Einführung in SQL: Druckversion: Grundlagen

Inhaltsverzeichnis

Wegen des Umfangs und der Komplexität der einzelnen Kapitel besteht die Druckversion aus mehreren Teilen.

Grundlagen

Dieser Abschnitt bietet grundlegende Informationen zur Arbeit mit SQL.

- SQL-Befehle enthält eine Übersicht:
  - DML (1) – Daten abfragen behandelt den SELECT-Befehl aus der Data Manipulation Language (DML)
  - DML (2) – Daten speichern behandelt INSERT, UPDATE, DELETE aus der Data Manipulation Language (DML)
  - DDL – Struktur der Datenbank behandelt die Data Definition Language (DDL)
  - TCL – Ablaufsteuerung behandelt die Transaction Control Language (TCL)
  - DCL – Zugriffsrechte behandelt die Data Control Language (DCL)
- Datentypen erläutert den Umgang mit Zahlen, Zeichenketten usw.
- Funktionen enthält eine erste Übersicht über Hilfsmittel, die immer wieder benutzt werden.

Die anderen Teile

Über das Inhaltsverzeichnis des Buches sind die anderen Teile der Druckversion zu erreichen:

- Einführung
- Mehr zu Abfragen
- Erweiterungen
- Anhang

Sonstiges

- Lizenz

SQL-Befehle

Manchen Einsteigern mag SQL sehr sperrig erscheinen. SQL ist weniger für improvisierte Einmalabfragen gedacht als vielmehr zur Entwicklung von stabilen, dauerhaft nutzbaren Abfragen. Wenn die Abfrage einmal entwickelt ist, wird sie meistens in eine GUI- oder HTML-Umgebung eingebunden, sodass der Benutzer mit dem SQL-Code gar nicht mehr in Berührung kommt.

Es gibt zwar einen SQL-Standard (siehe das Kapitel Einleitung), der möglichst von allen Datenbanken übernommen werden soll, aber leider hält sich kein angebotenes DBMS (Datenbank-Management-System) vollständig daran. Es kommt also je nach DBMS zu leichten bis großen Abweichungen, und für Sie führt kein Weg daran vorbei, immer wieder in der Dokumentation ihrer persönlichen Datenbank nachzulesen. Das gilt natürlich besonders für Verfahren, die im SQL-Standard gar nicht enthalten sind, von Ihrem DBMS trotzdem angeboten werden.

Die Abweichungen eines bestimmten DBMS vom SQL-Standard werden sehr oft als **SQL-Dialekt** bezeichnet.

Allgemeine Hinweise

Die gesamte Menge an Befehlen ist recht überschaubar; Schwierigkeiten machen die vielen Parameter mit zahlreichen Varianten.

Der Übersicht halber wurde SQL in **Teilbereiche** gegliedert; allerdings gibt es auch für diese Aufteilung Unterschiede bei den DBMS, den Dokumentationen und in Fachbüchern. Diese Aufteilung (siehe Inhaltsverzeichnis dieses Kapitels) dient aber nicht nur der Übersicht, sondern hat auch praktische Gründe:

- Data Manipulation Language (DML)-Befehle werden vor allem von „einfachen“ Anwendern benutzt.
- Data Definition Language (DDL)- und Transaction Control Language (TCL)-Befehle dienen Programmierern.
- Data Control Language (DCL)-Befehle gehören zum Aufgabenbereich von Systemadministratoren.

Dies sind die **Bestandteile** eines einzelnen Befehls:

- der Name des Befehls
- der Name des Objekts (Datenbank, Tabelle, Spalte usw.)
- ein Hinweis zur Maßnahme, soweit diese nicht durch den Befehl klar ist
- weitere Einzelheiten
- das Semikolon als Zeichen für den Abschluss eines SQL-Befehls

Damit dies alles eindeutig ist, gibt es eine Reihe von Schlüsselwörtern (key words, reserved words, non-reserved words), anhand derer das DBMS die Informationen innerhalb eines Befehls erkennt.

Die **Schreibweise** eines Befehls ist flexibel.

- Groß- und Kleinschreibung der Schlüsselwörter werden nicht unterschieden.
- Ein Befehl kann beliebig auf eine oder mehrere Zeilen verteilt werden; der wichtigste Gesichtspunkt dabei ist die Lesbarkeit auch für Sie selbst beim Schreiben.
- Das Semikolon ist nicht immer erforderlich, wird aber empfohlen. Bei manchen DBMS wird der Befehl erst nach einem folgenden GO o. ä. ausgeführt.

Für eigene **Bezeichner**, d. h. die Namen von Tabellen, Spalten oder eigenen Funktionen gilt:

- Vermeiden Sie unbedingt, Schlüsselwörter dafür zu verwenden; dies führt schnell zu Problemen auch dort, wo es möglich wäre.
- Das Wort muss in der Regel mit einem Buchstaben oder dem Unterstrich a…z A…Z _ beginnen. Danach folgen beliebig Ziffern und Buchstaben.
- Inwieweit andere Zeichen und länderspezifische Buchstaben (Umlaute) möglich sind, hängt vom DBMS ab.

Der Autor dieser Zeilen rät davon ab, aber das mag Geschmackssache sein. Bei den Erläuterungen zur Beispieldatenbank (Namen von Tabellen und Spalten) wird sowieso für englische Bezeichner plädiert.

Dieses Buch geht davon aus, dass Schlüsselwörter nicht als Bezeichner dienen und auch Umlaute nicht benutzt werden.

Kommentare können in SQL-Befehle fast beliebig eingefügt werden (nur die Schlüsselwörter dürfen natürlich nicht „zerrissen“ werden). Es gibt zwei Arten von Kommentaren:

```
-- (doppelter Bindestrich, am besten mit Leerzeichen dahinter)
```

Alles von den beiden Strichen an (einschließlich) bis zum Ende dieser Zeile gilt als Kommentar und nicht als Bestandteil des Befehls.

```
/* (längerer Text, gerne auch über mehrere Zeilen) */
```

Alles, was zwischen `'/*'` und `'*/'` steht (einschließlich dieser Begrenzungszeichen), gilt als Kommentar und nicht als Bestandteil des Befehls.

DML – Data Manipulation Language

DML beschäftigt sich mit dem **Inhalt** des Datenbestandes. „Manipulation“ ist dabei nicht nur im Sinne von „Manipulieren“ zu verstehen, sondern allgemeiner im Sinne von „in die Hand nehmen“ (lat. *manus* = Hand).

Das Kapitel DML (1) - Daten abfragen befasst sich mit dem SELECT-Befehl.

- Gelegentlich finden Sie dafür auch den Begriff *Data Query Language (DQL)*. Diese Einführung fasst ihn als Teilgebiet der DML auf; nur aus Gründen der Übersicht gibt es getrennte Kapitel.

Die Befehle INSERT, UPDATE, DELETE dienen der Speicherung von Daten und werden in DML (2) - Daten speichern behandelt.

Bei diesen Befehlen ist immer genau eine Tabelle – nur bei SELECT auch mehrere – anzugeben, dazu Art und Umfang der Arbeiten sowie in aller Regel mit WHERE eine Liste von Bedingungen, welche Datensätze bearbeitet werden sollen.

Die folgenden Erläuterungen sind einheitlich für alle DML-Befehle zu beachten.

Datenmengen, nicht einzelne Datensätze

Bitte beachten Sie, dass SQL grundsätzlich **mengenorientiert** arbeitet. DML-Befehle wirken sich meistens nicht nur auf einen Datensatz aus, sondern auf eine ganze Menge, die aus 0, einem oder mehreren Datensätzen bestehen kann. Auch die WHERE-Bedingungen sorgen „nur“ für den Umfang der Datenmenge; aber das Ergebnis ist immer eine Datenmenge.

Im Einzelfall wissen Sie als Anwender oder Programmierer natürlich häufig, ob die Datenmenge 0, 1 oder n Datensätze enthalten kann oder soll. Aber Sie müssen selbst darauf achten, denn SQL oder das DBMS können das nicht wissen.

Die Struktur als Daten*menge* führt auch dazu, dass es bei einem SELECT-Befehl keine „natürliche“ Reihenfolge gibt, in der die Daten angezeigt werden. Manchmal kommen sie in der Reihenfolge, in der sie gespeichert wurden; aber bei umfangreicheren Tabellen mit vielen Änderungen sieht es eher nach einem großen Durcheinander aus – es sei denn, Sie verwenden die ORDER BY-Klausel.

SQL-Ausdrücke

In vielen Fällen wird der Begriff **Ausdruck** verwendet. Dies ist ein allgemeiner Begriff für verschiedene Situationen:

- An Stellen, an denen ein einzelner Wert angegeben werden muss, kann auch ein Werte-Ausdruck verwendet werden: ein konstanter Wert (Zahl, Text, boolescher Wert), das Ergebnis einer Funktion, die einen solchen Wert zurückgibt, oder eine Abfrage, die als Ergebnis einen einzigen Wert liefert.
- An Stellen, an denen eine oder mehrere Zeilen einer Datenmenge angegeben werden müssen, kann auch ein SQL-Ausdruck (im SQL-Standard als *query expression* bezeichnet) verwendet werden. Dabei handelt es sich um einen SQL-Befehl (in der Regel einen SELECT-Befehl), der als Ergebnis eine Menge von Datensätzen liefert.
- An Stellen, an denen eine Liste einzelner Werte angegeben werden muss, kann ebenfalls ein SQL-Ausdruck verwendet werden; dieser muss dann als Ergebnis eine Menge passender Werte liefern.

Datenintegrität

Bei allen Datenmanipulationen ist zu beachten, dass die Bedingungen für Querverweise erhalten bleiben, siehe das Kapitel Fremdschlüssel-Beziehungen. Beispielsweise darf ein Datensatz in der Tabelle *Abteilung* erst dann gelöscht werden, wenn alle zugeordneten Mitarbeiter gelöscht oder versetzt wurden.

Hinweis für Programmierer: Parameter benutzen!

Ein SQL-Befehl in einem Programm sollte niemals als „langer String“ mit *festen* Texten erstellt werden, sondern mit Parametern. Das erleichtert die Wiederverwendung, die Einbindung von Werten, vermeidet Probleme mit der Formatierung von Zahlen und Datumsangaben und verhindert SQL-Injection.

Im Kapitel DML (2) - Daten speichern steht unter UPDATE so ein Beispiel:

Dies korrigiert die Schreibweise des Namens beim Mitarbeiter mit der Personalnummer 20001

```
UPDATE Mitarbeiter
   SET Name = 'Mayer'
 WHERE Personalnummer  = 20001;
```

Dieses Beispiel sieht besser so aus:

```
UPDATE Mitarbeiter
   SET Name = @Name
 WHERE Personalnummer  = @PersNr;
```

In einem Programm mit ADO.NET und Visual Basic für MS-SQL wird dieser Befehl wie folgt verwendet.

VB.NET-Quelltext

```
' zuerst der vollständige SQL-Befehl in einer oder mehreren Zeilen
Dim sel As String = "UPDATE ... @PersNr;"
' danach die Festlegungen zu den Parametern
Dim cmd As SqlCommand = new SqlCommand(sel)
```

```
cmd.Parameters.AddWithValue("@Name", "Mayer")
cmd.Parameters.AddWithValue("@PersNr", 20001)
```

Zwar unterscheidet sich die Art, wie die Parameter bezeichnet werden, nach Programmiersprache und DBMS. Auch sieht es nach mehr Schreibarbeit aus. Aber diese Unterschiede sind viel geringer als die Unterschiede nach den Datentypen; insgesamt wird die Arbeit viel sicherer.

Parameter gibt es nur für DML-Befehle, nicht für DDL oder DCL.

DDL – Data Definition Language

DDL **definiert** die Struktur einer Datenbank.

Hierzu gibt es die Befehle CREATE, ALTER, DROP; diese werden für Datenbank-Objekte DATABASE, TABLE, VIEW usw. verwendet.

Einzelheiten werden in DDL - Struktur der Datenbank behandelt.

TCL – Transaction Control Language

Damit die Daten dauerhaft zusammenpassen, also die Integrität der Daten gewahrt bleibt, sollen Änderungen, die zusammengehören, auch „am Stück“ übertragen und gespeichert werden. Falls eine einzelne dieser Änderungen nicht funktioniert, muss der gesamte Vorgang rückgängig gemacht werden.

Dies wird durch **Transaktionen** gesteuert, die mit COMMIT oder ROLLBACK abgeschlossen werden.

Einzelheiten werden in TCL - Ablaufsteuerung behandelt.

DCL – Data Control Language

Eine „vollwertige“ SQL-Datenbank regelt umfassend die **Rechte für den Zugriff** auf Objekte (Tabellen, einzelne Felder, interne Funktionen usw.). Hierzu kommen die Befehle GRANT und REVOKE zum Einsatz.

Mehr darüber steht in DCL - Zugriffsrechte.

Zusammenfassung

In diesem Kapitel lernten wir grundlegende Informationen zu SQL-Befehlen kennen:

- Die Befehle der DML (Data Manipulation Language) bearbeiten die Daten und gehören zum Aufgabenbereich eines jeden Benutzers.
- Mit SELECT werden Daten abgerufen, mit INSERT und UPDATE gespeichert und mit DELETE gelöscht.
- Die DML-Befehle arbeiten mengenorientiert; anstelle konkreter Werte werden häufig Ausdrücke verwendet.
- Die Befehle der DDL (Data Definition Language) steuern die interne Struktur der Datenbank und gehören zum Aufgabenbereich von Programmentwicklern.
- Die Befehle der TCL (Transaction Control Language) sorgen für die Integrität der Daten beim Speichern und gehören ebenfalls zum Aufgabenbereich von Programmentwicklern.
- Die Befehle der DCL (Data Control Language) sorgen für die Zugriffssicherheit und gehören zum Aufgabenbereich von Systemadministratoren.

Die Bestandteile der SQL-Befehle werden anhand von Schlüsselwörtern erkannt und ausgewertet.

Übungen

Übung 1	Begriffsklärung	Zur Lösung

Was versteht man unter „SQL“?

1. Süß – Quadratisch – Lecker
2. Server's Quick Library
3. Structured Query Language
4. Standard Quarterly Lectures

Übung 2	Begriffsklärung	Zur Lösung

Was versteht man unter „DBMS“?

1. Datenbankmanagementsprache
2. Datenbankmanagementsystem
3. Data-Base Manipulation Standard
4. Deutsche Bahn Mobilstation

Übung 3	Begriffsklärung	Zur Lösung

Was versteht man unter „SQL-Dialekt“?

1. die Sprache des Computers
2. die Sprache des Benutzers
3. die Sprache des Programmierers eines DBMS
4. die Abweichungen der SQL-Befehle vom SQL-Standard
5. die jeweilige Version eines DBMS

Übung 4	Teilbereiche von SQL	Zur Lösung

Zu welchem der SQL-Teilbereiche DML (Data Manipulation Language), DDL (Data Definition Language), TCL (Transaction Control Language), DCL (Data Control Language) gehören die folgenden Befehle?

1. wähle Daten aus
2. erzeuge eine Tabelle in der Datenbank
3. erzeuge eine Prozedur
4. ändere die Informationen, die zu einem Mitarbeiter gespeichert sind
5. ändere die Definition einer Spalte
6. bestätige eine Gruppe von zusammengehörenden Anweisungen
7. gewähre einem Benutzer Zugriffsrechte auf eine Tabelle
8. SELECT ID FROM Mitarbeiter
9. ROLLBACK
10. UPDATE Versicherungsvertrag SET ...
11. lösche einen Datensatz in einer Tabelle
12. lösche eine Tabelle insgesamt

Übung 5	SQL-Kommentare	Zur Lösung

Welche der folgenden Zeilen enthalten korrekte Kommentare? *Mit … werden weitere Teile angedeutet, die für die Frage unwichtig sind.*

1. `SELECT * FROM Mitarbeiter;`
2. `UPDATE Mitarbeiter SET Name = 'neu'; -- ändert den Namen aller Zeilen`

3. `DEL/*löschen*/ETE FROM Mitarbeiter WHERE ...`
4. `-- DELETE FROM Mitarbeiter WHERE ...`
5. `UPDATE Mitarbeiter /* ändern */ SET Name = ...; /* usw. */`

Lösungen

Lösung zu Übung 1	Begriffsklärung	Zur Übung

Antwort 3 ist richtig.

Lösung zu Übung 2	Begriffsklärung	Zur Übung

Antwort 2 ist richtig.

Lösung zu Übung 3	Begriffsklärung	Zur Übung

Antwort 4 ist richtig.

Lösung zu Übung 4	Teilbereiche von SQL	Zur Übung

1. DML – 2. DDL – 3. DDL – 4. DML
5. DDL – 6. TCL – 7. DCL – 8. DML
9. TCL – 10. DML – 11. DML – 12. DDL

Lösung zu Übung 5	SQL-Kommentare	Zur Übung

1. Diese Zeile enthält keinen Kommentar.
2. Der letzte Teil nach dem Semikolon ist ein Kommentar.
3. Dieser Versuch eines Kommentars zerstört das Befehlswort DELETE und ist deshalb unzulässig.
4. Die gesamte Zeile zählt als Kommentar.
5. Diese Zeile enthält zwei Kommentare: zum einen zwischen dem Tabellennamen und dem Schlüsselwort SET sowie den Rest der Zeile hinter dem Semikolon.

Siehe auch

Bei Wikipedia gibt es Erläuterungen zu Fachbegriffen:

- Grafische Benutzeroberfläche (GUI)
- HTML
- Parameter in der Informatik
- SQL-Injection
- .NET und ADO.NET
- Datenintegrität

Zur Arbeit mit SQL-Parametern unter C# und ADO.NET gibt es eine ausführliche Darstellung:

- [Artikelserie] Parameter von SQL-Befehlen

DML (1) – Daten abfragen

Eine Datenbank enthält eine Vielzahl von verschiedenen Daten. Abfragen dienen dazu, bestimmte Daten aus der Datenbank auszugeben. Dabei kann die Ergebnismenge entsprechend den Anforderungen eingegrenzt und genauer gesteuert werden.

Dieser Teilbereich der Data Manipulation Language (DML) behandelt den SQL-Befehl **SELECT**, mit dem Abfragen durchgeführt werden.

Zunächst geht es um einfache Abfragen. Vertieft wird der SELECT-Befehl unter „Abfragen für Fortgeschrittene“ behandelt, beginnend mit Ausführliche SELECT-Struktur; unten im Abschnitt Ausblick auf komplexe Abfragen gibt es Hinweise auf diese weiteren Möglichkeiten.

SELECT – Allgemeine Hinweise

SELECT ist in der Regel der erste und wichtigste Befehl, den der SQL-Neuling kennenlernt, und das aus gutem Grund: Man kann damit keinen Schaden anrichten. Ein Fehler im Befehl führt höchstens zu einer Fehlermeldung oder dem Ausbleiben des Abfrageergebnisses, aber nicht zu Schäden am Datenbestand. Trotzdem erlaubt der Befehl das Herantasten an die wichtigsten Konzepte von DML, und die anderen Befehle müssen nicht mehr so intensiv erläutert werden.

Dieser Befehl enthält die folgenden Bestandteile („Klauseln“ genannt).

```
SELECT [DISTINCT | ALL]
          <spaltenliste>  | *
  FROM    <tabellenliste>
[WHERE    <bedingungsliste>]
[GROUP BY <spaltenliste>   ]
[HAVING   <bedingungsliste>]
[UNION    <select-ausdruck>]
[ORDER BY <spaltenliste>   ]
;
```

Die Reihenfolge der Klauseln ist fest im SQL-Standard vorgegeben. Klauseln, die in [ ] stehen, sind nicht nötig, sondern können entfallen; der Name des Befehls und die FROM-Angaben sind unbedingt erforderlich, das Semikolon als Standard empfohlen.

Die wichtigsten Teile werden in den folgenden Abschnitten erläutert.

Die folgenden Punkte verlangen dagegen vertiefte Beschäftigung mit SQL:

- GROUP BY – Daten gruppieren
- HAVING – weitere Einschränkungen
- UNION – mehrere Abfragen verbinden

Diese Punkte sowie weitere Einzelheiten zu den wichtigsten Bestandteilen werden in Ausführliche SELECT-Struktur und anderen „fortgeschrittenen“ Kapiteln behandelt.

Die Beispiele beziehen sich auf den Anfangsbestand der Beispieldatenbank; auf die Ausgabe der selektierten Datensätze wird in der Regel verzichtet. Bitte probieren Sie alle Beispiele aus und nehmen Sie verschiedene Änderungen vor, um die Auswirkungen zu erkennen.

Die einfachste Abfrage

Gesucht wird der Inhalt der Tabelle der Fahrzeughersteller mit all ihren Spalten und Datensätzen (Zeilen).

```
SELECT * FROM Fahrzeughersteller ;
```

Schauen wir uns das Beispiel etwas genauer an:

- Die beiden Begriffe **SELECT** und **FROM** sind SQL-spezifische Bezeichner.
- *Fahrzeughersteller* ist der Name der Tabelle, aus der die Daten selektiert und ausgegeben werden sollen.
- Das Sternchen, Asterisk genannt, ist eine Kurzfassung für „alle Spalten“.

Eingrenzen der Spalten

Nun wollen wir nur bestimmte Spalten ausgeben, nämlich eine Liste aller Fahrzeughersteller; das Land interessiert uns dabei nicht. Dazu müssen zwischen den SQL-Bezeichnern SELECT und FROM die auszugebenden Spalten angegeben werden. Sind es mehrere, dann werden diese durch jeweils ein Komma getrennt.

So erhält man die Namensliste aller Fahrzeughersteller:

```
SELECT Name FROM Fahrzeughersteller ;
```

Folgendes Beispiel gibt die beiden Spalten für Name und Land des Herstellers aus. Die Spalten werden durch Komma getrennt.

```
SELECT Name, Land FROM Fahrzeughersteller ;
```

Für die Ausgabe kann eine (abweichende) Spaltenüberschrift festgelegt werden. Diese wird als **Spalten-Alias** bezeichnet. Der Alias kann dem Spaltennamen direkt folgen oder mit dem Bindewort **AS** angegeben werden. Das vorherige Beispiel kann also wie folgt mit dem Alias *Hersteller* für *Name* (ohne AS) und dem Alias *Staat* für *Land* (mit AS) versehen werden:

```
SELECT Name Hersteller, Land AS Staat
  FROM Fahrzeughersteller ;
```

DISTINCT – Keine doppelten Zeilen

Gesucht wird die Liste der Herstellerländer:

```
```

```
SELECT Land
  FROM Fahrzeughersteller ;
```

Dabei stellen wir fest, dass je Hersteller eine Zeile ausgegeben wird. Somit erscheint beispielweise 'Deutschland' mehrmals. Damit keine doppelten Zeilen ausgegeben werden, wird DISTINCT vor den Spaltennamen in das SQL-Statement eingefügt:

```
SELECT DISTINCT Land
  FROM Fahrzeughersteller ;
```

Damit erscheint jedes Herstellerland nur einmal in der Liste.

Die Alternative zu DISTINCT ist übrigens das in der Syntax genannte ALL: alle Zeilen werden gewünscht, ggf. auch doppelte. Dies ist aber der Standardwert, ALL kann weggelassen werden.

Vertiefte Erläuterungen sind unter Nützliche Erweiterungen zu finden.

WHERE – Eingrenzen der Ergebnismenge

Fast immer soll nicht der komplette Inhalt einer Tabelle ausgegeben werden. Dazu wird die Ergebnismenge mittels Bedingungen in der **WHERE**-Klausel eingegrenzt, welche nach dem Tabellennamen im SELECT-Befehl steht.

Eine Bedingung ist ein logischer Ausdruck, dessen Ergebnis WAHR oder FALSCH ist. In diesen logischen Ausdrücken werden die Inhalte der Spalten (vorwiegend) mit konstanten Werten verglichen. Hierbei stehen verschiedene Operatoren zur Verfügung, vor allem:

```
=  gleich              <>  ungleich; seltener auch:  !=
<  kleiner als         <=  kleiner als oder gleich
>  größer  als         >=  größer  als oder gleich
```

Bedingungen können durch die logischen Operatoren **OR** und **AND** und die Klammern **()** verknüpft werden. Je komplizierter solche Verknüpfungen werden, desto sicherer ist es, die Bedingungen durch Klammern zu gliedern. Mit diesen Mitteln lässt sich die Abfrage entsprechend eingrenzen.

Beispielsweise sollen alle Hersteller angezeigt werden, die ihren Sitz in Schweden oder Frankreich haben:

```
SELECT * FROM Fahrzeughersteller
 WHERE ( Land = 'Schweden' ) OR ( Land = 'Frankreich' );
```

Auf die Klammern kann hier verzichtet werden.

Hinter der WHERE-Klausel kann man also eine oder mehrere (mit einem booleschen Operator verknüpft) Bedingungen einfügen. Jede einzelne besteht aus dem Namen der Spalte, deren Inhalt überprüft werden soll, und einem Wert, wobei beide mit einem Vergleichsoperator verknüpft sind.

In einer anderen Abfrage sollen alle Fahrzeughersteller angezeigt werden, die außerhalb Deutschlands sitzen. Jetzt könnte man alle anderen Fälle einzeln in der WHERE-Klausel auflisten, oder man dreht einfach den Vergleichsoperator um.

```
SELECT * FROM Fahrzeughersteller
 WHERE Land <> 'Deutschland' ;
```

Das Gleichheitszeichen aus der oberen Abfrage wurde durch das Ungleichheitszeichen ersetzt. Dadurch werden jetzt alle Hersteller ausgegeben, deren Sitz ungleich Deutschland ist.

Vertiefte Erläuterungen sind unter WHERE-Klausel im Detail zu finden.

ORDER BY – Sortieren

Nachdem wir nun die Zeilen und Spalten der Ergebnismenge eingrenzen können, wollen wir die Ausgabe der Zeilen sortieren. Hierfür wird die **ORDER BY**-Klausel genutzt. Diese ist die letzte im SQL-Befehl vor dem abschließenden Semikolon und enthält die Spalten, nach denen sortiert werden soll.

So lassen wir uns die Liste der Hersteller nach dem Namen sortiert ausgeben:

```
SELECT * FROM Fahrzeughersteller
 ORDER BY Name;
```

Anstatt des Spaltennamens kann auch die Nummer der Spalte genutzt werden. Mit dem folgenden Statement erreichen wir also das gleiche Ergebnis, da *Name* die zweite Spalte in unserer Ausgabe ist:

```
SELECT * FROM Fahrzeughersteller
 ORDER BY 2;
```

Die Angabe nach Spaltennummer ist unüblich; sie wird eigentlich höchstens dann verwendet, wenn die Spalten genau aufgeführt werden und komplizierte Angaben – z. B. Berechnete Spalten – enthalten.

Die Sortierung erfolgt standardmäßig aufsteigend; das kann auch durch **ASC** ausdrücklich angegeben werden. Die Sortierreihenfolge kann mit dem **DESC**-Bezeichner in *absteigend* verändert werden.

```
SELECT * FROM Fahrzeughersteller
 ORDER BY Name DESC;
```

In SQL kann nicht nur nach einer Spalte sortiert werden. Es können mehrere Spalten zur Sortierung herangezogen werden. Hierbei kann für jede Spalte eine eigene Regel verwendet werden. Dabei gilt, dass die Regel zu einer folgend angegebenen Spalte der Regel zu der vorig angegebenen Spalte untergeordnet ist. Bei der Sortierung nach Land und Name wird also zuerst nach dem Land und dann je Land nach Name sortiert. Eine Neusortierung nach Name, die jene Sortierung nach Land wieder verwirft, findet also nicht statt.

Der folgende Befehl liefert die Hersteller – zuerst absteigend nach Land und dann aufsteigend sortiert nach dem Namen – zurück.

```
SELECT * FROM Fahrzeughersteller
 ORDER BY Land DESC, Name ASC;
```

FROM – Mehrere Tabellen verknüpfen

In fast allen Abfragen werden Informationen aus mehreren Tabellen zusammengefasst. Die sinnvolle Speicherung von Daten in getrennten Tabellen ist eines der Merkmale eines relationalen DBMS; deshalb müssen die Daten bei einer Abfrage nach praktischen Gesichtspunkten zusammengeführt werden.

Traditionell mit FROM und WH ERE

Beim „traditonellen“ Weg werden dazu einfach alle Tabellen in der FROM-Klausel aufgeführt und durch jeweils eine Bedingung in der WHERE-Klausel verknüpft.

Ermittle die Angaben der Mitarbeiter, deren Abteilung ihren Sitz in Dortmund oder Bochum hat.

```
SELECT mi.Name,
       mi.Vorname,
       mi.Raum,
       ab.Ort
  FROM Mitarbeiter mi, Abteilung ab
 WHERE mi.Abteilung_ID = ab.ID
   AND ab.Ort in ('Dortmund', 'Bochum')
 ORDER BY mi.Name, mi.Vorname;
```

Es werden also Informationen aus den Tabellen *Mitarbeiter* (Name und Raum) sowie *Abteilung* (Ort) gesucht. Für die Verknüpfung der Tabellen werden folgende Bestandteile benötigt:

- In der FROM-Klausel stehen die benötigten Tabellen.
- Zur Vereinfachung wird jeder Tabelle ein Kürzel als **Tabellen-Alias** zugewiesen.
- In der Spaltenliste wird jede einzelne Spalte mit dem Namen der betreffenden Tabelle bzw. dem Alias verbunden. (Der Tabellenname bzw. Alias kann sehr oft weggelassen werden; aber schon wegen der Übersichtlichkeit sollte er immer benutzt werden.)
- Die WHERE-Klausel enthält die Verknüpfungsbedingung „mi.Abteilung_ID = ab.ID“ – zusätzlich zur Einschränkung nach dem Sitz der Abteilung.

Jede Tabelle in einer solchen Abfrage benötigt mindestens eine direkte Verknüpfung zu einer anderen Tabelle. Alle Tabellen müssen zumindest indirekt miteinander verknüpft sein. Falsche Verknüpfungen sind eine häufige Fehlerquelle.

Vertiefte Erläuterungen sind unter Einfache Tabellenverknüpfung zu finden.

Modern mit JOIN...ON

Beim „modernen“ Weg wird eine Tabelle in der FROM-Klausel aufgeführt, nämlich diejenige, die als wichtigste oder „Haupttabelle“ der Abfrage angesehen wird. Eine weitere Tabelle wird durch **JOIN** und eine Bedingung in der **ON**-Klausel verknüpft.

Das obige Beispiel sieht dann so aus:

```
SELECT mi.Name,
       mi.Vorname,
       mi.Raum,
       ab.Ort
  FROM Mitarbeiter mi
       JOIN Abteilung ab
         ON mi.Abteilung_ID = ab.ID
 WHERE ab.Ort in ('Dortmund', 'Bochum')
 ORDER BY mi.Name, mi.Vorname;
```

Für die Verknüpfung der Tabellen werden folgende Bestandteile benötigt:

- In der FROM-Klausel steht eine der benötigten Tabellen.
- In der JOIN-Klausel steht jeweils eine weitere Tabelle.
- Die ON-Klausel enthält die Verknüpfungsbedingung „mi.Abteilung_ID = ab.ID“.
- Die WHERE-Klausel beschränkt sich auf die wirklich gewünschten Einschränkungen für die Ergebnismenge.

Ein Tabellen-Alias ist wiederum für alle Tabellen sinnvoll. In der Spaltenliste und auch zur Sortierung können alle Spalten aller Tabellen benutzt werden.

Vertiefte Erläuterungen sind unter Arbeiten mit JOIN zu finden.

Ausblick auf komplexe Abfragen

Das folgende Beispiel ist erheblich umfangreicher und geht über „Anfängerbedürfnisse“ weit hinaus. Es zeigt aber sehr schön, was alles mit SQL möglich ist:

Gesucht werden die Daten der Versicherungsnehmer im Jahr 2008, und zwar die Adresse, die Höhe des Gesamtschadens und die Anzahl der Schadensfälle.

```
select vn.Name,
       vn.Vorname,
       vn.Strasse,
       vn.Hausnummer as HNR,
       vn.PLZ,
       vn.Ort,
       SUM(sf.Schadenshoehe) as Gesamtschaden,
       COUNT(sf.ID) as Anzahl
  from Versicherungsnehmer vn
       join Versicherungsvertrag vv
         on vv.Versicherungsnehmer_ID = vn.ID
       join Fahrzeug fz
         on fz.ID = vv.Fahrzeug_ID
       join Zuordnung_SF_FZ zu
         on zu.Fahrzeug_ID = fz.ID
       join Schadensfall sf
         on sf.ID = zu.Schadensfall_ID
 where EXTRACT(YEAR from sf.Datum) = 2008
 group by vn.Name, vn.Vorname, vn.Strasse, vn.Hausnummer, vn.PLZ, vn.Ort
 order by Gesamtschaden, Anzahl;
```

Ausgabe

```
NAME                  VORNAME   STRASSE          HNR  PLZ    ORT            GESAMTSCHADEN  ANZAHL
--------------------  --------  ---------------  ---  -----  -------------  -------------  ------
Heckel Obsthandel GmbH          Gahlener Str.    40   46282  Dorsten             1.438,75       1
Antonius              Bernhard  Coesfelder Str.  23   45892  Gelsenkirchen       1.983,00       1
```

Hierbei kommen die Funktionen **SUM** (Summe) und **COUNT** (Anzahl) zum Einsatz. Diese können nur eingesetzt werden, wenn die Datenmenge richtig gruppiert wurde. Deshalb wird mit **GROUP BY** das Datenmaterial nach allen verbliebenen, zur Ausgabe vorgesehenen, Datenfeldern gruppiert.

Vertiefte Erläuterungen sind zu finden unter Funktionen sowie Gruppierungen.

Zusammenfassung

In diesem Kapitel lernten wir die Grundlagen eines SELECT-Befehls kennen:

- SELECT-Befehle werden zur Abfrage von Daten aus Datenbanken genutzt.
- Die auszugebenden Spalten können festgelegt werden, indem die Liste der Spalten zwischen den Bezeichnern SELECT und FROM angegeben wird.
- Mit DISTINCT werden identische Zeilen in der Ergebnismenge nur einmal ausgegeben.
- Die Ergebnismenge wird mittels der WHERE-Klausel eingegrenzt.
- Die WHERE-Klausel enthält logische Ausdrücke. Diese können mit AND und OR verknüpft werden.
- Mittels der ORDER BY-Klausel kann die Ergebnismenge sortiert werden.

Die Reihenfolge innerhalb eines SELECT-Befehls ist zu beachten. SELECT und FROM sind hierbei Pflicht, das abschließende Semikolon als Standard empfohlen. Alle anderen Klauseln sind optional.

Übungen

Hinweis: Der direkte Sprung zur jeweiligen Lösung funktioniert erst, wenn die Lösung sichtbar ist.

Bei den Übungen 2 ff. ist jeweils eine Abfrage zur Tabelle *Abteilung* zu erstellen.

Übung 1	**Pflichtangaben**	**Zur Lösung**

Welche Bestandteile eines SELECT-Befehls sind unbedingt erforderlich und können nicht weggelassen werden?

Übung 2	**Alle Angaben**	**Zur Lösung**

Geben Sie alle Informationen zu allen Abteilungen aus.

Übung 3	**Angaben mit Einschränkung**	**Zur Lösung**

Geben Sie alle Abteilungen aus, deren Standort *Bochum* ist.

Übung 4	**Angaben mit Einschränkungen**	**Zur Lösung**

Geben Sie alle Abteilungen aus, deren Standort *Bochum* oder *Essen* ist. Hierbei soll nur der Name der Abteilung ausgegeben werden.

Übung 5	Abfrage mit Sortierung	Zur Lösung

Geben Sie nur die Kurzbezeichnungen aller Abteilungen aus. Hierbei sollen die Abteilungen nach den Standorten sortiert werden.

Lösungen

Lösung zu Übung 1	Pflichtangaben	Zur Übung

SELECT, Spaltenliste oder '*', FROM, Tabellenname.

Lösung zu Übung 2	Alle Angaben	Zur Übung

```
select * from Abteilung;
```

Lösung zu Übung 3	Angaben mit Einschränkung	Zur Übung

```
select * from Abteilung
 where Ort = 'Bochum';
```

Lösung zu Übung 4	Angaben mit Einschränkungen	Zur Übung

```
select Bezeichnung from Abteilung
 where Ort = 'Bochum' or Ort = 'Essen';
```

Alternativ ist es auch so möglich:

```
select Bezeichnung from Abteilung
 where Ort in ('Bochum', 'Essen');
```

Lösung zu Übung 5	Abfrage mit Sortierung	Zur Übung

```
select Kuerzel from Abteilung
 order by Ort;
```

DML (2) – Daten speichern

Dieser Teilbereich der Data Manipulation Language (DML) behandelt die Befehle, mit denen die Inhalte der Datenbank geändert werden: Neuaufnahme, Änderung, Löschung.

Bitte beachten Sie, dass mit den Befehlen **INSERT, UPDATE, DELETE** (fast) immer nur Daten genau einer Tabelle bearbeitet werden können – anders als beim SELECT-Befehl, der Daten mehrerer Tabellen zusammenfassen kann.

INSERT – Daten einfügen

Der **INSERT**-Befehl dient dem Erstellen von neuen Datensätzen. Es gibt ihn in zwei Versionen – zum einen durch die Angabe einzelner Werte, zum anderen mit Hilfe eines SELECT-Befehls.

In beiden Versionen müssen die Datentypen der Werte zu den Datentypen der Spalten passen. Man sollte nicht versuchen, einer Spalte, die eine Zahl erwartet, eine Zeichenkette zuzuweisen. Man wird nur selten das Ergebnis erhalten, welches man erwartet. Das Kapitel Funktionen erläutert im Abschnitt „Konvertierungen“ Möglichkeiten, wie Werte implizit (also automatisch) oder explizit durch CAST oder CONVERT angepasst werden können.

Einzeln mit VALUES

Wenn ein einzelner Datensatz durch die Angabe seiner Werte gespeichert werden soll, gilt folgende Syntax:

```
INSERT INTO <tabellenname>
        [ ( <spaltenliste> ) ]
   VALUES ( <werteliste> )
   ;
```

Zu diesem Befehl gehören folgende Angaben:

- INSERT als Name des Befehls, INTO als feststehender Begriff
- <Tabellenname> als Name der Tabelle, die diesen Datensatz erhalten soll
- in Klammern ( ) gesetzt eine Liste von Spalten (Feldnamen), denen Werte zugewiesen werden
- der Begriff VALUES als Hinweis darauf, dass einzelne Werte angegeben werden
- in Klammern ( ) gesetzt eine Liste von Werten, die in den entsprechenden Spalten gespeichert werden sollen

Wenn eine Liste von Spalten fehlt, bedeutet das, dass alle Spalten dieser Tabelle in der Reihenfolge der Struktur mit Werten versehen werden müssen.

So wird (wie im Skript der Beispieldatenbank) ein Eintrag in der Tabelle *Mitarbeiter* gespeichert:

```
INSERT INTO Mitarbeiter
       ( Personalnummer , Name, Vorname,
         Telefon, Email, Raum, Ist_Leiter , Abteilung_ID , Geburtsdatum )
VALUES ( '20002', 'Schmitz', 'Michael',
         '0231/5556187' , 'michael.schmitz@unserefirma.de' , '212', 'N', 2, '1959-08-25' );
```

Wenn Sie diesen Befehl mit der Tabellenstruktur vergleichen, werden Sie feststellen:

- Die Spalte *ID* fehlt. Dieser Wert wird von der Datenbank automatisch vergeben.
- Die Spalte *Mobil* fehlt. In dieser Spalte wird folglich ein NULL-Wert gespeichert.
- Die Reihenfolge der Spalten weicht von der Tabellendefinition ab; das ist also durchaus möglich.

In der Beschreibung der Beispieldatenbank werden sehr viele Spalten als „Pflicht“ festgelegt. Folgender Befehl wird deshalb zurückgewiesen:

Fehler

```
INSERT INTO Mitarbeiter
       ( Personalnummer , Name, Vorname, Ist_Leiter , Abteilung_ID  )
VALUES ( '17999', 'Liebich', 'Andrea', 'N', 17);
```

Ausgabe

```
validation error for column GEBURTSDATUM, value "*** null ***".
```

Die Spalte *Geburtsdatum* darf laut Definition nicht NULL sein. Eine Angabe fehlt in diesem Befehl: das wird als NULL interpretiert, also mit einer Fehlermeldung quittiert.

Mengen mit SELECT

Wenn eine Menge von Datensätzen mit Hilfe eines SELECT-Befehls gespeichert werden soll, gilt folgende Syntax:

```
INSERT INTO <tabellenname>
        [ ( <spaltenliste>  ) ]
    SELECT  <select-Ausdruck>
   ;
```

Zu diesem Befehl gehören die folgenden Angaben:

- INSERT INTO <Tabellenname> (wie oben)
- in Klammern ( ) gesetzt eine Liste von Spalten (Feldnamen), sofern vorgesehen
- dazu ein vollständiger SELECT-Befehl, mit dem die passenden Inhalte geliefert werden

Da ein SELECT-Befehl auch ohne Bezug auf eine Tabelle nur mit konstanten Werten möglich ist, kann das obige Beispiel auch so formuliert werden:

```
INSERT INTO Mitarbeiter
       ( Personalnummer , Name, Vorname,
         Telefon, Email, Raum, Ist_Leiter , Abteilung_ID , Geburtsdatum  )
  select '20002', 'Schmitz', 'Michael',
         '0231/5556187' , 'michael.schmitz@unserefirma.de' , '212', 'N', 2, '1959-08-25'
 /* from rdb%database (bei Firebird) */  ;
 /* from dual         (bei Oracle)   */  ;
```

Hinweis: Firebird und Oracle kennen diese Kurzform des SELECT-Befehls nicht; dort ist die als Kommentar jeweils eingefügte FROM-Klausel erforderlich.

Wichtig ist diese Art des INSERT-Befehls, wenn neue Datensätze aus vorhandenen anderen Daten abgeleitet werden wie im Skript der Beispieldatenbank:

Für jeden Abteilungsleiter aus der Tabelle *Mitarbeiter* wird ein Eintrag in der Tabelle *Dienstwagen* gespeichert:

```
INSERT INTO Dienstwagen
          ( Kennzeichen, Farbe, Fahrzeugtyp_ID, Mitarbeiter_ID )
     SELECT 'DO-WB 42' || Abteilung_ID, 'elfenbein', 14, ID
       FROM Mitarbeiter
      WHERE Ist_Leiter = 'J';
```

Die Spalte *ID* wird automatisch zugewiesen. Alle anderen Spalten erhalten ausdrücklich Werte:

- Farbe und Fahrzeugtyp als Konstante
- dazu natürlich die ID des Mitarbeiters, dem der Dienstwagen zugeordnet wird
- und ein Kfz-Kennzeichen, das aus einem konstanten Teil mit der ID der Abteilung zusammengesetzt wird

Manche Datenbanken erlauben auch die Erstellung von Tabellen aus einem SELECT-Ausdruck wie bei MS-SQL (nächstes Beispiel) oder Teradata (anschließend):

```
SELECT [ ( <spaltenliste> ) ]
   INTO <tabellenname>
FROM <tabellenname>
```

```
 CREATE TABLE  <tabellennameA> AS
   SELECT [ ( <spaltenliste> ) ]
 FROM <tabellenname>
```

UPDATE – Daten ändern

Der **UPDATE**-Befehl dient zum Ändern einer Menge von Datensätzen in einer Tabelle:

```
UPDATE <Tabellenname>
   SET <Feldänderungen>
[WHERE <Bedingungsliste>];
```

Jede Änderung eines Feldes ist so einzutragen:

```
<Feldname> = <Wert>,
```

Zu diesem Befehl gehören die folgenden Angaben:

- UPDATE als Name des Befehls
- <Tabellenname> als Name der Tabelle, in der die Daten zu ändern sind
- SET als Anfang der Liste von Änderungen
- <Feldname> als Name der Spalte, die einen neuen Inhalt erhalten soll, dazu das Gleichheitszeichen und der <Wert> als neuer Inhalt
- ein Komma als Hinweis, dass ein weiteres Feld zu ändern ist; vor der WHERE-Klausel oder dem abschließenden Semikolon muss das Komma entfallen
- die WHERE-Klausel mit Bedingungen, welche Datensätze zu ändern sind: einer oder eine bestimmte Menge

Die Struktur der WHERE-Klausel ist identisch mit derjenigen beim SELECT-Befehl. Wenn alle Datensätze geändert werden sollen, kann die WHERE-Bedingung entfallen; aber beachten Sie unbedingt:

Ohne WHERE-Bedingung wird wirklich alles sofort geändert.

An den Beispielen ist zu sehen, dass die Änderung aller Datensätze nur selten sinnvoll ist und meistens mit WHERE-Bedingung gearbeitet wird.

Wie beim INSERT-Befehl muss der Datentyp eines Wertes zum Datentyp der Spalte passen. Beispiele:

Korrigiere die Schreibweise des Namens bei einem Mitarbeiter.

```
UPDATE Mitarbeiter
   SET Name = 'Mayer'
 WHERE Personalnummer = 20001;
```

Ändere nach einer Eingemeindung PLZ und Ortsname für alle betroffenen Adressen.

```
UPDATE Versicherungsnehmer
   SET Ort = 'Leipzig',
       PLZ = '04178'
 WHERE PLZ = '04430';
```

Erhöhe bei allen Schadensfällen die Schadenshöhe um 10 % (das ist natürlich keine sinnvolle Maßnahme):

```
UPDATE Schadensfall
   SET Schadenshoehe = Schadenshoehe * 1.1;
```

Berichtige das Geburtsdatum für einen Versicherungsnehmer:

```
update Versicherungsnehmer
   set Geburtsdatum = '14.03.1963'
 where Name = 'Zenep' and Geburtsdatum = '13.02.1963';
```

Ausgabe

```
0 row(s) affected.
```

Nanu, keine Zeilen wurden geändert? Bei diesem Befehl wurde zur Kontrolle, welcher Datensatz geändert werden sollte, nicht nur der Nachname, sondern auch das bisher notierte Geburtsdatum angegeben – und dieses war falsch.

Daran ist zu sehen, dass der UPDATE-Befehl tatsächlich eine Menge von Datensätzen ändert: je nach WHERE-Klausel null, einen, mehrere oder alle Zeilen der Tabelle.

DELETE – Daten löschen

Der **DELETE**-Befehl löscht eine Menge von Datensätzen in einer Tabelle:

```
DELETE FROM <Tabellenname>
    [ WHERE <Bedingungsliste> ] ;
```

Zu diesem Befehl gehören folgende Angaben:

- DELETE als Name des Befehls, FROM als feststehender Begriff
- <Tabellenname> als Name der Tabelle, aus der diese Datenmenge entfernt werden soll
- die WHERE-Klausel mit Bedingungen, welche Datensätze zu löschen sind: einer oder eine bestimmte Menge

Die Struktur der WHERE-Klausel ist identisch mit derjenigen beim SELECT-Befehl. Wenn alle Datensätze gelöscht werden sollen, kann die WHERE-Bedingung entfallen; aber beachten Sie unbedingt:

> **Ohne WHERE-Bedingung wird wirklich alles sofort gelöscht.**

Beispiele:

Der Mitarbeiter mit der Personalnummer 20001 ist ausgeschieden.

```
DELETE FROM Mitarbeiter
 WHERE Personalnummer = 20001;
```

Die Abteilung 1 wurde ausgelagert, alle Mitarbeiter gehören nicht mehr zum Unternehmen.

```
DELETE FROM Mitarbeiter
 WHERE Abteilung_ID = 1;
```

Dies leert den gesamten Inhalt der Tabelle, aber die Tabelle selbst bleibt mit ihrer Struktur erhalten.

```
DELETE FROM Schadensfall ;
```

Achtung: Dies löscht ohne weitere Rückfrage alle Schadensfälle. Ein solcher Befehl sollte unbedingt nur nach einer vorherigen Datensicherung ausgeführt werden. Auch der Versuch ist „strafbar“ und führt zum sofortigen Datenverlust.

TRUNCATE – Tabelle leeren

Wenn Sie entgegen den oben genannten Hinweisen wirklich alle Datensätze einer Tabelle löschen wollen, können Sie (soweit vorhanden) anstelle von DELETE den **TRUNCATE**-Befehl benutzen. Damit werden (ohne Verbindung mit WHERE) *immer* alle Datensätze gelöscht; dies geschieht schneller und einfacher, weil auf das

interne Änderungsprotokoll der Datenbank verzichtet wird.[1]

```
TRUNCATE TABLE Schadensfall;
```

Zusammenfassung

In diesem Kapitel lernten wir die SQL-Befehle kennen, mit denen der Datenbestand geändert wird:

- Mit INSERT + VALUES wird ein einzelner Datensatz eingefügt.
- Mit INSERT + SELECT wird eine Menge von Datensätzen mit Hilfe einer Abfrage eingefügt.
- Mit UPDATE wird eine Menge von Datensätzen geändert; die Menge wird durch WHERE festgelegt.
- Mit DELETE wird eine Menge von Datensätzen gelöscht; die Menge wird durch WHERE festgelegt.
- Mit TRUNCATE werden alle Datensätze einer Tabelle gelöscht.

Die WHERE-Bedingungen sind hier besonders wichtig, damit keine falschen Speicherungen erfolgen.

Übungen

Übung 1 **Daten einzeln einfügen** **Zur Lösung**

Welche Angaben werden benötigt, wenn ein einzelner Datensatz in der Datenbank gespeichert werden soll?

Übung 2 **Daten einzeln einfügen** **Zur Lösung**

Speichern Sie in der Tabelle *Mitarbeiter* einen neuen Datensatz und lassen Sie alle Spalten und Werte weg, die nicht benötigt werden.

Übung 3 **Daten einfügen** **Zur Lösung**

Begründen Sie, warum der Spalte *ID* beim Einfügen in der Regel kein Wert zugewiesen wird.

Übung 4 **Daten einfügen** **Zur Lösung**

Begründen Sie, warum die folgenden Befehle nicht ausgeführt werden können.

```
/* Mitarbeiter */
INSERT INTO Mitarbeiter
       ( ID, Personalnummer, Name, Vorname, Geburtsdatum, Ist_Leiter, Abteilung_ID )
values ( 4, 'PD-348', 'Çiçek', 'Yasemin', '23.08.1984', 'J', 9 );

/* Dienstwagen 1 */
INSERT INTO Dienstwagen
       ( Kennzeichen, Farbe, Fahrzeugtyp_ID, Mitarbeiter_ID )
values ( 'DO-UF 1234', null, null, null );

/* Dienstwagen 2 */
INSERT INTO Dienstwagen
       ( Kennzeichen, Farbe, Fahrzeugtyp_ID, Mitarbeiter_ID )
values ( 'HAM-AB 1234', 'rot', 7, null );
```

Übung 5 **Daten ändern und löschen** **Zur Lösung**

Warum gibt es selten UPDATE- oder DELETE-Befehle ohne eine WHERE-Klausel?

Übung 6	Daten ändern	Zur Lösung

Schreiben Sie einen SQL-Befehl für folgende Änderung: Alle Mitarbeiter, die bisher noch keinen Mobil-Anschluss hatten, sollen unter einer einheitlichen Nummer erreichbar sein.

Lösungen

Lösung zu Übung 1	Daten einzeln einfügen	Zur Übung

- der INSERT-Befehl selbst
- INTO mit Angabe der Tabelle
- bei Bedarf in Klammern die Liste der Spalten, die mit Werten versehen werden
- VALUES zusammen mit (in Klammern) der Liste der zugeordneten Werte

Lösung zu Übung 2	Daten einzeln einfügen	Zur Übung

Beispielsweise so:

```
insert into Mitarbeiter
       ( Personalnummer, Name, Vorname, Geburtsdatum, Ist_Leiter, Abteilung_ID )
values ( 'PD-348', 'Çiçek', 'Yasemin', '23.08.1984', 'J', 9 );
```

Lösung zu Übung 3	Daten einfügen	Zur Übung

Dieser soll von der Datenbank automatisch zugewiesen werden; er muss deshalb weggelassen oder mit NULL vorgegeben werden.

Lösung zu Übung 4	Daten einfügen	Zur Übung

- Mitarbeiter: Diese ID ist schon vergeben; sie muss aber eindeutig sein.
- Dienstwagen 1: Der Fahrzeugtyp ist eine Pflicht-Angabe; die *Fahrzeugtyp_ID* darf nicht null sein.
- Dienstwagen 2: Das Kennzeichen ist zu lang; es darf maximal eine Länge von 10 Zeichen haben.

Lösung zu Übung 5	Daten ändern und löschen	Zur Übung

Dies würde die gleiche Änderung bzw. die Löschung für alle Datensätze ausführen; das ist selten sinnvoll bzw. gewünscht.

Lösung zu Übung 6	Daten ändern	Zur Übung

```
update Mitarbeiter
   set Mobil = '(0177) 44 55 66 77'
 where Mobil is null or Mobil = '';
```

Siehe auch

1. Wegen solcher technischen Gründe könnte TRUNCATE auch als DDL-Befehl angesehen werden. Aus der Sicht des Anwenders gehört es zu DML – siehe auch die SQL-Dokumente 20nn (http://www.wiscorp.com/sql20nn.zip). Weitere Gesichtspunkte stehen auf der Diskussionsseite.

DDL – Struktur der Datenbank

Mit den Befehlen der Data Definition Language (DDL) wird die Struktur der Datenbank gesteuert. Diese Aufgaben gehen über eine Einführung in SQL hinaus. Hier werden deshalb nur einige grundlegende Informationen und Beispiele behandelt, für welche Objekte einer Datenbank diese Befehle verwendet werden.

Am Ende des Buches werden einige Erweiterungen behandelt:

- DDL – Einzelheiten
- Änderung der Datenbankstruktur
- SQL-Programmierung

Bitte beachten Sie, dass ein Benutzer vor allem bei DDL-Befehlen über die entsprechenden Rechte verfügen muss, siehe DCL – Zugriffsrechte.

Allgemeine Syntax

Die **DDL-Befehle** sind grundsätzlich so aufgebaut:

```
BEFEHL OBJEKTTYP <Objektname> [<weitere Angaben>]
```

CREATE

CREATE erzeugt ein Datenobjekt, zum Beispiel eine Datentabelle oder gar eine Datenbank.

ALTER

Mit ALTER kann das Objekt, z. B. die Tabelle, auch wieder geändert werden:

In neueren Versionen gibt es auch den gemeinsamen Aufruf (unterschiedlich je nach DBMS):

- CREATE OR ALTER
- CREATE OR REPLACE
- RECREATE

Das DBMS entscheidet dann selbst: Wenn das Objekt schon existiert, wird es geändert, andernfalls erzeugt.

DROP

Mit DROP kann das Objekt, z. B. eine Tabelle wieder gelöscht werden:

Hauptteile der Datenbank

DATABASE – die Datenbank selbst

Der Befehl zum Erstellen einer Datenbank lautet:

```
CREATE DATABASE <Dateiname> [ <Optionen> ] ;
```

Der <Dateiname> ist meistens ein vollständiger Name einschließlich Pfad; in einer solchen Datei werden alle Teile der Datenbank zusammengefasst. Zu den <Optionen> gehören z. B. der Benutzername des Eigentümers der Datenbank mit seinem Passwort, der Zeichensatz mit Angaben zur Standardsortierung, die Aufteilung in eine oder mehrere Dateien usw.

Jedes DBMS bietet sehr verschiedene Optionen; wir können hier keine Gemeinsamkeiten vorstellen und müssen deshalb ganz auf Beispiele verzichten.

Wegen der vielen Möglichkeiten ist zu empfehlen, dass eine Datenbank nicht per SQL-Befehl, sondern innerhalb einer Benutzeroberfläche erstellt wird.

Mit **ALTER DATABASE** werden die Optionen geändert, mit **DROP DATABASE** wird die Datenbank gelöscht. *Diese Befehle kennt nicht jedes DBMS.*

TABLE – eine einzelne Tabelle

CREATE TABLE

Um eine **Tabelle** zu erzeugen, sind wesentlich konkretere umfangreiche Angaben nötig.

```
CREATE TABLE <Tabellenname>
   (     <Spaltenliste>
     [ , <Zusatzangaben> ]
   );
```

Zum Erzeugen einer Tabelle werden folgende Angaben benutzt:

- der **Name** der Tabelle, mit dem die Daten über die DML-Befehle gespeichert und abgerufen werden
- die Liste der **Spalten** (Felder), und zwar vor allem mit dem jeweiligen Datentyp
- Angaben wie der **Primärschlüssel** (PRIMARY KEY, PK) oder weitere Indizes

Jede Spalte und Zusatzangabe wird mit einem Komma abgeschlossen; dieses entfällt vor der schließenden Klammer. Die Zusatzangaben werden häufig nicht sofort festgelegt, sondern durch anschließende ALTER TABLE-Befehle; sie werden deshalb weiter unten besprochen.

In der Beispieldatenbank wird eine Tabelle so erzeugt:

MySQL-Version

```
create table Dienstwagen
      ( ID              integer      not null auto_increment  primary key,
        Kennzeichen     varchar(30) not null,
        Farbe           varchar(30),
        Fahrzeugtyp_ID  integer      not null,
        Mitarbeiter_ID  integer
      );
```

Die einzelnen Spalten berücksichtigen mit ihren Festlegungen unterschiedliche Anforderungen:

- ID ist eine ganze Zahl, darf nicht NULL sein, wird automatisch hochgezählt und dient dadurch gleichzeitig als Primärschlüssel.
- Das Kennzeichen ist eine Zeichenkette von variabler Länge (maximal 30 Zeichen), die unbedingt erforderlich ist.
- Die Farbe ist ebenfalls eine Zeichenkette, deren Angabe entfallen kann.

- Für den Fahrzeugtyp wird dessen ID benötigt, wie er in der Tabelle *Fahrzeugtyp* gespeichert ist; diese Angabe muss sein – ein „unbekannter“ Fahrzeugtyp macht bei einem Dienstwagen keinen Sinn.
- Für den Mitarbeiter, dem ein Dienstwagen zugeordnet ist, wird dessen ID aus der Tabelle *Mitarbeiter* benötigt. Dieser Wert kann entfallen, wenn es sich nicht um einen „persönlichen“ Dienstwagen handelt.

ALTER TABLE

Die Struktur einer Tabelle wird wie folgt geändert:

```
ALTER TABLE <Aufgabe> <Zusatzangaben>
```

Mit der Aufgabe **ADD CONSTRAINT** wird eine interne Einschränkung – Constraint genannt – hinzugefügt:

Ein Primärschlüssel kann auch nachträglich festgelegt werden, z. B. wie folgt:

Firebird-Version

```
ALTER TABLE Dienstwagen
  ADD CONSTRAINT Dienstwagen_PK PRIMARY KEY (ID);
```

Die Einschränkung bekommt den Namen *Dienstwagen_PK* und legt fest, dass es sich dabei um den PRIMARY KEY unter Verwendung der Spalte *ID* handelt.

In der Tabelle *Mitarbeiter* muss auch die *Personalnummer* eindeutig sein (zusätzlich zur *ID*, die als PK sowieso eindeutig ist):

```
ALTER TABLE Mitarbeiter
  ADD CONSTRAINT Mitarbeiter_PersNr  UNIQUE (Personalnummer );
```

In der Tabelle *Zuordnung_SF_FZ* – Verknüpfung zwischen den Schadensfällen und den Fahrzeugen – wird ein Feld für sehr lange Texte eingefügt:

```
alter Table Zuordnung_SF_FZ
  add Beschreibung blob;
```

Mit **ALTER ... DROP Beschreibung** kann dieses Feld auch wieder gelöscht werden.

DROP TABLE

Damit wird eine Tabelle mit allen Daten gelöscht (diese Tabelle gab es in einer früheren Version der Beispieldatenbank):

```
DROP TABLE ZUORD_VNE_SCF ;
```

Warnung: Dies löscht die Tabelle einschließlich aller Daten unwiderruflich!

USER – Benutzer

Auf diese Weise wird ein neuer Benutzer für die Arbeit mit der aktuellen Datenbank registriert.

MySQL-Version

```
CREATE USER Hans_Dampf IDENTIFIED BY 'cH4y37X1P';
```

Dieser Befehl richtet einen neuen Benutzer mit dem Namen *Hans_Dampf* ein, der sich mit dem Passwort 'cH4y37X1P' anmelden muss. – Jedes DBMS kennt eigene Regeln und weitere Optionen für die Zuordnung des Passworts und die Verwendung von Anführungszeichen.

Ergänzungen zu Tabellen

Weitere Objekte in der Datenbank erleichtern die Arbeit mit Tabellen.

VIEW – besondere Ansichten

Eine VIEW ist eine spezielle Sicht auf eine oder mehrere Tabellen. Für den Anwender sieht es wie eine eigene Tabelle aus; es handelt sich aber „nur“ um eine fest gespeicherte Abfrage, die immer wieder in der gleichen Form benutzt und ausgeführt wird. Bitte beachten Sie: Nur die Abfrage wird fest gespeichert, nicht das Ergebnis; dieses muss bei jedem neuen Aufruf nach den aktuellen Daten neu erstellt werden.

Einzelheiten werden unter Erstellen von Views behandelt.

INDEX – Datenzugriff beschleunigen

Ein Index beschleunigt die Suche nach Datensätzen. Um beispielsweise in der Tabelle *Versicherungsnehmer* nach dem Namen „Schulze“ zu suchen, würde es zu lange dauern, wenn das DBMS alle Zeilen durchgehen müsste, bis es auf diesen Namen träfe. Stattdessen wird ein Index angelegt (ähnlich wie in einem Telefon- oder Wörterbuch), sodass schnell alle passenden Datensätze gefunden werden.

So wird ein Index mit der Bezeichnung *Versicherungsnehmer_Name* für die Kombination „Name, Vorname“ angelegt:

```
create index Versicherungsnehmer_Name
    on Versicherungsnehmer  (Name, Vorname);
```

Es ist dringend zu empfehlen, dass Indizes für alle Spalten bzw. Kombinationen von Spalten angelegt werden, die immer wieder zum Suchen benutzt werden.

Weitere Einzelheiten werden unter DDL – Einzelheiten behandelt.

IDENTITY – auch ein automatischer Zähler

Anstelle von AUTO_INCREMENT verwendet MS-SQL diese Erweiterung für die automatische Nummerierung neuer Datensätze:

MS-SQL-Version

```
CREATE TABLE Fahrzeug
    (ID              INTEGER        NOT NULL IDENTITY(1,1),
     Kennzeichen     VARCHAR(10)    NOT NULL,
     Farbe           VARCHAR(30),
     Fahrzeugtyp_ID  INTEGER        NOT NULL,
     CONSTRAINT Fahrzeug_PK  PRIMARY KEY (ID)
    );
```

Der erste Parameter bezeichnet den Startwert, der zweite Parameter die Schrittweite zum nächsten ID-Wert.

SEQUENCE – Ersatz für automatischen Zähler

Wenn das DBMS für Spalten keine automatische Zählung kennt (Firebird, Oracle), steht dies als Ersatz zur Verfügung.

Firebird-Version

```
/* zuerst die Folge definieren */
CREATE SEQUENCE Versicherungsnehmer_ID ;
/* dann den Startwert festlegen */
ALTER  SEQUENCE Versicherungsnehmer_ID  RESTART WITH 1;
/* und im Trigger (s.u.) ähnlich wie eine Funktion benutzen */
NEXT VALUE FOR Versicherungsnehmer_ID
```

Während der automatische Zähler, der durch AUTO_INCREMENT eingerichtet wird, genau zu der betreffenden Tabelle gehört, bezieht sich eine „Sequenz“ nicht auf eine einzelne Tabelle, sondern auf die gesamte Datenbank. Es ist ohne weiteres möglich, eine einzige Sequenz `AllMyIDs` zu definieren und die neue ID einer jeden Tabelle daraus abzuleiten. Dies ist durchaus sinnvoll, weil die *ID* als Primärschlüssel sowieso keine inhaltliche Bedeutung haben darf, sondern nur ein fortlaufender Zähler ist. In der Beispieldatenbank benutzen wir getrennte Sequenzen, weil sie für die verschiedenen DBMS „ähnlich“ aussehen soll.

Oracle arbeitet (natürlich) mit anderer Syntax (mit mehr Möglichkeiten) und benutzt dann NEXTVAL.

Programmieren mit SQL

Die Funktionalität einer SQL-Datenbank kann erweitert werden, und zwar auch mit Bestandteilen einer „vollwertigen“ Programmiersprache, z. B. Schleifen und IF-Abfragen.

Dies wird unter Programmierung behandelt; dabei gibt es relativ wenig Gemeinsamkeiten zwischen den DBMS.

FUNCTION – Benutzerdefinierte Funktionen

Eigene Funktionen ergänzen die (internen) Skalarfunktionen des DBMS.

PROCEDURE – Gespeicherte Prozeduren

Eine Prozedur – gespeicherte Prozedur, engl. StoredProcedure (SP) – ist vorgesehen für Arbeitsabläufe, die „immer wiederkehrende“ Arbeiten ausführen sollen. Es gibt sie mit und ohne Argumente und Rückgabewerte.

TRIGGER – Ereignisse beim Speichern

Ein Trigger ist ein Arbeitsablauf, der automatisch beim Speichern in einer Tabelle ausgeführt wird. Dies dient Eingabeprüfungen oder zusätzlichen Maßnahmen; beispielsweise holt sich Firebird durch einen Trigger den NEXT VALUE einer SEQUENCE (siehe oben).

TRIGGER werden unterschieden nach INSERT-, UPDATE- oder DELETE-Befehlen und können vor oder nach dem Speichern sowie in einer bestimmten Reihenfolge ausgeführt werden.

Zusammenfassung

In diesem Kapitel lernten wir die Grundbegriffe einer Datenbankstruktur kennen:

- Die DATABASE selbst sowie TABLE sind die wichtigsten Objekte, ein USER arbeitet damit.
- Eine VIEW ist eine gespeicherte Abfrage, die wie eine Tabelle abgerufen wird.
- Ein INDEX beschleunigt den Datenzugriff.

Darüber hinaus wurde auf weitere Möglichkeiten wie SEQUENCE, Funktionen, Prozeduren und Trigger hingewiesen.

Übungen

Übung 1	**Objekte bearbeiten**	**Zur Lösung**

Welche der folgenden SQL-Befehle enthalten Fehler?

1. `CREATE DATABASE C:\DBFILES\employee.gdb DEFAULT CHARACTER SET UTF8;`
2. `DROP DATABASE;`
3. `CREATE TABLE Person ( ID PRIMARY KEY, Name VARCHAR(30), Vorname VARCHAR(30) );`
4. `ALTER TABLE ADD UNIQUE KEY (Name);`

Übung 2	**Tabelle erstellen**	**Zur Lösung**

Auf welchen zwei Wegen kann der Primärschlüssel (Primary Key, PK) einer Tabelle festgelegt werden? *Ein weiterer Weg ist ebenfalls üblich, aber noch nicht erwähnt worden.*

Übung 3	**Tabelle ändern**	**Zur Lösung**

Skizzieren Sie einen Befehl, mit dem die Tabelle *Mitarbeiter* um Felder für die Anschrift erweitert werden kann.

Übung 4	**Tabellen**	**Zur Lösung**

Worin unterscheiden sich TABLE und VIEW in erster Linie?

Lösungen

Lösung zu Übung 1	Objekte bearbeiten	Zur Übung

Bei 2. fehlt der Name der Datenbank.
Bei 3. fehlt der Datentyp zur Spalte *ID*.
Bei 4. fehlt der Name der Tabelle, die geändert werden soll.

Lösung zu Übung 2	Tabelle erstellen	Zur Übung

1. Im CREATE TABLE-Befehl zusammen mit der Spalte, die als PK benutzt wird, z. B.:
   `ID INTEGER NOT NULL AUTO_INCREMENT PRIMARY KEY,`
2. Durch einen ALTER TABLE-Befehl, der den PK hinzufügt:
   `ALTER TABLE Dienstwagen ADD PRIMARY KEY (ID);`

Lösung zu Übung 3	Tabelle ändern	Zur Übung

```
alter table Mitarbeiter
  add PLZ CHAR(5),
  add Ort VARCHAR(24),
  add Strasse VARCHAR(24),
  add Hausnummer  VARCHAR(10);
```

Lösung zu Übung 4	Tabellen	Zur Übung

Eine TABLE (Tabelle) ist real in der Datenbank gespeichert. Eine VIEW (Sichttabelle) ist eine Sicht auf eine oder mehrere tatsächlich vorhandene Tabellen; sie enthält eine Abfrage auf diese Tabellen, die als Abfrage in der Datenbank gespeichert ist, aber für den Anwender wie eine eigene Tabelle aussieht.

Siehe auch

Bei diesem Kapitel sind die folgenden Erläuterungen zu beachten:

- Datentypen

Manche Themen werden in den folgenden Kapiteln genauer behandelt:

- Eigene Funktionen als Ergänzung zu denen, die in Funktionen und Funktionen (2) behandelt werden.
- Prozeduren
- Trigger

TCL – Ablaufsteuerung

Dieses Kapitel gibt eine kurze Einführung in die Transaction Control Language (TCL). Deren Befehle sorgen für die Datensicherheit innerhalb einer Datenbank.

MySQL verfolgt eine „offenere“ Philosophie und arbeitet neben Transaktionen auch mit anderen Sicherungsmaßnahmen. Der Ersteller einer Datenbank muss sich für ein Verfahren entscheiden, kann aber auch danach noch variieren.

Beispiele

Eine SQL-Datenbank speichert Daten in der Regel in verschiedenen Tabellen. Wenn Daten geändert werden sollen, müssen folglich mehrere Tabellen simultan verändert werden. Dazu wäre es nötig, dass die betreffenden Befehle immer gleichzeitig ausgeführt werden. Da der Computer Befehle nur nacheinander ausführen kann, muss sichergestellt sein, dass nicht der eine Befehl ausgeführt wird, während der andere Befehl scheitert.

- Zu einer Überweisung bei der Bank gehören immer zwei Buchungen: die Gutschrift auf dem einen und die Lastschrift auf dem anderen Konto; häufig gehören die Konten zu verschiedenen Banken. Es wäre völlig unerträglich, wenn die Gutschrift ausgeführt würde und die (externe) Lastschrift nicht, weil in diesem Moment die Datenleitung unterbrochen wird.

- Wenn in dem Versicherungsunternehmen der Beispieldatenbank ein neuer Vertrag abgeschlossen wird, gehören dazu mehrere INSERT-Befehle, und zwar in die Tabellen *Fahrzeug*, *Versicherungsnehmer*, *Versicherungsvertrag*. Zuerst müssen Fahrzeug und Versicherungsnehmer gespeichert werden; aber wenn das Speichern des Vertrags „schiefgeht“, hängen die beiden anderen Datensätze nutzlos in der Datenbank herum.

- Wenn dort eine Abteilung ausgelagert wird, werden alle ihre Mitarbeiter gestrichen, weil sie nicht mehr zum Unternehmen gehören. Wie soll verfahren werden, wenn nur ein Teil der DELETE-Befehle erfolgreich war?

- Eine Menge einzelner Befehle (z. B. 1000 INSERTs innerhalb einer Schleife) dauert „ewig lange“.

Solche Probleme können nicht nur durch die Hardware entstehen, sondern auch dadurch, dass parallel andere Nutzer denselben Datenbestand ändern wollen.

Transaktionen

Alle solche Ungereimtheiten werden vermieden, indem SQL-Befehle in **Transaktionen** zusammengefasst und ausgeführt werden:

- Entweder alle Befehle können ausgeführt werden. Dann wird die Transaktion bestätigt und erfolgreich abgeschlossen.
- Oder (mindestens) ein Befehl kann nicht ausgeführt werden. Dann wird die Transaktion für ungültig erklärt; alle Befehle werden rückgängig gemacht.
- Auch das Problem mit der langen Arbeitszeit von 1000 INSERTs wird vermieden, wenn das DBMS nicht jeden Befehl einzeln prüft und bestätigt, sondern erst alle 1000 am Schluss „am Stück“.

Es gibt verschiedene Arten von Transaktionen. Diese hängen vom DBMS und dessen Version, der Hardware (Einzel- oder Mehrplatzsystem) und dem Datenzugriff (direkt oder über Anwenderprogramme) ab.

- Wenn die Datenbank auf AUTOCOMMIT eingestellt ist, wird jeder SQL-Befehl als einzelne Transaktion behandelt und sofort gültig. (Das wäre die Situation mit der langen Arbeitszeit von 1000 INSERTs.)

- Wenn ein Stapel von Befehlen mit COMMIT bestätigt oder mit ROLLBACK verworfen wird, dann wird mit dem nächsten Befehl implizit eine neue Transaktion begonnen.

- Mit einem ausdrücklichen TRANSACTION-Befehl wird explizit eine neue Transaktion begonnen:

```
BEGIN TRANSACTION <TName>;   /* bei MS-SQL   */
SET   TRANSACTION <TName>;   /* bei Firebird */
START TRANSACTION;           /* bei MySQL    */
START TRANSACTION;          /* PostgreSQL 9.6.3 */
```

Eine Transaktion kann mit einem Namen versehen werden. Dies ist vor allem dann nützlich, wenn Transaktionen geschachtelt werden. Außerdem gibt es je nach DBMS noch viele weitere Optionen, mit denen eine Transaktion detailliert gesteuert werden kann.

- Eine Transaktion, die implizit oder explizit begonnen wird, ist ausdrücklich abzuschließen durch COMMIT oder ROLLBACK. Wenn dies vergessen wird, wird die Transaktion erst dadurch beendet, dass die Verbindung mit der Datenbank geschlossen wird.

Transaktion erfolgreich beenden

Eine Transaktion wird mit einem der folgenden Befehle erfolgreich abgeschlossen und beendet:

```
COMMIT [ TRANSACTION | WORK ]  <TName>;
```

Die genaue Schreibweise und Varianten müssen in der DBMS-Dokumentation nachgelesen werden.

Dieser Befehl bestätigt alle vorangegangenen Befehle einer Transaktion und sorgt dafür, dass sie „am Stück“ gespeichert werden.

Auszug aus dem Skript zum Erstellen der Beispieldatenbank:

Firebird-Quelltext

```
COMMIT;
/* damit wird die vorherige Transaktion abgeschlossen und implizit eine neue Transaktion gestartet */

 INSERT INTO Dienstwagen (Kennzeichen, Farbe, Fahrzeugtyp_ID, Mitarbeiter_ID)
   VALUES ('DO-WB 111', 'elfenbein', 16, NULL);
 INSERT INTO Dienstwagen (Kennzeichen, Farbe, Fahrzeugtyp_ID, Mitarbeiter_ID)
   SELECT 'DO-WB 3' || Abteilung_ID || SUBSTRING(Personalnummer FROM 5 FOR 1),
          'gelb', SUBSTRING(Personalnummer FROM 5 FOR 1), ID
     FROM Mitarbeiter
    WHERE Abteilung_ID IN (5, 8)
      AND Ist_Leiter = 'N';

/* damit wird diese Transaktion abgeschlossen */
COMMIT;
```

Sicherungspunkte

Mit dem folgenden Befehl wird eine Transaktion in „sichere“ Abschnitte geteilt:

```
SAVEPOINT <SPName>;
```

Bis zu diesem Sicherungspunkt werden die Befehle auch dann als gültig abgeschlossen, wenn die Transaktion am Ende für ungültig erklärt wird.

Transaktion rückgängig machen

Eine Transaktion wird wie folgt für ungültig erklärt:

```
ROLLBACK [ TRANSACTION | WORK ]  <TName> [ TO <SPName> ] ;
```

Damit werden alle Befehle der Transaktion <TName> für ungültig erklärt und rückgängig gemacht. Sofern ein Sicherungspunkt <SPName> angegeben ist, werden die Befehle bis zu diesem Sicherungspunkt für gültig erklärt und erst alle folgenden für ungültig.

Die genaue Schreibweise und Varianten müssen in der DBMS-Dokumentation nachgelesen werden.

Mit dem folgenden Beispiel werden zu Testzwecken einige Daten geändert und abgerufen; abschließend werden die Änderungen rückgängig gemacht.

```
update Dienstwagen
   set Farbe = 'goldgelb/violett gestreift'
 where ID >= 14;
select * from Dienstwagen;
ROLLBACK;
```

Zusammenfassung

In diesem Kapitel lernten Sie die Grundbegriffe von Transaktionen kennen:

- Eine Transaktion wird implizit oder explizit begonnen.
- Eine Transaktion wird mit einem ausdrücklichen Befehl (oder durch Ende der Verbindung) abgeschlossen.
- Mit COMMIT wird eine Transaktion erfolgreich abgeschlossen; die Daten werden abschließend gespeichert.
- Mit ROLLBACK werden die Änderungen verworfen, ggf. ab einem bestimmten SAVEPOINT.

Übungen

Übung 1	Zusammengehörende Befehle	Zur Lösung

Skizzieren Sie die Befehle, die gemeinsam ausgeführt werden müssen, wenn ein neues Fahrzeug mit einem noch nicht registrierten Fahrzeugtyp und Fahrzeughersteller gespeichert werden soll.

Übung 2	Zusammengehörende Befehle	Zur Lösung

Skizzieren Sie die Befehle, die gemeinsam ausgeführt werden müssen, wenn ein neuer Schadensfall ohne weitere beteiligte Fahrzeuge registriert werden soll.

Übung 3	Zusammengehörende Befehle	Zur Lösung

Skizzieren Sie die Befehle, die gemeinsam ausgeführt werden müssen, wenn ein neuer Schaden durch einen „Eigenen Kunden“ gemeldet wird; dabei sollen ein zweiter „Eigener Kunde“ sowie ein „Fremdkunde“ einer bisher nicht gespeicherten Versicherungsgesellschaft beteiligt sein. Erwähnen Sie dabei auch den Inhalt des betreffenden Eintrags.

Übung 4	Transaktionen	Zur Lösung

Welche der folgenden Maßnahmen starten immer eine neue Transaktion, welche unter Umständen, welche sind unzulässig?

1. das Herstellen der Verbindung zur Datenbank

2. der Befehl `START TRANSACTION;`
3. der Befehl `SET TRANSACTION ACTIVE;`
4. der Befehl `SAVEPOINT <name>` wird ausgeführt
5. der Befehl `ROLLBACK` wird ausgeführt

Übung 5	Transaktionen	Zur Lösung

Welche der folgenden Maßnahmen beenden immer eine Transaktion, welche unter Umständen, welche sind unzulässig?

1. das Schließen der Verbindung zur Datenbank
2. der Befehl `END TRANSACTION;`
3. der Befehl `SET TRANSACTION INACTIVE;`
4. der Befehl `SAVEPOINT <name>` wird ausgeführt
5. der Befehl `ROLLBACK` wird ausgeführt

Lösungen

Lösung zu Übung 1	Zusammengehörende Befehle	Zur Übung

- `INSERT INTO Fahrzeughersteller`
- `INSERT INTO Fahrzeugtyp`
- `INSERT INTO Fahrzeug`

Lösung zu Übung 2	Zusammengehörende Befehle	Zur Übung

- `INSERT INTO Schadensfall`
- `INSERT INTO Zuordnung_SF_FZ`

Lösung zu Übung 3	Zusammengehörende Befehle	Zur Übung

- `INSERT INTO Schadensfall`
- `INSERT INTO Versicherungsgesellschaft /* für den zusätzlichen Fremdkunden */`
- `INSERT INTO Versicherungsnehmer /* für den zusätzlichen Fremdkunden */`
- `INSERT INTO Fahrzeug /* für den zusätzlichen Fremdkunden */`
- `INSERT INTO Zuordnung_SF_FZ /* für den zusätzlichen Fremdkunden */`
- `INSERT INTO Zuordnung_SF_FZ /* für den beteiligten Eigenen Kunden */`
- `INSERT INTO Zuordnung_SF_FZ /* für den Eigenen Kunden laut Schadensmeldung */`

Lösung zu Übung 4	Transaktionen	Zur Übung

1. ja, sofern `AUTOCOMMIT` festgelegt ist
2. ja, aber nur, wenn das DBMS diese Variante vorsieht (wie MySQL)
3. ja, aber nur, wenn das DBMS diese Variante vorsieht (wie Firebird); denn das Wort „ACTIVE“ wird nicht als Schlüsselwort, sondern als Name der Transaction interpretiert
4. nein, das ist nur ein Sicherungspunkt *innerhalb* einer Transaktion
5. ja, nämlich implizit für die folgenden Befehle

Lösung zu Übung 5	Transaktionen	Zur Übung

1. ja, und zwar immer
2. nein, diesen Befehl gibt es nicht
3. nein, dieser Befehl ist unzulässig; wenn das DBMS diese Variante kennt (wie Firebird), dann ist es der Start einer Transaktion namens „INACTIVE“

4. teilweise, das ist ein Sicherungspunkt *innerhalb* einer Transaktion und bestätigt die bisherigen Befehle, setzen aber dieselbe Transaktion fort
5. ja, nämlich explizit durch Widerruf aller Befehle

DCL – Zugriffsrechte

Eine „vollwertige“ SQL-Datenbank enthält umfassende Regelungen über die Vergabe von Rechten für den Zugriff auf Objekte (Tabellen, einzelne Felder, interne Funktionen usw.). Am Anfang stehen diese Rechte nur dem Ersteller der Datenbank und dem System-Administrator zu. Andere Benutzer müssen ausdrücklich zu einzelnen Handlungen ermächtigt werden.

Da es sich dabei nicht um Maßnahmen für Einsteiger handelt, beschränken wir uns auf ein paar Beispiele.

GRANT – Zugriff gewähren

Der Benutzer Herr_Mueller darf Abfragen auf die Tabelle *Abteilungen* ausführen.

```
GRANT SELECT ON Abteilung TO Herr_Mueller
```

Die Benutzerin Frau_Schulze darf Daten in der Tabelle *Abteilungen* ändern.

```
GRANT UPDATE ON Abteilung TO Frau_Schulze
```

REVOKE – Zugriff verweigern

Herr_Mueller darf künftig keine solche Abfragen mehr ausführen.

```
REVOKE SELECT ON Abteilung FROM Herr_Mueller
```

Datentypen

SQL kennt verschiedene Arten von Datentypen: vordefinierte, konstruierte und benutzerdefinierte. Diese Arten und ihre Verwendungen werden in diesem Abschnitt erklärt.

Vordefinierte Datentypen

Laut SQL-Standard sind die folgenden Datentypen vordefiniert. Die fettgedruckten Begriffe sind die entsprechenden reservierten Schlüsselwörter.

Achtung
Bei allen Datentypen weichen die SQL-Dialekte mehr oder weniger vom Standard ab.

Dies betrifft vor allem die folgenden Punkte:

- bei den möglichen Werten, z. B. dem maximalen Wert einer ganzen Zahl oder der Länge einer Zeichenkette
- bei der Bearbeitung einzelner Typen; z. B. kennt Firebird noch nicht sehr lange den BOOLEAN-Typ und musste mit dem Ersatz „ganze Zahl gleich 1“ leben
- bei der Art der Werte, z. B. ob Datum und Zeit getrennt oder gemeinsam gespeichert sind

Zeichen und Zeichenketten

Es gibt Zeichenketten mit fester, variabler und sehr großer Länge.

- **CHARACTER(n), CHAR(n)**

Hierbei handelt es sich um Zeichenketten mit fester Länge von genau n Zeichen. Für ein einzelnes Zeichen muss die Länge `(1)` nicht angegeben werden.

- **CHARACTER VARYING(n), VARCHAR(n)**

Dies sind Zeichenketten mit variabler Länge bei maximal n Zeichen.

- **CHARACTER LARGE OBJECT**

Hierbei handelt es sich um beliebig große Zeichenketten. Diese Variante ist relativ umständlich zu nutzen; sie wird vorwiegend für die Speicherung ganzer Textdateien verwendet.

Zu allen diesen Varianten gibt es auch die Festlegung eines nationalen Zeichensatzes durch **NATIONAL CHARACTER** bzw. **NCHAR** oder **NATIONAL CHARACTER VARYING** bzw. **NVARCHAR**. Erläuterungen dazu siehe unten im Abschnitt über Zeichensätze.

Die maximale Länge von festen und variablen Zeichenketten hängt vom DBMS ab. In früheren Versionen betrug sie oft nur 255, heute sind 32767 verbreitet.

Die maximale Feldlänge bei Zeichenketten ist für Eingabe, Ausgabe und interne Speicherung wichtig. Die DBMS verhalten sich unterschiedlich, ob am Anfang oder Ende stehende Leerzeichen gespeichert oder entfernt werden. Wichtig ist, dass bei fester Feldlänge ein gelesener Wert immer mit genau dieser Anzahl von Zeichen zurückgegeben wird und bei Bedarf rechts mit Leerzeichen aufgefüllt wird.

In der Praxis sind folgende Gesichtspunkte von Bedeutung:

- Für **indizierte Felder** (also Spalten, denen ein Index zugeordnet wird) sind feste Längen vorzuziehen; beachten Sie aber die nächsten Hinweise.
- Als **CHAR(n)**, also mit fester Länge, sind Felder vorzusehen, deren Länge bei der überwiegenden Zahl der Datensätze konstant ist. Beispiel: die deutschen Postleitzahlen mit CHAR(5).
- Als **VARCHAR(n)**, also mit variabler Länge, sind Felder vorzusehen, deren Länge stark variiert. Beispiel: Namen und Vornamen mit VARCHAR(30).
- In **Zweifelsfällen** ist pragmatisch vorzugehen. Beispiel: Die internationalen Postleitzahlen (Post-Code, Zip-Code) benötigen bis zu 10 Zeichen. Wenn eine Datenbank überwiegend nur deutsche Adressen enthält, passt VARCHAR(10) besser, bei hohem Anteil von britischen, US-amerikanischen, kanadischen und ähnlichen Adressen ist CHAR(10) zu empfehlen.

Zahlen mit exakter Größe

Werte dieser Datentypen werden mit der genauen Größe gespeichert.

- **INTEGER** bzw. **INT**

Ganze Zahl mit Vorzeichen. Der Größenbereich hängt von der Implementierung ab; auf einem 32-bit-System entspricht es meistens $\pm 2^{31}$-1, genauer von –2 147 483 648 bis +2 147 483 647.

- **SMALLINT**

Ebenfalls ein Datentyp für ganze Zahlen, aber mit kleinerem Wertebereich als INTEGER, oft von -32 768 bis +32 767.

- **BIGINT**

Ebenfalls ein Datentyp für ganze Zahlen, aber mit größerem Wertebereich als INTEGER. Auch der SQL-Standard akzeptiert, dass ein DBMS diesen Typ nicht kennt.

- **NUMERIC(p,s)** sowie **DECIMAL(p,s)**

Datentypen für Dezimalzahlen mit exakter Speicherung, also „Festkommazahlen“, wobei p die Genauigkeit und s die Anzahl der Nachkommastellen angibt. Dabei muss $0 \le s \le p$ sein, und s hat einen Defaultwert von 0. Der Parameter $s = 0$ kann entfallen; der Vorgabewert für p hängt vom DBMS ab.

Diese Dezimalzahlen sind wegen der genauen Speicherung z. B. für Daten der Buchhaltung geeignet. Bei vielen DBMS gibt es keinen Unterschied zwischen NUMERIC und DECIMAL.

Zahlen mit „näherungsweiser“ Größe

Werte dieser Datentypen werden nicht unbedingt mit der genauen Größe gespeichert, sondern in vielen Fällen nur näherungsweise.

- **FLOAT, REAL, DOUBLE PRECISION**

Diese Datentypen haben grundsätzlich die gleiche Bedeutung. Je nach DBMS gibt FLOAT(p,s) die Genauigkeit oder die Anzahl der Dezimalstellen an; auch der Wertebereich und die Genauigkeit hängen vom DBMS ab.

Diese „Gleitkommazahlen“ sind für technisch-wissenschaftliche Werte geeignet und umfassen auch die Exponentialdarstellung. Wegen der Speicherung im Binärformat sind sie aber für Geldbeträge nicht geeignet, weil sich beispielsweise der Wert 0,10 € (entspricht 10 Cent) nicht exakt abbilden lässt. Es kommt immer wieder zu Rundungsfehlern.

Zeitpunkte und Zeitintervalle

Für Datum und Uhrzeit gibt es die folgenden Datentypen:

- **DATE**

Das Datum enthält die Bestandteile **YEAR, MONTH, DAY**, wobei die Monate innerhalb eines Jahres und die Tage innerhalb eines Monats gemeint sind.

- **TIME**

Die Uhrzeit enthält die Bestandteile **HOUR, MINUTE, SECOND**, wobei die Minute innerhalb einer Stunde und die Sekunden innerhalb einer Minute gemeint sind. Sehr oft werden auch Millisekunden als Bruchteile von Sekunden registriert.

- **TIMESTAMP**

Der „Zeitstempel“ enthält Datum und Uhrzeit zusammen.

Zeitangaben können **WITH TIME ZONE** oder **WITHOUT TIME ZONE** deklariert werden. Ohne die Zeitzone ist in der Regel die lokale Zeit gemeint, mit der Zeitzone wird die Koordinierte Weltzeit (UTC) gespeichert.

Bei Datum und Uhrzeit enden die Gemeinsamkeiten der SQL-Dialekte endgültig; sie werden unterschiedlich mit „eigenen“ Datentypen realisiert. Man kann allenfalls annehmen, dass ein Tag intern mit einer ganzen Zahl und ein Zeitwert mit einem Bruchteil einer ganzen Zahl gespeichert wird.

Beispiele:

Datenbanksystem	Datentyp	Geltungsbereich	Genauigkeit
MS-SQL Server 2005	datetime	01.01.1753 bis 31.12.9999	3,33 Millisekunden
	smalldatetime	01.01.1900 bis 06.06.2079	1 Minute
MS-SQL Server 2008	date	01.01.0001 bis 31.12.9999	1 Tag
	time	00:00:00.0000000 bis 23:59:59.9999999	100 Nanosekunden
	datetime	01.01.0001 bis 31.12.9999	3,33 Millisekunden
	smalldatetime	01.01.1900 bis 06.06.2079	1 Minute
Firebird	DATE	01.01.0100 bis 29.02.32768	1 Tag
	TIME	00:00 bis 23:59.9999	6,67 Millisekunden
MySQL 5.x	DATETIME	01.01.1000 00:00:00 bis 31.12.9999 23:59:59	1 Sekunde
	DATE	01.01.1000 bis 31.12.9999	1 Tag
	TIME	–838:59:59 bis 838:59:59	1 Sekunde
	YEAR	1901 bis 2055	1 Jahr

Bitte wundern Sie sich nicht: bei jedem DBMS gibt es noch weitere Datentypen und Bezeichnungen.

Die Deklaration von TIME bei MySQL zeigt schon: Es muss sich dabei nicht um eine Uhrzeit innerhalb eines Datums handeln, sondern kann auch einen Zeitraum, d. h. ein Intervall darstellen.

- **INTERVAL**
  Ein Intervall setzt sich – je nach betrachteter Zeitdauer – zusammen aus:

- **YEAR, MONTH** für längere Zeiträume (der SQL-Standard kennt auch nur die Bezeichnung "year-month-interval")
- **DAY, HOUR, MINUTE, SECOND** für Zeiträume innerhalb eines Tages oder über mehrere Tage hinweg

Große Objekte

BLOB (Binary Large Object, binäre große Objekte) ist die allgemeine Bezeichnung für unbestimmt große Objekte.

- **BLOB**

Allgemein werden binäre Objekte z. B. für Bilder oder Bilddateien verwendet, nämlich dann, wenn der Inhalt nicht näher strukturiert ist und auch Bytes enthalten kann, die keine Zeichen sind.

- **CLOB**

Speziell werden solche Objekte, die nur „echte“ Zeichen enthalten, zum Speichern von großen Texten oder Textdateien verwendet.

Je nach DBMS werden BLOB-Varianten durch *Sub_Type* oder spezielle Datentypen für unterschiedliche Maximalgrößen oder Verwendung gekennzeichnet.

Boolean

Der Datentyp **BOOLEAN** ist für logische Werte vorgesehen. Solche Felder können die Werte TRUE (wahr) und FALSE (falsch) annehmen; auch NULL ist möglich und wird als UNKNOWN (unbekannt) interpretiert.

Wenn ein DBMS diesen Datentyp (noch) nicht kennt – wie MySQL –, dann ist mit einem der numerischen Typen eine einfache Ersatzlösung möglich (wie früher bei Interbase und Firebird); siehe unten im Abschnitt über Domains.

Konstruierte und benutzerdefinierte Datentypen

Diese Datentypen, die aus den vordefinierten Datentypen zusammengesetzt werden, werden hier nur der Vollständigkeit halber erwähnt; sie sind in der Praxis eines Anwenders ziemlich unwichtig.

- **ROW**

Eine Zeile ist eine Sammlung von Feldern; jedes Feld besteht aus dem Namen und dem Datentyp. Nun ja, eine Zeile in einer Tabelle ist (natürlich) von diesem Typ.

- **REF**

Referenztypen sind zwar im SQL-Standard vorgesehen, treten aber in der Praxis nicht auf.

- **ARRAY, MULTISET**

Felder und Mengen sind Typen von Sammlungen ("collection type"), in denen jedes Element vom gleichen Datentyp ist. Ein Array gibt die Elemente durch die Position an, ein Multiset ist eine ungeordnete Menge. Wegen der Notwendigkeit, Tabellen zu normalisieren, sind diese Typen in der Praxis unwichtig.

- **Benutzerdefinierte Typen**

Dazu gehören nicht nur ein Typ, sondern auch Methoden zu ihrer Verwendung. Auch für solche Typen sind keine sinnvollen Anwendungen zu finden.

Spezialisierte Datentypen

Datentypen können mit Einschränkungen, also CONSTRAINTs versehen werden; auch der Begriff **Domain** wird verwendet. (Einen vernünftigen deutschen Begriff gibt es dafür nicht.) Der SQL-Standard macht nicht viele Vorgaben.

Der Befehl zum Erstellen einer Domain sieht allgemein so aus:

```
CREATE DOMAIN <Domain-Name> <zugehöriger Datentyp> [<Vorgabewert>] [<Einschränkungen>]
```

Unter Interbase/Firebird wurde auf diese Weise ein Ersatz für den BOOLEAN-Datentyp erzeugt:

Firebird-Version

```
CREATE DOMAIN BOOLEAN
  -- definiere diesen Datentyp
  AS INT
  -- als Integer
  DEFAULT 0     NOT NULL
  -- Vorgabewert 0, hier ohne NULL-Werte
  CHECK (VALUE BETWEEN 0 AND 1);
  -- Werte können nur 0 (= false) und 1 (= true) sein
```

Bei MySQL können Spalten mit **ENUM** oder **SET** auf bestimmte Werte eingeschränkt werden, allerdings nur auf Zeichen.

Nationale und internationale Zeichensätze

Aus der Frühzeit der EDV ist das Problem der nationalen Zeichen geblieben: Mit 1 Byte (= 8 Bit) können höchstens 256 Zeichen (abzüglich 32 Steuerzeichen sowie Ziffern und einer Reihe von Satz- und Sonderzeichen) dargestellt werden; und das reicht nicht einmal für die Akzentbuchstaben (Umlaute) aller westeuropäischen Sprachen. Erst mit Unicode gibt es einen Standard, der weltweit alle Zeichen (z. B. auch die chinesischen Zeichen) darstellen und speichern soll.

Da ältere EDV-Systeme (Computer, Programme, Programmiersprachen, Datenbanken) weiterhin benutzt werden, muss die Verwendung nationaler Zeichensätze nach wie vor berücksichtigt werden. Dafür gibt es verschiedene Maßnahmen – jedes DBMS folgt eigenen Regeln für **CHARACTER SET** (Zeichensatz) und **COLLATE** (alphabetische Sortierung) und benutzt eigene Bezeichner für die Zeichensätze und die Regeln der Reihenfolge.

Vor allem wegen der DBMS-spezifischen Bezeichnungen kommen Sie nicht um intensive Blicke in die jeweilige Dokumentation herum.

Zeichensatz festlegen mit CHARACTER SET / CHARSET

Wenn eine Datenbank erstellt wird, muss ein Zeichensatz festgelegt werden. Ohne eine Festlegung regelt jedes DBMS selbst je nach Version, welcher Zeichensatz als Standard gelten soll. Wenn ein Programm Zugriff auf eine vorhandene Datenbank nimmt, muss ebenso der Zeichensatz angegeben werden; dieser muss mit dem ursprünglich festgelegten übereinstimmen. Wenn Umlaute falsch angezeigt werden, dann stimmen in der Regel diese Angaben nicht.

Eine neue Datenbank sollte, wenn das DBMS und die Programmiersprache dies unterstützen, möglichst mit Unicode (in der Regel als UTF8) angelegt werden.

In neueren Versionen steht die Bezeichnung NCHAR (= NATIONAL CHAR) oft nicht für einen speziellen nationalen Zeichensatz, sondern für den allgemeinen Unicode-Zeichensatz. Bei CHAR bzw. VARCHAR wird ein spezieller Zeichensatz verwendet, abhängig von der Installation oder der Datenbank.

In diesem Abschnitt kann deshalb nur beispielhaft gezeigt werden, wie Zeichensätze behandelt werden.

- Firebird, Interbase; MySQL

CHARACTER SET beschreibt den Zeichensatz einer Datenbank. Dieser gilt als Vorgabewert für alle Zeichenketten: CHAR(n), VARCHAR(n), CLOB. Für eine einzelne Spalte kann ein abweichender Zeichensatz festgelegt werden. Beispiel:

Firebird-Version

```
CREATE DATABASE 'europe.fb' DEFAULT CHARACTER SET ISO8859_1;
ALTER TABLE xyz ADD COLUMN lname VARCHAR(30) NOT NULL CHARACTER SET CYRL;
```

Es kommt auch vor, dass ein Programm mit einem anderen Zeichensatz arbeitet als die Datenbank. Dann können die Zeichensätze angepasst werden:

Firebird-Version

```
SET NAMES DOS437;
CONNECT 'europe.fb' USER 'JAMES' PASSWORD 'U4EEAH';
--  die Datenbank selbst arbeitet mit ISO8859_1, das Programm mit DOS-Codepage 437
```

- MS-SQL

Die Dokumentation geht nur allgemein auf „Nicht-Unicode-Zeichendaten“ ein. Es gibt keinerlei Erläuterung, wie ein solcher Zeichensatz festgelegt wird.

Sortierungen mit COLLATE

COLLATE legt fest, nach welchen Regeln die Reihenfolge von Zeichenketten (englisch: Collation Order) bestimmt wird. Der Vorgabewert für die Datenbank bzw. Tabelle hängt direkt vom Zeichensatz ab. Abweichende Regeln können getrennt gesteuert werden für Spalten, Vergleiche sowie Festlegungen bei ORDER BY und GROUP BY.

Firebird-Version

```
CREATE DATABASE 'europe.fb' DEFAULT CHARACTER SET ISO8859_1;
--  dies legt automatisch die Sortierung nach ISO8859_1 fest
ALTER TABLE xyz ADD COLUMN lname VARCHAR(30) NOT NULL COLLATE FR_CA;
--  dies legt die Sortierung auf kanadisches Französisch fest
SELECT ... WHERE lname COLLATE FR_FR <= :lname_search;
--  dabei soll der Vergleich nach Französisch (Frankreich) durchgeführt werden
SELECT ...
  ORDER BY LNAME COLLATE FR_CA, FNAME COLLATE FR_CA
  GROUP BY LNAME COLLATE FR_CA, FNAME COLLATE FR_CA;
--  vergleichbare Festlegungen für Reihenfolge und Gruppierung bei SELECT
```

Diese Beispiele arbeiten mit den Kürzeln für Interbase/Firebird. Andere DBMS nutzen eigene Bezeichnungen; aber die Befehle selbst sind weitgehend identisch.

Zusammenfassung

In diesem Kapitel lernten Sie die Datentypen unter SQL kennen:

- Bei Zeichenketten ist zwischen fester und variabler Länge zu unterscheiden und der Zeichensatz – UNICODE oder national – zu beachten.
- Für Zahlen ist zwischen exakter und näherungsweiser Speicherung zu unterscheiden und die Genauigkeit zu beachten.
- Für Datums- und Zeitwerte ist vor allem auf den jeweiligen Geltungsbereich und die Genauigkeit zu achten.
- Für spezielle Zwecke gibt es weitere Datentypen wie BLOB oder BOOLEAN.

Übungen

Zahlen und Datumsangaben verwenden immer die im deutschsprachigen Raum üblichen Schreibweisen. Zeichen werden mit Hochkommata begrenzt.

Übung 1	**Texte und Zahlen**	**Zur Lösung**

Geben Sie zu den Werten jeweils an, welche Datentypen passen, welche fraglich sind (also u. U. möglich, aber nicht sinnvoll oder unklar) und welche falsch sind.

1. 'A' als Char, Char(20), Varchar, Varchar(20)
2. der Ortsname 'Bietigheim-Bissingen' als Char, Char(20), Varchar, Varchar(20)
3. das Wort 'Übungen' als Varchar(20), NVarchar(20)
4. 123.456 als Integer, Smallint, Float, Numeric, Varchar(20)
5. 123,456 als Integer, Smallint, Float, Numeric, Varchar(20)
6. 789,12 [€] als Integer, Smallint, Float, Numeric, Varchar(20)

Übung 2	**Datum und Zeit**	**Zur Lösung**

Geben Sie jeweils an, welche Datentypen passen, welche fraglich sind (also u. U. möglich, aber nicht sinnvoll oder unklar) und welche falsch sind.

1. '27.11.2009' als Date, Time, Timestamp, Char(10), Varchar(20)
2. '11:42:53' als Date, Time, Timestamp, Char(10), Varchar(20)
3. '27.11.2009 11:42:53' als Date, Time, Timestamp, Char(10), Varchar(20)
4. 'November 2009' als Date, Time, Timestamp, Char(10), Varchar(20)

Übung 3	**Personen**	**Zur Lösung**

Bereiten Sie eine Tabelle *Person* vor: Notieren Sie mit möglichst konsequenter Aufteilung der Bestandteile die möglichen Spaltennamen und deren Datentypen; berücksichtigen Sie dabei auch internationale Adressen, aber keine Kontaktdaten (wie Telefon).

Übung 4	**Buchhaltung**	**Zur Lösung**

Bereiten Sie eine Tabelle *Kassenbuch* vor: Notieren Sie die möglichen Spaltennamen und deren Datentypen; berücksichtigen Sie dabei auch, dass Angaben der Buchhaltung geprüft werden müssen.

Lösungen

Lösung zu Übung 1	Texte und Zahlen	Zur Übung

Die „richtige“ Festlegung hängt vom Zusammenhang innerhalb einer Tabelle ab.

1. Char und Varchar(20) passen, Varchar ist nicht sinnvoll, Char(20) ist falsch.
2. Char(20) und Varchar(20) passen, Char und Varchar sind falsch.
3. Je nach verwendetem Zeichensatz können beide Varianten richtig, ungeeignet oder falsch sein.
4. Integer und Numeric sind richtig, Float ist möglich, Smallint ist falsch, Varchar(20) ist nicht ganz ausgeschlossen.
5. Float und Numeric sind richtig, Integer und Smallint sind falsch, Varchar(20) ist nicht ganz ausgeschlossen.
6. Numeric ist richtig, Float ist möglich, Integer und Smallint sind falsch, Varchar(20) ist nicht ganz ausgeschlossen.

Lösung zu Übung 2	Datum und Zeit	Zur Übung

1. Date und Timestamp sind richtig, Char(10) und Varchar(20) sind möglich, Time ist falsch.
2. Time und Timestamp sind richtig, Varchar(20) ist möglich, Char(10) und Date sind falsch.
3. Timestamp ist richtig, Varchar(20) ist möglich, Date, Time und Char(10) sind falsch.
4. Varchar(20) ist richtig, alles andere falsch.

Lösung zu Übung 3	Personen	Zur Übung

Bitte wundern Sie sich nicht über unerwartete Unterteilungen: Bei der folgenden Lösung werden auch Erkenntnisse der Datenbank-Theorie und der praktischen Arbeit mit Datenbanken berücksichtigt.

- ID Integer
- Titel Varchar(15)
- Vorname Varchar(30)
- Adelszusatz Varchar(15) – Trennung ist wegen der alphabetischen Sortierung sinnvoll
- Name Varchar(30)
- Adresszusatz Varchar(30)
- Strasse Varchar(24)
- Hausnr Integer (oder Smallint)
- HausnrZusatz Varchar(10) – Trennung ist wegen der numerischen Sortierung sinnvoll
- Länderkennung Char(2) – nach ISO 3166, auch Char(3) möglich, Integer oder Smallint denkbar; der Ländername ist auf jeden Fall unpraktisch
- PLZ Char(10) oder Varchar(10) – international sind bis zu 10 Zeichen möglich
- Geburtsdatum Date

Lösung zu Übung 4	Buchhaltung	Zur Übung

- ID Integer
- Buchungsjahr Integer oder Smallint
- Buchungsnummer Integer
- Buchungstermin Timestamp – je nach Arbeitsweise genügt auch Date
- Betrag Numeric oder Decimal
- Vorgang Varchar(50) – als Beschreibung der Buchung
- Bearbeiter Varchar(30) – derjenige, der den Kassenbestand ändert
- Nutzer Varchar(30) – derjenige, der die Buchung registriert
- Buchhaltung Timestamp – Termin, zu dem die Buchung registriert wird

Wenn das Kassenbuch explizit ein Teil der Buchhaltung ist, werden auch Spalten wie Buchungskonto (Haupt- und Gegenkonten) benötigt.

Siehe auch

In Wikipedia gibt es zusätzliche Hinweise:

- Gleitkommazahlen mit ausführlicher Erläuterung ihrer Ungenauigkeiten
- Koordinierte Weltzeit (UTC)
- Unicode als umfassender Zeichensatz
- ISO 3166 – Postleitzahl – Liste der Postleitsysteme

Funktionen

Im SQL-Standard werden verschiedene Funktionen festgelegt, die in jedem SQL-Dialekt vorkommen. In aller Regel ergänzt jedes DBMS diese Funktionen durch weitere eigene.

- **Skalarfunktionen** verarbeiten Werte oder Ausdrücke aus einzelnen Zahlen, Zeichenketten oder Datums- und Zeitwerten. Sofern die Werte aus einer Spalte geholt werden, handelt es sich immer um einen Wert aus einer einzelnen Zeile.
- **Spaltenfunktionen** verarbeiten alle Werte aus einer Spalte.
- Ergänzend kann man für beide Varianten auch **benutzerdefinierte Funktionen** erstellen; dies wird unter Programmierung angesprochen.

Dieses Kapitel enthält als Grundlage nur die wichtigsten Skalarfunktionen. Unter Funktionen (2) gibt es viele Ergänzungen.

Allgemeine Hinweise

Die Funktionen können überall dort verwendet werden, wo ein SQL-Ausdruck möglich ist. Wichtig ist, dass das Ergebnis der Funktion zu dem Datentyp passt, der an der betreffenden Stelle erwartet wird. Auch wenn der SQL-Standard nur einige wenige Funktionen vorschreibt, können eine ganze Reihe von Funktionen als Standard angesehen werden, weil sie immer (oder fast immer) vorhanden sind. An vielen Stellen ist aber auf Unterschiede im Namen oder in der Art des Aufrufs hinzuweisen.

Firebird kennt ab Version 2.1 viele Funktionen, die vorher als benutzerdefinierte Funktionen (user defined functions, UDF) erstellt werden mussten.

Schreibweise für Funktionen: Bei Funktionen müssen die Parameter (auch „Argumente“ genannt) immer in Klammern gesetzt werden. Diese Klammern sind auch bei einer Funktion mit konstantem Wert wie PI erforderlich. Eine Ausnahme sind lediglich die Systemfunktionen CURRENT_DATE u. ä.

Schreibweise der Beispiele: Aus Platzgründen werden Beispiele meistens nur einfach in einen Rahmen gesetzt. Für Funktionen ohne Bezug auf Tabellen und Spalten genügt in der Regel ein einfacher SELECT-Befehl; in manchen Fällen muss eine Tabelle oder eine fiktive Quelle angegeben werden:

```
SELECT 2 * 3;                          /* Normalfall */
SELECT 2 * 3    FROM rdb$database;     /* bei Firebird und Interbase */
SELECT 2 * 3    FROM dual;             /* bei Oracle */
SELECT 2 * 3    FROM SYSIBM.SYSDUMMY1; /* bei IBM DB2 */
```

In diesem Kapitel und auch im zweiten Kapitel zu Funktionen werden diese Verfahren durch eine verkürzte Schreibweise zusammengefasst:

```
SELECT 2 * 3   [from fiktiv];
```

Das ist so zu lesen: Die FROM-Klausel ist für Firebird, Interbase und Oracle notwendig und muss die jeweils benötigte Tabelle angeben; bei allen anderen DBMS muss sie entfallen.

Funktionen für Zahlen

Bei allen numerischen Funktionen müssen Sie auf den genauen Typ achten. Es ist beispielsweise ein Unterschied, ob das Ergebnis einer Division zweier ganzer Zahlen als ganze Zahl oder als Dezimalzahl behandelt werden soll.

Operatoren

Für Zahlen stehen die üblichen Operatoren zur Verfügung:

```
+  Addition
-  Subtraktion, Negation
*  Multiplikation
/  Division
```

Dafür gelten die üblichen mathematischen Regeln (Punkt vor Strich, Klammern zuerst). Bitte beachten Sie auch folgende Besonderheit:

- Bei der Division ganzer Zahlen ist auch das Ergebnis eine ganze Zahl; man nennt das „Integer-Division“:

```
SELECT 3 / 5       [from fiktiv];     /* Ergebnis: 0      */
```

- Wenn Sie das Ergebnis als Dezimalzahl haben wollen, müssen Sie (mindestens) eine der beiden Zahlen als Dezimalzahl vorgeben:

```
SELECT 3.0 / 5     [from fiktiv];     /* Ergebnis: 0.6    */
```

- Ausnahme: MySQL liefert auch bei `"3/5"` eine Dezimalzahl. Für die „Integer-Division“ gibt es die DIV-Funktion:

```
SELECT 3 DIV 5;                      /* Ergebnis: 0      */
```

Division durch 0 liefert zunächst eine Fehlermeldung "division by zero has occurred" und danach das Ergebnis NULL.

MOD – der Rest einer Division

Die Modulo-Funktion **MOD** bestimmt den Rest bei einer Division ganzer Zahlen:

```
MOD( <dividend>, <divisor> )          /* allgemein */
<dividend> % <divisor>                /* bei MS-SQL und MySQL */
```

Beispiele:

```
SELECT MOD(7, 3)   [from fiktiv];                      /* Ergebnis: 1 */
SELECT ID FROM Mitarbeiter WHERE MOD(ID, 10) = 0;      /* listet IDs mit '0' am Ende '/
```

CEILING, FLOOR, ROUND, TRUNCA TE – die nächste ganze Zahl

Es gibt mehrere Möglichkeiten, zu einer Dezimalzahl die nächste ganze Zahl zu bestimmen.

CEILING oder **CEIL** liefert die nächstgrößere ganze Zahl, genauer: die kleinste Zahl, die größer oder gleich der gegebenen Zahl ist.

```
SELECT CEILING(7.3), CEILING(-7.3)    [from fiktiv];    /* Ergebnis: 8, -7 */
```

FLOOR ist das Gegenstück dazu und liefert die nächstkleinere ganze Zahl, genauer: die größte Zahl, die kleiner oder gleich der gegebenen Zahl ist.

```
SELECT FLOOR(7.3),   FLOOR(-7.3)      [from fiktiv];    /* Ergebnis: 7, -8 */
```

TRUNCATE oder **TRUNC** schneidet den Dezimalanteil ab.

```
SELECT TRUNCATE(7.3),TRUNCATE(-7.3)   [from fiktiv];    /* Ergebnis: 7, -7 */
SELECT TRUNCATE(Schadenshoehe) FROM Schadensfall;       /* Euro-Werte ohne Cent */
```

ROUND liefert eine mathematische Rundung: ab 5 wird aufgerundet, darunter wird abgerundet.

```
ROUND( <Ausdruck> [ , <Genauigkeit> ] )
```

<Ausdruck> ist eine beliebige Zahl oder ein Ausdruck, der eine beliebige Zahl liefert.

- Wenn <Genauigkeit> nicht angegeben ist, wird 0 angenommen.
- Bei einer positiven Zahl für <Genauigkeit> wird auf entsprechend viele Dezimalstellen gerundet.
- Bei einer negativen Zahl für <Genauigkeit> wird links vom Dezimaltrenner auf entsprechend viele Nullen gerundet.

Beispiele:

```
SELECT ROUND(12.248,-2)   [from fiktiv];               /* Ergebnis:  0,000   */
SELECT ROUND(12.248,-1)   [from fiktiv];               /* Ergebnis: 10,000   */
SELECT ROUND(12.248, 0)   [from fiktiv];               /* Ergebnis: 12,000   */
SELECT ROUND(12.248, 1)   [from fiktiv];               /* Ergebnis: 12,200   */
SELECT ROUND(12.248, 2)   [from fiktiv];               /* Ergebnis: 12,250   */
SELECT ROUND(12.248, 3)   [from fiktiv];               /* Ergebnis: 12,248   */
SELECT ROUND(12.25,  1)   [from fiktiv];               /* Ergebnis: 12,300   */
SELECT ROUND(Schadenshoehe) FROM Schadensfall;         /* Euro-Werte gerundet */
```

Funktionen für Zeichenketten

Zur Bearbeitung und Prüfung von Zeichenketten (Strings) werden viele Funktionen angeboten.

Verknüpfen von Strings

Als **Operatoren**, um mehrere Zeichenketten zu verbinden, stehen zur Verfügung:

```
||       als SQL-Standard
+        für MS-SQL oder MySQL
CONCAT   für MySQL oder Oracle
```

Der senkrechte Strich wird als „Verkettungszeichen“ bezeichnet und oft auch „Pipe“-Zeichen genannt. Es wird auf der deutschen PC-Tastatur unter Windows durch die Tastenkombination **Alt Gr** + < > erzeugt.

Ein Beispiel in diesen Varianten:

```
SELECT Name || ', ' || Vorname     from Mitarbeiter;
SELECT Name +  ', ' +  Vorname     from Mitarbeiter;
SELECT CONCAT(Name, ', ', Vorname) from Mitarbeiter;
```

Alle diese Varianten liefern das gleiche Ergebnis: Für jeden Datensatz der Tabelle *Mitarbeiter* werden Name und Vorname verbunden und dazwischen ein weiterer String gesetzt, bestehend aus Komma und einem Leerzeichen.

Länge von Strings

Um die Länge einer Zeichenkette zu erfahren, gibt es die folgenden Funktionen:

```
CHARACTER_LENGTH( <string> )     SQL-Standard
CHAR_LENGTH( <string> )          SQL-Standard Kurzfassung
LEN( <string> )                  nur für MS-SQL
```

Beispiele:

```
SELECT CHAR_LENGTH('Hello World')     [from fiktiv];   /* Ergebnis: 11 */
SELECT CHAR_LENGTH('')                [from fiktiv];   /* Ergebnis:  0 */
SELECT CHAR_LENGTH( NULL )            [from fiktiv];   /* Ergebnis: <null> */
SELECT Name FROM Mitarbeiter ORDER BY CHAR_LENGTH(Name) DESC;
    /* liefert die Namen der Mitarbeiter, absteigend sortiert nach Länge */
```

UPPER, LOWER – Groß- und Kleinbuchstaben

UPPER konvertiert den gegebenen String zu Großbuchstaben; **LOWER** gibt einen String zurück, der nur aus Kleinbuchstaben besteht.

```
SELECT UPPER('Abc Äöü Xyzß ÀÉÇ àéç')  [from fiktiv];   /* Ergebnis: 'ABC ÄÖÜ XYZß ÀÉÇ ÀÉÇ' */
SELECT LOWER('Abc Äöü Xyzß ÀÉÇ àéç')  [from fiktiv];   /* Ergebnis: 'abc äöü xyzß àéç àéç' */
SELECT UPPER(Kuerzel), Bezeichnung FROM Abteilung;     /* Kurzbezeichnungen in Großbuchstaben */
```

Ob die Konvertierung bei Umlauten richtig funktioniert, hängt vom verwendeten Zeichensatz ab.

SUBSTRING – Teile von Zeichenketten

SUBSTRING ist der SQL-Standard, um aus einem String einen Teil herauszuholen:

```
SUBSTRING( <text> FROM <start> FOR <anzahl> )     /* SQL-Standard          */
SUBSTRING( <text> ,    <start> ,   <anzahl> )     /* MS-SQL, MySQL, Oracle */
```

Diese Funktion heißt unter Oracle **SUBSTR** und kann auch bei MySQL so bezeichnet werden.

Der Ausgangstext wird von Position 1 an gezählt. Der Teilstring beginnt an der hinter FROM genannten Position und übernimmt so viele Zeichen wie hinter FOR angegeben ist:

```
SELECT SUBSTRING('Abc Def Ghi' FROM 6 FOR 4)  [from fiktiv];   /* Ergebnis: 'ef G'  */
SELECT CONCAT(Name, ', ', SUBSTRING(Vorname FROM 1 FOR 1), '.') from Mitarbeiter;
    /* liefert den Namen und vom Vornamen den Anfangsbuchstaben */
```

Wenn der <anzahl>-Parameter fehlt, wird alles bis zum Ende von <text> übernommen:

```
SELECT SUBSTRING('Abc Def Ghi' FROM 6)       [from fiktiv];   /* Ergebnis: 'ef Ghi'  */
```

Wenn der <anzahl>-Parameter 0 lautet, werden 0 Zeichen übernommen, man erhält also einen leeren String.

MySQL bietet noch eine Reihe weiterer Varianten.

Hinweis: Nach SQL-Standard liefert das Ergebnis von SUBSTRING seltsamerweise einen Text von gleicher Länge wie der ursprüngliche Text; die jetzt zwangsläufig folgenden Leerzeichen müssen ggf. mit der TRIM-Funktion (im zweiten Kapitel über Funktionen) entfernt werden.

Funktionen für Datums- und Zeitwerte

Bei den Datums- und Zeitfunktionen gilt das gleiche wie für Datum und Zeit als Datentyp: Jeder SQL-Dialekt hat sich seinen eigenen „Standard“ ausgedacht. Wir beschränken uns deshalb auf die wichtigsten Funktionen, die es so ähnlich „immer“ gibt, und verweisen auf die DBMS-Dokumentationen.

Vor allem MySQL bietet viele zusätzliche Funktionen an. Teilweise sind es nur verkürzte und spezialisierte Schreibweisen der Standardfunktionen, teilweise liefern sie zusätzliche Möglichkeiten.

Systemdatum und -uhrzeit

Nach SQL-Standard werden aktuelle Uhrzeit und Datum abgefragt:

```
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
```

In Klammern kann als <precision> die Anzahl der Dezimalstellen bei den Sekunden angegeben werden.

Beispiele:

```
SELECT CURRENT_TIMESTAMP    [from fiktiv];   /* Ergebnis: '19.09.2009 13:47:49' */
UPDATE Versicherungsvertrag SET Aenderungsdatum = CURRENT_DATE WHERE irgendetwas;
```

Bei MS-SQL gibt es nur CURRENT_TIMESTAMP als Standardfunktion, dafür aber andere Funktionen mit höherer Genauigkeit.

Teile von Datum oder Uhrzeit b estimmen

Für diesen Zweck gibt es vor allem **EXTRACT** als Standardfunktion:

```
EXTRACT ( <part> FROM <value> )
```

<value> ist der Wert des betreffenden Datums und/oder der Uhrzeit, die aufgeteilt werden soll. Als <part> wird angegeben, welcher Teil des Datums gewünscht wird:

```
YEAR | MONTH | DAY | HOUR | MINUTE | SECOND | MILLISECOND
```

Bei einem DATE-Feld ohne Uhrzeit sind HOUR usw. natürlich unzulässig, bei einem TIME-Feld nur mit Uhrzeit die Bestandteile YEAR usw. Beispiele:

```
SELECT ID, Datum, EXTRACT(YEAR FROM Datum) AS Jahr, EXTRACT(MONTH FROM Datum) AS Monat
  FROM Schadensfall     ORDER BY Jahr, Monat;
SELECT 'Stunde = ' || EXTRACT(HOUR FROM CURRENT_TIME) [from fiktiv];   /* Ergebnis: 'Stunde = 14' */
```

Bei MS-SQL heißt diese Standardfunktion DATEPART; als <part> können viele weitere Varianten genutzt werden.

Sehr oft gibt es weitere Funktionen, die direkt einen Bestandteil abfragen, zum Beispiel:

```
YEAR( <value> )         liefert das Jahr
MONTH( <value> )        liefert den Monat usw.
MINUTE( <value> )       liefert die Minute usw.
DAYOFWEEK( <value> )    gibt den Wochentag (als Zahl, 1 für Sonntag usw.) an
```

Wie gesagt: Lesen Sie in der DBMS-Dokumentation nach, was es sonst noch gibt.

Funktionen für logische und NULL-Werte

Wenn man es genau nimmt, gehören dazu auch **Prüfungen** wie „Ist ein Wert NULL?“. Der SQL-Standard hat dazu und zu anderen Prüfungen verschiedene spezielle Ausdrücke vorgesehen. Diese gehören vor allem zur WHERE-Klausel des SELECT-Befehls:

```
= < > usw.      Größenvergleich zweier Werte
BETWEEN AND     Werte zwischen zwei Grenzen
LIKE            Ähnlichkeiten (1)
CONTAINS u.a.   Ähnlichkeiten (2)
IS NULL         null-Werte prüfen
IN              genauer Vergleich mit einer Liste
EXISTS          schneller Vergleich mit einer Liste
```

Alle diese Ausdrücke liefern einen der logischen Werte TRUE, FALSE – also WAHR oder FALSCH – und ggf. NULL – also <unbekannt> – zurück und können als boolescher Wert weiterverarbeitet oder ausgewertet werden.

Operatoren

Zur **Verknüpfung logischer Werte** gibt es die „booleschen Operatoren“ NOT, AND, OR (nur bei MySQL auch XOR):

```
NOT als Negation
AND als Konjunktion
OR  als Adjunktion
XOR als Kontravalenz
```

Auch diese Operatoren werden bei der WHERE-Klausel behandelt.

Zur **Verknüpfung von NULL-Werten** gibt es vielfältige Regeln, je nach dem Zusammenhang, in dem das von Bedeutung ist. Man kann sich aber folgende Regel merken:

> Wenn ein Ausgangswert NULL ist, also <unbekannt>, und dieser „Wert“ mit etwas verknüpft wird (z. B. mit einer Zahl addiert wird), kann das Ergebnis nur <unbekannt> sein, also NULL lauten.

Konvertierungen

In der EDV – also auch bei SQL-Datenbanken – ist der verwendete Datentyp immer von Bedeutung. Mit Zahlen kann gerechnet werden, mit Zeichenketten nicht. Größenvergleiche von Zahlen gelten immer und überall; bei Zeichenketten hängt die Reihenfolge auch von der Sprache ab. Ein Datum wird in Deutschland

durch '11.09.2001' beschrieben, in England durch '11/09/2001' und in den USA durch '09/11/2001'. Wenn *wir* etwas aufschreiben (z. B. einen SQL-Befehl), dann benutzen wir zwangsläufig immer Zeichen bzw. Zeichenketten, auch wenn wir Zahlen oder ein Datum meinen.

In vielen Fällen „versteht“ das DBMS, was wir mit einer solchen Schreibweise meinen; dann werden durch „implizite Konvertierung“ die Datentypen automatisch ineinander übertragen. In anderen Fällen muss der Anwender dem DBMS durch eine Konvertierungsfunktion erläutern, was wie zu verstehen ist.

Implizite Konvertierung

Datentypen, die problemlos vergleichbar sind, werden „automatisch“ ineinander übergeführt.

- Die Zahl 17 kann je nach Situation ein INTEGER, ein SMALLINT, ein BIGINT, aber auch NUMERIC oder FLOAT sein.
- Die Zahl 17 kann in einem SELECT-Befehl auch als String '17' erkannt und verarbeitet werden.

```
SELECT ID, Name, Vorname     FROM Mitarbeiter    WHERE ID = '17';
```

- Je nach DBMS kann ein String wie '11.09.2001' als Datum erkannt und verarbeitet werden. Vielleicht verlangt es aber auch eine andere Schreibweise wie '11/09/2001'. Die Schreibweise '2009-09-11' nach ISO 8601 sollte dagegen immer richtig verstanden werden (aber auch da gibt es Abweichungen).

Es gibt bereits durch den SQL-Standard ausführliche Regeln, welche Typen immer, unter bestimmten Umständen oder niemals ineinander übergeführt werden können. Jedes DBMS ergänzt diese allgemeinen Regeln durch eigene.

Wenn ein solcher Befehl ausgeführt wird, dürfte es niemals Missverständnisse geben, sondern er wird „mit an Sicherheit grenzender Wahrscheinlichkeit“ korrekt erledigt. Wenn ein Wert nicht eindeutig ist, wird das DBMS eher zu früh als zu spät mit einer Fehlermeldung wie "conversion error from string ..." reagieren. Dann ist es Zeit für eine explizite Konvertierung, meistens durch CAST.

CAST

Die **CAST**-Funktion ist der SQL-Standard für die Überführung eines Wertes von einem Datentyp in einen anderen.

```
CAST ( <expression> AS <type> )
```

Als <expression> ist etwas angegeben, was den „falschen“ Typ hat, nämlich ein Wert oder Ausdruck. Mit <type> wird der Datentyp angegeben, der an der betreffenden Stelle gewünscht oder benötigt wird. Das Ergebnis des CASTings (der Konvertierung) ist dann genau von diesem Typ.

Beispiele für Datum:

```
SELECT EXTRACT(DAY FROM '07.14.2009')               [from fiktiv];
  /* expression evaluation not supported */
SELECT EXTRACT(DAY FROM CAST('07.14.2009' as date))  [from fiktiv];
  /* conversion error from string "07.14.2009" */
SELECT EXTRACT(DAY FROM CAST('14.07.2009' as date))  [from fiktiv];
  /* Ergebnis: '14' */
SELECT Name, Vorname, CAST(Geburtsdatum AS CHAR(10)) FROM Versicherungsnehmer;
  /* Ergebnis wird in der Form '1953-01-13' angezeigt */
```

Kürzere Zeichenketten können schnell verlängert werden, wenn es nötig ist:

```
SELECT ID, CAST(Kuerzel AS CHAR(20))     FROM Abteilung;
   -- 20 Zeichen Länge statt eigentlich 10 Zeichen
```

Das Verkürzen funktioniert nicht immer so einfach. Ob bei Überschreitung einer Maximallänge einfach abgeschnitten wird oder ob es zu einer Fehlermeldung "string truncation" kommt, hängt vom DBMS ab; dann müssen Sie ggf. eine SUBSTRING-Variante benutzen.

```
-- vielleicht funktioniert es so:
SELECT CAST(Name AS CHAR(15)) || Vorname  from Versicherungsnehmer;
-- aber sicherer klappt es so:
SELECT SUBSTRING(Name FROM 1 FOR 15) || Vorname  from Versicherungsnehmer;
-- unter Berücksichtigung des Hinweises bei SUBSTRING:
SELECT TRIM( SUBSTRING(Name FROM 1 FOR 15) ) || Vorname  from Versicherungsnehmer;
```

Bitte lesen Sie in Ihrer SQL-Dokumentation nach, zwischen welchen Datentypen implizite Konvertierung möglich ist und wie die explizite Konvertierung mit CAST ausgeführt wird.

CONVERT

Nach dem SQL-Standard ist **CONVERT** vorgesehen zum Konvertieren von Zeichenketten in verschiedenen Zeichensätzen:

```
CONVERT ( <text> USING <transcoding name> )       SQL-Standard
CONVERT ( <text>, <transcoding name> )            alternative Schreibweise
```

Firebird kennt diese Funktion überhaupt nicht. MS-SQL benutzt eine andere Syntax und bietet vor allem für Datums- und Zeitformate viele weitere Möglichkeiten:

```
CONVERT ( <type>, <text> [ , <style> ] )
```

Wegen dieser starken Abweichungen verzichten wir auf weitere Erläuterungen und verweisen auf die jeweilige Dokumentation.

Datum und Zeit

Vor allem für die Konvertierung mit Datums- und Zeitangaben bieten die verschiedenen DBMS Erweiterungen. Beispiele:

- MS-SQL hat die Syntax von **CONVERT** für diesen Zweck erweitert.
- MySQL ermöglicht die Konvertierung mit **STR_TO_DATE** und **DATE_FORMAT**.
- Oracle kennt eine Reihe von Funktionen wie **TO_DATE** usw..
- Wo es so etwas nicht gibt, kann man Day/Month/Year extrahieren, per CAST konvertieren und dann mit || neu zusammensetzen.

Noch ein Grund für das Studium der Dokumentation...

Spaltenfunktionen

Die **Spaltenfunktionen** werden auch als **Aggregatfunktionen** bezeichnet, weil sie eine Menge von Werten – nämlich aus allen Zeilen einer bestimmten Spalte – zusammenfassen und einen gemeinsamen Wert bestimmen. In der Regel wird dazu eine Spalte aus einer der beteiligten Tabellen verwendet; es kann aber auch ein sonstiger gültiger SQL-Ausdruck sein, der als Ergebnis einen einzelnen Wert liefert. Das Ergebnis der Funktion ist dann ein Wert, der aus allen passenden Zeilen der Abfrage berechnet wird.

Bei Abfragen kann das Ergebnis einer Spaltenfunktion auch nach den Werten einer oder mehrerer Spalten oder Berechnungen gruppiert werden. Die Aggregatfunktionen liefern dann für jede Gruppe ein Teilergebnis. Näheres siehe unter Gruppierungen.

COUNT – Anzahl

Die Funktion **COUNT** zählt alle Zeilen, die einen eindeutigen Wert enthalten, also nicht NULL sind. Sie kann auf alle Datentypen angewendet werden, da für jeden Datentyp NULL definiert ist. Beispiel:

```
SELECT COUNT(Farbe) AS Anzahl_Farbe FROM Fahrzeug;
```

Die Spalte *Farbe* ist als VARCHAR(30), also als Text variabler Länge, definiert und optional. Hier werden also alle Zeilen gezählt, die in dieser Spalte einen Eintrag haben. Dasselbe funktioniert auch mit einer Spalte, die numerisch ist:

```
SELECT COUNT(Schadenshoehe ) AS Anzahl_Schadenshoehe FROM Schadensfall;
```

Hier ist die Spalte numerisch und optional. Die Zahl *0* ist bekanntlich nicht NULL. Wenn in der Spalte eine *0* steht, wird sie mitgezählt.

Ein Spezialfall ist der Asterisk '*****' als Parameter. Dies bezieht sich dann nicht auf eine einzelne Spalte, sondern auf eine ganze Zeile. So wird also die Anzahl der Zeilen in der Tabelle gezählt:

```
SELECT COUNT(*) AS Anzahl_Zeilen FROM Schadensfall;
```

Die Funktion COUNT liefert niemals NULL zurück, sondern immer eine Zahl; wenn alle Werte in einer Spalte NULL sind, ist das Ergebnis die Zahl 0 (es gibt 0 Zeilen mit einem Wert ungleich NULL in dieser Spalte).

SUM – Summe

Die Funktion **SUM** kann (natürlich) nur auf numerische Datentypen angewendet werden. Im Gegensatz zu COUNT liefert SUM nur dann einen Wert zurück, wenn wenigstens ein Eingabewert nicht NULL ist. Als Parameter kann nicht nur eine einzelne numerische Spalte, sondern auch eine Berechnung übergeben werden, die als Ergebnis eine einzelne Zahl liefert. Ein Beispiel für eine einzelne numerische Spalte ist:

```
SELECT SUM(Schadenshoehe ) AS Summe_Schadenshoehe FROM Schadensfall;
```

Als Ergebnis werden alle Werte in der Spalte *Schadenshoehe* aufsummiert. Als Parameter kann aber auch eine Berechnung übergeben werden.

Hier werden Euro-Beträge aus *Schadenshoehe* zuerst in US-Dollar nach einem Tageskurs umgerechnet und danach aufsummiert.

```
SELECT SUM(Schadenshoehe * 1.5068) AS Summe_Schadenshoehe_Dollar FROM Schadensfall;
```

Eine Besonderheit ist das Berechnen von Vergleichen. Ein Vergleich wird als WAHR oder FALSCH ausgewertet. Sofern das DBMS (wie bei MySQL oder Access) das Ergebnis als Zahl benutzt, ist das Ergebnis eines Vergleichs daher 1 oder 0 (bei Access –1 oder 0). Um alle Fälle zu zählen, deren Schadenshöhe größer als 1000 ist, müsste der Befehl so aussehen:

```
SELECT SUM(Schadenshoehe > 1000) AS Anzahl_Schadenshoehe_gt_1000 FROM Schadensfall;
```

Dabei werden nicht etwa die Schäden aufsummiert, sondern nur das Ergebnis des Vergleichs, also 0 oder 1, im Grunde also gezählt. Die Funktion COUNT kann hier nicht genommen werden, da sie sowohl die 1 als auch die 0 zählen würde.

Einige DBMS (z. B. DB2, Oracle) haben eine strengere Typenkontrolle; Firebird nimmt eine Zwischenstellung ein. Dabei haben Vergleichsausdrücke grundsätzlich ein boolesches Ergebnis, das nicht summiert werden kann. Dann kann man sich mit der CASE-Funktion behelfen, die dem Wahrweitswert TRUE eine 1 zuordnet und dem Wert FALSE eine 0:

```
SELECT SUM(CASE WHEN Schadenshoehe > 1000 THEN 1
                ELSE                           0
           END) AS Anzahl_Schadenshoehe_gt_1000
  FROM Schadensfall;
```

MAX, MIN – Maximum, Minimum

Diese Funktionen können auf jeden Datentyp angewendet werden, für den ein Vergleich ein gültiges Ergebnis liefert. Dies gilt für numerische Werte, Datumswerte und Textwerte, nicht aber für z. B. BLOBs (binary large objects). Bei Textwerten ist zu bedenken, dass die Sortierreihenfolge je nach verwendetem Betriebssystem, DBMS und Zeichensatzeinstellungen der Tabelle oder Spalte unterschiedlich ist, die Funktion demnach auch unterschiedliche Ergebnisse liefern kann.

Suche den kleinsten, von NULL verschiedenen Schadensfall.

```
SELECT MIN(Schadenshoehe ) AS Minimum_Schadenshoehe FROM Schadensfall;
```

Kommen nur NULL-Werte vor, wird NULL zurückgegeben. Gibt es mehrere Zeilen, die den kleinsten Wert haben, wird trotzdem nur ein Wert zurückgegeben. Welche Zeile diesen Wert liefert, ist nicht definiert.

Für **MAX** gilt Entsprechendes wie für **MIN**.

AVG – Mittelwert

AVG (average = Durchschnitt) kann nur auf numerische Werte angewendet werden. Das für SUM Gesagte gilt analog auch für AVG. Um die mittlere Schadenshöhe zu berechnen, schreibt man:

```
SELECT AVG(Schadenshoehe) AS Mittlere_Schadenshoehe FROM Schadensfall;
```

NULL-Werte fließen dabei nicht in die Berechnung ein, Nullen aber sehr wohl.

STDDEV – Standardabweichung

Die Standardabweichung **STDDEV** oder **STDEV** kann auch nur für numerische Werte berechnet werden. NULL-Werte fließen nicht mit in die Berechnung ein, Nullen schon. Wie bei SUM können auch Berechnungen als Werte genommen werden. Die Standardabweichung der Schadensfälle wird so berechnet:

```
SELECT STDDEV(Schadenshoehe) AS StdAbw_Schadenshoehe FROM Schadensfall;
```

Zusammenfassung

In diesem Kapitel lernten wir die wichtigsten „eingebauten“ Funktionen kennen:

- Für Zahlen gibt es vor allem die Operatoren, dazu die modulo-Funktionen und Möglichkeiten für Rundungen.
- Für Zeichenketten gibt es vor allem das Verknüpfen und Aufteilen, dazu die Längenbestimmung und die Umwandlung in Groß- und Kleinbuchstaben.
- Für Datums- und Zeitwerte gibt es neben Systemfunktionen die Verwendung einzelner Teile.
- Für logische und NULL-Werte gibt es vor allem Vergleiche und Kombinationen durch Operatoren.
- Konvertierungen – implizite, CAST, CONVERT – dienen dazu, dass ein Wert des einen Datentyps anstelle eines anderen Typs verwendet werden kann.

Mit Spaltenfunktionen werden alle Werte einer Spalte gemeinsam ausgewertet.

Übungen

Übung 1	Definitionen	Zur Lösung

Welche der folgenden Feststellungen sind richtig, welche sind falsch?

1. Eine Skalarfunktion bestimmt aus allen Werten eines Feldes einen gemeinsamen Wert.
2. Eine Spaltenfunktion bestimmt aus allen Werten eines Feldes einen gemeinsamen Wert.
3. Eine Skalarfunktion kann keine Werte aus Spalten einer Tabelle verarbeiten.
4. Eine benutzerdefinierte Funktion kann als Skalarfunktion, aber nicht als Spaltenfunktion dienen.
5. Wenn einer Funktion keine Argumente übergeben werden, kann auf die Klammern hinter dem Funktionsnamen verzichtet werden.
6. Wenn eine Funktion für einen SQL-Ausdruck benutzt wird, muss das Ergebnis der Funktion vom Datentyp her mit dem übereinstimmen, der an der betreffenden Stelle erwartet wird.

Übung 2	Definitionen	Zur Lösung

Welche der folgenden Funktionen sind Spaltenfunktionen? Welche sind Skalarfunktionen, welche davon sind Konvertierungen?

1. Bestimme den Rest bei einer Division ganzer Zahlen.
2. Bestimme die Länge des Namens des Mitarbeiters mit der ID 13.
3. Bestimme die maximale Länge aller Mitarbeiter-Namen.
4. Bestimme die Gesamtlänge aller Mitarbeiter-Namen.
5. Verwandle eine Zeichenkette, die nur Ziffern enthält, in eine ganze Zahl.
6. Verwandle eine Zeichenkette, die keine Ziffern enthält, in einen String, der nur Großbuchstaben enthält.
7. Bestimme das aktuelle Datum.

Übung 3	**Funktionen mit Zahlen**	**Zur Lösung**

Benutzen Sie für die folgenden Berechnungen die Spalte *Schadenshoehe* der Tabelle *Schadensfall*. Welche Datensätze benutzt werden, also der Inhalt der WHERE-Klausel, soll uns dabei nicht interessieren. Auch geht es nur um die Formeln, nicht um einen SELECT-Befehl.

1. Berechnen Sie (ohne AVG-Funktion) die durchschnittliche Schadenshöhe.
2. Bestimmen Sie den prozentualen Anteil eines bestimmten Schadensfalls an der gesamten Schadenshöhe.

Übung 4	**Funktionen mit Zeichenketten**	**Zur Lösung**

Schreiben Sie Name, Vorname und Abteilung der Mitarbeiter in tabellarischer Form (nehmen wir an, dass das *Kuerzel* der Abteilung in der Tabelle *Mitarbeiter* stünde); benutzen Sie dazu nacheinander die folgenden Teilaufgaben:

1. Bringen Sie die Namen auf eine einheitliche Länge von 20 Zeichen.
2. Bringen Sie die Namen auf eine einheitliche Länge von 10 Zeichen.
3. Bringen Sie die Vornamen ebenso auf eine Länge von 10 Zeichen.
4. Setzen Sie diese Teilergebnisse zusammen und fügen Sie dazwischen je zwei Leerzeichen ein.

Übung 5	**Funktionen mit Datum und Zeit**	**Zur Lösung**

Gegeben ist ein Timestamp-Wert mit dem Spaltennamen *Zeitstempel* und dem Inhalt "16. Dezember 2009 um 19:53 Uhr". Zeigen Sie diesen Wert als Zeichenkette im Format "12/2009; 16. Tag; 7 Minuten vor 20 Uhr" an. (Für die String-Verknüpfung benutzen wir jetzt das Plus-Zeichen. Die Leerzeichen zwischen den Bestandteilen können Sie ignorieren. Auch muss es keine allgemeingültige Lösung sein, die alle Eventualitäten beachtet.)

Übung 6	**Funktionen mit Datum und Zeit**	**Zur Lösung**

Gegeben ist eine Zeichenkette *datum* mit dem Inhalt "16122009". Sorgen Sie dafür, dass das Datum für jedes DBMS gültig ist. Zusatzfrage: Muss dafür CAST verwendet werden und warum bzw. warum nicht?

Lösungen

Lösung zu Übung 1	**Definitionen**	**Zur Übung**

Richtig sind 2 und 6, falsch sind 1, 3, 4, 5.

Lösung zu Übung 2	**Definitionen**	**Zur Übung**

Spaltenfunktionen sind 3 und 4; beide benutzen das Ergebnis von Skalarfunktionen. Alle anderen sind Skalarfunktionen, wobei 5 eine Konvertierung ist und 7 eine Systemfunktion, die ohne Klammern geschrieben wird.

Lösung zu Übung 3	Funktionen mit Zahlen	Zur Übung

1. SUM(Schadenshoehe) / COUNT(Schadenshoehe)
2. ROUND(Schadenshoehe * 100 / SUM(Schadenshoehe))

Lösung zu Übung 4	Funktionen mit Zeichenketten	Zur Übung

```
1. CAST(Name AS CHAR(20))
2. SUBSTRING( CAST(Name AS CHAR(20)) FROM 1 FOR 10 )
3. SUBSTRING( CAST(Vorname AS CHAR(20)) FROM 1 FOR 10 )
4. SUBSTRING( CAST(Name AS CHAR(20)) FROM 1 FOR 10 )    || '  ' ||
   SUBSTRING( CAST(Vorname AS CHAR(20)) FROM 1 FOR 10 ) || '  ' || Kuerzel
```

Lösung zu Übung 5	Funktionen mit Datum und Zeit	Zur Übung

```
EXTRACT(MONTH FROM CURRENT_TIMESTAMP) + '/' + EXTRACT(YEAR FROM CURRENT_TIMESTAMP) + ';'
+ EXTRACT(DAY FROM CURRENT_TIMESTAMP) + '.Tag; '
+ CAST((60 - EXTRACT(MINUTE FROM CURRENT_TIMESTAMP)) AS Varchar(2)) + ' Minuten vor '
+ CAST( (EXTRACT(HOUR FROM CURRENT_TIMESTAMP) + 1) AS Varchar(2)) + ' Uhr'
```

Lösung zu Übung 6	Funktionen mit Datum und Zeit	Zur Übung

```
  SUBSTRING(datum FROM 5 FOR 4) + '-'
+ SUBSTRING(datum FROM 3 FOR 2) + '-'
+ SUBSTRING(datum FROM 1 FOR 2)
```

Auf CAST kann (fast immer) verzichtet werden, weil mit dieser Substring-Verwendung die Standardschreibweise '2009-12-16' nach ISO 8601 erreicht wird.

Siehe auch

Einige Hinweise sind in den folgenden Kapiteln zu finden:

- SQL-Befehle beschreibt auch den Begriff „SQL-Ausdruck“.
- Datentypen

Weitere Erläuterungen stehen bei Wikipedia:

- Senkrechter Strich oder „Pipe“-Zeichen
- Boolesche Operatoren
- ISO 8601 zur Schreibweise von Datumsangaben

Lizenz

Eine deutschsprachige Beschreibung für Autoren und Weiternutzer findet man in den Nutzungsbedingungen der Wikimedia Foundation.
