# 8 DML (1) – Daten abfragen

Eine Datenbank enthält eine Vielzahl verschiedener Daten. Abfragen dienen dazu, bestimmte Daten aus der Datenbank auszugeben. Dabei kann die Ergebnismenge gemäß 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 „Mehr zu Abfragen“ behandelt, beginnend mit *Ausführliche SELECT-Struktur*[1]; unten im Abschnitt „Ausblick auf komplexe Abfragen“ gibt es Hinweise auf diese weiteren Möglichkeiten.

## 8.1 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 Be-

1 Kapitel **??** auf Seite **??**

fehls 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

Dies sowie weitere Einzelheiten zu den wichtigsten Bestandteilen stehen in *Ausführliche SELECT-Struktur*[2] und anderen „fortgeschrittenen“ Kapiteln.

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.

## 8.2 Die einfachsten Abfragen

➤ **Aufgabe:** 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“.

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.

➤ **Aufgabe:** So erhält man die Namensliste aller Fahrzeughersteller:

2 Kapitel **??** auf Seite **??**

```
SELECT Name FROM Fahrzeughersteller;
```

➤ **Aufgabe:** 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* und dem Alias *Staat* für *Land* versehen werden:

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

## 8.3 DISTINCT – Keine doppelten Zeilen

Wenn Sie bei einem SELECT-Befehl den **DISTINCT**-Parameter angeben, erhalten Sie nur eindeutige Ergebnisse:

➤ **Aufgabe:** Gesucht werden die Fahrzeuge, für die Schadensfälle aufgetreten sind.

```
SELECT Fahrzeug_ID
  FROM Zuordnung_SF_FZ;
```

Tatsächlich wird für jeden Schadensfall eine Zeile ausgegeben, die Fahrzeuge 2, 5 und 7 erscheinen mehrmals. Damit keine doppelten Zeilen ausgegeben werden, wird DISTINCT vor den Spaltennamen in den SQL-Befehl eingefügt:

```
SELECT DISTINCT Fahrzeug_ID
  FROM Zuordnung_SF_FZ;
```

```
 Fahrzeug_ID
       3
       4
       5
       6
       7
```

Damit erscheint jede Fahrzeug-ID nur einmal in der Liste.

Bitte beachten Sie: Als „eindeutig“ gilt immer die gesamte Zeile, also alle aufgeführten Spalten gemeinsam. Die folgende Abfrage liefert alle Datensätze; Fahrzeuge mit mehreren Schadensfällen stehen auch mehrfach in der Liste.

Quelltext **Falsch**

```
SELECT DISTINCT Fahrzeug_ID, ID
  FROM Zuordnung_SF_FZ;
```

*Nur theoretisch DISTINCT, praktisch nicht*

Die Alternative zu DISTINCT ist das in der Syntax genannte **ALL**, das ausdrücklich alle Datensätze abfragt. Da dies der Standardwert ist, wird er äußerst selten benutzt, sondern kann weggelassen werden.

```
SELECT ALL Fahrzeug_ID
  FROM Zuordnung_SF_FZ
```

## 8.4 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.

➤ **Aufgabe:** 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 Bedingungen (mit einem booleschen Operator verknüpft) einfügen. Jede einzelne besteht aus dem Namen der Spalte, deren Inhalt überprüft werden soll, und einem Wert, wobei beide mit einem Vergleichsoperator verknüpft sind.

Für die Umkehrung einer solchen Bedingung gibt es mehrere Möglichkeiten:

- Man kann alle anderen Fälle einzeln auflisten.
- Man dreht einfach den Vergleichsoperator um.
- Man kehrt die Bedingung mit **NOT** um.

➤ **Aufgabe:** Es sollen alle Fahrzeughersteller angezeigt werden, die außerhalb Deutschlands sitzen.

```
- Vergleichsoperator ändern
SELECT * FROM Fahrzeughersteller
 WHERE Land <> 'Deutschland';
- Bedingung mit NOT umkehren
SELECT * FROM Fahrzeughersteller
 WHERE NOT (Land = 'Deutschland');
```

Vertiefte Erläuterungen sind unter *WHERE-Klausel im Detail*[3] zu finden.

## 8.5 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.

➤ **Aufgabe:** Die Liste der Hersteller wird nach dem Namen sortiert:

```
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;
```

3 Kapitel **??** auf Seite **??**

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*[4] – 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, sondern nach mehreren Spalten mit einer eigenen Regel für jede Spalte. Dabei werden die Zeilen zuerst nach der ersten Spalte sortiert und innerhalb dieser Spalte nach der zweiten Spalte usw. 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.

➤ **Aufgabe:** 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;
```

## 8.6 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.

### 8.6.1 Traditionell mit FROM und WHERE

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

➤ **Aufgabe:** 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
```

4 Kapitel **??** auf Seite **??**

```
    AND ab.Ort in ('Dortmund', 'Bochum')
ORDER BY mi.Name, mi.Vorname;
```

Es werden also Daten aus den Tabellen *Mitarbeiter* (Name und Raum) sowie *Abteilung* (Ort) gesucht. Für die Verknüpfung werden diese 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 oft weggelassen werden; aber schon zur Klarheit 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*[5] zu finden.

### 8.6.2 Modern mit JOIN . . . ON

Beim „modernen“ Weg steht eine Tabelle in der FROM-Klausel, nämlich diejenige, die als wichtigste oder „Haupttabelle“ der Abfrage anzusehen ist. 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.

5 Kapitel **??** auf Seite **??**

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*[6] zu finden.

## 8.7 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:

➤ **Aufgabe:** 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 Schaden,
       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;
```

| NAME | VORNAME | STRASSE | HNR | PLZ | ORT | SCHADEN | 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 in den Kapiteln *Funktionen*[7] sowie *Gruppierungen*[8].

6 Kapitel **??** auf Seite **??**
7 Kapitel **??** auf Seite **??**
8 Kapitel **??** auf Seite **??**

## 8.8 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.

## 8.9 Übungen

Die Lösungen beginnen auf Seite 67.

Bei den Übungen 3–6 ist jeweils eine Abfrage zur Tabelle *Abteilung* zu erstellen.

### Übung 1 – Richtig oder falsch?

Welche der folgenden Aussagen sind richtig, welche sind falsch?

1. Fehlerhafte Verwendung von SELECT kann keinen Schaden am Datenbestand hervorrufen.
2. Die gewünschten Spalten werden in einer Liste jeweils durch Leerzeichen getrennt.
3. Wenn alle Spalten gewünscht werden, sind auch alle Spalten in dieser Liste aufzuführen.
4. Es ist Standard, den Parameter ALL bei einem SELECT-Befehl anzugeben.
5. Bei SELECT DISTINCT werden nur solche Zeilen angezeigt, die sich in mindestens einer Spalte unterscheiden.
6. Die WHERE-Klausel dient dazu, das Ergebnis der Abfrage wunschgemäß zu formatieren.

7. Eine WHERE-Bedingung ist stets eine Prüfung auf WAHR/FALSCH: Entweder ein Datensatz erfüllt die Bedingung und gehört zum Abfrageergebnis, oder er gehört nicht dazu.
8. WHERE-Bedingungen können mit AND/OR kombiniert und mit NOT umgekehrt werden.
9. Solche Kombinationen von Bedingungen müssen durch Klammern gegliedert werden.
10. Um Daten aus mehreren Tabellen zu verknüpfen, ist es möglich, diese Tabellen einfach in der FROM-Klausel aufzuführen.

### Übung 2 – Pflichtangaben

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

### Übung 3 – Alle Angaben

Geben Sie alle Informationen zu allen Abteilungen aus.

### Übung 4 – Angaben mit Einschränkung

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

### Übung 5 – Angaben mit Einschränkungen

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

### Übung 6 – Abfrage mit Sortierung

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

### Übung 7 – DISTINCT und ALL

Bitte überprüfen Sie die folgenden Befehle:

```
--  Variante 1
SELECT DISTINCT COUNT(*)
  FROM Mitarbeiter
 GROUP BY Abteilung_ID
--  Variante 2
SELECT ALL COUNT(*)
  FROM Mitarbeiter
 GROUP BY Abteilung_ID
```

Worin unterscheidet sich die Ausgabe? Welche wichtige Information fehlt vor allem bei Variante 2?

## Lösungen

Die Aufgaben beginnen auf Seite 65.

### Lösung zu Übung 1 – Richtig oder falsch?

Die Aussagen 1, 5, 7, 8, 10 sind richtig. Die Aussagen 2, 3, 4, 6, 9 sind falsch.

### Lösung zu Übung 2 – Pflichtangaben

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

### Lösung zu Übung 3 – Alle Angaben

```
SELECT * FROM Abteilung;
```

### Lösung zu Übung 4 – Angaben mit Einschränkung

```
SELECT * FROM Abteilung
 WHERE Ort = 'Bochum';
```

### Lösung zu Übung 5 – Angaben mit Einschränkungen

```
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 6 – Abfrage mit Sortierung

```
SELECT Kuerzel FROM Abteilung
 ORDER BY Ort;
```

### Lösung zu Übung 7 – DISTINCT und ALL

Variante 1 nennt jede Anzahl von Mitarbeitern pro Abteilung genau einmal. Bei Variante 2 gibt es diese Anzeige für jede Abteilung einzeln. Dabei fehlt vor allem die Angabe der *Abteilung_ID*, ohne die die Ausgabe ziemlich sinnlos ist.