Mit unseren Übungen zu *ACCESS* können Sie Aufbau und Struktur einer *relationalen Datenbank* kennenlernen. Wir zeigen Ihnen wie Sie *Tabellen, Formulare* und *Berichte* erstellen und wie Sie *Abfragen* gestalten um entsprechende Auswertungen zu erhalten. Sie lernen wie Sie Daten nach bestimmten *Kriterien* filtern und wie Sie mit Hilfe von *Aktionsabfragen Datensätze* aktualisieren, anfügen und löschen.

Alle *Abfragen* wurden dabei nach dem gleichen Schema erstellt: In der *Entwurfsansicht* wurden die für die *Abfrage* relevanten *Tabellen* (oder *Abfragen*) in den *Abfrageentwurfsbereich* eingebunden und anschließend die erforderlichen *Felder* in den *Bearbeitungsbereich* übernommen um die Auswertung durchzuführen (*Abb. 1*). Das Ergebnis wurde anschließend in der *Datenblattansicht* ausgegeben (*Abb. 3*).

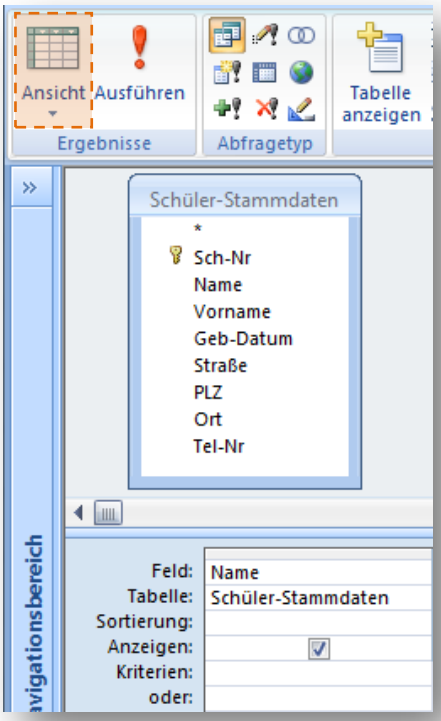


Abb. 1

Eine völlig andere Form *Abfragen* zu erstellen ist, sie mit Hilfe der *Datenbankabfragesprache SQL* zu gestalten.

Eine in *SQL* erstellte *Abfrage* kann direkt in der *SQL-Ansicht* formuliert werden und wird beim Wechseln in die *Entwurfsansicht* - wie in *Abb. 1* dargestellt - ausgegeben.

Andererseits wird eine in der *Entwurfsansicht* erstellte *Abfrage* (*Abb. 1*) immer auch in der *SQL-Ansicht* als *SQL-Anweisung* (*Abb. 2*) ausgegeben und wie folgt dargestellt:

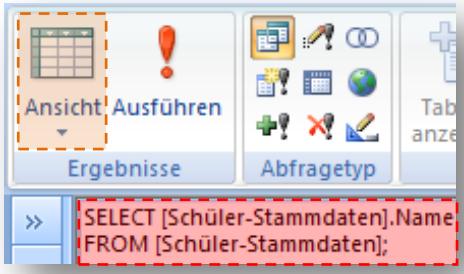


Abb. 2

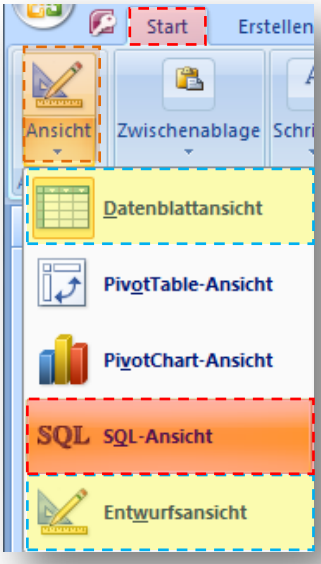


Abb. 3

Erläuterung:

Eine als *SQL-Anweisung* erstellte *Abfrage* beginnt immer mit: *SELECT*. Danach folgen eine Auflistung der in die *Abfrage* eingebundenen *Felder* und eine mit *FROM* eingeleitete Auflistung der *Tabellen* auf die sich die *SELECT*-Anweisung bezieht.

Vielleicht fragen Sie sich jetzt, warum Sie Ihr bewährtes System, *Abfragen* in der *Entwurfsansicht* zu erstellen, ändern sollen?

Nun, das ist nicht notwendig – es gibt allerdings neben *ACCESS* noch eine Reihe weiterer Datenbankanwendungen die nach unterschiedlichen Verfahren auf Daten zugreifen. Während *ACCESS* beide Methoden, das bewegungsorientierte und das relationale Verfahren beherrscht, können heute die am häufigsten verbreiteten *relationalen Datenbanken* nur mittels *SQL-Anweisung* abgefragt werden.

ACCESS kann über die *ODBC-Schnittstelle* auch auf diese *Datenbanken* zugreifen.

Mit *SQL* können komplexe *Abfragen* erstellt werden, die neben *Tabellen* auch andere *Abfragen* enthalten können und bestimmte Abfrageergebnisse können nur mit Hilfe einer *SQL-Anweisung* ermittelt werden.

In *Unterabfragen* können *SQL-Anweisungen* als *Kriterium* eingebunden werden.

ACCESS erstellt eine *SQL-Anweisung* wenn mit Hilfe des Assistenten beispielsweise ein *Kombinationsfeld* als *Suchfeld* eingerichtet wird und fügt die *Syntax* im *Eigenschaftenblatt* des *Steuerelements* ein (*Abb. 19*). Die *Syntax* kann anschließend verändert und Ihren Anforderungen entsprechend angepasst werden.

Außerdem ist eine in der *SQL-Ansicht* erstellte *Syntax* übersichtlicher und kann auch leichter gemailt oder gepostet werden.

Der Einstieg in *SQL* ist für Anfänger verständlicher, weil sie zwischen der *Entwurfsansicht* und der *SQL-Ansicht* hin- und herwechseln, und die *Syntax* im Ergebnis anschauen können.

Die in *Abb. 2* dargestellte *Syntax* kann noch wie folgt ergänzt werden:

SELECT DISTINCTROW [Schüler-Stammdaten].Name
FROM [Schüler-Stammdaten];

Der Zusatz *DISTINCTROW* bewirkt, dass alle doppelten *Datensätze* entfernt werden und eine Bearbeitung der *Abfrage* möglich ist; d.h. die Einträge im Feld *Name* können verändert werden - man spricht in diesem Fall von einem *Dynaset*.

Anders beim Zusatz *DISTINCT.* Zwar werden auch hier die doppelten *Datensätze* entfernt aber das Ergebnis der *Abfrage* ist gesperrt und kann **nicht** verändert werden.

Beispiel:

Mit dieser *Abfrage* sollen die Schüler ermittelt werden, deren Telefon-Nr. mit *092…* beginnt.

Die in *Abb. 4* dargestellte *Abfrage* zeigt den Aufbau in der *Entwurfsansicht*.

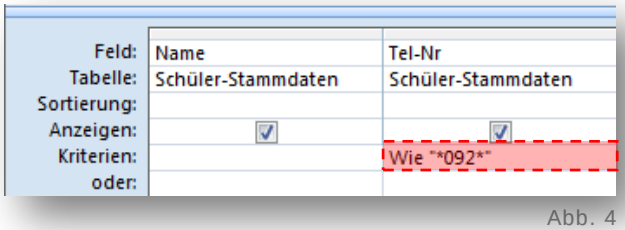


Abb. 4

Die automatisch von *ACCESS* generierte *SQL-Anweisung* (Abb. 5) lautet:

```
SELECT [Schüler-Stammdaten].Name, [Schüler-Stammdaten].[Tel-Nr]
FROM [Schüler-Stammdaten]
WHERE (([Schüler-Stammdaten].[Tel-Nr] Like "*092*"));
```
Abb. 5

Nach der *SELECT-Anweisung* kann der Zusatz *DISTINCTROW* folgen, wenn doppelte *Datensätze* eliminiert werden sollen und eine Bearbeitung der Daten möglich sein soll, gefolgt von der *Feldliste* – bei mehreren *Feldern* getrennt durch ein Komma. Anschließend wird mit der Anweisung *FROM* die Herkunftstabelle eingebunden – bei mehreren *Tabellen* wiederum getrennt durch ein Komma – und das Kriterium *WHERE* im Feld *[Tel-Nr]* eingefügt: *Like* (wie) "*092*", damit nur diese *Datensätze* angezeigt werden.
Das *Kriterium* muss erfüllt sein und so formuliert werden, dass es einen logischen Wert (*TRUE/FALSE*) zum Ergebnis hat. *Kriterien* können außerdem die Operatoren *AND* und *OR* enthalten.

Wird die Anweisung manuell in der *SQL-Ansicht* eingegeben, kann die wiederholte Benennung der *Tabelle* auch entfallen, wodurch die *Syntax* kürzer und übersichtlicher wird:

```
SELECT Name, [Tel-Nr]
FROM [Schüler-Stammdaten]
WHERE ([Tel-Nr]) Like "*092*";
```

Sollen alle *Felder* einer *Tabelle* angesprochen werden, kann die *Syntax* auch wie folgt lauten (der * hinter *SELECT* ist ein *Platzhalterzeichen* für alle *Felder* in der *Tabelle*):

```
SELECT *
FROM [Schüler-Stammdaten]
WHERE ([Tel-Nr]) Like "*092*";
```

Beispiel:

Dieses Beispiel zeigt eine Sortieranweisung im Feld *[Ort]*.

Die *Syntax* beginnt wie in den anderen Beispielen, ergänzt durch die Sortier-Anweisung: *ORDER BY* die noch mit den Parametern *DESC* (absteigende Sortierreihenfolge) oder *ASC* (aufsteigende Sortierreihenfolge) ergänzt werden kann.

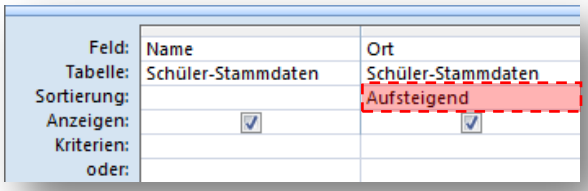


Abb. 6

```
SELECT [Schüler-Stammdaten].Name, [Schüler-Stammdaten].Ort
FROM [Schüler-Stammdaten]
ORDER BY [Schüler-Stammdaten].Ort;
```

Abb. 7

Beispiel:

Beispiel für eine *Gruppierung* im Feld *[Ort]*.

Die *Syntax* beginnt wieder wie in den anderen Beispielen, ergänzt durch die Gruppieren-Anweisung: *GROUP BY*.

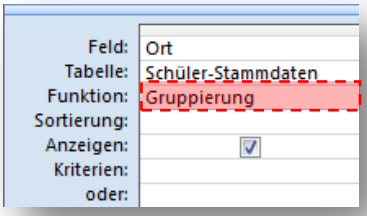


Abb. 8

```
SELECT [Schüler-Stammdaten].Ort
FROM [Schüler-Stammdaten]
GROUP BY [Schüler-Stammdaten].Ort;
```

Abb. 9

Beispiel:

In diesem Beispiel sollen im Feld *[Ort]* die Aggregatfunktion *Gruppierung* und im Feld *[Name]* die Aggregatfunktion *Anzahl* aktiviert werden.

Nach der *SELECT-Anweisung* folgen die Felder *[Ort]* und *[Name]*. Im Feld *[Name]* wird zusätzlich die Aggregatfunktion *Anzahl (Count)* eingebunden um die Anzahl der Schüler zu ermitteln und nach *AS* der neu generierte Feldname *AnzahlvonName*.

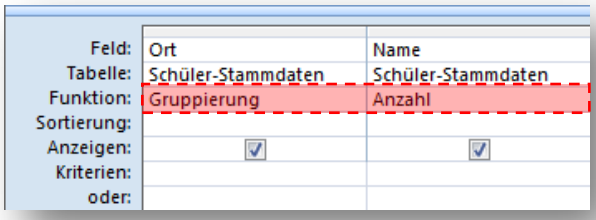


Abb. 10

```
SELECT [Schüler-Stammdaten].Ort, Count([Schüler-Stammdaten].Name) AS AnzahlvonName
FROM [Schüler-Stammdaten]
GROUP BY [Schüler-Stammdaten].Ort;
```

Abb. 11

Mit *FROM* wird anschließend die Herkunftstabelle, und mit *GROUP BY die* Gruppieren-Anweisung im Feld *[Ort]* eingefügt.

Kurzform: *SELECT Ort, Count(Name) AS AnzahlvonName*
FROM [Schüler-Stammdaten]
GROUP BY Ort;

Beispiel:

Dieses Beispiel zeigt, wie mittels *Abfrage* die Vermiettage und Vermietgebühren in einer Videothek ermittelt werden können.

```
SELECT Kunden.[Kd-Nr], Kunden.FName, Kunden.[Geb-Datum], Kunden.Mitglied,
Kunden.Eintritt, Vermietung.[Film-Nr], Vermietung.[Vermietung am],
Vermietung.[Rückgabe am], Filme.Titel, [FSK ab] & " Jahre" AS FSK,
DateDiff("d",[Vermietung am],[Rückgabe am]) AS Vermiettage, IIf([Mitglied]=True
And Not IsNull([Beitrag]),[Vermiettage]*[Tagespreis]/2,[Vermiettage]*[Tagespreis])
AS Betrag, Vermietung.bezahlt
FROM Kunden INNER JOIN (Filme INNER JOIN Vermietung ON Filme.[Film-Nr] =
Vermietung.[Film-Nr]) ON Kunden.[Kd-Nr] = Vermietung.[Kd-Nr];
```

Abb. 12

Nach der *SELECT-Anweisung* folgt zunächst eine Auflistung der *Felder,* die in die *Abfrage* eingefügt wurden. Enthält die *Abfrage* ein berechnendes *Feld*, wird dieses *zusätzliche Datenfeld* nach der *Formel* mit *AS* eingebunden (*Abb. 12 – schwarzer* Kreis).

Nach *FROM* folgt wieder die Auflistung der *Tabellen* auf die sich die *SELECT-Anweisung* bezieht.

Die Verknüpfung der in die *Abfrage* eingebundenen *Tabellen* (*Abb. 13*) erfolgt mit *JOIN*. Die Anweisung *INNER* bezieht sich dabei auf die verknüpften *Datensätze* und *ON* enthält die *Bedingung* für das *Kriterium*, wodurch die Beziehung zwischen *Primär-* und *Fremdschlüsselfeld* definiert wird.

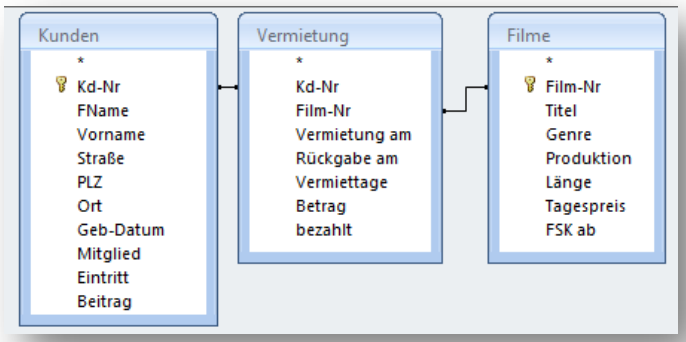


Abb. 13

Bei der *IIf-Funktion* handelt es sich um eine *VBA-Funktion,* die verwendet wird, wenn nur einer von zwei Werten (*WENN…DANN*) zurückgegeben werden soll.

In der *Entwurfsansicht* wird diese *SQL-Anweisung* wie folgt dargestellt:

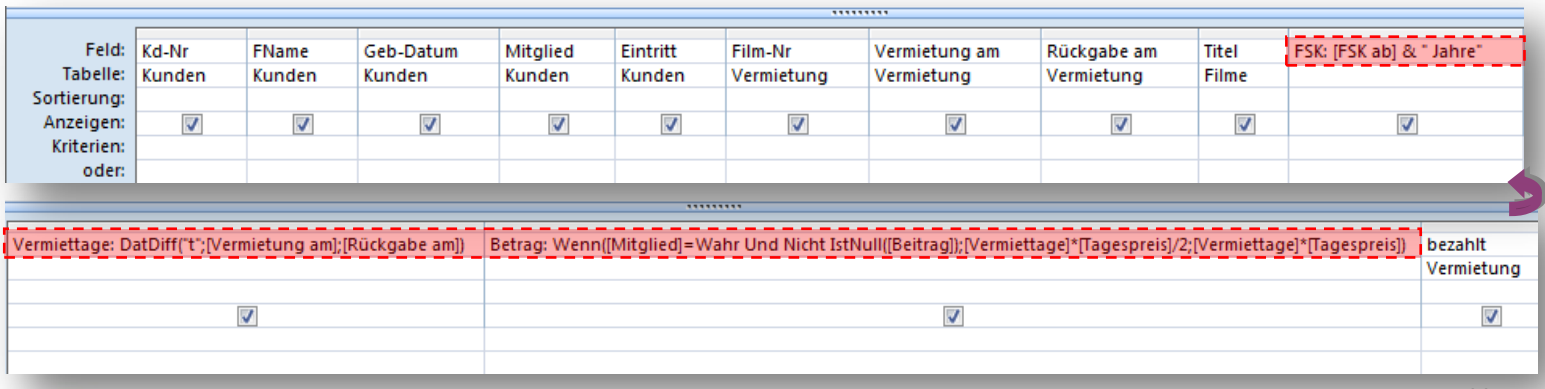
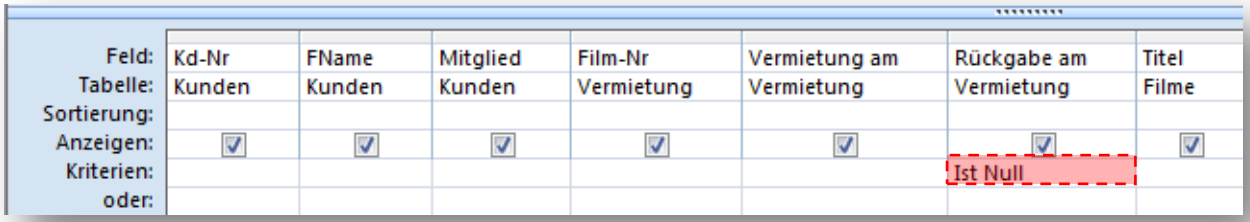


Abb. 14

Beispiel:

Dieses Beispiel zeigt, wie mit Hilfe einer *Abfrage* nicht zurückgegebene Videofilme ermittelt werden können. Mit dem Kriterium *Ist Null* wird im Feld *[Rückgabe am]* geprüft, ob das *Feld* einen Datumseintrag enthält - oder ob es 'leer' ist.



Feld:	Kd-Nr	FName	Mitglied	Film-Nr	Vermietung am	Rückgabe am	Titel
Tabelle:	Kunden	Kunden	Kunden	Vermietung	Vermietung	Vermietung	Filme
Sortierung:							
Anzeigen:	☑	☑	☑	☑	☑	☑	☑
Kriterien:						Ist Null	
oder:							

Abb. 15

Die entsprechende *SQL-Anweisung* hat folgende *Syntax:*

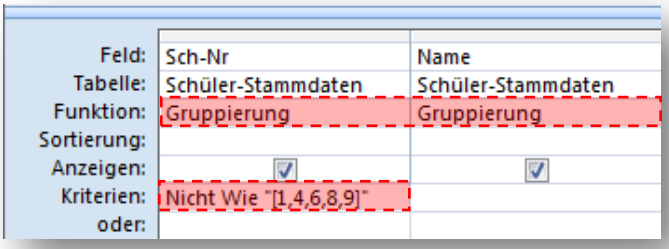
```
SELECT Kunden.[Kd-Nr], Kunden.FName, Kunden.Mitglied, Vermietung.[Film-Nr],
Vermietung.[Vermietung am], Vermietung.[Rückgabe am], Filme.Titel
FROM Kunden INNER JOIN (Filme INNER JOIN Vermietung ON Filme.[Film-Nr] =
Vermietung.[Film-Nr]) ON Kunden.[Kd-Nr] = Vermietung.[Kd-Nr]
WHERE (((Vermietung.[Rückgabe am]) Is Null));
```

Abb. 16

Mit der Anweisung *WHERE* kann in einer *Abfrage* ein *Kriterium* formuliert werden. Im vorliegenden Fall werden alle *Datensätze* ausgegeben, die im Feld *[Rückgabe am]*, Zeile *Kriterien* den Wert *Is Null* ausweisen – also keinen Eintrag enthalten.

Beispiel:

Dieses Beispiel zeigt, wie *Datensätze* gruppiert, und mit Hilfe eines *Kriteriums* von der Anzeige ausgeschlossen werden können.



Feld:	Sch-Nr	Name
Tabelle:	Schüler-Stammdaten	Schüler-Stammdaten
Funktion:	Gruppierung	Gruppierung
Sortierung:		
Anzeigen:	☑	☑
Kriterien:	Nicht Wie "[1,4,6,8,9]"	
oder:		

Abb. 17

Mit der Anweisung *HAVING* wird beim Gruppieren das *Kriterium* definiert, und mit der Anweisung *NOT LIKE* – nicht wie – können *Datensätze* von der Anzeige ausgeschlossen werden.

```
SELECT DISTINCTROW [Schüler-Stammdaten].[Sch-Nr], [Schüler-Stammdaten].Name
FROM [Schüler-Stammdaten]
GROUP BY [Schüler-Stammdaten].[Sch-Nr], [Schüler-Stammdaten].Name
HAVING ((([Schüler-Stammdaten].[Sch-Nr]) Not Like "[1,4,6,8,9]"));
```

Abb. 18

Kurzform der Syntax:

```
SELECT DISTINCTROW [Sch-Nr], Name
FROM [Schüler-Stammdaten]
GROUP BY [Sch-Nr], Name
HAVING .[Sch-Nr] Not Like "[1,4,6,8,9]";
```

Wie bereits erwähnt, wird beim Einrichten eines *Kombinationsfeldes* als *Suchfeld* und mit Hilfe des Assistenten von *ACCESS* eine *SQL-Anweisung* erstellt und diese im *Eigenschaftenblatt* des *Steuerelements* in Zeile *Datensatzherkunft* eingebunden:

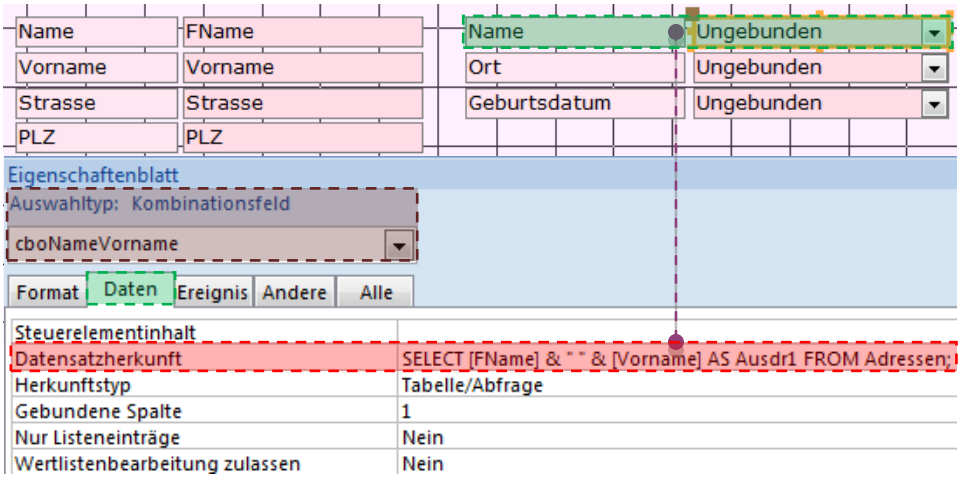


Abb. 19

Elemente der Syntax:

SELECT	-	Auswahl
Feldliste	-	Auflistung durch Komma trennen – bei gleichen Feldnamen: Name der Tabelle voranstellen
FROM	-	aus Tabelle
Tabellenliste	-	Auflistung durch Komma trennen
INNER JOIN	-	Verknüpfung der Datensätze zweier Tabellen
ON	-	Bedingung bei Übereinstimmung der Felder
WHERE	-	Einleitung Kriterium
Kriterium	-	Kriterium mit dem Ergebnis TRUE oder FALSE
GROUP BY	-	Gruppierung
HAVING	-	Einleitung Kriterium
Kriterium	-	Kriterium bei Gruppierung mit dem Ergebnis TRUE oder FALSE
AND/OR	-	Einleitung Bedingung
Bedingung	-	Bedingung mit dem Ergebnis TRUE oder FALSE
ORDER BY	-	Einleitung Sortierung
Feldname	-	Feld das sortiert werden soll
DESC/ASC	-	absteigende Sortierung / aufsteigende Sortierung (Standard)

Mehr Informationen zu diesem Thema finden Sie auch in der *ACCESS-Offline-Hilfe* nach Eingabe des Suchbegriffs: *SQL*.