# 10 SQL-Basics: Erweiterte Abfragen mit SELECT

Im Mittelpunkt dieses Kapitels steht nicht nur die einfache Auswahl von Spalten, sondern auch die Gestaltung eines Abfrageergebnisses mit Hilfe von Aggregatfunktionen und Gruppierungen.

# SQL-Basics: Erweiterte Abfragen mit SELECT

Nach den relativ einfachen `SELECT`-Anweisungen des letzten Kapitels lernen Sie in diesem Kapitel anspruchsvollere `SELECT`-Anweisungen kennen. Die Stichworte dazu sind:

- Aggregatfunktionen
- Gruppierungen
- Unterabfragen

## Aggregatfunktionen

Zu den am häufigsten verwendeten Funktionen unter SQL gehören die *Aggregatfunktionen*.

Aggregatfunktionen dienen zur Berechnung für eine ausgewählte Menge von Werten und geben aus dieser Menge lediglich einen einzelnen Wert zurück. Bei dieser Berechnung ignorieren Aggregatfunktionen – mit Ausnahme der Funktion `COUNT` – alle eventuell vorhandenen `NULL`-Werte in der Berechnungsmenge.

In der folgenden Tabelle finden Sie eine Aufstellung der Aggregatfunktionen mit einer kurzen Beschreibung.

| **Aggregatfunktion** | **Beschreibung** |
|---|---|
| `MIN()` | Niedrigster Wert in einem Spaltenausdruck |
| `MAX()` | Höchster Wert in einem Spaltenausdruck |
| `SUM()` | Summe der Werte in einem numerischen Ausdruck |

| | |
|---|---|
| AVG() | Durchschnitt der Werte in einem numerischen Spaltenausdruck |
| COUNT() | Anzahl der Werte in einem Spaltenausdruck |
| COUNT(*) | Anzahl der ausgewählten Zeilen |

**Tabelle 1: Aggregatfunktionen und ihre Beschreibung**

In den folgenden Abschnitten finden Sie Beispiele zur Anwendung der einzelnen Aggregatfunktionen.

## MIN()

Die Aggregatfunktion `MIN()` gibt als Resultatset den kleinsten Wert eines Ausdrucks zurück.

Das folgende Beispiel zeigt als Resultatset den Titel des Buches mit den niedrigsten Verkaufszahlen an.

```
SELECT MIN(ytd_sales)
AS 'Schlechtester Verkauf'
FROM titles
```

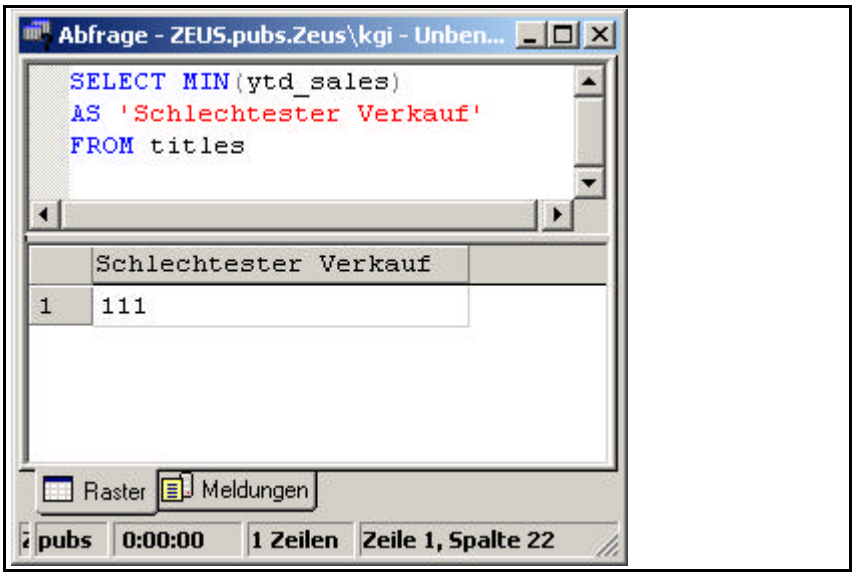


**Abbildung 1: Beispiel zur Funktion `MIN()` I**

Die Aggregatfunktion `MIN()` kann auch auf Spalten mit Zeichenketten angewandt werden. Das Resultatset gibt dann den kleinsten Wert abhängig von der Sortierreihenfolge zurück.

Das folgende Beispiel zeigt gemäß der Sortierreihenfolge den ersten Buchtitel an.

```
SELECT MIN(title)
FROM titles
```

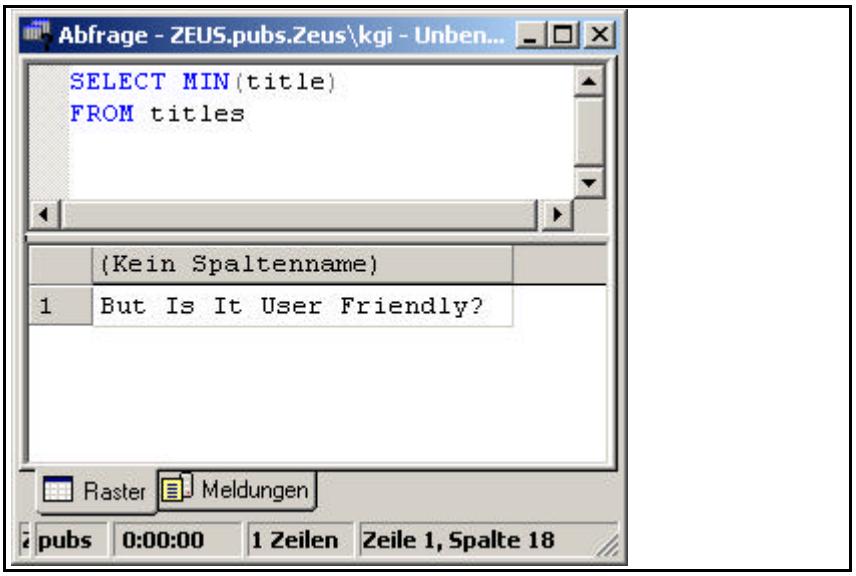


**Abbildung 2: Beispiel zur Funktion `MIN()` II**

## MAX()

Die Aggregatfunktion `MAX()` gibt als Resultatset den größten Wert eines Ausdrucks zurück.

Das folgende Beispiel zeigt als Resultatset den Titel des Buches mit den höchsten Verkaufszahlen an.

```
SELECT MAX(ytd_sales)
AS 'Bester Verkauf'
FROM titles
```

Bei der Abfrage von Zeichenkettenspalten gibt `MAX()` folgerichtig auch den größten Wert abhängig von der Sortierreihenfolge zurück.

**Aggregatfunktionen**

## SUM()

Die Aggregatfunktion `SUM()` gibt als Resultatset die Summe eines Ausdrucks zurück.

Das folgende Beispiel zeigt als Resultatset die Summe aller Verkaufszahlen an.

```
SELECT SUM(ytd_sales)
AS 'Summe der Verkäufe'
FROM titles
```

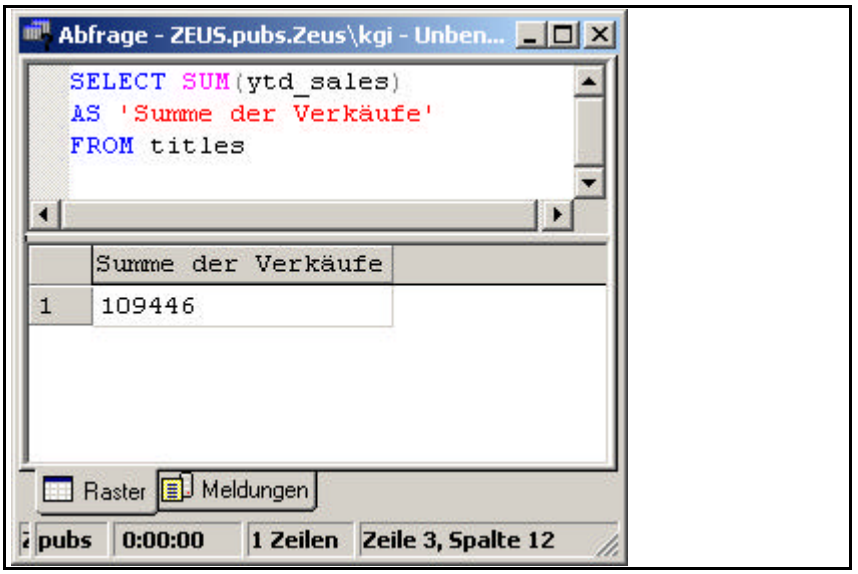


**Abbildung 3: Beispiel zur Funktion `SUM()`**

## AVG()

Die Aggregatfunktion `AVG()` gibt als Resultatset den Durchschnitt eines Ausdrucks zurück.

Das folgende Beispiel zeigt als Resultatset den Durchschnittswert aller Verkaufszahlen an:

```
SELECT AVG(ytd_sales)
AS 'Durchschnitt der Verkäufe'
FROM titles
```

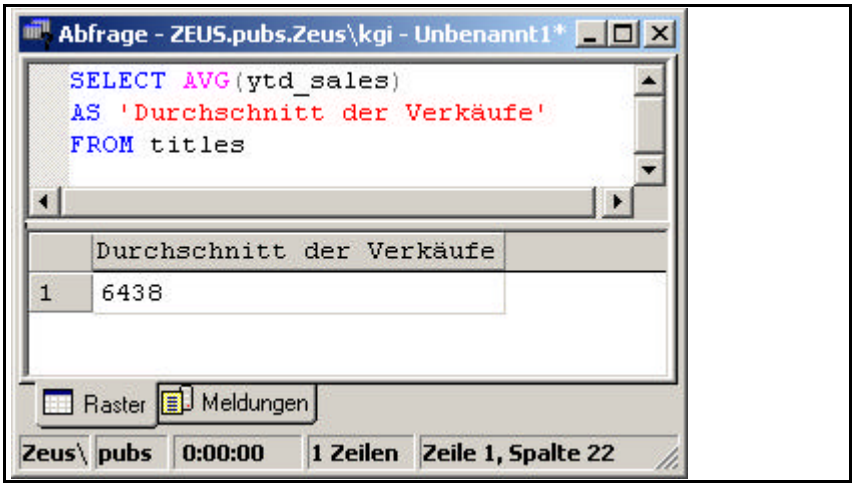


**Abbildung 4: Beispiel zur Funktion AVG( )**

### Kombination von Aggregatfunktionen

Aggregatfunktionen können auch miteinander kombiniert werden.

Das folgende Beispiel gibt als Resultatset den durchschnittlichen Vorschuss und die Summe der Verkäufe für Bücher aus dem Bereich Psychologie zurück.

```
SELECT AVG(advance) AS 'Durchschnitt',
SUM(ytd_sales) AS 'Summe'
FROM titles
WHERE type = 'psychology'
```

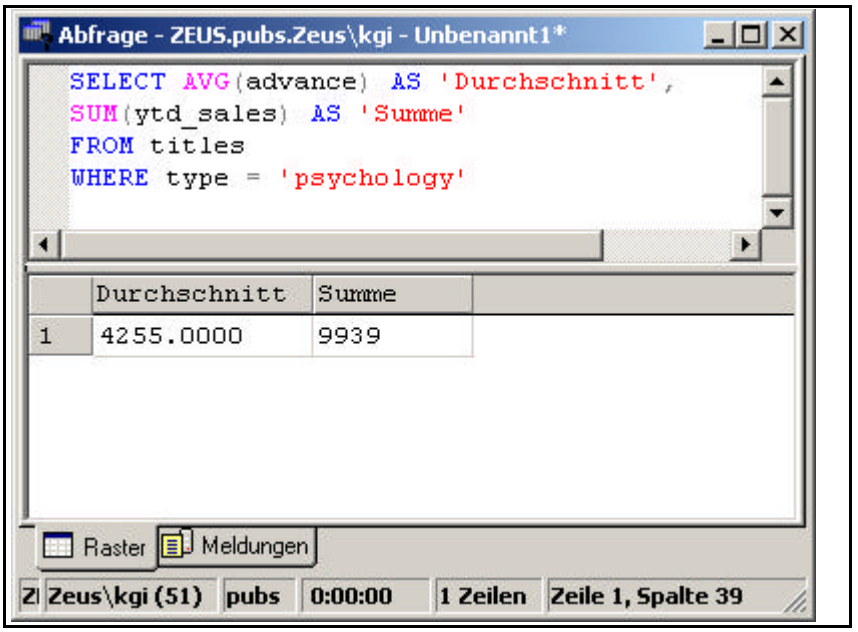


**Abbildung 5: Beispiel zur Kombination von Aggregatfunktionen**

## COUNT()

Die Aggregatfunktion `COUNT()` gibt als Resultatset die Anzahl von Datensätzen in einer Gruppe zurück.

Die `SELECT`-Anweisung

```
SELECT COUNT (type)
FROM titles
```

zeigt die Anzahl der Buchkategorien an, wobei `NULL`-Werte wiederum ignoriert werden.


**Abbildung 6: Beispiel zur Funktion `COUNT()` I**

Wenn Sie `COUNT()` in Verbindung mit dem Schlüsselwort `DISTINCT` verwenden, gibt das letzte Beispiel jede Buchkategorie nur einmal zurück:

```
SELECT COUNT(DISTINCT type)
FROM titles
```

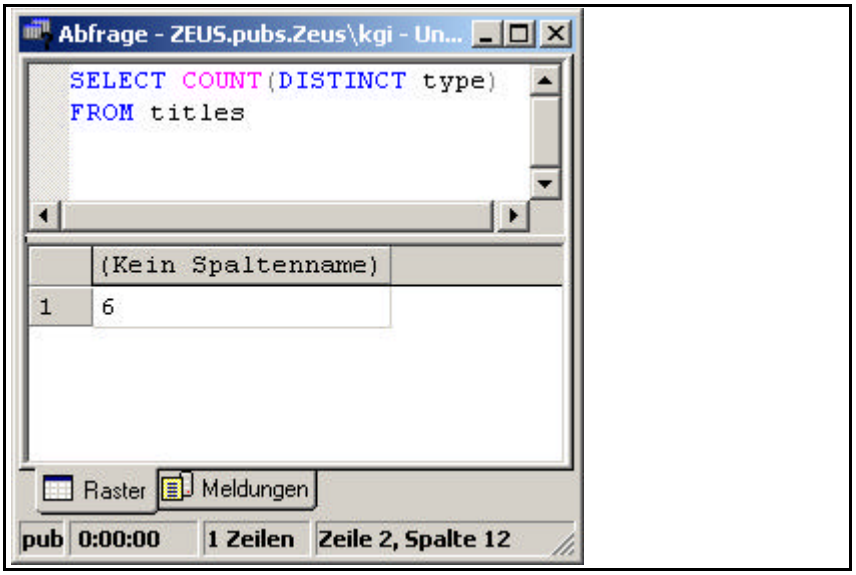


**Abbildung 7: Beispiel zur Funktion COUNT() II**

Falls die `SELECT`-Anweisung eine `WHERE`-Klausel erhält, liefert `COUNT()` als Resultatset die Anzahl der Datensätze zurück, die den Bedingungen der `WHERE`-Klausel entsprechen.

Das folgende Beispiel gibt die Anzahl der Titel zurück, deren Vorschuss unter dem Wert 5.000 liegt:

```
SELECT COUNT(title)
AS 'Vorschuss kleiner als 5000'
FROM titles
WHERE advance < 5000
```

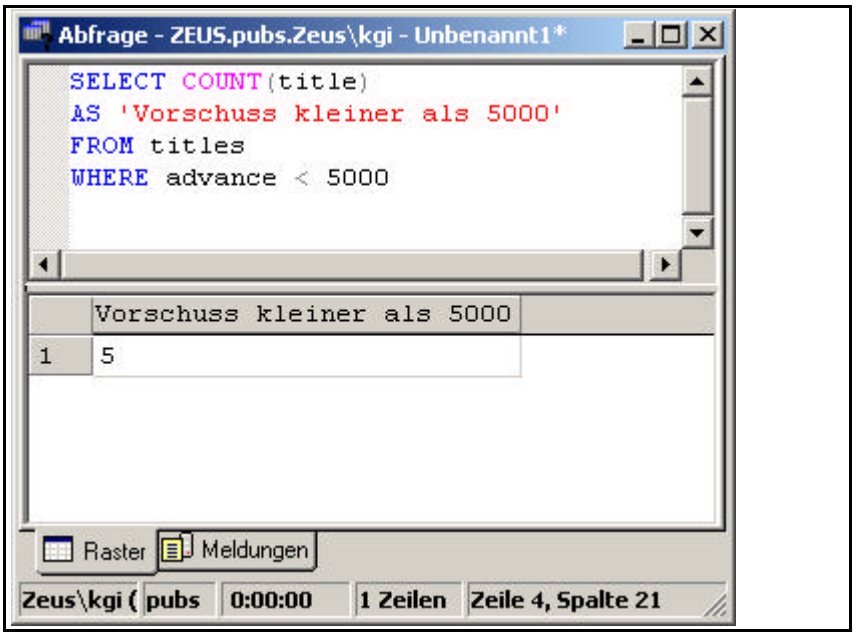


**Abbildung 8: Beispiel zur Funktion `COUNT()` III**

## COUNT(*)

Die Aggregatfunktion `COUNT(*)` gibt die Anzahl der Einträge in einer Datensatzgruppe einschließlich der `NULL`-Werte und Duplikate zurück.

Das erste einfache Beispiel zählt alle Zeilen der Tabelle *authors*.

```
SELECT COUNT(*)
FROM authors
```

Sie können die Aggregatfunktion `COUNT(*)` ohne weiteres auch mit anderen Aggregatfunktionen kombinieren. Das folgende Beispiel zeigt eine Kombination der Funktionen `COUNT(*)` und `AVG()`.

Die beiden Aggregatfunktionen berücksichtigen nur Daten aus den Zeilen, die die Bedingungen der `WHERE`-Klausel erfüllen.

**Die** GROUP BY-Klausel

Das folgende Beispiel berechnet den Durchschnittpreis aller Preise größer als 1.000 und zeigt die Anzahl der zurückgegebenen Datensätze an.

```
SELECT COUNT(*), AVG(price)
FROM titles
WHERE advance > 1000
```

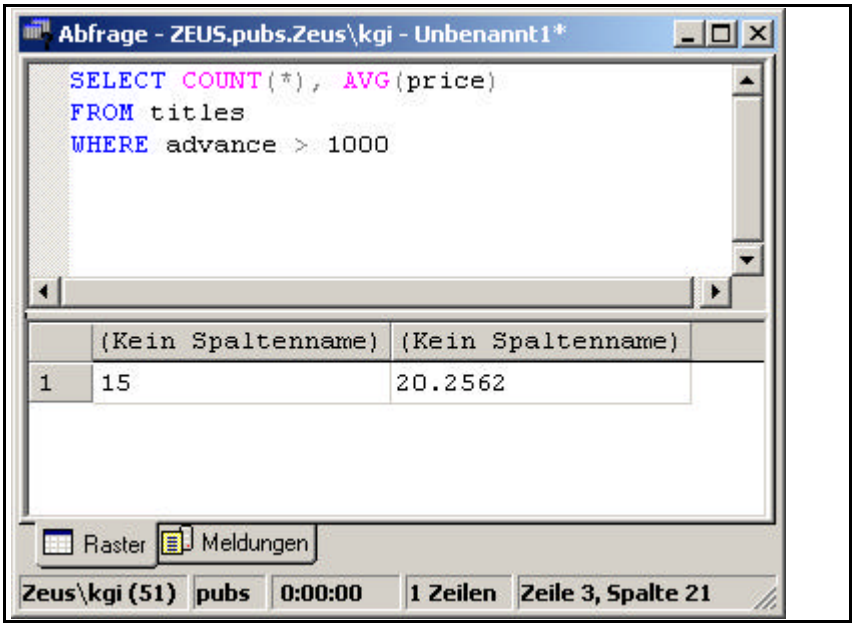


**Abbildung 9: Beispiel zur Funktion `COUNT(*)`**

# Die GROUP BY-Klausel

Die `GROUP BY`-Klausel erlaubt das Gruppieren von Abfrageergebnissen. Eine Gruppierung bedeutet hier nichts anderes als die Zusammenfassung nach bestimmten Kriterien.

Die allgemeine Syntax der `GROUP BY`-Klausel lautet:

```
SELECT column1,
SUM(column2)
FROM list-of-tables
GROUP BY column-list
```

Die `GROUP BY`-Klausel kann auf alle Elemente der `SELECT`-Liste angewandt werden.

Falls die `ORDER BY`-Klausel nicht angegeben wird, haben die von der `GROUP BY`-Klausel zurückgegebenen Gruppen keine feste Reihenfolge. Es ist daher sinnvoll, dass Sie immer auch die `ORDER BY`-Klausel zur Festlegung einer bestimmten Reihenfolge im Resultatset verwenden.

## Eindeutige Datensätze

Beginnen wir mit einem einfachen Beispiel, in dem die Tabelle *titles* nach der Buchkategorie *type* gruppiert wird.

```
SELECT type
FROM titles
GROUP BY type
```

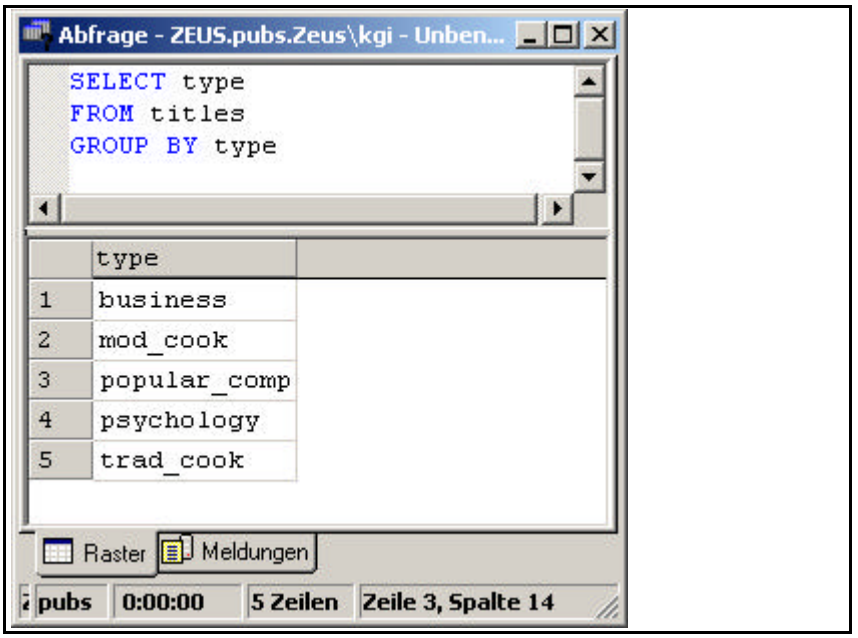


**Abbildung 10: Beispiel zur GROUP BY-Klausel I**

Das Resultatset gibt jede vorhandene Buchkategorie genau einmal zurück. Das gleiche Ergebnis hätten wir auch mit dem Schlüsselwort `DISTINCT` erzielt. Die `GROUP BY`-Klausel kann also zum Erzielen eindeutiger Datensätze im Resultatset eingesetzt werden.

## Berechnungen

Der eigentliche Sinn zur Verwendung der `GROUP BY`-Klausel liegt allerdings darin, für Datensätze, die zu einer Gruppe gehören, Berechnungen durchzuführen. Solche Berechnungen erfolgen mit Hilfe der Ihnen bereits bekannten Aggregatfunktionen.

Das folgende Beispiel ist eine Erweiterung des vorherigen und berechnet für die einzelnen Kategorien die Summe der Jahresverkäufe:

```
SELECT type, SUM(ytd_sales)
```

```
AS 'Summe der Verkäufe'
FROM titles
GROUP BY type
ORDER BY type
```

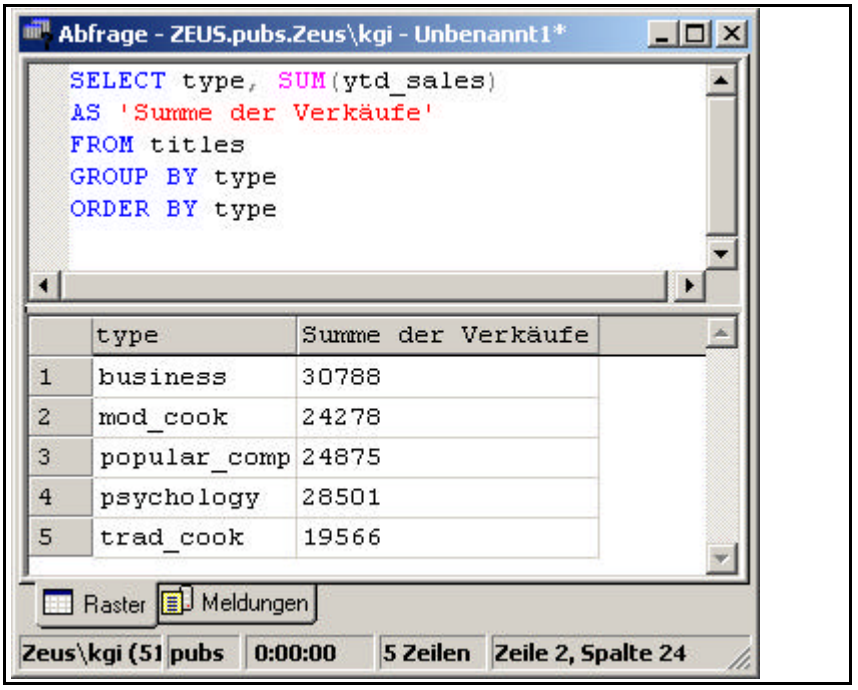


**Abbildung 11: Beispiel zur GROUP BY-Klausel II**

Erst damit wird die Verwendung der GROUP BY-Klausel auch richtig sinnvoll. Auf der gleichen Ebene liegt auch das nächste Beispiel. Es berechnet den Durchschnittspreis für die Titel jeder Buchkategorie, deren Vorschuss über 5.000 liegt.

```
SELECT type, AVG(price)
AS 'Durchschnittspreis'
FROM titles
WHERE advance > 5000
GROUP BY type
```

**Die** HAVING-Klausel

```
ORDER BY type
```

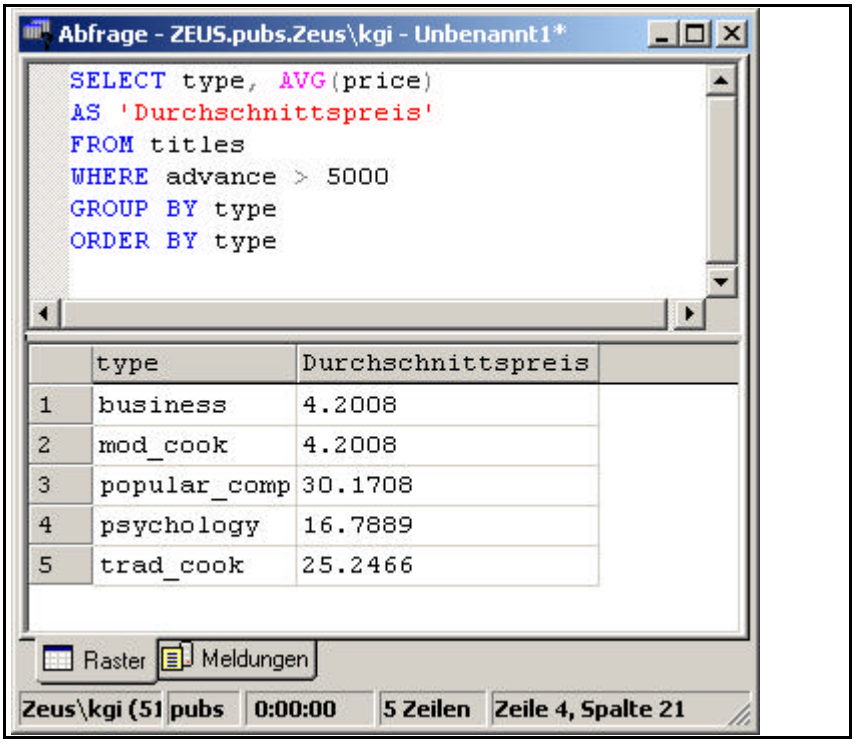


**Abbildung 12: Beispiel zur GROUP BY-Klausel III**

Halten Sie bitte bei der Anwendung der `WHERE`-Klausel fest, dass deren Bedingungen vor der Gruppierung abgearbeitet werden.

# Die HAVING-Klausel

Mit Hilfe der `HAVING`-Klausel können für eine `GROUP BY`-Klausel Bedingungen festgelegt werden. Die allgemeine Syntax der `HAVING`-Klausel lautet:

```
SELECT column1,
SUM(column2)
FROM list-of-tables
GROUP BY column-list
HAVING condition;
```

Die HAVING-Klausel kann auf alle Elemente der SELECT-Liste angewandt werden. Ihre Bedingungen werden dabei nach der Gruppierung angewandt.

Die Möglichkeiten in der HAVING-Klausel sind vielfältig und flexibel. Im Prinzip erfüllt die HAVING-Klausel in der GROUP BY-Klausel die gleiche Funktion wie die WHERE-Klausel in der FROM-Klausel.

Anders ausgedrückt: Die HAVING-Klausel ist die WHERE-Bedingung in einer GROUP BY-Klausel. Dazu verschiedene Beispiele.

## Einfache Berechnung

Das erste Beispiel sucht nach Verlegern, deren Gesamtsumme an Verkäufen über 30.000 liegt:

```
SELECT pub_id, SUM(ytd_sales)
AS 'Gesamtsumme'
FROM titles
GROUP BY pub_id
HAVING SUM(ytd_sales) > 30000
ORDER BY Gesamtsumme
```

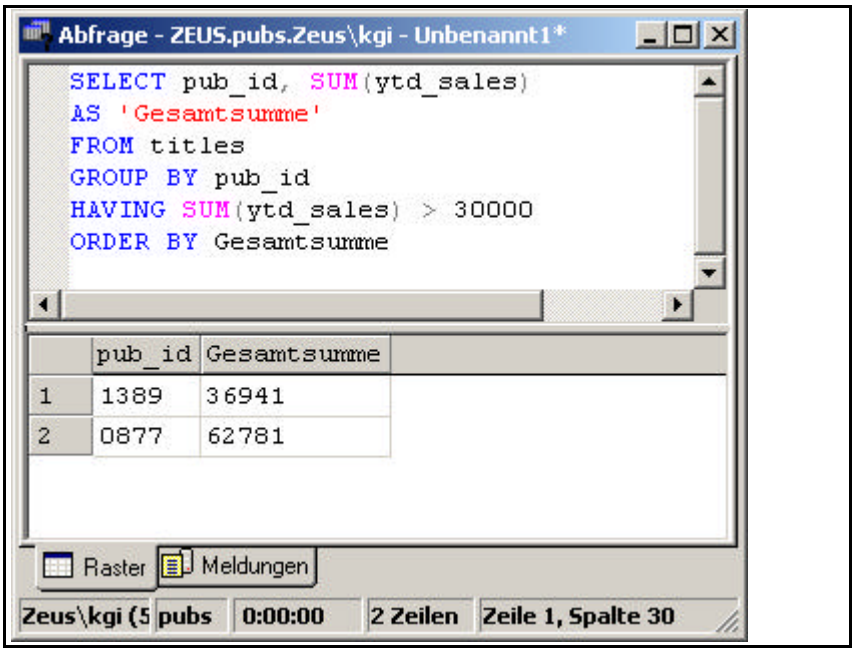


**Abbildung 13: Beispiel zur `HAVING`-Klausel I**

## Zählen von Datensätzen

Beim zweiten Beispiel werden mit dem Ausdruck `HAVING COUNT(*) > 3` die Verlage herausgefiltert, für die Gesamtzahlen von weniger als vier Büchern zurückgegeben werden.

```
SELECT pub_id, SUM(ytd_sales)
AS 'Gesamtsumme'
FROM titles
GROUP BY pub_id
HAVING COUNT(*) > 3
ORDER BY Gesamtsumme
```

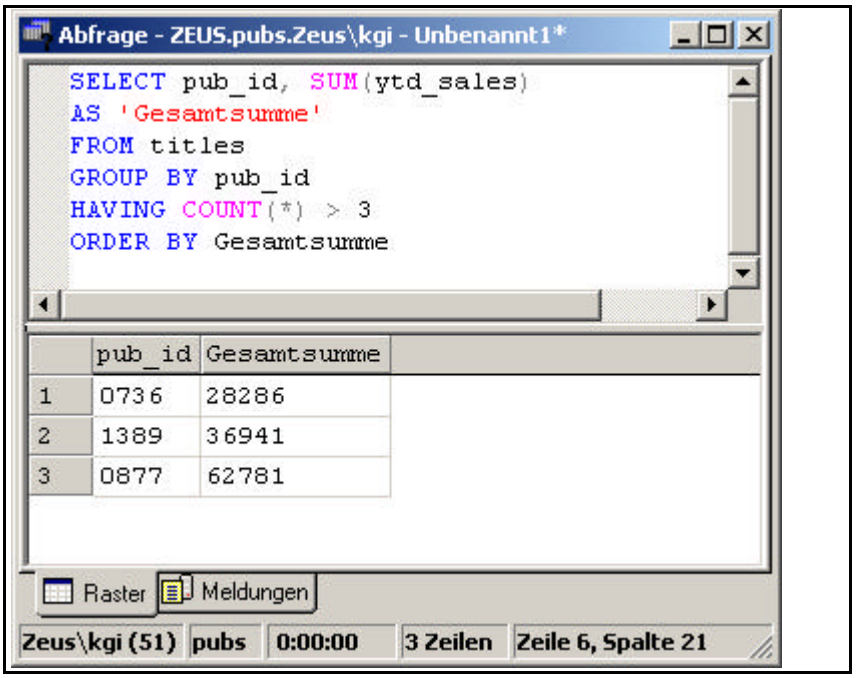


**Abbildung 14: Beispiel zur `HAVING`-Klausel II**

## Verknüpfte Bedingungen

In einer `HAVING`-Klausel können auch mehrere Bedingungen vorkommen. Diese werden dann mit `AND`, `OR` oder `NOT` verknüpft.

Das folgende Beispiel zeigt, wie sich die Tabelle *titles* nach Verlagen gruppieren lässt, wobei nur die Verlage berücksichtigt werden, deren *pub_id* größer als *0800* ist, die insgesamt mehr als 5.000 $ an Vorschüssen gezahlt haben und die Bücher durchschnittlich für unter 30 $ verkaufen:

```
SELECT pub_id, SUM(advance) AS 'Summe Vorschuss',
AVG(price) AS 'Durchschnittspreis'
FROM titles
WHERE pub_id > '0800'
```

```
GROUP BY pub_id
HAVING SUM(advance) > 5000
AND AVG(price) < 30
ORDER BY pub_id
```

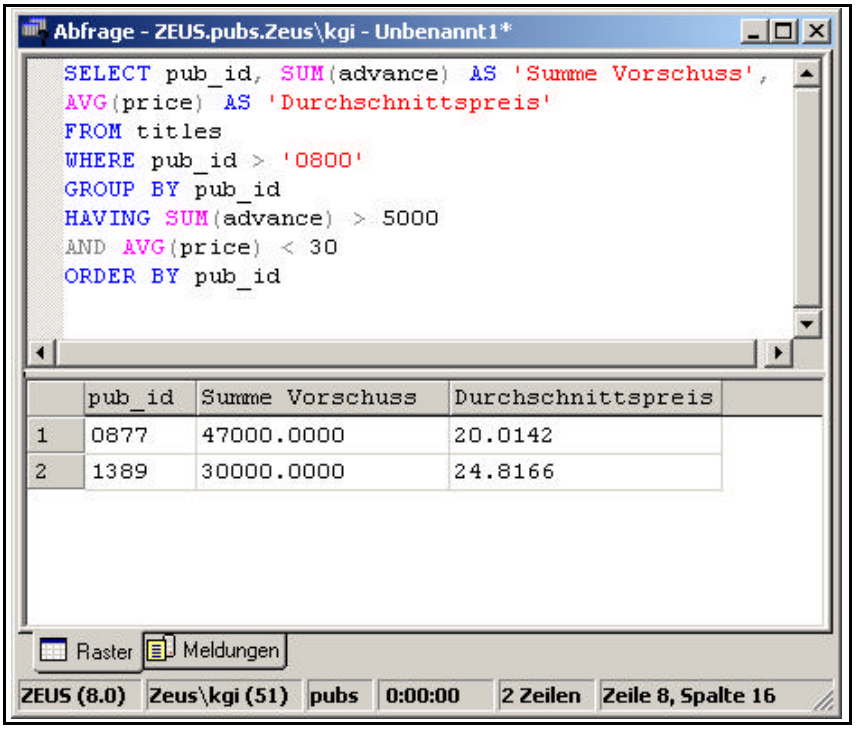


**Abbildung 15: Beispiel zur `HAVING`-Klausel III**

# Verknüpfungen

Dieser Abschnitt behandelt die zum Abfragen von Daten aus zwei oder mehreren Tabellen notwendigen Verknüpfungen. Dabei lernen Sie folgende Verknüpfungstypen kennen:

- Innere Verknüpfung
- Äußere Verknüpfung
- Kreuzverknüpfung
- Selbstverknüpfung

## Überblick

Bei einer Tabellenverknüpfung vergleichen Sie eine oder mehrere Spalten in einer Tabelle mit einer oder mehreren Spalten in einer anderen Tabelle. Das Ergebnis dieses Vergleichs erzeugt dann neue Zeilen, indem die in der `SELECT`-Spaltenliste genannten Spalten aus den verknüpften Tabellen entsprechend den Verknüpfungsbedingungen kombiniert werden.

Das Resultatset dieses Vergleichs gibt neue Zeilen zurück, indem die in der `SELECT`-Liste aufgeführten Spalten aus den verknüpften Tabellen entsprechend den Verknüpfungsbedingungen miteinander kombiniert werden.

Bei Verknüpfungen verwenden Sie in der `SELECT`-Anweisung das Schlüsselwort `JOIN`. Die allgemeine SQL-Syntax dafür lautet:

```
SELECT Tabellenname.Spaltenname, [...]
FROM { Tabellenname [Verknüpfungstyp]
JOIN Tabellenname
ON Suchkriterien}, [...]
WHERE Suchkriterien
```

Sie erkennen an der Syntax, dass die Verknüpfungsanweisungen in der `FROM`-Klausel der `SELECT`-Anweisung aufgeführt werden. Wir beginnen mit der Inneren Verknüpfung.

## Innere Verknüpfung

Die *Innere Verknüpfung* oder `Inner Join` verbindet zwei Tabellen auf der Grundlage einer Verknüpfungsbedingung miteinander. Das Resultatset ist eine neue Tabelle mit den Zeilen, die den Verknüpfungsbedingung entsprechen.

`Inner Joins` geben immer dann Resultatsets zurück, wenn die gesuchten Informationen in beiden Tabellen gefunden worden sind. Die häufigsten Typen innerer Verknüpfungen sind Gleichheits- und natürliche Verknüpfungen.

### Gleichheitsverknüpfung

Bei einer Gleichheitsverknüpfung werden die Spaltenwerte auf Gleichheit hin untersucht und alle übereinstimmenden Spalten im Resultset angezeigt.

Das folgende Beispiel gibt als Resultatset alle Spalten in beiden Tabellen und nur die Zeilen zurück, die einen übereinstimmenden Wert in der Verknüpfungsspalte enthalten.

```
SELECT * FROM authors AS t1
INNER JOIN publishers AS t2
ON t1.city = t2.city
ORDER BY t1.au_lname DESC
```

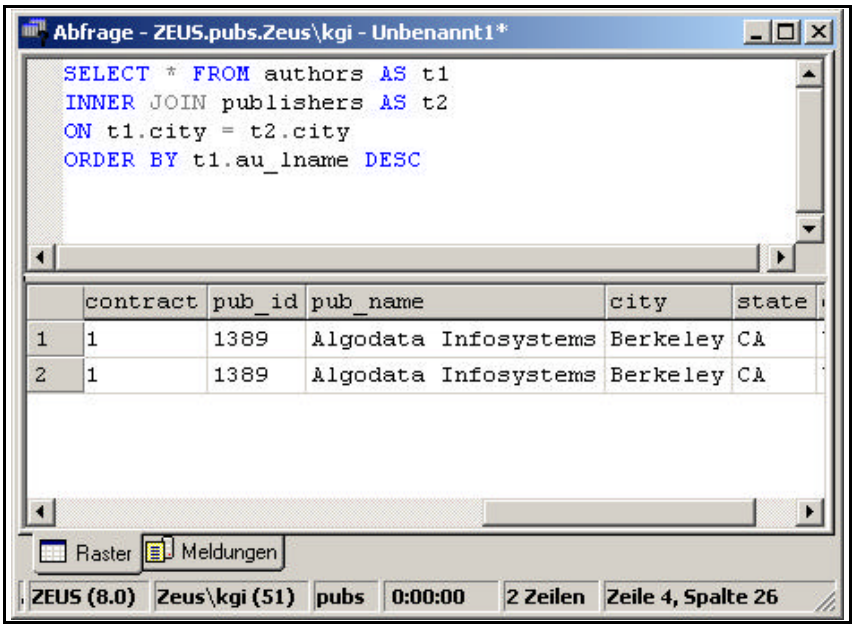


**Abbildung 16: Beispiel zur Gleichheitsverknüpfung**

### Natürliche Verknüpfung

Bei einer *natürlichen Verknüpfung* werden die Spaltenwerte auf Gleichheit hin untersucht Die übereinstimmenden Spalten werden dagegen aber nur einmal im Resultatset angezeigt.

Beim folgenden Beispiel wird die Spalte *publisher.city* nur ein einziges Mal vom Resultatset zurückgegeben.

```
USE pubs
SELECT p.pub_id, p.pub_name, p.state, a.*
FROM publishers p INNER JOIN authors a
ON p.city = a.city
ORDER BY a.au_lname ASC, a.au_fname ASC
```

**Verknüpfungen**

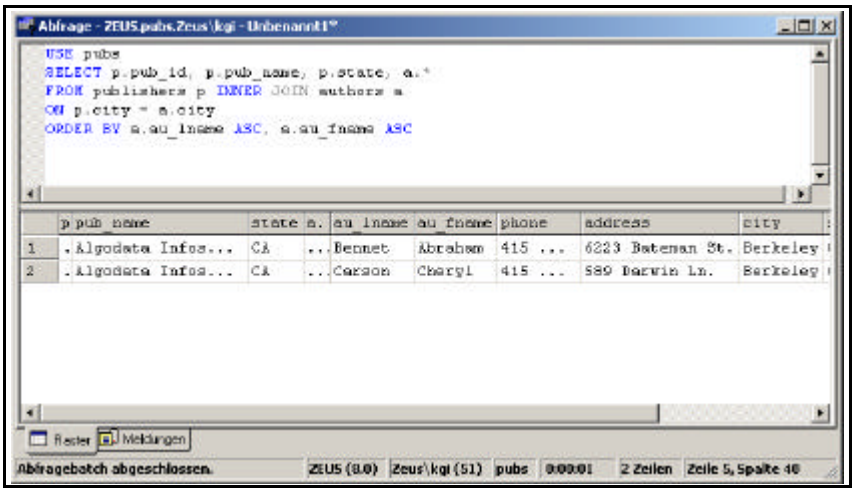


**Abbildung 17: Beispiel zur natürlichen Verknüpfung**

## Äußere Verknüpfungen

*Äußere Verknüpfungen* dienen zum Entfernen von Zeilen aus einer Tabelle, während alle Zeilen einer anderen Tabelle als Resultset zurückgegeben werden. Es gibt drei Typen von äußeren Verknüpfungen:

- *Linke äußere Verknüpfung* oder `Left Outer Join`

Das Resultatset gibt sämtliche Zeilen der linken Tabelle, die die angegebene Verknüpfungsbedingung nicht erfüllen, zurück. Die Ausgabespalten der rechten Tabelle werden auf `NULL` gesetzt.

Mit *links* bzw. *rechts* sind die beiden Seiten der Verknüpfung gemeint. Sie können sich das durchaus bildlich vorstellen, wenn Sie sich den Code der SQL-Anweisung anschauen.

- *Rechte äußere Verknüpfung* oder `Right Outer Join`

Das Resultatset gibt sämtliche Zeilen aus der *rechten* Tabelle, die die angegebene Verknüpfungsbedingung nicht erfüllen, zurück. Die Ausgabespalten der linken Tabelle werden auf `NULL` gesetzt.

- *Vollständige äußere Verknüpfung* oder `Full Outer Join`

Das Resultatset gibt eine Zeile aus der linken oder der rechten Tabelle, die die Verknüpfungsbedingung nicht erfüllt, zurück Die Ausgabespalten der anderen Tabelle werden in diesem Fall auf `NULL` festgelegt.

### Linke äußere Verknüpfung

Bei den folgenden drei Beispielen sind die Tabellen *authors* und *publishers* über die Spalte *city* miteinander verknüpft.

Das Resultatset des ersten Beispiels gibt alle Autoren zurück, unabhängig davon, ob sich in der gleichen Stadt auch ein Verlag befindet.

```
SELECT t1.au_fname, t1.au_lname, t2.pub_name
FROM authors t1
LEFT OUTER JOIN publishers t2
ON t1.city = t2.city
ORDER BY t2.pub_name ASC, t1.au_lname ASC, t1.au_fname
ASC
```

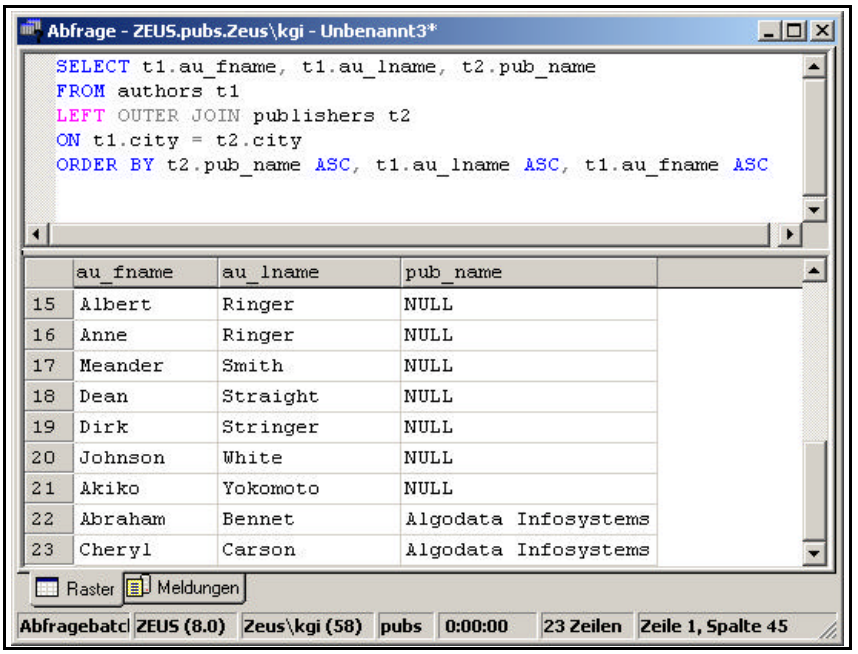


**Abbildung 18: Beispiel zur linken äußeren Verknüpfung**

### Rechte äußere Verknüpfung

Beim zweiten Beispiel gibt das Resultatset alle Verlage zurück, unabhängig davon, ob in der gleichen Stadt auch ein Autor wohnt.

```
SELECT t1.au_fname, t1.au_lname, t2.pub_name
FROM authors AS t1
RIGHT OUTER JOIN publishers AS t2
ON t1.city = t2.city
ORDER BY t2.pub_name ASC, t1.au_lname ASC, t1.au_fname
ASC
```

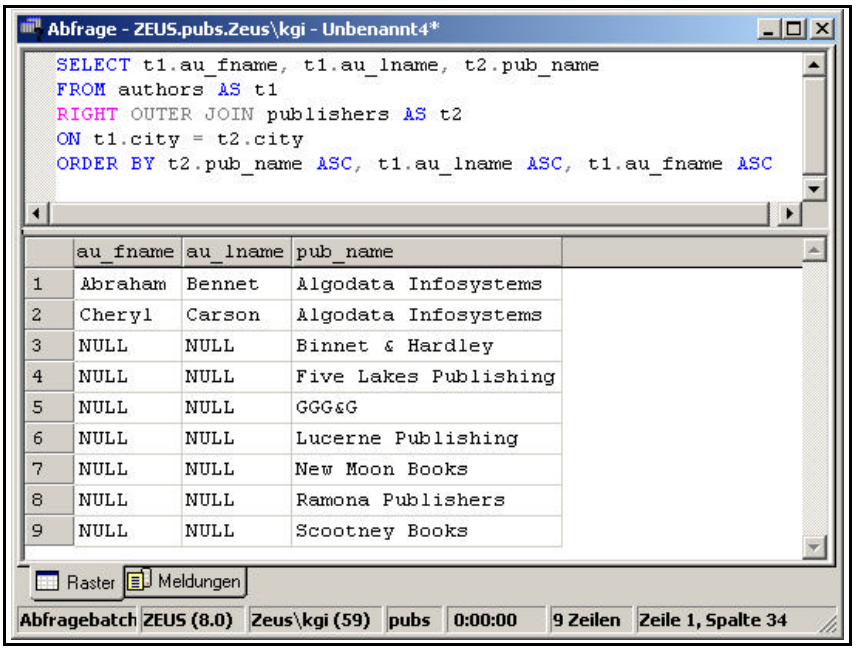


**Abbildung 19: Beispiel zur rechten äußeren Verknüpfung**

### Vollständige äußere Verknüpfung

Beim dritten Beispiel gibt das Resultatset alle Zeilen aus beiden Tabellen zurück, unabhängig davon, ob übereinstimmende Daten in den Verknüpfungsfeldern beider Tabellen enthalten sind.

```
SELECT t1.au_fname, t1.au_lname, t2.pub_name
FROM authors t1
FULL OUTER JOIN publishers t2
ON t1.city = t2.city
ORDER BY t2.pub_name ASC, t1.au_lname ASC, t1.au_fname
ASC
```

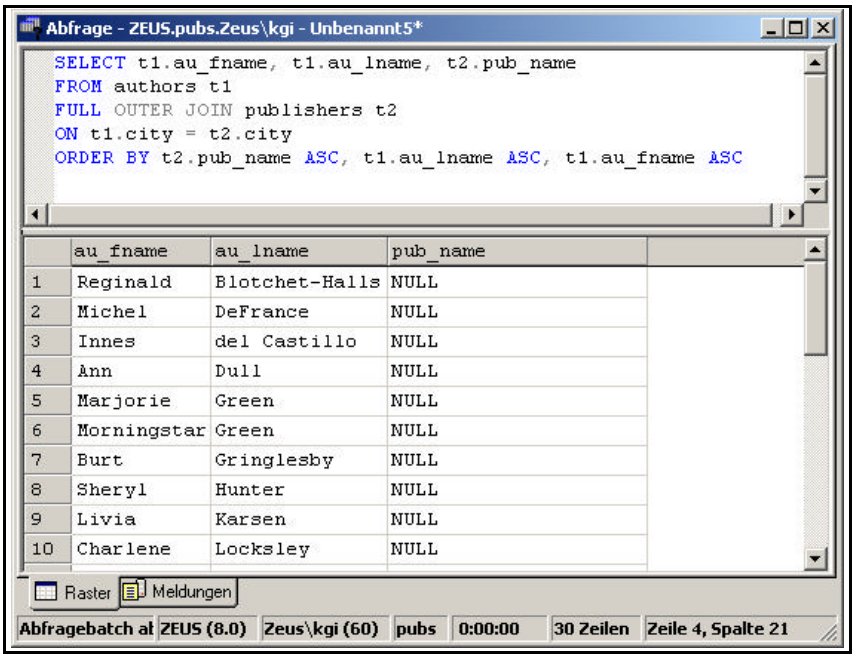


**Abbildung 20: Beispiel zur vollständigen äußeren Verknüpfung**

## Kreuzverknüpfung

Eine *Kreuz-* oder *unbeschränkte Verknüpfung* liefert als Resultatset eine Kombination sämtlicher Zeilen aller Tabellen in der Verknüpfung zurück.

Eine solche Kombination aller Zeilen aus allen in die Verknüpfung eingebundenen Tabellen wird auch *Kartesisches Produkt* bezeichnet.

Das folgende Beispiel gibt als Resultatset sämtliche Zeilen der Tabellen *authors* und *publishers* zurück

```
SELECT au_fname, au_lname, pub_name
FROM authors
CROSS JOIN publishers
ORDER BY au_lname
```

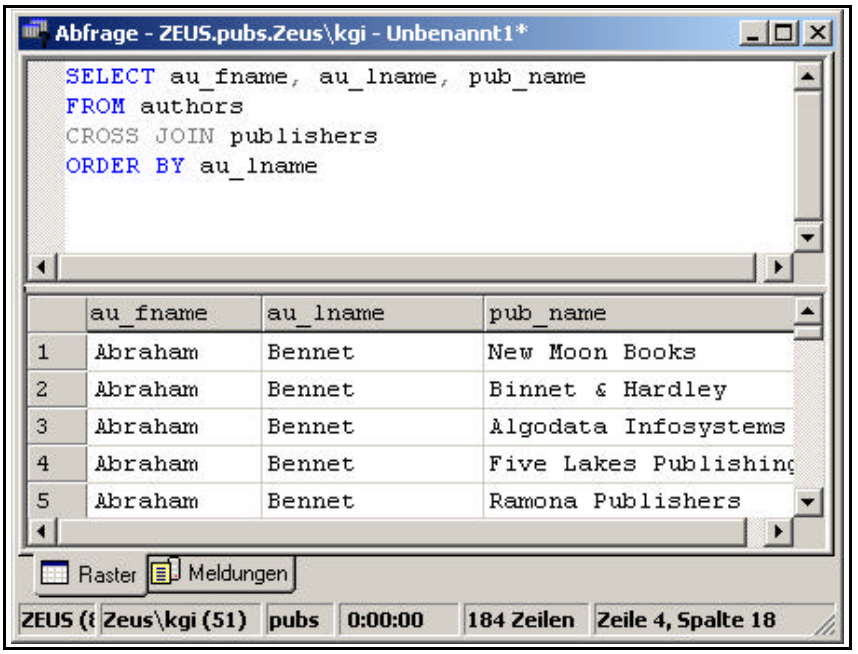


**Abbildung 21: Beispiel zur Kreuzverknüpfung**

## Selbstverknüpfung

Zum Erzielen hierarchischer Strukturen ist es zuweilen notwendig, eine Tabelle auf sich selbst zu verknüpfen.

Das folgende Beispiel gibt als Resultatset alle Autoren zurück, die in derselben Stadt mit derselben Postleitzahl leben.

Bei einer *Selbstverknüpfung* auf eine Tabelle müssen Sie natürlich einen Alias für den Tabellennamen erstellen, damit eine Tabelle logisch wie zwei Tabellen behandelt werden kann.

```
SELECT au1.au_fname, au1.au_lname, au2.au_fname,
au2.au_lname,au1.city, au1.zip

FROM authors au1

INNER JOIN authors au2
```

```
ON au1.city = au2.city AND au1.zip = au2.zip
WHERE au1.au_id < au2.au_id
ORDER BY au1.city, au1.zip
```

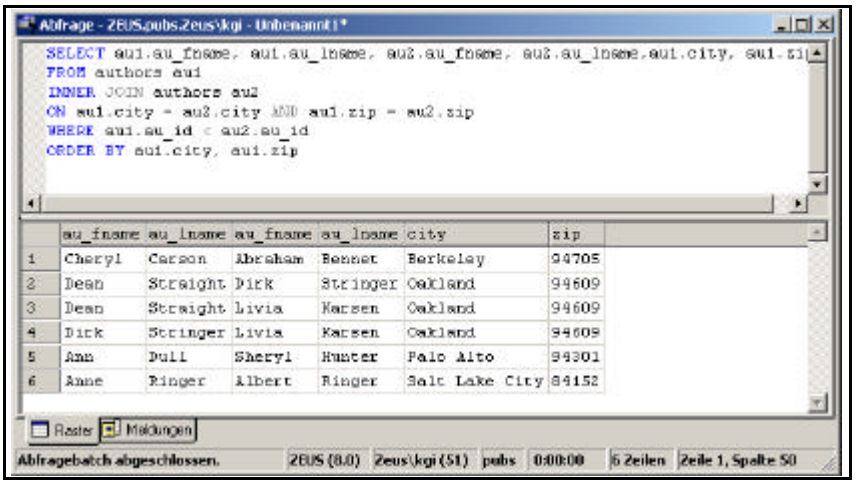


**Abbildung 22: Beispiel zur Selbstverknüpfung**

# Unterabfragen

*Unterabfragen* sind SELECT-Anweisungen innerhalb von SELECT-Anweisungen und werden deshalb auch als verschachtelte SELECT-Anweisungen bezeichnet.

Unterabfragen können oftmals zu denselben Ergebnissen wie eine Verknüpfungsoperation führen. Im folgenden Abschnitt lernen Sie Möglichkeiten und Grenzen von Unterabfragen näher kennen.

Die allgemeine Syntax einer Unterabfrage lautet:

```
SELECT *
FROM Tabelle1
```

```
WHERE Tabelle1.Spalte1 =
(SELECT Spalte2
FROM Tabelle2
WHERE Spalte2 = Wert)
```

Die Unterabfrage ist immer in Klammern eingeschlossen und wird immer vor der Verarbeitung der äußeren Abfrage beendet. Eine Unterabfrage kann eine weitere Unterabfrage enthalten, die dann wiederum eine Unterabfrage enthalten kann usw. Die Verschachtelungstiefe von Unterabfragen hängt nur von der Entwicklungsumgebung bzw. den zur Verfügung stehenden Systemressourcen ab.

## Gebrauch von Unterabfragen

Eine `SELECT`-Anweisung, die als Unterabfrage fungiert, kann in folgenden anderen SQL-Anweisungen verwendet werden:

- `SELECT`-Anweisung
- `INSERT`-Anweisung
- `UPDATE`-Anweisung
- `DELETE`-Anweisung

Falls eine Unterabfrage nur einen einzelnen Wert – beispielsweise ein Aggregat – zurückliefert, kann sie überall dort benutzt werden, wo auch der einzelne Wert benutzt werden kann. Liefert die Unterabfrage dagegen eine Ergebnisliste – wie etwa eine einzelne Spalte aus mehreren Werten – zurück, muss diese Unterabfrage in der `WHERE`-Klausel verwendet werden.

In vielen Fällen kann anstelle einer Unterabfrage auch eine Tabellenverknüpfung eingesetzt werden.

## Typen von Unterabfragen

SQL kennt verschiedene Arten von Unterabfragen. Deswegen können Unterabfragen auch an vielen Stellen verwendet bzw. zu verschiedenen Zwecken benutzt werden:

- Unterabfragen mit Aliasnamen
- Unterabfragen in Verbindung mit den Schlüsselwörtern `IN` bzw. `NOT IN`
- Unterabfragen in `UPDATE`-, `INSERT`- und `DELETE`-Anweisungen
- Unterabfragen mit Vergleichsoperatoren
- Unterabfragen in Verbindung mit den Schlüsselwörtern `EXISTS` bzw. `NOT EXISTS`
- Unterabfragen anstelle von Ausdrücken

In den folgenden Abschnitten lernen Sie dazu verschiedene Beispiele kennen.

## Einfache Unterabfragen

In den ersten beiden Beispielen wird die Unterabfrage mit dem einfachen Vergleichsoperator = eingeleitet.

Das erste Beispiel verwendet Tabellen aus der Datenbank *northwind* und gibt als Resultatset alle Datensätze zurück, deren Preis mit dem Preis des Artikels *Chai* identisch ist:

```
SELECT Products.ProductName, Products.UnitPrice
FROM Products
WHERE Products.UnitPrice =
(SELECT UnitPrice
FROM Products
WHERE ProductName = 'Chai')
```

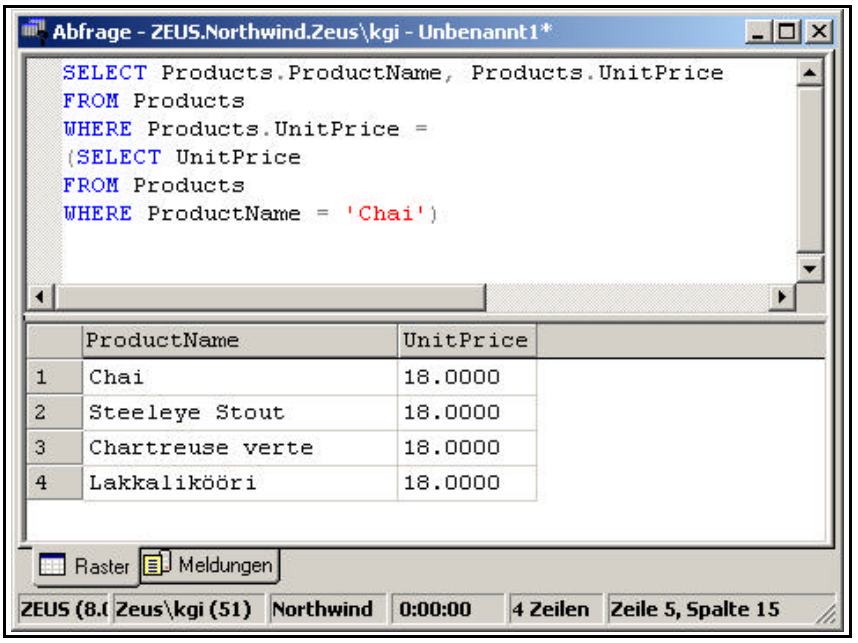


**Abbildung 23: Beispiel zu Unterabfragen I**

Die SELECT-Anweisung der Unterabfrage gibt ohne weiteres auch allein ein sinnvolles Resultatset zurück:

```
SELECT UnitPrice
FROM Products
WHERE ProductName = 'Chai')
```

Das zweite Beispiel findet die Namen aller Autoren, die in der Stadt wohnen, in der auch der Verleger *Algodata Infosystems* ansässig ist:

```
SELECT au_lname, au_fname
FROM authors
WHERE city =
(SELECT city
FROM publishers
WHERE pub_name = 'Algodata Infosystems')
ORDER By au_lname
```

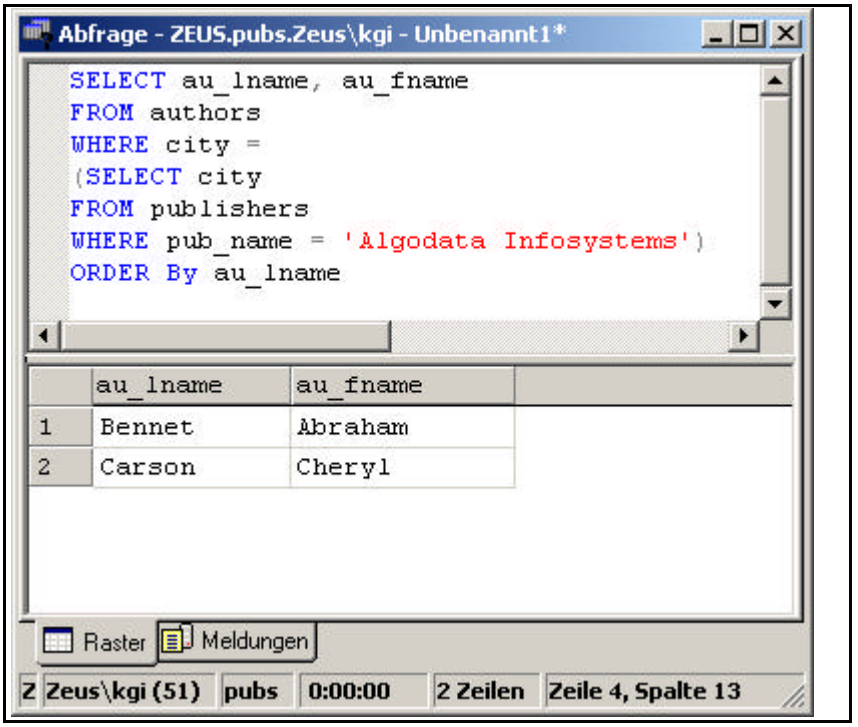


**Abbildung 24: Beispiel zu Unterabfragen II**

## Unterabfragen mit Aggregatfunktionen

Sie wissen bereits, dass die Aggregatfunktionen `AVG`, `MIN`, `MAX`, `SUM` und `COUNT` jeweils einen einzelnen Wert zurückgeben.

Das folgende Beispiel ermittelt beispielsweise die Namen aller Bücher, deren Preis über dem derzeit niedrigsten Preis liegt.

```
SELECT DISTINCT title
FROM titles
WHERE price >
(SELECT MIN(price)
```

```
FROM titles)
ORDER BY title
```

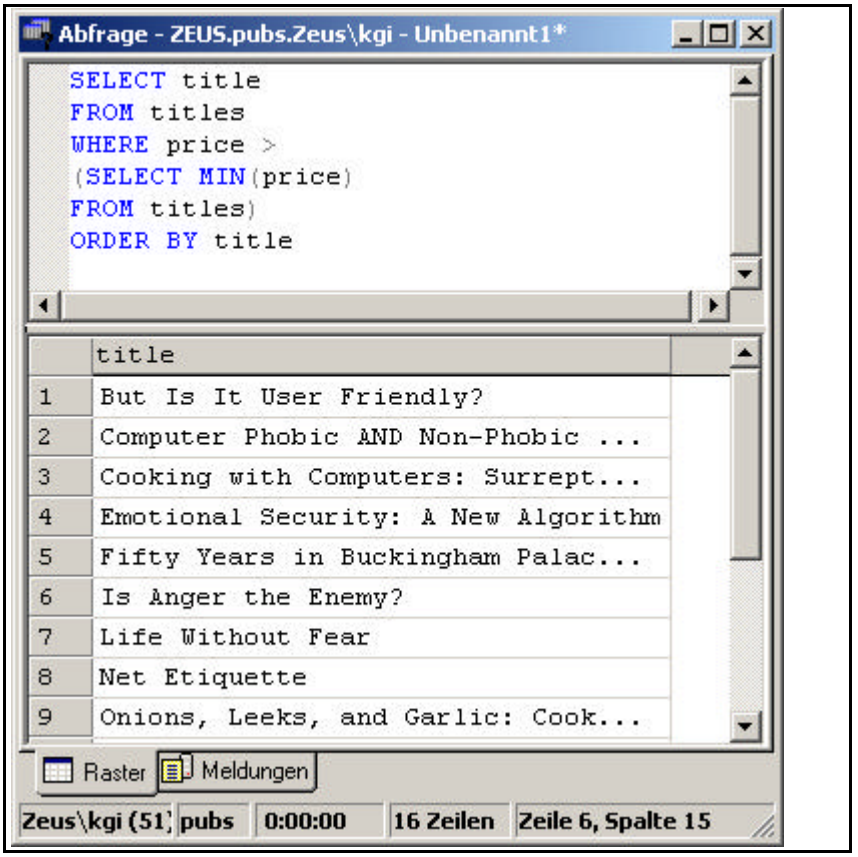


**Abbildung 25: Beispiel zu Unterabfragen III**

Das folgende Beispiel enthält zwei verschachtelte Unterabfragen, die jeweils die Aggregatfunktion `AVG` verwenden. Bei einer verschachtelten Unterabfrage ist in einer Unterabfrage eine oder mehrere Unterabfrage(n) eingebettet.

Das Beispiel ermittelt den Preis von Computerbüchern, den Durchschnittspreis aller Bücher und die Differenz der beiden Preise.

```
SELECT title, price,
```

```
(SELECT AVG(price)
FROM titles) AS Durchschnitt,
price-(SELECT AVG(price) FROM titles) AS Differenz
FROM titles
WHERE type='popular_comp'
ORDER BY title
```

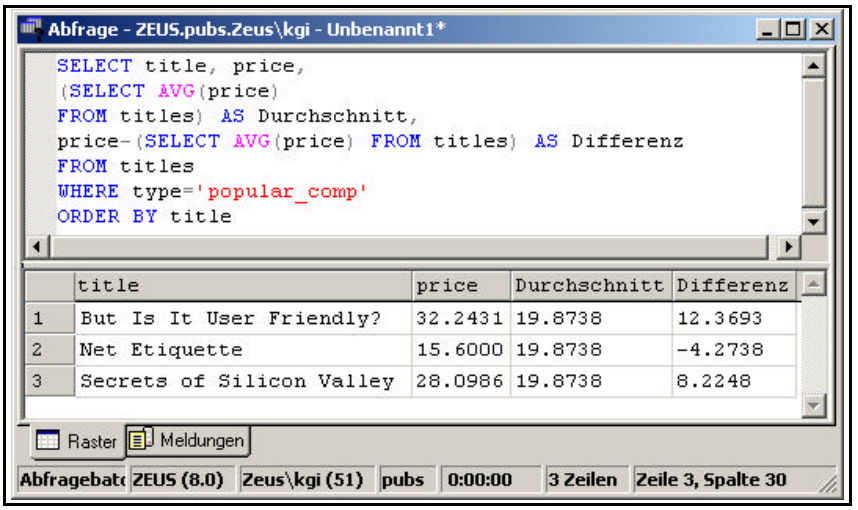


**Abbildung 26: Beispiel zu Unterabfragen IV**

Gehen Sie bei der Erstellung komplexerer Unterabfragen in folgender Reihenfolge vor: Erstellen Sie die Abfragen zunächst ganz normal als Auswahlabfragen und testen Sie diese. Fügen Sie nach erfolgreichem Test diese Abfragen dann als Unterabfrage ein.

## Laufender Zähler in Abfragen

Mit Hilfe einer Unterabfrage können Sie auch einen laufenden Zähler der Datensätze des Resultatsets erstellen.

```
SELECT
(Select Count (*)
```

```
FROM Orders AS MyTempTable
WHERE MyTempTable.OrderID < Orders.OrderID)+1
AS 'Laufende Nummer', Orders.CustomerID
FROM Orders
order by 'Laufende Nummer'
```

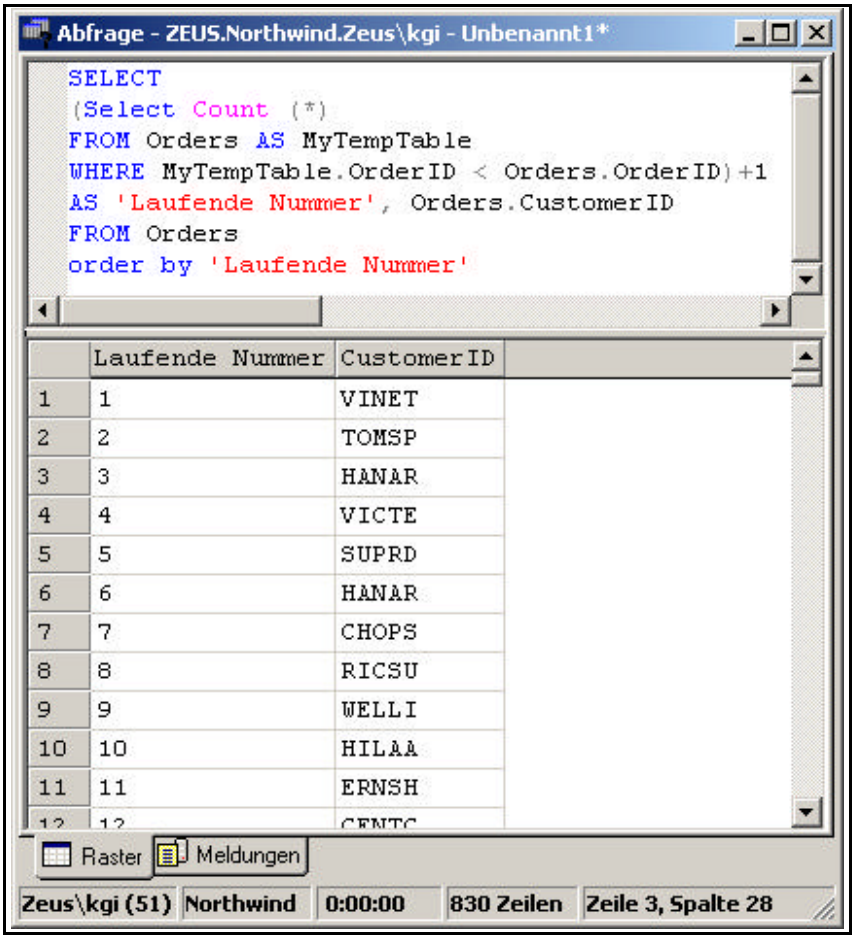


**Abbildung 27: Laufender Zähler im Resultatset einer Abfrage**

Die berechnete Spalte können Sie dann beliebig weiterverwenden.

**Unterabfragen**

Die Unterabfrage arbeitet bei diesem Beispiel mit Hilfe der temporären Tabelle *MyTempTable*.

## Unterabfragen mit IN und EXISTS

Unterabfragen können auch mit den Schlüsselwörtern `IN` und `EXISTS` verwendet werden. Die Einzelheiten dazu lernen Sie anhand von Beispielen in dem folgenden Abschnitt kennen.

### IN bzw. NOT IN

Das Resultatset einer Unterabfrage mit dem Schlüsselwort `IN` bzw. `NOT IN` gibt eine Liste von Werten aus. Diese Werte können dann von der äußeren Abfrage verwendet werden.

Das folgende Beispiel gibt alle Autoren mit Wohnsitz in dem Bundesstaat zurück, in dem sich auch die Buchhandlung befindet, die die Bücher ihrer jeweiligen Verleger verkauft:

```
SELECT DISTINCT au_fname, au_lname, state
FROM authors
WHERE state
IN
(SELECT state
FROM stores)
```

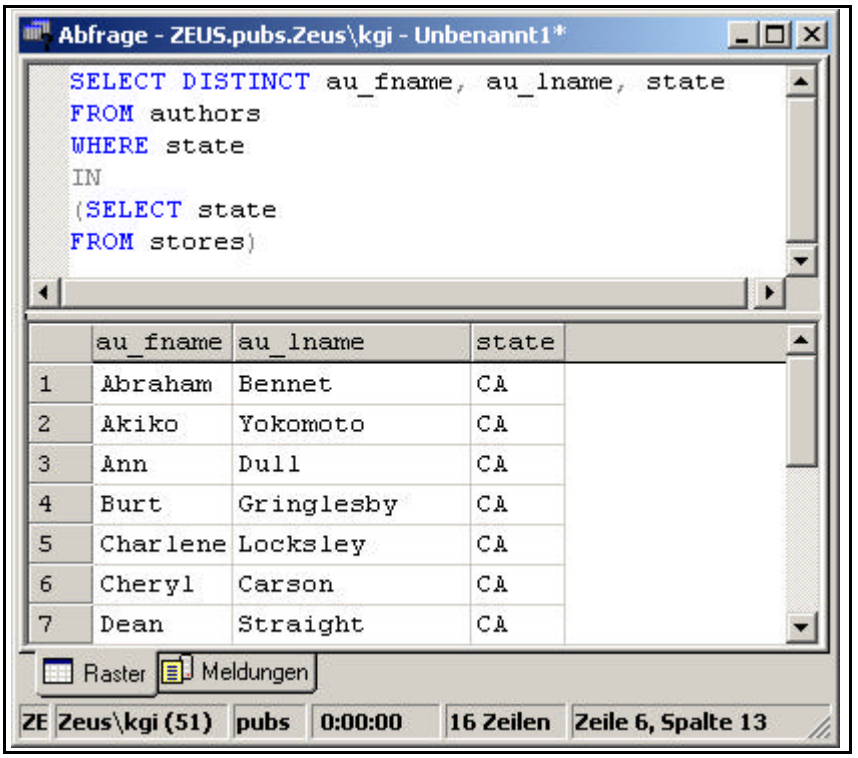


**Abbildung 28: Unterabfrage mit dem Schlüsselwort `IN`**

In der Spaltenliste darf bei der Verwendung von `IN` nicht mehr als eine Spalte angegeben sein.

### EXISTS bzw. NOT EXISTS

Die Einleitung einer Unterabfrage mit dem Schlüsselwort `EXISTS` bedeutet, dass diese Unterabfrage zur Überprüfung auf das Vorhandensein bestimmter Daten dient.

Die `WHERE`-Klausel der äußeren Abfrage überprüft, ob die Unterabfrage ein Resultatset zurückgibt. Die Unterabfrage gibt dabei keine tatsächlichen Daten, sondern lediglich den Wert `TRUE` oder `FALSE` zurück.

**Unterabfragen**

Das folgende Beispiel ermittelt die Namen aller Verlage, die Bücher aus dem Bereich *Wirtschaft* veröffentlichen:

```
SELECT pub_name
FROM publishers
WHERE EXISTS
(SELECT *
FROM titles
WHERE pub_id = publishers.pub_id AND type = 'business')
```

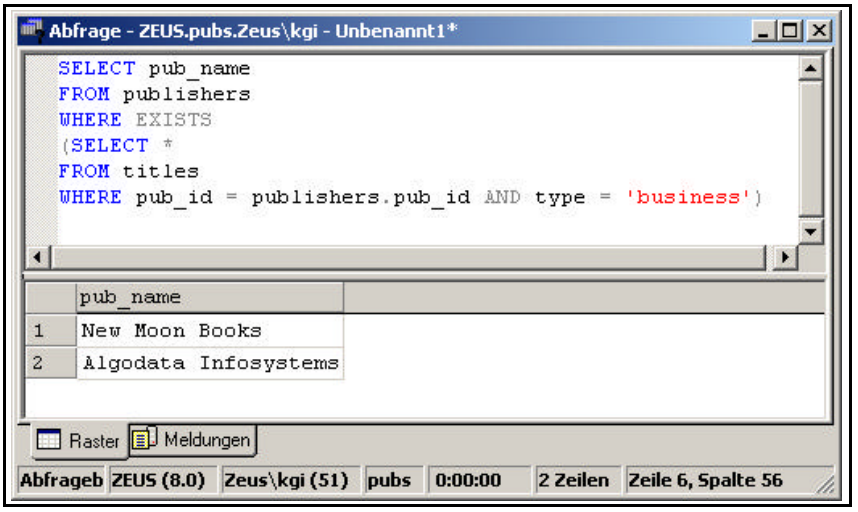


**Abbildung 29: Unterabfrage mit dem Schlüsselwort EXISTS**

Bei Verwendung von `EXISTS` muss die Unterabfrage die Anweisung `SELECT *` enthalten.

## Einschränkungen bei der Verwendung von Unterabfragen

Bei der Verwendung von Unterabfragen sind bestimmte Einschränkungen zu beachten. Dazu zählen im Einzelnen:

- Eine Unterabfrage muss immer in Klammern stehen.
- Es darf nicht mehr als eine Spalte in der Spaltenliste angegeben sein, wenn sie in der `IN`-Klausel verwendet wird.
- Eine Unterabfrage muss einen Einzelwert zurückliefern, wenn man sie in einem Einzelwertausdruck verwendet.
- Die `ORDER BY`- oder `SELECT INTO`-Klauseln können nicht in einer Unterabfrage verwendet werden.

## UNION-Abfragen

Eine `UNION`-Abfrage kombiniert Felder aus zwei oder mehr Tabellen oder Abfragen in einem Feld in den Abfrageergebnissen. Sie können also eine `UNION`-Abfrage verwenden, um Daten aus zwei Tabellen zu kombinieren.

Für die Kombination der Resultsets von zwei Abfragen mit `UNION` gelten die folgenden beiden Grundregeln:

- Die Anzahl und die Reihenfolge der Spalten muss für alle Abfragen identisch sein.
- Die Datentypen der beiden Abfragen müssen kompatibel sein.

Das folgende Beispiel fasst bestimmte Spalten der Tabellen *Customers* und *Suppliers* zusammen. Entsprechend der Herkunftstabelle werden die einzelnen Datensätze als *Kunden* bzw. *Lieferanten* gekennzeichnet.

```
SELECT City, CompanyName, ContactName, ContactTitle,
'Kunden' AS Beziehung

FROM Customers

UNION SELECT City, CompanyName, ContactName,
ContactTitle,  'Lieferanten'

FROM Suppliers

ORDER BY City DESC , CompanyName
```

**UNION-Abfragen**

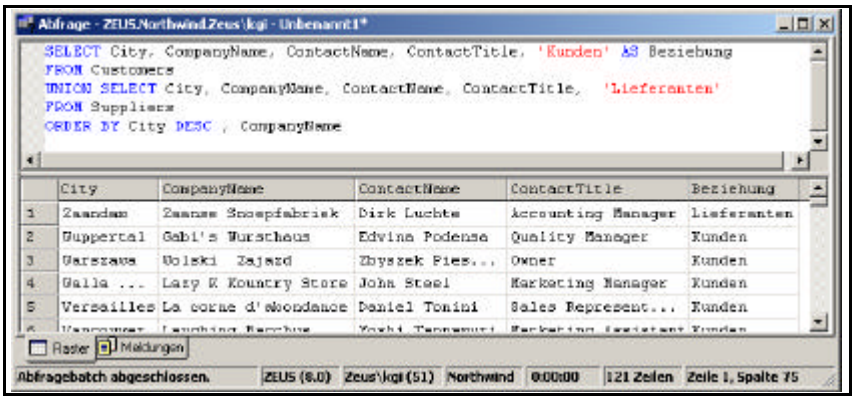


**Abbildung 30: `UNION`-Abfrage**