Relationenalgebra und SQL

- Begriff der Algebra
- klassische Mengenoperationen
  (Vereinigung, Durchschnitt, Differenz)
- Operationen auf Relationen
  - Projektion
  - Selektion
  - Kreuzprodukt
  - Innerer Verbund / Gleichverbund

Zum Begriff der Algebra

In der Mathematik ist eine Algebra ein Paar von Trägermenge T und einer Menge von Operationen Op, also (T,Op).
Die Operationen sind auf T erklärt. Jede Operation muss als Ergebnis wieder ein Element von T liefern.

Relationenalgebra:
In der Relationenalgebra ist die Trägermenge die Menge der Relationen MR. Die Menge der Operationen enthält die Elemente:

- Vereinigung
- Durchschnitt
- Differenz
- Projektion
- Selektion
- Kreuzprodukt
- Inner Join (mit seinen Varianten)
- Outer Join (mit seinen Varianten)

Klassische Mengenoperationen (1)

Vereinigung:
Es seien R und S zwei Relationen, die das gleiche Relationenformat E besitzen müssen. Dann heißt

$R \cup S = \{ r \mid r \in R \text{ ODER } r \in S \}$ die Vereinigung von R und S.

Die Vereinigung ist unersetzbar. In SQL ist dieser Operator durch UNION repräsentiert und wird von nahezu allen DBMS realisiert.

Durchschnitt:
Es seien R und S zwei Relationen, die das gleiche Relationenformat E besitzen müssen. Dann heißt

$R \cap S = \{ r \mid r \in R \text{ UND } r \in S \}$ der Durchschnitt von R und S.

Der Durchschnitt ist ersetzbar durch die Selektion mit EXISTS-Prädikat. In SQL ist dieser Operator durch INTERSECT repräsentiert, wird aber von wenigen DBMS realisiert.

Klassische Mengenoperationen (2)

Differenz:
Es seien R und S zwei Relationen, die das gleiche Relationenformat E besitzen müssen. Dann heißt

$R - S = \{ r \mid r \in R \text{ UND } r \notin S \}$ die Differenz von R und S.

Die Differenz ist ersetzbar durch die Selektion mit NOT EXISTS-Prädikat. In SQL ist dieser Operator durch EXCEPT repräsentiert, wird aber von wenigen DBMS realisiert.

Operationen auf Relationen - Projektion

Die Projektion ist eine der am meisten benutzten relationalen Operationen. Es wird dabei auf eine Teilmenge von Eigenschaften des Relationenformats einer Relation R projiziert.

Projektion:
Es sei R eine Relation mit dem Relationenformat E. Dann heißt

$$\Pi_{Ea,Eb,Ec,...} (R) = \{ (r.E_a, \mathbf{r.E_b, r.E_c,...}) \mid r \in R \text{ UND } \{E_a,E_b,E_c,...\} \subseteq E \}$$

die Projektion von R auf die Eigenschaften $\{E_a,E_b,E_c,...\}$.

Die Projektion wird in SQL in der SELECT-Klausel kodiert. Die projizierten Eigenschaften werden hinter SELECT als Liste angegeben. Wird keine Projektion durchgeführt, d.h. es sind alle Eigenschaften des Relationenformats ausgewählt, wird dies in SQL durch einen Stern (*) anstelle der Liste angegeben.

Operationen auf Relationen – Selektion (1)

Die Selektion ist neben der Projektion eine der am meisten benutzten relationalen Operationen. Es wird dabei ein Prädikat P über Eigenschaften des Relationenformats einer Relation R gebildet. Nur Tupel r ϵ R , für die das Prädikat erfüllt ist, werden in die Ergebnisrelation Σ(R) aufgenommen.

Selektion:
Es sei R eine Relation mit dem Relationenformat E. Dann heißt

Σ (R) = { r | r ϵ R UND P(r)=wahr }
die Selektion von r bezüglich des Prädikates P.

Zur Kodierung des Prädikats gibt es mehrere Möglichkeiten. Es kann durch einfache Vergleichsausdrücke realisiert werden. Teilausdrücke können durch die Junktoren AND, OR, NOT verknüpft werden. In diesen Ausdrücken sind auch Standardfunktionen erlaubt. Weiterhin können auch bestimmte Standardprädikate, wie IN, IS, LIKE auftreten. Bei Benutzung von Subselects treten standardmäßig Prädikate ALL, ANY, EXISTS auf.

Operationen auf Relationen – Selektion (2)

Das Prädikat P wird in SQL in der WHERE-Klausel kodiert. Beispielhaft sollen hier einige Möglichkeiten für die Nutzung von Prädikaten angegeben werden.

Vergleichsausdrücke:

Gehalt>=2000 AND Geschlecht= ‘m‘

YEAR(Geburtstag)<1980

Standardprädikate:

Name LIKE ‘%nn%‘

Ort IN (‘Bautzen‘, ‘Dresden‘)

Telefon IS NULL

Operationen auf Relationen – Kreuzprodukt

In der Datenbanktechnologie wird das Kreuzprodukt selbst nur selten benutzt. Es bildet aber das Gerüst für die sehr wichtigen und häufig angewendeten Join-Operationen.

Kreuzprodukt:
Es seien R eine Relation mit dem Relationenformat ER und S eine Relation mit dem Relationenformat ES. Dann heißt

$$R * S = \{ (er_1,er_2,\ldots,er_n,es_1,es_2,\ldots,es_m) \mid (er_1,er_2,\ldots,er_n) \in R \text{ UND } (es_1,es_2,\ldots,es_m) \in R \text{ UND } E = ER \cup ES\}$$

der Cross Join von R und S. E ist das Relationenformat von R * S

Das Kreuzprodukt (Cross Join) liefert als Resultat alle zusammengesetzten Tupel, die sich aus allen Kombinationen der Tupel der beiden Relationen ergeben. In SQL werden alle (expliziten) Join-Operatoren in der FROM-Klausel kodiert. R * S wird durch R CROSS JOIN S repräsentiert. Alle DBMS realisieren diese Operation.

Operationen auf Relationen – JOIN

Der **Equi-Join** ist die am meisten benutzte Join-Operation.
Das =-Zeichen ist die Vergleichsoperation in der Join-Bedingung.

Equi-Join:
Es seien R eine Relation mit dem Relationenformat ER und S eine Relation mit dem Relationenformat ES. Es existiert eine Eigenschaft ER_k von R und eine Eigenschaft ES_j von S deren Wertebereiche gleich sind, $W_k = W_j$. Dann heißt

$$R \underset{B}{\times} S = \{ q \mid q \in R * S \wedge q.ER_k = q.ES_j \wedge E = ER \cup ES \}$$

der Theta-Join von R u. S. E ist das Relationenformat von $R \underset{B}{\times} S$ und *B* ist die Join-Bedingung $q.ER_k = q.ES_j$.

Der Equi-Join liefert als Resultat nur diejenigen zusammengesetzten Tupel q aus R * S, die gleiche Werte in den bezogenen Eigenschaften ER_k und ES_j haben, d.h. die die Join-Bedingung *B* erfüllen. Alle DBMS realisieren Equi-Joins.

Operationen auf Relationen – JOIN

Equi-Join (Vereinigung über Gleichheit ausgewählter Eigenschaften)

SQL:

Select * FROM Verkaeufe , Produkte WHERE Verkaeufe.ProduktID = Produkte.ProduktID;

oder

Select * FROM Verkaeufe INNER JOIN Produkte ON Verkaeufe.ProduktID = Produkte.ProduktID;

Operationen –THETA JOIN (1)

Der Theta-Join ist eine Verallgemeinerung des Equi-Join. Das =-Zeichen des Gleichverbundes kann jetzt durch einen beliebigen Vergleichsoperator, wie z.B. <, >, !=, <= usw. ersetzt werden. Das Zeichen θ (Theta) steht als Platzhalter für einen beliebigen Vergleichsoperator.

Es seien R eine Relation mit dem Relationenformat ER und S eine Relation mit dem Relationenformat ES. Es existiert eine Eigenschaft ER_k von R und eine Eigenschaft ES_j von S deren Wertebereiche gleich sind, $W_k = W_j$. Dann heißt

$$R \underset{B}{\times} S = \{ q \mid q \in R * S \wedge q.ER_k \, \theta \, q.ES_j \wedge E = ER \cup ES \}$$

der Theta-Join von R u. S. E ist das Relationenformat von $R \underset{B}{\times} S$ und *B* ist die Join-Bedingung $q.ER_k \, \theta \, q.ES_j$.

Operationen – THETA JOIN (2)

Der Theta-Join liefert als Resultat nur diejenigen zusammengesetzten Tupel q aus R * S, die bezüglich des θ-Operators vergleichbare Werte in den bezogenen Eigenschaften ER_k und ES_j haben, d.h. die die Join-Bedingung *B* erfüllen.

Operationen – OUTER JOINS

OUTER JOINS – sind Zusammensetzungen aus Tabellen, wobei eine Tabelle eine konditionale Beziehungen ausdrückt, z.B. Assoziationstyp (mc,mc).

Beispiel: M-Mitarbeiter, P-Projekte

M | P | leftjoin_Beispiel

MNR	MName
1	Schulze
2	Müller
3	Lehmann
4	Meier
5	Wolf

M | P | leftjoin_Beispiel | rightjoin_Beispiel

PNR	MNR	PROJNAME	PROZENT
1	1	Großprojekt ABC	50
2	3	Kundenbetreuung XY	70
3	2	Kleinarbeiten	100
4	3	Interne Forschung	30
5	1	Megaprojekt in Übersee	50
6		Zukunftsprojekt 2044	0
7		Projekt "Auf Eis gelegt"	0

Es gibt Mitarbeiter ohne Projekt, und es gibt Projekte ohne Mitarbeiter. Bei Outer Joins werden diese berücksichtigt.

Hinweis: In einer vollständigen Variante wären drei Tabellen laut Entwurfsregeln nötig: M, P und eine Zuordnung von M zu P

Operationen – OUTER JOINS

LEFT OUTER JOIN: Als Ergebnis werden alle Mitarbeiter mit ihren Projekten gewünscht, aber auch jene, die in keinem Projekt eingebunden sind.

SELECT MNR,PROJNAME FROM M LEFT JOIN P WHERE M.MNR = P.MNR

Ergebnis:

M | P | leftjoin_Beispiel | rightjoin_Beispiel

MNR	PROJNAME
1	Megaprojekt in Übersee
1	Großprojekt ABC
2	Kleinarbeiten
3	Interne Forschung
3	Kundenbetreuung XY
4	
5	

Hier kommen von links auch die Mitarbeiter dazu, die in keinem Projekt aufgeführt wurden.

Operationen – OUTER JOINS

RIGHT OUTER JOIN: Als Ergebnis werden alle Projekte mit zugeordneten Mitarbeiterdaten gewünscht, auch jene denen kein Mitarbeiter zugeordnet ist.

SELECT MNR,PROJNAME FROM M RIGHT JOIN P WHERE M.MNR = P.MNR

Ergebnis:

M | P | leftjoin_Beispiel | rightjoin_Beispiel

MNR	PROJNAME
1	Großprojekt ABC
3	Kundenbetreuung XY
2	Kleinarbeiten
3	Interne Forschung
1	Megaprojekt in Übersee
	Zukunftsprojekt 2044
	Projekt "Auf Eis gelegt"

Hier kommen von rechts auch die Projekte dazu, die (noch) keine Verbindung zu Mitarbeitern haben.

Inter- und Intrarelationales JOIN (1)

Interrelationale Join-Operationen
verknüpfen entsprechend einer Verbundbedingung Tupel (Entitäten) von zwei verschiedenen Tabellen.

Beispiel: ***Select * FROM Verkaeufe, Produkte WHERE Verkaeufe.ProduktID = Produkte.ProduktID;***

Intrarelationale Join-Operationen (auch so genannte „Self-Joins“)
verknüpfen entsprechend einer Verbundbedingung Tupel (Entitäten) ein und derselben Tabelle.

Inter- und Intrarelationales JOIN (2)

Intrarelationale Join-Operationen (Self-Joins)
verknüpfen entsprechend einer Verbundbedingung Tupel ein und derselben Tabelle. Die Anwendung kann sinnvoll sein, wenn im Diskursbereich eine Beziehung in der gleichen Entitätsmenge definiert ist, wie z.B. Menge der Buchautoren mit der Beziehung „ist Mitautor“. Aber auch ohne eine solche Beziehung gibt es sinnvolle Anwendungen, zum Beispiel die Anfrage: „Welche Mitarbeiter wohnen im gleichen Ort (PLZ)?“ für die unten angegebene Tabelle.

Person

MNR	MNAME	PLZ	STRASSE
1235	Oberhuber	80995	Moshammerstr. 1
2467	Schulze	10117	Zillestr. 23
2477	Meier	01069	Bergstr. 12
2612	Höflmayr	80995	Alpenweg 8

SQL-Anfrage:

SELECT E.MNR, Z.MNR, E.PLZ FROM Person AS E, Person AS Z
WHERE E.PLZ=Z.PLZ

Da hier gleiche Namen verwendet werden, ist es erforderlich, die Tabelle und ihre Kopie durch Aliasnamen (hier E und Z) auseinander zu halten.

Inter- und Intrarelationales JOIN (3)

Das Ergebnis der letzten Anfrage wäre:

MNR	MNR	PLZ
1235	1235	80995
1235	2612	80995
2467	2467	10117
2477	2477	01069
2612	1235	80995
2612	2612	80995
.....	.....	.....

Wir sehen, dass leider alle Kombinationen von Mitarbeitern kommen, sogar die Verknüpfung mit sich selbst. Um das auszuschließen, ergänzen wir die Abfrage:

*SELECT **E**.MNR,**Z**.MNR, E.PLZ FROM Person AS **E**,Person AS **Z***
*WHERE **E**.PLZ=**Z**.PLZ **AND E**.MNR<**Z**.MNR*

Inter- vs. Intrarelationales JOIN (4)

Jetzt erhalten wir das gewünschte Ergebnis mit paarweise verschiedenen Personen, die im gleichen Ort wohnen:

MNR	**MNR**	**PLZ**
1235	2612	80995
.....	.....	.....

Auch mit geschachtelten SELECT-Anweisungen (Subselects) lassen sich ähnliche Fragestellungen bearbeiten.

Datenbank-(Programmier)-Sprachen (1)

Mittels Visual Basic ist es möglich, auf Tabellen durch s.g. Recordsets zuzugreifen, siehe Abschnitt 6, ab Folie 8. Diese Zugriffsart greift nicht auf Operationen der Relationenalgbra zurück. D.h. die Auswahl von Spalten oder Zeilen müsste im Anwendungsprogramm auf imperative Weise programmiert werden.

Es gibt deshalb s.g. ***relational vollständige Sprachen***, mit denen der Zugriff auf Datenbanken erfolgt und die eine Menge von Operationen der Relationenalgebra enthalten.

Datenbank-(Programmier)-Sprachen (2)

Relational vollständige Sprache:
Eine Datenbankabfragesprache heißt relational vollständig im Sinne der Relationenalgebra, wenn sie mindestens die mengenorientierten Operationen Vereinigung, Subtraktion, kartesisches Produkt, sowie die relationenorientierten Operationen Projektion und Selektion ermöglicht.

Die mengen- und relationenorientierten Operationen dienen vorrangig der Abfrage der Datenbank. Daneben müssen noch Manipulationsoperationen durch die Sprache bereitgestellt werden, z.B. für das Einfügen, Löschen oder Verändern von Tupelmengen.

Datenbank-(Programmier)-Sprachen (3)

Zusatzoperationen für eine relationale Sprache:

Manipulation:

- Definition neuer Tabellen mit deren Eigenschaften
- Löschen von Tabellen
- Einfügen neuer Tupel in eine Tabelle
- Löschen von Tupeln
- Verändern von Werten innerhalb der Tupel

Aggregationsfunktionen für Spaltenwerte:

- Summenbildung
- Minimum, Maximum, Durchschnitt

Formatierung und Sortierung:

- sortierte Ausgabe der Tupel nach Spaltenwerten
- Reihenfolge der Spalten

Datenbank-(Programmier)-Sprachen (4)

Zusatzoperationen für eine relationale Sprache (Fortsetzung):

- Nutzerverwaltung und Zugangsschutz
- Unterstützung von Berechnungen und arithmetischen Ausdrücken
- Unterstützung des Mehrnutzerbetriebs

Ein **Mehrnutzerbetrieb** bedarf ein zusätzlicher Verfahren, sobald einer der Nutzer Daten ändern kann, die andere Nutzer ggf. auch ändern oder abfragen.

Beispiel: Ticketbuchungssystem mit Abfrage freier Sitzplätze und Belegung der Sitzplätze, Mehrfachbuchungen wären möglich

Lösungsmöglichkeit:

- Sperren von Tabellen – einfach, aber für große Systeme ungeeignet
- Transaktionen – komplexe Mechanismen zur Konfliktlösung

SQL: Structured Query Language

- Datenbankspezifische Manipulations- und Abfragesprache
- eine relational vollständige Sprache
- Eine Deklarative Sprache – Man beschreibt das Ziel der Operation, Unterschied gegenüber imperativer Programmierung, wie z.B. mit Visual Basic
- In den siebziger Jahren entwickelt, inzwischen ISO-Norm
- Weit verbreitet: MS-ACCESS, MS-SQL-Server, mySQL u.v.a.m.
- In der Vorlesung bislang schon zur Erklärung der Operationen der Relationenalgebra benutzt

Tabellen und Abfragen mittels SQL (1)

Typische Benutzungs-Fälle:

Abfragen innerhalb eines Anwendungsprogramms, mit DAO-Anbindung

```
dbe = New dao.DBEngine()
db = dbe.OpenDatabase("C:\\TEMP\\firma.mdb")
rs = db.OpenRecordset(SQLAnfrage)

Listbox.Items.Clear()
If Not rs.EOF Then rs.MoveFirst()
  Listbox.BeginUpdate()
  Do While Not rs.EOF
    str = ",,
    For i = 0 To rs.Fields.Count - 1
        str = str + " " + rs.Fields(i).Name + " : "+
                                    Format(rs.Fields(i).Value)
```

SQL-Anfrage in einem String

Tabellen und Abfragen mittels SQL (2)

Abfragen innerhalb eines Anwendungsprogramms,
mit OLE-Data-Provider
(**O**bject **L**inking and **E**mbedding, als interne Microsoft-Schnittstelle)

```
Imports System.Data.OleDb
Public Class Form1
  Private dt As New DataTable
  Private da As OleDbDataAdapter
  Private ds As New DataSet
  Private bs As New BindingSource

  Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs)
                                Handles Me.Load
    Dim ConStr As String = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\TEMP\firma.mdb;"
    Dim Con As New OleDbConnection(ConStr)
    ds.Tables.Add(dt)
    da = New OleDbDataAdapter("SELECT * FROM personen", Con)
    da.Fill(dt)
    bs.DataSource = dt
   Me.DataGridView1.DataSource = bs
End Sub
```

SQL-Anfrage in einem String

Tabellen und Abfragen mittels SQL (3)

Abfragen innerhalb eines Anwendungsprogramms,
mit OLE-Data-Provider (Fortsetzung)

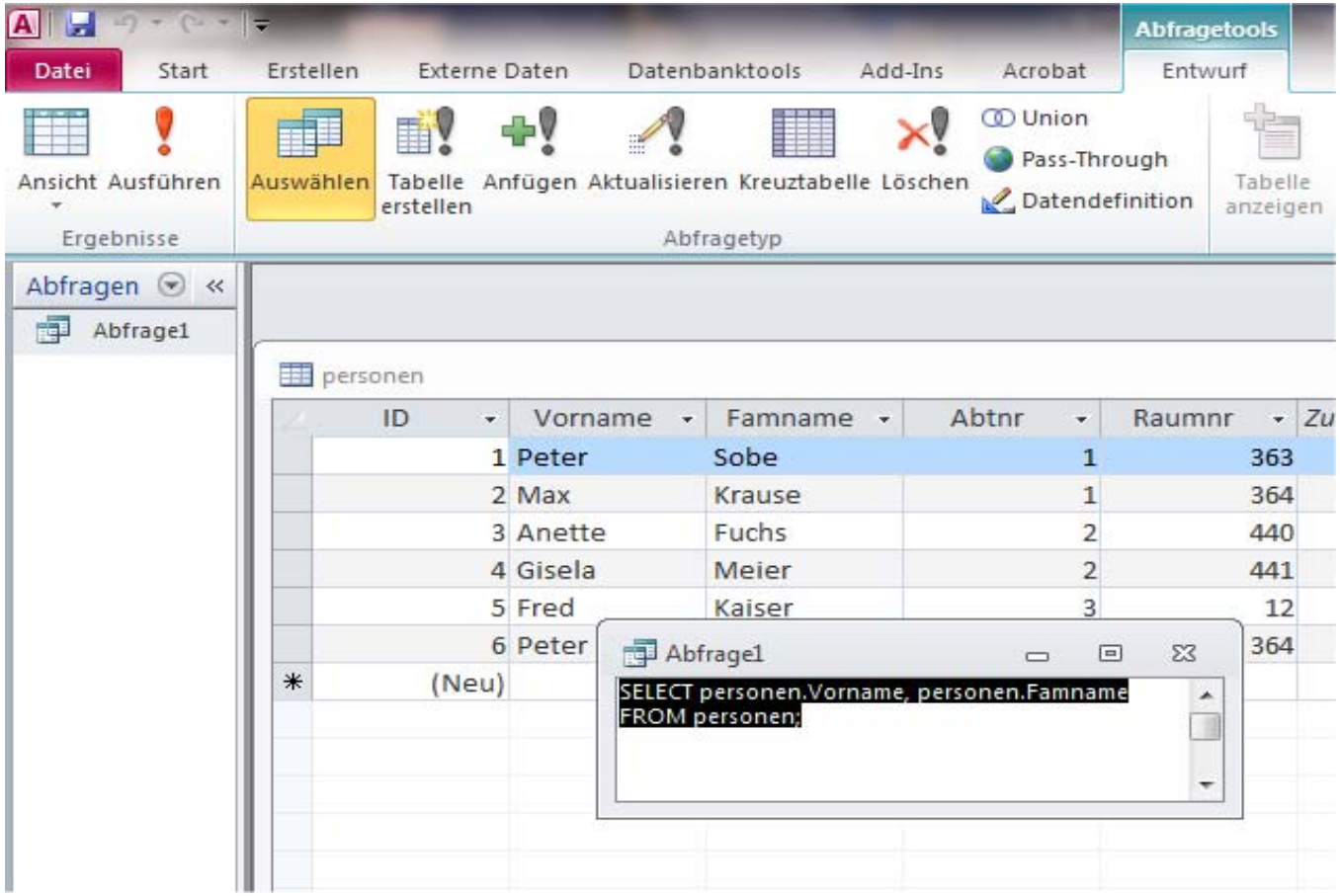
```
Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
        System.Windows.Forms.FormClosingEventArgs) Handles Me.FormClosing
            If ds.HasChanges Then
              If MsgBox("Aenderungen speichern?",
                          MsgBoxStyle.Question Or MsgBoxStyle.YesNo) Then
                  bs.EndEdit()
                  da.Update(ds.Tables(0))
              End If
            End If
End Sub

End Class
```

Änderungen der Daten durch das Programm können so auf die Datenbank zurückgeschrieben werden.

Tabellen und Abfragen mittels SQL (4)

Abfragen innerhalb einer Datenbanksoftware:

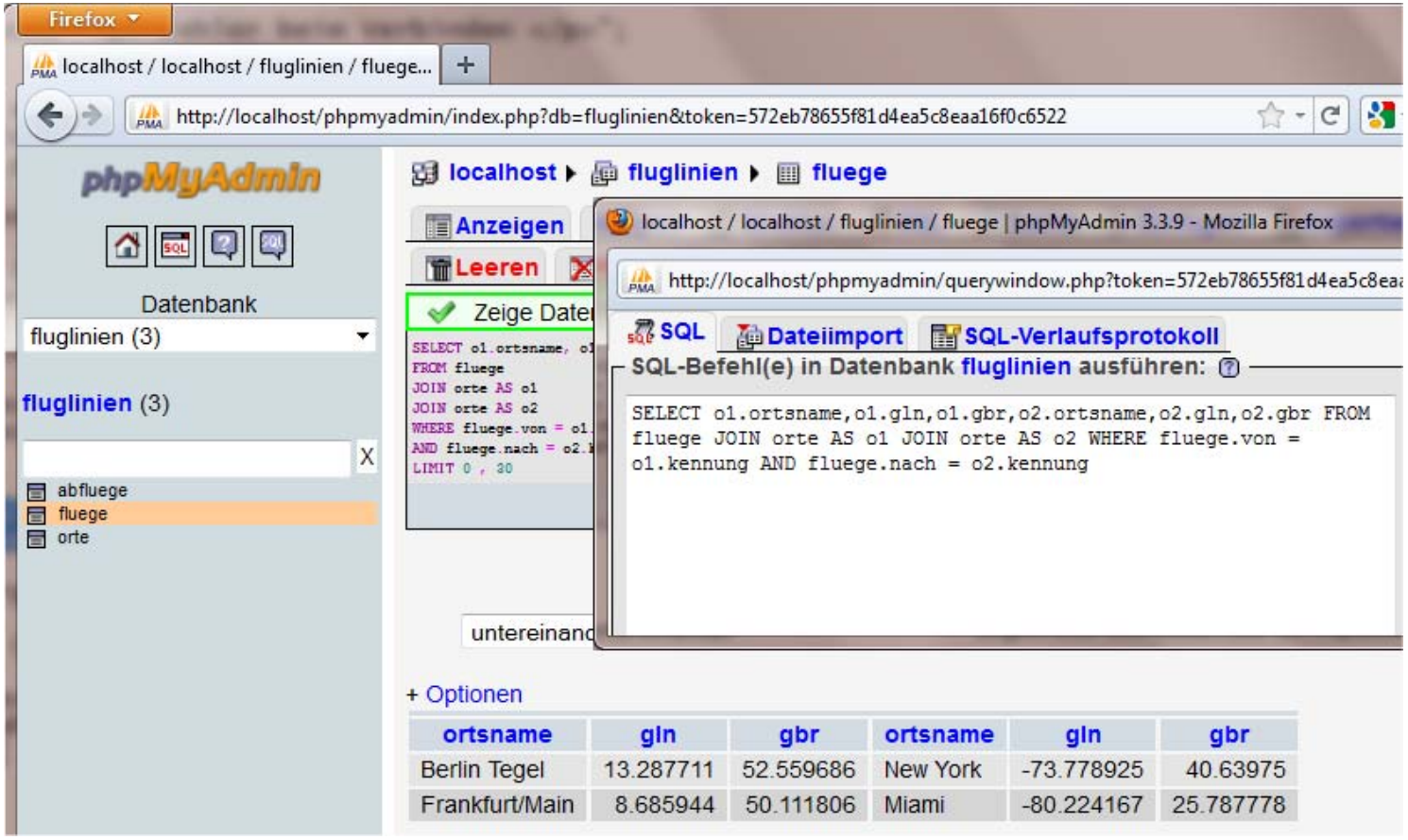
z.B. Microsoft Office Access –
Abfrageeditor, mit SQL-Ansicht



Tabellen und Abfragen mittels SQL (5)

Abfragen innerhalb einer Datenbanksoftware:

PHPMyAdmin für mySQL-Datenbanken
(vorrangig im LINUX-Umfeld)



Tabellen und Abfragen mittels SQL (6)

SQL – Syntax und Funktionalität

Grundform einer Abfrage:

SELECT "Spalten_Name" FROM "Tabellen_Name" WHERE "Bedingung“

Beispiel mit Tabelle, die auch für weiter SQL-Abfragen benutzt wird:

ID	Stadt	Bundesland	Einwohner	Vorwahl	KFZ-Kennzeichen	Höhe
1	Dresden	Sachsen	523000	351	DD	113
2	Leipzig	Sachsen	522000	341	L	113
3	Chemnitz	Sachsen	243000	371	C	296
4	Zwickau	Sachsen	93750	375	Z	267
5	Erfurt	Thüringen	204000	361	EF	195
6	Jena	Thüringen	104000	3641	J	155
7	Gera	Thüringen	100000	365	G	205
8	Berlin	Berlin	3459000	30	B	34
9	Cottbus	Brandenburg	102000	355	CB	70
10	Postdam	Brandenburg	155000	331	P	35
11	FrankfurtOder	Brandenburg	61000	335	FF	40
12	Rostock	Mecklenburg-Vorpommern	201000	381	HRO	13
13	Schwerin	Mecklenburg-Vorpommern	95000	385	SN	38
14	Neubrandenb	Mecklenburg-Vorpommern	65000	395	NB	20

Tabellen und Abfragen mittels SQL (7)

Beispiel einer Abfrage:
SELECT Stadt, Bundesland FROM Staedte WHERE Einwohner>=100000

DISTINCT - Option:

Allgemeine Form:
SELECT DISTINCT "Spalten_Name“ FROM "Tabellen_Name"
gibt für die als DISTINCT markierte Spalte nur ein Tupel je Wert zurück

Beispiel:
SELECT DISTINCT Bundesland FROM Staedte;
Ergebnis:

mit DISTINCT-Option

Bundesland
Berlin
Brandenburg
Mecklenburg-Vorpommern
Sachsen
Thüringen

ohne DISTINCT-Option

Bundesland
Sachsen
Sachsen
Sachsen
Sachsen
Thüringen
Thüringen
Thüringen
….

Tabellen und Abfragen mittels SQL (8)

ORDER BY :

Allgemeine Form:

SELECT "Spalten_Name“ FROM "Tabellen_Name“ [WHERE "Bedingung"]
ORDER BY "Spalten_Name" [ASC, DESC]

Ordnet die Ausgabe nach Werten in der angebenen Tabellenspalte, auf Anwendungsseite kann das Sortierverfahren eingespart werden.

Beispiel:

SELECT Stadt, Höhe FROM Staedte
ORDER BY Höhe;

Ergebnis rechts …

Stadt	Höhe
Rostock	13
Neubrandenb	20
Berlin	34
Postdam	35
Schwerin	38
FrankfurtOder	40
Cottbus	70
Leipzig	113
Dresden	113
Jena	155
Erfurt	195
Gera	205
Zwickau	267
Chemnitz	296

Tabellen und Abfragen mittels SQL (9)

COUNT - Funktion:

Allgemeine Form:
SELECT COUNT("Spalten_Name") FROM "Tabellen_Name“

- gibt die Anzahl verschiedener Werte in der angegebenen Spalte zurück.
- wird oft zum Zählen der Datensätze benutzt, die einem bestimmten Kriterium entsprechen.

Beispiel:
Ergebnis: 14 … weil *SELECT Count(Bundesland) FROM Staedte*
14 Zeilen im Abfrageergebnis ernthält

SELECT COUNT (DISTINCT Bundesland) FROM Staedte
Ergebnis: 5 … da 5 nur verschiedene Zeilen
(funktioniert leider nicht mit ACCESS, aber mit SQL-SERVER)

Tabellen und Abfragen mittels SQL (10)

Allgemeines Konzept in SQL:
Aggregation von Spaltenwerten:

COUNT – Zählen der Werte
Beispiel: *SELECT COUNT (DISTINCT Bundesland) FROM Staedte*

SUM – Summieren
Beispiel: *SELECT SUM(Einwohner) FROM Staedte*

MIN,MAX – Minimum bzw. Maximum
Beispiel: *SELECT MAX(Einwohner) FROM Staedte*

AVG – Arithmetischer Mittelwert
Beispiel: *SELECT AVG(Einwohner) FROM Staedte*

Tabellen und Abfragen mittels SQL (11)

Group By

Allgemeines Beispiel:

SELECT "Spalten_Name1", SUM("Spalten_Name2")
FROM "Tabellen_Name" GROUP BY "Spalten_Name1„

- GROUP BY gruppiert die Ausgabe entsprechend gleicher Ausgaben in einer Spalte. Auf andere Spalten kann dann eine Aggregationsoperation ausgeführt werden.
- Die Gruppierung erfolgt immer nach der Spalte, die nicht in die Aggregationsoperation einbegriffen ist.

Beispiel:

SELECT Bundesland, COUNT(Stadt) FROM Staedte
GROUP BY Bundesland;

Tabellen und Abfragen mittels SQL (12)

Group By - Beispiel:

SELECT Bundesland, COUNT(Stadt) FROM Staedte GROUP BY Bundesland

Ergebnis:

Bundesland	Expr1001
Berlin	1
Brandenburg	3
Mecklenburg-Vorpommern	3
Sachsen	4
Thüringen	3

Ein weiteres Beispiel:

SELECT Bundesland, SUM(Einwohner) FROM Staedte GROUP BY Bundesland

Bundesland	Expr1001
Berlin	3459000
Brandenburg	318000
Mecklenburg-Vorpommern	361000
Sachsen	1381750
Thüringen	408000

Tabellen und Abfragen mittels SQL (12)

Having

Allgemeine Form:

SELECT "Spalten_Name1", SUM("Spalten_Name2")
FROM "Tabellen_Name„ GROUP BY "Spalten_Name1"
HAVING (arithmetische Funktionsbedingung)

mit der HAVING Klausel kann man Tupel über Bedingungen auswählen, welche die aggregierte Information betreffen

Beispiel:

SELECT Bundesland, SUM(Einwohner) FROM Staedte GROUP BY Bundesland HAVING (SUM(Einwohner) >400000)

Bundesland	Expr1001
Berlin	3459000
Sachsen	1381750
Thüringen	408000

Tabellen und Abfragen mittels SQL (13)

Tabellen JOINS

Allgemeines Beispiel:

SELECT "Spalten_Name1", “Spalten_Name2“ FROM "Tabellen_Name1“, “Tabellen_Name2“ WHERE “JoinBedingung“

Die relationenalgebraischen Konzepte der JOIN-Operation wurden bereits unter Relationenalgebra behandelt.

In Access gibt man das Komma zwischen den Tabellen anstelle des JOIN Schlüsselworts an.

Alias-Namen: Wird aus auf Spalten mit gleicher Bezeichnung aus verschiedenen Tabellen Bezug genommen, so ist für die Tabellen ein Alias-Name in der Anfrage anzugeben.

Beispiel mit Alias:

SELECT s1.Stadt, s2.Stadt FROM Staedte AS s1, Bahnstrecken, Staedte AS s2 WHERE s1.Stadt=Von AND s2.Stadt=Nach

(Tabelle Bahnstrecken wird auf Folie 94 eingeführt)

Tabellen und Abfragen mittels SQL (14)

Unterabfragen

Allgemeines Beispiel:

SELECT "Spalten_Name1", "Spalten_Name2"
FROM "Tabellen_Name1“ WHERE Spalten_Name2 IN (SELECT “Spalten_Name3“ FROM "Tabellen_Name2“ WHERE Bedingung)

Beispiel:
SELECT Bundesland, SUM(Einwohner) FROM Staedte WHERE Stadt in (SELECT Stadt FROM Staedte WHERE Höhe>100) GROUP BY Bundesland

Ergebnis:

Bundesland	Expr1001
Sachsen	1381750
Thüringen	408000

Hier werden nur jene Städte ausgewertet, die höher als 100 Meter liegen.

Tabellen und Abfragen mittels SQL (15)

Aggregation unterschiedlicher Werte innerhalb der Tupel:

Innerhalb einer Tabelle können Spaltenwerte aus anderen berechnet werden.

Beispiel:

Gehaelter

Mnr	MName	Basisgehalt	Zusatzgehalt
1	Herr Meier	2000	599
2	Frau Weber	1899	700
3	Herr Schmidt	1500	1000
4	Frau Lehmann	2380	125

SELECT MName, Basisgehalt+Zusatzgehalt AS Gesamtgehalt FROM Gehaelter;

SELECT MName, (Basisgehalt+Zusatzgehalt)/2 AS Mittelgehalt FROM Gehaelter;

Tabellen und Abfragen mittels SQL (16)

Aggregation unterschiedlicher Werte innerhalb der Tupel:

Auch innerhalb einer Abfrage über mehrere Tabellen können Spaltenwerte aus anderen berechnet werden.

Beispiel:

Streckenpreise

Von	Nach	Preis
Von	Nach	
A	B	1,99
B	C	2,5
C	D	1,3

SELECT A.Von, B.Nach, A.Preis + B.Preis AS Preis
FROM Streckenpreise AS A, Streckenpreise AS B
WHERE A.Nach=B.Von;

ergibt

Streckenpreise | Abfrage_Streckenpreise

Von	Nach	Preis
A	C	4,49
B	D	3,8

Views (1)

Ein „View“ ist eine virtuelle Relation, die durch eine SQL-Abfrage erstellt wird. Ist der View definiert, so kann der View wir eine Tabelle angesehen werden und in neue SQL-Abfragen eingeschlossen werden.

Allgemeines Beispiel:

CREATE VIEW “View_name“ AS SELECT “Spalten_name1“, “Spalten_name2“ FROM "Tabellen_Name“ WHERE Bedingung

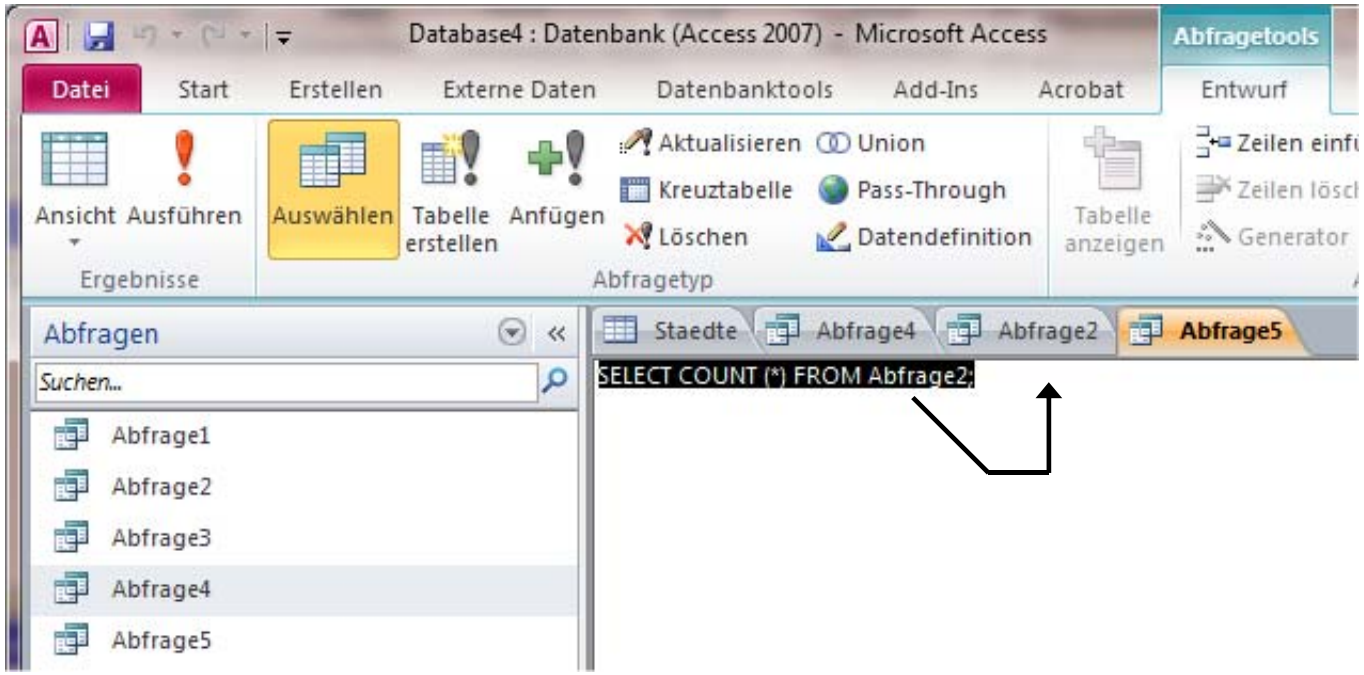
Beispiel:

CREATE VIEW Laender AS SELECT DISTINCT Bundesland FROM Staedte;
SELECT COUNT (*) FROM Laender;

Views (2)

Ein „View“ kann innerhalb der Access-Bedienoberfläche als Abfrage abgespeichert werden.

Es ist möglich, neue Abfragen auf bestehenden, gespeicherten Abfragen zu definieren.



Tabellen-Manipulation mittels SQL (1)

SQL – Erzeugen neuer Tabellen: Create Table

Allgemeine Form:

CREATE TABLE "Tabellen_Name"
("Spalte1" "Datentyp_für_Spalte1", "Spalte2" "Datentyp_für_Spalte2",
...)

Beispiel:

CREATE TABLE Bahnstrecke (Von CHAR, Nach CHAR, Distanz INTEGER)

Einfügen neuer Tupel mittels Insert Into:

Allgemeine Form:

INSERT INTO "Tabellen_Name" ("Spalte1", "Spalte2", ...)
VALUES ("Wert1", "Wert2", ...)

Allgemeine Form:

INSERT INTO Bahnstrecke (Von, Nach, Distanz)
VALUES (‘Dresden‘, ‘Berlin‘, 185)

Tabellen-Manipulation mittels SQL (8)

SQL – Ändern einzelner Werte: Update

Allgemeine Form:
UPDATE "Tabellen_Name" SET "Spalte1" = [Wert]
WHERE {Bedingung}

Beispiel:
UPDATE Staedte SET Einwohner=524000
WHERE Stadt='Dresden'

Es ist immer eine Selektion eines (oder mehrerer) Tupel vorzunehmen, die geändert werden sollen.

Tabellen-Manipulation mittels SQL (8)

SQL – Löschen einzelner Tupel einer Tabelle: Delete From

Allgemeine Form:
DELETE FROM "Tabellen_Name“ WHERE “Bedingung“

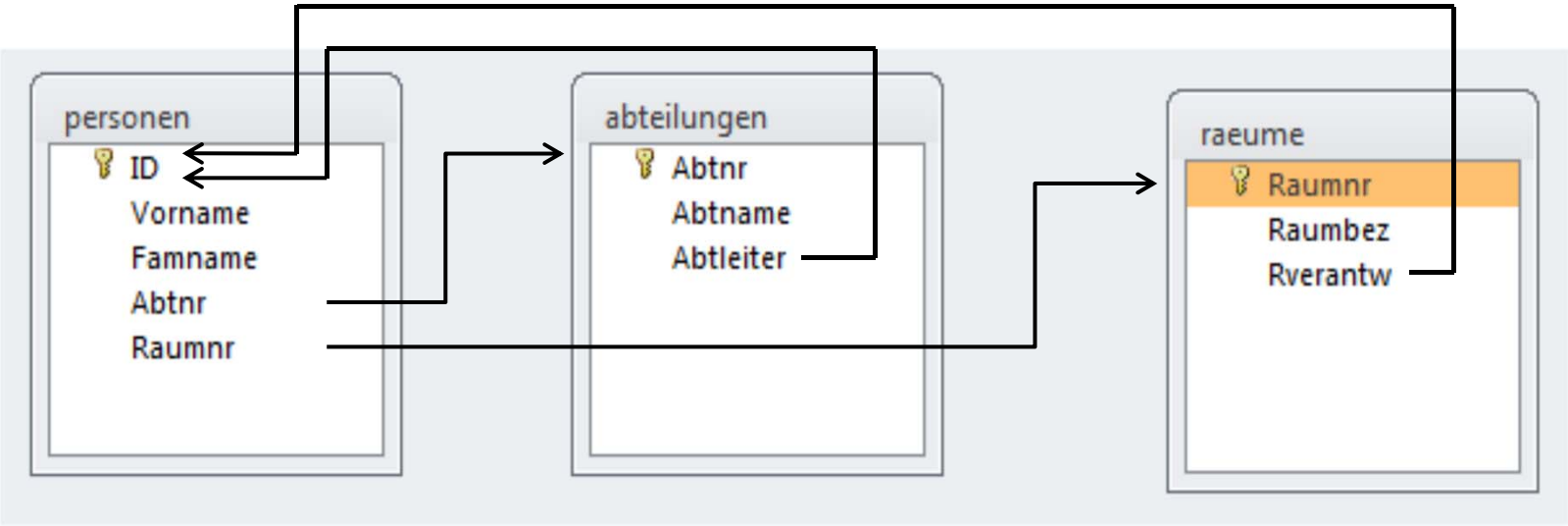
Beispiel:
DELETE FROM Bahnstrecke WHERE Von=“Cottbus‘
AND Nach=‘Dresden‘

Löschen einer Tabelle oder eines Views: Drop bzw. Truncate
Allgemeine Form:
DROP TABLE "Tabellen_Name"
Wird von Access nicht unterstützt, DELETE stattdessen

TRUNCATE TABLE "Tabellen_Name“
Nur die Daten (Tupel) werden gelöscht. Das Tabellenformat bleibt bestehen.

Weitere SQL-Beispiele (1)

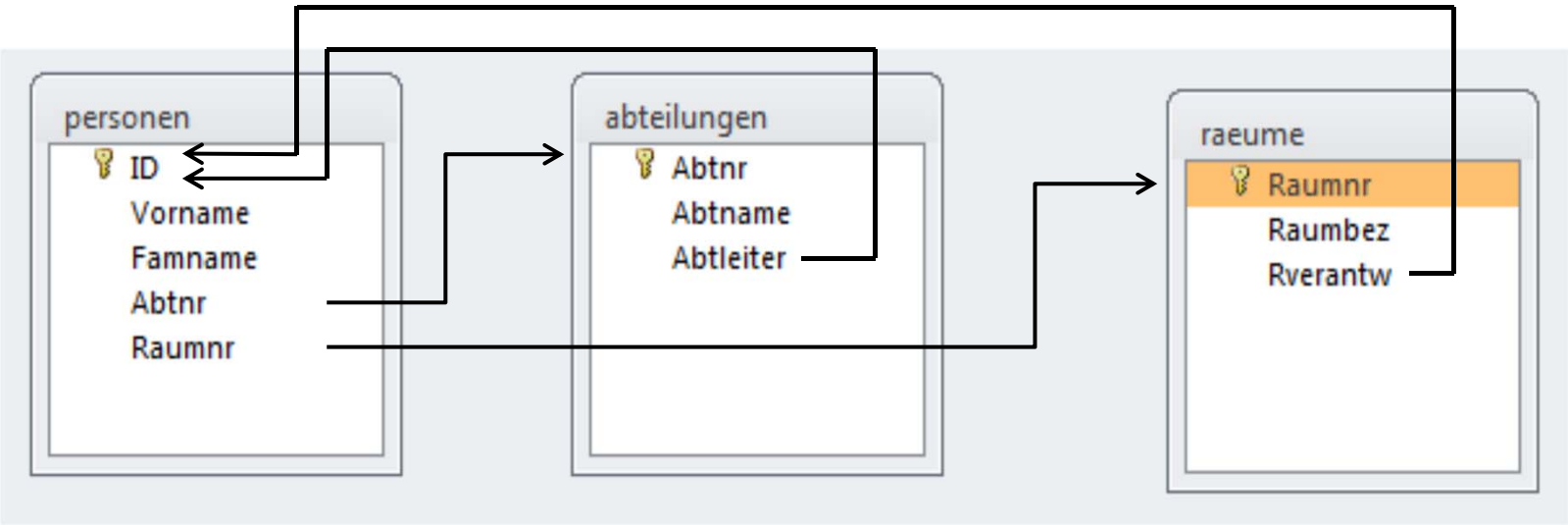
Ausgangsbasis ist die Datenbank firma.mdb.



Frage: Anzahl der Personen, die in ihren Räumen selbst Raumverantwortliche sind?

SELECT Count(ID) FROM personen AS p, raeume AS r
WHERE p.Raumnr = r.Raumnr AND ID=Rverantw

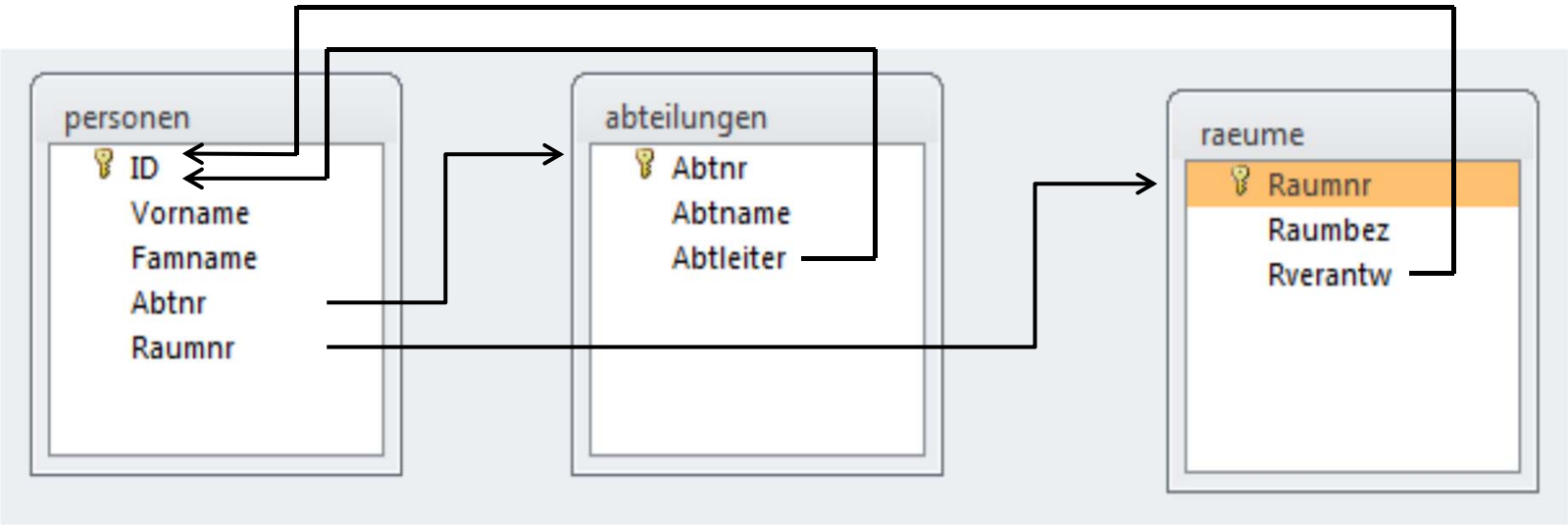
Weitere SQL-Beispiele (2)



Frage: Alle Abteilungsleiter und die jeweilige Anzahl der unterstellten Personen.

SELECT Abtleiter, COUNT(ID) FROM abteilungen AS a , personen AS p WHERE a.Abtnr = p.Abtnr
AND a.Abtleiter <>p.ID GROUP BY Abtleiter

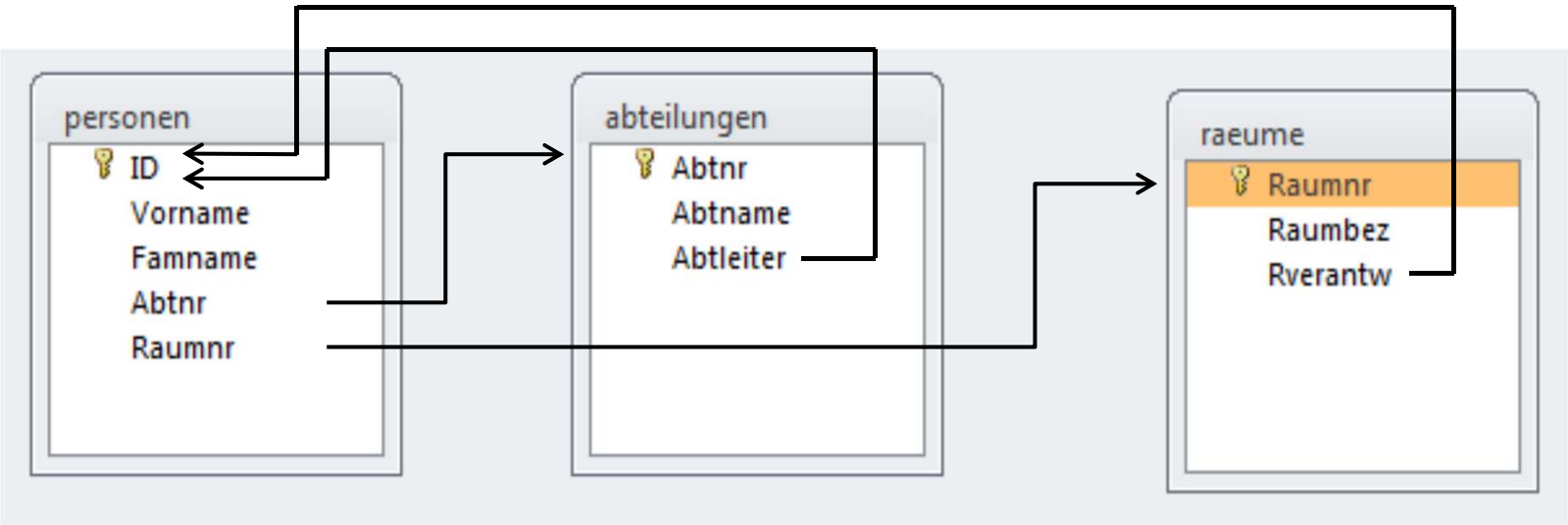
Weitere SQL-Beispiele (3)



Frage: Alle Abteilungsleiter, die mindestens eine andere Person in ihrer Abteilung leiten.

SELECT Vorname, Famname FROM personen WHERE ID IN (SELECT DISTINCT Abtleiter FROM abteilungen AS a , personen AS p WHERE a.Abtnr = p.Abtnr AND a.Abtleiter <>p.ID)

Weitere SQL-Beispiele (4)



Frage:
Maximale Anzahl Personen, die in einem Raum arbeiten.

Erster Schritt … Anzahl von Personen gruppiert nach Raumnr
SELECT Count (ID) FROM personen GROUP BY Raumnr

Zweiter Schritt …erste Abfrage als Unterfrage
SELECT Max(Count (ID)) FROM (SELECT Count (ID) FROM personen GROUP BY Raumnr)

Weitere SQL-Beispiele (5)

Zweiter Schritt …erste Abfrage als Unterfrage

SELECT Max(Count (ID)) FROM (SELECT Count (ID) FROM personen GROUP BY Raumnr)

… mit MS-Access wird angezeigt:

Fehler: Aggregatfunktion im Ausdruck (Max(COUNT(ID))) nicht möglich

Lösung über Aliasnamen:

SELECT Max(x) FROM (SELECT Count(ID) AS x FROM personen GROUP BY Raumnr)

Forms-Anwendung mit Suche in Datenbank (1)

Basis:

- Visual-Basic Forms Anwendung
- Öffnen eines Recordsets mit SQL-Anfrage
- SQL-Anfrage wird im Programm aus den eingegebenen Daten zusammengestellt

Button-Click-Prozedur für die Suche:

```
Private Sub Suche_Click(…)   Handles Suche.Click
    Dim vorname As String
    Dim famname As String
    Dim SQLSuchstring As String
    vorname = SuchVornameBox.Text
    famname = SuchFamnameBox.Text

    SQLSuchstring = "SELECT Vorname, Famname, Abtname, Raumnr FROM personen,
                     abteilungen WHERE personen.Abtnr=abteilungen.Abtnr AND
                      Vorname='" + Format(vorname) + "' AND Famname='" +
                      Format(famname) + "'")

    Search_in_Database("C:\\TEMP\\firma.mdb", SQLSuchstring)
End Sub
```

Search_in_Database ist eine Prozedur, die auf der nächsten Seite folgt

Forms-Anwendung mit Suche in Datenbank

Anstelle der Anzeige mehrerer Tupel wird nur ein einziges durch die SQL-Anfrage ausgewählt, falls die gesuchten Daten vorhanden sind.

Prozedur *Search_in_Database*:

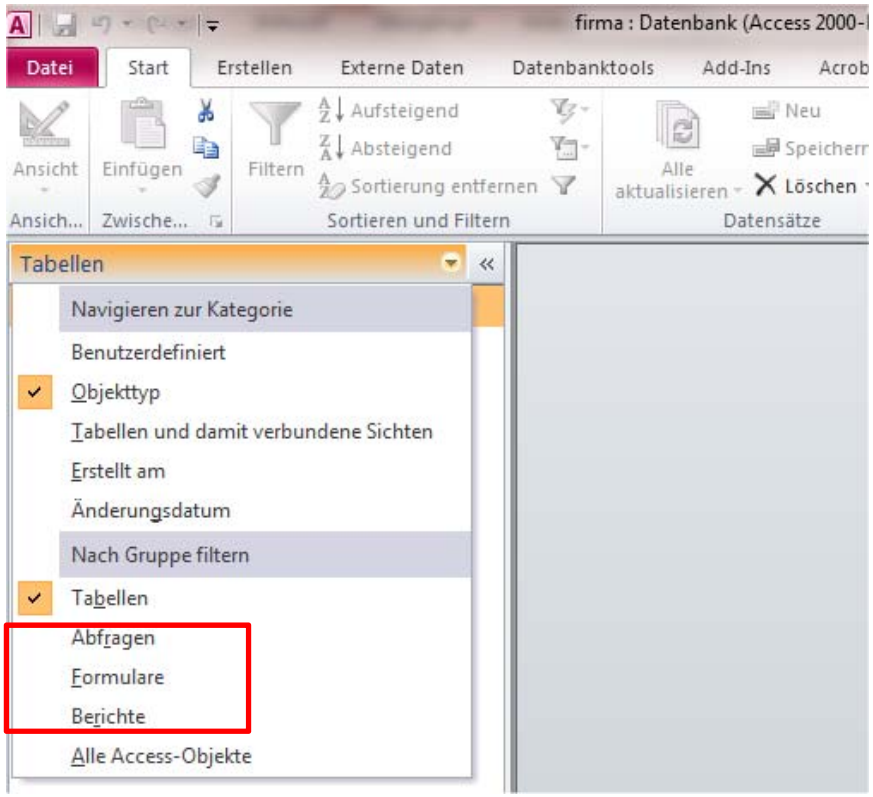
```
Private Sub Search_in_Database(ByVal database_path As String,
                               ByVal sql As String)
    Dim str As String =""
    dbe = New dao.DBEngine()
    db = dbe.OpenDatabase(database_path)
    rs = db.OpenRecordset(sql)
    If rs.EOF Then
      MsgBox("Nichts gefunden!")
    Else
      rs.MoveFirst()
      For i = 0 To rs.Fields.Count - 1
        str = str + Format(rs.Fields(i).Value) + vbTab
      Next
      MsgBox(str)
    End If
  End Sub
```

Weitere Datenbank-Funktionen

Neben Tabellen (Formate und Daten) speichert eine Access-Datenbank

- Abfragen
  (in SQL als Views gehandhabt)
- Formulare zur Dateneingabe und Datenansicht
- Berichte (Reports)

Diese “Access-Objekte“ werden in die Speicherung der Datenbank einbezogen und können immer wieder benutzt werden.



Datenbank – Formulare (1)

Formulare dienen der nutzergerechten Eingabe und Ansicht der Daten

Vorteile:

- bessere Anordnung der Felder
- Bezeichungen müssen nicht mehr den Attributnamen entsprechen, sondern können vollständige Namen benutzen
  (z.B. Famname -> Familienname)
- Auto-Werte (IDs, die automatisch hochgezählt werden) können versteckt werden.
- Suche in Tabellen möglich
- Neue Felder können berechnete Inhalte aus den Datenbankfeldern wiedergeben

Datenbank – Formulare (2)

Mehrere Formular-Varianten möglich:

- Einspaltig
- Tabellarisch
- Datenblatt
- Blockdarstellung

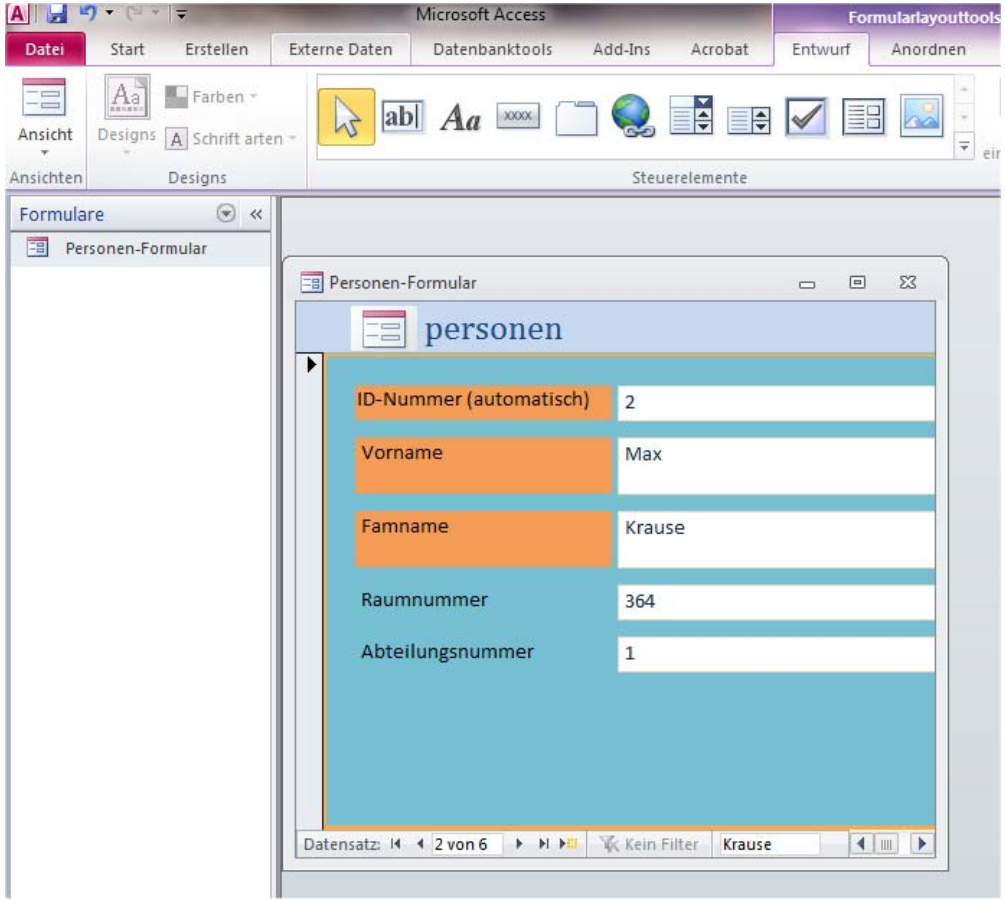
In Access-Bedienung unter

- Erstellen -> Formular
- Erstellen -> Formularassistent

Jedes Formular kann in der Entwurfsansicht angezeigt werden. Dann sind Layout-Änderungen möglich.

Datenbank – Formulare (2)

Beispiel für ein einspaltig erstelltes Formular zur Tabelle personen.

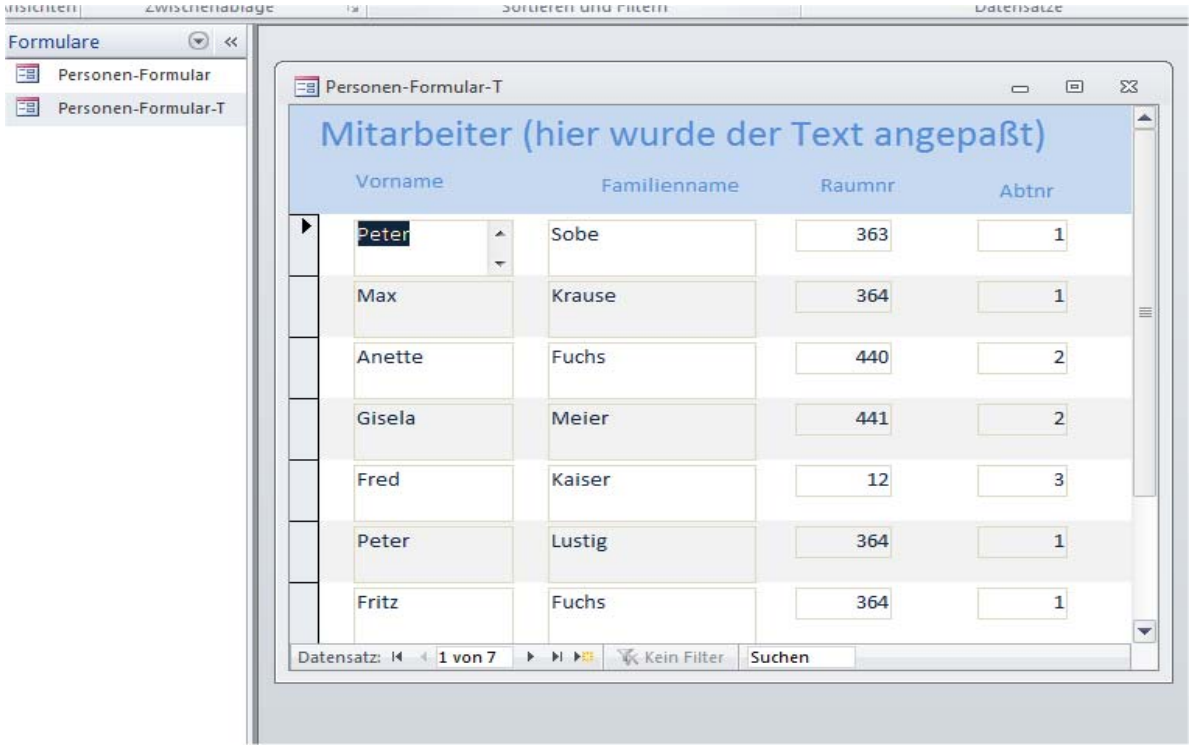


Im Formular kann durch die Datensätze (Tupel) mittels Vor- und Zurück-Tasten gewandert werden.

Suchfunktion

Datenbank – Formulare (3)

Beispiel für ein Formular zur Tabelle personen in tabellarischer Form



Datenbank – Formulare (4)

Beispiel für ein Formular zur Tabelle personen in Datenblatt-Ansicht

personen

Vor- und Zuname	Famname	Raumnr	Abtnr
Peter	Sobe	363	1
Max	Krause	364	1
Anette	Fuchs	440	2
Gisela	Meier	441	2
Fred	Kaiser	12	3
Peter	Lustig		
Fritz	Fuchs		

Datensatz: 1 von 7 Kein Filter Suche

Ansicht kann gewechselt werden

personen

Mitarbeiter

Vor- und Zuname Fritz Fuchs

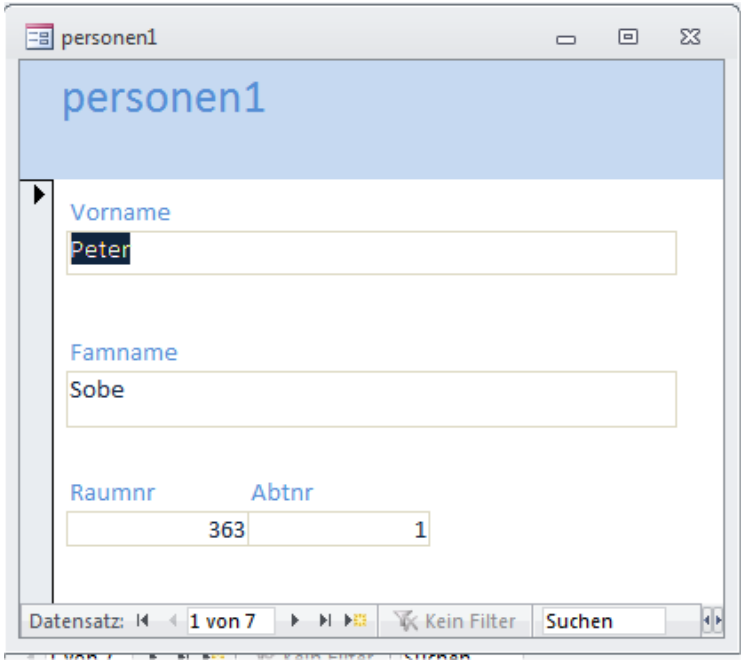
Raumnr 364

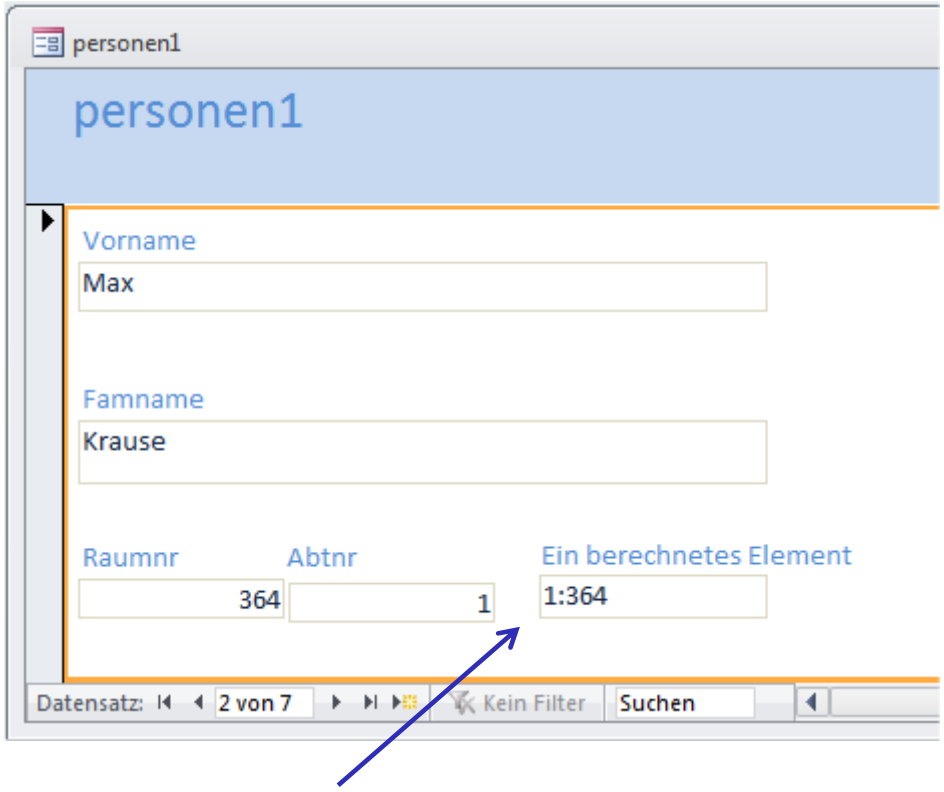
Abtnr 1

Datensatz: 7 von 7 Kein Filter Fuchs

Datenbank – Formulare (5)

Beispiel für ein Formular zur Tabelle personen in Blocksatz-Ansicht





Berechnetes Element:

=Format([Abtnr])+":"+Format([Raumnr])

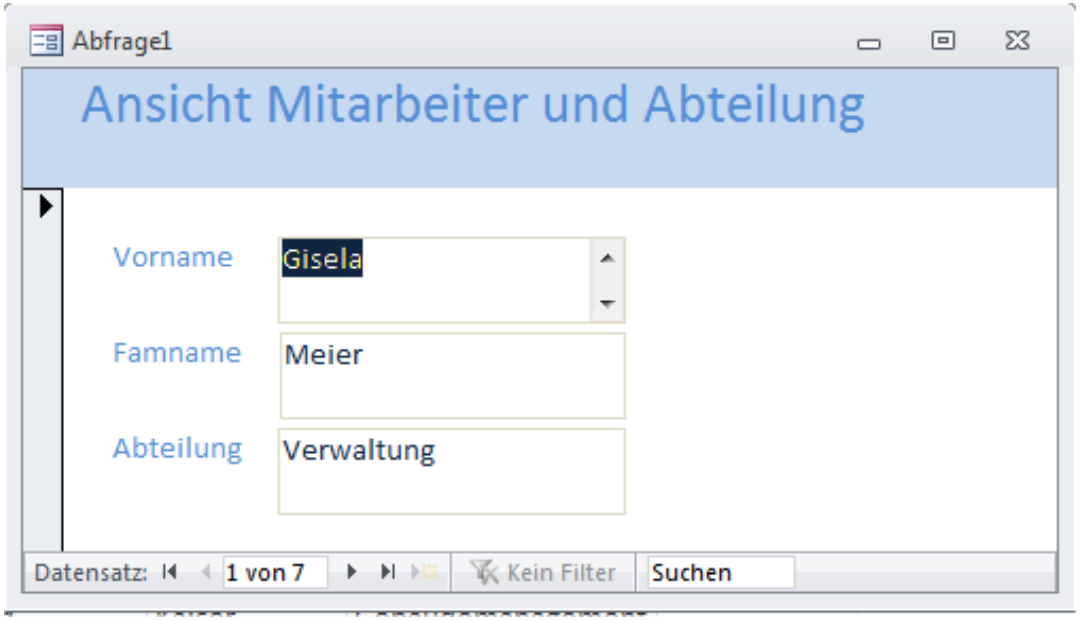
Datenbank – Formulare (6)

Beispiel für ein Formular zu einer Abfrage:

Abfrage1 in Access mit der Datenbank zusammen abgespeichert:
SELECT personen.Vorname, personen.Famname, Abtname FROM personen, abteilungen WHERE personen.Abtnr = abteilungen.Abtnr;

Formular kann auf Basis der Abfrage1 erstellt werden.

Sinnvollerweise ist kein Einfügen neuer Datensätze möglich



Datenbank – Reports (Berichte)

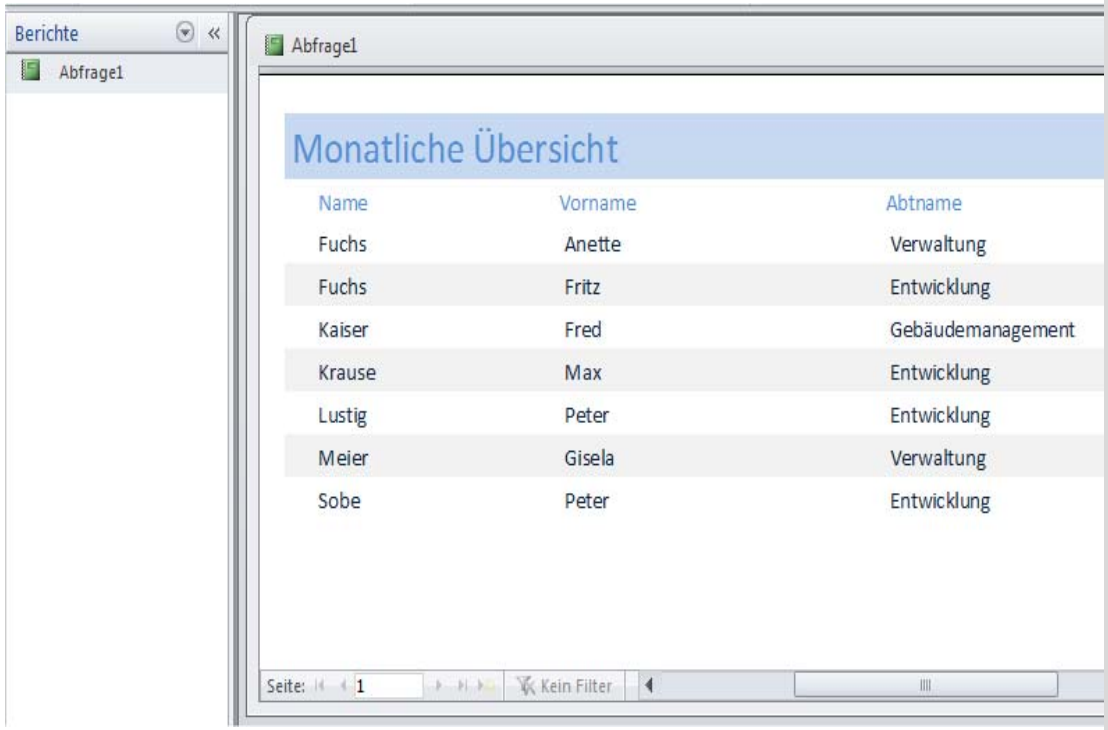
Reports sind formatierbare Ausdrucke ausgehend von Datenbankinhalten.

Reports können erstellt werden und als “Objekte“ mit der Datenbank gespeichert werden. Ein neuer Aufruf eines Reports spiegelt Änderungen in der Datenbank wieder.

Formatierung kann so erfolgen, dass Listenübersichten, Etikettendrucke oder Grafiken entstehen.

Datenbank – Reports (Berichte)

Beispiel: Bericht aus Abfrage 1
(*SELECT personen.Vorname, personen.Famname, Abtname FROM personen, abteilungen WHERE personen.Abtnr = abteilungen.Abtnr;)*



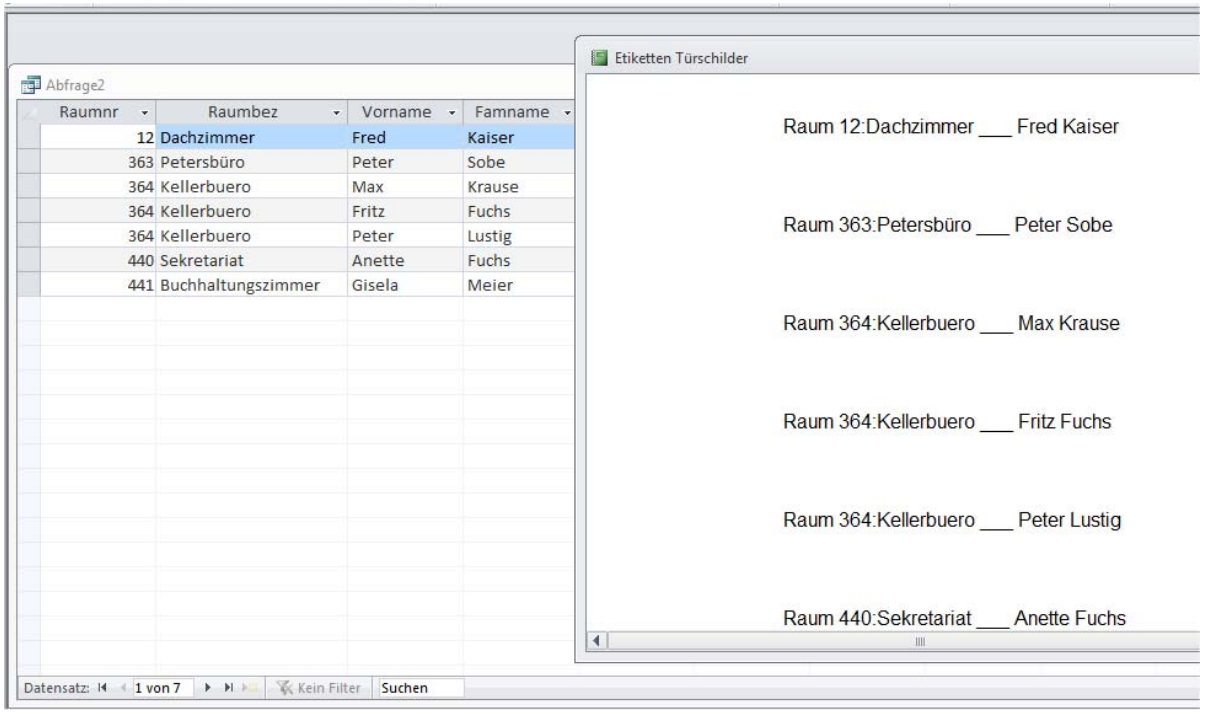
Nach Erstellen des Berichts mit dem Berichtsassistenten:

- Abfrage1 erscheint unter „Berichte“
- Abfrage „Abfrage1“ hat jetzt eine Berichtsansicht

Datenbank – Reports (Berichte)

Beispiel: Bericht aus Abfrage 2 für Etikettendruck

(*SELECT raeume.Raumnr, Raumbez, Vorname, Famname*
FROM raeume, personen
WHERE raeume.Raumnr = personen.Raumnr
ORDER BY raeume.Raumnr;)



Architektur rechnergestützter Anwendungen

Typische 3-Schichten-Architektur

Präsentation	Konsole, Windows-bzw. UNIX/X-Benutzeroberfläche (z.B. FORMS) oder Webformular
Funktion	mathematische Berechnungen, Simulationsverfahren, Such- und Auswahlalgorithmen, Logik- und Wissensverarbeitung
Datenverwaltung	Laden und Speichern des Anwendungskontexts und der Daten - Dateisystemzugriffe - Datenbanken