Version 3
Oracle-Variante

Prof. Dr. Ingo Claßen
HTW Berlin

SQL-Übersicht

Inhaltsverzeichnis

1 Einleitung **2**

2 Beispieldatenmodell und Daten **2**

3 Tabellenerzeugung **2**

4 Datenmanipulation **4**

5 Einfach strukturierte Abfragen **5**
5.1 Abfragen auf einer Tabelle 5
5.2 Abfragen mit Null-Werten 6
5.3 Verbunde 7
5.4 Äußere Verbunde 8
5.5 Aggregationsfunktionen 9
5.6 Gruppierungen 9

6 Komplex strukturierte Abfragen **11**

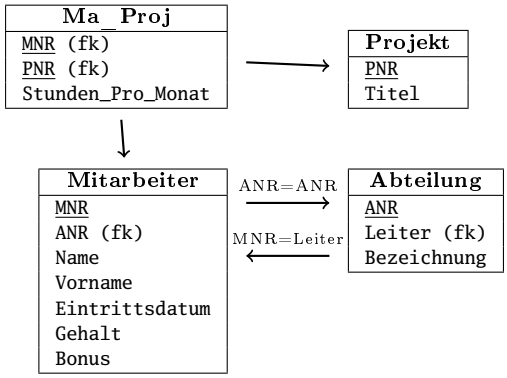
7 Sichten **13**

1 Einleitung

SQL (Structured Query Language) ist *die* Abfragesprache für relationale Datenbanksysteme. Ihre Kenntnis ist für Wirtschaftsinformatiker und -informatikerinnen im unerlässlich. Im Laufe der Jahre hat es verschiedene Standards gegeben, die von den Datenbankherstellern in unterschiedlichem Umfang eingehalten wurden.

Konkrete Datenbanksysteme liegen üblicherweise ein Stück quer dazu, d.h. einerseits erfüllen sie nur Teile des Standards, andererseits bieten sie Spezialitäten, die über ihn hinausgehen. Aus diesem Grund gibt es diese Arbeitsunterlage in verschiedenen Versionen, entsprechend des Datenbanksystems, das ich in der jeweiligen Vorlesung verwende. Die Gemeinsamkeiten unter den Systemen überwiegen und letztendlich ist das eingesetzte System zweitrangig, da die Konzepte der Sprache vermittelt werden sollen.

2 Beispieldatenmodell und Daten



Projekt	
PNR	Titel
P01	SAP HR Einfuehrung
P02	SAP PPS Einfuehrung
P03	Restrukturierung Vertrieb

Ma_Proj		
MNR	PNR	Stunden_Pro_Monat
M02	P01	20
M02	P02	5
M03	P01	10
M05	P01	10
M05	P02	10
M05	P03	10
M08	P02	15

Mitarbeiter						
MNR	ANR	Name	Vorname	Eintritts-datum	Gehalt	Bonus
M01	A01	Meier	Hans	01.01.1980	80.000	
M02	A01	Mueller	Sandra	07.01.1991	38.000	1.000
M03	A01	Krause	Helmut	05.01.1989	42.000	
M04	A02	Schrader	Christian	01.06.1982	85.000	2.000
M05	A02	Schimmel	Alfred	05.01.1977	91.000	
M06	A02	Blaschke	Joerg	10.01.1992	53.000	1.000
M07	A03	Kramer	Sabine	03.01.1995	50.000	1.000
M08	A03	Hensel	Fred	07.01.1994	20.000	2.500
M09	A04	Hermanns	Klaus	08.01.1994	21.000	1.500

Abteilung		
ANR	Leiter	Bezeichnung
A01	M01	Personal
A02	M04	Produktion
A03	M07	Lager
A04	M07	Versand

3 Tabellenerzeugung

Der Befehl `create table` erzeugt eine Tabelle in der Datenbank. Dabei müssen Spaltennamen und Datentypen angegeben werden.

```
create table ABTEILUNG (
  ANR char(3) not null,
  LEITER char(3),
  BEZEICHNUNG varchar2(100) not null,
  constraint PK_ABTEILUNG primary key (ANR)
```

```
);
```

Die Spalte ANR hat den Datentyp `char(3)`; es handelt sich hierbei um eine Zeichenkette fester Länge mit 3 Zeichen. Der Datentyp `varchar2(100)` beschreibt eine Zeichenkette variabler Länge mit maximal 100 Zeichen. Im Gegensatz zu `char` werden hier nur die Zeichen gespeichert, die die Zeichenkette tatsächlich hat. Die Kennzeichnung `not null` legt eine zwingende Dateneingabe für die Spalte fest. In dem Beispiel muss jede Abteilung eine Abteilungsnummer und eine Bezeichnung haben. Ein Leiter muss nicht vorhanden sein. Die Abteilungsnummer (ANR) ist der Primärschlüssel und wird durch die `primary key`-Klausel festgelegt. Der Name dieses so genannten *Constraints* ist PK_ABTEILUNG und hat keine Bedeutung für die Schlüsseldefinition an sich, sondern ist die Kennzeichnung des Schlüssels im *Data Dictionary*[1]

In der Tabelle MITARBEITER kommen als weitere Datentypen `date` für Datumsangaben und `decimal(n,m)` für gebrochene Zahlen mit fester Genauigkeit (insgesamt n Stellen, davon m hinter dem Komma) hinzu. Des Weiteren wird ein Fremdschlüssel ANR auf die Abteilungstabelle mit der `foreign key`-Klausel definiert.

```
create table MITARBEITER (
  MNR char(3) not null,
  ANR char(3) not null,
  NAME varchar2(200) not null,
  VORNAME varchar2(200) not null,
  EINTRITTSDATUM date not null,
  GEHALT decimal(9,2) not null,
  BONUS  decimal(9,2),
  constraint PK_MITARBEITER primary key (MNR),
  constraint FK_MA_ABT foreign key (ANR)
    references ABTEILUNG (ANR),
  constraint CHK_GEHALT
    check(GEHALT > 10000)
);
```

Die `check`-Klausel legt Bedingungen für die Spalten einer Tabelle fest, die durch das Datenbanksystem überprüft werden. In diesem Fall können nur Gehälter größer als 10.000 Euro eingegeben werden.

Da in der Abteilungstabelle die Spalte LEITER ebenfalls ein Fremdschlüssel sein soll, wird das mit der folgenden `alter table`-Anweisung festgelegt. Ein direkte Angabe in der Abteilungstabelle war nicht möglich, da zu dem Zeitpunkt die Mitarbeitertabelle noch nicht existierte. Mit `alter table` lassen sich nachträglich in begrenztem Maße Änderungen an einer Tabelle vornehmen.

```
alter table ABTEILUNG add constraint FK_LEITER
  foreign key (LEITER)references MITARBEITER (MNR);
```

[1] Ein relationales Datenbanksystem verwaltet seine Metadaten, d.h. Daten über Tabellen, Spalten, Schlüssel usw., selbst in Tabellen. Diese Metadaten bilden das *Data Dictionary*.

4 Datenmanipulation

Die SQL-Anweisungen zur Datenmanipulation ermöglichen das Einfügen, Verändern und Löschen von Daten. Das soll am Beispiel der Tabelle `PERSON` erläutert werden.

```
create table PERSON (
  NAME varchar2(200) not null,
  VORNAME varchar2(200) not null ,
  constraint PK_PERSON primary key (NAME)
);
```

Das Einfügen von Daten erfolgt mit der Anweisung `insert into`. Die Werte der einzeln Spalten werden hinter der `values`-Klausel angegeben.

```
insert into PERSON
  values ('Franzen', 'Edgar');
insert into PERSON
  values ('Holler', 'Marianne');
```

```
NAME      VORNAME
--------- ---------
Franzen   Edgar
Holler    Marianne
```

Es ist auch möglich, Inhalte aus einer anderen Tabelle einzufügen. Wird die `values`-Klausel durch eine `select`-Klausel ersetzt, so werden die ausgewählten Zeilen der anderen Tabelle übernommen. Die Anzahl und Datentypen der selektierten Spalten müssen passen.

```
insert into PERSON
  select NAME, VORNAME
  from MITARBEITER
  where ANR = 'A03';
```

```
NAME      VORNAME
--------- ---------
Franzen   Edgar
Hensel    Fred
Holler    Marianne
Kramer    Sabine
```

Die `update`-Anweisung verändert Daten. In der `set`-Klausel werden die zu ändernden Spalten und ihre neuen Werte angegeben. Die `where`-Klausel schränkt die Veränderungen auf bestimmte Zeilen ein. Fehlt sie, werden alle Zeilen modifiziert.

```
update PERSON
  set NAME = 'Grabe'
  where NAME = 'Franzen';
```

```
NAME      VORNAME
--------- ---------
Grabe     Edgar
Hensel    Fred
Holler    Marianne
Kramer    Sabine
```

Die `delete`-Anweisung ist für das Löschen zuständig. Die `where`-Klausel legt den Bereich der Löschung fest. Im Beispiel werden alle Person gelöscht, deren Nachname mit einem *H* beginnen. Das Prozentzeichen steht für eine beliebige Zeichenkette.

```
delete from PERSON
  where NAME like 'H%';
```

```
NAME      VORNAME
--------- ---------
Grabe     Edgar
Kramer    Sabine
```

5 Einfach strukturierte Abfragen

Einfach strukturierte Abfragen beinhalten die Auswahl von Zeilen und Spalten einer Tabelle, die Verbindung von Tabellen über gemeinsame Merkmale und die Gruppierung (Aggregation) von Daten nach festgelegten Kriterien.

5.1 Abfragen auf einer Tabelle

Die Ausgabe der kompletten Abteilungstabelle wird mit folgenden SQL-Anweisungen bewerkstelligt. Der Stern (*) steht als Abkürzung für alle Spalten. Will man nur bestimmte Spalten ausgeben, so gibt man nur diese an.

```
select *
from ABTEILUNG;

select
  ANR, LEITER,
  BEZEICHNUNG
from ABTEILUNG;
```

```
ANR  LEITER BEZEICHNUNG
---- ------ -----------
A01  M01    Personal
A02  M04    Produktion
A03  M07    Lager
A04  M07    Versand
```

In der folgenden Abfrage werden Werte in einer Spalte berechnet. Es sollen `MNR`, `NAME` und Unternehmenszugehörigkeit aller Mitarbeiter in Jahren ausgegeben werden.

```
select MNR, NAME,
  extract(
    year from current_date)-
  extract(
    year from EINTRITTSDATUM)
  AS JAHRE
from MITARBEITER;
```

```
MNR  NAME      JAHRE
---- --------- -----------
M01  Meier     24
M02  Mueller   13
M03  Krause    15
M04  Schrader  22
M05  Schimmel  27
M06  Blaschke  12
M07  Kramer    9
M08  Hensel    10
M09  Hermanns  10
```

Die Funktion `current_date` stellt das aktuelle Systemdatum bereit. Die `extract(year from datum)`-Funktion extrahiert die Jahresangabe aus einem Datum.

Bisher wurden immer alle Zeilen ausgegeben und nur Spalten manipuliert. Die Einschränkung auf bestimmte Zeilen erfolgt mit der `where`-Klausel.

```
select NAME,GEHALT
from MITARBEITER
where ANR = 'A02';
```

```
NAME      GEHALT
--------- -----------
Schrader  85000.00
Schimmel  91000.00
Blaschke  53000.00
```

Bedingungen in einer `where`-Klausel können logisch mit `and`, `or` oder `not` verknüpft werden.

```
select NAME,GEHALT
from MITARBEITER
where
  ANR = 'A02'
  and
  EINTRITTSDATUM
    > '01-JAN-1980';
```

```
NAME      GEHALT
--------- -----------
Schrader  85000.00
Blaschke  53000.00
```

5.2 Abfragen mit Null-Werten

Null-Werte spielen bei relationalen Datenbanken eine besondere Rolle. Sie sind nicht mit Werten wie die Zahl *0* und die leere Zeichenkette zu verwechseln. Sie zeigen das Nichtvorhandensein einer Information an. Dafür gibt es verschiedene Gründe.

- Die Information wird bewusst nicht bereit gestellt. Bei der Angabe von persönlichen Informationen kann man z.B. die private Telefonnummer nicht mit angeben.
- Die Information ist prinzipiell vorhanden aber zur Zeit der Dateneingabe nicht bekannt. Zum Beispiel könnte das Geburtsdatum einer Person nicht bekannt sein.
- Die Information ist noch nicht vorhanden. Ein Beispiel ist das Exmatrikulationsdatum in einem Studierenden-Datensatz. Dieses existiert erst nach der Exmatrikulation.
- Die Spalte macht für den betreffenden Datensatz keinen Sinn. In einem Hochschulverwaltungssystem könnte es eine Personentabelle geben, die Daten von Studierenden und Professorinnen und Professoren enthält. Die Spalte `Matrikelnummer` würde dann nur für Studierende Sinn machen.

Die Selektion aller Mitarbeiter ohne Bonus erfolgt mit der `is null`-Klausel.

```
select ANR, NAME
from MITARBEITER
where BONUS is null;
```

```
ANR  NAME
---- ---------
A01  Meier
A01  Krause
A02  Schimmel
```

Mit der folgenden Abfrage sollen alle Mitarbeiter ausgegeben werden, deren Summe aus Gehalt und Bonus den Wert von 60.000 Euro übersteigt.

```
select ANR, NAME
from MITARBEITER
where
  (GEHALT + BONUS) > 60000;
```

```
ANR  NAME
---- ---------
A02  Schrader
```

Der erste Versuch bringt nicht das gewünschte Ergebnis, da z.B. Mitarbeiter, deren Gehalt alleine 60.000 Euro übersteigt, nicht im Ergebnis auftreten. Das hängt mit der besonderen Behandlung von Null-Werten zusammen. Es werden nur die Zeilen ausgegeben, in denen ein Bonuswert vorhanden ist. Eine Lösung für das Problem ist die explizite Umwandlung von Null-Werten in die Zahl *0* wie im nächsten Beispiel.

```
select ANR, NAME
from MITARBEITER
where
  (GEHALT +
    case
      when BONUS is null
      then 0
      else BONUS
    end) > 60000
```

```
ANR  NAME
---- ---------
A01  Meier
A02  Schrader
A02  Schimmel
```

Im Gegensatz zu vorigem Beispiel werden auch die Datensätze ausgegeben, in denen zwar kein Bonus vorhanden ist, bei denen das Gehalt alleine schon über 60.000 Euro liegt. Mit der `case`-Funktion lassen sich beliebige Spaltenumrechnungen durchführen.

5.3 Verbunde

Ein normalisierter Datenbankentwurf führt dazu, dass zusammengehörende Informationen wie z.B. Abteilungsnummer und -bezeichnung in einer Tabelle gespeichert werden. Informationen bzgl. unterschiedlicher Einheiten wie Personen und Abteilungen befinden sich dann in verschiedenen Tabellen. Verbundoperationen bringen diese Informationen wieder zusammen, wobei meistens Fremdschlüssel die Verbindungsglieder sind.

In der Mitarbeitertabelle befindet sich z.B. ein Fremdschlüssel auf die Abteilungstabelle. Möchte man Mitarbeiternamen zusammen mit zugehörigen Abteilungsbezeichnungen ausgeben, so geht das mit folgender SQL-Anweisung.

```
select
  NAME, BEZEICHNUNG
from
  MITARBEITER M,
  ABTEILUNG A
where
  A.ANR = M.ANR;
```

```
NAME      BEZEICHNUNG
--------- -----------
Meier     Personal
Mueller   Personal
Krause    Personal
Schrader  Produktion
Schimmel  Produktion
Blaschke  Produktion
Kramer    Lager
Hensel    Lager
Hermanns  Versand
```

Die Bedingung `A.ANR = M.ANR` selektiert aus dem kartesischen Produkt der Tabellen `ABTEILUNG` und `MITARBEITER` die Zeilen heraus, bei denen die Abteilungsnummern übereinstimmen. Das führt dazu, dass aus der Menge aller Mitarbeitern-Abteilungs-Kombinationen nur diejenigen im Ergebnis erscheinen, bei denen die Abteilungszuordnung stimmt. Es handelt sich bei dem Beispiel um einen Gleichheitsverbund, da die Verbundbedingung (`A.ANR = M.ANR`) ein Gleichheitsprädikat hat. Der Verbund ist typisch, da hier der Fremdschlüssel einer Tabelle mit dem Primärschlüssel der anderen in Beziehung gesetzt wird. Das muss nicht der Fall sein, wie folgendes Beispiel zeigt.

```
select M1.MNR, M2.MNR
from
  MITARBEITER M1,
  MITARBEITER M2
where
  M1.GEHALT > M2.GEHALT and
  (M1.MNR = 'M02' or
   M1.MNR = 'M03')
order by M1.MNR;
```

```
MNR  MNR
---- ----
M02  M08
M02  M09
M03  M02
M03  M08
M03  M09
```

Bei dem Beispiel geht es um einen Gehaltsvergleich. Es werden Paare von Mitarbeiternummern ausgegeben, bei denen der Mitarbeiter mit der ersten Nummer mehr als der mit der zweiten verdient. Es werden nur die Gehaltsvergleiche bezüglich Mitarbeiter 'M02' und 'M03' betrachtet. Zu beachten ist die Klammerung des Oder-Ausdrucks. Ohne diese Klammern kommt ein anderes Ergebnis heraus (welches ?). Die Ausgabe wird nach Mitarbeiternummern sortiert.

Die Verbundoperation lässt sich auf mehr als zwei Tabellen ausdehnen, wie das Beispiel mit der Projektbeteiligung von Mitarbeitern zeigt. Hier spielen drei Tabellen eine Rolle: `MITARBEITER`, `MA_PROJ` und `PROJEKT`, da eine n-zu-m-Beziehung zwischen Mitarbeitern und Projekten besteht.

```
select
  M.MNR, NAME, TITEL
from
  MITARBEITER M,
  MA_PROJ MP,
  PROJEKT P
where
  M.MNR = MP.MNR and
  MP.PNR = P.PNR;
```

```
MNR  NAME      TITEL
---- --------- -------------------------
M02  Müller    SAP HR Einfuehrung
M02  Müller    SAP PPS Einfuehrung
M03  Krause    SAP HR Einfuehrung
M05  Schimmel  SAP HR Einfuehrung
M05  Schimmel  SAP PPS Einfuehrung
M05  Schimmel  Restrukturierung Vertrieb
M08  Hensel    SAP PPS Einfuehrung
```

Da der Verbund nur die Zeilen im Ergebnis aufführt, die miteinander verbunden sind, tauchen nur Mitarbeiter im Ergebnis auf, die mindestens einem Projekt zugeordnet sind.

5.4 Äußere Verbunde

Es sollen die Namen aller Mitarbeiter ausgegeben werden zusammen mit der Bezeichnung der Abteilung, die sie leiten. Die Abteilungsbezeichnung darf nur bei den Mitarbeitern erscheinen, die auch tatsächlich Abteilungsleiter sind. Dazu müssen die Tabellen `MITARBEITER` und `ABTEILUNG` kombiniert werden. Allerdings versagt hier der Verbund, da er nur die Zeilen der Mitarbeiter ausgeben würde, deren Wert in der Spalte `MNR` mit Werten in der Spalte `LEITER` aus der Abteilungstabelle übereinstimmen. Damit würden aber nur die Abteilungsleiter im Ergebnis erscheinen und nicht alle Mitarbeiter. Da man in diesem Fall alle Mitarbeiter sehen möchte, benötigt man eine Spezialform des Verbunds, den so genannten *äußeren Verbund*.

```
select                          MNR  NAME      LEITET
  MNR, NAME,                    ---- --------- ----------
  BEZEICHNUNG as LEITET         M01  Meier     Personal
from                            M02  Mueller   NULL
  MITARBEITER                   M03  Krause    NULL
  left outer join               M04  Schrader  Produktion
  ABTEILUNG                     M05  Schimmel  NULL
  on MNR = LEITER;              M06  Blaschke  NULL
                                M07  Kramer    Lager
                                M07  Kramer    Versand
                                M08  Hensel    NULL
                                M09  Hermanns  NULL
```

Es gibt drei Formen äußerer Verbunde: den linken, rechten und vollen Außenverbund. Im Beispiel wurde ein linker Außenverbund angewendet (`left outer join`). Er gibt alle Zeilen der vor ihm stehenden Tabelle aus. Das Attribut `LEITET` aus der rechten Tabelle wird nur in den Zeilen ausgegeben, die verbunden sind. In allen anderen Zeilen hat `LEITET` einen Null-Wert, was in der Ausgabe mit der Zeichenkette gekennzeichnet ist.

5.5 Aggregationsfunktionen

Aggregation bedeutet die Zusammenfassung bzw. Verdichtung von Informationen. Im relationalen Datenmodell werden Daten aus mehreren Zeilen in einer Zeile aggregiert, z.B. bei der Mittelwertbildung.

```
select avg(GEHALT)                DURCHSCHNITTSGEHALT
  as DURCHSCHNITTSGEHALT          --------------------
from MITARBEITER;                 53333.333333
```

Die Zeilen der Tabelle `MITARBEITER` werden zu einer Zeile zusammengefasst und im Ergebnis wird nur eine berechnete Spalte `DURCHSCHNITTSGEHALT` ausgegeben. Der Spaltenwert in der einzigen Ausgabezeile wird als Mittelwert der Spalte `GEHALT` bzgl. aller Zeilen der Ausgangstabelle gebildet. Die Funktion `avg` ist eine so genannte Aggregationsfunktionen. Weitere Beispiele sind `max`, `min` und `count` für die Maximum-, Minimumbildung bzw. für die Zeilenanzahlberechnung.

5.6 Gruppierungen

Gruppierungen sind eine spezielle Form der Aggregation bei der die Zeilen einer Tabelle nach definierten Kriterien in Gruppen eingeteilt werden. Auf diese lassen sich Aggregationsfunktionen anwenden. In der `MITARBEITER`-Tabelle kann man z.B. nach Abteilungsnummern gruppieren und dann Maximal-, Minimalgehalt und Gehaltssumme pro Abteilung berechnen.

```
select ANR,
  max(GEHALT)as MAX,
  min(GEHALT) as MIN,
  sum(GEHALT) as SUM
from MITARBEITER
group by ANR;
```

```
ANR  MAX          MIN          SUM
---- ---------- ---------- ---------
A01  80000.00    38000.00    160000.00
A02  91000.00    53000.00    229000.00
A03  50000.00    20000.00    70000.00
A04  21000.00    21000.00    21000.00
```

Im Ergebnis erscheint nur eine Zeile für jede Abteilung. Die Funktionen `min`, `max` und `sum` werden auf alle Datensätze der jeweiligen Abteilung angewendet.

Die `count`-Funktion tritt in zwei Varianten auf, wobei `count(*)` alle Zeilen des Ergebnisses zählt und `count(BONUS)` nur die Anzahl der Zeilen, die keine Null-Werte in der Bonusspalte haben.

```
select
  ANR, count(*)
    as ANZ_MA
from MITARBEITER
group by ANR;
```

```
ANR  ANZ_MA
---- -----------
A01  3
A02  3
A03  2
A04  1
```

```
select
  ANR,
  count(BONUS)
  as ANZAHL_BONUS
from MITARBEITER
group by ANR;
```

```
ANR  ANZAHL_BONUS
---- ------------
A01  1
A02  2
A03  2
A04  1
```

An den Bespielen wird der Einfluss von Null-Werten auf die Berechnung sichtbar. Noch deutlicher wird das bei der Aggregationsfunktion `avg`, da sie nur die Zeilen ohne Null-Werte berücksichtigt. Würde stattdessen für alle Mitarbeiter ohne Bonus als Wert die Zahl *0* angenommen, so ergäbe sich ein anderer Mittelwert.

Eine Auswahl aus den gebildeten Gruppen trifft man mit der `having`-Klausel. Im folgenden Beispiel werden bei der Durchschnittsberechnung des Bonus nur die Abteilungen betrachtet, in denen mindestens zwei Mitarbeiter einen Bonus haben.

```
select
  ANR,
    avg(BONUS) as AVG_BON
from MITARBEITER
group by ANR
having  count(BONUS) >= 2;
```

```
ANR  AVG_BON
---- -----------
A02  1500.00
A03  1750.00
```

Die Klausel `having count(BONUS) >= 2` entspricht einer `where`-Klausel auf dem Ergebnis der Gruppierung. D.h. die Ergebniszeilen der Gruppierung werden bezüglich der gegebenen Bedingung geprüft.

6 Komplex strukturierte Abfragen

Komplex strukturierte Abfragen sind durch die Schachtelung von Tabellenausdrücken gekennzeichnet. Das folgende Beispiel zeigt ihre Notwendigkeit auf. Wenn wir alle Mitarbeiter auswählen wollen, deren Gehalt über dem Durchschnitt liegt, so müssen wir den Wert des Durchschnittgehalts kennen. Nehmen wir an, er beträgt 57.375 Euro. Dann lässt sich die Auswahl der entsprechenden Mitarbeiter mit einer einfachen `select`-Anweisung bewerkstelligen.

```
select MNR, NAME, GEHALT
from MITARBEITER
where GEHALT > 57375;
```

```
MNR  NAME      GEHALT
---- --------- -----------
M01  Meier     80000.00
M04  Schrader  85000.00
M05  Schimmel  91000.00
```

Ein Nachteil dieser Abfrage ist die notwendige Kenntnis des Durchschnittgehalts, das sich mit Modifikationen an den Gehälter ändert. Dieses Problem kann man umgehen, indem das Durchschnittgehalt in der Abfrage berechnet wird.

```
select MNR, NAME, GEHALT
from MITARBEITER
where
  GEHALT > (
    select avg(GEHALT)
    from MITARBEITER
  );
```

```
MNR  NAME      GEHALT
---- --------- -----------
M01  Meier     80000.00
M04  Schrader  85000.00
M05  Schimmel  91000.00
```

Das Beispiel beinhaltet eine Unterabfrage, die als Ergebnis den Durchschnittswert aller Gehälter abliefert. Damit dieses Ergebnis mit dem Gehalt der jeweiligen Mitarbeiter verglichen werden kann, muss die Unterabfrage eine Tabelle mit einer einzigen Zeile und Spalte ergeben. Ein solche degenerierte Tabelle kann als Wert interpretiert werden.

Im folgenden Beispiel werden die Mitarbeiter mit Maximalgehalt pro Abteilung ausgegeben.

```
select MNR, NAME, GEHALT
from MITARBEITER M1
where
  GEHALT = (
    select max(GEHALT)
    from MITARBEITER M2
    where M2.ANR = M1.ANR
  )
order by M1.MNR;
```

```
MNR  NAME      GEHALT
---- --------- -----------
M01  Meier     80000.00
M05  Schimmel  91000.00
M07  Kramer    50000.00
M09  Hermanns  21000.00
```

Hier ist die Unterabfrage mit der übergeordneten Abfrage *korreliert*. D.h. die Ergebniszeilen der Unterabfrage beziehen sich auf die jeweilige Ergebniszeile der Oberabfrage. Das entspricht einer Schleife, bei der für jeden Datensatz der Tabelle `M1` alle Datensätze der Tabelle `M2` durchlaufen werden, wobei nur bzgl. der Datensätze das Maximum gebildet wird, die in der Abteilungsnummer übereinstimmen.

Die folgenden Beispiele zeigen Varianten für die Abfrage nach allen Mitarbeitern, die in zwei Projekte eingebunden sind.

```
select MNR, NAME                                    MNR  NAME
from MITARBEITER M                                  ---- ---------
where                                               M02  Mueller
  (select count(*)
   from MA_PROJ
   where MA_PROJ.MNR = M.MNR) = 2;
```

Die erste Variante wird durch eine korrelierte Unterabfrage gebildet, wobei sich die Unterabfrage – wie bei den vorherigen Beispielen – in der `where`-Klausel der übergeordneten Abfrage befindet.

```
select M.MNR, NAME                                  MNR  NAME
from                                                ---- ---------
  MITARBEITER M,                                    M02  Mueller
  (select MNR, count(PNR) as ANZAHL
   from MA_PROJ
   group by MNR) X
where
  M.MNR = X.MNR and
  X.ANZAHL = 2;
```

Die zweite Variante setzt die Unterabfrage in der `from`-Klausel ein, wobei sie an dieser Stelle einen Aliasnamen haben muss, in diesem Fall `X`.

```
select M.MNR, NAME                                  MNR  NAME
from                                                ---- ---------
  MITARBEITER M, MA_PROJ MP                         M02  Mueller
where
  M.MNR = MP.MNR
group by M.MNR, NAME
having count(PNR) = 2;
```

Die dritte Variante kommt ohne Unterabfrage aus.

In der folgenden Abfrage werden die Mitarbeiter ausgegeben, die an den meisten Projekten beteiligt sind.

```
select M.MNR, NAME                                  MNR  NAME
from                                                ---- ---------
  MITARBEITER M,                                    M05  Schimmel
  (select MNR, count(PNR) as ANZAHL
   from MA_PROJ
   group by MNR) X
where
  M.MNR = X.MNR and
  X.ANZAHL >= all
    (select count(PNR)
     from MA_PROJ
     group by MNR);
```

Ein neues Sprachelement in diesem Beispiel ist die `all`-Klausel (`X.ANZAHL >= all ...`). Die Unterabfrage hinter `all` liefert eine Tabelle mit den Projektbeteilungsanzahlen aller Mitarbeiter ab. Damit diese Tabelle mit dem skalaren Wert `X.ANZAHL` verglichen werden kann, ist `all` notwendig. Damit wird festgelegt, dass `X.ANZAHL` größer oder gleich alle Werte aus der Unterabfrage ist.

Mit der `in`-Klausel kann ein Enthaltensein festgestellt werden. Im Beispiel werden alle Mitarbeiter ausgegeben, die `'M07'` als Chef haben, der zwei Abteilungen leitet. Die Unterabfrage ermittelt die Menge der entsprechenden Abteilungsnummern und die Oberabfrage stellt das Enthaltensein der Abteilungsnummern der Mitarbeiter in dieser Menge fest.

```
select MNR, Name
from MITARBEITER M
where M.ANR in
  (select ANR
   from ABTEILUNG
   where LEITER = 'M07');
```

```
MNR  Name
---- ---------
M07  Kramer
M08  Hensel
M09  Hermanns
```

Äquivalent dazu kann die `any`-Klausel eingesetzt werden.

```
select MNR, Name
from MITARBEITER M
where M.ANR = any
  (select ANR
   from ABTEILUNG
   where LEITER = 'M07');
```

```
MNR  Name
---- ---------
M07  Kramer
M08  Hensel
M09  Hermanns
```

Die `exists`-Klausel stellt fest, ob das Ergebnis einer Abfrage mindestens eine Zeile enthält. Was liefert das folgende Beispiel?

```
select MNR, NAME
from MITARBEITER M1
where
  exists
   (select *
    from MITARBEITER M2
    where M2.EINTRITTSDATUM >
          M1.EINTRITTSDATUM and
          M2.GEHALT > M1.GEHALT);
```

```
MNR  NAME
---- ---------
M01  Meier
M02  Mueller
M03  Krause
M08  Hensel
M09  Hermanns
```

7 Sichten

Sichten sind mit einem Namen versehene gespeicherte Abfragen, die in anderen Abfragen wie Tabellen eingesetzt werden können.

```
create view PRODUKTION as
  select *
  from MITARBEITER
  where ANR='A02'
```

Die Sicht PRODUKTION wird im folgenden Beispiel wie eine Tabelle verwendet.

```
select MNR, NAME
from PRODUKTION;
```

```
MNR  NAME
---- ---------
M04  Schrader
M05  Schimmel
M06  Blaschke
```

In begrenztem Umfang können Sichten auch zur Datenmanipulation wie Einfügungen, Löschungen und Änderungen verwendet werden. Dazu darf die Sicht nur auf einer Tabelle in der from-Klausel basieren. Die genauen Regel hängen vom verwendeten Datenbanksystem ab und müssen in der Dokumentation nachgeschlagen werden.

```
insert into Produktion values
  ('M10', 'A02', 'Feder', 'Klaus', '01-JAN-1997', 80000, null);
```

Besitzt eine Sicht eine where-Klausel, so kann mit check option die angegebene Bedingung bei der Eingabe und Änderung von Daten über die Sicht überprüft werden.

```
create view PRODUKTION1 as
  select *
  from MITARBEITER
  where ANR='A02'
with check option
```

Ein Einfügung eines Mitarbeiter mit einer falschen Abteilungsnummer ist dann nicht mehr möglich. Die folgende SQL-Anweisung würde vom Datenbanksystem zurückgewiesen werden.

```
insert into Produktion1 values
  ('M11', 'A01', 'Baum', 'Karl', '01-JUL-1995', 60000, null);
```

Sichten können auch mittels komplex strukturierter Abfragen gebildet werden, wie das folgende Beispiel zeigt.

```
create view ABTLEITER as
  select * from MITARBEITER
  where MNR in
    (select LEITER
     from ABTEILUNG)
```

Des Weiteren kann eine Sicht auf Verbunden beruhen. In diesem Fall ist eine Manipulation von Daten über die Sicht nicht mehr möglich.

```
create view PROJ_MA_STD as
  select TITEL, M.MNR, NAME, VORNAME, STUNDEN_PRO_MONAT
  from PROJEKT P, MA_PROJ MP, MITARBEITER M
  where P.PNR = MP.PNR and MP.MNR = M.MNR
```