der Verteilung der Datenbank auf mehrere Speicherorte

Im Folgenden werden diese Anforderungen näher betrachtet.

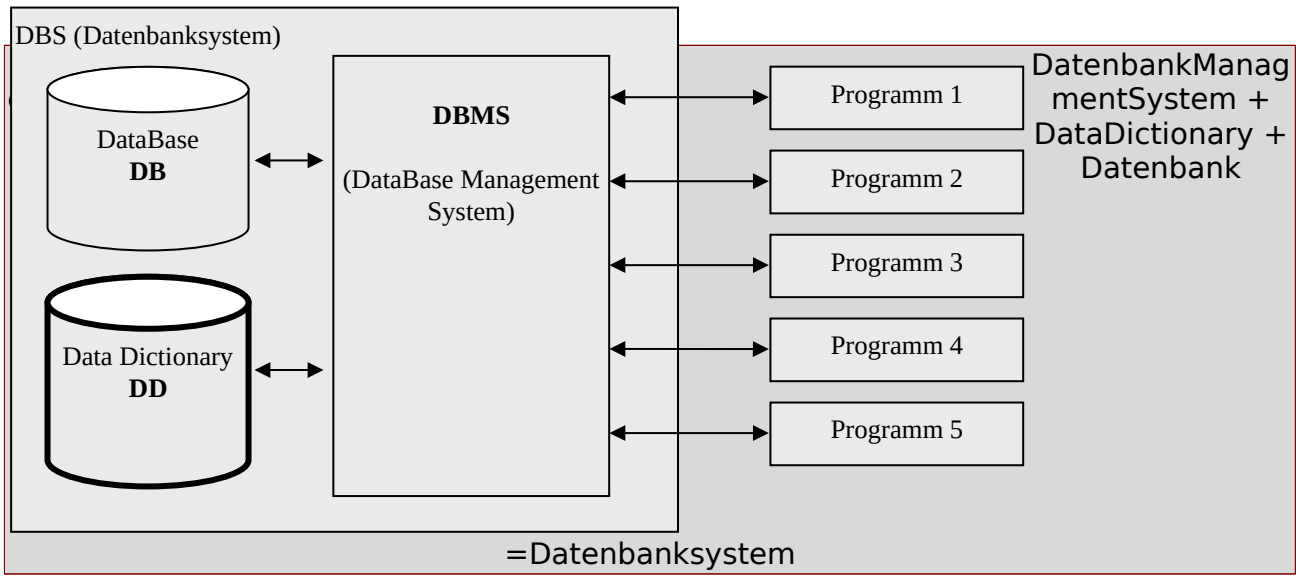
3.1 Datenunabhängigkeit

Wenn man die Daten in einer Datei ablegt, ist man gezwungen ein festes Format zu nutzen. Dieses bedeutet, dass beim Programmieren der Anwendung die Struktur der Datenablage, d.h. die Feldanordnung und Feldlänge, in der Anwendung integriert werden. Bei Erweiterung der Anwendung muss somit auch das Dateiformat verändert werden. Das daraus entstehende Problem nennt man *Datenabhängigkeit*.

Das Problem kennen wir von den Office-Anwendungen. So kann zum Beispiel ein Textdokument im Word 2007-Format von älteren Word-Versionen nicht gelesen werden.

Durch den Einsatz eines Datenbanksystems wird die Unabhängigkeit der Programme von den Daten angestrebt. Das bedeutet, dass der Entwickler nicht mehr wissen muss, in welchem physischen Dateien und Dateiformaten seine Daten abgelegt werden.

Die Realisierung wird durch die so genannte 3-Ebenen-Architektur ermöglicht. Hierbei werden die Darstellung der Daten, deren logische Organisation und die physikalische Speicherung getrennt. Dabei werden die Beschreibung der Struktur der Daten und die Regeln für den Zugriff auf diese in einem weiteren Speicher ausgelagert. Dieser wird als **Data Dictionary** bezeichnet



3.2 Datenschutz

Unter Datenschutz versteht man den Schutz der Daten vor unerlaubten Zugriff. Es muss gewährleistet werden, dass Benutzer nur Daten lesen und ändern dürfen, zu denen ihnen Zugriffsrechte eingeräumt wurden.

Stellen wir uns hierzu vor, dass die Daten von Mitarbeitern in Form von Tabellen verwaltet werden sollen. Diese Daten sollen von der Gehaltsbuchhaltung und den Abteilungssekretariaten genutzt werden. Dies könnte wie folgt realisiert werden:

Vorname	Nachname	Geburtsdatum	Anschrift	Abt.	Eintrittsdatum	Gehalt
Hanns	Otto	22.07.53	Mühlengasse 12 45789, Mülheim	CON	01.04.96	3.600,00 €
Pirmin	Zübli	12.05.74	Teichstr. 7 35356, Hassberg	ZE8	01.07.06	2.450,00 €
Astrid	Zübli	29.06.76	Teichstr. 7 35356, Hassberg	CON	15.03.96	2.500,00 €
Karl-Heinz	Brenner	04.12.60	Waldweg 7b 22456 Erlenhain	GF	02.01.90	5.500,00 €

Diese Tabelle enthält alle Informationen, die für die Personalabteilung zur Gehaltsabrechnung benötigt werden.

Selbstverständlich kann die Tabelle in dieser Form nicht an die Abteilungen weitergegeben werden, da sie die Daten aller Abteilungen sowie das Gehalt enthält. Zur Lösung könnte man die Daten in weitere Tabellen kopieren, die dann den Abteilungen zugänglich gemacht werden.

Die Abteilung CON enthält somit die Tabelle:

Vorname	Nachname	Geburtsdatum	Anschrift	Abt.	Eintrittsdatum
Hanns	Otto	22.07.53	Mühlengasse 12 45789, Mülheim	CON	01.04.96
Astrid	Zübli	29.06.76	Teichstr. 7 35356, Hassberg	CON	15.03.96

Dieses Vorgehen sichert den Datenschutz. Aber wir erhalten durch die Aufteilung in mehrere Datenbestände ein neues Problem: Unser Datenbestand ist nun *redundant.*

Ein Datenbanksystem muss in der Lage sein, die Anforderungen an den Datenschutz zu erfüllen, ohne die Datenbestände mehrfach Vorhalten zu müssen. Dies kann dadurch geschehen, dass man **eine** Tabelle mit allen Daten beibehält, aber den Benutzern nur den Zugriff auf bestimmte Daten erlaubt. Dies geschieht z.B. durch *Sichten* (engl. *Views*) oder *Abfragen*.

3.3 Vermeidung von Redundanzen

Das mehrfache Vorhandensein derselben Information wird als ***Redundanz*** bezeichnet.

Redundante Daten sind zu vermeiden, da diese die Pflege des Datenbestandes erschweren, oder zu widersprüchlichen Daten, so genannten *Inkonsistenzen*, führen.

Wenn Astrid und Pirmin Zübli umziehen, muss diese Änderung in drei Tabellen durchgeführt werden: Die Abteilungen CON, ZE8 und die Personalabteilung müssen diese Anschriftenänderung vornehmen. Dieses bedeutet natürlich einen höheren Aufwand, als wenn die Daten nur in einer Tabelle vorhanden wären.

3.4 Vermeidung von Dateninkonsistenzen

Falls eine der Abteilungen die Änderung der Daten nicht oder fehlerhaft vornimmt, kommt es zu Unstimmigkeiten im Datenbestand.

Personalabteilung:

Vorname	Nachname	Geburtsdatum	Anschrift	Abt.	Eintrittsdatum	Gehalt
Hanns	Otto	22.07.53	Mühlengasse 12 45789, Mülheim	CON	01.04.96	3.600,00 €
Pirmin	Zübli	12.05.74	**Seeweg 28a** 35356, Hassberg	ZE8	01.07.06	2.450,00 €
Astrid	Zübli	29.06.76	**Seeweg 28a** 35356, Hassberg	CON	15.03.96	2.500,00 €
Karl-Heinz	Brenner	04.12.60	Waldweg 7b 22456 Erlenhain	GF	02.01.90	5.500,00 €

Abteilung CON:

Vorname	Nachname	Geburtsdatum	Anschrift	Abt.	Eintrittsdatum
Hanns	Otto	22.07.53	Mühlengasse 12 45789, Mülheim	CON	01.04.96
Astrid	Zübli	29.06.76	**Seeweg 280** 35356, Hassberg	CON	15.03.96

Wo wohnen nun die Züblis: Seeweg 28a oder Seeweg 280?

Durch die Vermeidung von Redundanzen wird die Gefahr inkonsistenter Daten minimiert. Falls die Adresse von Astrid Zübli nur ein einziges Mal in dem Datenbestand vorkommt, ist es ausgeschlossen, dass Unstimmigkeiten auftreten.

Daher ist es wichtig, Redundanzen zu vermeiden.

3.5 Parallele Zugriffe auf den Datenbestand (Transaktionen)

Zur Vermeidung von Redundanzen und Inkonsistenzen ist es sinnvoll, die Daten nur einmal, an einem Ort, zu speichern. Es werden also alle Mitarbeiter in einer Tabelle gespeichert. Daraus folg aber ein weiteres Problem:

Alle Abteilungen werden nun den gleichen Datenbestand nutzen. Es können dadurch parallele Zugriffe auf die Daten entstehen.

Diese Zugriffe müssen durch das Verwaltungsprogramm der Datenbank geregelt werden. Es werden z.B. Datensätze, die von einem Benutzer bearbeitet werden, für andere Benutzer gesperrt. Dieses Sperren kann aber auch Probleme aufwerfen. Stellen wir uns hierzu den Ablauf einer Überweisung von einem Konto bei der Bank A zu einem Konto bei der Bank B vor:

Bank A

Kontoinhaber	Kontonummer	Kontostand
Franz Meiser	13254647598	1.200,00 €

Bank B

Kontoinhaber	Kontonummer	Kontostand
Hubert Specht	40729347756	450,00 €

Franz Meiser möchte Hubert Specht 500,- € überweisen und füllt dazu einen Überweisungsauftrag aus, den er bei seiner Bank einreicht.

Der Überweisungsauftrag wird nun in drei Schritten ausgeführt:

- Das Geld wird bei Bank A von Meisers Konto abgebucht

Bank A

Kontoinhaber	Kontonummer	Kontostand
Franz Meiser	13254647598	700,00 €

- Der Überweisungsauftrag wird der Bank B übermittelt
- Das Geld wird bei Bank B Spechts Konto gutgeschrieben

Bank B

Kontoinhaber	Kontonummer	Kontostand
Hubert Specht	40729347756	950,00 €

Was geschieht nun, wenn Hubert Specht genau in dem Moment, in dem das Geld seinem Konto gutgeschrieben wird, am Geldautomaten eine Abhebung tätigt?

Durch die Abhebung ist sein Konto für weitere Zugriffe gesperrt und die Überweisung kann seinem Konto nicht gutgeschrieben werden!

Was ist in diesem Fall zu tun? Denkbar sind zwei Lösungsmöglichkeiten:

- Die Gutschrift auf Spechts Konto wird beendet nachdem er mit seiner Abhebung fertig ist.
- Die Überweisung wird rückgängig gemacht, also das Geld wieder dem Konto von Meiser gutgeschrieben.

Das beschriebene Problem wird durch das Konzept der ***Transaktion*** gelöst. Eine Transaktion beschreibt dabei in der Informatik, eine feste Folge von Operationen, die als eine logische Einheit betrachtet werden. Der Zugriff auf Daten wird durch das Transaktionssystem gesteuert.

Das Transaktionssystem muss dabei folgende Eigenschaften garantieren:

- **Atomarität:** Eine Transaktion ist unteilbar und wird ganz oder gar nicht ausgeführt. Wird eine Transaktion abgebrochen, muss das System in den Ursprungszustand zurückversetzt werden. (Meiser wird das Geld wieder gutgeschrieben)
- **Konsistenz:** Nach einer Transaktion muss der Datenbestand wieder Konsistent sein, also keine Unstimmigkeiten aufweisen.
- **Isolation:** Bei gleichzeitiger Ausführung von Transaktionen dürfen sich diese nicht beeinflussen. (Das Abheben darf das Gutschreiben nicht verhindern).
- **Dauerhaftigkeit**: Eine erfolgreiche Transaktion muss den Datenbestand dauerhaft verändern.

Damit Transaktionen rückgängig gemacht werden können, werden die Einzelschritte in einer Log-Datei gespeichert. Dies ist besonders bei Verteilten Transaktionen (Transaktionen die auf mehreren Datenbanken ausgeführt werden) wichtig. In unserem Beispiel muss sich Bank A das Abbuchen von Meisers Konto „merken“ bis Bank B bestätigt, dass das Geld Specht gutgeschrieben wurde (dies geschieht durch den speziellen SQL-Befehl COMMIT) oder bis Bank B meldet, dass die Transaktion fehlschlug (z.B. durch ein fehlerhafte Kontonummer). Dann wird durch den SQL-Befehl ROLLBACK die Transaktion rückgängig gemacht, Meiser also das Geld wieder gutgeschrieben.

3.6 Schutz vor Datenverlust

Die Daten einer Datenbank können auf vielfältige Weise verloren gehen. Ursachen können sein:

- Ungewollte Manipulationen, z. B. versehentliches Löschen
- Ausfall des Speichermediums bzw. Überlauf des Speichers
- Gezielte Angriffe auf den Datenbestand

Durch geeignete Mechanismen kann in einem Datenbanksystem das Risiko des Datenverlustes verringert werden. Zu diesem Zweck muss das Datenbanksystem insbesondere folgende Anforderungen erfüllen:

- Es muss eine Anmeldung an das System erfolgen (Identifikation und Authentisierung)
- Es müssen verschieden Berechtigungsstufen (Rollen) vorhanden sein
- Es muss ein vollständiges Transaktionssystem integriert sein
- Es müssen Mechanismen zur Datensicherung implementiert sein

3.7 Möglichkeit der Verteilung der Datenbank auf mehrere Speicherorte

Die Daten eines Datenbanksystems werden, bei kommerziell eingesetzten Programmen, nicht in einer, sondern in mehreren Dateien gespeichert. Dabei unterscheidet man zwischen der primären Datei und sekundären Dateien. In der primären Datei werden die Verweise auf die sekundären Dateien verwaltet. Reicht der Speicherplatz auf einem Medium nicht mehr aus, können sekundäre Dateien auf andere Speichermedien ausgelagert werden, um so die Datenbank zu vergrößern.

Die Verteilung der Daten auf unterschiedlichen Speichermedien wird von dem Datenbankadministrator festgelegt. Der Anwender muss sich nicht um den Ablageort der Daten kümmern.

Notizen

Kapitel 4 Das Entity-Relationship-Modell

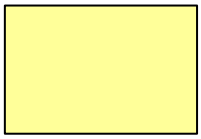
Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie wissen was das Entity-Relationship-Modell ist
- Sie kennen Entitäten, Attribute, Kardinalität, Primär- und Fremdschlüssel
- Sie kennen die graphische Darstellung und die Klammernotation

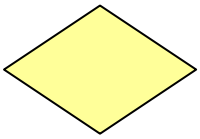
Das ERM gehört zu den semantischen Datenmodellen. Übersetzt heißt es Entitäten-Beziehungsmodell. Es zählt zu den wichtigsten Entwurfsmodellen.

Für den konzeptionellen Entwurf verwendet man das Entity-Relationship Modell.

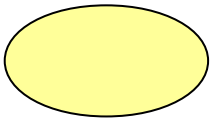
Die graphischen Elemente des ERM:



Entitätstyp



Beziehung



Attribut

Kurzer Planungsausblick: Nach dem konzeptionellen Entwurf erfolgt eine Transformation des konzeptionellen Schemas in ein zweites Datenmodell, das **Relationale Datenmodell**. Für diese Transformation gibt es festgelegte Transformationsregeln. Aus Entity-Relationship-Modellen können relativ einfach andere Datenmodelle abgeleitet werden.

4.1 Klammernotation

In den folgenden Kapiteln wird neben den Begriffen des ERM auch die textmäßige Formulierung der Elemente erklärt. Die textmäßige Erfassung von ERM nennt man **Klammernotation** oder auch Klammerschreibweise. Bei ihr wird der Entitätstyp vorweggestellt und die Attribute in Klammern aufgelistet. Mehrwertige Attribute werden in geschweiften Klammern dargestellt, zusammengesetzte Attribute in eckigen Klammer.

Beispiel: Mitarbeiter(Name, Vorname, {Telefon}, Adresse[Straße, Plz, Ort])

4.2 Entität (Ennametity)

Andere Bezeichnungen hierfür sind auch Objekt, Ausprägung, occurence oder Instanz. **Entitäten sind eindeutig identifizierbare Dinge, Lebewesen, Begriffe oder auch Ereignisse der realen oder der Vorstellungs- oder Geschäftswelt.**

Eine Entität kann z.B. sein

- eine bestimmte Person: Sabine Meier, München, Hauptstraße 4
- ein realer Gegenstand: das Buch „Datenbanksysteme“
- ein Objekt der Vorstellung: die Comic-Figur „Asterix“
- ein abstraktes Konzept: das hierarchische Datenmodell
- ein Ereignis: Tee trinken
- ein Objekt der Geschäftswelt: eine Bestellung

4.3 Attribut (Eigenschaft, Merkmal)

Die Entitäten einer Entitätsmenge besitzen Eigenschaften, die sie als Elemente dieser Entitätsmenge kennzeichnen.

Eigenschaften von Entitätsmengen nennt man **Attribute**.

Jedes Attribut einer Entität hat einen Wert. Die Menge aller Werte, die ein Attribut annehmen kann, nennt man **Wertebereich** (Domain, Value-Set).

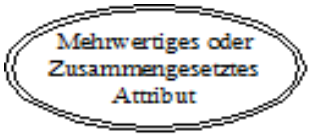
Entitätsmenge: die Mitarbeiter einer Firma

Namen der verwendeten Attribute: pers_nr, vname, nname, gebdat

	ATTRIBUT	pers_nr	vname	nname	gebdat
	WERTEBEREICH	TEXT (5)	TEXT (15)	TEXT (30)	DATUM
1. Entität	ATTRIBUTWERT	P100	Harry	Hirsch	1944-04-04
2. Entität	ATTRIBUTWERT	P200	Susi	Sorglos	1977-07-07

Man unterscheidet hierbei:

- **Atomare Attribute**: Diese beinhalten genau einen, nicht teilbaren Wert
  Beispiele: Geburtsdatum, Geschlecht, Schulnote
- **Mehrwertige Attribute**: Sie bestehen aus mehreren gleichartigen Werten
  Beispiele: Kontaktperson, da ein Kunde mehrere Kontaktpersonen haben kann. Hobby, da eine Person durchaus mehrere Hobbys haben.
- **Zusammengesetzte Attribute**: Diese sind in mehrere Attribute teilbar
  Beispiele: Adresse ist teilbar in Postleitzahl, Ort und Straße. Name ist teilbar in Vor- und Nachname. Zusammengesetzte und mehrwertige Attribute werden als Ellipse oder Kreis mit doppelter Umrandung dargestellt.



Im Gegensatz zu Entitätsmengen werden Attribute (nicht die Werte!) als zeitunabhängig angenommen.

4.4 Entitätsmenge (Entity-Set)

Eine Entitätsmenge (Objektmenge) ist die Zusammenfassung einzelner Entitäten, die ähnliche oder vergleichbare Eigenschaften haben.

Hierzu ein paar Beispiele:

- alle Bücher einer Bibliothek
- die Rechnungsverbuchungen einer Firma
- alle Einwohner von Köln

Entitätsmengen sind in der Regel von der **Zeit abhängig**: Die Einwohnerzahl von Köln oder der Buchbestand einer Bibliothek ändern sich im Laufe der Zeit.

4.5 Entitätstyp (Entity Typ)

Andere Bezeichnungen: **Objekttyp**, **Objektart**.

Ein Entitätstyp beschreibt einen Oberbegriff für eine Menge (Sorte) von speziellen Entitäten mit gleichen Attributen.

Der Entitätstyp MITARBEITER könnte zum Beispiel durch die Attribute `pers_nr, vname, nname, gebdat` definiert werden. Man schreibt dann den Entitätstyp in der Klammerschreibweise:

MITARBEITER (pers_nr, vname, nname, gebdat)

Ein spezieller Mitarbeiter ist eine Entität (Objekt, Ausprägung) des Entitätstyps MITARBEITER (`pers_nr, vname, nname, gebdat`).

Jede Entität eines Entitätstyps kann über die Attributwerte **eindeutig identifiziert** werden. Oft genügen aber schon die Werte nur einiger Attribute, um eine Entität eindeutig zu bestimmen. Solche Attribute nennt man Schlüsselattribute und ihre Zusammenfassung Schlüssel.

4.6 Schlüssel (key)

Der Schlüssel einer Entitätsmenge ist eine minimale Kombination von Attributen {A1, A2, ..} (Schlüsselattribute), deren Werte zusammengenommen eine Entität eindeutig identifizieren. Das Wort Minimal ist so gemeint, dass kein Attribut weggelassen werden kann, ohne die eindeutige Identifizierung zu verlieren. Es kann also Schlüssel mit einzelnen Attributen als auch mit mehreren Attributen geben.

PERSON (`PersNr, Name, Vorname, Adresse, Geburtsdatum, Beruf`)

- {PersNr, Name, Vorname, Adresse, Geburtsdatum, Beruf}

ist **kein** Schlüssel (die Kombination ist nicht minimal).

- {Name, Vorname, Adresse}

ist ein Schlüssel (Alternativschlüssel zu Personalnummer)

- {PersNr}

ist ein Schlüssel.

4.7 Primärschlüssel (primary key)

Ein Primärschlüssel ist ein ausgezeichneter (festgelegter) Schlüssel. Primärschlüssel werden in der Klammerschreibweise unterstrichen.

MITARBEITER (pers_nr, vname, nname, gebdat)

Die Werte eines Primärschlüssel-Attributes sind immer eindeutig und nie „leer“. Primärschlüsselfelder sollten sich nie ändern und aus möglichst künstlichen, nicht existierenden Werten bestehen. Ein Primärschlüsselfeld wird automatisch auch als Indexfeld definiert und kann als Fremdschlüssel (foreign key) in anderen Tabellen eingesetzt werden.

4.8 Sekundärschlüssel (secondary key)

Ein Sekundärschlüssel besteht aus einem beliebigen Attribut oder einer Attributkombination. Sekundärschlüsseln kennzeichnen Entitäten mit einer weiteren wichtigen Orientierung.

Zweck ist das Auffinden von Entitäten mit vorgegebenen Attributwerten (Klassifizierung). Oft werden Sekundärschlüssel auch als Indizes bezeichnet.

Der Begriff **Index** bedeutet, dass auf dieses Attribut oder auf diese Attribute besonders oft zugegriffen werden wird. Aus diesem Grund bieten DBMS immer eine Indizierung der Daten an. Indizierte Daten haben eine bessere Antwortzeit. Der Aufwand beim Schreiben und Ändern indizierter Daten ist aber höher.

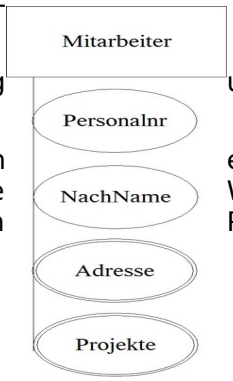
BUCH (InvNr, Titel, Autor, Verlag, ISBN)

Die Attributkombination {Titel, Autor} kann als Sekundärschlüssel dienen.

4.9 Grafische Darstellung des ERM nach Chen

Neben der Klammerschreibweise ist die grafische Erfassung des ERM wichtig. Das Entity-Relationship Modell wurde von **P. P. Chen** 1976 vorgeschlagen, um das Datenbank-Design zu erleichtern. Mit dem ERM wird die Struktur einer Datenbank beschrieben und dargestellt.

Ein **Entitätstyp** wird durch ein beschriftetes Rechteck dargestellt.



Die zugehörigen **Attribute** werden elliptisch oder kreisförmig umrahmt und mit dem Rechteck verbunden.

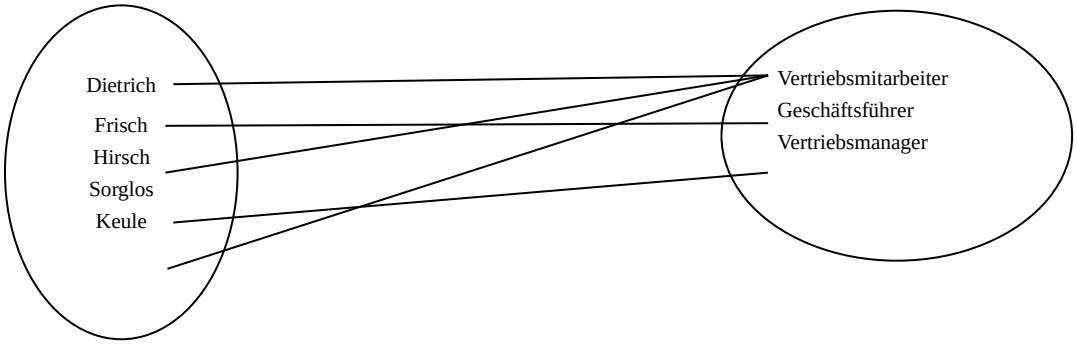
Mehrwertige Attribute oder **multiple Attribute** erhalten eine Doppelumrahmung. Ein Attribut, das für eine einzelne Entität mehrere Werte gleichzeitig haben kann (z.B. „projekt“: eine Person kann an mehreren Projekten arbeiten).

Beziehungen zwischen Entitätstypen werden als beschriftete Raute dargestellt (siehe nächstes Kapitel).

4.10 Beziehungen (Relationshiptypen) und ihre Darstellung

Wenn Entitäten in verschiedenen Entitätstypen in einer Beziehung zueinander stehen, spricht man von einer Beziehung (Relationship).

In der Mengenlehre würde man den Sachverhalt beispielsweise so darstellen:



Mitarbeiter bekleidet Position

Es werden also einzelne Entitäten (=Mengenelemente) aus verschiedenen Entitätsmengen einander zugeordnet. Einzelne Beziehungen sind hier als Verbindungslinien gekennzeichnet. Die Bedeutung der Grafik steht aber noch als Text darunter.

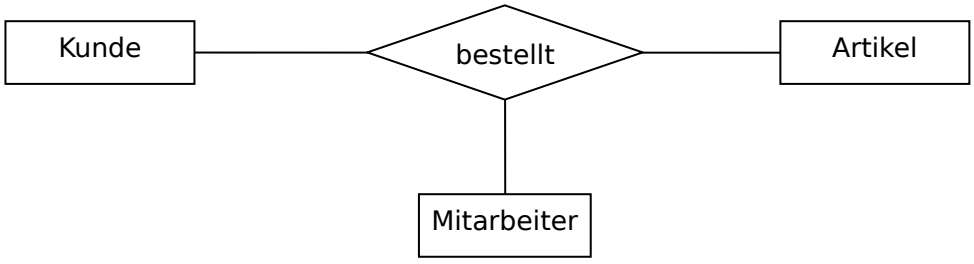
Wenn die einzelnen Beziehungen einer Regel folgen, stehen die Entitätsmengen selbst in Beziehung miteinander. Dann spricht man von einem **Beziehungstyp** oder **Relationshiptyp**:

In der Darstellung des ERM werden die Beschriftungen in die grafischen Elemente des ERM eingebaut:

Der Entitätstyp wird als beschriftete Box dargestellt (Mitarbeiter und Position).

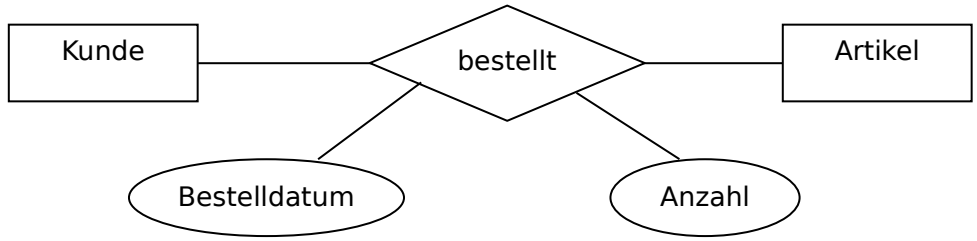
Der Beziehungstyp („bekleidet“) wird durch eine Raute dargestellt.

Diese Konfiguration aus zwei Entitätstypen und einem Beziehungstyp nennt man eine **zweistellige** oder **lineare** Beziehung. Es können aber auch mehrstellige Beziehungen vorkommen:



Man spricht hier auch von dem **Grad** der Beziehung. Die zweistellige Beziehung ist der praktisch häufigste Fall.

In einigen Fällen können auch **Beziehungstypen Attribute** haben:



Aus den drei letzten ERM-Beispielen kann man noch **nicht** sagen, welche Regeln die Beziehung der Entitäten zueinander beschreiben. Es fehlen noch Angaben über die **Komplexität** jeder der Beziehungen.

4.11 Komplexität einer Beziehung

Die Komplexität hat zwei Fragen zu klären:

- Die **Kardinalität**: Haben die einzelnen Entitäten keine, eine oder gar mehrere Beziehungen zu den Entitäten im anderen Entitätstyp?
- Die **Partizität** oder strukturelle Bedingung: Kann eine Beziehung auch ungenutzt bleiben? Wie oft darf eine Beziehung genutzt werden?

4.11.1 Kardinalität

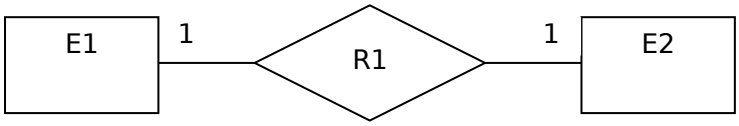
Als Beispiel beziehen wir uns auf unseren Sachverhalt „Mitarbeiter bekleiden Positionen“. Hier ist die Komplexität zunächst ungeklärt. Aus den folgenden Fragen ergeben sich folgende Kardinalitäten:

Frage	Kardinalität
Hat ein Mitarbeiter niemals eine Position?	1 zu 0
Hat ein Mitarbeiter eine Position?	1 zu 1
Hat ein Mitarbeiter mehrere Positionen?	1 zu n
Haben mehrere Mitarbeiter die gleiche Position	m zu 1
Hat ein Mitarbeiter mehrere Positionen und haben mehrere Mitarbeiter die gleiche Position?	m zu n
Gibt es nur Positionen ohne Mitarbeiter?	0 zu 1

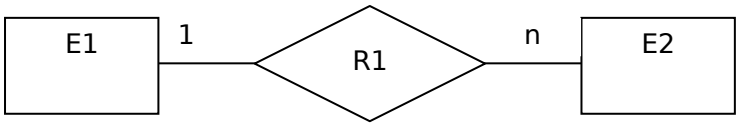
Die erste und die letzte Variante scheiden beim ERM aus, da hier keine Beziehungen mehr vorliegen. Es kann aber sein, dass zum Beispiel einige Positionen in einem Betrieb zurzeit gerade nicht besetzt sind. Um so einen Sachverhalt in einem relationalen Datenbanksystem abzulegen, müssen man auch die **Partizitäten** klären.

Darstellung der Kardinalität im ERM

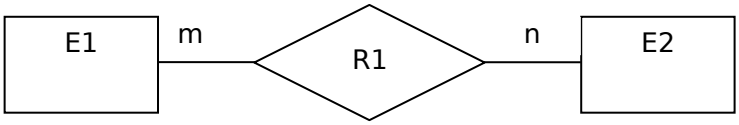
Fall **„1 zu 1“**: Jede Entität einer Entitätsmenge des Typs E1 kann zu einer bestimmten Zeit (Zeitraum) mit höchstens einer Entität der Menge des Typs E2 in Beziehung R1 stehen:



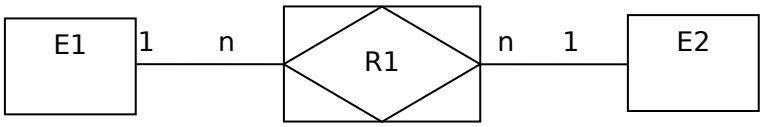
Fall **„1 zu N“**: Je eine Entität einer Entitätsmenge des Typs E1 kann zu einer bestimmten Zeit (Zeitraum) mit mehreren Entitäten der Menge des Typs E2 in Beziehung stehen:



Fall **„M: N“** jede Entität einer Entitätsmenge des Typs E1 kann zu einer bestimmten Zeit (Zeitraum) mit mehreren Entitäten **der Menge des Typs E2 in Beziehung stehen und jede Entität einer Entitätsmenge des Typs E2 kann zu einer bestimmten Zeit (Zeitraum) mit mehreren Entitäten der Menge des Typs E1 in Beziehung stehen:**



Diese Konstruktion muss aber immer aufgelöst werden in zwei 1 zu n Beziehungstypen.



Sie muss auch angewendet werden, wenn sich einzelne Beziehungen zeitlich verändern und dies für das DBS relevant sein soll. Der **Beziehungstyp** wird nun als **neue Entitätsmenge** definiert. Deswegen hat er nun auch zusätzlich einen Kasten um das Rautensymbol. In einer neuen Entitätsmenge kann man dann auch spezielle Attribute integrieren, welche zum Beispiel Zeitdaten darstellen.

Es existieren also drei Beziehungsarten. Diese kann man durch die Angabe ihres **Komplexitätsgrades** folgendermaßen zusammenfassen:

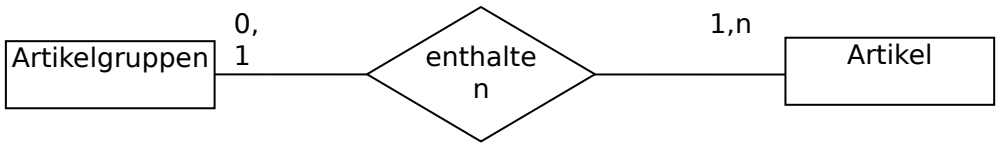
- eins zu eins 1:1
- eins zu viele 1:n
- viele zu viele m:n

4.11.2 Partizität (Strukturelle Bedingung)

Der Komplexitätsgrad eines Beziehungstyps gibt einen allgemeinen Überblick über die möglichen Beziehungen zwischen den Entitätstypen.

Will man bei den Beziehungen nicht nur 0, 1, n und m angeben, sondern deren genaue Ober- und Untergrenze spezifizieren, so bietet sich die **Min-Max Notation** an.

Durch Angabe der so genannten strukturellen Bedingung **(min, max)** kann man darstellen, ob eine bestimmte Entität einer Entitätsmenge (des Entitätstyps) in einem bestimmten Beziehungstyp enthalten sein kann: nicht (0), einmal (1) oder maximal n-mal (n) enthalten sein muss. Beispielsweise bedeutet eine strukturelle Bedingung (0, 1), dass eine Entität in Beziehung stehen kann, aber nur maximal einmal im Beziehungstyp enthalten sein darf. Beispiel:



- **Untergrenze = 0 ⛁ KANN-Beziehung**
  Eine Artikelgruppe kann null Artikel enthalten
- **Untergrenze = 1 ⛁ MUSS-Beziehung**
  Ein Artikel muss in einer Artikelgruppe enthalten sein

4.12 Beispiel für die Entwicklung eines ERM

Zur Verdeutlichung soll in diesem Kapitel die Entwicklung eines ERM an einem Beispiel beschrieben werden.

Folgende Aufgabenstellung ist gegeben:

Sie werden beauftragt eine Datenbank für den Spieß zu entwickeln, um seine Stubenbelegung zu verwalten. Sie haben folgende Informationen.

- Er möchte wissen welcher Soldat, also Name und Dienstgrad, auf welcher Stube seit wann liegt.
- Es gibt verschiedene Stuben in den Gebäuden, was die Anzahl der Betten betrifft, sowie solche mit und ohne Waschbecken.
- Damit er die Stube schnell findet, benötigt er die Angabe der Straße, in der das Gebäude liegt und das Baujahr des Gebäudes.

Aus der Aufgabenstellung kann man leicht die Entitätstypen entnehmen:

- SOLDATEN
- STUBEN
- GEBÄUDE

Hierbei handelt es sich jeweils um Entitäten, die wirklich vorhanden sind, also „unfassbare“ Objekte darstellen.

Die Attribute lassen sich ebenfalls direkt aus dem Text herauslesen. In der Klammerschreibweise sehen die Entitätstypen wie folgt aus:

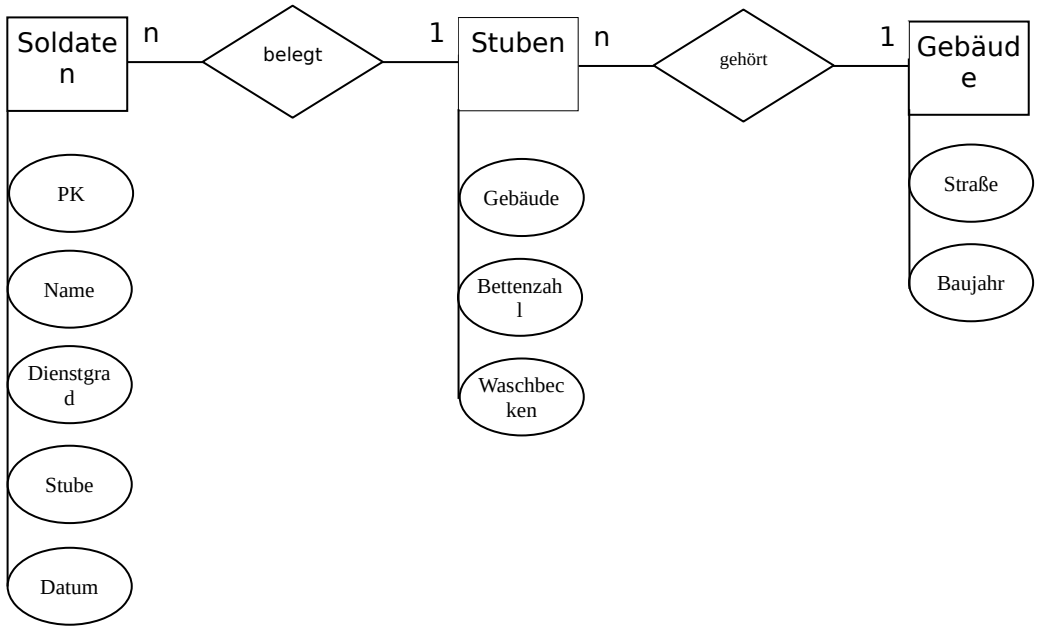
- SOLDATEN (PK, Name, Dienstgrad, Stube, Datum)
- STUBEN (Gebäude, Bettenzahl, Waschbecken)
- GEBÄUDE(Straße, Baujahr)

Da alle Soldaten eine PK haben, wurde diese in die Entitätsmenge übernommen. Diese PK ist eindeutig und dient der Identifizierung eines Soldaten. Es ist somit der Primärschlüssel eines Soldaten und daher unterstrichen. Über weitere Schlüssel brauchen wir uns an dieser Stelle noch keine Gedanken machen. Diese werden in den nächsten Entwicklungsschritten bis zum fertigen Datenbanksystem hinzukommen.

Nun werden die Beziehungen zwischen den Entitätstypen untersucht:

- Ein Soldat belegt eine Stube,
  eine Stube wird von mehreren Soldaten belegt
- Eine Stube gehört zu einem Gebäude,
  zu einem Gebäude gehören mehrere Stuben

Das Entity-Relationship-Diagramm sieht also wie folgt aus:



Damit ist das Entity-Relationship-Modell für die Aufgabenstellung entworfen. Dieses muss in ein relationales Modell überführt werden. Damit werden wir uns in den nächsten Kapiteln beschäftigen.

Notizen

Kapitel 5 Normalisierung

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie wissen, was eine Normalisierung ist
- Sie kennen die einzelnen Normalformen und wissen diese einzusetzen
- Sie sind sicher im Umgang mit Primär- und Fremdschlüsseln

Bei komplexen Aufgabenstellungen ist die Modellierung von Datenstrukturen nicht einfach. Leicht können Redundanzen entstehen oder Schwierigkeiten bei der späteren Datenpflege auftreten.

Um diesen Schwierigkeiten bei der Erstellung der Tabellen zu begegnen, wurden im Laufe der Zeit verschiedene Regeln aufgestellt, die man Normalisierungsregeln oder Normalformen (NF) nennt. Zurzeit gibt es neun Normalisierungsregeln, von diesen werden in der Praxis die ersten drei bis fünf Regeln verwendet. Zu 99,9% deckt die 3.Normalform die Bedürfnisse ab (Industriestandard).

Tabellen werden normalisiert, indem die Tabellen nach bestimmten Normalisierungsregeln aufgebaut beziehungsweise umgebaut werden.

Eine Tabelle ist zunächst in der n-ten Normalform (1.NF, 2.NF, 3.NF), wenn sie die n-te Normalisierungseigenschaft besitzt. Durch Normalisierung entsteht aus einem Entitiy-Relationshipmodell (ERM) ein relationales Datenmodell (RDM).

Die wichtigsten Ziele, die man mit der Normalisierung einer Tabelle erreichen will, sind:

- Vermeidung von Redundanzen
- einfacher Aufbau der Tabelle
- einfache Datenpflege
- Flexibilität beim Ausbau der Datenbank

Vorgehensweise:

Zuerst werden die Tabellen daraufhin untersucht, ob es berechnete Felder gibt. So lässt sich z.B. das Alter aus dem Geburtsdatum ermitteln. Oder der Umsatz, den man mit einem Kunden gemacht hat, lässt sich aus der Summe der einzelnen Bestellungen errechnen. Diese berechneten Felder werden aus den Tabellen entfernt.

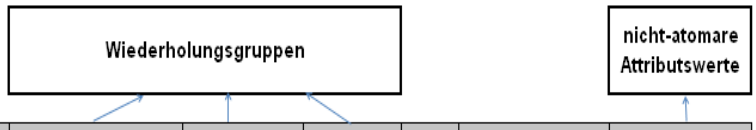
Tabellen dürfen keine berechneten Felder beinhalten!

Von der 0. NF wird eine Tabelle Stück für Stück in die 3. NF (Industriestandard) gebracht.

5.1 Die 0. Normalform

Eine Relation *Buch* liegt in der Nullform vor, das entsprechende Relationenschema sieht wie folgt aus:

BUCH(`ISBN, Titel, Autor, Autorfunktion, Geburtsjahr, Seiten, Verlagname, Verlagsort`)



ISBN	Titel	Autor	Autorfunktion	Geburtsjahr	Seiten	Verlagsname	Verlagsort
0345342968	Fahrenheit 451	Ray Bradbury	Allein-Autor	1920	179	Ballantine Books	New York 10036
3596144965	Die Mitte der Welt	Andreas Steinhöfel	Allein-Autor	1962	459	Fischer Tb.	60596 Frankfurt
3453861434	Nichts als die Wahrheit	Dieter Bohlen, Katja Kessler	Allein-Autor, Mitarbeiter	1954, 1969	335	Heyne	80336 München
3423071516	Der kleine Hobbit	John R. R. Tolkien, Klaus Ensikat	Allein-Autor, Illustrator	1892, 1937	381	DTV	80801 München

Die unnormalisierte Relation Buch in tabellarischer Darstellung

Anmerkung: Es wird davon ausgegangen, dass der Name des Autors, der im Attribut Autor festgehalten wird, eindeutig ist, dass es also keine zwei Autoren mit dem gleichen Namen gibt.

5.2 Die 1. Normalform

In der ersten Normalform werden alle Attribute, die nichtatomare Attributswerte enthalten, in mehrere Attribute aufgeteilt. Das nichtatomare Attribut *Verlagsort* könnte z.B. in die beiden atomaren Attribute *PLZ* und *Ort* aufgeteilt werden.

Wiederholungsgruppen werden aufgelöst, indem man für jeden Einzelwert des nichtatomaren Attributswerts ein eigenes Tupel (Datensatz) verwendet. Die atomaren Attributswerte des ursprünglichen Tupels werden hierbei kopiert.

Anmerkung: Durch die Auflösung von Wiederholungsgruppen in der ersten Normalform entstehen viele sich vertikal wiederholende Attributswerte, d.h., zunächst entsteht Redundanz. Diese Redundanz ist aber beabsichtigt und wird in der zweiten und dritten Normalform wieder aufgelöst.

Wichtig: Für eine Relation in der ersten Normalform muss zudem ein Primärschlüssel definiert werden.

Das veränderte Relationenschemata der Relation *Buch* sieht wie folgt aus:

BUCH(ISBN, Titel, Autor, Autorfunktion, Geburtsjahr, Seiten, Verlagname, PLZ, Ort)

ISBN	Titel	Autor	Autorfunktion	Geburtsjahr	Seiten	Verlagsname	Ort	PLZ
0345342968	Fahrenheit 451	Ray Bradbury	Allein-Autor	1920	179	Ballantine Books	New York	10036
3596144965	Die Mitte der Welt	Andreas Steinhöfel	Allein-Autor	1962	459	Fischer Tb.	Frankfurt	60596
3453861434	Nichts als die Wahrheit	Dieter Bohlen	Allein-Autor	1954	335	Heyne	München	80336
3453861434	Nichts als die Wahrheit	Katja Kessler	Mitarbeiter	1969	335	Heyne	München	80336
3423071516	Der kleine Hobbit	John R. R. Tolkien	Allein-Autor	1892	381	DTV	München	80801
3423071516	Der kleine Hobbit	Klaus Ensikat	Illustrator	1937	381	DTV	München	80801

Die Relation Buch *in der ersten Normalform*

Hinweis: Grundsätzlich könnte das Attribut *Autor* als nichtatomares Attribut aufgefasst werden und in die beiden atomaren Attribute *AutoVorname* und *AutorNachname* aufgeteilt werden. Da dies aber weder den Vorgang der Normalisierung eingehender verdeutlicht, wird auf eine weitere Aufteilung des Attributs *Autor* verzichtet.

Allgemeine Definition:

Eine Relation liegt in der ersten Normalform vor, wenn

- alle Attributwerte der Attribute der Relation atomar sind,
- innerhalb der Relation keine Wiederholungsgruppen vorliegen und
- die Relation durch einen eindeutigen Primärschlüssel bestimmt wird.

Eine Relation die in der ersten Normalform vorliegt, wird kurz als 1NF-Relation bezeichnet.

5.3 Die 2. Normalform

Die zweite Normalform baut auf der ersten Normalform auf. Eine Relation die in der zweiten Normalform vorliegt, wird üblicherweise auch als *2NF-Relation* bezeichnet.

Eine Relation liegt in der zweiten Normalform vor, wenn die Relation bereits in der ersten Normalform vorliegt und jedes Nichtschlüsselattribut der Relation vom Primärschlüssel der Relation voll funktional abhängig ist. Mit anderen Worten: Es darf keine funktionale Abhängigkeit eines Nichtschlüsselattributs von einer echten Teilmenge der Menge der Schlüsselattribute geben.

Hinweis: Wenn der Primärschlüssel einer Relation sich aus nur einem einzigen Attribut zusammensetzt, so liegt diese Relation notwendigerweise in der zweiten Normalform vor, wenn sie bereits in der ersten Normalform vorliegt. Sie müssen die zweite Normalform also nur beachten, wenn Sie für die Relation als Primärschlüssel einen zusammengesetzten Primärschlüssel definiert haben und die Relation außerdem mindestens ein Nichtschlüsselattribut enthält.

Um eine 1NF-Relation in eine 2NF-Relation zu überführen, müssen alle Attribute der 1NF-Relation, die vom Primärschlüssel nicht voll funktional abhängig sind, in eine oder mehrere neue Relation(en) ausgegliedert werden. Die Anzahl der neuen Relationen richtet sich nach der Anzahl der unterschiedlichen Teilmengen des Primärschlüssels, von denen die diversen Nichtschlüsselattribute abhängig sind. Die Teilmenge des Primärschlüssels, von dem die jeweils in eine Relation ausgelagerten Attribute voll funktional abhängig sind, wird Primärschlüssel der neuen Relation.

Das veränderte Relationenschemata sieht wie folgt aus:

- BUCH(ISBN, AutorID, Autorfunktion)
- BUCHDETAILS(ISBN, Titel, Seiten, Verlagsname, Ort, PLZ)
- AUTORDETAILS(AutorID, Autor, Geburtsjahr)

Bei der Relation AutorDetails wurde ein Surrogatschlüssel eingeführt, da das Attribut Autor sowohl den Vor- und Nachmane eines Autors enthält und im Vergleich zu einem Surrogatschlüssel relativ speicherintensiv ist.

Buch

ISBN	AutorID	Autorfunktion
0345342968	1	Allein-Autor
3596144965	2	Allein-Autor
3453861434	3	Allein-Autor
3453861434	4	Mitarbeiter
3423071516	5	Allein-Autor
3423071516	6	Illustrator

Die Relation Buch *in der zweiten Normalform*

BuchDetails

ISBN	Titel	Seiten	Verlagsname	Ort	PLZ
0345342968	Fahrenheit 451	179	Ballantine Books	New York	10036
3596144965	Die Mitte der Welt	459	Fischer Tb.	Frankfurt	60596
3453861433	Nichts als die Wahrheit	335	Heyne	München	80336
3423071516	Der kleine Hobbit	381	DTV	München	80801

Die Relation BuchDetails *in der zweiten Normalform*

AutorDetails

AutorID	Autor	Geburtsjahr
1	Ray Bradbury	1920
2	Andreas Steinhöfel	1962
3	Dieter Bohlen	1954
4	Katja Kessler	1969
5	John R. R. Tolkien	1892
6	Klaus Ensikat	1937

Die Relation AutorDetails *in der zweiten Normalform*

5.4 Die 3. Normalform

Eine Relation ist in der dritten Normalform, wenn sie bereits in der zweiten Normalform vorliegt und kein Nichtschlüsselattribut vom Primärschlüssel transitiv abhängig ist. Alle Nichtschlüsselattribute müssen also vom Primärschlüssel voll funktional abhängig sein (dies ist Bedingung für eine 2NF-Relation) und dürfen nicht von anderen Nichtschlüsselattributen funktional abhängig sein, da sonst eine transitive Abhängigkeit der betreffenden Nichtschlüsselattribute vom Primärschlüssel vorliegt. Als Ausnahme ist zu berücksichtigen, dass Nichtschlüsselattribute weiterhin von Schlüsselkandidaten abhängig sein dürfen.

Eine Relation die in der dritten Relation vorliegt, wird als 3NF-Relation bezeichnet.

Hinweis: Die dritte Normalform kommt nur dann zur Anwendung, wenn die betreffende Relation neben dem einfachen oder zusammengesetzten Primärschlüssel mindestens zwei Nichtschlüsselattribute enthält, die nicht Bestandteil eines Schlüsselkandidaten sind.

Um eine 2NF-Relation in eine 3NF-Relation zu überführen, werden die transitiv vom Primärschlüssel abhängigen Attribute der 2NF-Relation in eine oder mehrere neue Relation(en) ausgegliedert. Eine neue Relation erhält als Primärschlüssel die Menge der Nichtschlüsselattribute.

Das veränderte Relationenschemata sieht wie folgt aus:

- Buch(ISBN, AutorID, Autorfunktion)
- BuchDetails(ISBN, Titel, Seiten, VerlagID)
- AutorDetails(AutorID, Autor, Geburtsjahr)
- Verlag(VerlagID, Verlagsname, Ort, PLZ)

Buch

ISBN	AutorID	Autorfunktion
0345342968	1	Allein-Autor
3596144965	2	Allein-Autor
3453861434	3	Allein-Autor
3453861434	4	Mitarbeiter
3423071516	5	Allein-Autor
3423071516	6	Illustrator

Die Relation Buch *in der dritten Normalform*

BuchDetails

ISBN	Titel	Seiten	VerlagID
0345342968	Fahrenheit 451	179	1
3596144965	Die Mitte der Welt	459	2
3453861434	Nichts als die Wahrheit	335	3
3423071516	Der kleine Hobbit	381	4

Die Relation BuchDetails *in der dritten Normalform*

AutorDetails

AutorID	Autor	Geburtsjahr
1	Ray Bradbury	1920
2	Andreas Steinhöfel	1962
3	Dieter Bohlen	1954
4	Katja Kessler	1969
5	John R. R. Tolkien	1892
6	Klaus Ensikat	1937

Die Relation AutorDetails *in der dritten Normalform*

Verlag

VerlagID	Verlagsname	Ort	PLZ
1	Ballantine Books	New York	10036
2	Fischer Tb.	Frankfurt	60596
3	Heyne	München	80336
4	DTV	München	80801

Die Relation Verlag *in der dritten Normalform*

Jetzt werden die Tabellen auf Anomalien und Redundanzen hin überprüft.

In seltenen Fällen können Relationen auftreten, die sich in der dritten Normalform befinden und dennoch Redundanzen aufweisen. Dies hat dazu geführt, dass weitere Normalformen definiert worden sind.

Bei einer Neuentwicklung kann man die Normalisierung auch etwas rationeller durchführen. Eine Normalisierung findet dann folgendermaßen statt:

- Aus allen **multiplen Attributen** im Universalentitätstyp, die im ERM als eigene Entitätstypen auftreten, werden eigene Relationen.
- Alle **eigenen Relationen** erhalten einen eigenen Primärschlüssel. Wenn dieser nicht schon in der Aufgabenstellung bekannt war, muss er spätestens jetzt als neues Schlüsselattribut oder auch als zusammengesetzter Primärschlüssel hinzugefügt werden.

- Bei **1:1 Relationships** bekommt die Relation mit weniger Datensätzen einen Fremdschlüssel zur anderen 1-Relation.

- Bei **1:n Relationships** bekommt die n-Relation einen Fremdschlüssel , welche zur 1-Seite referenziert.

- Bei **n:m Relationships** entsteht die 3. Relation aus zwei Fremdschlüsseln, welche auf die beiden ursprünglich bekannten Relationen referenzieren. 3-stellige Relationships erhalten eine neu Relation mit 3 Fremdschlüsseln; 4-stellige Relationships erhalten eine neue Relation mit 4 Fremdschlüsseln... usw.

Notizen

Kapitel 6 Relationenoperationen

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie wissen was Relationenoperationen sind
- Sie kennen den Unterschied zwischen Projektion und Selektion
- Sie kennen Joins und den Einsatz von Vergleichsoperationen

Die Datenmanipulation im RDM beruht auf der Relationenalgebra. Die verschiedenen Operationen sind heute in SQL implementiert. E. F. Codd schreibt zwar nicht speziell SQL als Datenbanksprache vor, aber er verlangt irgendeine einfache Sprache, welche die hier besprochenen Operationen umsetzen kann.

Ergebnisse von Relationenoperationen sind wieder Relationen, die ebenfalls als Tabellen dargestellt werden können. So ist es möglich, an die Quellrelationen beliebig Umwandlungen zu schalten, die als Views oder Sichten gespeichert werden und ihrerseits wie eine „Virtuelle Tabelle“ behandelt werden können. Beispielsweise kann man in MS ACCESS Abfragen definieren, welche auf Tabellen beruhen und deren Daten aufbereiten. Diese Abfragen kann man aber dann wieder als Quelle von weiteren Abfragen verwenden. Auch die Berichtsobjekte, und Formularobjekte akzeptieren als Datenherkunft Abfragen ebenso wie Tabellen.

Schreibt man Daten in eine Abfrage, so ist es grundsätzlich möglich, dass die Daten in die dahinter geschalteten Tabellen zurück geschrieben werden. Diese Eigenschaft nennt man dann Dynaset. Ausnahme sind hier u.a. jedoch Abfragen, die Daten zusammenfassen, also **Aggregieren**.

Im Folgenden werden die wichtigsten Operationen mit Relationen beschrieben.

6.1 Projektion

Spaltenauswahl, Streichen von Spalten. Die Projektion ist die Auswahl bestimmter Attribute (Spalten) aus einer Relation.

Beispiel: ***Von welchen Verlagen stammen die Bücher der Bibliothek?***

Relationentyp: Buch (ISBN, Titel, Autor, Verlag, Erscheinungsjahr)

Projektion: Buch [Verlag]

6.2 Selektion

Zeilenauswahl (Entitätenauswahl). Die Selektion sortiert aus einer Relation alle Tupel (Zeilen) aus, die eine gegebene Bedingung (logischer Ausdruck) erfüllen.

Beispiel*:* ***Wie lauten die Artikel, welche zur Kategorie 1 (Systemkomponenten) gehören?***

Relationentyp: Artikel (`ID, Bezeichnung, Preis, Kategorie`)

Selektion: Artikel hat Kategorie=1[Systemkomponenten]

Neben diesen zwei sehr häufig verwendeten Grundoperationen gibt es drei weitere fundamentale Operationen, die unmittelbar mit der Verknüpfung von Mengen verbunden sind:

6.3 Vereinigung, Durchschnitt und Differenz von Relationen (Tabellen).

Diese Operationen sind nur auf Relationen anwendbar, die vom selben Relationentyp sind. Das bedeutet: die Relationen haben dieselben Attribute in der gleichen Reihenfolge.

Im Folgenden seien die Relationen R1 und R2 vom selben Relationentyp.

- **Vereinigung** (Union):

Die VEREINIGUNG der beiden Relationen R1 und R2 ist die Menge aller Tupel, die in R1 **oder** in R2 enthalten sind: **R1 ● R2**

- **Schnittmenge** (Intersection):

Der DURCHSCHNITT der beiden Relationen R1 und R2 ist die Menge aller Tupel, die in R1 **und** R2 enthalten sind: **R1 🖫 R2**

- **Differenz** (Minus):

Die DIFFERENZ ist die Menge aller Tupel, die in R1 **aber nicht in** R2 enthalten sind: **R1 ⁝≡ R2**

6.4 Joins von Relationen

Vorüberlegung: Abbildung von Beziehungstypen nach Grad (1:1, 1:n, m:n) unter Beachtung der transformations Regeln.

Die wichtigste Operation zur Verknüpfung von Relationen ist der Verbund (JOIN).

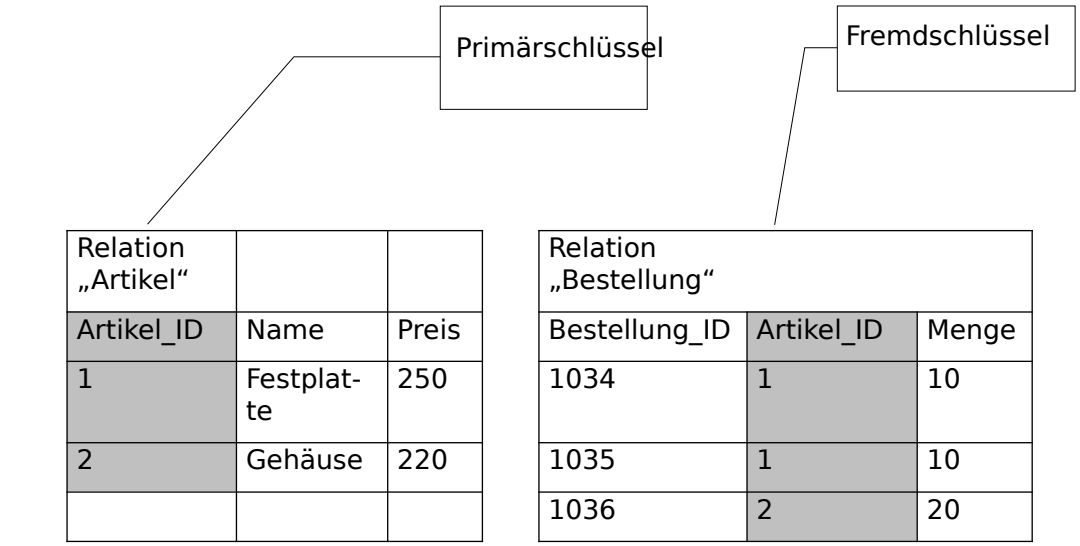
Der **JOIN** (VERBUND) ist ein Zusammenfügen von Tabellen

Während man zur Minimierung von Redundanz die Daten in unterschiedliche kleine Relationen aufteilt (Normalisierung), müssen später zum Betrieb in den Applikationen die Daten wieder zur ursprünglichen Struktur zusammengesetzt werden (denormalisiert werden). Zu diesem Zweck hat man das Verknüpfen/Verbinden von Relationen entwickelt.

Zum Verknüpfen von Tabellen braucht man ein Attribut **aus jeder Relation, welches mindestens vom gleichen Datentyp ist, und welches in jeder Relation übereinstimmende oder wenigstens vergleichbare Inhalte hat. Das folgende Beispiel zeigt die Fähigkeiten eines Joins, bei dem die Schlüssel auf Übereinstimmung geprüft werden: Oben stehen zwei Quellrelationen, unten eine.**

Beispiel für eine Sicht mit einem Join:

Zwei Relationen, die ein gemeinsames Feld besitzen:



Relation „Artikel“		
Artikel_ID	Name	Preis
1	Festplat-te	250
2	Gehäuse	220

Relation „Bestellung“		
Bestellung_ID	Artikel_ID	Menge
1034	1	10
1035	1	10
1036	2	20

Verknüpfung der Relation „Artikel“ mit der Relation „Bestellung“:

Artikel_ID	Bestellung. Artikel_ID	Bestellung. Menge	Artikel. Name	Postenpreis (=Artikel.Preis * Bestellung.Menge)
1	1	10	Festplat-te	2500
1	1	10	Festplat-te	2500
2	2	20	Gehäuse	4400

automatisch übernommene Daten aus Relation „Artikel“

Berechnung von Daten aus beiden Relationen

6.5 Vergleichsoperationen

Über einen Vergleichsoperator findet dann die Verknüpfung statt. Als **Vergleichsoperatoren** können folgende Möglichkeiten dienen:

=	Entities (Datensätze aus verschiedenen Relationen, aber mit gleichen Werten im Join werden kombiniert.
>, <	Die Joins werden über Ungleichungen gebildet
between... and...	Die Joins werden über ein Intervall gebildet

Zum Verknüpfen von Tabellen verwendet man in der Praxis einen Primärschlüssel **und den dazu passenden Fremdschlüssel aus der 2. Tabelle. Dies wird aber beim Thema referentielle Integrität noch einmal angesprochen. Der Natural Join verzichtet aber auf Schlüssel und hat eine wichtige Bedeutung in der Rasterfahndung innerhalb verschiedener Datenbanken. Dort ist immer nur ein Empirisches Ergebnis zu erwarten. Die verschiedenen Arten von Joins werden aber im später in den SQL-Abschnitten genauer behandelt. Hier soll nur das Verknüpfen selbst vorgestellt werden.**

Notizen

Kapitel 7 Integritätsregeln

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie kennen die verschiedenen Integritätsregeln
- Sie kennen den Sinn der referentiellen Integrität

Im Relationenmodell verzichtet man einerseits auf logisch Verknüpfung der Daten im Anwendungsprogramm (logische Datenunabhängigkeit). Die fachlich und datenbankmäßig korrekte Verarbeitung der Daten wird stattdessen über sogenannte Integritätsregeln realisiert.

Unter Integrität oder Konsistenz versteht man die Vollständigkeit und Stimmigkeit der zueinander gehörigen Daten. Im internationalen Sprachgebrauch verwendet man hier auch den Begriff Constraints. Man unterscheidet dabei in:

- **Objektintegrität**. (E. F. Codd sagte es treffend: „One Fact at One Place“)
- **Semantische Integrität**: Sind die Daten inhaltlich korrekt?
- **Referentielle Integrität**: Haben alle Daten eines Fremdschlüssel ausschließlich gültige Daten?
- **Benutzerdefinierte Integritätsregeln**: z.B. „beim Überziehen des Dispokredit erhöht sich der Zinssatz auf x%.“ oder: „Bei Überschreiten des Obligo erfolgt Liefersperre”.

7.1 Objektintegrität

Die Objektintegrität besagt, dass jedes Objekt in einer Relation nur **einmal** vorkommen darf und **eindeutig identifizierbar** sein muss. Dies wird grundsätzlich durch das Konzept des Primärschlüssels sichergestellt. Darüber hinaus dürfen für keinen Teil des Primärschlüssels Nullwerte zugelassen werden, da sie der Forderung nach Identifizierbarkeit widersprechen würden, denn Nullwerte entsprechen einem unbestimmten Wert.

Die meisten Datenbanksysteme kennen in diesem Zusammenhang eine eindeutige, systemseitig erzeugte Objektkennzeichnung wie zum Beispiel die ROWID, die unabhängig von der konkreten Modellierung der Daten eine eindeutige Identifikation eines Objekts darstellt.

Zur Objektintegrität gehören auch grundlegende Domänendefinitionen, wie zum Beispiel die Einschränkung des Wertebereichs, der durch den Datentyp vorgegeben wird.

7.2 Semantische Integrität

Unter der semantischen Integrität **versteht man die logische Widerspruchsfreiheit** und **exakte Abbildung** der Realität (keine falschen Einträge).

Durch fehlerhaft eingegebene Mengenoperationen kann jedoch eine Verletzung der semantischen Integrität auftreten, die eine Rekonstruktion des ursprünglichen Informationsgehalts nicht zulässt. Da die Einhaltung der semantischen Integrität nur schwer geprüft werden kann, unterstützen Datenbanksysteme die Einhaltung der semantischen Integrität im Allgemeinen nicht.

7.3 Referentielle Integrität

Referentielle Integrität besagt, dass jeder Fremdschlüsselwert in einer anderen Relation als Primärschlüsselwert definiert sein muss oder den NULL-Wert annehmen darf.

Im ersten Fall wird damit die Beziehung zu einem konkreten Objekt hergestellt, welches durch den Primärschlüsselwert identifiziert wird. Dies stellt eine Form der Existenzabhängigkeit dar. Sie wird durch Geschäftsregeln ausgedrückt wie "kein Auftrag ohne Kundennummer", "keine Abteilung ohne Abteilungsleiter" oder "jeder Mitarbeiter gehört zu einer Abteilung".

Unter dem Begriff der referentiellen Integrität versteht man Regeln, die dafür sorgen sollen, dass beim Löschen und Einfügen von Datensätzen die festgelegten Beziehungen zwischen den Tabellen beibehalten werden. Wenn man die referentielle Integrität aktiviert, verhindert ein DBMS so genannte Anomalien, also **Inkonsistenzen**.

Darunter versteht man folgendes:

- Das Hinzufügen von Datensätzen in einer Detailtabelle, für die kein Primärdatensatz vorhanden ist, heißt **Hinzufügeanomalie**.
- Änderungen von Werten in einer Mastertabelle, die zu nicht mehr zuzuordnenden Datensätzen in einer Detailtabelle führen würden, (**Änderungsanomalie**).
- Das Löschen von Datensätzen aus einer Mastertabelle, wenn noch übereinstimmend verknüpfte Datensätze vorhanden sind. (**Löschanomalie**).

Konsequenz: In der Detailtabelle können dann nur Datensätze eingegeben werden, deren Fremdschlüsselfeld einen Wert hat, der auch in der Mastertabelle im Primärschlüsselfeld vorkommt. Umgekehrt kann man einen Datensatz aus der Mastertabelle nicht löschen, ohne den oder die damit verbundenen Datensätze aus der Detailtabelle ebenfalls zu löschen. Um dies zu erreichen, gibt es zwei Möglichkeiten:

Entweder man sorgt manuell für die Durchsetzung der oben angegebenen Vorgaben, oder man aktiviert einen Automatismus für die **Aktualisierungs- und Löschweitergabe**, indem man sogenannte Einfüge-, Änderungs- und Löschregeln definiert. Falls diese Optionen aktiv sind, erlaubt ein DBMS zwar Änderungen und Löschungen, ändert und löscht aber zugleich auch die entsprechenden Detaildatensätze, um sicherzustellen, dass die Regeln der referentiellen Integrität **gewahrt bleiben.**

Beispiel: Falls eine Zeile in einer Relation gelöscht wird, werden auch alle über Fremdschlüssel verknüpften Zeile gelöscht. Diesen Vorgang nennt man auch kaskadierendes Löschen oder Löschweitergabe.

Sinn der referentiellen Integrität: In der Detailtabelle sollen niemals verwaiste Datensätze existieren, das heißt keine Datensätze, für die keine Entsprechungen im Primärschlüsselfeld vorliegen.

Notizen

Kapitel 8 Datenbankobjekte

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie kennen die einzelnen Datenbankobjekte
- Sie wissen was das Data Dictionary ist und wofür es verwendet wird

Im Folgenden werden die einzelnen Objekte, die in einem relationalen Datenbanksystem vorhanden sind vorgestellt.

8.1 Tabellen

Relationen werden innerhalb von relationalen Datenbanken als Tabellen abgebildet, wobei die grundlegenden Begriffe einer Relation alle übertragen werden können.

8.2 Constraints

Jede Spalte einer Tabelle hat einen Datentyp und kann zusätzlich mit Integritätsregeln, oder Constraints, definiert werden. Solche Constraints können zum Beispiel festlegen, dass eine Spalte keine Nullwerte annehmen darf, einen Primärschlüssel darstellt oder nur die Buchstaben "A", "E", und "F" enthalten darf.

8.3 Trigger

Trigger können auf Tabellenebene definiert werden. Bei einem Trigger handelt es sich um ein gespeichertes Programm, das bei Datenänderungen in einer Tabelle aufgerufen wird. Die Datenänderung kann gewählt und der Inhalt des Triggers kann frei definiert werden.

Trigger sind an das Auftreten der Operationen Einfügen, Löschen oder Ändern geknüpft und somit ereignisgesteuert. Durch das Setzen eines bestimmten Status kann die Ausführung der gewünschten Operation auch abgewiesen werden, zum Beispiel Löschen. Deshalb können mit einem Trigger weitergehende Integritätsregeln implementiert werden.

Die Verwendung von Triggern ist allerdings nicht auf die Implementierung komplexer Integrationsregeln beschränkt. Man kann Trigger auch zur Etablierung umfangreicher Geschäftsregeln verwenden, die dann anwendungsunabhängig ausgeführt werden. Ein Beispiel hierfür sei das automatische Protokollieren von Änderungen an der Datenbank.

8.4 Views (Sichten)

Views **haben eine zentrale Funktion innerhalb relationaler Datenbanken, die zum Denormalisieren der relationalen Strukturen, also zum Wiederaufbereiten der Daten dienen. Sie ermöglichen in der Hauptsache folgende Dinge:**

- Umbenennen von Spalten
- Konvertieren von Spaltenwerten
- Umordnen, Auslassen und Hinzufügen von Spalten
- Verknüpfen von Relationen
- Speicherung von Selektionen bzw. anderen Operationen
- Einschränkung von Zugriffsrechten auf Tabellen

- Verringerung der Komplexität des Datenmodells gegenüber dem Benutzer oder Anwendungsentwickler

Views erfüllen somit sehr viele Aufgaben und sind somit Bestandteil fast jeder Datenbank

Eine wichtige Eigenschaft von Views ist, dass ihr Ergebnis (auch wenn es bei jedem Aufruf der View neu erzeugt wird) eine Relation darstellt, auf die grundsätzlich alle Operationen angewendet werden können, die auch auf Tabellen möglich sind. Bei der Verwendung von Views muss allerdings beachtet werden, dass für bestimmte Views Einschränkungen gelten.

8.5 Schemata

Da innerhalb einer Datenbank Informationen von unterschiedlichen Abteilungen, Anwendungssystemen etc. gespeichert werden, muss es natürlich auch eine Möglichkeit geben, den Zugang zu den Daten zu regeln. Jedes System von Relationen kann dabei in einem separaten Schema verwaltet werden.

In einigen DBMS ist das Schema assoziiert mit dem Benutzer: Jeder Benutzer hat ein eigenes Schema. Der Zugang zu einem gemeinsamen Schema muss dann für diesen User noch freigeschaltet werden.

Jedes Schema innerhalb der Datenbank hat einen eindeutigen Namen und kann durch ein Passwort geschützt werden. Innerhalb eines Schemas können Datenbankobjekte (zum Beispiel Tabellen oder Views) angelegt werden, auf die nur das Schema selbst Zugriff hat. Alle anderen Schemata wissen nicht einmal von der Existenz dieser Objekte.

Natürlich ist es möglich, anderen Schemata Rechte zu erteilen, die ein Selektieren, Löschen, Ergänzen oder Verändern von Daten innerhalb eines Schemas (meist Tabellenbezogen) ermöglichen.

8.6 Tablespaces und Datendateien

Mit der logischen Organisation von Daten in Schemata und Tabellen muss natürlich auch eine physikalische Speicherung korrespondieren.

Datenbanken benutzen deshalb grundsätzlich das Konzept der Tablespaces. Ein Tablespace kann als ein Container von DB-Objekten aufgefasst werden, der diese DB-Objekte logisch zusammenfasst. Eine solche Zusammenfassung bringt vor allem administrative Vorteile mit sich, da zum Beispiel einzelne Tablespaces gewartet, gesichert und generell administriert werden können.

Die eigentliche Speicherung der Daten erfolgt in Datendateien, die nichts anderes als physikalische Betriebssystemdateien entsprechender Größe sind. Ein Tablespace kann aus mehreren Datendateien bestehen, die normalerweise auch auf mehrere Dateisysteme verteilt sein können, so dass die Größe eines Tablespace (und damit der Datenbank) nicht durch Betriebssystemdatei- oder Festplattengröße beschränkt wird.

Moderne Datenbanksysteme unterstützen neben Datendateien auch die Nutzung eines unorganisierten, das heißt nicht mit einem Dateisystem versehen Datenträger, meist als RAW Device bezeichnet. Raw Devices haben das Merkmal, dass die Administration der Daten auf dem Gerät vollständig vom Datenbanksystem übernommen wird, ohne zusätzlich Verwaltungsfähigkeiten des Betriebssystems zu nutzen, wodurch Zeit eingespart werden kann.

8.7 Weitere Objekte

- **Indizes**: Um den suchenden oder sortierenden Zugriff auf eine in der Datenbank gespeicherte Relation zu optimieren, können Indizes verwendet werden. Durch einen Index werden die Daten einer oder mehrerer (im Index verknüpfter) Spalten so organisiert, dass sehr schnell einzelne Werte innerhalb des Indexes gefunden werden können.

- **Synonyme**: Relationen werden in einem anderen Schema unter einem anderen Namen präsentiert.

- **Datenbanklinks** bieten die Möglichkeit, komfortabel auf Objekte in einer anderen Datenbank zuzugreifen, ohne als Client direkt mit dieser Datenbank verbunden zu sein. Eine Verbindung über einen Datenbanklink erfolgt serverseitig. Aus diesem Grunde kann zum Beispiel ein Datenbanklink genutzt werden, um Tabellen die in unterschiedlichen Datenbanken gespeichert sind, zu verknüpfen und daraus einen View zu bilden.

8.8 Pakete, Prozeduren (Stored Procedures) und Funktionen

Alle modernen Datenbanken unterstützen die Speicherung von Programmen in der Datenbank. Solche Programme werden nicht nur zur Triggerdefinition benötigt, sondern auch zur Bildung eigener SQL-Funktionen und gespeicherter Prozeduren, die zum Beispiel Client-Anwendungen entlasten können.

Neben allein stehenden Prozeduren und Funktionen können meist auch Pakete gebildet werden, die eine Gruppierung von zusammengehörenden Programmen ermöglichen. Standardmäßig werden zudem vielfach umfangreiche Pakte ausgeliefert, um spezielle Funktionen bzw. Erweiterungen innerhalb der Datenbank zu liefern.

8.9 Data Dictionary

Das Data Dictionary, zu Deutsch „Datenlexikon“, oder auch Metadatenbank genannt, enthält alle Informationen über den Aufbau der Datenbank. Beispielsweise:

- Die Semantik der Daten (Bedeutung)
- Die Codierung der Daten (Formate, Typen)
- Die Beziehungen der Daten (Abhängigkeiten)
- Die Speicherung der Daten (Medien)
- Die Verwendung der Daten in Programmen
- Vorhandene Tabellen
- Views
- Benutzerrechte
- Benutzer

Ein Data Dictionary dient zunächst dazu, die **Ergebnisse einer Datenmodellierungsphase** abzulegen, so dass Applikationen über eine einheitliche aber flexible Abfragesprache die Strukturen wieder einlesen und speichern.

Es bildet die Quelle aller Verwaltungsinformationen. So finden sich z. B: Angaben darüber, welche Tabellen mit welchen Spalten von wem erzeugt wurden, wo sie abgelegt sind, welchen Inhalt sie besitzen, wie sie indiziert sind und mit welchen Zugriffsrechten sie versehen sind.

Ohne ein Data Dictionary zusammen mit seinen Views wäre die DB-Administration unmöglich. Deshalb wird es von relationalen DBMS systemseitig erzeugt. Weiterhin muss es einesteils gut zugänglich sein, auf der anderen Seite muss der Zugriff auf das Data Dictionary kontrolliert werden, um irreführende Manipulationen an den Unternehmensdaten zu verhindern.

Aus den oben angegebenen Gründen ist der direkte Zugriff auf die komplexen Informationen des Data Dictionaries nicht möglich. Stattdessen stehen eine Reihe gut dokumentierter Sichten auf die Daten des DD zur Verfügung, welche die Komplexität verbergen, eventuell kodierte Inhalte auflösen und unterschiedliche Datenquellen zu sinnvollen Einheiten verbinden.

Neben dem systemseitig erzeugten Basis Data-Dictionary, gibt es vielfach anwendungsbezogene Data Dictionaries, sogenannte **Repositories**. Ein Repository ist sozusagen ein unterneh-

mensübergreifendes Data Dictionary, welches die Datenbasis inclusive zugehöriger Schnittstelle (Definition, Speicherung, Manipulation und Kontrolle) für sämtliche Informationen über die Informationsressourcen des Unternehmens bildet, die auf den verschiedenen organisatorische

Notizen

Kapitel 9 Datenbanksprache SQL

Nach Ablauf dieser Lektion haben Sie folgende Ziele erreicht:

- Sie haben einen ersten Einblick in SQL erhalten
- Sie wissen um die einzelnen Hauptbereiche von SQL
- Sie kennen die DML-Befehle SELECT, INSERT, DELETE, UPDATE und können diese einsetzen
- Sie wissen um den Einsatz einer UNION-Abfrage

SQL ist eine Datenbankprogrammiersprache, deren Ursprünge unmittelbar mit dem Konzept der Relationalen Datenbanken von E. F. Codd zusammenhängen. Ein Vorläufer von SQL war „SEQUEL“ (simple english query Language). Die Sprache **SQL** (= **S**tructured **Q**uery **L**anguage“) wurde schließlich von Chamberlein ersonnen.

Bei SQL verfolgt man die Idee vom mengenorientierten Verarbeiten von Daten. Das führt dazu, dass es in SQL zunächst keine datensatzorientierten oder bewegungsorientierten Befehlssätze gibt.

Der Programmierer beschreibt in SQL nicht das Verfahren, wie er an sein Ergebnis kommt, sondern er beschreibt das Ergebnis selbst. Der Rechner, der SQL versteht, soll selbst das passende Verfahren finden. Unter diesem Aspekt spricht man bei SQL auch von einer 4GL-Sprache.

Die Entwicklung von SQL ist noch nicht abgeschlossen. Daher gibt es verschiedene SQL-Standards, welche in der ISO 9075 beziehungsweise beim ANSI jeweils veröffentlicht wurden:

- SQL-86 von 1987
- SQL-89 von 1989
- SQL-2 oder SQL-92 von 1992
- SQL-3 oder SQL-99 von 1999
- SQL:2003 von 2003
- SQL/XML:2006 von 2006
- SQL:2008 von 2008
- SQL:2011 von 2011

Die Softwarefirmen, welche Relationale Datenbankmanagementsysteme vertreiben, bemühen sich, den jeweiligen Standard möglichst vollständig einzuhalten. Das bedeutet negativ ausgedrückt: Je nach System gibt es immer noch Unterschiede bei diversen SQL-Befehlen. Aus diesem Grund unterscheidet man noch drei **Level** als Maß für die Vollständigkeit der SQL-Implementation: Entry Level, Intermediate Level und Full-SQL Level

Gebräuchliche Server haben zusätzliche, nicht genormte SQL**-Befehle, die sich mit den speziellen Softwareeigenschaften auseinandersetzen.**

Die SQL-Sprache setzt sich aus drei Hauptbereichen zusammen:

- **Datendefinitionssprache** (Data Definition Language, **DDL**), mit der alle Objekte in einer SQL-Datenbank definiert und verwaltet werden,
- **Datenbearbeitungssprache** (Data Manipulation Language, **DML**), mit der Daten in Objekten ausgewählt, aktualisiert und gelöscht oder in die Objekte eingefügt werden, die mithilfe von DDL definiert wurden und
- **Datenkontrollsprache** (**DCL**) mit der Berechtigungen und Transaktionssteuerungen definiert und verwaltet werden.

Notizen

Kapitel 10 Data Definition Language (DDL)

Die DDL SQL, die zum Verwalten von Objekten (wie z. B. Datenbanken, Tabellen und Sichten) verwendet wird, basiert auf DDL-Anweisungen von SQL-92 mit Erweiterungen. Für jede Objektklasse gibt es normalerweise spezifische CREATE-, ALTER- und DROP-Anweisungen, wie z. B. CREATE TABLE, ALTER TABLE **und DROP TABLE.**

In diesem Kapitel erfahren Sie Folgendes:

- Sie können Tabellen erstellen.
- Sie können die Datentypen beschreiben, die beim Spezifizieren einer Spaltendefinition verwendet werden.
- Sie können Tabellendefinitionen ändern.
- Sie können Tabellen löschen, umbenennen und leeren

10.1 Datentypen

Folgende Datentypen haben Sie in SQL zur Verfügung, wobei nur die ersten 4 in diesem Kurs eine Rolle spielen:

Generelle SQL Datentypen

Datentyp	Beschreibung
CHARACTER(n)	Zeichenkette mit fester Länge n
VARCHAR(n) oder CHARACTER VARYING(n)	Zeichenkette mit variabler Länge. Maximale Länge n
BINARY(n)	Binäre Daten mit fester Länge n
BOOLEAN	speichert TRUE oder FALSE Werte Binäre Daten mit fester Länge n
VARBINARY(n) oder BINARY VARYING(n)	Binäre Daten mit variabler Länge n
INTEGER(p)	Ganzzahlig mit Größe p
SMALLINT	Ganzzahlig mit Größe 5
INTEGER	Ganzzahlig mit Größe 10
BIGINT	Ganzzahlig mit Größe 19
DECIMAL(p,s)	Genauer numerischer Wert, Genauigkeit p, Nachkommastellen s. Beispiel: *decimal(5,2)* ist eine Zahl mit 3 Ziffern vor und 2 Ziffern nach dem Komma
NUMERIC(p,s)	Genauer numerischer Wert, Genauigkeit p (wie DECIMAL)
FLOAT(p)	Fließkommazahl geringer Genauigkeit, minimale Ge-

Datentyp	Beschreibung
	nauigkeit p
REAL	Fließkommazahl, minimale Genauigkeit 7
FLOAT	Fließkommazahl, minimale Genauigkeit 16
DOUBLE PRECISION	Fließkommazahl, minimale Genauigkeit 16
DATE	speichert Jahr, Monat, und Tag
TIME	speichert Stunde, Minute, und Sekunde
TIMESTAMP	speichert Jahr, Monat, Tag, Stunde, Minute, und Sekunde
INTERVAL	Integerfeld, dass eine Zeitspanne repräsentiert je nach Typ
ARRAY	Eine geordnete Sammlung von Elementen mit fester Länge
MULTISET	Eine ungeordnete Sammlung von Elementen mit variabler Länge
XML	speichert XML Daten

Datentyp	Oracle	MS SQL
CHARACTER(n)	CHAR(n) NCHAR(n) (nur Unicode)	CHAR(n) NCHAR(n) (nur Unicode)
VARCHAR(n) oder CHARACTER VARYING(n)	VARCHAR2(n) NVARCHAR2(n) (nur Unicode)	VARCHAR2(n) NVARCHAR2(n) (nur Unicode)
BINARY(n)	RAW(n)	BINARY(n)
BOOLEAN		BIT
VARBINARY(n) oder BINARY VARYING(n)		VARBINARY(n)
INTEGER(p)	NUMBER(p,0)	DECIMAL(p,s) NUMERIC(p,s)
SMALLINT	SMALLINT	SMALLINT
INTEGER	INTEGER, int	int
BIGINT		BIGINT
DECIMAL(p,s)	NUMBER(p, s)	DECIMAL(p,s)
NUMERIC(p,s)	NUMBER(p, s)	NUMERIC(p,s)

Anforderungen an ein Datenbanksystem

Datentyp	Oracle	MS SQL
FLOAT(p)	FLOAT (p) (besser NUMBER)	FLOAT(p)
REAL	NUMBER (p, S)	FLOAT(p)
FLOAT	FLOAT	FLOAT
DOUBLE PRECISION	NUMBER(p, s)	FLOAT(p)
DATE	DATE	DATE(p) DATETIME2(p)
TIME	DATE	TIME(p) DATETIME2(p)
TIMESTAMP	TIMESTAMP	TIMESTAMP ROWVERSION
INTERVAL	INTERVAL YEAR INTERVAL DAY	
ARRAY		
MULTISET		
XML		XML
BLOB	BLOB	BINARY(p) VARBINARY(p) 0 < p < 8001
CLOB	CLOB NCLOB (nur Unicode)	TEXT, NTEXT

Wie erstellt man Tabellen?

Zum Erstellen einer Tabelle können Sie das CREATE TABLE Statement verwenden.

Oracle	MS SQL
SQL> CREATE TABLE dept (deptno NUMBER(2), dname VARCHAR2(14), loc VARCHAR2(13));	SQL> CREATE TABLE dept (deptno NUMERIC(2), dname VARCHAR(14), loc VARCHAR(13));

Zum Überprüfen der Tabellenstruktur verwenden Sie bitte folgenden Befehl:

Oracle	MS SQL
SQL> DESC tabelle;	SQL> EXEC sp_column tabelle

Namenskonventionen für Tabellen:

- Müssen mit einem Buchstaben beginnen
- Können 1–30 Zeichen lang sein
- Dürfen nur A–Z, a–z, 0–9, _, $ und # enthalten
- Dürfen nicht den gleichen Namen wie ein anderes Objekt haben, das demselben Benutzer gehört und im selben Namensraum liegt
- Dürfen kein Server-reserviertes Wort enthalten.

Sie können sehr einfach eine Kopie einer bestehenden Tabelle erstellen:

Oracle	MS SQL
`SQL> CREATE TABLE neuetabelle AS SELECT * FROM altetabelle;`	`SQL> SELECT * INTO neuetabelle FROM altetabelle`

Mit dieser Vorgehensweise übernehmen Sie alle Inhalte und die Struktur der Original-Tabelle, nicht jedoch irgendwelche Einschränkungen oder Berechtigungen.

10.2 Wie ändert man Tabellen?

Nachdem Sie Ihre Tabellen erstellt haben, müssen Sie möglicherweise die Tabellenstrukturen ändern, da Sie vielleicht eine Spalte weggelassen haben oder die Spaltendefinition geändert werden muss. Dies können Sie mit der Anweisung `ALTER TABLE` tun. Sie können einer Tabelle Spalten hinzufügen, indem Sie die Anweisung `ALTER TABLE` mit der ADD-Klausel verwenden.

Oracle	MS SQL
`SQL> ALTER TABLE tabelle ADD COLUMN spaltenname datentyp;`	`SQL> ALTER TABLE tabelle ADD spaltenname datentyp;`

Sollten in der Tabelle schon Werte enthalten sein, so bekommen alle Felder der neuen Spalte den Wert NULL.

Natürlich können Sie auch bestehende Spalten in ihren Eigenschaften verändern. Sie können hierbei den Datentyp, die Größe und eventuell einen Standardwert angeben. Sollten schon Werte in der Spalte enthalten sein, können Sie allerdings den Datentyp nicht mehr verändern, sondern nur noch die Feldlänge vergrößern.

Oracle	MS SQL
`SQL> ALTER TABLE tabelle MODIFY (spalte) datentyp(feldlänge) DEFAULT (wert);`	`SQL> ALTER TABLE tabelle ALTER COLUMN spalte datentyp(feldlänge);`

ALTER TABLE-Richtlinien

Es ist **immer** möglich,

- eine NOT NULL Spalte zu einer NULL-Spalte zu machen,
- die Größe einer Spalte heraufzusetzen,
- eine NULL-Spalte hinzuzufügen.

Eine Spalte, die noch keinen Wert enthält,

- kann in der Größe verkleinert werden,
- kann im Datentyp verändert werden.

Eine Spalte, die für jeden Datensatz einen Wert enthält,

- kann zu einer NOT-NULL Spalte gemacht werden.

Wenn die **gesamte Tabelle** leer ist,

- kann eine NOT-NULL Spalte hinzugefügt werden.

Richtlinien für die ALTER TABLE-Anweisung

- Erhöhen Sie die Breite oder Genauigkeit einer numerischen Spalte.
- Verkleinern Sie die Breite einer Spalte, wenn die Spalte nur Nullwerte enthält, oder wenn die Tabelle keine Zeilen hat.
- Ändern Sie den Datentyp, wenn die Spalte Nullwerte enthält.
- Konvertieren Sie eine CHAR-Spalte in den Datentyp VARCHAR2 oder eine VAR-CHAR2-Spalte in den Datentyp CHAR, wenn die Spalte Nullwerte enthält, oder wenn Sie die Größe nicht verkleinern.
- Eine Änderung des Standardwerts einer Spalte wirkt sich nur auf nachfolgende Einfüge-Operationen in die Tabelle aus.

Sie können eine Spalte aus einer Tabelle über die Anweisung ALTER TABLE mit der Klausel DROP COLUMN löschen.

Richtlinien

- Die Spalte kann Daten enthalten oder nicht.
- Zu einem Zeitpunkt kann nur eine Spalte gelöscht werden.
- Die Tabelle muss nach der Änderung mindestens noch eine Spalte besitzen.
- Eine gelöschte Spalte kann nicht wiederhergestellt werden.

Oracle	MS SQL
SQL> ALTER TABLE tabelle DROP COLUMN spaltenname; SQL> ALTER TABLE tabelle RENAME altername TO neuername	SQL> ALTER TABLE tabelle DROP COLUMN spaltenname; SQL> EXEC sp_rename ‘altername’, ‘neuername’, ‘column’

10.3 Wie löscht man Tabellen?

Tabellen können Sie mit der Anweisung DROP TABLE löschen. Achten sie dabei darauf, dass Sie wirklich die richtige Tabelle löschen. Ein Rollback ist NICHT möglich. Der Server fragt auch nicht nach, ob Sie wirklich löschen möchten.

Oracle	MS SQL

`SQL> DROP TABLE tabelle;`
 	
`SQL> DROP TABLE tabelle PURGE;`
	`SQL> DROP TABLE tabelle;`
	
	`keine Entsprechung`

Alle Daten werden gelöscht, ebenso alle Berechtigungen und Einschränkungen.

Wenn Sie das Schlüsselwort PURGE weglassen, landet die Tabelle im Papierkorb von Oracle. Hierfür ist allerdings eine entsprechende Konfiguration des Servers notwendig. Sie lässt sich daraus komplett mit allen Daten wiederherstellen mit folgender Syntax:

Oracle	MS SQL
`SQL> FLASHBACK TABLE tabelle TO BEFORE DROP;`	`SQL> keine Entsprechung`

Diese Möglichkeit gibt es aber erst ab der Version 10g. In vorherigen Versionen existiert das Schlüsselwort PURGE nicht. Dort ist eine Löschung einer Tabelle nicht mehr rückgängig zu machen.

10.4 Wie ändert man Tabellennamen?

Oracle	MS SQL
`SQL> RENAME altername TO neuername;`	`SQL> EXEC sp_rename 'altername', 'neuername';`

`Ein denkbar einfaches Kommando!`

10.5 Wie löscht man den Inhalt einer Tabelle?

Falls Sie einmal schnell und umfassend eine Tabelle leeren möchten und sie auch keine ROLLBACK-Funktionalität benötigen, verwenden Sie die TRUNCATE-Anweisung.

Oracle	MS SQL
`SQL>  TRUNCATE TABLE tabelle;`	`SQL>  TRUNCATE TABLE tabelle;`

Der Inhalt wird gelöscht, der Speicherplatz freigegeben, die Tabelle in ihrer Struktur bleibt aber erhalten.

Übungen

1. Erzeugen Sie in Ihrer Datenbank die Tabellen emp, dept und salgrade entsprechend der Tabellenübersicht auf Seite 153 bis 156

2. Erstellen Sie die Tabelle DEPARTMENT basierend auf dem Table Instance-Diagramm unten.

Spaltenname	ID	Name
Schlüsseltyp		
Nullen/Eindeutig		
FK-Tabelle		
FK-Spalte		
Datentyp	Number	Varchar
Länge	7	25

3. Erstellen Sie die Tabelle EMPLOYEE basierend auf dem Table Instance-Diagramm unten.

Spaltenname	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Schlüsseltyp				
Nullen/Eindeutig				
FK-Tabelle				
FK-Spalte				
Datentyp	Number	Varchar2	Varchar2	Number
Länge	7	25	25	7

4. Ändern Sie die Tabelle EMPLOYEE so, dass längere Nachnamen der Mitarbeiter in die Spalte passen. Überprüfen Sie die Änderung.

5. Erstellen Sie die Tabelle EMPLOYEE2 basierend auf der Struktur der Tabelle EMP, und fügen Sie nur die Spalten EMPNO, ENAME und DEPTNO ein. Benennen Sie die Spalten in der neuen Tabelle ID, LAST_NAME und DEPT_ID entsprechend.

6. Löschen Sie die Tabelle EMPLOYEE.
7. Benennen Sie die Tabelle EMPLOYEE2 in EMPLOYEE um.

8. Löschen Sie die Spalte LAST_NAME der Tabelle EMPLOYEE. Überprüfen Sie die Änderung.

Notizen

Kapitel 11 Constraints

In diesem Kapitel erfahren Sie Folgendes:

- Sie können Einschränkungen und Schlüssel beschreiben.
- Sie können Einschränkungen und Schlüssel erstellen.

11.1 Was sind Constraints?

- Constraints erzwingen Regeln.
- Constraints wirken rückwirkend.
- Constraints verhindern z. B. das Löschen einer Tabelle oder Zeile, wenn Abhängigkeiten bestehen.
- Folgende Constraint-Typen sind zulässig:

A. NOT NULL

B. UNIQUE

C. PRIMARY KEY

D. FOREIGN KEY

E. CHECK

Server verwenden Constraints, um ungültige Dateneinträge in Tabellen zu verhindern.

Mit Constraints können Sie Folgendes tun:

- Regeln auf Tabellenebene erzwingen, sooft eine Zeile eingefügt, aktualisiert oder aus der Tabelle gelöscht wird. Das Constraint muss erfüllt sein, damit die Operation gelingt.
- Das Löschen einer Tabelle oder Tabellenzeile verhindern, falls Abhängigkeiten zu anderen Tabellen vorhanden sind.

Constraint	***Beschreibung***
NOT NULL	Gibt an, dass diese Spalte keine Nullwerte enthalten darf.
UNIQUE Key	Gibt eine Spalte oder Spaltenkombination an, deren Werte für alle Zeilen in der Tabelle eindeutig sein müssen.
PRIMARY KEY	Identifiziert eindeutig jede Zeile der Tabelle.
FOREIGN KEY	Legt eine Fremdschlüsselbeziehung zwischen der Spalte und einer Spalte der referenzierten Tabelle fest und erzwingt sie.
CHECK	Gibt eine Bedingung an, die wahr sein muss.

11.2 Richtlinien für Constraints

Alle Constraints sind in einem Data Dictionary gespeichert. Constraints lassen sich einfach referenzieren, wenn Sie ihnen einen sinnvollen Namen geben. Constraint-Namen müssen den Standardregeln zum Benennen von Objekten folgen. Wenn Sie Ihren Constraints keine Namen geben, generieren die Server einen Namen mit einem serverspezifischen Format, um einen eindeutigen Constraint-Namen zu erstellen. Der Constraint-Name muss im Schema eindeutig sein. Constraints können gleichzeitig mit dem Erstellen der Tabelle definiert werden oder nachdem die Tabelle erstellt worden ist. Sie können die für eine spezifische Tabelle definierten Constraints anzeigen, indem Sie im Data Dictionary in der Tabelle USER_CONSTRAINTS nachschauen.

Sie können Constraints bereits beim Erstellen einer Tabelle angeben. Ein Beispiel für einen Primärschlüssel auf der Spalte empno und eine NOT-NULL-Einschränkung für die Spalte deptno:

Oracle	MS SQL
SQL> CREATE TABLE emp(empno NUMBER(4), ename VARCHAR2(10), … deptno NUMBER(2) NOT NULL, CONSTRAINT emp_empno_pk PRIMARY KEY (empno));	SQL> CREATE TABLE emp2(empno NUMERIC(4), ename VARCHAR(10), ... deptno NUMERIC(2) NOT NULL, CONSTRAINT emp_empno_pk PRIMARY KEY (empno));

Da ein NOT-NULL-Constraint über eine MODIFY-Anweisung geändert werden kann, wird kein Name vergeben!

11.3 NOT NULL

Das NOT NULL Constraint verhindert das Einfügen von NULL-Werten in eine Spalte.

11.4 UNIQUE

Ein UNIQUE Key-Integritäts-Constraint setzt voraus, dass jeder Wert einer Spalte oder einer Spaltenkombination (Schlüssel) eindeutig ist ¾ d. h. zwei Zeilen einer Tabelle dürfen nicht den gleichen Wert in einer angegebenen Spalte oder Spaltenkombination haben. Die Spalte (oder Spaltengruppe), die in der Definition des Constraints UNIQUE Key enthalten ist, wird *eindeutiger Schlüssel* genannt. Wenn UNIQUE Key mehr als eine Spalte umfaßt, spricht man von einem *zusammengesetzten eindeutigen Schlüssel*.

UNIQUE Key-Constraints erlauben die Eingabe von NULL, sofern Sie keine NOT NULLConstraints für dieselben Spalten definieren. Tatsächlich kann eine beliebige Anzahl Zeilen NULL für Spalten ohne NOT NULL-Constraints enthalten, da NULL nicht als gleichwertig mit etwas anderem betrachtet wird. Ein NULL in einer Spalte (oder in allen Spalten eines zusammengesetzten UNIQUE Key) erfüllt immer ein UNIQUE Key-Constraint. Bei Erstellung eines Unique-Constraints wird automatisch ein Index eingerichtet.

11.5 PRIMARY KEY

Ein PRIMARY KEY-Constraint erstellt einen Primärschlüssel für die Tabelle. Für jede Tabelle kann nur ein Primärschlüssel erstellt werden. Das PRIMARY KEY-Constraint ist eine Spalte oder eine Gruppe von Spalten, welche jede Zeile einer Tabelle eindeutig identifizieren. Dieses Constraint erzwingt Eindeutigkeit der Spalte bzw. Spaltenkombination und stellt sicher, dass keine Spalte, die Teil des Primärschlüssels ist, einen Nullwert enthalten kann:

```
PRIMARY KEY = UNIQUE KEY + NOT NULL
```

11.6 FOREIGN KEY

Das FOREIGN KEY- oder referentielle Integritäts-Constraint bestimmt eine Spalte oder Spaltenkombination als Fremdschlüssel und legt eine Beziehung zwischen einem Primärschlüssel bzw. einem eindeutigen Schlüssel in derselben oder einer anderen Tabelle fest. In unserer Beispieldatenbank wurde DEPTNO als Fremdschlüssel in der Tabelle EMP definiert (abhängige oder referenzierende Tabelle); er referenziert die Spalte DEPTNO der Tabelle DEPT (referenzierte Tabelle).

Ein Fremdschlüsselwert muss mit einem vorhandenen Wert in der referenzierten Tabelle übereinstimmen oder NULL sein. Fremdschlüssel basieren auf Datenwerten und sind rein logische und keine physikalischen Zeiger.

Der Fremdschlüssel wird in der referenzierenden Tabelle definiert; die Tabelle mit der referenzierten Spalte ist die referenzierte Tabelle. Der Fremdschlüssel wird mit Hilfe einer Kombination aus folgenden Schlüsselwörtern definiert:

FOREIGN KEY wird verwendet, um die Spalte in der referenzierenden Tabelle auf der Ebene des Tabellen-Constraints zu definieren.

- REFERENCES identifiziert die Tabelle und Spalte in der referenzierten Tabelle.
- ON DELETE CASCADE gibt an, dass beim Löschen der Zeile in der referenzierten Tabelle die abhängigen Zeilen in der referenzierenden Tabelle ebenfalls gelöscht werden.
- Ohne die Option ON DELETE CASCADE kann die Zeile in der referenzierten Tabelle nicht gelöscht werden, wenn sie in der referenzierenden Tabelle referenziert wird.

Hierfür ein Beispiel, was gleichzeitig erläutert, wie man für eine bereits bestehende Tabelle Constraints ergänzt:

Oracle	MS SQL
SQL> ALTER TABLE EMP ADD CONSTRAINT emp_deptno_fk FOREIGN KEY(deptno) REFERENCES DEPT(deptno);	SQL> ALTER TABLE EMP ADD CONSTRAINT emp_deptno_fk FOREIGN KEY(deptno) REFERENCES DEPT(deptno);

11.7 CHECK

Das CHECK-Constraint definiert eine Bedingung, die jede Zeile erfüllen muss.

Oracle	MS SQL

SQL> ALTER TABLE DEPT ADD CONSTRAINT emp_deptno_ck CHECK(deptno between 10 and 99);	SQL> ALTER TABLE DEPT ADD CONSTRAINT emp_deptno_ck CHECK(deptno between 10 and 99);

Mit diesem Constraint wird festgelegt, dass in der Spalte nur 2-stellige Werte enthalten sein dürfen. Strenggenommen ist ein NOT NULL Constraint ebenfalls ein CHECK Constraint.

11.8 Deaktivieren von Constraints

Wenn man ein Constraint nur vorübergehend entfernen, aber nicht dauerhaft löschen möchte, gibt es die Möglichkeit des Deaktivierens. Die Cascade-Klausel erlaubt es dem User, bei Deaktivierung eines Primärschlüssels auch gleich alle abhängigen Fremdschlüssel mit auszuschalten.

Ein Beispiel:

Oracle	MS SQL
SQL> ALTER TABLE EMP DISABLE CONSTRAINT emp_empno_pk CASCADE;	SQL> ALTER TABLE EMP NOCHECK CONSTRAINT emp_dept_id_fk; (nicht beim PRIMARY KEY)

Ohne Verwendung von CASCADE deaktivieren Sie nur den Primärschlüssel.

11.9 Aktivieren von Constraints

Um einen vorher ausgeschalteten Constraint wieder einzuschalten, muss man folgenden Code eingeben:

Oracle	MS SQL
SQL> ALTER TABLE EMP ENABLE CONSTRAINT emp_empno_pk;	SQL> ALTER TABLE EMP WITH CHECK CHECK CONSTRAINT emp_empno_pk; (nicht beim PRIMARY KEY)

Ein CASCADE ist hierbei NICHT möglich. Sie müssen also alle abhängigen Schlüssel manuell jeden für sich wieder aktivieren.

11.10 Das Löschen von Constraints

Um einen Constraint löschen zu können, muss ich den Namen wissen. Dann ist es ganz einfach:

Oracle	MS SQL

SQL> ALTER TABLE tabelle DROP CONSTRAINT name;	SQL> ALTER TABLE tabelle DROP CONSTRAINT name;

Übungen

1. Fügen Sie der EMPNO der Tabelle EMP zunächst ein NOT NULL Constraint und anschließend ein PRIMARY KEY-Constraint EMP_ID_PK auf hinzu. Das Constraint soll beim Erstellen benannt werden.

2. Erstellen Sie für die ID-Spalte der Tabelle DEPT ein PRIMARY KEY-Constraint DEPT_ID_PK. Das Constraint soll beim Erstellen benannt werden. Beim Microsoft-Server muss zunächst ein NOT NULL Constraint hinzugefügt werden.

3. Fügen Sie in der Tabelle EMP einen Fremdschlüssel EMP_DEPT_ID_FK hinzu, der sicherstellt, dass der Mitarbeiter nicht einer nicht vorhandenen Abteilung zugeordnet wird.

4. Erstellen Sie für die GRADE-Spalte der Tabelle SALGRADE ein PRIMARY KEY-Constraint SALGRADE_GRADE_PK. Das Constraint soll beim Erstellen benannt werden.

Notizen

Kapitel 12 Data Manipulation Language (DML)

12.1 INSERT, UPDATE, DELETE

In diesem Kapitel erfahren Sie Folgendes:

- Sie können jede DML-Anweisung beschreiben.
- Sie können Zeilen in eine Tabelle einfügen.
- Sie können Zeilen in einer Tabelle aktualisieren.
- Sie können Zeilen aus einer Tabelle löschen.
- Sie können Transaktionen steuern.

12.2 Tabellen mit Daten füllen

Bei der Eingabe von Daten in Tabellen können Sie mit zwei Arten von Eingabemöglichkeiten arbeiten.

Entweder geben Sie alle Daten in der Reihenfolge der Spalten an oder Sie benennen die Spalten, in die Werte eingegeben werden sollen.

12.3 Wie gebe ich einen kompletten Datensatz ein?

Wir fangen bei der emp-Tabelle an. Die Anweisung

Oracle	MS SQL
SQL> INSERT INTO emp VALUES (7698,'BLAKE','MANAGER', 7839, '01-05-1981', 2850,NULL,30);	SQL> INSERT INTO emp VALUES (7698,'BLAKE','MANAGER', 7839, '01-05-1981', 2850,NULL,30);

legt einen kompletten Datensatz an, welcher alle Daten zu dem Mitarbeiter BLAKE beinhaltet.

Sie sehen im Beispiel eine der beiden Möglichkeiten, NULL-Werte einzugeben. Die andere liegt darin, die Spalte in der Spaltenliste einfach wegzulassen: Die Spaltenliste kann immer dann entfallen, wenn in alle Spalten ein Wert eingelesen werden soll.

12.4 Wie lege ich einzelne Daten in verschiedenen Spalten an?

Wir geben wieder einen Datensatz in die Tabelle emp ein. Im Unterschied zur Anlage eines kompletten Datensatzes benennen wir die Spaltennamen und weisen diesen die jeweiligen Daten zu. Geben Sie folgende Anweisung im SQL Editor ein:

Oracle	MS SQL

SQL> INSERT INTO emp (empno, ename) VALUES (7566, 'JONES');	SQL> INSERT INTO emp (empno, ename) VALUES (7566, 'JONES');

Mit dieser Syntax können Sie jeweils nur einen Datensatz einfügen. Falls Sie mehrere Datensätze auf einmal, z.B. aus einer bestehenden Tabelle, einfügen wollen, verwenden Sie eine SELECT-Anweisung anstelle der VALUES-Klausel:

Oracle	MS SQL
SQL> INSERT INTO tabelle SELECT-Statement;	SQL> INSERT INTO tabelle SELECT-Statement;

Keine Klammer um das SELECT-Statement, keine VALUES-Klausel, kein AS.

Die Anzahl der Spalten in der SELECT-Liste muss mit der Anzahl der Spalten in der zu füllenden Tabelle übereinstimmen. Ebenso wird eine Überprüfung der Datentypen von Oracle durchgeführt.

12.5 Wie kann ich Austauschvariablen verwenden?

Oracle	MS SQL
SQL> INSERT INTO emp (empno, ename) VALUES (&nummer,'&name');	SQL> keine Entsprechung

Der Benutzer kann also zur Laufzeit die Werte festlegen. Sie können natürlich auch einen Script zum Einfügen schreiben und die Meldung für den Benutzer mit Hilfe des ACCEPT-Befehls übersichtlicher gestalten.

12.6 Wie ändere ich einen Datensatz?

Schauen Sie in der zu ändernden Tabelle nach, welche Datensätze betroffen sind.

Der UPDATE Befehl gibt an welche Tabelle geändert werden soll. Danach folgt die SET Klausel, welche den zu ändernden Wert angibt. Zuletzt wird über die WHERE Klausel die zu ändernde Zeile ausgewählt.

Bitte geben Sie folgenden Code ein, um den Namen des Mitarbeiters mit der Nummer 7566 zu ändern.

Oracle	MS SQL
SQL> UPDATE emp SET ename = 'JANE' WHERE empno = 7566;	SQL> UPDATE emp SET ename = 'JANE' WHERE empno = 7566;

Falls die alle Datensätze auf einmal ändern wollen, lassen Sie die WHERE-Klausel weg.

12.7 Wie lösche ich einen Datensatz?

Schauen Sie sich die Daten des Datensatzes an welchen Sie aus der Tabelle löschen möchten.

Mit dem DELETE FROM Befehl legen Sie fest aus welcher Tabelle die Daten gelöscht werden sollen. Mit der anschließenden WHERE Klausel legen Sie die Zeile(n) fest welche gelöscht werden sollen.

Für das Löschen des Mitarbeiters JONES/ JANE geben Sie folgenden Code ein:

Oracle	MS SQL
SQL> DELETE FROM emp WHERE empno = 7566;	SQL> DELETE FROM emp WHERE empno = 7566;

Sowohl beim Löschen als auch beim Ändern bekommen Sie eine Meldung, wieviele Zeilen betroffen waren. Diese Zahl sollten Sie als Überprüfung verwenden. Erscheint Ihnen die Zahl zu hoch, könnte eventuell die WHERE-Klausel falsch oder gar nicht vorhanden sein!

12.8 COMMIT und ROLLBACK

Der Server gewährleistet Datenkonsistenz basierend auf dem Transaktionskonzept. Transaktionen geben Ihnen mehr Flexibilität und Kontrolle beim Ändern von Daten und sie gewährleisten Datenkonsistenz, falls Benutzer prozesse fehlschlagen oder Systeme ausfallen.

Transaktionen bestehen aus DML-Anweisungen, die zu einer konsistenten Datenänderung führen. Beispielsweise sollte eine Überweisung zwischen zwei Konten zur Belastung eines Kontos und zur Gutschrift auf dem anderen über denselben Betrag führen. Beide Aktionen sollten entweder fehlschlagen oder erfolgreich durchgeführt werden. Die Gutschrift sollte nicht ohne die Belastung gespeichert werden.

Eine Transaktion beginnt, wenn die erste ausführbare SQL-Anweisung angetroffen wird und endet wenn folgendes eintritt:

- Wenn eine COMMIT- oder ROLLBACK-Anweisung ausgegeben wird
- Wenn eine DDL-Anweisung, wie z.B. CREATE, ausgegeben wird
- Wenn eine DCL-Anweisung ausgegeben wird
- Wenn bestimmte Fehler, wie Deadlocks, auftreten
- Wenn der Benutzer SQL-Editor beendet
- Wenn ein Rechner oder ein System ausfällt

Nachdem eine Transaktion endet, startet die nächste ausführbare SQL-Anweisung automatisch die nächste Transaktion.

Eine DDL-Anweisung oder eine DCL-Anweisung wird automatisch gespeichert und beendet deshalb implizit eine Transaktion.

Ein Commit beendet eine Transaktion und schreibt ihre Änderungen in der Datenbank fest. Ein Rollback macht sämtliche Änderungen, die Sie seit Beginn einer Transaktion durchgeführt haben, rückgängig. Bitte beachten Sie, dass Sie jedes Mal von COMMIT oder ROLLBACK eine Erfolgsmeldung erhalten, egal, ob wirklich ein COMMIT oder ROLLBACK durchgeführt wurde.

Ein Schließen des SQL-Fensters führt zu einem ROLLBACK! Beenden Sie also Ihre Sitzung mit einem EXIT-Befehl.

Sie können zusätzlich hinter jedem INSERT, UPDATE oder DELETE einen SAVEPOINT setzen.

Oracle	MS SQL
SQL> SAVEPOINT a;	SQL> BEGIN TRANSACTION name SQL> SAVE TRANSACTION name oder: COMMIT TRANSACTION

Nun können Sie auch nur Abschnitte einer Transaktion zurückrollen, indem Sie folgende Anweisung schreiben:

Oracle	MS SQL
SQL> ROLLBACK TO a;	SQL> ROLLBACK TRANSACTION

Sie können aber jederzeit die gesamte Transaktion zurückrollen, egal ob sie Savepoints verwendet haben oder nicht.

Übungen

1. Befüllen Sie die Tabellen dept, emp und salgrade mit den Werten auf Seite 153 bis 156

9. Erstellen Sie die Tabelle MY_EMPLOYEE unter Verwendung der folgenden Ansicht.

ID	NUMBER(4)	NOT NULL
LAST_NAME	VARCHAR2(25)	
FIRST_NAME	VARCHAR2(25)	
USERID	VARCHAR2(8)	
SALARY	NUMBER(9,2)	

ID	*LAST_NAME*	*FIRST_NAME*	*USERID*	*SALARY*
1	Patel	Ralph	rpatel	795
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audry	aropebur	1550

10. Fügen Sie die erste Zeile in die Tabelle ein, lassen Sie dabei die Spaltenliste weg.
11. Fügen Sie die restlichen Zeilen hinzu, diesmal mit Spaltenliste.
12. Ändern Sie den Nachnamen von Zeile 3 in Drexler.
13. Ändern Sie für alle Mitarbeiter, die weniger als 900 verdienen, das Gehalt auf 1000.
14. Löschen Sie Betty Dancs.
15. Schreiben Sie alle Änderungen fest.
16. Setzen Sie einen Savepoint.
17. Löschen Sie alle Daten aus der Tabelle.
18. Schauen Sie, ob die Tabelle wirklich leer ist.
19. Machen Sie das Löschen der Daten rückgängig. Sie sollten aber danach 4 Zeilen in der Tabelle haben!
20. Schreiben Sie alle Änderungen fest.

Kapitel 13 SELECT

In diesem Kapitel erfahren Sie Folgendes:

- Sie können die Möglichkeiten von SELECT-Anweisungen auflisten.
- Sie können einfache SELECT-Anweisungen ausführen.

13.1 Tabellendaten ausgeben

Hauptbestandteile zum Anzeigen und Auslesen der Daten aus Tabellen sind die Klauseln SELECT und FROM.

Mit diesen beiden geben wir zum einem an, aus welchen Spalten wir die Informationen haben wollen (SELECT), und zum anderen, aus welchen Tabellen diese stammen (FROM).

13.2 Wie lasse ich mir alle Daten einer Tabelle anzeigen?

Um alle Spalten aus einer Tabelle auslesen zu können, verwenden Sie den * in der SELECT-Klausel. Versuchen Sie also folgenden Code, um sich die Daten der Tabelle dept anzuzeigen:

Oracle	MS SQL
`SQL>  SELECT * FROM dept;`	`SQL>  SELECT * FROM dept;`

Nach dem Ausführen sollten Sie folgende Zeilen aus der Datenbank zurück erhalten

```
DEPTNO     DNAME          LOC
---------  --------------  -------------
      10   ACCOUNTING     NEW YORK
      20   RESEARCH       DALLAS
      30   SALES          CHICAGO
      40   OPERATIONS     BOSTON
```

Ihr Ergebnis sollte dem oben gezeigten weitestgehend entsprechen (Formatierung kann abweichen).

13.3 Wie wähle ich einzelne Spalten aus, die ich mir anzeigen lassen möchte?

Da Sie in vielen Fällen nicht immer alle Werte einer Tabelle brauchen bzw. es bei sehr großen Tabellen auch hinderlich sein kann, müssen Sie eine Auswahl der Spalten treffen, die Sie benötigen.

Geben Sie die Spaltennamen, die Sie anzeigen möchten, hinter der SELECT Klausel ein.

Beim Anzeigen mehrerer Spalten trennen Sie diese bitte durch ein Komma. Sie können den Spalten auch andere Überschriften in der Anzeige geben. Geben Sie die Überschrift mit einer Leerstelle getrennt nach dem Spaltennamen ein. Dieser wird bei der Ausgabe in Großbuchstaben angezeigt. Möchten Sie Groß- und Kleinschreibung, Leerstellen, Sonderzeichen oder Zahlen verwenden, so müssen Sie diese nach dem Spaltennamen in doppelte Anführungszeichen setzen.

Geben Sie bitte folgenden Code ein, um alle Mitarbeiter mit Ihrer Nummer und dem Namen auszugeben. Die Spalten sind mit Nummer und NAMEN zu beschriften.

Oracle	MS SQL
`SQL> SELECT empno "Nummer", ename namen FROM emp;`	`SQL> SELECT empno "Nummer", ename namen FROM emp;`

Nach dem Ausführen erhalten Sie folgende Zeilen zurück.

```
Nummer                    NAMEN
--------------- ------------------ ------------------
7369                        SMITH
7499                        ALLEN
7521                        WARD
…                           …

14 Zeilen ausgewählt
```

13.4 Kann ich mit den Daten aus Spalten auch rechnen?

Man kann ohne weiteres mit Spalten bzw. den Daten in den Spalten rechnen, einzige Bedingung ist, dass diese Spalte den Datentyp NUMBER bzw. NUMERIC besitzt.

Somit können Sie jetzt auch nachvollziehen, warum es beim Anlegen von Tabellen so wichtig ist, die richtigen Datentypen auszuwählen.

Sie können Zahlen durch Operatoren (* / + -) auf Spalten anwenden.

Den folgenden Code geben Sie bitte ein, um sich die Jahresgehälter plus ein Weihnachtsgeld von 500,- für alle Mitarbeiter anzuzeigen (es gilt Punktrechnung vor Strichrechnung).

Oracle	MS SQL

Oracle	MS SQL
SQL> SELECT ename, 12*sal+500 FROM emp;	SQL> SELECT ename, 12*sal+500 FROM emp;

Bei der Ausgabe müssten Sie folgendes Ergebnis erhalten.

```
ENAME         SAL      12*SAL+500
--------------------- -------- -----------------
SMITH         800      10100
ALLEN         1600     19700
WARD          1250     15500
JONES         2975     36200
MARTIN        1250     15500
…
```

14 Zeilen ausgewählt.

Um die Punktrechnung vor Strichrechnung zu umgehen, müssen Sie Klammern verwenden.

Sie möchten z.B. wissen, wieviel jeder Mitarbeiter jährlich verdient, wenn er eine monatliche Gehaltserhöhung von 100,- erhält.

Geben Sie dieses Beispiel ein:

Oracle	MS SQL
SQL> SELECT ename, sal, 12*(sal+100) FROM emp;	SQL> SELECT ename, sal, 12*(sal+100) FROM emp;

Nach dem Ausführen sollten Sie folgende Ausgabe erhalten

```
ENAME         SAL       12*(SAL+100)
--------------------- -------- -------------------
SMITH         800       10800
ALLEN         1600      20400
WARD          1250      16200
JONES         2975      36900
MARTIN        1250      16200
…
```

14 Zeilen ausgewählt.

Wie Sie sehen, wird zuerst die Berechnung in der Klammer ausgeführt und danach dieses Ergebnis mit 12 multipliziert.

13.5 Kann ich Spaltendaten auch miteinander verknüpfen?

Es ist ohne weiteres möglich, Spaltendaten miteinander zu verknüpfen, für eine gute Lesbarkeit ist es sogar nötig.

Das Einzige, was Sie dafür benötigen, ist der Verkettungsoperator (|| bzw. +), welcher zwischen den zu verknüpfenden Spalten oder Literalen stehen muss. Nehmen wir an, Sie möchten wissen, wer was im Unternehmen so macht.

Somit müssten Sie aus der Tabelle ***emp*** die Spalten **ename** und **job** verbinden. Für die Optik bei der Ausgabe schreiben wir noch ein wenig Text dazwischen. Geben Sie für die oben genannte Ausgabe folgenden Code ein:

Oracle	MS SQL
`SQL> SELECT ename||' ist von Beruf '|| job "Wer macht was" FROM emp;`	`SQL> SELECT ename + ' ist von Beruf ' + job "Wer macht was" FROM emp;`

Nach Ihrer Bestätigung durch die Eingabetaste erhalten Sie folgende Ausgabe.

```
Wer macht was
--------------------------------------------
SMITH ist von Beruf CLERK
ALLEN ist von Beruf SALESMAN
WARD ist von Beruf SALESMAN
JONES ist von Beruf MANAGER
MARTIN ist von Beruf SALESMAN
…

14 Zeilen ausgewählt.
```

13.6 Wie unterdrückt man doppelte Ausgaben?

Möchten Sie doppelte Ausgaben unterdrücken, so verwenden Sie das Schlüsselwort DISTINCT.

Oracle	MS SQL
`SQL> SELECT DISTINCT deptno FROM emp;`	`SQL> SELECT DISTINCT deptno FROM emp;`

```
deptno
------
10
20
30

(3 row(s) affected)
```

Diese Abfrage liefert Ihnen gleich lautende Einträge in ein Feld der Spalte *deptno* nur ein einziges Mal. In unserem Beispiel teilt sie Ihnen also mit, welche Abteilungen in Ihrer Tabelle vorhanden sind.

Oracle	MS SQL
SQL> SELECT DISTINCT deptno, job FROM emp;	SQL> SELECT DISTINCT deptno, job FROM emp;

```
deptno job
------ ---------------
10     CLERK
10     MANAGER
10     PRESIDENT
20     ANALYST
20     CLERK
20     MANAGER
30     CLERK
30     MANAGER
30     SALESMAN

(9 row(s) affected)
```

Wenden Sie das Schlüsselwort DISTINCT auf mehrere Spalten an, wird das Vorkommen doppelter Kombinationen unterdrückt.

Übungen

1. Geben Sie alle Orte aus, die sich in der Tabelle dept befinden.

21. Zeigen Sie alle Mitarbeiter an und das dazugehörige Einstellungsdatum.

22. Zeigen Sie alle Gehälter mit den jeweiligen Mitarbeiternamen an und wie viel Gehalt Sie per Monat, per Quartal und per Jahre bekommen.

23. Geben Sie für alle Mitarbeiter folgenden Text mit der Gliederung wie in der folgenden Beispielausgabe aus:
    SCOTT arbeitet als ANALYST und bekommt dafür 3000 Euro im Monat

Kapitel 14 WHERE

In diesem Kapitel erfahren Sie Folgendes:

- Sie können die durch eine Abfrage abgerufenen Zeilen einschränken.
- Sie können die durch eine Abfrage abgerufenen Zeilen sortieren.

Beim Abrufen von Daten aus der Datenbank möchten Sie vermutlich die angezeigten Datenzeilen einschränken oder festlegen, in welcher Reihenfolge die Zeilen angezeigt werden. Dieses Kapitel behandelt die SQL-Anweisungen, die Sie verwenden, um diese Aktionen auszuführen.

14.1 Die WHERE-Klausel

SELECT Spaltenliste oder *

FROM Tabelle

WHERE Bedingung; (Wenn die Bedingung Zeichen oder Datumswerte enthält, müssen diese in Hochkommata gesetzt werden; Ziffern funktionieren auch ohne.)

Oracle	MS SQL
SQL> SELECT ename, job from emp where deptno = 20; SQL> SELECT ename, job from emp where job = 'MANAGER';	SQL> SELECT ename, job from emp where deptno = 20; SQL> SELECT ename, job from emp where job = 'MANAGER';

Achtung:
Bei Zeichenketten und Datumswerten müssen Hochkommata gesetzt werden, bei Zeichenketten wird auf Groß- und Kleinschreibung geachtet!

14.2 Vergleichsoperatoren

Sie haben folgende Vergleichsoperatoren zur Verfügung, um nach Bedingungen suchen zu können:

`LIKE, BETWEEN – AND, IN, =, != (oder <>) , >, >=, <, <= , IS NULL , NOT`

Hier die genauen Beschreibungen:

LIKE

%	steht für 0 bis unendlich viele Zeichen
_	steht für genau ein Zeichen

Oracle	MS SQL

SQL> SELECT * from emp where ename like '_A%' (alles, wo an zweiter Stelle ein „A“ steht) SQL> SELECT ename from emp where job **like** 'SALES%' (alles, was mit Sales anfängt, egal wie es weitergeht)	SQL> SQL> SELECT * from emp where ename like '_A %' (alles, wo an zweiter Stelle ein A steht) SQL> SELECT ename from emp where job **like** 'SALES%' (alles, was mit Sales anfängt, egal wie es weitergeht)

Achtung:
`SELECT ename from emp where job = 'SALES%'` (hier wird mit SALES% verglichen!!)

Ein Problem kann sich ergeben, wenn Sie nach einem der Platzhalter suchen, also _ oder %.

Nehmen wir an, dass die Tabelle dept folgenden Inhalt hat:

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        50 HEAD_QUARTERS  DETROIT
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

Es soll nun nach allen Einträgen in der Spalte dname gesucht werden, die einen Unterstrich enthalten, also eigentlich einen Platzhalter.

Die Syntax dafür lautet folgendermaßen:

Oracle	MS SQL
SQL> select dname from dept where dname like '%_%' escape '\’;	SQL> select dname from dept where dname like '%[_] %';

```
DNAME
--------------
HEAD_QUARTERS
```

Nach der ganz normalen Abfrage verwenden Sie das Schlüsselwort “escape” und definieren danach in Hochkomma ein Zeichen. Das erste Platzhalterzeichen, was nach dem definierten Zeichen, hier ein ‚\‘ kommt, wird von Oracle dann als Suchbegriff interpretiert.

BETWEEN – AND

Oracle	MS SQL
SQL> SELECT ename from emp where sal between 1000 and 2000; (alles, wo sal zwischen 1000 und 2000 liegt, inklusive der angegebenen Werte, der erste Wert MUSS kleiner sein als der zweite)	SQL> SELECT ename from emp where sal between 1000 and 2000; (alles, wo sal zwischen 1000 und 2000 liegt, inklusive der angegebenen Werte, der erste Wert MUSS kleiner sein als der zweite)

IN

Oracle	MS SQL
SQL> SELECT * FROM emp WHERE deptno IN (10,30); (alles, was mit einem der Werte innerhalb der Klammer übereinstimmt, also wie ein **OR**)	SQL> SELECT * FROM emp WHERE deptno IN (10,30); (alles, was mit einem der Werte innerhalb der Klammer übereinstimmt, also wie ein **OR**)

IS NULL

Oracle	MS SQL
SQL> SELECT * from emp WHERE comm is null; (alles, wo in der Spalte Comm ein NULL-Wert steht)	SQL> SELECT * from emp WHERE comm is null; (alles, wo in der Spalte Comm ein NULL-Wert steht)

Alle diese Operatoren können mit NOT verneint werden!

14.3 Verknüpfungsoperatoren

Zum Verknüpfen von mehreren Bedingungen stehen Ihnen die beiden Verknüpfungsoperatoren AND und OR zur Verfügung.

AND:
SELECT spalte1, spalte2

FROM tabelle

WHERE bedingung1

AND bedingung2;

Zur Ausgabe müssen **beide** Bedingungen erfüllt sein.

OR:

SELECT spalte1, spalte2

FROM tabelle

WHERE bedingung1

OR bedingung2;

Zur Ausgabe muss nur **eine** Bedingung erfüllt sein.

Falls Sie in einer Abfrage AND und OR mischen müssen, beachten Sie bitte, dass zuerst AND und danach OR ausgewertet wird.

SELECT spalte1,spalte2

FROM tabelle

WHERE bedingung1

AND bedingung2

OR bedingung3;

Ein Beispiel dafür:

Oracle	MS SQL
SQL> select ename, sal from emp where job = 'CLERK' and sal < 1100 or job = 'MANAGER'	SQL> select ename, sal from emp where job = 'CLERK' and sal < 1100 or job = 'MANAGER'

SMITH 800

JONES 2975

BLAKE 2850

CLARK 2450

JAMES 950

Oracle	MS SQL
SQL> select ename, sal from emp where job = 'CLERK' or sal < 1100 and job = 'MANAGER'	SQL> select ename, sal from emp where job = 'CLERK' or sal < 1100 and job = 'MANAGER'

SMITH 800

ADAMS 1100

JAMES 950

MILLER 1300

Sollten Sie eine andere Reihenfolge benötigen, können Sie genau wie in der Mathematik mit Klammern arbeiten.

14.4 Die ORDER BY-Klausel

Die Reihenfolge der in einem Abfrageergebnis zurückgegebenen Zeilen ist undefiniert. Die ORDER BY-Klausel kann verwendet werden, um die Zeilen zu sortieren. Falls Sie sie verwenden, müssen Sie die ORDER BY-Klausel ans Ende stellen. Sie können einen Ausdruck oder ein Alias zum Sortieren festlegen.

Oracle	MS SQL
SQL> SELECT ename,job,sal FROM emp ORDER BY sal;	SQL> SELECT ename,job,sal FROM emp ORDER BY sal;

Die Standardreihenfolge ist von unten nach oben, also bekommen Sie den Mitarbeiter mit dem geringsten Gehalt als Erstes angezeigt. Wollen Sie diese Reihenfolge ändern, verwenden Sie die Anweisung DESC.

Oracle	MS SQL
SQL> SELECT ename,job,sal FROM emp ORDER BY sal DESC;	SQL> SELECT ename,job,sal FROM emp ORDER BY sal DESC;

Übungen

1. Erstellen Sie eine Abfrage, um Namen und Gehälter der Mitarbeiter anzuzeigen, die mehr als $2850 verdienen.

2. Erstellen Sie eine Abfrage, um Namen und Abteilungsnummer für die Mitarbeiternummer 7566 anzuzeigen.

3. Zeigen Sie die Namen, Beruf und Eintrittsdatum aller Mitarbeiter an, die zwischen 20. Februar 1981 und 1. Mai 1981 eingestellt wurden. Ordnen Sie die Abfrage in aufsteigender Reihenfolge nach dem Eintrittsdatum.

4. Zeigen Sie die Namen aller Mitarbeiter an, deren Namen zwei L enthalten und in Abteilung 30 arbeiten oder deren Manager der Mitarbeiter mit der Nummer 7782 ist.

Notizen

Kapitel 15 SINGLE-ROW-Funktionen

In diesem Kapitel erfahren Sie Folgendes:

- Sie können verschiedene in SQL verfügbare Funktionstypen beschreiben.
- Sie können Zeichenkettenfunktionen, arithmetische Funktionen und Datumsfunktionen in SELECT-Anweisungen beschreiben.

15.1 Funktionen

Funktionen sind SQL-Bestandteile, die die nachfolgenden Aufgaben erfüllen:

- Berechnungen durchführen
- Einzelne Datenelemente ändern
- gruppierte Ausgaben erstellen
- Datumsangaben und Zahlen in nicht-standardisierter Form darstellen
- Spaltendatentypen umwandeln

15.2 Einteilung der Funktionen

Es gibt zwei Arten von Funktionen: **Single-Row-Funktionen**, also einzeilige Funktionen, und **Multi-Row-Funktionen**, also mehrzeilige.

In diesem Projektschritt werden wir uns mit Single-Row-Funktionen beschäftigen, die Multi-Row-Funktionen kommen in einem späteren Projektschritt zur Sprache.

15.3 Überblick über die Single-Row-Funktionen

Funktionsname(Spalte/Zeichenkette/Ausdruck,[Argument1,Argument2,...])

- Single-Row-Funktionen werden auf jede einzelne Zeile angewendet und liefern auch für jede Zeile ein Ergebnis.
- An Single-Row-Funktionen können Argumente übergeben werden, die dann bearbeitet werden.
- Single-Row-Funktionen können verschachtelt werden.
- Single-Row-Funktionen können in SELECT-, WHERE- und ORDER BY-Klauseln verwendet werden.
- Der zurückgegeben Datentyp kann sich vom referenzierten Datentyp unterscheiden.

15.4 Single-Row-Funktionen

- Zeichenkettenfunktionen
- Arithmetische Funktionen
- Datumsfunktionen
- Konvertierungsfunktionen
- allgemeine Funktionen:
  - NVL-Funktion (Oracle) bzw. COALESCE-Funktion (Standard-SQL), ISNULL-Funktion (MS SQL)
  - Decode-Funktion
  - CASE-Funktion

15.5 Zeichenkettenfunktionen

Zeichenkettenfunktionen bearbeiten Zeichen als Eingabe und geben entweder Zeichen oder Zahlenwerte zurück.

Es gibt Funktionen zur Umwandlung der Groß- bzw. Kleinschreibung sowie Funktionen zum Bearbeiten von Zeichen.

LOWER, UPPER und INITCAP (nur Oracle) sind die drei Funktionen zur Umwandlung der Groß-/Kleinschreibung von Zeichenketten.

- LOWER: Konvertiert gemischte Zeichen oder Großbuchstaben in Kleinbuchstaben
- UPPER: Konvertiert gemischte Zeichen oder Kleinbuchstaben in Großbuchstaben
- INITCAP: Konvertiert jeweils den ersten Buchstaben eines Worts in einen Großbuchstaben und die verbleibenden Zeichen in Kleinbuchstaben.

In MS SQL muss diese Funktion durch

```
SQL>   Left(upper(ename) , 1) + substring(lower(ename), 2, len(ename))
```

ersetzt werden. Hiebei liefert `Left(upper(ename) , 1)` den ersten Buchstaben als Großbuchstaben. Der Ausdruck `substring(lower(ename), 2, len(ename))` liefert den zunächst mit `lower(ename)` in Kleinbuchstaben umgewandelten Feldinhalt vom zweiten Buchstaben bis zum Ende des Feldinhaltes zurück.

Oracle	MS SQL
`SQL> SELECT INITCAP(ename) FROM emp;`	`SQL> SELECT Left(upper(ename) , 1) + substring(lower(ename), 2, len(ename)) FROM emp;`

Zur Bearbeitung von Zeichenketten stehen die folgenden Funktionen zur Verfügung:

Oracle	MS SQL
CONCAT verkettet zwei Ausdrü-	CONCAT verkettet zwei Ausdrü-

	cke miteinander. Mit CONCAT können höchstens zwei Parameter verwendet werden.		cke miteinander. Mit CONCAT können höchstens zwei Parameter verwendet werden.
SUBSTR	extrahiert eine Zeichenkette der angegebenen Länge.	SUBSTRING	extrahiert eine Zeichenkette der angegebenen Länge.
LENGTH	gibt die Länge einer Zeichenkette mit einem numerischen Wert wieder.	LEN	gibt die Länge einer Zeichenkette mit einem numerischen Wert wieder.
LPAD	füllt den Zeichenwert so mit Leerzeichen auf, dass ein rechtsbündiger Blocksatz entsteht. (RPAD entsprechend umgekehrt)	RIGHT(REPLICATE(*Zeichen*, L*änge*) + LEFT(*Wert*, L*änge*), L*änge*)	füllt den Zeichenwert so mit Leerzeichen auf, dass ein rechtsbündiger Blocksatz entsteht.
		LEFT(LEFT(*Wert*, L*änge*) + REPLICATE(*Zeichen*, L*änge*), L*änge*)	
TRIM	entfernt Zeichen am Anfang oder Ende (oder beides) einer Zeichenkette. Falls das zu entfernende Zeichen oder die zu bearbeitende Zeichenkette ein Zeichen-Literal ist, muss es in einfache Anführungszeichen gesetzt werden.	LTRIM RTRIM	entfernt Leerzeichen am Anfang oder Ende (oder beides) einer Zeichenkette. Falls das zu entfernende Zeichen oder die zu bearbeitende Zeichenkette ein Zeichen-Literal ist, muss es in einfache Anführungszeichen gesetzt werden.

Schauen wir uns anhand von einfachen Beispielen die einzelnen Funktionen einmal an.

- LOWER('HERMANN') liefert als Ausgabe herrmann .
- UPPER ('Hermann') liefert als Ausgabe HERRMANN.
- INITCAP ('HERRMANN meier') liefert als Ausgabe Herrmann Meier. Diese Funktion ist in MS SQL so nicht möglich.
- CONCAT ('Hermann', 'Meier') liefert als Ausgabe HermannMeier, also beide Strings zusammengefasst ohne Lücke!
- LENGTH ('HERRMANN') bzw. LEN ('HERRMANN') liefert als Ausgabe die Zahl 8, also die Anzahl der Buchstaben.
- SUBSTR('HERRMANN', 1,3) bzw. SUBSTRING('HERRMANN', 1,3) liefert als Ausgabe HER.

Das erste Argument ist eine Zeichenkette oder eine Spalte, die Zeichenketten enthält. Das zweite Argument eine Zahl, die den Startpunkt angibt, in diesem Beispiel der erste Buchstabe der Zeichenkette.

Das dritte Argument gibt an, wie viele Zeichen ab dem Startpunkt ausgegeben werden sollen. Falls bis zum Ende der Zeichenkette ausgegeben werden soll, darf das letzte Argument entfallen.

Oracle	MS SQL
SQL> SELECT INSTR('HERRMANN','M') FROM DUAL;	SQL> CHARINDEX('M', 'HERRMANN');

liefert als Ausgabe die Zahl 5, also steht das M an der 5ten Position.

Oracle	MS SQL
SQL> SELECT LPAD(sal,10,'*') FROM emp WHERE LOWER(ename) = 'king';	SQL> SELECT RIGHT(REPLICATE('*', 10) + LEFT(sal, 10), 10) FROM emp WHERE LOWER(ename) = 'king';

liefert als Ausgabe ******5000 bzw. ***5000.00 beim Datensatz KING, das erste Argument wählt also die Spalte aus, das zweite die Breite der Spalte für die Ausgabe, und das dritte füllt mit dem angegeben Zeichen die Spalte auf, bis diese mit in diesem Beispiel 10 Zeichen gefüllt ist. MS-SQL gibt wegen des Datentyps der Spalte zwei Stellen nach dem Komma aus. Dies lässt sich aber mit einer im folgenden noch besprochenen Funktion ändern.

Oracle	MS SQL
SQL> SELECT TRIM('H' from 'HERRMANN') FROM DUAL	SQL> SELECT RIGHT('HERRMANN', LEN('HERRMANN') - 1);

liefert als Ausgabe ERRMANN, schneidet also den ersten Buchstaben einfach ab!

Oracle	MS SQL

SQL> SELECT ename, CONCAT (ename, job), LENGTH(ename), INSTR(ename,'A') FROM emp WHERE SUBSTR(job,1,4)='PRES';	SQL> SELECT ename, CONCAT (ename, job), LEN(ename), CHARINDEX('A', ename) FROM emp WHERE SUBSTRING(job,1,4)='PRES' ;

Ausgabe:

```
ENAME      CONCAT(ENAME,JOB)   LENGTH(ENAME) INSTR(ENAME,'A')
-------------------------------------------------------------
KING       KINGPRESIDENT               4               0
```

Oracle	MS SQL
SQL> SELECT INITCAP(ename) FROM emp WHERE LOWER(ename)='king';	SQL> SELECT Left(upper(ename) , 1) + substring(lower(ename), 2, len(ename)) FROM emp WHERE LOWER(ename)='king';

Ausgabe:

```
INITCAP(ENAME)
----------
King
```

15.6 Arithmetische Funktionen

Oracle	MS SQL
SQL> ROUND(Spalte/Ausdruck, n)	SQL> ROUND(Spalte/Ausdruck, n)

rundet die Spaltenwerte oder den Ausdruck auf die in N angegebene Stellenzahl auf oder ab. Wird N nicht angegeben, wird auf ganze Zahlen gerundet, -1 rundet auf die erste Zahl links neben dem Dezimalpunkt gerundet, -2 auf die zweite etc.

Oracle	MS SQL
SQL> TRUNC(Spalte/Ausdruck, n)	SQL> ROUND(Spalte/Ausdruck, n, 1)

TRUNC(Spalte/Ausdruck, n) unterscheidet sich von ROUND in der Eigenschaft, dass nicht gerundet, sondern einfach abgeschnitten wird. MS-SQL verwendet hierfür ROUND mit einem dritten Parameter. Ist dieser Parameter 0 (default) wird gerundet, ist der Wert ungleich 0 wird abgeschnitten.

ROUND (45.6777, 2)	liefert also 45.68,
TRUNC (45.6777, 2)	[bzw. ROUND(45.6777, 2, 1)] aber 45.67.
MOD(1600,300)	gibt den Restbetrag der Division des ersten Argumentes durch das zweite an, in diesem Beispiel also 1600/300 = 3 Rest 100.

Ausgabe ist also 100.

15.7 Behandlung des Datumformats DATE?

Die Tabelle DUAL (nur Oracle)

Die Tabelle DUAL gehört bei Oracle dem Benutzer SYS; alle Benutzer können auf sie zugreifen. Sie enthält eine Spalte, DUMMY, und eine Zeile mit dem Wert X. Die Tabelle DUAL ist nützlich, wenn Sie einen Wert nur einmal zurückgeben möchten, z. B. den Wert einer Konstanten, einer Pseudospalte oder eines Ausdrucks, der nicht aus einer Tabelle mit Benutzerdaten abgeleitet ist. Für verschiedene Berechnungen werden keine Daten aus tatsächlichen Tabellen benötigt, so dass DUAL im Allgemeinen aufgrund der erforderlichen vollständigen Syntax einer SELECT-Anweisung (SELECT- und FROM-Klausel sind zwingend erforderlich) verwendet wird.

Die Tabelle DUAL ist in MS-SQL nicht vorhanden und wird auch nicht benötigt, da hier eine SELECT-Anweisung auch ohne Tabellenangabe syntaktisch erlaubt ist.

Beispiel zum Auslesen der Systemzeit:
SYSDATE (Oracle) bzw. SYSDATETIME oder GETDATE (MS-SQL)

Oracle	MS SQL
Zeigen Sie das aktuelle Datum mit der Tabelle DUAL an.	Zeigen Sie das aktuelle Datum an.

`SQL> SELECT sysdate FROM dual;`	`SQL> SELECT sysdatetime() AS Systemzeit;`
	
	`SQL> SELECT getdate() AS Systemzeit;`

SYSDATE
-------- 	
22.10.02
	`Systemzeit`
	
	`2017-02-02 13:59:25.0966364`
	
	`2017-02-02 14:04:00.330`

15.8 Datumsarithmetik

Sie können eine Zahl zu einem Datum addieren oder sie abziehen und erhalten als Ergebnis ein Datum.

Beispiel:

Oracle	MS SQL
SQL> SELECT sysdate – 2 from DUAL;	SQL> SELECT getdate() – 2 AS Systemzeit;
liefert als Ausgabe das Datum von vorgestern, also in unserem Fall:	
SYSDATE-2 -------- 20.10.02	Systemzeit 2017-01-31 14:11:10.430 Aber: SQL> select sysdatetime()-2 as Systemzeit; liefert: Meldung 206, Ebene 16, Status 2, Zeile 1 Operandentypkollision: datetime2 ist inkompatibel mit int

Des Weiteren können Sie zwei Datumswerte subtrahieren und erhalten als Ergebnis die Anzahl der Tage zwischen diesen Daten.

Oracle	MS SQL
SQL> SELECT SYSDATE – hiredate FROM emp WHERE ename ='KING';	SQL> SELECT getdate() - cast(hiredate as DateTime) AS 'Systemzeit – hiredate' FROM emp WHERE ename ='KING';
sagt Ihnen, wieviele Tage KING in der Firma ist.	
SYSDATE-HIREDATE ---------- 7644,62065	Systemzeit – hiredate 1935-03-20 14:21:54.347

15.9 Andere Datumsfunktionen

Datumsfunktionen werden auf Oracle-Datumsangaben angewendet. Alle Datumsfunktionen geben einen Wert vom Datentyp DATE zurück, mit Ausnahme der Funktionen zur Bildung von Datums-Differenzen, die einen numerischen Wert zurückgeben.

Oracle	MS SQL
MONTHS_BETWEEN(*datum1,* datum2):	DATEDIFF(datepart, *datum1,* datum2)
Findet die Anzahl der Monate zwischen datum1 und datum2. Das Ergebnis kann positiv oder negativ sein. Wenn *datum1* später als *datum2 ist*, ist das Ergebnis positiv; wenn *datum1* früher als *datum2* ist, ist das Ergebnis negativ. Der nicht ganzzahlige Teil des Ergebnisses steht für einen Teil des Monats.	
`SQL> Select months_between(hiredate, sysdate) from emp where ename = 'KING';`	`SQL> SELECT datediff(month, hiredate, getdate()) AS 'Monate' FROM emp WHERE ename ='KING';`
`Liefert: -422,5375907258064516129 03225806451612903`	`Liefert: 423 (Ergebnis ist ganzzahlig da int)`

Oracle	MS SQL
ADD_MONTHS(*datum, n*):	DATEADD(datepart, n, datum)
Fügt dem Datum *datum n* Kalendermonate hinzu. n muss eine Ganzzahl sein und kann negativ sein.	
`SQL> SELECT ADD_MONTHs(sysdate, 2) FROM dual;`	`SQL> SELECT DATEADD(month, 2, sysdatetime());`

Oracle	MS SQL
NEXT_DAY(*datum,* '*char*'): Findet das Datum des Wochentags ('*char*'), der auf *datum* folgt. *char* kann eine Zeichenkette sein oder eine Zahl, die für einen Tag steht.	Keine Entsprechung in MS-SQL
`SQL> SELECT NEXT_DAY(sysdate, 'Montag') FROM dual;`	`SQL>`

Oracle	MS SQL
LAST_DAY(*datum*):	EOMONTH(datum)
Findet das Datum des letzten Tags des Monats, der *datum* enthält.	

SQL> SELECT last_day(sysdate) FROM dual;	SQL> SELECT eomonth(sysdatetime());

Oracle	MS SQL
ROUND(*datum*[,'*fmt*']): Gibt das Datum *datum* gerundet auf die im Formatmodell *fmt* angegebene Einheit zurück. Wenn das Formatmodell *fmt* fehlt, wird *Datum* auf das nächste Datum gerundet.	Keine Entsprechung in MS-SQL
SQL> SELECT round(to_date('27.10.2017'), 'year') FROM dual;	SQL>
Liefert: 01.01.18	

Oracle	MS SQL
TRUNC(*datum*[, '*fmt*']): Gibt *Datum* zurück, wobei der Teil mit der Uhrzeit auf die im Formatmodell *fmt* angegebene Einheit abgeschnitten wird. Fehlt das Formatmodell *fmt*, wird *datum* auf den am nächsten liegenden Tag abgeschnitten.	Keine Entsprechung in MS-SQL
SQL> SELECT trunc(to_date('27.10.2017'), 'year') FROM dual;	SQL>
Liefert: 01.01.17	

15.10 Funktion zur Behandlung von NULL-Werten

Schauen wir uns einmal folgendes Problem an:

```
SQL>   SELECT ename,sal,comm,sal+comm
       FROM emp;

ENAME           SAL       COMM      SAL+COMM
---------- ---------- ---------- ----------
KING            5000     (null)    (null)
BLAKE           2850     (null)    (null)
CLARK           2450     (null)    (null)
JONES           2975     (null)    (null)
MARTIN          1250     1400      2650
ALLEN           1600     300       1900
TURNER          1500     0         1500
JAMES           950      (null)    (null)
WARD            1250     500       1750
FORD            3000     (null)    (null)
SMITH           800      (null)    (null)
SCOTT           3000     (null)    (null)
ADAMS           1100     (null)    (null)
MILLER          1300     (null)    (null)
```

Wie Sie sicher schon wissen, kann mit NULL-Werten nicht gerechnet werden, was bedeutet, dass im Falle eines NULL-Wertes in einer Berechnung das ganze Ergebnis NULL wird.

Nun muss es aber eine Möglichkeit geben, die Summe aus sal und comm zu berechnen und dafür gibt es folgende Funktionen.

Oracle	MS SQL
SQL> SELECT ename,sal,comm,sal+NVL(comm,0) FROM emp;	SQL> SELECT ename,sal,comm,sal+isNull(comm,0) FROM emp;

```
ENAME           SAL       COMM      SAL+NVL(COMM,0)
---------- ---------- ---------- ---------------
KING            5000     (null)    5000
BLAKE           2850     (null)    2850
CLARK           2450     (null)    2450
JONES           2975     (null)    2975
MARTIN          1250     1400      2650
ALLEN           1600     300       1900
TURNER          1500     0         1500
JAMES           950      (null)    950
WARD            1250     500       1750
FORD            3000     (null)    3000
```

SMITH	800	(null)	800
SCOTT	3000	(null)	3000
ADAMS	1100	(null)	1100
MILLER	1300	(null)	1300

Wie funktioniert diese Umwandlung nun?
Schauen wir uns das Beispiel einmal näher an.

NVL(comm, 0) bzw. isNull(comm. 0) heißt übersetzt nichts anderes als: Falls in der Spalte comm für Datensatz ein Wert gefunden wird, so wird dieser verwendet, siehe unsere 4 SALESMAN.

Sollte sich aber im Datenfeld comm eines Datensatzes ein NULL-Wert befinden, so wird statt des NULL-Wertes der Wert genommen, der als zweites Argument in der Klammer steht, in unserem Beispiel also die 0.

Sie können diese Funktion für Zeichenketten, Datums- und Zahlenwerte verwenden. Bedingung ist aber, dass die Datentypen der Spalte und des Ersatzwertes übereinstimmen.

Zum gleichen Ergebnis führt auf allen Server die Standard-SQL-Funktion

COALESCE(comm, 0)

15.11 Die Funktion CASE

Die Funktion **CASE** ermöglicht Ausgaben, die in klassischen Programmiersprachen mit IF-THEN-ELSE oder SWITCH-CASE Anweisungen gelöst würden.

```
SQL>   select ename, sal,
       case job
       when 'SALESMAN' then sal*1.2
       when 'CLERK'    then sal*1.3
       when 'MANAGER'  then sal*1.05
       else sal
       end as "NEUGEHALT"
       from emp;

ENAME             SAL  NEUGEHALT
---------- ---------- ----------
KING             5000       5000
BLAKE            2850     2992,5
CLARK            2450     2572,5
JONES            3000       3150
MARTIN           1250       1500
ALLEN            1600       1920
TURNER           1500       1800
JAMES            1250       1625
WARD             1250       1500
FORD             3000       3000
SMITH             800       1040
SCOTT            3000       3000
ADAMS            1100       1430
MILLER           1300       1690

14 Zeilen ausgewählt.
```

Die Funktion CASE beginnt mit dem Schlüsselwort CASE. In diesem Fall hier steht nach dem Schlüsselwort die Spalte, in der nach einer Bedingung gesucht werden soll (job). Die einzelnen Bedingungen werden jeweils mit WHEN begonnen. Hinter THEN steht dann die Aktion, die ausgeführt werden soll, wenn die Bedingung zutrifft. ELSE bietet die Möglichkeit, eine Aktion zu definieren für alle Fälle, die nicht zu einer der genannten Bedingungen passen. Die ELSE-Bedingung ist optional. Wenn sie nicht geschrieben wird, bekommen die Zeilen, die keiner Bedingung entsprechen, in der Ausgabe einen NULL-Wert.

```
SQL>   select ename, sal,
       case job
       when 'SALESMAN' then sal*1.2
       when 'CLERK'    then sal*1.3
       when 'MANAGER'  then sal*1.05
       end as "NEUGEHALT"
       from emp;

ENAME             SAL  NEUGEHALT
---------- ---------- ----------
KING             5000   (null)
BLAKE            2850     2992,5
CLARK            2450     2572,5
JONES            3000       3150
MARTIN           1250       1500
ALLEN            1600       1920
TURNER           1500       1800
JAMES            1250       1625
WARD             1250       1500
FORD             3000   (null)
SMITH             800       1040
SCOTT            3000   (null)
ADAMS            1100       1430
MILLER           1300       1690

14 Zeilen ausgewählt.
```

Oracle bietet noch eine alternative Funktion mit verkürzter Schreibweise. Diese steht allerdings in MS_SQL nicht zur Verfügung. Die Methode CASE ist aber auch die schnellere Variante

```
SQL>   select ename, sal,
       DECODE(job
       'SALESMAN', sal*1.2,
       'CLERK', sal*1.3,
       'MANAGER', sal*1.05,
       sal) "NEUGEHALT"
       from emp;
```

Übungen

1. Zeigen Sie empno, ename, sal und das um 16% erhöhte Gehalt als Ganzzahl aus der Tabelle emp an.

24. Ändern Sie die Abfrage, indem Sie eine zusätzliche Spalte einfügen, die die Gehaltserhöhung anzeigt.

25. Geben Sie die Namen aller Mitarbeiter an zusammen mit der Anzahl der Monate, die die Mitarbeiter schon in der Firma beschäftigt sind. Zeigen Sie nur vollendete Monate an.

26. Zeigen Sie die Namen aller Mitarbeiter an zusammen mit ihrem Gehalt. Geben Sie als Spaltenbreite 15 an und füllen Sie die Lücke mit * rechts auf.

27. Erstellen Sie eine Abfrage, die den Namen mit Grossbuchstaben zu Beginn und folgenden Kleinbuchstaben ausgibt. Ermitteln Sie die Länge des jeweiligen Namens.

28. Einschränkung: Bitte nur für alle Namen, die mit J, A oder M beginnen.

Notizen

Kapitel 16 Konvertierung

In diesem Kapitel erfahren Sie Folgendes:

- Sie können die Verwendung von Konvertierungsfunktionen beschreiben.

16.1 Was bedeutet Datenkonvertierung?

Bei der Erstellung von Tabellen haben Sie für jede Spalte einen Datentypen festgelegt.

In bestimmten Fällen gestattet die Server andere als die erwarteten Datentypen. Dies ist dann zulässig, wenn die Server die Daten automatisch in den erwarteten Datentyp konvertieren können. Die Konvertierung kann *implizit* von den Servern oder *explizit* vom Benutzer durchgeführt werden.

Implizite Datentyp-Konvertierung

Bei Zuweisungen können die Server folgende Konvertierungen automatisch durchführen:

- VARCHAR2 (Zeichenkette variabler Länge) oder CHAR (Zeichenkette fester Länge) in NUMBER (Zahl) (nur Oracle)
- VARCHAR2 oder CHAR in DATE (Datum) (nur Oracle)
- NUMBER in VARCHAR2
- DATE in VARCHAR2

Die Zuweisung gelingt, wenn die Server den Datentyp des in der Zuweisung verwendeten Werts in den Datentyp des Ziels der Zuweisung konvertieren kann, wenn also z. B. eine Zahl in einem Feld mit Datentyp Varchar2 verarbeitet werden soll.

Bei Ausdrucksauswertungen kann der Server folgende Konvertierungen automatisch:

- VARCHAR2 oder CHAR in NUMBER
- VARCHAR2 oder CHAR in DATE

Wenn eine Datentyp-Konvertierung an einer Stelle benötigt wird, die nicht von der Regel für Zuweisungskonvertierungen abgedeckt wird, verwenden die Server normalerweise die Regel für Ausdrücke.

Hinweis: Konvertierungen CHAR in NUMBER gelingen nur, wenn die Zeichenkette eine gültige Zahl darstellt. Die Konvertierungen CHAR in DATE gelingen nur, wenn die Zeichenkette das Standardformat DD-MM-YY hat.

Explizite Datentyp-Konvertierungen werden mit Konvertierungsfunktionen durchgeführt. Konvertierungsfunktionen konvertieren einen Wert von einem Datentyp in einen anderen. In der Regel hat die Funktion das Format *datatype* TO *datatype*. Der erste Datentyp ist der Eingabedatentyp; der letzte Datentyp ist der Ausgabedatentyp.

Hinweis: Auch wenn die implizite Datentyp-Konvertierung zur Verfügung steht, empfehlen wir Ihnen die Verwendung der expliziten Konvertierung, um die Zuverlässigkeit Ihrer SQL-Anweisungen sicherzustellen.

16.2 Was für Konvertierungsfunktionen gibt es?

Es stehen die folgenden Konvertierungsfunktionen zur Verfügung:

- TO_CHAR (Oracle)
- TO_NUMBER (Oracle)
- TO_DATE (Oracle)
- CAST (Oracle und MS-SQL)
- CONVERT (MS-SQL)
- FORMAT (MS-SQL)
- PARSE (MS SQL)

16.3 Wie arbeite ich mit TO_CHAR?

`TO_CHAR(number/date,[fmt])`

Wir übergeben also an die Funktion TO_CHAR einen Wert mit dem Datentyp *number* oder *date* und wandeln diesen Datentyp für die Ausgabe in eine Zeichenkette um.

[fmt] gibt an, wie die Ausgabe aussehen soll.

Wenn wir also den Einstellungsmonat für einen Mitarbeiter angezeigt haben möchten, uns aber der Tag nicht interessiert, müssen wir dies Oracle sagen, weil wir standardmäßig das Datum in der Form 22.10.02 angezeigt bekommen. Nun ist in unserem Beispiel aber die Form 10.02 gefragt. Also teilen wir dies Oracle mit, indem wir mit der Funktion TO_CHAR arbeiten.

```
SQL>   SELECT ename, TO_CHAR(hiredate,'MM.YY')
       FROM emp
       WHERE ename = 'KING';


ENAME      TO_CH
---------- -----
KING       11.81
```

Einige Beispiele für gültige Datumsformatelemente:

Oracle		MS SQL	
YYYY	Jahr 4-stellig	y	Jahr von 0 bis 99
YEAR	Jahr ausgeschrieben	yy	Jahr von 00 bis 99
		yyyy	Jahr vierstellig
MM	Monat 2-stellig	MM	Monat 2-stellig
Month	Monat ausgeschrieben	MMMM	Monat ausgeschrieben
Mon	Monatsabkürzung auf die ersten drei Buchstaben	MMM	Monatsabkürzung auf die ersten drei Buchstaben
Day	Tag ausgeschrieben	dddd	Tag ausgeschrieben
DD	Tag zweistellig	dd	Tag zweistellig
DDD	Tag des Jahres	ddd	Tag abgekürzt
D	Tag der Woche		
HH12	Stunde (0-12)	hh	Stunde (0-12)
HH24	Stunde (0-23)	HH	Stunde (0-23)
MI	Minute	mm	Minuten zweistellig
SS	Sekunde	ss	Sekunden zweistellig
"of the"	die Zeichenkette erscheint in der Ausgabe (DD "of" Month -> 12 of Oktober)		
TH	Ordinalzahl (DDTH -> 4^{TH})		
SP	für ausgeschriebene Zahl (DDSP -> FOUR)		
fm	fill Mode entfernt nachfolgende Leerzeichen aus der Spalte		
WW	Kalenderwoche		

TO_CHAR-Funktion mit numerischen Formaten

Wenn Sie mit numerischen Werten in Form von Zeichenketten arbeiten, sollten Sie diese Zahlen mit der TO_CHAR-Funktion in Zeichen konvertieren. Dabei wird ein Wert des Datentyps NUMBER in den Datentyp VARCHAR2 übertragen. Diese Technik ist besonders bei Verkettungen hilfreich.

Die gebräuchlichsten numerischen Formate:

Oracle		MS SQL	
9	stellt eine Zahl dar	#	stellt eine Zahl dar
0	zeigt führende Nullen an	0	zeigt führende Nullen an
$	zeigt führendes Dollarzeichen an		
L	zeigt führendes lokales Währungszeichen an	C oder c	zeigt führendes lokales Währungszeichen an
. oder D	druckt Dezimalpunkt	.	druckt Dezimalpunkt
, oder G	druckt einen Tausenderindikator	,	druckt einen Tausenderindikator

Ein Beispiel

Oracle	MS SQL
SQL> SELECT TO_CHAR(sal,'L99,999.99') FROM emp WHERE ename ='KING'; TO_CHAR(SAL ---------- $5,000.00	SQL> SELECT format(sal,'C', 'de-DE') FROM emp WHERE ename ='KING'; 5.000,00 €

16.4 Wie arbeite ich mit TO_DATE, TO_NUMBER bzw. PARSE?

Funktionen TO_NUMBER und TO_DATE
Wenn Sie eine Zeichenkette in eine Zahl oder ein Datum konvertieren möchten, können Sie dazu die Funktionen TO_NUMBER oder TO_DATE verwenden. Das ausgewählte Formatmodell basiert auf den bereits gezeigten Formatelementen.

MS-SQL stellt hierfür die Funktion PARSE zur Verfügung, mit der sich sowohl numerische als auch Datumsdatentypen in Zeichenketten umwandeln lassen.

Beispiel:
Zeigen Sie Namen und Einstellungsdatum für alle Mitarbeiter an, die am 22. Februar 1981 in die Firma eingetreten sind.

Oracle	MS SQL
SQL> SELECT ename, hiredate FROM emp WHERE hiredate = TO_DATE('22 FEBRUAR 1981','DD MONTH YYYY'); ENAME HIREDATE ---------- -------- WARD 22.02.81	SQL> SELECT ename, hiredate FROM emp WHERE hiredate = parse('22 FEBRUAR 1981' as datetime2); ENAME HIREDATE ---------- -------- WARD 1981-02-22

16.5 Kann man Funktionen verschachteln?

Sie können Single-Row-Funktionen in beliebiger Tiefe verschachteln. Die Auswertung erfolgt dabei von innen nach außen.

Oracle	MS SQL
SQL> SELECT ename, NVL(TO_CHAR(mgr),'No Manager') FROM emp WHERE mgr IS NULL;	SQL> SELECT ename, isNull(cast(mgr as varchar),'No Manager') FROM emp WHERE mgr IS NULL;
ENAME NVL(TO_CHAR(MGR),'NOMANAGER') ---------- KING No Manager	ENAME (kein Spaltenname) ---------- KING No Manager

In diesem Beispiel wird zuerst die Spalte mgr in eine Zeichenkette umgewandelt und danach die Funktion NVL bzw. isNull angewandt.

Übungen

1. Geben Sie die Systemzeit mit Datum und Uhrzeit im Format Stunden:Minuten:Sekunden Tag.Monat.Jahr aus. Dabei alle Angaben zweistellig, das Jahr vierstellig (denken Sie bei Oracle an die Tabelle dual).

29. Geben Sie ename und comm aus der Tabelle emp aus. Falls ein Mitarbeiter keine Provision erhält, soll ’keine Provision’ als Ausgabe erscheinen.

30. Zeigen Sie Namen, Hiredate, und das Datum 6 Monate nach dem Einstellungsdatum an. Beschriften sie die letzte Spalte mit Vorschau. Anzeige des Datums in der letzten Spalte bitte in folgender Form: Montag, the Twenty-Fourth of Mai, 1982.

	ENAME	HIREDATE	VORSCHAU
1	KING	17.11.81	Montag, the Twenty-Fourth of Mai, 1982
2	BLAKE	01.05.81	Montag, the Second of November, 1981
3	CLARK	09.06.81	Montag, the Fourteenth of Dezember, 1981
4	JONES	02.04.81	Montag, the Fifth of Oktober, 1981
5	MARTIN	02.09.81	Montag, the Eighth of März, 1982
6	ALLEN	20.02.81	Montag, the Twenty-Fourth of August, 1981
7	TURNER	08.09.81	Montag, the Fifteenth of März, 1982
8	JAMES	03.12.81	Montag, the Seventh of Juni, 1982
9	WARD	22.02.81	Montag, the Twenty-Fourth of August, 1981
10	FORD	03.12.81	Montag, the Seventh of Juni, 1982
11	SMITH	17.12.80	Montag, the Twenty-Second of Juni, 1981
12	SCOTT	09.12.82	Montag, the Thirteenth of Juni, 1983
13	ADAMS	12.01.83	Montag, the Eighteenth of Juli, 1983
14	MILLER	23.01.82	Montag, the Twenty-Sixth of Juli, 1982

31. Zeigen Sie Namen, Hiredate und Wochentag des Arbeitsbeginns jedes Mitarbeiters an. Überschrift Tag für diese Spalte. Sortieren Sie bitte nach dem Wochentag, beginnend bei Montag.

	ENAME	HIREDATE	Tag
1	KING	17.11.81	DIENSTAG
2	TURNER	08.09.81	DIENSTAG
3	CLARK	09.06.81	DIENSTAG
4	ADAMS	12.01.83	MITTWOCH
5	MARTIN	02.09.81	MITTWOCH
6	SMITH	17.12.80	MITTWOCH
7	JONES	02.04.81	DONNERSTAG
8	JAMES	03.12.81	DONNERSTAG
9	SCOTT	09.12.82	DONNERSTAG
10	FORD	03.12.81	DONNERSTAG
11	ALLEN	20.02.81	FREITAG
12	BLAKE	01.05.81	FREITAG
13	MILLER	23.01.82	SAMSTAG
14	WARD	22.02.81	SONNTAG

Kapitel 17 JOIN

In diesem Kapitel erfahren Sie Folgendes:

- Sie können SELECT-Anweisungen schreiben, um mit Equijoins und Non-Equijoins auf Daten aus mehreren Tabellen zuzugreifen.
- Sie können Daten, die normalerweise keiner JOIN Bedingung entsprechen, mit Outer Joins anzeigen.
- Sie können eine Tabelle mit sich selbst verknüpfen.

17.1 Was ist ein JOIN?

Wenn Daten aus mehreren Tabellen in der Datenbank benötigt werden, wird eine *JOIN*-Bedingung verwendet. Anhand der gemeinsamen Werte in den entsprechenden Spalten, d. h. häufig in Primärschlüssel- und Fremdschlüsselspalten, können Zeilen einer Tabelle mit Zeilen einer anderen Tabelle verknüpft werden.

Um Daten aus zwei oder mehreren Tabellen anzuzeigen, die zueinander in Beziehung stehen, schreiben Sie eine einfache JOIN-Bedingung in die WHERE-Klausel. Verwenden Sie folgende Syntax:

```
…
where table1.column1 = table2.column2 ;
```

Richtlinien

- Wenn Sie eine SELECT-Anweisung schreiben, die Tabellen verknüpft, stellen Sie dem Spaltennamen aus Gründen der Eindeutigkeit und zur Verbesserung des Datenbankzugriffs den Tabellennamen voran.
- Wenn derselbe Spaltenname in mehreren Tabellen erscheint, stellen Sie dem Spaltennamen den Tabellennamen voran.
- Zur Verknüpfung von *n* Tabellen benötigen Sie mindestens (*n-1*) JOIN-Bedingungen, d. h. Sie benötigen mindestens drei JOIN-Bedingungen, um vier Tabellen zu verknüpfen. Wenn Ihre Tabelle einen verketteten Primärschlüssel enthält, kann diese Regel ggf. nicht angewendet werden. In diesem Fall sind zur Identifizierung der einzelnen Zeilen mehrere Spalten erforderlich.

Sie sollten niemals diese JOIN-Bedingung vergessen. Falls Ihnen das passieren sollte, werden Sie es anhand der ausgegeben Zeilen sehr schnell merken. Eine fehlende JOIN-Bedingung führt nämlich zu einer Verknüpfung jeder Zeile der einen Tabelle mit jeder Zeile der anderen Tabelle. Dieses Ergebnis nennt man CROSS-JOIN oder Kreuzprodukt.

17.2 Was ist ein CROSS-JOIN?

Wie genau schreibt man nun einen CROSS-JOIN?

Oracle-Versionen bis einschließlich 8.i:

```
SQL>   SELECT emp.ename, dept.dname, dept.deptno
       FROM emp, dept;
```

Es wird also einfach die Where-Klausel weggelassen. So eine Abfrage kann recht lange dauern, weil, wie oben schon erwähnt, alle Zeilen der ersten Tabelle mit allen Zeilen der zweiten Tabelle verknüpft werden. Falls beide Tabellen so etwa 100.000 Zeilen enthalten, hat der Datenbankserver reichlich zu tun!

Oracle-Versionen ab 9.i können eine abweichende Schreibweise für Joins verwenden (Hier erstmal nur der CROSS-JOIN, alle anderen Join-Typen finden Sie später im Kapitel):

```
SQL>   SELECT emp.ename, dept.dname, dept.deptno
       FROM emp CROSS JOIN dept;
```

Sie müssen bei dieser Syntax also extra schreiben, dass Sie einen CROSS-JOIN absenden wollen.

17.3 Welche JOIN-Typen gibt es?

Wir werden uns mit 4 Typen von JOINS beschäftigen, die da lauten:

- EQUIJOIN
- (Unterform: NATURAL JOIN)
- NON-EQUIJOIN
- OUTER JOIN
- SELF JOIN

17.4 Was ist ein EQUIJOIN?

Wenn Sie den Namen der Abteilung eines Mitarbeiters ermitteln möchten, vergleichen Sie den Wert in der Spalte DEPTNO der Tabelle EMP mit dem Spaltenwert DEPTNO in der Tabelle DEPT. Bei der Beziehung zwischen den Tabellen EMP und DEPT handelt es sich um einen *Equijoin*, das heißt, die Werte in der Spalte DEPTNO beider Tabellen müssen identisch sein. Zu diesem Jointyp gehören häufig Primär- und Fremdschlüsselpaare.

Hinweis: Equijoins werden auch als einfache oder innere Joins bezeichnet.

```
SQL>   SELECT emp.ename, dept.dname, dept.deptno
       FROM emp, dept
       WHERE emp.deptno=dept.deptno;
```

Im Beispiel oben gilt:

Die SELECT-Klausel gibt die Spaltennamen an, die abgerufen werden sollen:

- die Spalte Mitarbeitername (ename) aus der Tabelle EMP
- die Spalten Abteilungsnummer (deptno) und Abteilungsname (dname) aus der Tabelle DEPT

Die FROM-Klausel gibt die beiden Tabellen an, auf die die Datenbank zugreifen muss:

- Tabelle EMP

- Tabelle DEPT

Die WHERE-Klausel gibt an, wie die Tabellen verknüpft werden:

- EMP.DEPTNO=DEPT.DEPTNO

Schreibweise, die Sie ab Oracle 9.i und im MS-SQL-Server auch verwenden können:

```
SQL>   SELECT emp.ename, dept.dname, dept.deptno
       FROM emp INNER JOIN dept
       ON emp.deptno=dept.deptno;
```

Vergleich der beiden Möglichkeiten:

- Die Select-Liste bleibt unverändert.
- In der From-Klausel trennen Sie die Tabellen nicht durch ein Komma, sondern Sie verwenden den Join-Operator. (tabelle 1 INNER JOIN tabelle 2)
- Sie verwenden zur Angabe der Verknüpfungsbedingung nicht die Where-Klausel. Sie benutzen die ON-Syntax. Der Inhalt ist allerdings derselbe, den Sie vorher in der Where-Klausel geschrieben haben.

Sie müssen die Spaltennamen in der WHERE-Klausel mit dem Tabellennamen spezifizieren, um Zweideutigkeiten zu vermeiden. Ohne das Tabellenpräfix könnte die Spalte DEPTNO sowohl aus der Tabelle DEPT als auch aus der Tabelle EMP stammen. Sie müssen das Tabellenpräfix hinzufügen, um die Abfrage auszuführen.

Wenn die beiden Tabellen keine identischen Spalten haben, müssen die Spalten nicht genauer spezifiziert werden. Die Verwendung von Tabellenpräfixen erhöht jedoch immer die Leistung, da Oracle Server genaue Angaben zum Standort der Spalten erhält.

Die genaue Spezifizierung von Spaltennamen ist auch für Spalten notwendig, die in anderen Klauseln (der SELECT- oder ORDER BY-Klausel) zweideutig sein könnten. Falls wir nun für KING die Spalten empno, ename, deptno und loc ausgegeben haben möchten, müssen wir folgendes tun:

- erst selektieren:

```
SQL>   SELECT emp.empno, emp.ename, emp.deptno, dept.loc
```

- dann die Tabellen angeben:

```
SQL>   FROM emp, dept
```

- dann verknüpfen:

```
SQL>   WHERE emp.deptno=dept.deptno
```

- als letztes nun noch die Bedingung anfügen

```
SQL>   AND emp.ename = 'KING';
```

oder:

- erst selektieren:

```
SQL>   SELECT emp.empno, emp.ename, emp.deptno, dept.loc
```

- dann die Tabellen und den JOIN-Typ angeben:

```
SQL>   FROM emp INNER JOIN dept
```

- dann verknüpfen:

```
SQL>   ON emp.deptno=dept.deptno
```

- als letztes nun noch die Bedingung anfügen, also eine ganz normale Where-Klausel

```
SQL>   WHERE emp.ename = 'KING';
```

Wie Sie sehen, haben wir etliche Male emp. bzw. dept. geschrieben. Stellen Sie sich einmal vor, die Tabelle emp hieße MITARBEITERDATEN. Wer will dieses lange Wort denn als Tabellenpräfix schreiben. Muss auch keiner, denn dafür gibt es natürlich eine Lösung, nämlich den Tabellenalias.

Wir vergeben der Tabelle emp den Alias e und der Tabelle dept den Alias d. Warum funktioniert das aber überhaupt?

Nun, der Server wertet als erstes die FROM-Klausel aus und damit kann in allen folgenden Klauseln mit diesen Aliasnamen gearbeitet werden.

Hier unser Beispiel von eben in verkürzter Form:

```
SQL>   SELECT e.empno, e.ename, e.deptno, d.loc
       FROM emp e, dept d
       WHERE e.deptno=d.deptno
       AND e.ename = 'KING';
```

oder:

```
SQL>   SELECT e.empno, e.ename, e.deptno, d.loc
       FROM emp e inner join dept d
       ON e.deptno=d.deptno
       WHERE  e.ename = 'KING';
```

Richtlinien

- Tabellen-Aliasnamen können bis zu 30 Zeichen lang sein, aber je kürzer sie sind, desto besser.
- Wenn ein Tabellen-Alias in der FROM-Klausel für einen bestimmten Tabellennamen verwendet wird, muss dieser Alias auch in der SELECT-Anweisung den Tabellennamen ersetzen.
- Tabellen-Aliasnamen sollten sinnvoll gewählt werden.
- Der Tabellen-Alias gilt nur für die aktuelle SELECT-Anweisung.

Der NATURAL JOIN, eine Oracle-Sonderform des EQUI-JOIN(INNER JOIN)

Eine Kurzform des EQUI JOIN(INNER JOIN) ist der NATURAL JOIN. Er sieht folgendermaßen aus:

```
SQL>   select ename, dname
       from emp NATURAL JOIN dept;

ENAME      DNAME
---------- --------------
KING       ACCOUNTING
BLAKE      SALES
```

```
CLARK      ACCOUNTING
JONES      RESEARCH
MARTIN     SALES
ALLEN      SALES
TURNER     SALES
JAMES      SALES
WARD       SALES
FORD       RESEARCH
SMITH      RESEARCH
SCOTT      RESEARCH
ADAMS      RESEARCH
MILLER     ACCOUNTING

14 Zeilen ausgewählt.
```

Die ON-Bedingung entfällt, wie Sie sehen. Die Verknüpfungsbedingung ermittelt Oracle beim NATURAL JOIN automatisch. Dafür ist es aber zwingend erforderlich, dass die Verknüpfungsspalten exakt die gleiche Bezeichnung aufweisen! Einen Tabellenpräfix wie z.B. **emp.**ename dürfen sie keinesfalls verwenden. Wenn Sie es trotzdem versuchen, bekommen Sie von Oracle eine Fehlermeldung.

Diese Form wird von MS-SQL nicht unterstützt.

17.5 Was ist ein NON-EQUIJOIN?

Die Beziehung zwischen der Tabelle EMP und der Tabelle SALGRADE ist ein Non-Equijoin, d. h. dass keine Spalte der Tabelle EMP direkt einer Spalte der Tabelle SALGRADE entspricht. Die Beziehung zwischen den beiden Tabellen ist so, dass die Spalte SAL der Tabelle EMP zwischen den Spalten LOSAL und HISAL der Tabelle SALGRADE anzuordnen ist. Diese Beziehung kann nicht mit dem Gleichheitsoperator dargestellt werden.

Schauen wir uns das Problem einmal näher an.

Wir wollen einmal die Namen und die Gehaltsstufen aller Mitarbeiter aus der Abteilung 10 uns anzeigen lassen.

Da die Werte e.sal zwischen den Werten von s.losal und s.hisal liegen, bietet sich eine BETWEEN – AND-Konstruktion an.

```
SQL>   SELECT e.ename, s.grade
       FROM emp e, salgrade s
       WHERE e.sal between s.losal and s.hisal
       and e.deptno=10;
```

oder:

```
SQL>   SELECT e.ename, s.grade
       FROM emp e inner join salgrade s
       ON e.sal between s.losal and s.hisal
       WHERE e.deptno=10;

ENAME         GRADE
---------- ----------
MILLER        2
CLARK         4
KING          5
```

17.6 Was ist ein OUTER JOIN?

Wenn ein Datensatz nicht der Verknüpfungsbedingung entspricht, wird sie nicht ausgegeben, d. h. der Datensatz mit der deptno 40 aus der Tabelle dept fällt heraus, wenn wir einen normalen EQUIJOIN schreiben.

```
SQL>   SELECT e.ename, d.dname, d.deptno
       FROM emp e, dept d
       WHERE e.deptno=d.deptno;

SQL>   SELECT e.ename, d.dname, d.deptno
       FROM emp e inner join dept d
       ON e.deptno=d.deptno;

ENAME         DNAME          DEPTNO
---------- --------------     ----------
KING          ACCOUNTING      10
BLAKE         SALES           30
CLARK         ACCOUNTING      10
JONES         RESEARCH        20
MARTIN        SALES           30
ALLEN         SALES           30
TURNER        SALES           30
JAMES         SALES           30
WARD          SALES           30
FORD          RESEARCH        20
SMITH         RESEARCH        20
SCOTT         RESEARCH        20
ADAMS         RESEARCH        20
MILLER        ACCOUNTING      10
```

Die fehlende(n) Zeile(n) kann/können zurückgegeben werden, wenn in der JOIN-Bedingung ein *Outer Join*-Operator verwendet wird. Der Operator ist ein in Klammern eingeschlossenes Plus-

zeichen (+) und wird *auf der „Seite" des Join mit den fehlenden Informationen plaziert*. Dieser Operator erzeugt eine oder mehrere NULL-Zeilen, mit der eine oder mehrere Zeilen der Tabelle verknüpft werden können, die die Informationen enthält.

Folgende Syntax ist zu verwenden:

- *table1.column = table2.column* ist die Bedingung, die die Tabellen verknüpft (zueinander in Beziehung setzt).
- (+) ist das Outer Join-Symbol; es kann auf jeder Seite der WHERE-Klauselbedingung plaziert werden, aber nicht auf beiden Seiten.Plazieren Sie das Symbol nach dem Namen der Spalte der Tabelle ohne die übereinstimmende Zeile.

Erneut unser Beispiel, aber nun als OUTER JOIN (nur Oracle):

```
SQL>   select e.ename, d.dname, d.deptno
       FROM emp e, dept d
       * WHERE e.deptno(+)=d.deptno

SQL  > /

ENAME           DNAME             DEPTNO
---------- --------------        ----------
KING            ACCOUNTING        10
CLARK           ACCOUNTING        10
MILLER          ACCOUNTING        10
JONES           RESEARCH          20
SCOTT           RESEARCH          20
ADAMS           RESEARCH          20
SMITH           RESEARCH          20
FORD            RESEARCH          20
BLAKE           SALES             30
MARTIN          SALES             30
ALLEN           SALES             30
TURNER          SALES             30
JAMES           SALES             30
WARD            SALES             30
                OPERATIONS        40
```

Es ist also eine zusätzliche Zeile in der Ausgabe vorhanden mit den Informationen über das Department 40.

Diese Version funktioniert auf beiden Server, da es sich um einen SQL-Standard handelt:

```
SQL>   SELECT e.ename, d.dname, d.deptno
       FROM emp e RIGHT [OUTER] JOIN dept d
       ON e.deptno=d.deptno;
```

Bei dieser Abfrage werden neben den übereinstimmenden Werten, die in beiden Tabellen vorhanden sind, noch die Werte ausgegeben, die nur in der rechts vom join Schlüsselwort stehenden Tabelle vorhanden sind, in diesem Beispiel die deptno 40. Falls Sie in der Tabelle emp nach zusätzlichen Einträgen suchen möchten, nehmen Sie einen **LEFT OUTER JOIN.** Falls Sie in beiden Tabellen suchen müssen, dann schreiben sie **FULL OUTER JOIN**.

17.7 Was ist ein SELF JOIN?

Es kann vorkommen, dass Sie eine Tabelle mit sich selbst verknüpfen müssen. Um den Namen des Managers jedes Mitarbeiters zu finden, müssen Sie die Tabelle EMP mit sich selbst verknüpfen. Um z. B. den Namen für den Manager von Blake zu finden, müssen Sie wie folgt vorgehen:

- Suchen Sie Blake in der Tabelle EMP anhand der Spalte ENAME.
- Suchen Sie die Managernummer für Blake in der Spalte MGR. Blake ist die Managernummer 7839 zugeordnet.
- Suchen Sie den Namen des Managers mit der EMPNO 7839 in der Spalte ENAME. Die Mitarbeiternummer (EMPNO) von King ist 7839. Daher ist King Blakes Manager.
- Bei diesem Verfahren sehen Sie zweimal in der Tabelle emp nach. Beim ersten Mal suchen Sie nach Blake in der Spalte ENAME und einem Wert von 7839 in der Spalte MGR. Beim zweiten Mal suchen Sie in der Spalte EMPNO nach 7839 und in der Spalte ENAME nach King.

Wir müssen also in der Abfrageformulierung dafür sorgen, dass es die Tabelle emp zweimal gibt. Also vergeben wir einfach zwei verschiedene Aliase für emp.

```
SQL>   SELECT a.ename Angestellter, m.ename manager
       FROM emp a, emp m
       WHERE a.mgr = m.empno;
```

oder:

```
SQL>   SELECT a.ename Angestellter, m.ename manager
       FROM emp a inner join emp m
       ON a.mgr = m.empno;

ANGESTELLT      MANAGER
----------      ----------
BLAKE           KING
CLARK           KING
JONES           KING
MARTIN          BLAKE
ALLEN           BLAKE
TURNER          BLAKE
JAMES           BLAKE
WARD            BLAKE
FORD            JONES
SMITH           FORD
SCOTT           JONES
ADAMS           SCOTT
MILLER          CLARK
```

17.8 Wie verknüpfe ich mehrere Tabellen miteinander?

Angenommen, wir möchten den Namen, den Standort und die Gehaltsgruppe eines bestimmten Mitarbeiters ermitteln. Das bedeutet, dass wir Daten aus drei Tabellen holen müssen, nämlich den Namen aus `emp`, den Standort aus `dept` und die Gehaltsgruppe aus `salgrade`.

Schauen wir uns an, wie das funktioniert:

```
SQL>   SELECT e.ename, d.loc, s.grade
       FROM emp e, dept d, salgrade s
       WHERE e.deptno=d.deptno
       AND e.sal BETWEEN s.losal AND s.hisal
       AND e.ename ='KING';

ENAME      LOC                GRADE
---------- ------------ ----------
KING       NEW YORK               5
```

In der FROM-Klausel werden alle beteiligten Tabellen aufgeführt.

In der WHERE-Klausel werden sie dann wie gewohnt miteinander verknüpft, und zwar nacheinander.

oder:

```
SQL>   SELECT e.ename, d.loc, s.grade
       FROM emp e inner join dept d
       ON e.deptno=d.deptno
       inner join salgrade s
       ON e.sal BETWEEN s.losal AND s.hisal
       WHERE e.ename ='KING';
```

Sie sehen, dass Sie erst zwei Tabellen komplett miteinander verknüpfen müssen, also inklusive ON-Bedingung, danach fügen Sie die nächste mit dem Join-Operator hinzu usw.

Übungen

1. Zeigen Sie alle Berufe aus Abteilung 30 jeweils einmal. Der Standort der Abteilung soll mit ausgegeben werden.

32. Erstellen Sie eine Abfrage, die alle Mitarbeiter mit Namen, Beruf, Abteilungsnummer und Abteilungsnamen anzeigt, die in DALLAS arbeiten.

33. Zeigen Sie den Namen und die Nummer jedes Mitarbeiters zusammen mit Namen und Mitarbeiternummer des dazugehörigen Managers an.

34. Erstellen Sie eine Abfrage, welche Namen, Abteilungsnummer und alle Mitarbeiter anzeigt, die in derselben Abteilung arbeiten wie ein bestimmter Mitarbeiter. Lösung ergibt 28 Zeilen! Sortieren Sie nach der Spaltenreihenfolge.

35. Ändern Sie Aufgabe 3 so, dass KING mit angezeigt wird.

36. Zeigen Sie Namen und hiredate aller Mitarbeiter an, die nach BLAKE eingestellt wurden.

Notizen

Kapitel 18 Multi-Row-Funktionen

In diesem Kapitel erfahren Sie Folgendes:

- Sie können die Verwendung der Gruppenfunktionen beschreiben.
- Sie können Daten mit der GROUP BY-Klausel gruppieren.
- Sie können gruppierte Zeilen mit der HAVING-Klausel ein- oder ausschließen.

18.1 Worin unterscheidet sich eine Multi-Row-Funktion von einer Single Row Funktion?

Während Single-Row-Funktionen pro Zeile ein Ergebnis liefern, wirken sich Multi-Row-Funktionen auf mehrere Zeilen aus, um pro Zeilengruppe ein Ergebnis zu liefern, z.B. das Durchschnittsgehalt der Firma. Hierbei werden alle Gehälter aus der Tabelle emp für die Berechnung herangezogen. Das Ergebnis gilt dann für die komplette Tabelle. Wir bekommen also nur einen Datensatz zurück.

18.2 Welche Multi-Row-Funktionen gibt es?

AVG	Durchschnittswert
MAX	Maximalwert
MIN	Minimalwert
COUNT	Anzahl von Zeilen
SUM	Summe
STDDEV	Standardabweichung (Oracle)
STDEV	Standardabweichung (MS-SQL)
VARIANCE	Abweichung (Oracle)
VAR	Abweichung (MS-SQL)

Für uns spielen die letzten vier Funktionen im Laufe des Kurses keine Rolle.

18.3 Wie sieht die Syntax aus?

```
SQL>   SELECT funktionsname(spalte)
       FROM tabelle;
```

18.4 Wie arbeite ich mit Multi-Row-Funktionen?

Die Funktionen MIN und MAX können bei jedem beliebigen SQL-Datentypen angewendet werden, alle Funktionen können mit numerischen Werten arbeiten.

Schauen wir uns einmal ein Beispiel an:

```
SQL>   SELECT AVG(sal), MAX(sal), MIN(SAL), SUM(SAL)
       FROM emp
       WHERE job like 'SALES%';

AVG(SAL)     MAX(SAL)     MIN(SAL       SUM(SAL)
----------   ----------   ----------    ----------
1400         1600         1250          5600
```

Wir erhalten also als Ergebnis aus einer Berechnung über 14 Zeilen jeweils eine Zeile zurück.

Ein weiteres Beispiel soll zeigen, wie MIN und MAX bei Zeichenketten und Datumswerten funktionieren.

```
SQL>   SELECT MIN(ename), MAX(ename), MIN(hiredate), MAX(hiredate)
       FROM emp;

MIN(ENAME)     MAX(ENAME)      (HIRE         MAX(HIRE
----------     ----------      --------      --------
ADAMS          WARD            17.12.80      12.01.83
```

Bei Zeichenketten ist also der Buchstabe A der kleinste Wert und bei Datumswerten wird die Jahreszahl zur Ermittelung des Ergebnisses herangezogen.

18.5 Was macht die Funktion COUNT?

Die Funktion COUNT hat zwei Ausprägungen:

COUNT(*)
COUNT(*spalte*)

- COUNT(*) liefert die Anzahl der Zeilen einer Tabelle, einschliesslich Spalten mit NULL-Werten und ist die einzige Multi-Row-Funktion, die NULL-Werte berücksichtigt.
- COUNT(*spalte*) liefert die Anzahl der NOT NULL Zeilen der angegebenen Spalte zurück.

```
SELECT COUNT(*)          SELECT COUNT(comm)
FROM emp;                FROM emp;

 COUNT(*)                COUNT(COMM)
 --------------          -----------------
      14                 4
```

18.6 Wie verhalten sich Multi-Row-Funktionen bei NULL-Werten?

Alle Multi-Row-Funktionen bis auf COUNT(*) ignorieren NULL-Werte!

Das bedeutet, dass bei der Berechnung der durchschnittlichen Provision nur 4 Spalten herangezogen werden.

```
SQL>   SELECT AVG(comm)
       FROM emp;

 AVG(COMM)
----------
       550
```

Was müssen wir also tun, wenn wir den Wert basierend auf allen Spalten errechnen wollen?

Richtig, da gab es eine Funktion namens NVL.

Oracle	MS SQL
`SQL>  SELECT AVG(NVL(comm,0))` `       FROM emp`	`SQL>  SELECT AVG(isNull(comm,0))` `       FROM emp`

```
AVG(NVL(COMM,0))
----------------
      157,142857
```

Es ist also möglich, Single-Row-Funktionen mit Multi-Row-Funktionen zu verbinden.

18.7 Wie gruppiere ich Daten?

Bis hierher haben wir Werte basierend auf der ganzen Tabelle ausgegeben.

Schauen wir uns nun an, wie man eine Aufteilung in kleinere Gruppen vornehmen kann.

Ermitteln wir doch einmal das durchschnittliche Gehalt pro Abteilung:

```
SQL>   SELECT deptno, AVG(sal)
       FROM emp
       GROUP BY deptno;

DEPTNO        AVG(SAL)
----------    ----------
10            2916,66667
20            2175
30            1566,66667
```

GROUP BY-Klausel verwenden

Sie können die GROUP BY-Klausel verwenden, um die Zeilen in einer Tabelle in Gruppen zu unterteilen. Sie können dann die Gruppenfunktionen verwenden, um Summeninformationen für jede Gruppe zurückzugeben.

In der Syntax werden durch:

group_by_expression

die Spalten festgelegt, deren Werte die Grundlage zum Gruppieren von Zeilen bilden.

Richtlinien

- Wenn Sie eine Gruppenfunktion in eine SELECT-Klausel einschließen, können Sie nicht zusätzlich Einzelergebnisse auswählen, *außer* die jeweilige Spalte erscheint in der GROUP BY-Klausel. Sie erhalten eine Fehlermeldung, wenn Sie die Spaltenliste nicht einfügen.
- Mit einer WHERE-Klausel können Sie vorab Zeilen ausschließen, ehe Sie diese in Gruppen aufteilen.
- Sie müssen die *Spalten*, die Sie ausgeben wollen, in die GROUP BY-Klausel einfügen.
- Sie können das Spalten-Alias nicht in der GROUP BY-Klausel verwenden (Abarbeitungsreihenfolge --> GROUP BY wird vor dem SELECT vom Server ausgeweitet).
- Standardmäßig werden Zeilen in aufsteigender Reihenfolge der in der GROUP BY-Liste enthaltenen Spalten sortiert. Sie können dies durch die ORDER BY-Klausel überschreiben.

Also, wenn Sie eine Einzelspalte zusätzlich zu einer Gruppenfunktion ausgeben wollen, müssen Sie diese Spalte in der GROUP BY Klausel aufführen.

Dieses gilt aber nicht umgekehrt. Sie können sehr wohl nach einer Spalte gruppieren, die nicht in der SELECT Klausel erwähnt wird.

Schauen wir uns ein weiteres Beispiel an:

```
SQL>   SELECT AVG(sal)
       FROM emp
       WHERE deptno in(10,30)
       GROUP BY deptno;

  AVG(SAL)
----------
2916,66667
1566,66667
```

Wir haben nun in einer WHERE Klausel bestimmt, welche Abteilungen ausgewählt werden sollen und haben nach einer Spalte gruppiert, die in der SELECT Klausel nicht aufgeführt ist.

18.8 Kann nach mehreren Gruppen sortiert werden?

Was müssen wir tun, wenn wir innerhalb der Abteilungen die Summer der Gehälter pro Berufsgruppe ermitteln wollen?

```
SELECT deptno, job, SUM(sal)
FROM emp
GROUP BY deptno, job;
```

```
    DEPTNO       JOB            SUM(SAL)
----------       ---------      ----------
        10       CLERK          1300
        10       MANAGER        2450
        10       PRESIDENT      5000
        20       ANALYST        6000
        20       CLERK          1900
```

```
20      MANAGER      2975
30      CLERK        950
30      MANAGER      2850
30      SALESMAN     5600
```

Sie können Summenergebnisse für Gruppen und Untergruppen zurückgeben, indem Sie in der GROUP BY Klausel mehrere Spalten auflisten. Sie können die Standard-Sortierfolge der Ergebnisse durch die Reihenfolge der Spalten in der GROUP BY-Klausel festlegen. So wird die obige SELECT-Anweisung, welche eine GROUP BY-Klausel enthält, ausgewertet.

Die SELECT-Klausel legt fest, welche Spalte abgerufen werden soll:

- Spalte DEPTNO der Tabelle EMP
- Spalte JOB der Tabelle EMP
- Die Summe aller Gehälter in der Gruppe, die Sie in der GROUP BY-Klausel festgelegt haben
- Die FROM-Klausel legt die Tabellen fest, auf welche die Datenbank zugreifen muss: die Tabelle EMP.
- Die GROUP BY-Klausel legt fest, wie Sie die Zeilen gruppieren müssen:
- Zuerst werden die Zeilen nach Abteilungsnummer gruppiert.
- Danach werden die Zeilen innerhalb der Abteilungsnummer-Gruppen nach Berufsgruppen gruppiert.
- Somit wird die Funktion SUM für alle Funktionen innerhalb der einzelnen Abteilungsnummer-Gruppen auf die Spalte SAL angewendet.

18.9 Was muss getan werden, um Gruppen einschränken zu können?

Wie ist die Vorgehensweise, wenn ich zum Beispiel bei der Ermittlung des Durchschnittsgehaltes pro Abteilung nur diejenigen Abteilungen angezeigt haben möchte, deren Wert oberhalb von 2000 liegt?

Probieren wir es aus:

```
SQL>   SELECT deptno, AVG(sal)
       FROM emp
       WHERE AVG(sal)>2000
       GROUP BY deptno;

 WHERE AVG(sal)>2000
       *
 FEHLER in Zeile 3:
 ORA-00934: Gruppenfunktion ist hier nicht zulässig
```

Für diese Einschränkung darf also die WHERE Klausel nicht verwendet werden.

Hierfür liefert uns SQL eine weitere Klausel, nämlich die HAVING Klausel.

Korrigieren wir das obige Beispiel:

```
SQL>   SELECT deptno, AVG(sal)
       FROM emp
       GROUP BY deptno
       HAVING AVG(sal) >2000;

    DEPTNO      AVG(SAL)
----------      ----------
        10      2916,66667
        20      2175
```

Wir müssen also erst gruppieren, bevor wir dann bestimmen können, welche der Gruppenergebnisse wir angezeigt bekommen wollen.

Die Auswertereihenfolge des Servers ist nämlich WHERE, GROUP BY, HAVING.

Übungen

1. Ermitteln Sie die Anzahl der Mitarbeiter pro Berufsgruppe.

2. Zeigen Sie die Differenz zwischen dem höchsten und dem niedrigsten Gehalt an.

3. Zeigen Sie die Spalten mgr und sal des am schlechtesten bezahlten Mitarbeiters dieses Managers an. Schließen Sie jeden aus, der keine mgr hat. Berücksichtigt werden sollen nur Gruppen mit Werten über 1000.

4. Erstellen Sie eine Abfrage, welche einmal die Gesamtzahl der Mitarbeiter in einer Spalte TOTAL anzeigt, und dann zeigen Sie an, wieviele Mitarbeiter pro Jahr eingestellt wurden. Beschriften Sie hierbei jede Jahresspalte mit der entsprechenden Jahreszahl. Überprüfen Sie vorher, in welchen Jahren Mitarbeiter eingestellt wurden.

5. Zeigen Sie das höchste und niedrigste Gehalt, die Summe aller Gehälter sowie das Durchschnittsgehalt aller Mitarbeiter an. Beschriften Sie die Spalten mit Maximum, Minimum, Sum und Average. Runden Sie Ihre Ergebnisse auf die Dezimalstelle.

6. Bearbeiten Sie die vorherige Aufgabe, um das minimale und maximale, Gehalt sowie die Summe aller Gehälter und das Durchschnittsgehalt für jede Funktion (Job Type) anzuzeigen.

37. Erstellen Sie eine Abfrage, um den Namen der Abteilung (Department), des Standorts (location), die Anzahl der Mitarbeiter und das Durchschnittsgehalt (Average Salary) aller Mitarbeiter in dieser Abteilung anzuzeigen. Beschriften Sie die Spalten mit dname, loc, Number of People und Salary

Notizen

Kapitel 19 Unterabfragen

In diesem Kapitel erfahren Sie Folgendes:

- Sie können die Arten von Problemen beschreiben, die Unterabfragen lösen können.
- Sie können Unterabfragen definieren.
- Sie können Unterabfragetypen auflisten.

19.1 Normale Unterabfrage

Eine Unterabfrage ist eine **verschachtelte SELECT-Abfrage**, die in der Klausel einer anderen SQL – Anweisung eingebettet ist. Die innere **SELECT**-Anweisung wird nur einmal ausgeführt. Die Unterabfrage (innere Abfrage) gibt einen Wert zurück, der dann von der Hauptabfrage (äußere Abfrage) verwendet wird.

Unterabfragen können für folgende **Zwecke** verwendet werden:

- Um Werte für Bedingungen in den Klauseln WHERE, HAVING und START WITH von SELECT-, UPDATE-, und DELETE – Anweisungen zur Verfügung zu stellen,
- um die Zeilensätze zu definieren, die in die Zieltabelle einer INSERT – oder CREATE TABLE – Anweisung eingefügt werden sollen,
- um die Zeilensätze zu definieren, die in eine View oder einen Snapshot in einer CREATE VIEW – oder CREATE SNAPSHOT - Anweisung aufgenommen werden sollen,
- um einen oder mehrere Werte zu definieren, die bestehenden Zeilen in einer UPDATE – Anweisung zugeordnet werden sollen,
- um eine Tabelle zu definieren, auf die eine Abfrage durchgeführt werden soll. Dafür können Sie die Unterabfrage in die FROM – Klausel stellen. Dies kann auch in INSERT -, UPDATE – und DELETE – Anweisungen geschehen.

Die Syntax für eine Unterabfrage sieht wie folgt aus:

```
SQL>   SELECT spalte
       FROM tabelle
       WHERE wert operator( SELECT spalte FROM tabelle);
```

In der Syntax: OPERATOR umfasst einen Vergleichsoperator, wie z.B. **>**, **=** oder **IN**.

Hinweis:
Es gibt zwei Typen von Operatoren:

- Single-Row-Operatoren sind welche, die nur Unterabfragen bearbeiten können, die genau EINE Zeile zurückgeben. Es sind die bekannten mathematischen Operatoren.
- Multi-Row-Operatoren können auch mehrere Rückgabewerte verarbeiten. Sie lauten **ANY, ALL** oder **IN**.

Beispiel für einen Single Row Operator:

```
SQL>   SELECT ename
       FROM emp
       WHERE sal > (SELECT sal FROM emp WHERE ename = 'JONES');
```

Hier wird also erst in der Unterabfrage das Gehalt von JONES ermittelt, in diesem Falle 2975, und dann an die äussere Abfrage als Vergleichswert übergeben. Der Datenbankserver führt also im zweiten Schritt folgende Abfrage aus:

```
SQL>   SELECT ename
       FROM emp
       WHERE sal > 2975;
```

Beispiel für einen Multi Row Operator:

```
SQL>   SELECT ename,sal,job FROM emp
       where sal < ANY(select sal from emp where job = 'CLERK' )
       and job != 'CLERK',
```

Was macht <ANY?
Die Ergebnismenge der Unterabfrage wird ermittelt, der größte Wert wird genommen und mit der äußeren Abfrage verglichen. >ANY entsprechend umgekehrt

=ANY entspricht IN.

Das gleiche Ergebnis liefert demzufolge auch diese Abfrage:

```
SQL>   SELECT ename,sal,job FROM emp
       where sal < (select max(sal) from emp where job = 'CLERK')
       and job != 'CLERK';
```

Was macht >ALL??
Ergebnismenge der Unterabfrage wird ermittelt, größter Wert wird genommen und mit äußerer Abfrage verglichen. <ALL entsprechend umgekehrt.

Ein Beispiel:

```
SQL>   select ename, sal
       from emp
       where sal > ALL
       (SELECT avg(sal) from emp group by deptno);
```

Hier sollen alle ausgegeben werden, die mehr als das höchste Durchschnittsgehalt aller Abteilungen verdienen.

Das gleiche Ergebnis liefert demzufolge auch diese Abfrage:

```
SQL>   select ename, sal
       from emp
       where sal >
       (SELECT max(avg(sal)) from emp group by deptno);
```

Dann gibt es noch den Operator **EXISTS**

Was macht nun EXISTS?
Nehmen wir eine Aufgabenstellung an: In welcher Abteilung werden Provisionen gezahlt?

Lösung:

```
SQL>   select dname
       from dept
       where exists(select * from emp
                 where emp.deptno = dept.deptno
                 and comm is not null);
```

Diese Lösung sieht zunächst etwas seltsam aus.

Es gilt folgende Regel:

WAS für Daten die Unterabfrage liefert, ist egal. Es wird nur auf true oder false geprüft. Falls die Unterabfrage einen Treffer liefert, wird die äußere Abfrage eine Ausgabe geben, und zwar nur einmal pro Treffer. EXISTS unterdrückt also doppelte Ausgaben!

Das gleiche Ergebnis liefert diese Abfrage:

```
SQL>   select DISTINCT dname from dept d
       inner join emp e
       on e.deptno=d.deptno
       and comm is not null;
```

19.2 Synchronisierte Unterabfrage

Die synchronisierte Unterabfrage ist eine verschachtelte Abfrage, die einmal für jede von der Hauptabfrage verarbeitete Zeile ausgewertet wird und die bei der Ausführung einen Wert aus einer Spalte in der äußeren Abfrage verwendet.

Im Vergleich zu einer normalen Unterabfrage:

- In einer **normalen verschachtelten Unterabfrage** läuft zuerst die innere SELECT-Anweisung und wird einmal ausgeführt, wobei Werte zurückgegeben werden, die von der Hauptabfrage verwendet werden.
- Eine **synchronisierte Unterabfrage** dagegen wird einmal für jede Kandidatenzeile ausgeführt, die von der äußeren Abfrage als solche betrachtet wird. Die innere Abfrage wird durch die äußere Abfrage gesteuert.

Die Ausführung synchronisierter Unterabfragen sieht wie folgt aus:

- Eine Kandidatenzeile holen (abgerufen durch die äußere Abfrage).
- Die innere Abfrage unter Verwendung des Werts der Kandidatenzeile ausführen.
- Mit dem von der inneren Abfrage stammenden Wert den Kandidaten qualifizieren oder disqualifizieren.
- Wiederholen, bis keine Kandidatenzeilen mehr übrig bleiben.Die **Syntax** für eine synchronisierte Unterabfrage sieht wie folgt aus:_

```
SQL>   SELECT outer1, outer2, …
       FROM table alias1
       WHERE outer1 operator ( SELECT inner1
                 FROM    table2 alias2
                 WHERE   alias1.expr1 = alias2.expr2 );
```

Hinweis: In synchronisierten Unterabfragen können die Operatoren ANY und ALL verwendet werden.

Beispiel:

```
SQL>   SELECT empno, sal, deptno
       FROM emp o
       WHERE sal > (    SELECT AVG(sal)
                 FROM emp i
                 WHERE o.deptno = i.deptno);

     EMPNO       SAL       DEPTNO
----------       ----------    ----------
      7839       5000          10
      7698       2850          30
      7566       2975          20
      7499       1600          30
      7902       3000          20
      7788       3000          20
```

Erläuterung: In diesem Beispiel werden die Angestellten bestimmt, die mehr als das Durchschnittsgehalt ihrer jeweiligen Abteilung verdienen. In diesem Fall wird durch die synchronisierte Unterabfrage das Durchschnittsgehalt für jede Zeile gesondert berechnet. Da sowohl die äußere als auch die innere Abfrage die Tabelle EMP in der FROM-Klausel verwendet, wird EMP in jeder separaten SELECT-Anweisung **zur Klarheit ein Alias gegeben**. Der Alias bewirkt nicht nur eine bessere Lesbarkeit der gesamten SELECT-Anweisung, ohne den Alias würde die Abfrage nicht richtig funktionieren, da die innere Anweisung nicht zwischen der Spalte der inneren Tabelle und der Spalte der äußeren Tabelle unterscheiden könnte.

19.3 Unterabfragen mit mehreren Spalten

Bisher haben Sie Single Row-Unterabfragen und mehrzeilige Unterabfragen erstellt, bei denen nur eine Spalte in der WHERE-Klausel oder der HAVING-Klausel der SELECT-Anweisung verglichen wurde. Wenn Sie zwei oder mehr Spalten vergleichen möchten, müssen Sie mit Hilfe logischer Operatoren eine zusammengesetzte WHERE-Klausel erstellen. Unterabfragen, die mehrere Spalten zurückgeben, bieten Ihnen die Möglichkeit, doppelte WHERE-Bedingungen zu einer einzigen WHERE-Klausel zu kombinieren. Die Spaltenanzahl und Reihenfolge der äusseren Abfrage muss mit der inneren Abfrage übereinstimmen.

Syntax:

Oracle	MS SQL
SQL> SELECT column, column, … FROM table WHERE (column, column, ...) IN (SELECT column, column, … FROM table WHERE condition);	SQL> SELECT column, column, … FROM table WHERE (columnA) IN (SELECT columnA FROM table WHERE condition) AND WHERE (columnB) IN (SELECT columnB FROM table WHERE condition) AND . . . . WHERE (columnN) IN
Ein Beispiel:	
SQL> SELECT ename, deptno, sal FROM emp WHERE (deptno, sal) IN	

Oracle	MS SQL
(SELECT deptno, sal FROM emp WHERE ename = 'MARTIN') AND ename < > 'MARTIN';	(SELECT columnN FROM table WHERE condition) ; Ein Beispiel: SQL> SELECT ename, deptno, sal FROM emp WHERE deptno IN (SELECT deptno FROM emp WHERE ename = 'MARTIN') AND sal in (select sal from emp where ename = 'MARTIN') AND ename < > 'MARTIN';

Gesucht werden alle Mitarbeiter, die in derselben Abteilung wie Martin arbeiten **und zusätzlich** auch noch das gleiche Gehalt beziehen. Falls so eine genaue Übereinstimmung nicht gefragt ist, also das Gehalt von irgendeinem Mitarbeiter herangezogen werden kann in Kombination mit irgendeiner Abteilungsnummer eines vielleicht auch anderen Mitarbeiters, dann ist folgende Syntax zu verwenden:

```
SQL>   SELECT ename, deptno, sal
       FROM emp
       WHERE deptno =
                 (SELECT deptno FROM emp
                 WHERE ename = 'MARTIN')
                 AND sal = (   SELECT sal FROM emp
                         WHERE ename = 'MARTIN')
                 AND ename < > 'MARTIN';
```

19.4 NULL-Werte in Unterabfragen

Erwünscht ist die Ausgabe aller Mitarbeiter, die nicht Vorgesetzter eines anderen Mitarbeiters sind. Die nahe liegende Lösung sieht so aus:

```
SQL>   select ename from emp
       where empno not in (select mgr from emp);
```

Leider bekommen Sie als Ergebnis keine einzige Zeile angezeigt, obwohl es 8 der Bedingung entsprechende Datensätze in der Tabelle emp gibt. Warum?

Weil in der Spalte mgr ein NULL-Wert enthalten ist, nämlich bei KING! Liefert eine Unterabfrage beim Operator **NOT IN** einen NULL_Wert, wird das ganze Ergebnis auf NULL gesetzt!

NVL bzw isNull!

Oracle	MS SQL
SQL> select ename from emp where empno not in (select nvl(mgr,0) from emp);	SQL> select ename from emp where empno not in (select isNul(mgr,0) from emp);

19.5 Unterabfragen in der FROM-Klausel

Sie können Unterabfragen auch in der FROM-Klausel verwenden. Sehen wir uns ein Beispiel an:

```
SQL>   SELECT a.ename, a.sal, a.deptno, b.salavg
       FROM emp a, (    SELECT deptno, avg(sal) salavg
                 FROM emp
                 GROUP BY deptno) b
       WHERE a.deptno = b.deptno
                 AND a.sal > b.salavg;
```

Es werden alle Mitarbeiter angezeigt, die mehr verdienen als das Durchschnittsgehalt ihrer eigenen Abteilung. Die Unterabfrage **MUSS** einen Aliasnamen bekommen und wird wie eine Tabelle behandelt, die allerdings physikalisch nicht vorhanden ist. Sie besteht nur, solange die Abfrage ausgeführt wird.

Übungen

1. Erstellen Sie eine Abfrage, um Namen und Einstellungsdatum aller Mitarbeiter anzuzeigen, die in derselben Abteilung arbeiten wie Blake. Schließen Sie Blake aus der Anzeige aus.

38. Erstellen Sie eine Abfrage, um die Mitarbeiternummer und Namen aller Mitarbeiter anzuzeigen, die mehr als das Durchschnittsgehalt verdienen. Sortieren Sie die Ergebnisse in absteigender Reihenfolge nach dem Gehalt.

39. Zeigen Sie Abteilungsnummer, Namen und Beruf aller Mitarbeiter der Abteilung Sales an.

40. Erstellen Sie eine Abfrage, um Namen, Abteilungsnummer und Gehalt aller Mitarbeiter anzuzeigen, deren Abteilungsnummer UND Gehalt mit Abteilungsnummer UND Gehalt eines Mitarbeiters übereinstimmt, der eine Provision erhält.

41. Zeigen Sie alle Mitarbeiter an, die mehr verdienen als jeder CLERK. Sortieren Sie absteigend nach Gehalt.

42. Zeigen Sie alle mit Namen und Gehalt und Abteilungsnummer an, die das höchste Gehalt in ihrer Abteilung beziehen.

Notizen

Kapitel 20 Views

In diesem Kapitel erfahren Sie Folgendes:

- Sie können eine View beschreiben.
- Sie können eine View erstellen.
- Sie können Daten über eine View abrufen.
- Sie können DML-Aktionen über eine View durchführen.
- Sie können eine View löschen.

Eine View ist kurz gesagt optisch eine Tabelle, die aber physikalisch in der angezeigten Zusammenstellung gar nicht existiert. Wie kann das sein? Nun, eine View stellt eine Sicht auf Auswahl aus Daten einer oder mehrerer Tabellen oder auch anderer Views dar. Wir schauen also wie durch ein Fenster auf verteilt in der Datenbank gespeicherte Daten. Eine View enthält also - noch einmal sei darauf hingewiesen - niemals eigene Daten. Sie ermöglicht uns nur, sie zu sehen. Im Data Dictionary ist eine View als SELECT-Abfrage abgespeichert. Diese Abfrage wird ausgeführt, sobald wir mit der View arbeiten. Die Tabelle(n), auf denen eine View basiert, nennt man Basistabellen. Sollte eine Basistabelle gelöscht werden, wird eine View dadurch natürlich unbrauchbar, weil sie keine Daten mehr zeigen kann.

20.1 Was für Arten von Views gibt es?

Es gibt die einfache View und die komplexe View.

- Eine ***einfache View*** liegt in folgenden Fällen vor:
  - Daten werden nur von einer Tabelle abgeleitet.
  - Sie enthält keine Funktionen oder Datengruppen.
  - DML kann über die View durchgeführt werden.
- Eine ***komplexe View*** liegt in folgenden Fällen vor:
  - Daten werden von vielen Tabellen abgeleitet.
  - Sie enthält Funktionen oder Datengruppen.
  - DML kann nicht immer über die View durchgeführt werden.

20.2 Warum verwendet man Views?

Eine View schränkt den Zugriff auf Daten ein, da sie nur bestimmte Spalten einer Tabelle anzeigen kann. Das heißt, ein Sachbearbeiter für Urlaubsfragen braucht keine Informationen über das Gehalt eines Mitarbeiters. Da man bei Zugriffsberechtigungen aber nur ein SELECT für die **ganze** Tabelle vergeben kann, hätte man am Beispiel der Tabelle emp keine Möglichkeit, den Zugriff auf die Spalten comm und sal zu verhindern. Erstellt man aber eine View, die diese beiden Spalten nicht enthält und vergibt eine SELECT-Berechtigung nur für die View und nicht für die Tabelle, hat man das Problem elegant gelöst. Und der Benutzer merkt es nicht einmal! Eine View wird nämlich bei Abfragen genauso vom User behandelt wie eine Tabelle.

Views ermöglichen dem Benutzer einfache Abfragen, um Ergebnisse aus komplizierten Abfragen abzurufen. Benutzer können beispielsweise Informationen aus mehreren Tabellen abfragen, ohne dass sie wissen müssen, wie man verknüpfte Anweisungen schreibt. Ein weiterer wichtiger Punkt. Da eine View ja eigentlich nichts anderes ist als eine benannte, im Data Dictionary abgespeicherte Abfrage, hat man die Möglichkeit, komplizierte Abfragen als View zu speichern.

Eine Abfrage auf diese View besteht dann vielleicht nur aus einem SELECT * FROM *viewname;*. Das ist aber viel einfacher zu schreiben als eine Abfrage, in der zum Beispiel 3 Tabellen betroffen sind.

20.3 Wie erstellt man Views?

Schauen wir uns die Syntax an:

Oracle	MS SQL

SQL> CREATE [OR REPLACE]
[FORCE|NOFORCE] VIEW view
[(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION
[CONSTRAINT constraint]]
[WITH READ ONLY];
 	
OR REPLACE erstellt die View neu, falls sie bereits vorhanden ist.
 	
FORCE erstellt die View unabhängig davon, ob die Basistabelle existiert.
 	
NOFORCE erstellt die View nur, wenn die Basistabelle existiert. Dies ist der Standard.
 	
view ist der Name der View.
 	
alias gibt Namen für die Ausdrücke an, die durch die View-Abfrage ausgewählt wurden. Die Anzahl der Aliasnamen muss mit der Anzahl der durch die View ausgewählten Ausdrücke übereinstimmen.
 	
subquery ist eine komplette SELECT-Anweisung. Für die Spalten in der SELECT-Liste können Sie Aliasnamen verwenden.
 	
WITH CHECK OPTION gibt an, dass nur solche Zeilen eingefügt oder aktualisiert werden können, die über die View zugänglich sind.
 	
constraint ist der dem Constraint CHECK OPTION zugewiesene Name.
 	
WITH READ ONLY stellt sicher, dass keine DML-Operationen auf diese View durchgeführt werden können.
	SQL> CREATE [OR ALTER] VIEW
	[schema_name .] view
	[(alias[,...alias])
	]
	[WITH <view_attribute>
	[,...n]]
	AS subquery
	[WITH CHECK OPTION]
	[;]
	
	<view_attribute> ::=
	{
	[ENCRYPTION]
	[SCHEMABINDING]
	[VIEW_METADATA]
	
	}
	
	view ist der Name der View.
	
	alias gibt Namen für die Ausdrücke an, die durch die View-Abfrage ausgewählt wurden. Die Anzahl der Aliasnamen muss mit der Anzahl der durch die View ausgewählten Ausdrücke übereinstimmen.
	
	subquery ist eine komplette SELECT-Anweisung. Für die Spalten in der SELECT-Liste können Sie Aliasnamen verwenden.
	
	WITH CHECK OPTION gibt an, dass nur solche Zeilen eingefügt oder aktualisiert werden können, die über die View zugänglich sind.

Ein Beispiel, in dem wir eine View erstellen, die alle Daten aus der Abteilung 10 der Tabelle emp enthält:

```
SQL>   CREATE VIEW abt10
       AS SELECT  empno, ename, job
       FROM emp
       WHERE deptno=10;
```

Über den Befehl DESCRIBE *viewname* können Sie sich die Struktur der View anschauen.

Falls man Aliasnamen für Spalten verwenden möchte, gibt es zwei Möglichkeiten. Entweder Sie geben die Aliasnamen in der SELECT Klausel an:

```
SQL>   CREATE  VIEW  abt10
       AS SELECT  empno Nummer, ename Name, job Beruf
       FROM emp
       WHERE deptno=10;
```

Oder Sie geben die gewünschten Spaltennamen in der CREATE VIEW Klausel an:

```
SQL>   CREATE  VIEW  abt10(Nummer, Name, Beruf)
       AS SELECT  empno, ename, job
       FROM emp
       WHERE deptno=10;
```

Schauen wir uns jetzt noch die Erstellung einer komplexen View an. Im Beispiel wird eine komplexe View von Abteilungsnamen, Mindestgehalt, Höchstgehalt und Durchschnittsgehalt nach Abteilung erstellt. Beachten Sie, dass für die View alternative Namen angegeben werden. Dies ist dann erforderlich, wenn eine beliebige Spalte der View aus einer Funktion oder einem Ausdruck abgeleitet wurde.

```
SQL>   CREATE VIEW dept_sum_vu (name, minsal, maxsal, avgsal)
       AS SELECT d.dname, MIN(e.sal), MAX(e.sal), AVG(e.sal)
       FROM emp e, dept d
       WHERE e.deptno = d.deptno
       GROUP BY d.dname;
```

Richtlinien zum Erstellen einer View

- Die Abfrage, die eine View definiert, kann komplexe SELECT-Syntax, inklusive Joins, Gruppenfunktionen und Subqueries enthalten.
- Die Abfrage, die eine View definiert, darf keine ORDER BY-Klausel enthalten. Die ORDER BY-Klausel wird beim Abrufen von Daten aus der View angegeben.
- Wenn Sie keinen Constraint-Namen für eine View angeben, die mit CHECK OPTION erstellt wurde, weist das System einen Standardnamen zu.
- Sie können die Option OR REPLACE verwenden, um die Definition der View zu ändern, ohne sie zu löschen oder neu zu erstellen oder erneut Objektprivilegien vergeben zu müssen.

20.4 Wie fragt man eine View ab?

Sie können eine View genauso behandeln wie eine Tabelle.

```
SQL>   SELECT * FROM dept_sum_vu;
```

Diese Abfrage liefert für die Departments 10, 20 und 30 das Mindest-, Maximal- und Durchschnittsgehalt.

Möchten Sie dieses nur für die Abteilung ACCOUNTING haben?

```
SQL>   SELECT * FROM dept_sum_vu
       WHERE  dname = 'ACCOUNTING';
```

Vielleicht ist nur das Mindestgehalt gewünscht?

```
SQL>   SELECT MINSAL
       FROM dept_sum_vu;
```

Zur Datenabfrage müssen Sie also nichts Neues lernen.

20.5 DML-Operationen auf Views

Sie können über eine View DML-Operationen auf der Basistabelle durchführen, vorausgesetzt, diese Operationen halten gewisse Regeln ein.

Eine Zeile können Sie aus einer View entfernen, sofern die View nicht eines der folgenden Elemente enthält:

- Gruppenfunktionen
- Eine GROUP BY-Klausel
- Das Schlüsselwort DISTINCT
- Die Pseudo-Spalte ROWNUM
- Set-Operatoren (Mengenoperatoren)

Hinweis: Die DELETE-Anweisung löscht in der Basistabelle, die View ist nur eine gespeicherte SELECT-Anweisung.

Sie können Daten in einer View ändern, sofern nicht eine der oben erwähnten und die folgende Bedingung vorliegt:

- Spalten, die durch Ausdrücke definiert sind, z. B. SAL * 12

Sie können Daten über eine View hinzufügen, sofern sie nicht eines der obigen Elemente sowie keine NOT NULL-Spalten enthält, ohne Standardwerte in der Basistabelle, die nicht durch die View ausgewählt sind. Alle erforderlichen Werte müssen in der View vorhanden sein. Denken Sie daran, dass Sie *über* die View direkt Werte in der zugrundeliegenden Tabelle hinzufügen.

Falls Sie verhindern wollen, dass überhaupt DML-Operationen über die View durchgeführt werden, müssen Sie beim Erstellen der View die WITH READ ONLY Klausel verwenden (nur Oracle).

```
SQL>   CREATE  VIEW  abt10(Nummer, Name, Beruf)
       AS SELECT  empno, ename, job
       FROM emp
       WHERE deptno=10
       WITH READ ONLY;
```

Falls Änderungen nur innerhalb des Sichtbereiches der View zulässig sein sollen, verwenden Sie die WITH CHECK OPTION Klausel.

```
SQL>   CREATE OR REPLACE  VIEW  abt10(Nummer, Name, Beruf)
       AS SELECT  empno, ename, job
       FROM emp
       WHERE deptno=10
       WITH CHECK OPTION;
```

Alle Änderungen, die dafür sorgen würden, dass ein Datensatz nach der Änderung nicht mehr für die View sichtbar wäre, werden durch Oracle verhindert. Die CHECK OPTION überprüft hierzu die WHERE Klausel.

20.6 Wie löscht man eine View?

Nichts leichter als das!

```
SQL>   DROP VIEW viewname;
```

Daten können dadurch nicht gefährdet werden, denn eine View enthält ja keine! Man sollte nur beachten, dass eventuell eine andere View auf die zu löschende zugreift. Diese wird dann nämlich ungültig.

Übungen

1. Erstellen Sie eine View mit Namen EMP_VU, die auf Mitarbeiternummer, Mitarbeitername und Abteilungsnummer für die Tabelle EMP basiert. Ändern Sie den Titel für die Mitarbeiternamen in EMPLOYEE.

43. Zeigen Sie den Inhalt der View EMP_VU an.

44. Geben Sie unter Verwendung der View EMP_VU eine Abfrage ein, um alle Mitarbeiternamen und Abteilungsnummern anzuzeigen.

45. Erstellen Sie eine View mit Namen DEPT20, die Mitarbeiternummer, Mitarbeitername und Abteilungsnummer für alle Mitarbeiter der Abteilung 20 enthält.Beschriften Sie die View- Spalten mit EMPLOYEE_ID, EMPLOYEE und DEPARTMENT_ID. Ein Mitarbeiter darf über die View keiner anderen Abteilung zugeordnet werden.

46. Versuchen Sie, Smith der Abteilung 30 zuzuordnen.

47. Erstellen Sie eine View mit Namen SALARY_VU, die auf Mitarbeiternamen, Abteilungsnamen, Gehalt und Gehaltsstufen aller Mitarbeiter basiert. Beschriften Sie die Spalten mit Employee, Department, Salary und Grade entsprechend.

Notizen

Kapitel 21 Weitere Datenbankobjekte

In diesem Kapitel erfahren Sie Folgendes:

- Sie können Sequenzen erstellen, verwalten und verwenden.
- Sie können Indexe erstellen und verwenden.
- Sie können Synonyme erstellen.

In dieser Unterrichtseinheit lernen Sie, wie Sie einige der häufig verwendeten Datenbankobjekte erstellen und verwalten. Zu diesen Objekten gehören Sequenzen, Indizes und Synonyme.

Viele Anwendungen erfordern eindeutige Nummern als Primärschlüsselwerte. Um diese Anforderung zu erfüllen, können Sie entweder entsprechenden Code in die Anwendung schreiben oder eine Sequenz verwenden, die eindeutige Nummern generiert.

Wenn Sie die Performance einiger Abfragen verbessern möchten, sollten Sie einen Index in Betracht ziehen. Indizes können Sie auch verwenden, um die Eindeutigkeit in einer Spalte oder einer Spaltengruppe zu erzwingen.

Synonyme stellen alternative Namen für Objekte zur Verfügung.

21.1 Sequenzen

Eine Sequenz ist ein Datenbankobjekt, das von einem Benutzer erstellt wird und mit vielen anderen Benutzern gemeinsam zur Generierung eindeutiger ganzer Zahlen genutzt werden kann.

Eine typische Verwendungsmöglichkeit von Sequenzen ist das Erstellen eines Primärschlüsselwerts, der für jede Zeile eindeutig sein muss. Die Sequenz wird von einer internen Routine generiert und erhöht (bzw. niedriger gesetzt). Damit sparen Sie viel Zeit, da die Menge an Anwendungscode reduziert wird, den Sie zum Generieren einer Sequenz für eine Routine schreiben müssen.

Sequenznummern werden unabhängig von Tabellen gespeichert und generiert. Deshalb kann dieselbe Sequenz für mehrere Tabellen verwendet werden.

Sehen wir uns die Grundstruktur des Codes zur Erstellung einer Sequence an:

Oracle	MS SQL
SQL> CREATE SEQUENCE sequence [INCREMENT BY n] [START WITH n] [{MAXVALUE n | NOMAXVALUE}] [{MINVALUE n | NOMINVALUE}] [{CYCLE | NOCYCLE}] [{CACHE n | NOCACHE}];	SQL> CREATE SEQUENCE sequence [INCREMENT BY n] [START WITH n] [{MAXVALUE n | NO MAXVALUE}] [{MINVALUE n | NO MINVALUE}] [{CYCLE | NO CYCLE}] [{CACHE n | NO CACHE}];

Mit der Anweisung CREATE SEQUENCE erstellen Sie sequentielle Nummern automatisch.

- *sequence* ist der Name des Sequenz-Generators.
- INCREMENT BY *n* gibt das Intervall zwischen Sequenznummern an, wobei *n* eine Ganzzahl ist. Wird diese Klausel weggelassen, so wird die Sequenz um 1 erhöht.

- START WITH *n* gibt die erste zu generierende Sequenznummer an. Wird diese Klausel weggelassen, so beginnt die Sequenz mit 1.
- MAXVALUE *n* gibt den Höchstwert an, den die Sequenz generieren kann.
- NOMAXVALUE gibt einen Höchstwert von 10^27 für eine aufsteigende Sequenz und –1 für eine absteigende Sequenz an. Dies ist die Standardoption.
- MINVALUE *n* gibt den Mindestwert der Sequenz an.
- NOMINVALUE gibt einen Mindestwert von 1 für eine aufsteigende Sequenz und -(10^26) für eine absteigende Sequenz an. Dies ist die Standardoption.
- CYCLE | NOCYCLE gibt an, dass die Sequenz auch Werte erzeugt, nachdem entweder der Höchst- oder der Mindestwert erreicht ist oder keine zusätzlichen Werte generiert. Die Standardoption ist NOCYCLE .
- CACHE *n* | NOCACHE gibt an, wie viele Werte der Server zuweisen und im Speicher halten wird.

Oracle	MS SQL
SQL> CREATE SEQUENCE dept_deptno INCREMENT BY 1 START WITH 91 MAXVALUE 100 NOCACHE NOCYCLE;	SQL> CREATE SEQUENCE dept_deptno INCREMENT BY 1 START WITH 91 MAXVALUE 100 NO CACHE NO CYCLE;

Das Beispiel oben erstellt eine Sequenz namens DEPT_DEPTNO, die für die Spalte DEPTNO der Tabelle DEPT verwendet wird. Die Sequenz beginnt bei 91, erlaubt kein Schreiben in den Cache, und sie darf nicht zyklisch sein. Verwenden Sie die CYCLE-Option nicht, wenn die Sequenz Primärschlüsselwerte generieren soll; es sei denn, Sie haben einen zuverlässigen Mechanismus, durch den alte Zeilen schneller systematisch gelöscht werden, als die Sequenz zyklisch durchläuft.

Beachten Sie bitte, dass keine Kommata oder Semikolon mit Ausnahme des Abschlusszeichens gesetzt werden.

Sie können die Werte einer Sequence über die Data Dictionary View user_sequences abfragen.

Beispiel:

```
SQL>   SELECT sequence_name, min_value, max_value,
       increment_by, last_number
       FROM user_sequences;
```

Bitte beachten: Die Spalte last_number liefert die nächste verfügbare Sequencenummer an!

Nach dem Erstellen der Sequenz können Sie damit sequentielle Nummern zur Verwendung in Ihren Tabellen generieren. Referenzieren Sie die Sequenzwerte mit Hilfe der Pseudospalten NEXTVAL und CURRVAL.

Die Pseudospalten NEXTVAL und CURRVAL (Oracle):
Mit der Pseudospalte NEXTVAL extrahieren Sie aufeinanderfolgende Sequenznummern aus einer angegebenen Sequenz. Als Qualifizierer für NEXTVAL müssen Sie den Sequenznamen angeben. Wenn Sie *sequence*.NEXTVAL referenzieren, wird eine neue Sequenznummer generiert und die aktuelle Sequenznummer wird in CURRVAL abgelegt. Mit der Pseudospalte CURRVAL beziehen Sie sich auf eine Sequenznummer, die der aktuelle Benutzer gerade generiert hat. NEXTVAL muss für die Generierung einer Sequenznummer in der aktuellen Benutzersitzung verwen-

det werden, bevor CURRVAL referenziert werden kann. Als Qualifizierer für CURRVAL müssen Sie den Sequenznamen angeben. Wenn *sequence*.CURRVAL referenziert wird, wird der zuletzt zurückgegebene Wert für diesen Benutzerprozess angezeigt.

In MS-SQL gibt es diese Pseudospalten nicht. Hier wird statt sequence.NEXTVAL die Funktion NEXT VALUE FOR sequence verwendet.

Regeln für die Verwendung von CURRVAL und NEXTVAL bzw. NEXT VALUE FOR:
NEXTVAL und CURRVAL können Sie für Folgendes verwenden:

- in einer SELECT-Anweisung, die nicht Teil einer Unterabfrage ist
- in der SELECT-Liste einer Unterabfrage einer INSERT-Anweisung
- in der VALUES-Klausel einer INSERT-Anweisung
- in der SET-Klausel einer UPDATE-Anweisung

NEXTVAL und CURRVAL bzw. NEXT VALUE FOR können Sie für Folgendes ***nicht*** verwenden:

- in der SELECT-Anweisung der CREATE VIEW-Anweisung
- für eine SELECT-Anweisung mit dem Schlüsselwort DISTINCT
- für eine SELECT-Anweisung mit den Klauseln GROUP BY, HAVING oder ORDER BY
- für eine Unterabfrage einer SELECT-, DELETE- oder UPDATE-Anweisung
- für einen DEFAULT-Ausdruck in einer Anweisung CREATE TABLE oder ALTER TABLE

Eine Sequence verwenden Sie wie folgt:

Oracle	MS SQL
SQL> INSERT INTO dept(deptno, dname, loc) VALUES (dept_deptno.NEXTVAL, 'MARKETING', 'SAN DIEGO');	SQL> INSERT INTO dept(deptno, dname, loc) VALUES (NEXT VALUE FOR dept_deptno, 'MARKETING', 'SAN DIEGO');

Im Beispiel wird eine neue Abteilung in die Tabelle DEPT eingefügt. Zum Generieren einer neuen Abteilungsnummer wird die DEPT_DEPTNO -Sequenz verwendet.

Cache-Sequenzen im Speicher ermöglichen einen schnelleren Zugriff auf diese Sequenzwerte. Der Cache wird bei der ersten Referenz auf die Sequenz gefüllt. Jede Anforderung des nächsten

Sequenzwerts wird von der Sequenz im Cache abgerufen. Nachdem die letzte Sequenz verwendet wurde, zieht die nächste Anforderung für eine Sequenz eine weitere Cache-Sequenz in den Speicher. Achten Sie auf Lücken in Ihrer Sequenz. Obwohl Sequenz-Generatoren sequentielle Nummern ohne Lücken ausgeben, geschieht diese Aktion unabhängig von einem Commit oder Rollback. Deshalb geht die Nummer verloren, wenn Sie eine Anweisung zurückrollen, die eine Sequenz enthält. Ein weiteres Ereignis, das Lücken in der Sequenz verursachen kann, ist ein Systemabsturz. Wenn der Server Sequenzwerte in den Cache schreibt, gehen diese Werte bei einem Systemabsturz verloren.

Da Sequenzen nicht direkt mit Tabellen verbunden sind, kann dieselbe Sequenz für mehrere Tabellen verwendet werden. In diesem Fall kann jede Tabelle Lücken in den sequentiellen Nummern aufweisen.

Es ist möglich, sich den nächsten verfügbaren Sequenzwert durch Abfrage der Tabelle USER_SEQUENCES anzusehen, ohne den Wert zu erhöhen, jedoch nur, wenn die Sequenz mit NOCACHE erstellt wurde.

Wenn Sie die Grenze MAXVALUE für Ihre Sequenz erreichen, werden keine weiteren Werte aus der Sequenz zugewiesen, und Sie erhalten eine Fehlermeldung darüber, dass die Sequenz den Wert MAXVALUE überschreitet. Um diese Sequenz weiter zu verwenden, können Sie sie mit der Anweisung ALTER SEQUENCE ändern.

Oracle	MS SQL
SQL> ALTER SEQUENCE dept_deptno INCREMENT BY 1 MAXVALUE 999999 NOCACHE NOCYCLE;	SQL> ALTER SEQUENCE dept_deptno INCREMENT BY 1 MAXVALUE 999999 NO CACHE NO CYCLE;

Richtlinien

- Sie müssen Eigentümer sein oder das Privileg ALTER für die Sequenz besitzen, um sie zu ändern.
- Es sind nur zukünftige Sequenznummern von der Anweisung ALTER SEQUENCE betroffen.
- Die Option START WITH kann mit ALTER SEQUENCE nicht geändert werden. Die Sequenz muss gelöscht und neu erstellt werden, damit die Sequenz bei einer anderen Nummer beginnen kann.
- Es werden Validierungen durchgeführt. Es kann kein neuer MAXVALUE gesetzt werden, der niedriger als die aktuelle Sequenznummer ist.

Das Löschen einer Sequence ist wie üblich sehr einfach:

```
DROP SEQUENCE name;
```

21.2 Index

Ein Index ist ein Datenbank-Objekt, das durch Nutzung einer effektiven Speicherstruktur den Abruf von Zeilen beschleunigen kann. Indizes können explizit oder automatisch erstellt werden.

Falls Sie keinen Index für eine Spalte haben, wird die gesamte Tabelle durchsucht. Ein Index bietet schnellen, direkten Zugriff auf Zeilen in einer Tabelle. Er dient dazu, die Notwendigkeit von Festplattenzugriffen (I/O) zu reduzieren. Dafür verwendet er einen indizierten Pfad, um Daten schnell zu finden. Der Index wird automatisch vom Server verwendet und verwaltet. Sobald ein Index erstellt ist, sind keine direkten Aktivitäten durch den Benutzer mehr erforderlich.

Indizes sind logisch und physikalisch von der Tabelle, die sie indizieren, unabhängig. Dies bedeutet, dass sie jederzeit erstellt oder gelöscht werden können und keine Auswirkungen auf die Basistabellen oder andere Indizes haben.

Hinweis: Beim Löschen einer Tabelle werden die zugehörigen Indizes ebenfalls gelöscht.

Es können zwei Typen von Indizes erstellt werden. Einer davon ist der eindeutige Index. Der Server erstellt diesen Index automatisch, wenn Sie eine Spalte in einer Tabelle mit einem PRIMARY KEY- oder einem UNIQUE Key-Constraint definieren. Der Name des Index ist der Name des Constraint.

Benutzer können sowohl eindeutige als auch nicht-eindeutige Indizes anlegen. Sie können beispielsweise einen Index auf eine FOREIGN KEY-Spalte legen, um die Abfrage zu beschleunigen. Ein Beispiel:

```
SQL>   CREATE UNIQUE INDEX emp_ename_idx
       ON emp(ename);
```

Wenn Sie einen nicht-eindeutigen Index erstellen wollen, lassen Sie UNIQUE weg.

Enthält eine Tabelle mehrere Indizes, so bedeutet dies nicht, dass Abfragen schneller werden. Bei jeder DML-Operation in einer Tabelle mit Indizes müssen diese aktualisiert werden. Je mehr Indizes einer Tabelle zugeordnet sind, desto mehr Aufwand hat der Server beim Aktualisieren aller Indizes nach einer DML-Operation.

In diesen Fällen sollte ein Index erstellt werden:

- Die Spalte wird häufig in der WHERE-Klausel oder in einer JOIN-Bedingung verwendet.
- Die Spalte enthält einen großen Wertebereich.
- Die Spalte enthält eine große Anzahl von NULL-Werten.
- Zwei oder mehr Spalten werden häufig zusammen in einer WHERE-Klausel oder JOIN Bedingung verwendet.
- Es handelt sich um eine große Tabelle, und die meisten Abfragen rufen erwartungsgemäß weniger als 2-4% der Zeilen ab.

Denken Sie daran, dass Sie ein eindeutiges Constraint in der Tabellendefinition festlegen sollten, wenn Sie Eindeutigkeit erzwingen möchten. Dann wird automatisch ein eindeutiger Index erstellt.

In diesen Fällen sollten Sie auf die Erstellung eines Indexes verzichten:

- Es handelt sich um eine kleine Tabelle.
- Die Spalten werden nicht oft als Bedingung in der Abfrage verwendet.
- Die meisten Abfragen rufen erwartungsgemäß mehr als 2–4% der Zeilen ab.
- Die Tabelle wird häufig aktualisiert. Wenn mehr als ein Index in einer Tabelle vorhandensind, erfordern die DML-Anweisungen, die auf die Tabelle zugreifen, aufgrund der Verwaltung der Indizes mehr Zeit.

Das Löschen eines Indexes:

```
SQL>   DROP INDEX name;
```

21.3 Synonyme

Um einen Bezug zu einer Tabelle herzustellen, die einem anderen Benutzer gehört, müssen Sie dem Tabellennamen den Namen des Benutzers als Präfix voranstellen, der die Tabelle erstellt hat. Auf das Präfix muss ein Punkt folgen. Durch das Erstellen eines Synonyms entfällt die Notwendigkeit, den Objektnamen mit dem Schema eindeutig zu kennzeichnen; außerdem steht Ihnen damit ein alternativer Name für eine Tabelle, View, Sequenz, Prozedur oder für andere Objekte zur Verfügung. Diese Methode ist besonders bei langen Objektnamen, wie z. B. Views, nützlich.

Syntax:

Oracle	MS SQL

SQL> CREATE PUBLIC SYNONYM synonym FOR object; Wenn Sie die Option PUBLIC verwenden, ist das Synonym für alle Benutzer zugänglich. Objekt steht für das Datenbankobjekt, für das das Synonym erstellt wird.	SQL> CREATE SYNONYM synonym FOR object;

Ein Beispiel :

Oracle	MS SQL
SQL> CREATE PUBLIC SYNONYM s_emp FOR scott.emp;	SQL> CREATE SYNONYM s_emp FOR scott.emp;

Das Löschen:

Oracle	MS SQL
SQL> DROP SYNONYM s_emp;	SQL> DROP SYNONYM s_emp;

Übungen

1. Erstellen Sie eine Sequenz für die Primärschlüsselspalte der Tabelle DEPARTMENT. Die Sequenz soll bei 60 beginnen und einen Höchstwert von 200 haben. Sie soll um zehn Nummern erhöht werden. Nennen Sie die Sequenz DEPT_ID_SEQ.

48. Schreiben Sie eine SQL-Anweisung, die eine Zeile in die Tabelle DEPARTMENT einfügt. Verwenden Sie auch die Sequenz, die Sie für die ID-Spalte erstellt haben. Fügen Sie die die Abteilung Education hinzu. Überprüfen Sie die Änderungen

Notizen

Kapitel 22 Lösungen zu den Übungen

22.1 Lösungen zu Kapitel 10 – Tabellen erstellen, bearbeiten und löschen

1. Erzeugen Sie in Ihrer Datenbank die Tabellen emp, dept und salgrade entsprechend der Tabellenübersicht auf Seite 153 bis 156

Oracle	MS SQL
emp	
SQL> CREATE TABLE EMP (EMPNO number(4, ENAME VARCHAR2(10), JOB VARCHAR2(9), MGR number(4), HIREDATE date, SAL number(7,2), COMM number(7,2), DEPTNO number(2);	SQL> CREATE TABLE EMP (EMPNO numeric(4, ENAME VARCHAR(10), JOB VARCHAR(9), MGR numeric(4), HIREDATE date, SAL numeric(7,2), COMM numeric(7,2), DEPTNO numeric(2);
dept	
SQL> CREATE TABLE DEPT (DEPTNO number(2), DNAME VARCHAR2(14), LOC VARCHAR2(13));	SQL> CREATE TABLE DEPT (DEPTNO numeric(2, DNAME VARCHAR(14), LOC VARCHAR(13));
salgrade	
SQL> CREATE TABLE SALGRADE (GRADE number(4), LOSAL number(4), HISAL number(4));	SQL> CREATE TABLE SALGRADE (GRADE numeric(4), LOSAL numeric(4), HISAL numeric(4));

2. Erstellen Sie die Tabelle DEPARTMENT basierend auf dem Table Instance-Diagramm unten.

Spaltenname	Id	Name
Schlüsseltyp		
Nullen/Eindeutig		
FK-Tabelle		
FK-Spalte		

Datentyp	Number	Varchar2
Länge	7	25

Oracle	**MS SQL**
`SQL> CREATE TABLE DEPARTMENT ( id number(7), name VARCHAR2(25));`	`SQL> CREATE TABLE DEPARTMENT ( id numeric(7), name VARCHAR(25));`

3. Erstellen Sie die Tabelle EMPLOYEE basierend auf dem Table Instance-Diagramm unten.

Spaltenname	ID	LAST_NAME	FIRST_NAME	DEPT_ID
Schlüsseltyp				
Nullen/Eindeutig				
FK-Tabelle				
FK-Spalte				
Datentyp	Number	Varchar2	Varchar2	Number
Länge	7	25	25	7

Oracle	**MS SQL**
`SQL> CREATE TABLE employee (id NUMBER(7), last_name VARCHAR2(25), first_name VARCHAR2(25), dept_id NUMBER(7));`	`SQL> CREATE TABLE employee (id NUMERIC(7), last_name VARCHAR(25), first_name VARCHAR(25), dept_id NUMERIC(7));`

4. Ändern Sie die Tabelle EMPLOYEE so, dass längere Nachnamen der Mitarbeiter in die Spalte passen. Überprüfen Sie die Änderung.

Oracle	**MS SQL**

SQL> ALTER TABLE employee MODIFY (last_name VARCHAR2(50));	SQL> ALTER TABLE employee ALTER COLUMN last_name VARCHAR(50);
SQL> DESC employee;	SQL> EXEC sp_column employee;

5. Erstellen Sie die Tabelle EMPLOYEE2 basierend auf der Struktur der Tabelle EMP, und fügen Sie nur die Spalten EMPNO, ENAME und DEPTNO ein. Benennen Sie die Spalten in der neuen Tabelle ID, LAST_NAME und DEPT_ID entsprechend.

Oracle	MS SQL
SQL> CREATE TABLE employee2 AS SELECT empno ID, ename LAST_NAME, deptno DEPT_ID FROM emp;	SQL> SELECT empno ID, ename LAST_NAME, deptno DEPT_ID into eployee2 FROM emp

6. Löschen Sie die Tabelle EMPLOYEE.

Oracle	MS SQL
SQL> DROP TABLE employee;	SQL> DROP TABLE employee;

7. Benennen Sie die Tabelle EMPLOYEE2 in EMPLOYEE um.

Oracle	MS SQL
SQL> RENAME employee2 TO employee;	SQL> EXEC sp_rename 'employee2', 'employee';

8. Löschen Sie die Spalte LAST_NAME der Tabelle EMPLOYEE. Überprüfen Sie die Änderung.

Oracle	MS SQL
SQL> ALTER TABLE employee DROP COLUMN last_name; SQL DESC employee	SQL> ALTER TABLE employee DROP COLUMN last_name; SQL> EXEC sp_column employee;

22.2 Lösungen zu Kapitel 11 – Constraints

1. Fügen Sie der ID-Spalte der Tabelle EMP ein PRIMARY KEY-Constraint EMP_ID_PK auf hinzu. Das Constraint soll beim Erstellen benannt werden.

Oracle	MS SQL
SQL> ALTER TABLE emp ADD CONSTRAINT emp_id_pk PRIMARY KEY (empno);	SQL> Alter TABLE emp ALTER COLUMN empno NUMERIC(4) NOT NULL; SQL> ALTER TABLE emp ADD CONSTRAINT emp_id_pk PRIMARY KEY (empno);

49. Erstellen Sie für die ID-Spalte der Tabelle DEPT ein PRIMARY KEY-Constraint DEPT_ID_PK. Das Constraint soll beim Erstellen benannt werden.

Oracle	MS SQL
SQL> ALTER TABLE dept ADD CONSTRAINT dept_id_pk PRIMARY KEY(deptno);	SQL> Alter TABLE dept ALTER COLUMN deptno NUMERIC(2) NOT NULL; SQL> ALTER TABLE dept ADD CONSTRAINT dept_id_pk PRIMARY KEY(deptno);

50. Fügen Sie in der Tabelle EMP einen Fremdschlüssel EMP_DEPT_ID_FK hinzu, der sicherstellt, dass der Mitarbeiter nicht einer nicht vorhandenen Abteilung zugeordnet wird.

Oracle	MS SQL
SQL> ALTER TABLE emp ADD CONSTRAINT emp_dept_id_fk FOREIGN KEY (deptno) REFERENCES dept(deptno);	SQL> Alter TABLE emp ALTER COLUMN deptno NUMERIC(2) NOT NULL; SQL> ALTER TABLE emp ADD CONSTRAINT emp_dept_id_fk FOREIGN KEY (deptno) REFERENCES dept(deptno);

51. Erstellen Sie für die GRADE-Spalte der Tabelle SALGRADE ein NOT NULL-Constraint.

Oracle	MS SQL
SQL> Alter TABLE salgrade MODIFY (grade Not NULL);	SQL> Alter TABLE salgrade ALTER COLUMN grade NUMERIC(4) NOT NULL;

52. Erstellen Sie für die GRADE-Spalte der Tabelle SALGRADE ein PRIMARY KEY-Constraint SALGRADE_GRADE_PK. Das Constraint soll beim Erstellen benannt werden.

Oracle	MS SQL
SQL> ALTER TABLE salgrade ADD CONSTRAINT salgrade_grade_pk PRIMARY KEY(grade);	SQL> ALTER TABLE salgrade ADD CONSTRAINT salgrade_grade_pk PRIMARY KEY(grade);

22.3 Lösungen zu Kapitel 12 – Insert, Update, Delete

1. Befüllen Sie die Tabellen emp, dept und salgrade mit den Werten auf Seite 153 bis 156

Oracle	MS SQL
Tabelle dept	
SQL> INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK'); SQL> INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS'); SQL> INSERT INTO DEPT VALUES (30,'SALES','CHICAGO'); SQL> INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON'); SQL> INSERT INTO dept values (50, 'HEAD_QUARTERS', 'DETROIT');	SQL> INSERT INTO DEPT VALUES (10,'ACCOUNTING','NEW YORK'); SQL> INSERT INTO DEPT VALUES (20,'RESEARCH','DALLAS'); SQL> INSERT INTO DEPT VALUES (30,'SALES','CHICAGO'); SQL> INSERT INTO DEPT VALUES (40,'OPERATIONS','BOSTON'); SQL> INSERT INTO dept values (50, 'HEAD_QUARTERS', 'DETROIT');
Tabelle emp	
INSERT INTO EMP VALUES (7839,'KING','PRESIDENT', NULL,'17.11.1981',5000,NULL,10); SQL> INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER',7839,'01.05.1981',2850,NULL,30); SQL> INSERT INTO EMP VALUES (7782,'CLARK','MANAGER',7839,'09.06.1981',2450,NULL,10); SQL> INSERT INTO EMP VALUES (7566,'JONES','MANAGER',7839,'02.04.1981',2975,NULL,20);	INSERT INTO EMP VALUES (7839,'KING','PRESIDENT', NULL,'17.11.1981',5000,NULL,10); SQL> INSERT INTO EMP VALUES (7698,'BLAKE','MANAGER',7839,'01.05.1981',2850,NULL,30); SQL> INSERT INTO EMP VALUES (7782,'CLARK','MANAGER',7839,'09.06.1981',2450,NULL,10); SQL> INSERT INTO EMP VALUES (7566,'JONES','MANAGER',7839,'02.04.1981',2975,NULL,20);

```
SQL>   INSERT INTO EMP VALUES
       (7654,'MARTIN','SALESMAN'
       ,7698,'02.09.1981',1250,1
       400,30);

SQL>   INSERT INTO EMP VALUES
       (7499,'ALLEN','SALESMAN',
       7698,'20.02.1981',1600,30
       0,30);

SQL>   INSERT INTO EMP VALUES
       (7844,'TURNER','SALESMAN'
       ,7698,'08.09.1981',1500,0
       ,30);

SQL>   INSERT INTO EMP VALUES
       (7900,'JAMES','CLERK',769
       8,'03.12.1981',950,NULL,3
       0);

SQL>   INSERT INTO EMP VALUES
       (7521,'WARD','SALESMAN',7
       698,'22.02.1981',1250,500
       ,30);

SQL>   INSERT INTO EMP VALUES
       (7902,'FORD','ANALYST',75
       66,'03.12.1981',3000,NULL
       ,20);

SQL>   INSERT INTO EMP VALUES
       (7369,'SMITH','CLERK',790
       2,'17.12.1980',800,NULL,2
       0);

SQL>   INSERT INTO EMP VALUES
       (7788,'SCOTT','ANALYST',7
       566,'09.12.1982',3000,NUL
       L,20);

SQL>   INSERT INTO EMP VALUES
       (7876,'ADAMS','CLERK',778
       8,'12.01.1983',1100,NULL,
       20);

SQL>   INSERT INTO EMP VALUES
       (7934,'MILLER','CLERK',77
       82,'23.01.1982',1300,NULL
       ,10);
```


```
SQL>   INSERT INTO EMP VALUES
       (7654,'MARTIN','SALESMAN'
       ,7698,'02.09.1981',1250,1
       400,30);

SQL>   INSERT INTO EMP VALUES
       (7499,'ALLEN','SALESMAN',
       7698,'20.02.1981',1600,30
       0,30);

SQL>   INSERT INTO EMP VALUES
       (7844,'TURNER','SALESMAN'
       ,7698,'08.09.1981',1500,0
       ,30);

SQL>   INSERT INTO EMP VALUES
       (7900,'JAMES','CLERK',769
       8,'03.12.1981',950,NULL,3
       0);

SQL>   INSERT INTO EMP VALUES
       (7521,'WARD','SALESMAN',7
       698,'22.02.1981',1250,500
       ,30);

SQL>   INSERT INTO EMP VALUES
       (7902,'FORD','ANALYST',75
       66,'03.12.1981',3000,NULL
       ,20);

SQL>   INSERT INTO EMP VALUES
       (7369,'SMITH','CLERK',790
       2,'17.12.1980',800,NULL,2
       0);

SQL>   INSERT INTO EMP VALUES
       (7788,'SCOTT','ANALYST',7
       566,'09.12.1982',3000,NUL
       L,20);

SQL>   INSERT INTO EMP VALUES
       (7876,'ADAMS','CLERK',778
       8,'12.01.1983',1100,NULL,
       20);

SQL>   INSERT INTO EMP VALUES
       (7934,'MILLER','CLERK',77
       82,'23.01.1982',1300,NULL
       ,10);
```


Tabelle salgrade

```
SQL>   INSERT INTO SALGRADE
```


```
SQL>   INSERT INTO SALGRADE
```

```
       VALUES (1, 700, 1200)

SQL>   INSERT INTO SALGRADE
       VALUES (2, 1201, 1400)

SQL>   INSERT INTO SALGRADE
       VALUES (3, 1401, 2000)

SQL>   INSERT INTO SALGRADE
       VALUES (3, 2001, 3000)

SQL>   INSERT INTO SALGRADE
       VALUES (5, 3001, 9999)
```

```
       VALUES (1, 700, 1200)

SQL>   INSERT INTO SALGRADE
       VALUES (2, 1201, 1400)

SQL>   INSERT INTO SALGRADE
       VALUES (3, 1401, 2000)

SQL>   INSERT INTO SALGRADE
       VALUES (3, 2001, 3000)

SQL>   INSERT INTO SALGRADE
       VALUES (5, 3001, 9999)
```

53. Erstellen Sie die Tabelle MY_EMPLOYEE unter Verwendung obiger Ansicht.

ID	NUMBER(4)	NOT NULL
LAST_NAME	VARCHAR2(25)	
FIRST_NAME	VARCHAR2(25)	
USERID	VARCHAR2(8)	
SALARY	NUMBER(9,2)	

Oracle

```
SQL>   CREATE TABLE MY_EMPLOYEE
       (ID       NUMBER(4) NOT
       NULL,
       LAST_NAME VARCHAR2(25),
       FIRST_NAME  VARCHAR2(25),
       USERID    VARCHAR2(8),
       SALARY    NUMBER(9,2));
```

MS SQL

```
SQL>   CREATE TABLE MY_EMPLOYEE
       (ID       NUMERIC(4) NOT
       NULL,
       LAST_NAME VARCHAR(25),
       FIRST_NAME  VARCHAR(25),
       USERID    VARCHAR(8),
       SALARY    NUMERIC(9,2));
```

ID	LAST_NAME	FIRST_NAME	USERID	SALARY
1	Patel	Ralph	rpatel	795
2	Dancs	Betty	bdancs	860
3	Biri	Ben	bbiri	1100
4	Newman	Chad	cnewman	750
5	Ropeburn	Audry	aropebur	1550

54. Fügen Sie die erste Zeile in die Tabelle ein, lassen Sie dabei die Spaltenliste weg.

Oracle

```
SQL>   INSERT INTO my_employee
       VALUES (1, 'Patel',
       'Ralph', 'rpatel', 795);
```

MS SQL

```
SQL>   INSERT INTO my_employee
       VALUES (1, 'Patel',
       'Ralph', 'rpatel', 795);
```

55. Fügen Sie die restlichen Zeilen hinzu, diesmal mit Spaltenliste.

Oracle	MS SQL
`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);`	`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (2, 'Dancs', 'Betty', 'bdancs', 860);`
`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (3, 'Biri', 'Ben', 'bbiri', 1100);`	`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (3, 'Biri', 'Ben', 'bbiri', 1100);`
`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (4, 'Newman', 'Chad', 'cnewman', 750);`	`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (4, 'Newman', 'Chad', 'cnewman', 750);`
`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (5, 'Ropeburn', 'Audry', 'aropebur', 860);`	`SQL> INSERT INTO my_employee (id, last_name, first_name, userid, salary) VALUES (5, 'Ropeburn', 'Audry', 'aropebur', 860);`

56. Ändern Sie den Nachnamen von Zeile 3 in Drexler.

Oracle	MS SQL
`SQL> UPDATE my_employee SET last_name = 'Drexler' WHERE id = 3;`	`SQL> UPDATE my_employee SET last_name = 'Drexler' WHERE id = 3;`

57. Ändern Sie für alle Mitarbeiter, die weniger als 900 verdienen, das Gehalt auf 1000.

Oracle	MS SQL
`SQL> UPDATE my_employee SET salary = 1000 WHERE salary < 900;`	`SQL> UPDATE my_employee SET salary = 1000 WHERE salary < 900;`

58.

59. Löschen Sie Betty Dancs.

Oracle	MS SQL
SQL> DELETE FROM my_employee WHERE id = 2;	SQL> DELETE FROM my_employee WHERE id = 2;

60. Schreiben Sie alle Änderungen fest.

Oracle	MS SQL
SQL> COMMIT;	SQL> BEGIN TRANSACTION; (Zu Beginn der Einfügeoperationen) SQL> COMMIT;

61. Setzen Sie einen Savepoint.

Oracle	MS SQL
SQL> SAVEPOINT a;	SQL> BEGIN TRANSACTION; SQL> SAVE TRANSACTION a;

62. Löschen Sie alle Daten aus der Tabelle.

Oracle	MS SQL
SQL> DELETE FROM my_employee;	SQL> DELETE FROM my_employee;

63. Schauen Sie ob die Tabelle wirklich leer ist.

Oracle	MS SQL
SQL> SELECT * FROM my_employee;	SQL> SELECT * FROM my_employee;

64. Machen Sie das Löschen der Daten rückgängig. Sie sollten aber danach 4 Zeilen in der Tabelle aben!

Oracle	MS SQL
SQL> ROLLBACK TO a;	SQL> ROLLBACK TRANSACTION;

65.

66. Schreiben Sie alle Änderungen fest.

Oracle	MS SQL
SQL> COMMIT;	SQL> entfällt

22.4 Lösungen zu Kapitel 13 – Select

1. Geben Sie alle Orte aus, die sich in der Tabelle dept befinden.

Oracle	MS SQL
SQL> SELECT loc FROM dept;	SQL> SELECT loc FROM dept;

67. Zeigen Sie alle Mitarbeiter an und das dazugehörige Einstellungsdatum.

Oracle	MS SQL
SQL> SELECT ename, hiredate FROM emp;	SQL> SELECT ename, hiredate FROM emp;

68. Zeigen Sie alle Gehälter mit den jeweiligen Mitarbeiternamen an und wie viel Gehalt Sie per Monat, per Quartal und per Jahre bekommen.

Oracle	MS SQL
SQL> SELECT ename, sal, 3*sal, 12*sal FROM emp;	SQL> SELECT ename, sal, 3*sal, 12*sal FROM emp;

69. Geben Sie für alle Mitarbeiter folgenden Text vom der Gliederung her aus:
SCOTT arbeitet als ANALYST und bekommt dafür 3000 Euro im Monat

Oracle	MS SQL
SQL> SELECT ename||' arbeitet als '||job|| ' und bekommt dafür '|| sal|| ' Euro im Monat' FROM emp;	SQL> SELECT ename+' arbeitet als '+job+ ' und bekommt dafür '+ convert(varchar, sal)+ ' Euro im Monat' FROM emp;

22.5 Lösungen zu Kapitel 14 – Where

1. Erstellen Sie eine Abfrage, um Namen und Gehälter der Mitarbeiter anzuzeigen, die mehr als $2850 verdienen.

Oracle	MS SQL
SQL> SELECT ename,sal FROM emp WHERE sal > 2850;	SQL> SELECT ename,sal FROM emp WHERE sal > 2850;

70. Erstellen Sie eine Abfrage, um Namen und Abteilungsnummer für die Mitarbeiternummer 7566 anzuzeigen.

Oracle	MS SQL
SQL> SELECT ename,deptno FROM emp WHERE empno = 7566;	SQL> SELECT ename,deptno FROM emp WHERE empno = 7566;

71. Zeigen Sie die Namen, Beruf und Eintrittsdatum aller Mitarbeiter an, die zwischen 20. Februar 1981 und 1. Mai 1981 eingestellt wurden. Ordnen Sie die Abfrage in aufsteigender Reihenfolge nach dem Eintrittsdatum.

Oracle	MS SQL
SQL> SELECT ename,job,hiredate FROM emp WHERE hiredate BETWEEN '20.02.1981' AND '01.05.1981';	SQL> SELECT ename,job,hiredate FROM emp WHERE hiredate BETWEEN '20.02.1981' AND '01.05.1981';

72. Zeigen Sie die Namen aller Mitarbeiter an, deren Namen zwei *L*s enthalten und in Abteilung 30 arbeiten oder deren Manager der Mitarbeiter mit der Nummer 7782 ist.

Oracle	MS SQL
SQL> SELECT ename FROM emp WHERE ename like '%L%L%' AND deptno = 30 OR mgr = 7782;	SQL> SELECT ename FROM emp WHERE ename like '%L%L%' AND deptno = 30 OR mgr = 7782;

22.6 Lösungen zu Kapitel 15 – Single-Row-Funktionen

1. Zeigen Sie empno, ename,sal und das um 16% erhöhte Gehalt als Ganzzahl aus der Tabelle emp an.

Oracle	MS SQL
SQL> SELECT empno,ename,sal, ROUND(sal*1.16,0) FROM emp;	SQL> SELECT empno,ename,sal, CAST(ROUND(sal*1.16,0) AS int) FROM emp;

73. Ändern Sie die Abfrage, indem Sie eine zusätzliche Spalte einfügen, die die Gehaltserhöhung anzeigt.

Oracle	MS SQL
SQL> SELECT empno, ename, sal, round(sal*1.16,0),round(sal*1.16,0)-sal FROM emp;	SQL> SELECT empno, ename, sal, cast(round(sal*1.16,0) as int), cast(round(sal*1.16,0)-sal as int) FROM emp;

74. Geben Sie die Namen aller Mitarbeiter an zusammen mit der Anzahl der Monate,die die Mitarbeiter schon in der Firma beschäftigt sind. Zeigen Sie nur vollendete Monate an.

Oracle	MS SQL
SQL> SELECT ename, TRUNC(MONTHS_BETWEEN(sysdate,hiredate),0) FROM emp;	SQL> SELECT ename, ROUND(DATEDIFF(month, hiredate, sysdatetime()), 0, 1) FROM emp;

75. Zeigen Sie die Namen aller Mitarbeiter an zusammen mit ihrem Gehalt. Geben Sie als Spaltenbreite 15 an und füllen Sie die Lücke mit * rechts auf.

Oracle	MS SQL
SQL> SELECT ename, RPAD(sal,15,'*') FROM emp;	SQL> SELECT ename, LEFT((LEFT(cast(sal as int), 15)) + REPLICATE('*', 15), 15) FROM emp;

76. Erstellen Sie eine Abfrage, die den Namen mit Grossbuchstaben zu Beginn und folgenden Kleinbuchstaben ausgibt. Ermitteln Sie die Länge des jeweiligen Namens. Einschränkung: Bitte nur für alle Namen,die mit J, A oder M beginnen.

Oracle	MS SQL
SQL> SELECT INITCAP(ename), LENGTH(ename) FROM emp WHERE ename LIKE 'J%' OR ename LIKE 'M%' OR ename LIKE 'A%';	SQL> SELECT Left(upper(ename) , 1) + substring(lower(ename), 2, len(ename)), LEN(ename) FROM emp WHERE ename LIKE 'J%' OR ename LIKE 'M%' OR ename LIKE 'A%';

22.7 Lösungen zu Kapitel 16 – Konvertierung

1. Geben Sie die Systemzeit mit Datum und Uhrzeit im Format Stunden:Minuten;Sekunden Tag.Monat.Jahr aus. Dabei alle Angaben zweistellig, das Jahr vierstellig (denken Sie bei Oracle an die Tabelle dual).

Oracle	MS SQL
SQL> SELECT TO_CHAR(sysdate, 'HH24:MI:SS DD.MM.YYYY') FROM dual;	SQL> SELECT format(getDate(),'HH:mm:s s dd.MM.yyyy');

77. Geben Sie ename und comm aus der Tabelle emp aus. Falls ein Mitarbeiter keine Provision erhält, soll ’keine Provision’ als Ausgabe erscheinen.

Oracle	MS SQL
SQL> SELECT ename, NVL(TO_CHAR(comm),'keine Provision') FROM emp;	SQL> SELECT ename, isNull(convert(varchar, comm),'keine Provision') FROM emp;

78. Zeigen Sie Namen, Hiredate, und das Datum des Montags 6 Monate nach dem Einstellungsdatum an. Beschriften sie die letzte Spalte mit Vorschau. Anzeige des Datums in der letzten Spalte bitte in folgender Form: Montag, the Twenty-Fourth of Mai, 1982 .

Oracle	MS SQL
SQL> SELECT ename, hiredate, TO_CHAR(NEXT_DAY(ADD_MONTHS(hiredate, 6) ,'MONDAY'), 'fmDay, "the" Ddspth "of" Month, YYYY') VORSCHAU FROM emp;	SQL> DECLARE @NextDayID INT; SET @NextDayID = 2; -- Next Monday SELECT format(DATEADD(DAY, (DATEDIFF(DAY, ((@NextDayID + 5) % 7), hiredate) / 7) * 7 + 7, ((@NextDayID + 5) % 7)), 'dddd", the "dd"th of "MMMM, yyyy') AS Vorschau from emp;

79. Zeigen Sie Namen, Hiredate und Wochentag des Arbeitsbeginns jedes Mitarbeitersan. Überschrift Tag für diese Spalte. Sortieren Sie bitte nach dem Wochentag, beginnend bei Montag.

Oracle	MS SQL
SQL> SELECT ename, hiredate,TO_CHAR(hiredate, 'DAY') "Tag" FROM emp ORDER BY TO_CHAR(hiredate, 'd');	SQL> SELECT ename, hiredate, format(hiredate, 'dddd') "Tag" FROM emp ORDER BY DATEPART(dw, hiredate);

22.8 Lösungen zu Kapitel 17 – Join

1. Zeigen Sie alle Berufe aus Abteilung 30 jeweils einmal. Der Standort der Abteilung soll mit ausgegeben werden.

Oracle	MS SQL
SQL> SELECT DISTINCT e.job, d.loc FROM emp e, dept d WHERE e.deptno= d.deptno AND e.deptno=30; oder:	SQL> SELECT DISTINCT e.job, d.loc FROM emp e, dept d WHERE e.deptno= d.deptno AND e.deptno=30; oder:

```
SQL>   SELECT DISTINCT e.job,
       d.loc
       FROM emp e inner join
       dept d
       ON e.deptno = d.deptno
       WHERE e.deptno = 30;
```

```
SQL>   SELECT DISTINCT e.job,
       d.loc
       FROM emp e inner join
       dept d
       ON e.deptno = d.deptno
       WHERE e.deptno = 30;
```

80. Erstellen Sie eine Abfrage, die alle Mitarbeiter mit Namen, Beruf, Abteilungsnummer und Abteilungsnamen anzeigt, die in DALLAS arbeiten.

Oracle

```
SQL>   SELECT e.ename, e.job,
       d.deptno, d.dname
       FROM emp e, dept d
       WHERE e.deptno=d.deptno
       AND d.loc ='DALLAS';

oder:

SQL>   SELECT e.ename, e.job,
       d.deptno, d.dname
       FROM emp e INNER JOIN
       dept d
       ON e.deptno=d.deptno
       WHERE d.loc ='DALLAS';
```

MS SQL

```
SQL>   SELECT e.ename, e.job,
       d.deptno, d.dname
       FROM emp e, dept d
       WHERE e.deptno=d.deptno
       AND d.loc ='DALLAS';

oder:

SQL>   SELECT e.ename, e.job,
       d.deptno, d.dname
       FROM emp e INNER JOIN
       dept d
       ON e.deptno=d.deptno
       WHERE d.loc ='DALLAS';
```

81. Zeigen Sie den Namen und die Nummer jedes Mitarbeiters zusammen mit Namen und Mitarbeiternummer des dazugehörigen Managers an.

Oracle

```
SQL>   SELECT e.ename, e.empno,
       m.ename, m.empno
       FROM emp e, emp m
       WHERE e.mgr= m.empno;

oder:

SQL>   SELECT e.ename, e.empno,
       m.ename, m.empno
       FROM emp e INNER JOIN emp
       m
       ON e.mgr= m.empno;
```

MS SQL

```
SQL>   SELECT e.ename, e.empno,
       m.ename, m.empno
       FROM emp e, emp m
       WHERE e.mgr= m.empno;

oder:

SQL>   SELECT e.ename, e.empno,
       m.ename, m.empno
       FROM emp e INNER JOIN emp
       m
       ON e.mgr= m.empno;
```

82. Erstellen Sie eine Abfrage, welche Namen, Abteilungsnummer und alle Mitarbeiter anzeigt, die in derselben Abteilung arbeiten wie ein bestimmter Mitarbeiter. Lösung ergibt 28 Zeilen! Sortieren Sie nach der Spaltenreihenfolge.

Oracle	MS SQL
SQL> SELECT e.deptno , e.ename Mitarbeiter,c.ename Kollege FROM emp e, emp c WHERE e.deptno = c.deptno AND e.empno > c.empno ORDER BY e.deptno, e.ename, c.ename; oder: SQL> SELECT e.deptno , e.ename Mitarbeiter,c.ename Kollege FROM emp e INNER JOIN emp c ON e.deptno = c.deptno WHERE e.empno > c.empno ORDER BY e.deptno, e.ename, c.ename;	SQL> SELECT e.deptno , e.ename Mitarbeiter,c.ename Kollege FROM emp e, emp c WHERE e.deptno = c.deptno AND e.empno > c.empno ORDER BY e.deptno, e.ename, c.ename; oder: SQL> SELECT e.deptno , e.ename Mitarbeiter,c.ename Kollege FROM emp e INNER JOIN emp c ON e.deptno = c.deptno WHERE e.empno > c.empno ORDER BY e.deptno, e.ename, c.ename;

83. Ändern Sie Aufgabe 3 so, dass KING mit in der Anzeige erscheint.

Oracle	MS SQL
SQL> SELECT e.ename, e.empno, m.ename, m.empno FROM emp e , emp m WHERE e.mgr= m.empno(+); oder: SQL> SELECT e.ename, e.empno, m.ename, m.empno FROM emp e LEFT OUTER JOIN emp m ON e.mgr= m.empno;	SQL> SELECT e.deptno, e.ename Mitarbeiter, c.ename Kollege FROM emp e INNER JOIN emp c ON e.deptno = c.deptno WHERE e.empno > c.empno ORDER BY e.deptno, e.ename, c.ename;

84. Zeigen Sie Namen und hiredate aller Mitarbeiter an, die nach BLAKE eingestellt wurden.

Oracle	MS SQL
SQL> SELECT emp.ename, emp.hiredate FROM emp, emp blake WHERE blake.ename = 'BLAKE' AND blake.hiredate < emp.hiredate;	SQL> SELECT emp.ename, emp.hiredate FROM emp, emp blake WHERE blake.ename = 'BLAKE' AND blake.hiredate < emp.hiredate;

oder: SQL> SELECT emp.ename, emp.hiredate FROM emp INNER JOIN emp blake ON blake.hiredate < emp.hiredate WHERE blake.ename = 'BLAKE';	oder: SQL> SELECT emp.ename, emp.hiredate FROM emp INNER JOIN emp blake ON blake.hiredate < emp.hiredate WHERE blake.ename = 'BLAKE';

22.9 Lösungen zu Kapitel 18 – Multi-Row-Funktionen

1. Ermitteln Sie die Anzahl der Mitarbeiter pro Berufsgruppe.

Oracle	MS SQL
SQL> SELECT job, COUNT(*) FROM emp GROUP BY job;	SQL> SELECT job, COUNT(*) FROM emp GROUP BY job;

85. Zeigen Sie die Differenz zwischen dem höchsten und dem niedrigsten Gehalt an.

Oracle	MS SQL
SQL> SELECT MAX(sal) – MIN(sal) from emp;	SQL> SELECT MAX(sal) – MIN(sal) from emp;

86. Zeigen Sie die mgr und sal des am schlechtesten bezahlten Mitarbeiters dieses Managers an. Schließen Sie jeden aus, der keine mgr hat. Berücksichtigt werden sollen nur Gruppen mit Werten über 1000.

Oracle	MS SQL
SQL> SELECT mgr, MIN(sal) FROM emp WHERE mgr IS NOT NULL GROUP BY mgr HAVING MIN(sal) > 1000;	SQL> SELECT mgr, MIN(sal) FROM emp WHERE mgr IS NOT NULL GROUP BY mgr HAVING MIN(sal) > 1000;

87. Erstellen Sie eine Abfrage, welche einmal die Gesamtzahl der Mitarbeiter in einer Spalte TOTAL anzeigt, und dann zeigen Sie an, wieviele Mitarbeiter pro Jahr eingestellt wurden. Beschriften Sie hierbei jede Jahresspalte mit der entsprechenden Jahreszahl.

Oracle	MS SQL
SQL> SELECT COUNT(*) TOTAL, SUM(DECODE(TO_CHAR(hireda te,'YYYY'), '1980',1,0)) "1980", SUM(DECODE(TO_CHAR(hireda te,'YYYY'), '1981',1,0)) "1981", SUM(DECODE(TO_CHAR(hireda te,'YYYY'), '1982',1,0)) "1982", SUM(DECODE(TO_CHAR(hireda te,'YYYY'), '1983',1,0)) "1983" FROM emp; oder: SQL> SELECT COUNT(*) TOTAL, SUM(case TO_CHAR(hiredate,'yyyy') when '1980' then 1 else 0 end) "1980", SUM(case TO_CHAR(hiredate,'yyyy') when '1981' then 1 else 0 end) "1981", SUM(case TO_CHAR(hiredate,'yyyy') when '1982' then 1 else 0 end) "1982", SUM(case TO_CHAR(hiredate,'yyyy') when '1983' then 1 else 0 end) "1983" FROM emp;	SQL> SELECT COUNT(*) TOTAL, SUM(case format(hiredate,'yyyy') when '1980' then 1 else 0 end) "1980", SUM(case format(hiredate,'yyyy') when '1981' then 1 else 0 end) "1981", SUM(case format(hiredate,'yyyy') when '1982' then 1 else 0 end) "1982", SUM(case format(hiredate,'yyyy') when '1983' then 1 else 0 end) "1983" FROM emp;

88. Zeigen Sie die höchsten, niedrigsten, Summen- und Durchschnittsgehälter aller Mitarbeiter an. Beschriften Sie die Spalten mit Maximum, Minimum, Sum und Average. Runden Sie Ihre Ergebnisse auf die Dezimalstelle.

Oracle	MS SQL
SQL> SELECT ROUND(MIN(sal),0) "Minimum", ROUND(SUM(sal),0) "Sum", ROUND(AVG(sal),0) "Average" FROM emp;	SQL> SELECT ROUND(MIN(sal),0) "Minimum", ROUND(SUM(sal),0) "Sum", ROUND(AVG(sal),0) "Average" FROM emp;

89. Bearbeiten Sie die vorherige Aufgabe, um das minimale und maximale, Gehalt sowie die Summe aller Gehälter und das Durchschnittsgehalt für jede Funktion (Job Type) anzuzeigen.

Oracle	MS SQL
SQL> SELECT JOB, ROUND(MAX(sal),0) "Maximum", ROUND(MIN(sal),0) "Minimum", ROUND(SUM(sal),0) "Sum", ROUND(AVG(sal),0) "Average" FROM emp GROUP BY JOB;	SQL> SELECT emp.ename, emp.hiredate FROM emp, emp blake WHERE blake.ename = 'BLAKE' AND blake.hiredate < emp.hiredate; oder: SQL> SELECT emp.ename, emp.hiredate FROM emp INNER JOIN emp blake ON blake.hiredate < emp.hiredate WHERE blake.ename = 'BLAKE';

90. Erstellen Sie eine Abfrage, um den Namen der Abteilung (Department), des Standorts (location), die Anzahl der Mitarbeiter und das Durchschnittsgehalt (Average Salary) aller Mitarbeiter in dieser Abteilung anzuzeigen. Beschriften Sie die Spalten mit dname, loc, Number of People und Salary.

Oracle	MS SQL
SQL> SELECT d.dname, d.loc, COUNT(*) "Number of People", ROUND(AVG(sal),2) "Salary" FROM emp e, dept d WHERE e.deptno = d.deptno GROUP BY d.dname, d.loc;	SQL> SELECT d.dname, d.loc, COUNT(*) "Number of People", ROUND(AVG(sal),2) "Salary" FROM emp e, dept d WHERE e.deptno = d.deptno GROUP BY d.dname, d.loc;

22.10 Lösungen zu Kapitel 19 – Unterabfragen

1. Erstellen Sie eine Abfrage, um Namen und Einstellungsdatum aller Mitarbeiter anzuzeigen, die in derselben Abteilung arbeiten wie Blake. Schließen Sie Blake aus der Anzeige aus.

Oracle	MS SQL

```
SQL>   SELECT ename, hiredate
       FROM emp
       WHERE deptno IN (SELECT
       deptno
       FROM emp
       WHERE ename = 'BLAKE')
       AND ename != 'BLAKE';
```

```
SQL>   SELECT ename, hiredate
       FROM emp
       WHERE deptno IN (SELECT
       deptno
       FROM emp
       WHERE ename = 'BLAKE')
       AND ename != 'BLAKE';
```

91. Erstellen Sie eine Abfrage, um die Mitarbeiternummer und Namen aller Mitarbeiter anzuzeigen, die mehr als das Durchschnittsgehalt verdienen. Sortieren Sie die Ergebnisse in absteigender Reihenfolge nach dem Gehalt.

Oracle

```
SQL>   SELECT empno, ename
       FROM emp
       WHERE sal > (SELECT
       AVG(sal)
       FROM emp)
       ORDER BY sal DESC;
```

MS SQL

```
SQL>   SELECT empno, ename
       FROM emp
       WHERE sal > (SELECT
       AVG(sal)
       FROM emp)
       ORDER BY sal DESC;
```

92. Zeigen Sie Abteilungsnummer, Namen und Beruf aller Mitarbeiter der Abteilung Sales an.

Oracle

```
SQL>   SELECT deptno, ename, job
       FROM emp
       WHERE deptno IN (SELECT
       deptno
       FROM dept
       WHERE dname = 'SALES');
```

MS SQL

```
SQL>   SELECT deptno, ename, job
       FROM emp
       WHERE deptno IN (SELECT
       deptno
       FROM dept
       WHERE dname = 'SALES');
```

93. Erstellen Sie eine Abfrage, um Namen, Abteilungsnummer und Gehalt aller Mitarbeiter anzuzeigen, deren Abteilungsnummer **UND** Gehalt mit Abteilungsnummer **UND** Gehalt eines Mitarbeiters übereinstimmt, der eine Provision erhält.

Oracle	MS SQL
SQL> SELECT ename, deptno, sal FROM emp WHERE (sal, deptno) IN (SELECT sal, deptno FROM emp WHERE comm IS NOT NULL);	SQL> SELECT ename, deptno, sal FROM emp WHERE (sal) IN (SELECT sal FROM emp WHERE comm IS NOT NULL) AND (deptno) IN (SELECT deptno FROM emp WHERE comm IS NOT NULL);

94. Zeigen Sie alle Mitarbeiter an, die mehr verdienen als jeder CLERK. Sortieren Sie absteigend nach Gehalt.

Oracle	MS SQL
SQL> SELECT ename, job, sal FROM emp WHERE sal > ALL (SELECT sal FROM emp WHERE job = 'CLERK') ORDER BY sal DESC;	SQL> SELECT ename, job, sal FROM emp WHERE sal > ALL (SELECT sal FROM emp WHERE job = 'CLERK') ORDER BY sal DESC;

95.

96. Zeigen Sie alle mit Namen und Gehalt und Abteilungsnummer an, die das höchste Gehalt in ihrer Abteilung beziehen.

Oracle	MS SQL

SQL> select a.ename, a.sal, a.deptno
from emp a, (select deptno, max(sal) maximal
from emp group by deptno) b
where a.deptno = b.deptno
and a.sal = b.maximal;
 	
oder:
 	
SQL> select a.ename, a.sal, a.deptno
from emp a inner join
(select deptno, max(sal) maximal from emp group by deptno) b
on a.deptno = b.deptno
where a.sal = b.maximal;
 	
oder:
 	
SQL> select o.ename, o.sal, o.deptno
from emp o
where o.sal = (select max(sal) from emp i where o.deptno = i.deptno);
	SQL> select a.ename, a.sal, a.deptno
	from emp a, (select deptno, max(sal) maximal
	from emp group by deptno) b
	where a.deptno = b.deptno
	and a.sal = b.maximal;
	
	oder:
	
	SQL> select a.ename, a.sal, a.deptno
	from emp a inner join
	(select deptno, max(sal) maximal from emp group by deptno) b
	on a.deptno = b.deptno
	where a.sal = b.maximal;
	
	oder:
	
	SQL> select o.ename, o.sal, o.deptno
	from emp o
	where o.sal = (select max(sal) from emp i where o.deptno = i.deptno);

22.11 Lösungen zu Kapitel 20 – Views

1. Erstellen Sie eine View mit Namen EMP_VU, die auf Mitarbeiternummer, Mitarbeitername und Abteilungsnummer für die Tabelle EMP basiert. Ändern Sie den Titel für die Mitarbeiternamen in EMPLOYEE.

Oracle	MS SQL
SQL> CREATE VIEW emp_vu AS SELECT empno, ename employee, deptno FROM emp;	SQL> CREATE VIEW emp_vu AS SELECT empno, ename employee, deptno FROM emp;

97.

98. Zeigen Sie den Inhalt der View EMP_VU an.

Oracle	MS SQL
SQL> SELECT * FROM EMP_VU;	SQL> SELECT * FROM EMP_VU;

99. Geben Sie unter Verwendung der View EMP_VU eine Abfrage ein, um alle Mitarbeiternamen und Abteilungsnummern anzuzeigen.

Oracle	MS SQL
SQL> SELECT employee, deptno FROM emp_vu;	SQL> SELECT employee, deptno FROM emp_vu;

100. Erstellen Sie eine View mit Namen DEPT20, die Mitarbeiternummer, Mitarbeitername und Abteilungsnummer für alle Mitarbeiter der Abteilung 20 enthält.Beschriften Sie die View- Spalten mit EMPLOYEE_ID, EMPLOYEE und DEPARTMENT_ID. Ein Mitarbeiter darf über die View keiner anderen Abteilung zugeordnet werden.

Oracle	MS SQL
SQL> CREATE VIEW dept20 AS SELECT empno employee_id, ename employee, deptno department_id FROM emp WHERE deptno = 20 WITH CHECK OPTION CONSTRAINT emp_dept_20;	SQL> CREATE VIEW dept20 AS SELECT empno employee_id, ename employee, deptno department_id FROM emp WHERE deptno = 20 WITH CHECK OPTION;

101. Versuchen Sie, Smith der Abteilung 30 zuzuordnen.

Oracle	MS SQL
SQL> UPDATE dept20 SET department_id = 30 WHERE employee = 'SMITH'; klappt aber nicht!!!!	SQL> UPDATE dept20 SET department_id = 30 WHERE employee = 'SMITH'; klappt aber nicht!!!!

102. Erstellen Sie eine View mit Namen SALARY_VU, die auf Mitarbeiternamen, Abteilungsnamen, Gehalt und Gehaltsstufen aller Mitarbeiter basiert. Beschriften Sie die Spalten mit Employee, Department, Salary und Grade entsprechend.

Oracle	MS SQL
SQL> CREATE VIEW salary_vu AS SELECT ename employee, dname department, sal salary, grade FROM emp e, dept d, salgrade s WHERE e.deptno = d.deptno AND e.sal between s.losal and s.hisal;	SQL> CREATE VIEW salary_vu AS SELECT ename employee, dname department, sal salary, grade FROM emp e, dept d, salgrade s WHERE e.deptno = d.deptno AND e.sal between s.losal and s.hisal;

22.12 Lösungen zu Kapitel 21 – Weitere Datenbankobjekte

1. Erstellen Sie eine Sequenz für die Primärschlüsselspalte der Tabelle DEPARTMENT. Die Sequenz soll bei 60 beginnen und einen Höchstwert von 200 haben. Sie soll um zehn Nummern erhöht werden. Nennen Sie die Sequenz DEPT_ID_SEQ.

Oracle	MS SQL
SQL> CREATE SEQUENCE dept_id_seq START WITH 60 INCREMENT BY 10 MAXVALUE 200;	SQL> CREATE SEQUENCE dept_id_seq START WITH 60 INCREMENT BY 10 MAXVALUE 200;

103. Schreiben Sie eine SQL-Anweisung, die eine Zeile in die Tabelle DEPARTMENT einfügt. Verwenden Sie auch die Sequenz, die Sie für die ID-Spalte erstellt haben. Fügen Sie die die Abteilung Education hinzu. Überprüfen Sie die Änderungen.

Oracle	MS SQL
SQL> INSERT INTO department (id, name) VALUES (dept_id_seq.NEXTVAL, '&name') SQL> select * from department;	SQL> INSERT INTO department (id, name) VALUES (NEXT VALUE FOR dept_id_seq, 'EDUCATION'); SQL> select * from department;

Kapitel 23 Anhang

23.1 Tabellenübersicht

23.1.1 Tabelle emp

```
SQL> DESCRIBE emp
Name                                      Null?    Type
 ----------------------------------------- -------- ----
 EMPNO                                     NOT NULL NUMBER(4)
 ENAME                                              VARCHAR2(10)
 JOB                                                VARCHAR2(9)
 MGR                                                NUMBER(4)
 HIREDATE                                           DATE
 SAL                                                NUMBER(7,2)
 COMM                                               NUMBER(7,2)
 DEPTNO                                    NOT NULL NUMBER(2)
```

```
SQL> SELECT * FROM emp;
    EMPNO ENAME      JOB              MGR HIREDATE         SAL       COMM  DEPTNO
--------- ---------- --------- --------- --------- --------- --------- -------
     7839 KING       PRESIDENT           17.11.81      5000                10
     7698 BLAKE      MANAGER        7839 01.05.81      2850                30
     7782 CLARK      MANAGER        7839 09.06.81      2450                10
     7566 JONES      MANAGER        7839 02.04.81      2975                20
     7654 MARTIN     SALESMAN       7698 28.09.81      1250      1400      30
     7499 ALLEN      SALESMAN       7698 20.02.81      1600       300      30
     7844 TURNER     SALESMAN       7698 08.09.81      1500         0      30
     7900 JAMES      CLERK          7698 03.12.81       950                30
     7521 WARD       SALESMAN       7698 22.02.81      1250       500      30
     7902 FORD       ANALYST        7566 03.12.81      3000                20
     7369 SMITH      CLERK          7902 17.12.80       800                20
     7788 SCOTT      ANALYST        7566 09.12.82      3000                20
     7876 ADAMS      CLERK          7788 12.01.83      1100                20
     7934 MILLER     CLERK          7782 23.01.82      1300                10
```

23.1.2 Tabelle dept

```
SQL> DESCRIBE dept
Name                                      Null?    Type
 ----------------------------------- -------- ----
 DEPTNO                                    NOT NULL NUMBER(2)
 DNAME                                              VARCHAR2(14)
 LOC                                                VARCHAR2(13)
```

```
SQL> SELECT * FROM dept;

   DEPTNO DNAME          LOC
--------- -------------- -------------
       10 ACCOUNTING     NEW YORK
       20 RESEARCH       DALLAS
       30 SALES          CHICAGO
       40 OPERATIONS     BOSTON
       50 HEAD_QUARTERS  DETROIT
```

23.1.3 Tabelle customer

```
SQL> DESCRIBE customer
   Name                      Null?    Type
   ------------------------- -------- ------------
   LAST_NAME                 NOT NULL VARCHAR2(30)
   STATE_CD                           CHAR(2)
   SALES                              NUMBER
```

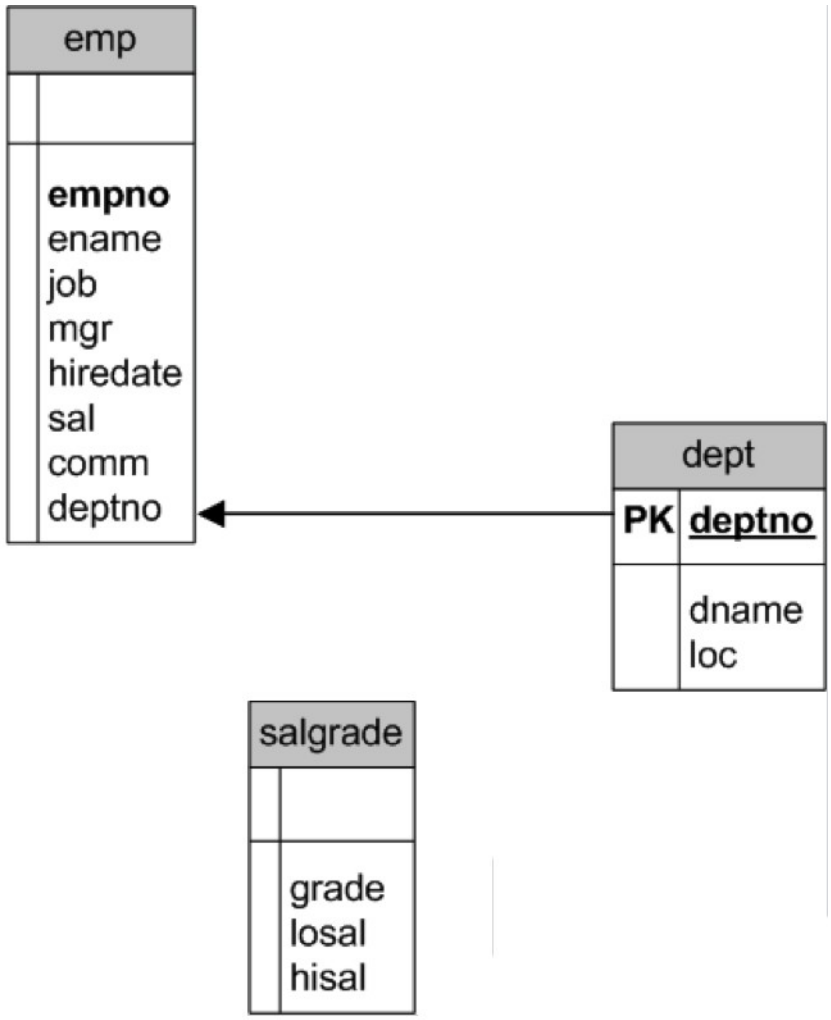
```
SQL> SELECT * FROM customer;
LAST_NAME                     ST      SALES
----------------------------- -- ----------
Nicholson                     CA    6998,99
Martin                        CA    2345,45
Laursen                       CA      34,34
Bambi                         CA    1234,55
McGraw                        NJ     123,45
Teplow                        MA   23445,67
Abbey                         CA    6969,96
```

23.1.4 Tabelle salgrade

```
SQL> DESCRIBE salgrade
 Name                                      Null?    Type
  ----------------------------------------- -------- ----
  GRADE                                              NUMBER
  LOSAL                                              NUMBER
  HISAL                                              NUMBER
```

```
SQL> SELECT * FROM salgrade;
    GRADE     LOSAL     HISAL
--------- --------- ---------
        1       700      1200
        2      1201      1400
        3      1401      2000
        4      2001      3000
        5      3001      9999
```

23.1.5 Tabellenrelationsmodell



Kapitel 24 Index

N

O

P

R

S

T

U

V

W

Z