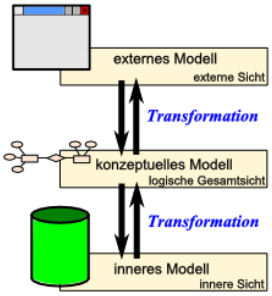
Informatik

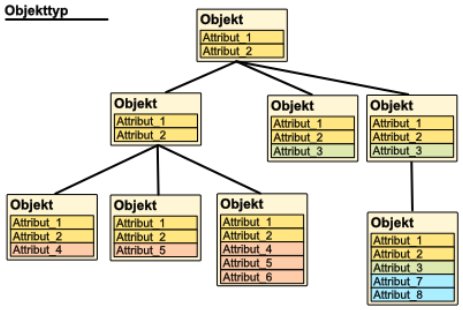
für die Sekundarstufe II

- Datenbanken -

Teil 2: Implementierungen

Autor: L. Drews

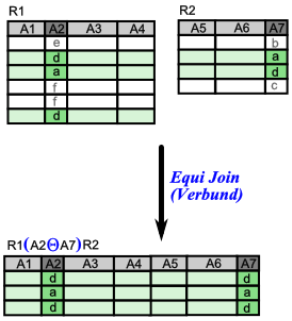




ACCESS

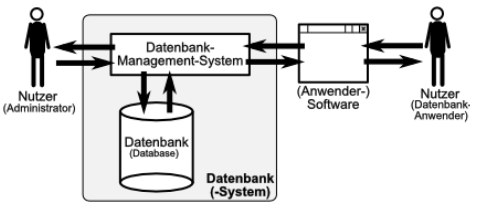
AND | OR | XOR

BASE



Wie jagt ein typischer Programmierer afrikanische Elefanten?

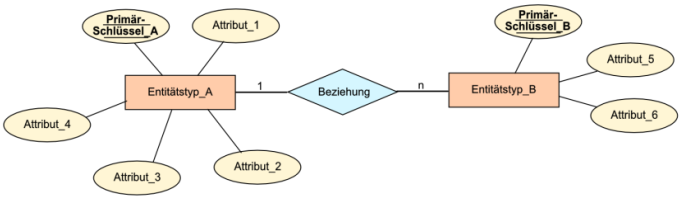
```
{
Gehe nach Afrika
Beginne am Kap der guten Hoffnung
Durchkreuze Afrika von Süden nach Nor-
den bidirektional in Ost-West-Richtung
Für jedes Durchkreuzen tue
   {
   Fange jedes_Tier, das Du siehst
   Vergleiche jedes_Tier mit Elefant
   Halte an bei übereinstimmung
   }
}
```



SQL
NoSQL

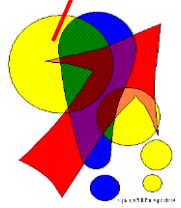
Wie jagt ein SQL-ler afrikanische Elefanten?

```
SELECT elefant
FROM afrika
```



V. 0.12b (2024)

Legende:
mit diesem Symbol werden zusätzliche Hinweise, Tips und weiterführende Ideen gekennzeichnet



Nutzungsbestimmungen / Bemerkungen zur Verwendung durch Dritte:

(1) Dieses Skript (Werk) ist zur freien Nutzung in der angebotenen Form durch den Anbieter (lern-soft-projekt) bereitgestellt. Es kann unter Angabe der Quelle und / oder des Verfassers gedruckt, vervielfältigt oder in elektronischer Form veröffentlicht werden.

(2) Das Weglassen von Abschnitten oder Teilen (z.B. Aufgaben und Lösungen) in Teildrucken ist möglich und sinnvoll (Konzentration auf die eigenen Unterrichtsziele, -inhalte und -methoden). Bei angemessen großen Auszügen gehört das vollständige Inhaltsverzeichnis und die Angabe einer Bezugsquelle für das Originalwerk zum Pflichtteil.

(3) Ein Verkauf in jedweder Form ist ausgeschlossen. Der Aufwand für Kopierleistungen, Datenträger oder den (einfachen) Download usw. ist davon unberührt.

(4) Änderungswünsche werden gerne entgegen genommen. Ergänzungen, Arbeitsblätter, Aufgaben und Lösungen mit eigener Autorenschaft sind möglich und werden bei konzeptioneller Passung eingearbeitet. Die Teile sind entsprechend der Autorenschaft zu kennzeichnen. Jedes Teil behält die Urheberrechte seiner Autorenschaft bei.

(5) Zusammenstellungen, die von diesem Skript - über Zitate hinausgehende - Bestandteile enthalten, müssen verpflichtend wieder gleichwertigen Nutzungsbestimmungen unterliegen.

(6) Diese Nutzungsbestimmungen gehören zu diesem Werk.

(7) Der Autor behält sich das Recht vor, diese Bestimmungen zu ändern.

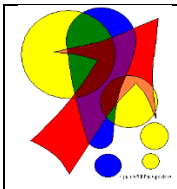
(8) Andere Urheberrechte bleiben von diesen Bestimmungen unberührt.

Rechte Anderer:

Viele der verwendeten Bilder unterliegen verschiedensten freien Lizenzen. Nach meinen Recherchen sollten alle genutzten Bilder zu einer der nachfolgenden freien Lizenzen gehören. Unabhängig von den Vorgaben der einzelnen Lizenzen sind zu jedem extern entstandenen Objekt die Quelle, und wenn bekannt, der Autor / Rechteinhaber angegeben.

public domain (pd)	Zum Gemeingut erklärte Graphiken oder Fotos (u.a.). Viele der verwendeten Bilder entstammen Webseiten / Quellen US-amerikanischer Einrichtungen, die im Regierungsauftrag mit öffentlichen Mitteln finanziert wurden und darüber rechtlich (USA) zum Gemeingut wurden. Andere kreative Leistungen wurden ohne Einschränkungen von den Urhebern freigegeben.
gnu free document licence (GFDL; gnu fdl)	
creative commens (cc) creative commons	BY: od. neu … Namensnennung … nichtkommerziell … in der gleichen Form … unter gleichen Bedingungen

Die meisten verwendeten Lizenzen schließen eine kommerzielle (Weiter-)Nutzung aus!



Bemerkungen zur Rechtschreibung:

Dieses Skript folgt nicht zwangsläufig der neuen **ODER** alten deutschen Rechtschreibung. Vielmehr wird vom Recht auf künstlerische Freiheit, der Freiheit der Sprache und von der Autokorrektur des Textverarbeitungsprogramms microsoft ® WORD ® Gebrauch gemacht.

Für Hinweise auf echte Fehler ist der Autor immer dankbar.

Inhaltsverzeichnis:

Seite

0. Einleitung

Skript enthält diverse Wiederholungen. Das ist zum Einen der Entstehungs-Geschichte(n) geschuldet und zum Anderen dient es dem flexiblen Einsatz-Konzept. Die Erfahrung sagt auch, das Wiederholungen selten schaden.

Verweise auf andere Skripte dieser Reihe 🕮 **Rechner, Netzwerke und Protokolle**

Eine weitere Möglichkeit zur Erschließung des Thema's "Datenbanken" ist ein Kurs beim Portal OpenHPI (→) des Hasso PLANTNER-Instituts. Der Kurs ist 2013 gelaufen und steht derzeit immer noch als Lese-Version zur Verfügung. D.h. Sie können den Kurs benutzen, die Video's anschauen und die Quize lösen. Leider gibt es keinen Zugriff mehr auf die Hausaufgaben und die abschließende Testung. Aber das wird in Ihrem Kurs vielleicht auch vom Kursleiter organisiert und in vielleicht auch in einer Klausur enden.
Auch die Lern-Plattform AppCamps (→) bietet einen SQL-Kurs an. Dieser beschränkt sich aber eben auch vorrangig auf SQL. Wem es also nur um diese Datenbank-Sprache geht, kann auch dort mal vorbeischauen. Eine Absprache mit dem Kursleiter ist auch hier angebracht.
Unter der Adresse (→) findet man einen weiteren Datenbank / SQL-Kurs. Dieser ist primär in englisch, wird aber auch übersetzt angeboten. Da es sich hierbei um eine automatische Übersetzung handelt, sollte man sich nicht so sehr auf Ausdruck und Grammatik fixieren. Inhaltlich ist der Kurs anspruchsvoll.
Jeder möge den für sich am besten geeigneten Weg nutzen, um seine Bildungs-Ziele zu erreichen. Dieses Skript bietet einen breiten, weit gefächerten Zugang zum Thema. Dabei sind die vielen Wiederholungen und Anspielungen oder Verweise als Hilfestellung zu verstehen. Wer es konzentriert und ohne stetige Wiederholungen braucht, der muss springend lesen oder zu einem anderen Material greifen.

Um dem unvorbereiteten Leser ein wenig Orientierung zu geben, was für ihn interessant bzw. notwendig ist, sind die Abschnitte mit Niveau-Kennungen versehen.
Entweder handelt es sich um Bereich von bis oder um Einzel-Festlegungen.
Die Niveau-Stufen sind von mir folgendermaßen gekennzeichnet:
Das Grundlagen-Niveau ist für alle Leser gedacht. Hier werden allgemeine Sachverhalte vorgestellt, die auch nicht immer zwangsläufig zur Informatik gehören.

Im Basis-Niveau betrachten wir die Dinge, die man einem Fach oder Grundkurs zuordnen würde. Ich orientiere mich dabei auch am Kern-Curriculum für Berlin, Brandenburg und Mecklenburg-Vorpommern. Ev. passt es genau so auch in anderen Bundesländern? Dabei können einzelne Themen mehr oder weniger ausgefüllt sein. Vieles liegt im Ermessen des Kurs-Leiters oder der Planung in der Schule.
Für die Leistungskurse oder das Hauptfach sind die Inhalte mit dem Fortgeschrittenen-Niveau. Auch hier gilt, dass ohne weiteres Gebiete aus dem untergeorneten Niveau schon die Grenze des Lehrplans sind, aber es könnten auch Themen des – von mir subjektiv eingestuften - Experten-Niveaus inhaltlich vorgeschrieben sein.

Der Kurs-Leiter sollte immer eine auf die Bedingungen und Anforderungen zugeschnittene Themen-Auswahl vornehmen. Einige deutlich weitergehende Themen sind ausschließlich als Experten-Niveau klassifiziert.

Wer das Skript als Wiederholung / Vorbereitung für das Studium benutzt, muss natürlich explizit auf die Anforderungen des Lehrstuhls achten. Sonst könnte es sein, dass man mit seinem Niveau deutlich unter den Forderungen liegt.
Experten-Themen eigenen sich auch für Projekte, Grob-Orientierungen für Schüler-Vorträge usw. usf. Wenn sie nicht interessieren, der überspringt sie einfach. Man muss nicht alles wissen, man muss nur wissen, wo es steht und wo man sich ev. wieder belesen kann.

Links:
http://home.f1.htw-berlin.de/scheibl/DB/index.htm
https://www.kstbb.de/informatik/rdb/index.html (sehr umfangreiche Schritt-für-Schritt-Aufarbeitung der Datenbank-Entwicklung)

nur für den internen Gebrauch!:

https://books.google.de/books?id=5HXM-lFIXXcC

Definition(en): Informatik
Informatik ist die Wissenschaft von der Verarbeitung von Informationen.

Definition(en): Information
Informationen sind Daten mit einer Bedeutung.

3. Datenbanken – Implementierung

Problem-Fragen für Selbstorganisiertes Lernen
Wie macht man ausgehend von einer Problem-Situation / einer allgemein formulierten Aufgabe eine passende Datenbank? Welche Entwicklungs-Schritte muss man zur sicheren Konzeption einer Datenbank unternehmen? Was muss man bei der Erstellung von Datenbanken unbedingt beachten? Gibt es dabei immer nur eine Lösung? Mit welchem Programm / Datenbanksystem / Datenbank-Management-System arbeitet man am Besten?

Voraussetzung ist konzeptionelles Modell

Welche Daten müssen erfasst werden? In welchen Beziehungen stehen sie zueinander?

Analyse: **Wer** braucht **Welche** Daten sollen **Wie** dargestellt werden und **Was** soll mit den Daten gemacht werden?
Welche zusätzlichen Daten sind notwendig (z.B. um Datenintegrität oder Datenschutz zu realisieren)?

Extension sind die Daten in den Tabellen (zum Datenbank-Entwurf)

verschiedene Modell-Typen (→ 2.1.1. Datenbank-Modelle) bekannt, die jeweils bestimmte Aspekte besser oder weniger gut beachten

Objekte mit ihren Eigenschaften
Beziehungen (Verbindungen) zwischen Objekten oder Beziehungen zu anderen Beziehungen

Menge der betrachteten / relevanten Objekte und Beziehungen → Miniwelt

3.1. Problem-Lösen – von der Realität zum Modell, vom Problem zur Lösung

Nachdem wir schon die verschiedenen Daten-Modelle besprochen haben und uns auf die relationellen Datenbanken konzentrieren wollen, kommen wir nun zur praktischen Umsetzung in eine Datenbank-Management-System (DBMS).
Als Modellierungs-Werkzeug haben wir die Entity-Relationship-Diagramme (ERD) gewählt.
Es folgt die Umsetzung von Sachverhalten aus der Realität oder ausgedachter Situationen in Datenbank-Anwendungen.

Vielleicht ist die Wahl des Begriffs Problem zu Anfang etwas überzogen. Bei Problemen liegt die Schwierigkeit im Vergleich zu einer sofort lösbaren Aufgabe darin, dass kein direkter Lösungs-Algorithmus vorliegt, sondern ein grobes Anleitungs-Schema umgesetzt werden muss. Sachverhalte orientieren sich im Wesentlichen am bisher Gelernten, den vorhandenen Fähigkeiten und Fertigkeiten.
Bei echten Problemen sind i.A. neue Wege und Lösungen gesucht. Hier kommen zur Lösung / besser Bearbeitung vor allem Heuristiken zur Anwendung.
Problem-Lösen ist aber in der Informatik ein gängiger Begriff für Aufgaben / Realisierungen. Die überhöhte Begriffs-Verwendung Bauch-pinselt eher unser Ego, wenn wir dann die Lösung gefunden haben.
Später und Praxis-orientiert werden wir es aber dann auch wirklich mit Problemen zu tun haben. Vor allem, wenn neuartige Umsetzungen von Anwendungs-Szenarien gefragt sind.

aus praktischer Sicht:

1. Daten-Aquise (Datenbeschaffung)	Sammlung aller Daten / Daten-Arten oder der Strukturen, die in der Datenbank gespeichert, … werden sollen
2. Daten-Analyse	Untersuchung der Daten auf Redundanzen (Mehrfachspeicherung), Strukturen und Muster
3. Daten-Modellierung	Erstellen eines abstrakten Schemas der Daten

Bei vorliegenden Aufgaben / Aufgabenstellungen ist es immer notwendig, eine gründliche Analyse vorzunehmen. Vielfach sind die Aufgaben-Steller (Auftraggeber) keine Informatiker oder Datenbank-Profi's – sie geben die Aufgabe so vor, wie ihnen der Schnabel gewachsen ist. Es ist also die erste Aufgabe, das eigentlliche Problem oder die Aufgabe an sich zu erkennen und sauber zu umreißen. In der Praxis bietet es sich an z.B. die Aufgabe zu paraphrasieren, also den Sachverhalt an den Aufgabensteller zurück zu erläutern (Wiedergeben des Gesagten mit eigenen Worten.). Das kann man z.B. so anfangen: "Wenn ich Sie recht verstanden habe, …. " oder "Gehe ich recht in der Annahme, dass …?" oder "Meinten Sie, ?".
Wenn der Auftraggeber dann zustimmt, hat man zumindestens eine gemeinsame Verständnis-Ebene gefunden. Ansonsten nähert man sich durch die erneuten Erläuterungen oder Richtigstellungen schrittweise einer gemeinsamen Betrachtung an.
Wichtig ist weiterhin, sich auf die Aufgabe zu konzentrieren. Man sollte mit eigenen Ideen, Interpretationen und Erweiterungen sehr sparsam umgehen. Erst die Aufgabe, wie sie vorgegen ist, lösen und dann, wenn dafür Zeit und Raum ist, die eigenen kreativen Ideen beitragen. Die Lösung der Aufgabe / des Problem's darf für sich natürlich kreativ sein.

Für zusätzlichen "Schnick-Schnack" bezahlt ein Auftraggeber nur, wenn seine Grundanforderungen (des sogenannten "Pflichtenheftes") erfüllt sind und er die Zusätze als wirklich Gewinn-bringend ansieht.

Schritte der praktischen Datenbank-Erstellung

1. Datenbank planen	
2. Datenbank-Datei erstellen	
3. Tabellen erstellen	
4. Formulare, Abfragen und Berichte hinzufügen	zuerst nur in Prototyp-Form

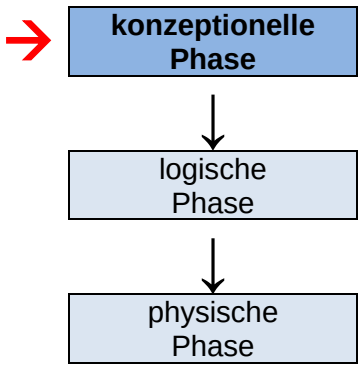
typische Datenbank-Anwendungen

- **Lager-Verwaltung**
- **Schul-Datenbank**
- **Bibliotheks-Verwaltung**
- **Warenwirtschaftssyteme**
- **Vereins-Verwaltung**
- **Lexika**
- **…**

3.1.1. konzeptionelle Phase – Grenzen ziehen

Problemwelt (Miniwelt, Modellwelt, …) als Ausschnitt der Realität, für die eine informatische Lösung gesucht wird
häufig separat als externe Phase (auch Spezifikations- und Anforderungs-Analyse) verstanden

- Beschreibung der Daten
- Festlegung / Erkundung des Informations-Bedarf's der Nutzer
- Strukurieren des Informations-Bedarf's
- …



→ informelle Beschreibung des Fach-Problems

Modell als eingeschränktes Abbild der Realität für bestimmte Einsatzzwecke
Grund-Eigenschaften eines Modells

- Abbildung
- Verkürzung / Vereinfachung
- Pragmatismus / Handhabarkeit

eigentliche konzeptionelle Phase
Ziel ist formale Beschreibung des ausgewählten Sachverhalts
gesucht ist ein semantisches Modell des Sachverhaltes
eine Möglichkeit der Darstellung ist das ER-Modell

→ Fachkonzept der Datenbank

Manchmal ist es einfacher zu beschreiben, was alles nicht zum Problem gehört. Sollte dabei etwas auftauchen, wass eigentlich in der Aufgabenstellung gefordert ist, muss passend korrigiert und abgegrenzt werden.
Erfahrungsgemäß laufen hier die meisten Modellierungen aus dem Ruder. Den Modellierern fällt es meist sehr schwer, sich an den harten Fakten und den Verfahren (Algorithmen) zu orientieren.

Klassifizierung der Real-Objekte in Klassen (Entitäts-Typen), Objekten (Entitäten), Beziehungen

Zusammensammeln der abzubildenden Elemente der Miniwelt und Einordnung in Datenbank-Kategorien

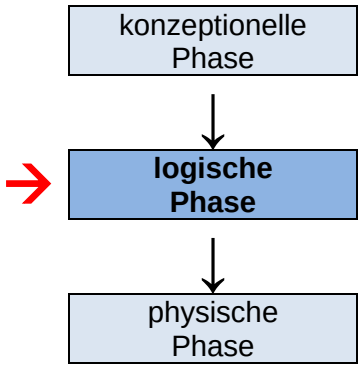
abzubildendes Teil / Element / … aus der Miniwelt	**allg. Kategorie**	**Datenbank-Kategorie**	**Bemerkungen / Hinweise**

Umsortieren und mit etwas Erfahrung auch gleich systematisch zusammengestellt

allg. Kategorie	**Datenbank-Kategorie**	**abzubildendes Teil / Element / … aus der Miniwelt**	**Bemerkungen / Hinweise**

3.1.2. logische Phase – die Realität abbilden – ein Modell erstellen

Beim Modellieren ist es wichtig, sich auf das Wesentliche zu konzentrieren. Für den Computer und für die Datenbank-Arbeit ist es z.B. bei einem Auto gleichwichtig, mit welcher Farbe, mit welcher Leistung (PS) und mit welchem Preis es daherkommt. Für uns Menschen sieht das sich anders aus. Während ich mich vielleicht sehr stark an der Anzahl der Pferdestärken orientiere, interessiert sich meine Frau vielleicht nur für die Farbe (um mal die Klischee's zu bedienen). Mein Sohn wird dagegen nur auf den Preis achten, weil er noch kein eigenes Geld verdient.



Erstellung des logischen Modell's

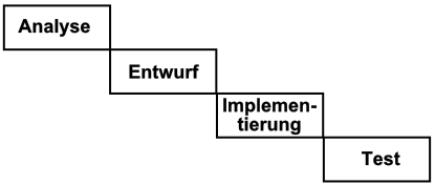
Entscheidung für ein Daten-Modell (z.B. Tabelle → Relationale Datenbank)
Transformation des ER-Modell in das Daten-Modell / Datenbank-Schema
Optimierung des Relationalen Schema's (/ der Daten-Strukturen) z.B. durch Normalisierungen

Prozess der Modell-Bildung

• **Abgrenzung**	Ausschluss / Ausgrenzung irrelevanter Objekte und Beziehungen
• **Reduktion**	Eingrenzung der Objekt-Merkmale / Weglassen nichtrelevanter Eigenschaften und Beziehungen
• **Dekomposition**	Zerlegung der übriggebliebenen Problemwelt (Modellwelt) in Segemente / einzelne Objekte und Beziehungen
• **Aggregation**	Zusammensetzen / Vereinigen der Segemente / Objekte und Beziehungen
• **Abstraktion**	Bildung von Begriffen und Klassen
• **Testung**	Prüfen, ob mit dem Modell die Realität angemessen abgebildet wird

Vorgehens-Modelle für die Entwicklung einer Datenbank

- **Wasserfall-Modell**

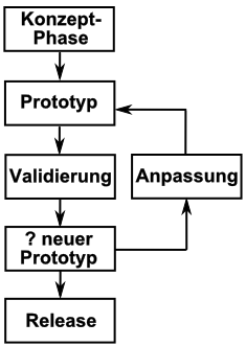


die Phasen Analyse, Entwurf, Implementierung und Test werden als sequenzielle Abfolge betrachtet
Phasen werden streng hintereinander abgearbeitet
es ist kein Rücksprung in die vorhergehende Phase vorgesehen (nur im erweiterten Wasserfall-Modell)
dieses Vorgehen ist dann möglich, wen die Anforderungen, Leistungen und Abläufe klar definiert wurden

Vorteile:
- saubere Phasen(-Grenzen) / Abrechnungs- / Kontroll-Punkte

Nachteile:
- Ergebnisse einer Phase erst sehr spät sichtbar
- wenig (Phasen-übergreifende) parallele Arbeiten möglich
- Entwicklung kann nicht interativ erfolgen
- wenige Richtlinien für die Projekt-Dokumentation

- **Rapid Prototyping**



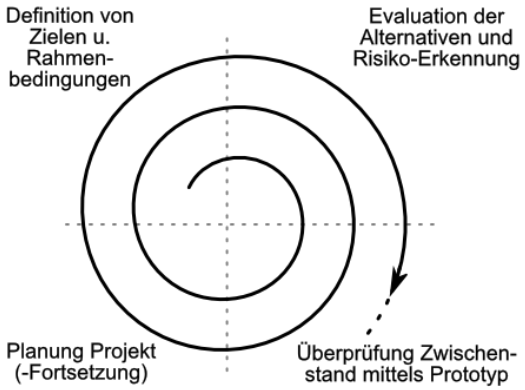
agiles, schnelles Erstellen von (teilweise) funktions-fähigen Prototypen, die beim Auftraggeber immer schon mal (an-)getestet werden können
Prototypen werden immer weiter verbessert, Leistungs-fähiger gemacht, bis sie die Anforderungen erfüllen

Vorteile:
- schnelles Bereitstellen eines Prototypen, der dem Ziel und der der Endversion schon in Grenzen entspricht
- setzt i.A. großes und Ziel-gerichtetes Entwicklungs-Potential frei

Nachteile:
- hohe Fehler-Anfälligkeit bei Erst-Projekten oder bei Anfängern
- Kosten können schnell Butget sprengen (da aus der Anwendungs-Praxis schnell neue Ideen / Ziele / Wünsche) entstehen

- **Spiral-Modell**

Entwicklung läuft in einem interativen Zyklus
nach jedem Durchlauf ist ein Prototyp der IST-Zustands vorhanden (reduziert Entwicklungs-Risiken)



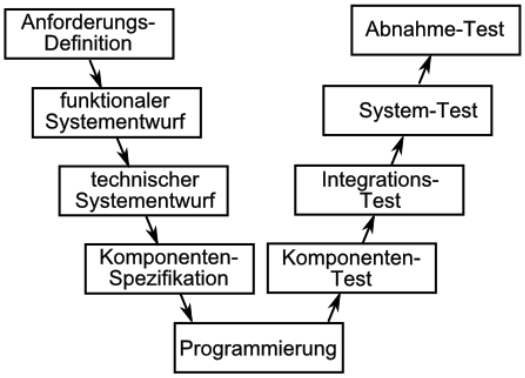
Vorteile:
- interatives Arbeiten
- häufige Risiko-Analyse
- Erstellung / Vorhandensein von Prototypen

Nachteile:
- keine Festlegung von Verantwortlichkeiten
- keine parallelen Abläufe vorgesehen

Vorgehens-Modelle für die Entwicklung einer Datenbank (Fortsetzung)

- **V-Modell**
  (V-Modell XT)

Entwicklung wird besonders aus technischer und funktionaler Sicht betrachtet
ähnelt dem Wasserfall-Modell, sieht aber darüber hinaus auch Rückkopplungen auf vorherige Phase vor
verstärkte Berücksichtigung von Qualitätssicherungs-Maßnahmen
beinhaltet auch Festlegung von Verantwortlichen



Vorteile:
- für alle Unternehmes-Arten / Projekte geeignet
- gut für kleine und mittlere Projekte geeignet
- interatives Arbeiten
- Verringerung von Risiken / Begrenzung der Projekt-Kosten
- gute Qualitätssicherung
- verbesserte Kommunikation zwischen Projekt-Beteiligten
- Erstellung von (teilweise nutzbaren) Prototypen im Projekt-Verlauf

Nachteile:
- Vergabe von Dienstleistungen ist nicht definiert
- keine Unterscheidung von Projekt-Auftraggeber und –nehmer bei der Einführung und Pflege des Projekts
- Organisation und Ablaufsteuerung nicht bestimmt

- **Scrum**

leichtes Framework für komplexe Projekte
stark auf Team-Arbeit und Eigenverantwortlichkeit orientiert
Team besteht minimal aus drei Rollen: Product Owner, Entwicklungs-Team und Scrum-Master
viel Kommunikation, Kreativität und Transparenz; hohe Produktivität
ständige Festlegungen und Weiterentwicklungen / auch Richtungs-Änderungen möglich

Vorteile:
- saubere Festlegung von "Definition of Done" (DoD, Fertigstellungs-Kriterium (unabhängig vom weiteren Verbesserungs-Potential))

Nachteile:
- sehr viel Eigenverantwortung, Transparenz und Team-Fähigkeit notwendig
- viele Team-Sitzungen (aufwendige Planung und Koordinierung)

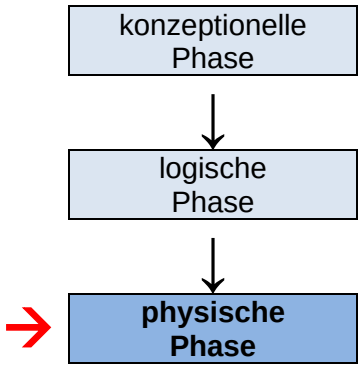
-

Aufgaben:

1. *In einem AccessPoint mit 5 Kanälen (Antennen) soll in einer Datenbank die aktuelle Verbindungs-Situation gespeichert werden. Jedem Kanal sind bestimmte veränderliche und eigenständige Eigenschaften (Frequenz, Orientierung, Verschlüsselung) zugeordnet. Die sich verbindenden Endgeräte (Client's) sind durch eine MAC-Adresse und einem Gerätenamen gekennzeichnet. Neue Client's sollen später durch andere Funktionen des AccessPoint's hinzugefügt werden. Alte Client's bleiben in der Datenbank.*
   Erstellen Sie ein Entity-Relationship-Diagramm für die Verbindungs-Datenbank!
2. *Die Fahrrad-Ausleihe "Ostsee-Bike" hat 25 Fahrräder für Kunden bereitstehen. Einige der Fahrräder sind eBike's. Außer der Rahmennummer haben die Fahrräder eine fortlaufende interne Nummer. Die Fahrräder sind in verschiedenen Rad-Größen und Rahmenfarben verfügbar. Es gibt Fahrräder extra für Frauen und Männer sowie Unisex-Versionen. Aus den umliegenden Hotels leihen sich die Gäste die Fahrräder immer für ein, zwei Tage aus. Einige Gäste legen Wert auf die gleiche Rahmenfarbe. Die Abrechnung erfolgt auf der Basis der Ausleihdauer in Tagen und der gewählten Fahrrad-Art (normal oder eBike). Die Hotelgäste bezahlen über ihre Hotel-Rechnung (Aufbuchung auf ihre Zimmer-Rechnung). Die Hotel's überweisen die Beträge direkt auf ein Konto. Erstellen Sie auf der Basis eines Entity-Relationship-Diagramms eine Datenbank-Struktur, um den beschriebenen Sachverhalt in einer Datenbank zu modellieren!*
3. *Eine Schüler-Firma will Nachhilfe vermitteln! Überlegen Sie sich, welche Daten für eine sinnvolle Datenverarbeitung in einer Datenbank erfasst werden müssten. Erstellen Sie ein passendes ER-Diagramm!*

3.1.3. physische Phase – Umsetzung / Implementierung in ein DBS

Jetzt kommen wir zur Qual der Wahl: Mit welchem Datenbank-System wollen wir unser(e) Projekt(e) umsetzen?
In der Praxis sind die Wahl-Möglichkeiten vor allem von den technischen und finanziellen Aspekten geprägt.
In der Schule werden uns nur einige wenige Systeme zu Verfügung stehen. Da in der Schule die finanziellen Ressourcen eher mager gestrickt sind, orientieren wir hier vorrangig auf kostenfrei zu habene Software. Und diese kann sich in vielen Punkten gut gegen Kosten-pflichtige Systeme behaupten.
Die Empfehlung geht in Richtung SQLite-Studio, welches weiter hinten vorgestellt wird (→ 3.1.7.1.3. Arbeiten mit dem SQLiteStudio).



Für die Anwender klassischer Office-Suiten sind es Libre-Office bzw. OpenOffice, die in die engere Wahl genommen werden sollten. In ihnen findet sich z.B. das Programm BASE, was viele Aspekte von microsoft ® ACCESS ® mitbringt. BASE besprechen wir im Kapitel → 3.1.5. Datenbanken mit BASE.
Alle empfohlenen Programme / Suiten sind kostenfrei und mit freien Lizenzen im Internet zu haben. Sie sind auch auf dem IoStick installiert.
Da ACCESS quasi ein Standard im Office-Bereich darstellt, werden wir auch die Umsetzung einer Datenbank vorstellen (→ 3.1.6. Datenbanken mit ACCESS).
Alle Vorstellungen verwenden das gleiche Daten-Modell. Es werden aber Programm-spezifische Unterschiede bei der Reihenfolge und den vorgestellten Programm-Leistungen gemacht.
Windows ® ist unser bevorzugtes Betriebssystem. Ab und zu werden wir auch auf die Möglichkeiten von Datenbank-Nutzungen auf dem Raspberry Pi eingehen. Gerade SQLite lässt sich hier hervorragend nutzen (→ 3.1.7.0.1. SQLite auf einem Raspberry Pi). Der kleine Rechner ist und bleibt die Geheimwaffe der Schul-Informatik.

physische Phase ist durch Entwicklung einer leeren Datenbank geprägt
aus dem logischen Modell wird durch die Nutzung einer Daten-Definitionssprache (DDL; z.B. SQL) eine nutzbare Datenbank (i.w.S.)

Aufgaben:

1. ***Erstellen Sie eine Skizze, in der Sie die Nutzer-Oberfläche (Eingabe-Bereiche und Bedien-Elemente) einer Adressbuch-Applikation (z.B. für ein Smartphone oder ein Tablet) mit Name, Vorname sowie mindestens fünf weiteren Attributen planen! (Für ev. Unter-Fenster etc. geben Sie weitere Skizzen an!)***

Aufgaben:

1. ***Stellen Sie die Attribute (Farbe, Türenanzahl, Motorstärke, Modellname, Preis) für die Autos eines Autohändlers als Entity-Relationship-Diagramm dar!***
2. ***Setzen Sie das Diagramm von Aufgabe 1 in einem Zeichen-Programm (z.B. Dia, LibreOffice- bzw. OpenOffice-Draw) um! Die Datei soll "ERD_Autohändler_V1" heißen!***
3. ***Bestimmen Sie die Kardinalitäten der nachfolgenden Beziehungen! Erläutern Sie Ihre Entscheidung!***

a)	Klasse	-----	hat	-----	Schüler
b)	Elternpaar	-----	hat	-----	Kind
c)	Frau	-----	heiratet	-----	Mann
d)	Frau	-----	kauft	-----	DesignerSchuh
e)	Schüler	-----	erhält	-----	Zeugnis
f)	Künstler	-----	malt / hat gemalt	-----	Bild
g)	Student	-----	besucht	-----	Kurs
h)	Schüler	-----	ist_befreundet_mit	-----	Schüler
i)	Leser	-----	leiht_aus	-----	Buch
j)	Schüler	-----	hat	-----	Zeugnis
k)	Kind	-----	hat	-----	Elter

4. ***Eine Video-DVD-Ausleihe möchte ihre Videos und DVDs sowie die Kunden und Ausleihen in einer Datenbank erfassen und verwalten. Erstellen Sie ein ERD und eine passende Skizze am Computer (z.B. mit Dia, LibreOffice Draw, …)!***
5. ***Ein Fitness-Studio möchte seine Mitglieder, Kurse und Kursleiter in einer Datenbank verwalten. Erstellen Sie gemeinsam an der Tafel / am Whiteboard ein ERD für eine praktikable Datenbank!***
6. ***Die Rechtsanwalts-Gemeinschaft "Dr. Schröder, Schröder und Schmittchen" will ihre Termine verwalten. Als notwendige Angaben haben sich weiterhin herausgestellt:***
   - ***die Mandanten mit Name (können als Kläger oder Beklagte auftauchen), ev. Rechtsschutzversicherung, Anschrift, eMail-Kontakt, Telefonnummer***
   - ***Fall, zuständiges Gericht, Aktenzeichen***
   - ***Termin (Datum, Uhrzeit), Ort, Zweck, Dauer***

   Erstellen Sie eine ERD am Computer! Wenn Sie weitere Daten als notwendig erachten, dann können Sie diese ergänzen.
7. ***Für eine kleine lokale Verleihstation von Elektro-Autos soll eine einfache Verwaltungs-Datenbank erstellt werden. Analysieren Sie die praktische Situation und erstellen Sie ein ERD als Skizze auf einem Blatt Papier!***
   Diskutieren Sie die verschiedenen Vorschläge innerhalb des Kurses!
   Erstellen Sie ein gemeinschaftlich akzeptiertes Diagramm am Computer!
8. ***Die nachmittäglichen Arbeitsgemeinschaften, Sportgruppen usw. einer Schule sollen in einer Datenbank (für den Koordinator / verantwortlichen Lehrer / die Schulleitung) verwaltet werden. Erstellen Sie ein ERD zu dieser Situation am Computer!***

9. Übernehmen Sie folgende Situations-Beschreibung in ein Dokument (microsoft Word oder LibreOffice Writer)! Als Überschrift nehmen Sie bitte: "Machbarkeits-Analyse Wohnungs-Genossenschaft". Setzen Sie sich als Bearbeiter darunter.

Eine Wohnungs-Genossenschaft hat mehrere Häuser mit jeweils unterschiedlich vielen Wohnungen und unterschiedlicher Größe. Zwei Hausmeister betreuen die Wohnungen häuserweise.

Von den Mietern sollen in der Wohnungs-Verwaltung nur die Nachnamen und Vornamen der Haupt-Mieter (Ansprechpartner) erfasst werden!

Da die Menge an Häusern und Wohnungen das Mass für einen kleinen Karteikasten mittlerweile überschreitet und die Kommunikation über Computer-Dokumente erfolgen soll, überlegt die Wohnungs-Genossenschaft eine Datenbank anzuschaffen. Weiterhin soll auf einer Tafel im Verwaltungs-Gebäude eine Übersicht zu sehen sein, welche Wohungen frei sind, wo und wie sie liegen, wie groß sie sind und was sie kostet.

Zuerst soll nur die Umsetzbarkeit und der inhaltliche Aufwand geprüft werden.

a) Überlegen Sie sich, welche Daten für eine Umsetzung des Sachverhaltes "Wohnungs-Verwaltung" erfasst werden müssten! Klassifizieren Sie die Merkmale / Daten nach Wichtigkeit in drei Gruppen (1 … "unbedingt notwendig"; 2 … "wichtig"; 3 … "verzichtbar"!

b) Erstellen Sie mit Hilfe eines Zeichen-Programm's (z.B. Dia, LibreOffice Draw, …) ein geeignetes ERD! Bauen Sie das ERD in das Dokument ein! Das Dokument darf nicht größer als 1 Seite (A4) werden.

c) Erläutern Sie in kurzen Texten, warum Sie die Merkmale den bestimmten Gruppen zugeordnet haben!

d) Drucken Sie das Dokument 1x aus und tauschen Sie es innerhalb des Kurses im Rotations-Verfahren (immer über zwei Teilnehmer weiterreichen) aus!

e) Analysieren Sie den vorliegenden fremden Vorschlag und notieren Sie in einem Dokument Fragen, Widersprüche usw.! Machen Sie mit Bleistift Fragezeichen an die Stellen im Ausdruck der "Machbarkeits-Analyse"! Geben Sie dann die Analyse mit Ihrem ausgedruckten Fragen usw. zurück!

f) Setzen Sie sich mit den aufgeworfenen Fragen usw. auseinander und verbessern Sie u.U. Ihre Analyse!

3.1.4. Aufbau klassischer relationaler Datenbanken

manchmal als Objekt-Typen bezeichnete unterschiedliche Teile / Nutzungs-Prinzipien einer Datenbank
sind die Datenbank-Elemente, die wir erstellen, verändern und letztendlich mit Daten füllen bzw. sie Problem-orientiert abwandeln können
orientiert am Platzhirsch für Desktop-Datenbanken im Office-Bereich → microsoft® office ACCESS

Datenbanken-Bestandteile

• **Tabellen**	Datenbasis in Tabellen (Spalten-Zeilen-Anordnung) echte Daten(-Tabellen) dazu Beziehungen zwischen den Tabellen eine spezielle Art der Tabellen sind die sogenannten Indices
• **Abfragen**	temporäre Tabellen durch Umordnung, Sortierung, Kombination, Gruppierung und Filterung der echten Daten (aus einer oder mehrerer "Tabellen" praktisch sind nur die Vorschriften (zur Bildung einer Abfrage) gespeichert, die dann bei Nutzung aufgerufen und ausgeführt werden
• **Berichte**	Zusammenfassung der Daten für eine bestimmte Datenbank-Situation praktisch sind nur die Auswerte-Vorschriften und –Kriterien gespeichert; sie werden bei Bedarf aufgerufen und ausgeführt und produzieren ein abspeicherbares bzw. ausdruckbares Dokument
• **Formulare**	dienen der Benutzer-freundlichen Eingabe und Darstellung der Daten (es können Daten aus mehreren Tabellen oder Abfragen benutzt werden; eingegebene Daten werden automatisch richtig den betreffenden Tabellen zugeordnet)
• **Makros / Programme**	dienen der Automatisierung von Nutzer-Aktionen, realisieren Kontrollen und spezielle Daten-Umsetzungen (z.B. Importe od. Exporte) usw. usf. es besteht die Möglichkeit der Erstellung einer scheinbar (vom Datenbank-System) unabhängigen Applikation

die ersten vier werden als klassische Bestandteile für Datenbanken a'la ACCESS (→ 3.1.6. Datenbanken mit ACCESS) verstanden
für die frei nutzbaren Office-Sammlung LibreOffice bzw. OpenOffice ist es das Programm BASE, welches diese Bestandteile anbietet (→ 3.1.5. Datenbanken mit BASE)
in den Anwendungen, die sich mehr auf den Bereich des Datenbank-Management-System's sehen, sind meist nur Tabellen, Abfragen und (einfache) Formulare umgesetzt (z.B. SQLite-Studio (→ 3.1.7. Datenbanken mit SQLite))
die meisten enthalten auch Programmier-Sprachen für die Erzeugung von Makros bzw. Applikationen
die ersten drei – der oben aufgezählten – Bestandteile sind praktisch in allen Datenbanken und Datenbank-Systemen vorhanden

Definition(en): Index
Ein Index (Mehrzahl: Indices) ist eine Zeiger-orientierte Schnell-Zugriffs-Möglichkeit auf die Datensätze einer Relation (Tabelle).

Indices werden als einfache zwei-spaltige Tabellen, als Hash-Tabellen oder in Form von sehr effektiven B*-Bäumen gespeichert.
Der Haupt-Index wird Primär-Index genannt und enthält meist den Schnell-Zugriff auf das gespeicherte Objekt (im Datensatz).
Weitere Indices werden Sekundär-Indices genannt und enthalten meist Schnell-Zugriff über untergeordnete Attribute oder Attribut-Kombinationen.

3.1.4.1. Tabellen

reine Datensammlungen mit fester Struktur
deshalb sehr schnell

besonders wichtig ist es, auf die Definition des Primär-Schlüssels einer jeden Tabelle zu achten
fehlt dieser in der erarbeiteten Tabelle, dann erstellt das System meist selbstständigen einen Primär-Schlüssel als neue Spalte
ev. sind Tabellen-Struktur-Korrekturen notwendig oder die gesamte Planung (ERD, Zusammenhänge, …) muss angepasst werden
Tabellerische Daten sind aber auch erst einmal als erste "Datenbank" ok. Natürlich muss dann schnell eine Optimierung erfolgen (→ Normalisierung (→ 2.2.1. Normalisierung)), damit wir dann auch wirklich einen Datenbank mit all ihren Leistungen und Vorteilen zur verfügung haben.
Will man lediglich die Daten in einer Tabelle halten und bearbeiten, dann reicht vielleicht auch EXCEL® (von microsoft ®) bzw. CALC aus Open- oder Libre-Office. Diese bieten erste "Datenbank"-Funktionen an. Für komplexe Datenstrukturen sind Tabellenkalkulationen aber nicht geeignet.

Um den verfügbaren Speicherplatz optimal zu nutzen, müssen die Daten in geeigneten Formaten gespeichert werden. Auch die weitere Verwendung spielt bei der Auswahl von Datenformaten eine Rolle. Zahlen, die als Text – also Zeichenfolgen (vielleicht sogar noch mit Leerzeichen usw. usf.) – gespeichert sind, eignen sich kaum zum Verrechnen.
Ständiges Neuformatieren, Umrechnen usw. beinhaltet immer ein großes Fehler-Potential. Schließlich müssen oft auch wieder Ergebnisse in die "seltsamen" Formate zurück übertragen werden.
In der nächsten Tabelle sind diverse Daten-Formate zusammengestellt. Ev. muss man aber die eigenen System-Bedingungen beachten. Nicht jedes System versteht ein Format so wie vielleicht die Mehrheit oder die Standards. Gerade proprietäre System sind da sehr erfinderisch, um Kunden mit ihren unübertragbaren Formaten Kunden an sich zu binden.

Definition(en): NULL-Wert
Ein NULL-Wert ist ein spezieller Eintrag für ein Attribut. Es charakterisiert einen leeren Eintrag.

Ein NULL-Wert entspricht **nicht** dem Zahlenwert einer Null. Die Nutzung des NULL-Wertes in realtionalen Datenbanken impliziert eine ternäre Verarbeitungs-Logik für alle Datenbank-Operationen.

Daten-Typen

Bezeichnung	Werte-Bereich (Domäne)	Name in SQL	Erläuterung	Speicher-Bedarf	Name in BASE	Name in ACCESS		
ganze Zahl	0 .. 255			1 Byte		Byte		
	-32'768 .. 32'767	SMALLINT		2 Byte		Integer		
	-2'147'483'648 .. 2'147'483'647	INTEGER		4 Byte		Longinteger		
Autowert Zähler				4 – 16 Byte		Auto		
	1, 2, 3 ...			16 Byte		Replication ID		
reele Zahl		DECIMAL (s,n)	Fließkomma-Zahl mit mindestens **s** Stellen (insgesamt) und davon **n** Nachkommastellen					
		NUMERIC (s,n)	Fließkomma-Zahl mit genau **s** Stellen (insgesamt) und davon **n** Nachkommastellen					
		FLOAT (n)						
	$-3{,}403 * 10^{38}$ .. $3{,}03 * 10^{38}$		Genauigkeit: 10^{-7}	4 Byte		Single		
	$-1{,}798 * 10^{308}$ .. $1{,}798 * 10^{308}$		Genauigkeit: 10^{-15}	8 Byte		Double		
Währung					Währung			
logischer Wert			True oder False Wahr oder Falsch Ja oder Nein	1 bit (1 Byte)				
Zeichenkette		CHAR (n)						
					Text	Text		
variable Zeichenkette		VARCHAR (n)						
			variabler Text	→ 64'000 Byte		Memo		
Datum		DATE			Datum	Datum		
Uhrzeit					Zeit	Zeit		

eingebettete Objekte			z.B. Bilder, Graphiken, Videos, Musik	→ 1 GB		OLE-Objekt		

3.1.4.2. Abfragen

In den wenigsten Datenbanken dürfen alle Nutzer auch auf alle Daten zugreifen. Vielfach macht es auch keinen Sinn, alle Daten einer Tabelle mit einem Mal darzustellen. Z.B. darf nicht jeder Nutzer einer Firmen-Datenbank die Löhne einsehen oder sich in einem online-shop die Kreditkarten-Daten anzeigen lassen.
Um die Auswahl der Spalten und Zeilen für den Nutzer bzw. seinem Bedarf entsprechend passgerecht zu machen, werden **Abfrage**n benutzt. Andere Namen für Abfragen sind **View**'s oder **Sicht**en.
Praktisch unterscheiden wir die folgenden Arten von Abfragen:

• **Auswahl-Abfragen**	ermöglicht die eingeschränkte Anzeige bzw. Tabellierung von Inhalten aus einer oder mehrerer Tabellen bzw. Abfragen praktisch wird eine temporäre Auswahl von Spalten (Felder, Attributen) und Zeilen (Datensätzen) aus den Tabellen u. / od. Abfragen erzeugt
• **Auswahl-Abfragen mit Auswertungen (bzw. Berichts-Charakter)**	neben den Beschränkungen (Auswahl) hinsichtlich der Zeilen (Datensätze) und Spalten (Felder, Attribute) können bestimmte Feldwerte gezählt, summiert sowie Minimum und Maximum bestimmt werden innerhalb von Zeilen lassen sich auch Felder berechnen (z.B. "volljährig" aus Geburtsdatum und aktuellem Datum)
• **Parameter-Abfragen**	hierbei ist die nachfolgende Abfrage von einer oder mehrerer zu tätigen Eingaben (- den Parametern -) abhängig; sie stellen die Möglichkeit dar, flexible Abfragen zu erstellen / nutzen

Von der Art und Weise der Erstellung unterscheiden sich diese Abfrage-Arten recht wenig, so dass wir hier nicht so einen Wert auf die Unterschiede legen.
Manchmal wird auch noch nach den folgenden Abfrage-Arten unterschieden:

- **Erstellungs-Abfragen**
- **Anfüge-Abfragen**
- **Lösch-Abfragen**
- **Aktualisierungs-Abfragen**

3.1.4.2.1. einfache Tabellen-Datenbanken mit microsoft EXCEL bzw. Libre-Office-/OpenOffice CALC

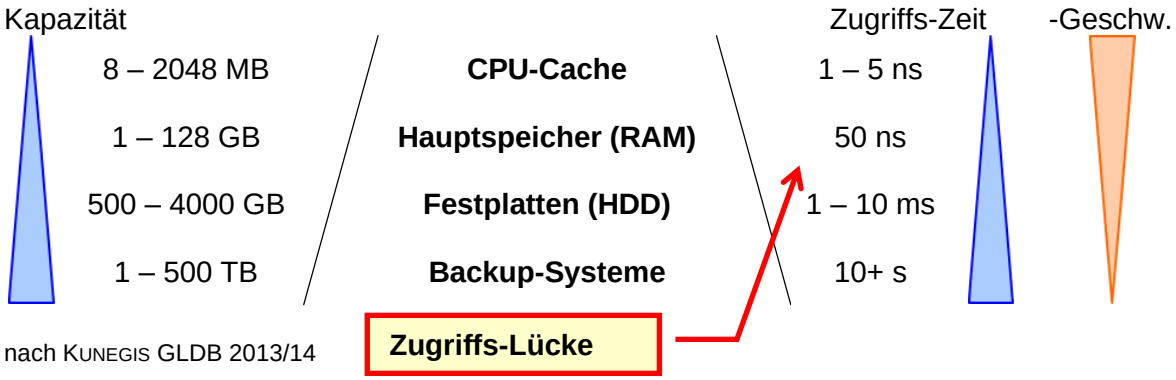
Für die Erstellung und Verarbeitung von Tabellen sind die Office-Programme EXCEL und CALC oft schon eine gute Lösung. Wir müssen hier aber gleich klarstellen, bei den Tabellen-Kalkulationen handelt es sich nicht um Datenbank(-System-)en im Sinne dieses Skriptes und der Informatik. Es sind vorrangig Tabellen, die durch Beziehungen (z.B. Verweis-Funktionen) verknüpft werden können und für die einige Datenbank-orientierte Tabellenblatt-Funktionen bereitgestellt werden.
Die bereitgestellten DB-Tabellenblatt-Funktionen ermöglichen es vorrangig Kennwerte über den Datenbestand zu berechnen. Bei den Funktionen handelt es sich um Zählungen, Summen, Mittelwerte sowie ausgewählte Statistik-Kennzahlen. Im Unterschied zu den klassischen Tabellenblatt-Funktionen lassen sich Abfrage-ähnliche Konstrukte aufbauen.
Mit einem recht großen Aufwand lassen sich die Daten(-Sätze) selbst abfragen und darstellen. Dies geht aber deutlich über das Schul-Niveau hinaus (→ Links).
Man soll mich hier nicht falsch verstehen, Tabellen-Kalkulations-Programme können ohne weiteres ein völlig ausreichendes Mittel zum Lösen eines Datenbank-Problem's sein. Man muss nicht immer mit Kanonen auf Spatzen schießen. Die Verbindung mehrerer Tabellen und die Erstellung Berichts-orientierter Abfragen sind (z.B. für Rechnungen, Inventuren, …) sind sehr gut möglich.

Links:
https://www.tabellenexperte.de/besser-als-sverweis-alle-werte-finden/ (erläutertes Beispiel zur Abfrage von Daten / Datensätzen aus einer EXCEL-Datenbank)

```
{=INDEX($B$2:$B$309;KGRÖSSTE(($A$2:$A$309=$F$1)*(ZEILE($A$2:$A$309)-
1);ZÄHLENWENN($A$2:$A$309;$F$1)+1-ZEILE(A1)))}


=INDEX(Bezug; Zeile)

=KGRÖSSTE(Matrix; k)
KGRÖSSTE(($A$2:$A$309=$F$1)*(ZEILE($A$2:$A$309)-
1);ZÄHLENWENN($A$2:$A$309;$F$1)+1-ZEILE(A1))

($A$2:$A$309=$F$1)*(ZEILE($A$2:$A$309)-1)

=ZÄHLENWENN($A$2:$A$309;$F$1)+1-ZEILE(A1)
```

3.1.4.3. Indicies

Tabellen lassen sich praktisch nur nach einem Attribut ordnen / sortieren. Natürlich sind untergeordnete (sekundäre) Ordnungen / Sortierungen möglich.
Häufig braucht man aber auch schnelle Zugriffe auf andere Attribute. Liegen diese ungeordnet vor, ist der Zugriff meist sehr aufwändig.
Mit einem Index erzeugt man sich eine geordnete Hilfs-Tabelle, die neben dem geordneten Wert nur noch den Primärschlüssel aus der Daten-Tabelle enthält. Wenn es notwendig und sinnvoll ist, kann man sich für jedes Attribut – und praktisch auch für Attribut-Kombinationen – einzelne Indicies anlegen.
Die eigentliche Daten-Tabelle wird von Aktionen auf den Indicies nicht im Geringsten beeinflusst. Die Indicies sind quasi Schnell-Zugriffs-Listen zu Daten-Tabellen. Da die Indicies nur sehr kleine Tabellen sind, können sie häufiger aktuallisert und optimiert werden.
Ein weiterer Grund für die Nutzung von Index-Tabellen ist die sogenannte Zugriffs-Lücke bei Datenspeichern. Sie erstreckt sich über rund fünf Zehner-Potenzen.



Festplatten sind für ein ständiges Daten-Hin-und-Her-Schaufeln zu langsam. Schließlich muss ja die superschnelle CPU mit Daten versorgt werden. Hauptspeicher ist zwar deutlich schneller, aber eben nicht als dauerhafter Speicher geeignet. RAM ist auch relativ Kostenintensiv.
Der Kompromiß ist eben eine leichte (kleine) Datenstruktur im Speicher und schnellere Zugriffe auf die Festplatten-Daten über Indicies.
Durch moderne SSD-Festplatten (Solid Disk) wird die Zugriffs-Lücke nur geringfügig ausgeglichen. Praktisch kommte es zur Verbesserung um eine Zehner-Potenz. Größere SSD's sind aber auch noch relativ teuer. In den kommenden Jahren wird sich dies aber sehr schnell ändern.

"normale" Telefonbuch-Liste

Tel.Nr.	Name	Strasse
20386	Müller	Hauptstr. 3
20389	Holatov	Rosenweg 25
20390	Schulze	Friedensstr. 4
20402	Asch	Rosenweg 2
20408	Niemeyer	Friedensstr. 4
22222	Meier	Rosenweg 5
23456	Hausmann	Am See 23
…	…	…
33333	Holtz	Am See 24
34567	Friederich	Friedensstr. 22
34555	Diederich	Hauptstr. 7
…	…	…
40000	Vielmann	Am See 24
40404	Niemann	Am See 24
44444	Krämer	Hauptstr. 31
48884	Meyer	Wiesensteig 6
…	…	…
86208	Schulz	Hauptstr. 9
88888	Hausmann	Friedensstr. 5

sortierte Namens-Liste

NID	Name	Tel.Nr.	Strasse
3	Asch	20402	Rosenweg 2
32	Diederich	34555	Hauptstr. 7
78	Friederich	34567	Friedensstr. 22
124	Hausmann	23456	Am See 23
125	Hausmann	88888	Friedensstr. 5
221	Holatov	20389	Rosenweg 25
356	Holtz	33333	Am See 24
…	…	…	…
411	Krämer	44444	Hauptstr. 31
437	Meier	22222	Rosenweg 5
446	Meyer	48884	Wiesensteig 6
479	Müller	20386	Hauptstr. 3
488	Niemann	40404	Am See 24
583	Niemeyer	20408	Friedensstr. 4
…	…	…	…
843	Schulz	86208	Hauptstr. 9
844	Schulze	20390	Friedensstr. 4
863	Vielmann	40000	Am See 24
…	…	…	…

angenommener Speicher-Aufwand:
Tel.Nr. 3 Byte
Name 20 Byte
Strasse 20 Byte
ID's 3 Byte

Beispiel:
Telefonbuch: 16 x 43 Byte = 688 Byte
Namens-Liste: 16 x 46 Byte = 736 Byte
Straßen-Liste: 16 x 46 Byte = 736 Byte

2'160 Byte

Nachteile:
- hohe Redundanz
- großer Sortier-Aufwand
- hohe Einfüge- und Neusortier-Aufwand

sortierte Straßen-Liste

SID	Strasse	Tel.Nr.	Name
2	Am See 23	23456	Hausmann
3	Am See 24	33333	Holtz
4	Am See 24	40000	Vielmann
8	Am See 24	40404	Niemann
…	…	…	…
41	Hauptstr. 3	20386	Müller
43	Hauptstr. 7	34555	Diederich
44	Hauptstr. 9	86208	Schulz
47	Hauptstr. 31	44444	Krämer
82	Friedensstr. 4	20390	Schulze
84	Friedensstr. 4	20408	Niemeyer
95	Friedensstr. 5	88888	Hausmann
99	Friedensstr. 22	34567	Friederich
…	…	…	…
119	Rosenweg 2	20402	Asch
134	Rosenweg 5	22222	Meier
136	Rosenweg 25	20389	Holatov
…	…	…	…
453	Wiesensteig 6	48884	Meyer

"normale" Telefonbuch-Liste

Tel.Nr.	Name	Strasse
20386	Müller	Hauptstr. 3
20389	Holatov	Rosenweg 25
20390	Schulze	Friedensstr. 4
20402	Asch	Rosenweg 2
20408	Niemeyer	Friedensstr. 4
22222	Meier	Rosenweg 5
23456	Hausmann	Am See 23
…	…	…
33333	Holtz	Am See 24
34567	Friederich	Friedensstr. 22
34555	Diederich	Hauptstr. 7
…	…	…
40000	Vielmann	Am See 24
40404	Niemann	Am See 24
44444	Krämer	Hauptstr. 31
48884	Meyer	Wiesensteig 6
…	…	…
86208	Schulz	Hauptstr. 9
88888	Hausmann	Friedensstr. 5

sortierter Namens-Index

NID	Name	Tel.Nr.
3	Asch	20402
32	Diederich	34555
78	Friederich	34567
124	Hausmann	23456
125	Hausmann	88888
221	Holatov	20389
356	Holtz	33333
…	…	…
411	Krämer	44444
437	Meier	22222
446	Meyer	48884
479	Müller	20386
488	Niemann	40404
583	Niemeyer	20408
…	…	…
843	Schulz	86208
844	Schulze	20390
863	Vielmann	40000
…	…	…

angenommener Speicher-Aufwand:
Tel.Nr. 3 Byte
Name 20 Byte
Strasse 20 Byte
ID's 3 Byte

Beispiel:

Telefonbuch:	16 x 43 Byte	=	688	Byte
Namens-Index:	16 x 6 **(26)** Byte	=	96	**(412)** Byte
Straßen-Index:	16 x 6 **(26)** Byte	=	96	**(412)** Byte
			880	**(1'520) Byte**

Vorteile:

- wenig / keine Redundanz
- kleinerer Sortier-Aufwand
- kleinerer Einfüge- und Neusortier-Aufwand

sortierter Straßen-Index

SID	Strasse	Tel.Nr.
2	Am See 23	23456
3	Am See 24	33333
4	Am See 24	40000
8	Am See 24	40404
…	…	…
41	Hauptstr. 3	20386
43	Hauptstr. 7	34555
44	Hauptstr. 9	86208
47	Hauptstr. 31	44444
82	Friedensstr. 4	20390
84	Friedensstr. 4	20408
95	Friedensstr. 5	88888
99	Friedensstr. 22	34567
…	…	…
119	Rosenweg 2	20402
134	Rosenweg 5	22222
136	Rosenweg 25	20389
…	…	…
453	Wiesensteig 6	48884

Aufgaben:

1. *Berechnen Sie die Speicher-Einsparung für die optimale Index-Lösung!*
2. *Wie verhält sich der Speicher-Bedarf und der Spar-Effekt bei linear steigender Zunahme der Datensätze (Telefonbuch-Einträge)? Begründen Sie Ihrer Meinung!*

3.1.4.4. Berichte

enthalten Zusammenfassungen, Gruppierungen und Berechnungen
sonst praktisch den Sichten und Tabellen sehr ähnlich
in Berichten können die elementaren Daten auch fehlen (für die Erstellung des Berichts werden sie natürlich benutzt, aber im Bericht selbst nicht angezeigt)

häufig in besondere Druck-Formen gepresset

es gibt hier spezielle Software, die sich ausgehend von beliebigen Datenbanken, nur um die Erstellung geeigneter / spezieller / Leistungs-stärkerer Berichte kümmert (z.B. CrystalReports)
der Datenbank-Zugriff erfolgt über spezielle Schnittstellen (→ 6.2. universelle Datenbank-Schnittstellen)

3.1.4.5. Formulare

nicht in allen Datenbank-System enthalten
hier ist dann spezielle Anwender-Software notwendig, die über die klassischen Schnittstellen (→ 6.2. universelle Datenbank-Schnittstellen) auf die Datenbank zugreift

wenn Formulare im DBMS machbar sind, dann sind diese meist sehr System-spezifisch und kaum auf neuere / andere Datenbanken (selbst mit gleicher Struktur) übertragbar

bilden wesentliche Bausteine für programmierte Datenbanken (→ 6. Datenbanken - Programmierung)

3.1.4.5. Makros / Programmierung

Makro's sind aufgezeichnete oder programmierte Arbeits-Schritte, die über Tastatur-Kürzel, spezielle Schaltflächen oder andere Makro's bzw. Programme aufgerufen werden können
gehören nicht zu klassischen DBMS

eignen sich besonders dazu – bestimmte, sich häufig wiederholende Tastatur-Befehle oder Maus-Aktivitäten od.ä. – als Gesamtheit automatisch ausführen zu lassen

benutzen zumeist eine DBMS-eigene Programmiersprache
oft von anderen Programmiersprachen abgeleitet oder Teil-Klassen großer Programmiersprachen
bei microsoft ACCESS ist das eine Version des VisualBASIC
entgegen des langläufigen BASIC ist VisualBASIC eine Objekt-orientierte und sehr Leistungs-fähige Programmiersprache

Um sich in die Nutzung und Programmierung von Makro's einzuarbeiten, empfiehlt sich das folgende stufige Vorgehen:

typische Phasen bei der Makro-Nutzung durch Anwender:

0. Phase: einfache Makro-Nutzung

vorhandene Makro's werden – z.T. ohne Kenntnis darüber, dass es sich um solche handelt – durch Vorgaben / Programm-Beschreibungen usw. usf. genutzt

1. Phase: reine Makro-Erstellung und -Nutzung

Aufzeichnen der Nutzer-Aktivitäten unter einem (Makro-)Namen und Zuordnung einer Aufruf-Möglichkeit (Name, Tasten-Kürzel, Schaltfläche)

2. Phase: Ansicht des Quell-Codes und kleine Änderungen

aufgezeichnete Makro's im Quellcode ansehen und z.B. unnötige Befehle herauslöschen oder bestimmte Befehle umgruppieren oder z.B. Durchlaufzahlen bei Schleifen verändern

3. Phase: Programmierung von Makro's

Erstellen von speziellen Makro's / Funktionen, die spezielle Aufgaben übernehmen
damit sind meist solche gemeint, die nicht über die Programm-Oberfläche realisierbar sind oder nur mit einem extrem hohen Aufwand

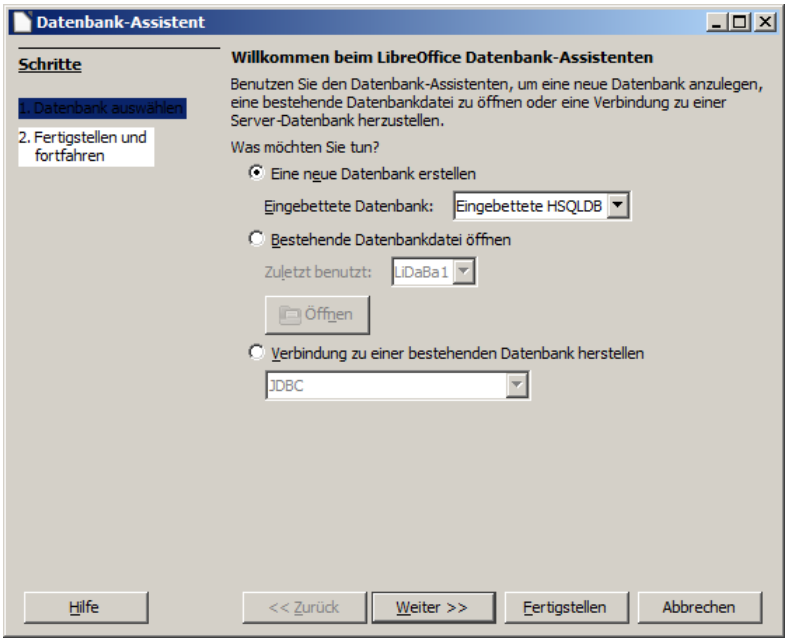
Es folgt dann meist die echte Programmierung von Datenbanken bzw. Programmen, in deren Mittelpunkt echte Datenbanken stehen.
Die meisten Programmiersprachen stellen Bibliotheken zur Benutzung von Datenbank-Schnittstellen – wie z.B. ODBC – oder direkte Datenbank-Zugriffe (meist über SQL-Statement's) bereit. Somit kann man in praktisch jeder Programmiersprache effektive relationale Datenbanken erstellen und nutzen.

3.1.5. Datenbanken mit BASE

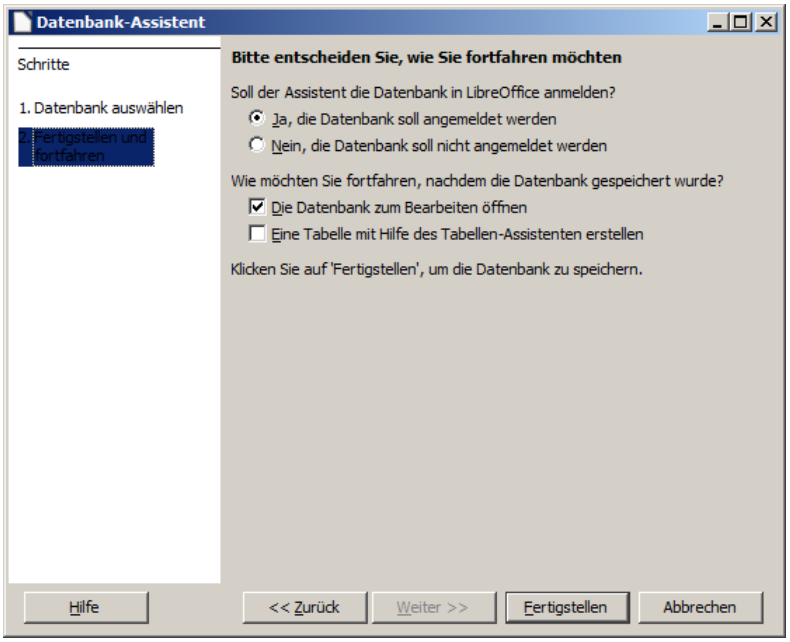
Die Nutzung von Datenbank-Komponenten aus Office-Produkten wird in einzelnen Fach-Rahmenplänen (bzw. Curricula) abgelehnt.
Für Einsteiger-Projekte sind sie aber sehr gut geeignet!

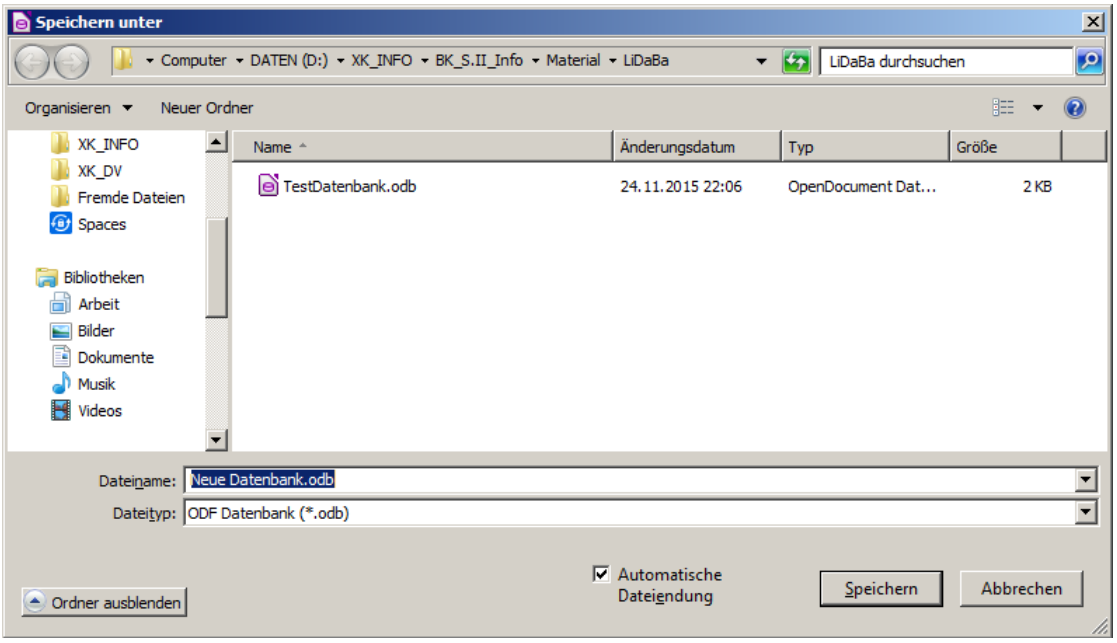
Gemeint sind hier die Datenbank-Systeme aus den Schwester-Programmen Libre- und Open-Office. Beide enthalten als Datenbank-Bestandteil das Datenbank-Management-System (DBMS) BASE.
Viele Tätigkeiten / Assistenten / … sind sehr ähnlich zu microsoft ACCESS (bis Version 2003; → 3.1.6. Datenbanken mit ACCESS) – dem Standard-Programm für Desktop-Datenbanken (kann als Vorlage für BASE angesehen werden).
Üblicherweise ist die Arbeitsgrundlage für die Implementierung einer Datenbank ein Entity-Relationship-Diagramm (ERD; → 2.1.4. Entity-Relationship-Diagramme (ERD)).
Da BASE über ein Assistenten-System verfügt, wollen wir anhand diesem die Erstellung einer Datenbank aufzeigen. Später folgen dann auch die etwas professionelleren Erstellungs-Möglichkeiten (→ 3.1.5.1.2. Erstellen einer Tabelle in der Entwurfs-Ansicht)

Start mit Assistenten zum Erstellen oder Auswählen (Öffnen) einer Datenbank. Das ist praktisch eine Container-Datei, in der dann alle Bestandteile gespeichert werden.
Erstellen einer neuen Datenbank, das Öffnen einer vorhandenen Datenbank oder die Kontakt-Aufnahme zu einer externen Datenbank.
Im letzteren Fall wird meist eine lokale Schatten-Datenbank angelegt, die quasi dem Original entspricht, aber BASE-typisch dargestellt wird.



"Anmeldung" bedeutet hier, dass andere Programme – hier besonders WRITER und CALC – relativ einfach auf die Daten der neuen Datenbank zugreifen können. Das ist z.B. für die Erstellung von Serienbriefen sinnvoll.
Es geht aber später auch ohne diese vorherige Anmeldung.





Zu beachten ist hier, dass ein Überspeichern nicht möglich ist, es muss also immer ein neuer Dateiname vergeben werden.

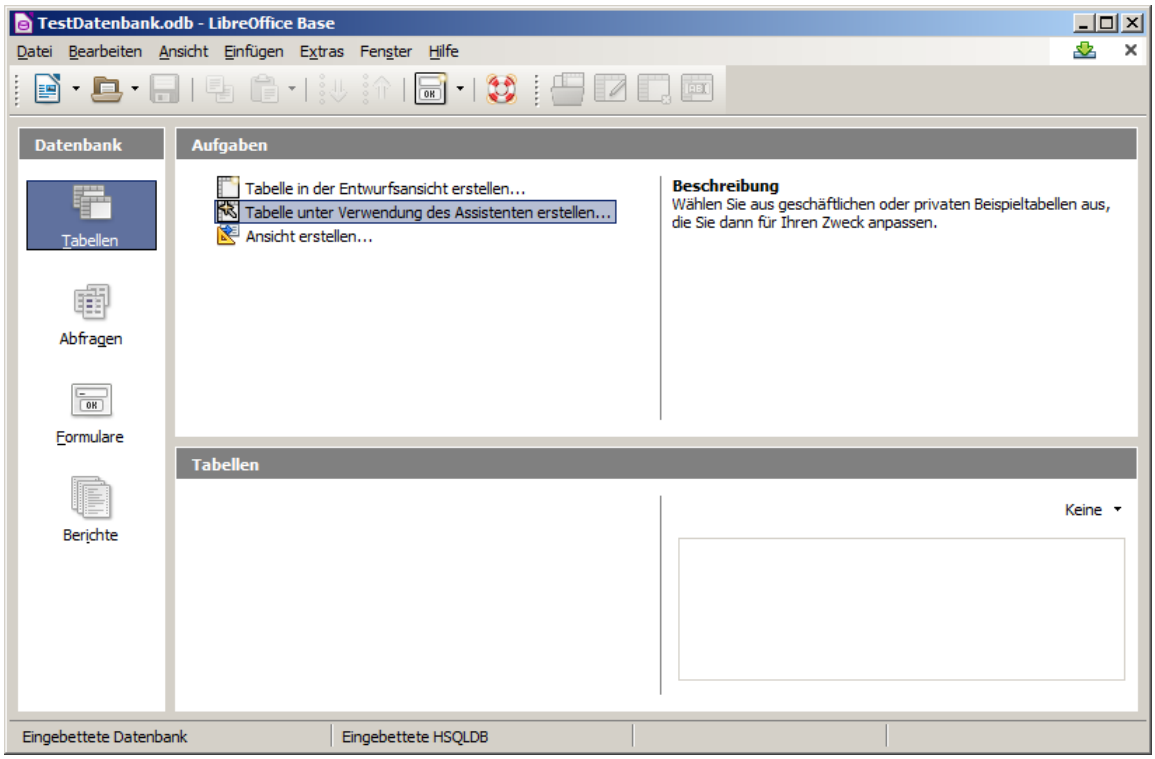
3.1.5.1. Tabellen

Tabellen lassen sich in BASE auf unterschiedliche Art und Weisen erstellen. Da wäre zum Ersten die Möglichkeit, einen Assistenten zu nutzen. Das werden wir auch zuerst einmal so tun (→ 3.1.5.1.1. Erstellen einer Tabelle mit Hilfe des Assistenten). Die Assistenten eignen sich vor allem für Anfänger und bedacht arbeitende Nutzer, da man innerhalb des Assistenten-Dialoges ständig "vor" oder "zurück" gehen kann. Mögliche Fehler lassen sich bis zum

"Fertigstellen" immer noch korrigieren. Ein weiterer Einsatzfall für Assistenten sind die klassischen Datenbank-Anwendungen, wie z.B. eine Kontakt-Datenbank. Da spart man sich eine Menge Tipp-Arbeit, da die Elemente meist nur ausgewählt werden müssen.
Zum Zweiten kann man Tabellen in der sogenannten Entwurfs-Ansicht erstellen (→ 3.1.5.1.2. Erstellen einer Tabelle in der Entwurfs-Ansicht). Das ist praktisch ein Tabellen-Struktur-Editor. Hier legen wir die Tabellen und ihre Struktur an. Das Befüllen mit Daten erfolgt dann in der "Tabellen-Ansicht" (→ 3.1.5.1.3. Eingeben von Daten in eine Tabelle).

3.1.5.1.1. Erstellen einer Tabelle mit Hilfe des Assistenten

Bei einer geöffneten Datenbank sehen wir im Bereich "Tabellen" die möglich "Aufgaben", die wir erledigen können.



Für uns ist nun die Aufgabe "Tabelle unter Verwendung des Assistenten erstellen …" interessant.

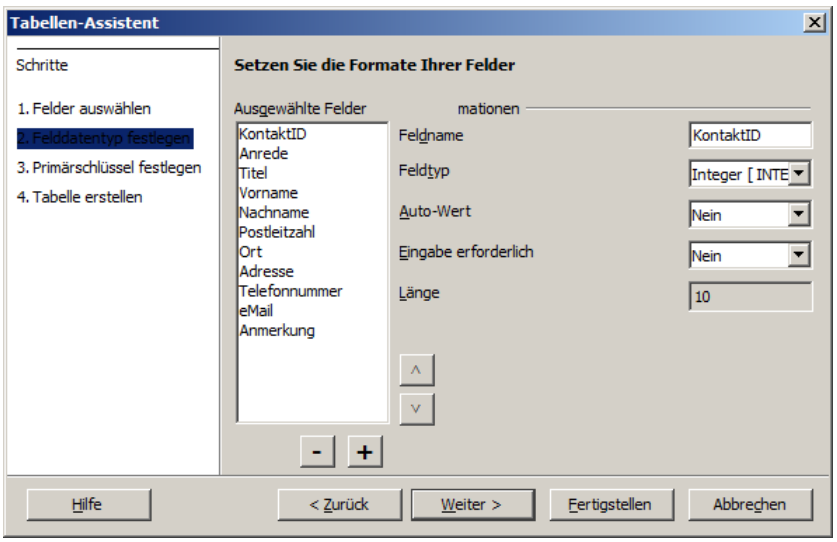
Im ersten Schritt wählen eine möglichst geeignete Beispieltabelle aus. Da unten die dann verfügbaren Felder (Attribute) gleich angezeigt werden, fällt uns die Wahl meist leicht.
Es gibt zwei große gedachte Bereiche, einmal die geschäftlichen und dann die privaten Datenbanken.
Fehlen Felder, dann kann man sie später immer noch ergänzen.

Es folgt das Zusammenstellen der gebrauchten Spalten (Attribute).

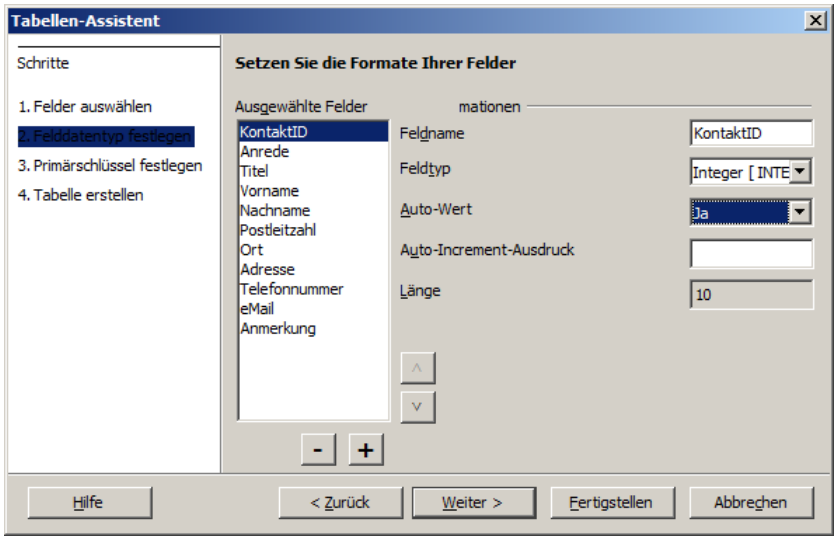
Wichtig ist hier die Fortsetzung mit "Weiter >"!

"Fertigstellen" nur dann, wenn Details, wie z.B der Primär-Schlüssel für die Datenbank-Nutzung uninteressant sind.

Der zweite Assitenten-Schritt ermöglicht uns jetzt etwas Fein-Tuning hinsichtlich der einzelnen Felder.
Für die meisten Felder sind Änderungen nicht notwendig.
Profi's können hier ihre Datenbank zumindestens hinsichtlich der Daten-Größe optimieren.



Für den zukünftigen Primär-Schlüssel unserer Tabelle nehmen wir aber noch einige Änderungen vor.
So setzen wir "Auto-Wert" auf "Ja". Dadurch wird die Vergabe der KontaktID als Schlüsselwert automatisiert. Wollen wir eigene ID's eingeben, dann belassen wir die Einstellung auf "Nein".

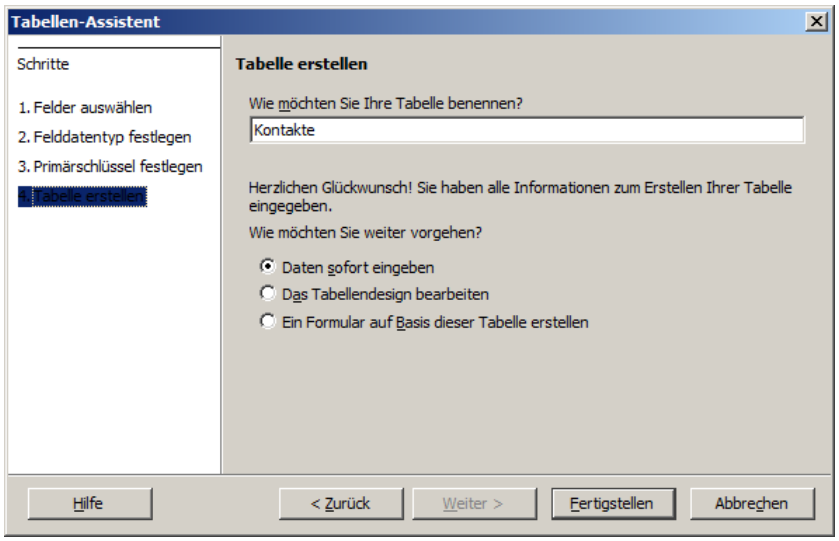


Der dritte Assistenten-Schritt ist ganz für die Festlegung / Auswahl des Primär-Schlüssel's der Tabelle gedacht. Wir haben hier die Möglichkeit eine neue Spalte mit einem neuen Primärschlüssel hinzuzufügen. Die Autowert-Funktion haben wir ja schon besprochen.
Die zweite Möglichkeit ist die Nutzung einer vorhandenen Feldes als Primär-Schlüssel.

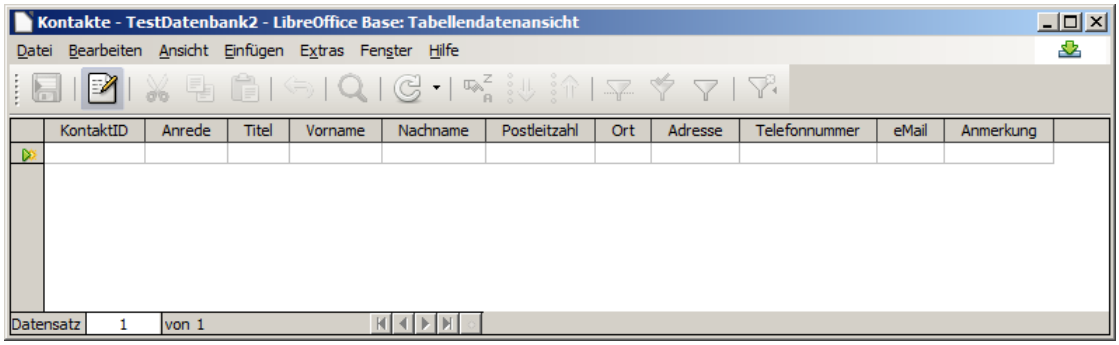
Und zu guter Letzt kann ein Primär-Schlüssel auch aus mehreren vorhandenen Feldern zusammengesetzt werden.
Da wir schon eine ID-Spalte geplant haben, nutzen wir natürlich die zweite Option und wählen in dem Auswahl-Feld den richtigen Feld-Namen aus.
Eine Erstellung des Autowertes geben wir ebenfalls an.
Da wir dass schon weiter vorne festgelegt haben, kann die Eintragung hier eigentlich übergangen werden, aber sicher ist sicher.

Es folgt das Abspeichern der Tabellen-Definition innerhalb der Datenbank-Datei (eingekapselt).

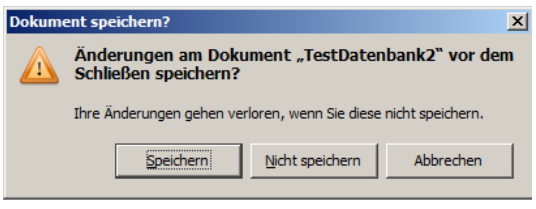
Wer will, kann jetzt sofort zum Eingeben der Daten wechseln (s.a. → 3.1.5.1.3. Eingeben von Daten in eine Tabelle).
Nacharbeiten für besondere Konstrukte sind über die zweite Vorgehens-Option möglich. Dazu findet man noch ein paar Hinweise im Kapitel → 3.1.5.1.3. Ändern des Relationsschema einer Tabelle – Korrekturen am Entwurf.

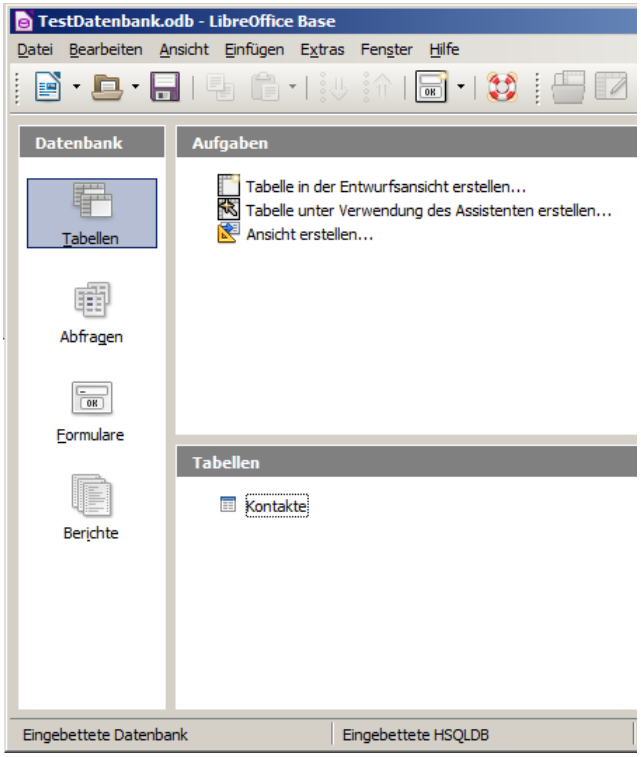


Wählt man "Daten sofort eingeben", dann sieht man die konstruierte Tabelle in ihrer ganzen Pracht – nur eben ohne Daten.
Liegen die Daten der Tabelle in Papier-Form vor, dann bleibt i.A. nur der händische Weg der Daten-Eingabe.



Die gleiche Ansicht erhält man, wenn im Datenbank-Menü auf die betreffende Tabelle klickt.
Es gibt verschiedene Möglichkeiten Daten aus strukturierten Dateien zu importieren. Dazu eignen sich CSV-, TXT-, XML- und JSON-Dateien.
Obwohl wir vielleicht noch gar keine Daten eingegeben haben, werden wir am Schluss zum Abspeichern aufgefordert. Dieses Speichern bezieht sich auch gar nicht auf Daten – wir werden noch sehen darum müssen wir uns überhaupt nicht mehr kümmern – sondern auf die vorgenommenen Definitionen sowie die Format-Einstellungen und -Veränderungen in unserer Datenbank.



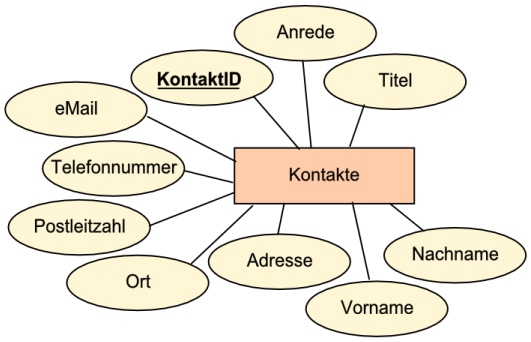


Aufgaben:

1. ***Vollziehen Sie den Weg zur Erstellung einer Tabelle mit dem Assistenten nach!***
2. ***Erstellen Sie mit Hilfe der Assistenten in einer neuen Datenbank ("CD-Sammlung") eine Tabelle für Ihre eigene CD-, Album-/MP3-Sammlung! (Benutzen Sie mindestens 7 Attribute!)***

3.1.5.1.1.1. Rück-Ableitung eines Entity-Relationship-Diagramms

Wenn man nun so eine Tabelle mit dem Assistenen erstellt hat oder auch vielleicht von wo anders her bezogen hat (z.B. aus einer externen Datenbank), dann benötigt man vielleicht für eigene Weiterentwicklungen wieder das ERD. Die Ableitung ist denkbar einfach. Der Tabellenname ist der Entitäts-Typ. Die Spalten-Überschriften entsprechen dem Relationsschema und stellen somit die Attribute dar. Das Schlüssel-Attribut ist ebenfals schnell aus dem Primär-Schlüssel abgeleitet.

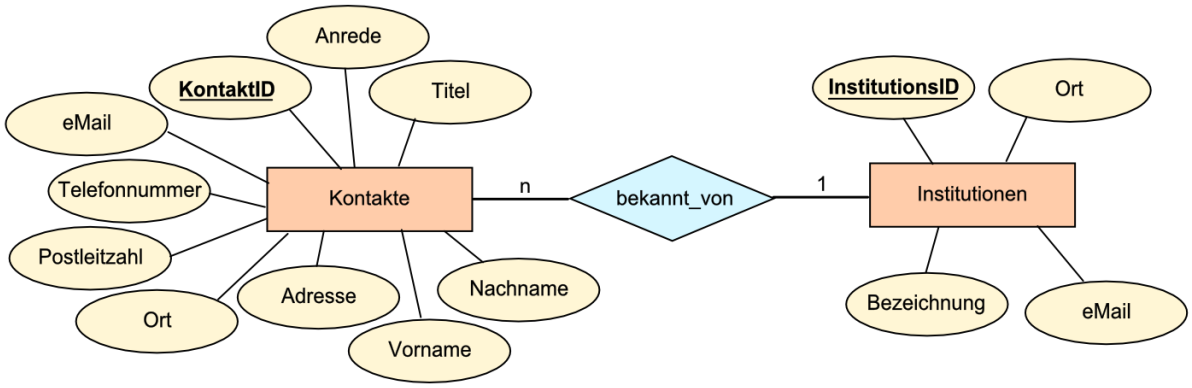


3.1.5.1.2. Erstellen einer Tabelle in der Entwurfs-Ansicht

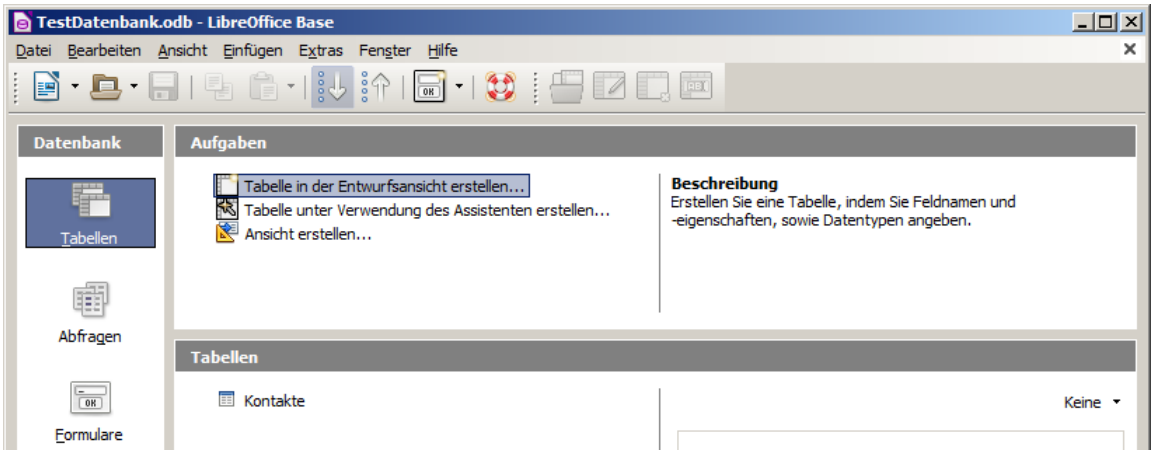
Der Assistent bietet aber nur begrenzte Möglichkeiten. Wir tun hier mal so, als könnten wir die nächste Entwicklungsstufe unserer Beispiel-Datenbank nicht mit dem Tabellen-Assistenten realisieren.
Gewünscht soll eine Erweiterung der Kontakte-Datenbank sein, in der die Institutionen (Schule, Verein, …) für die Kontakte gespeichert sind.

böse Frage zwischendurch:
Ist die bisherige Konstruktion unserer "Datenbank" wirklich schon eine Datenbank oder ein Datenbank-System?

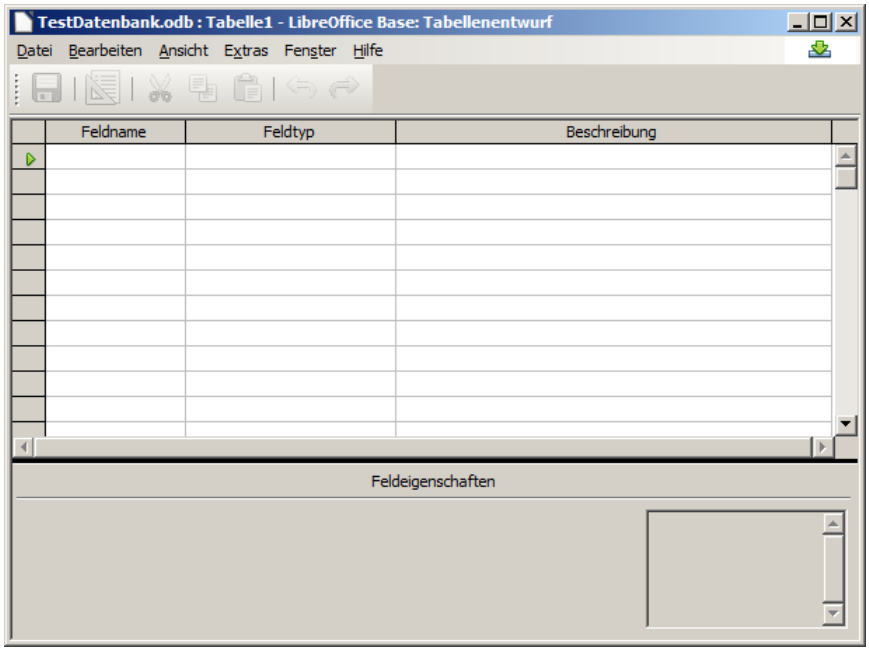
Das ERD für diese Miniwelt könnte dann so aussehen:



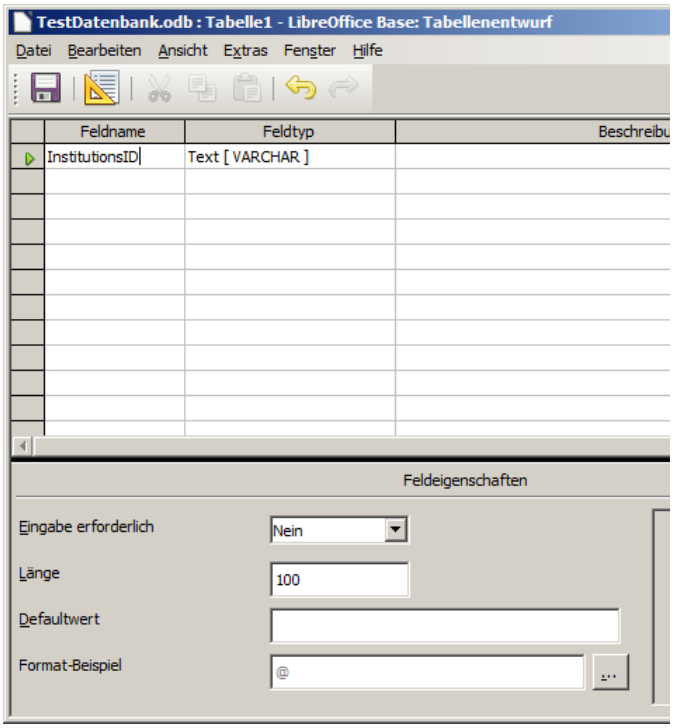
Zumindestens die Tabelle "Institutionen" können wir mit unseren Kenntnissen schon erstellen. Als Herausforderung erstellen wir die Tabelle eben nicht mit dem Assistenten, sondern "manuell". Der Aufgaben-Punkt dafür lautet "Tabelle in der Entwurfsansicht erstellen …".



Offensichtlich benötigen wir für jedes Feld (= Attribut) ersteinmal genau drei Angaben, den Namen, einen Typ und eine Beschreibung.
Die Beschreibung ist optional und dient nur der Dokumentation und Kommentierung (ev. kryptischer Feldnamen / Attribute).

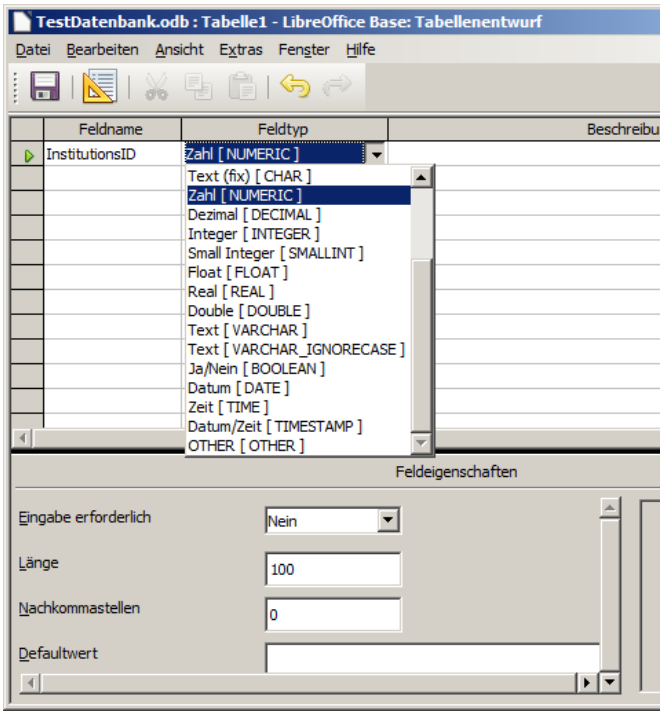


Für jede zukünftige Spalte unserer neuen Tabelle müssen wir hier jetzt eine Zeile anlegen.
Zuerst geben wir einen Feldnamen (Attributs-Namen) an. Achten Sie auf exakte Schreibung. Buchstaben-Dreher sind hier immer Stolpersteine, wenn dann später Daten aus der "falsch" geschriebenen Spalte geholt werden sollen.
Als Datentyp schlägt BASE immer Text vor. Für einfache Zwecke reicht das.
Zumindestens bei der Spalte des zukünftigen Primär-Schlüssels nehmen wir aber eine Korrektur vor.

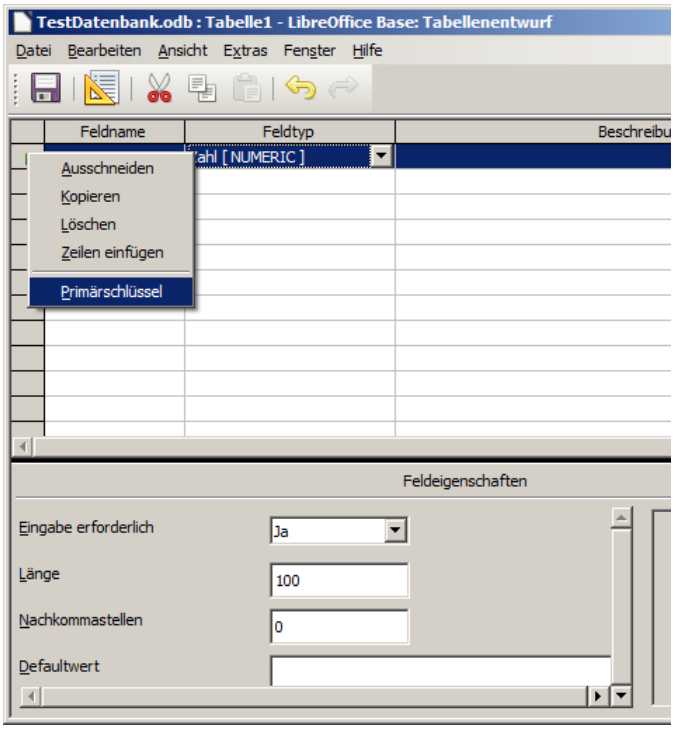


Für die zukünftige Primärschlüssel-Spalte ändern wir den Feldtyp (Datentyp) auf Zahl. Unten erscheinen dann die speziellen Möglichkeiten für Optionen.

Eine passgenaue Auswahl der Daten-Typen zu unseren Daten ist später immer dann von Vorteil, wenn diese Daten weiter verarbeitet werdn soll. In vielen DBMS gibt es für diverse Spezial-Funktionen, um z.B. Zeit-Angaben zu verrechnen.
Kaum jemand, der es schon mal gemacht hat, wird freiwillig die Differenz zwischen zwei Zeitwerten über eine eigenen Funktion berechnen wollen.



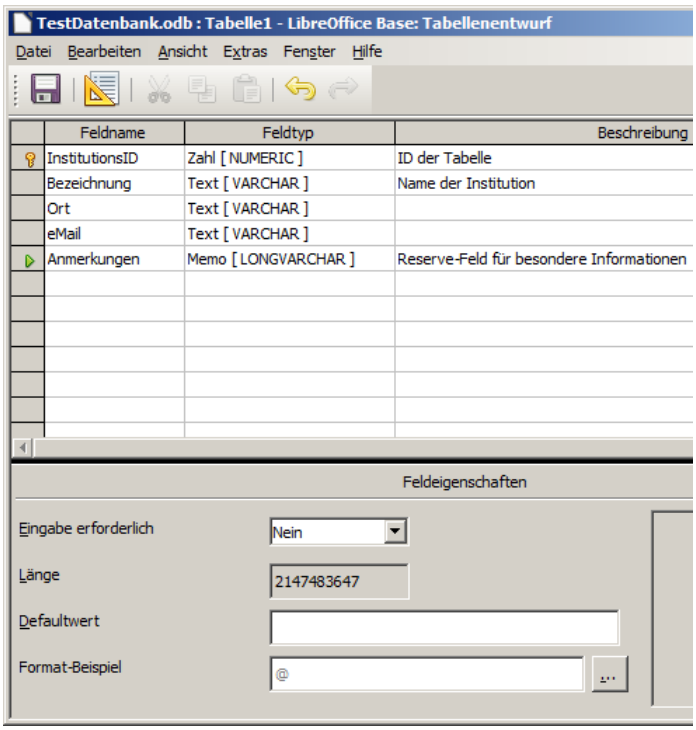
Auch wenn es an dieser Stelle noch nicht unbedingt notwendig ist, legen Sie gleich den Primärschlüssel fest.
Dazu klicken wir rechts auf die Feld-Zeile und wählen den passenden Kontextmenü-Punkt aus.



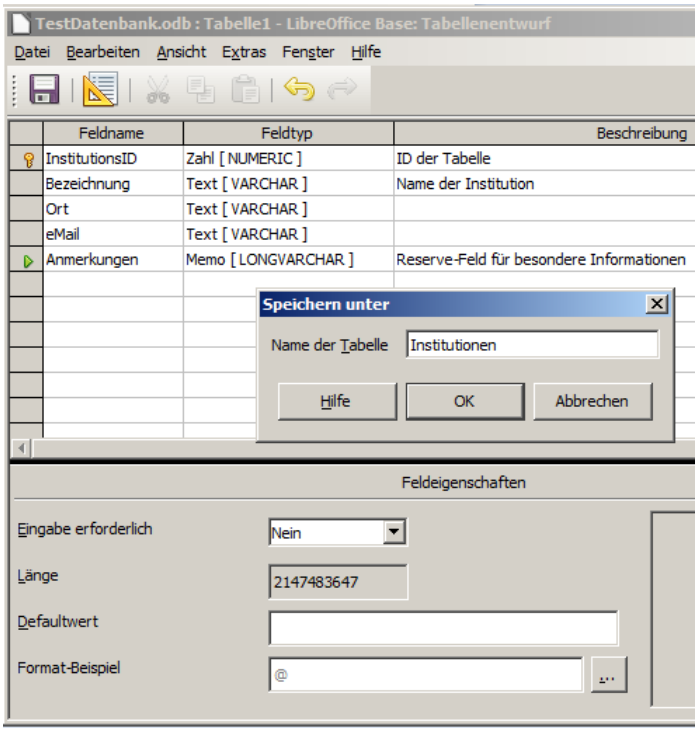
Aus Erfahrung weiss ich, wie oft man das sonst am Ende der Tabellen-Definition vergisst. Darum legt BASE eine eigene Schlüssel-Spalte an und man muss wieder nacharbeiten, oder seine Daten-Modelle verändern.

Der kleine gelbe Schlüssel vor der Attribut-Definition (Feld-Definition) zeigt eine erfolgreiche Zuordnung an.

Ist die gesamte Definition vorgenommen worden, dann könnte man eigentlich die Entwurfs-Ansicht verlassen.
In der Praxis kommt es fast immer vor, das irgendwann mal zusätzliche Daten zu einem Objekt auftauchen. Vielleicht ist es nur der Hinweis, dass dieser Datensatz noch korrigiert / überprüft werden muss oder ähnliches. Für solche Fälle empfehle ich, immer ein zusätzliches Feld "Anmerkungen" in die Tabellen-Definition aufzunehmen. Wählt man für diese Feld den Datentyp "Memo", dann wird fast nur soviel Speicherplatz verbraucht, wie Daten im Feld notiert sind (s.a. → Daten-Typen unter → 3.1.4.1. Tabellen).
Leere Memo-Felder nehmen fast keinen Speicherplatz weg.
Kommen bei vielen Objekten einer Tabelle ähnliche Daten zusammen, dann sollte man natürlich über eine zusätzlich zu definierende Spalte nachdenken.



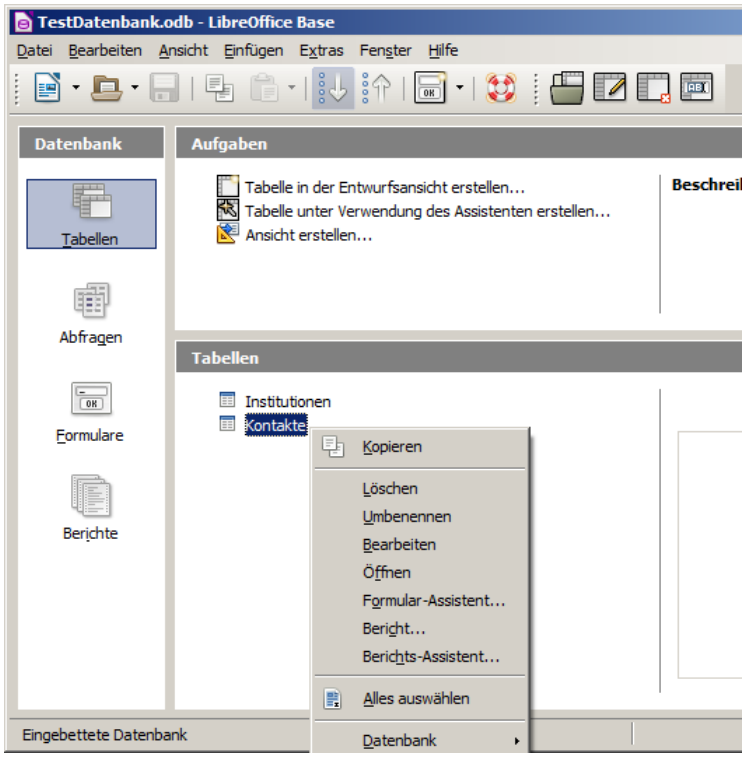
Tabellen ohne schon festgelegtem Namen müssen bei Verlassen der Entwurfs-Ansicht noch mit einem Namen versehen werden.



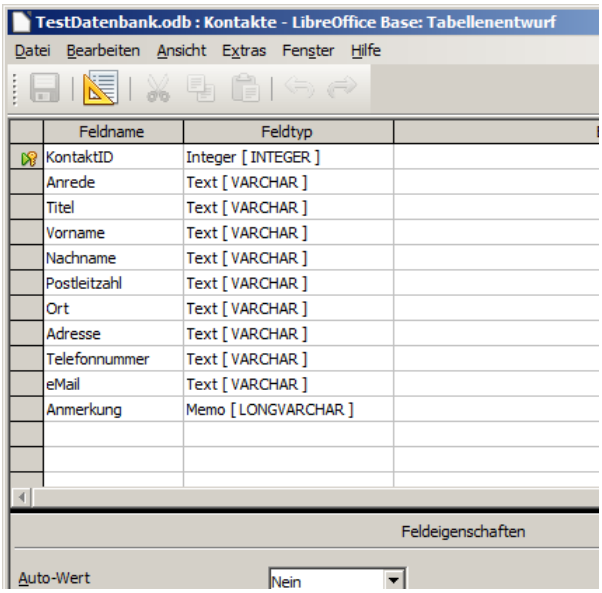
3.1.5.1.3. Ändern des Relationsschema einer Tabelle – Korrekturen am Entwurf

Wollen wir eine Tabellen-Defintion nachträglich ändern, dann geht das über das Kontext-Menü zur entsprechenden Tabelle. Der Menü-Punkt "Bearbeiten" ist vielleicht etwas verwirrend. Was bearbeiten wir, die Tabellen-Inhalte oder die Tabellen-Definition?

Es ist die Tabellen-Definition – also kommen wir zur Entwurfs-Ansicht.
Änderungen an vorhandenen Felder müssen – zumindestens, wenn schon Daten eingetragen worden – sehr sehr vorsichtig vorgenommen werden. Da droht Daten-Verlust. Lieber neue Felder anlegen, die Daten gezielt und kontrolliert ins neue Feld übertragen und dann erst das akte Feld löschen.

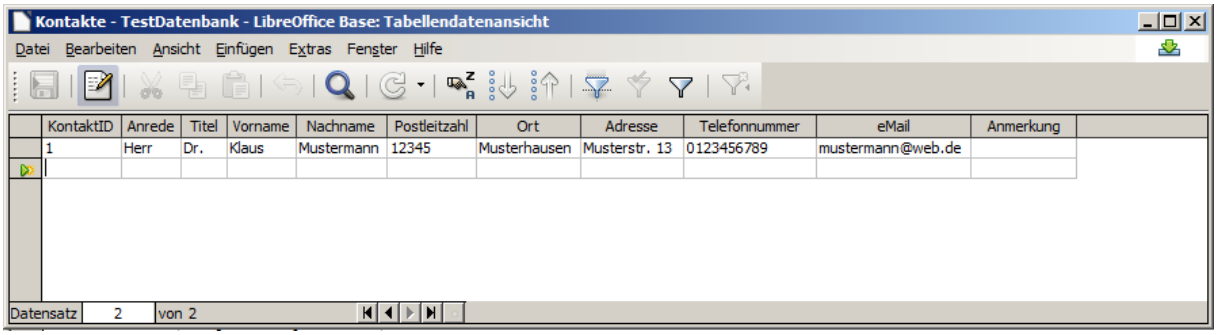


Das Hinzufügen neuer Felder ist i.A. unproblematisch.
So ergänzen wir die "Kontakte"-Tabelle um eine "Anmerkung"s-Spalte, so wie wir das bei der Institutions-Tabelle besprochen haben. Als Typ wählen wir das übliche Memo-Format.
Jede Veränderung an der Tabellen-Definition erfordert ein Speichern beim Verlassen der Entwurfs-Ansicht.
Machen wir das nicht, sind die Änderungen an der Tabellen-Struktur auch nicht permanent.



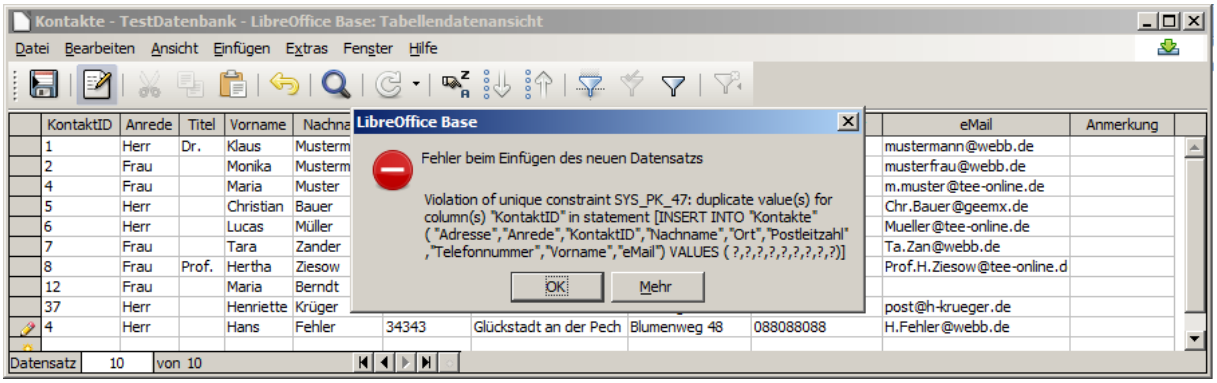
3.1.5.1.4. Eingeben von Daten in eine Tabelle

Klickt man in die zweite Zeile oder wechselt man z.B. mit [Tab] aus dem ersten Datensatz (1. Zeile) dorthin, dann ergeben sich zwei unscheinbare Effekte:

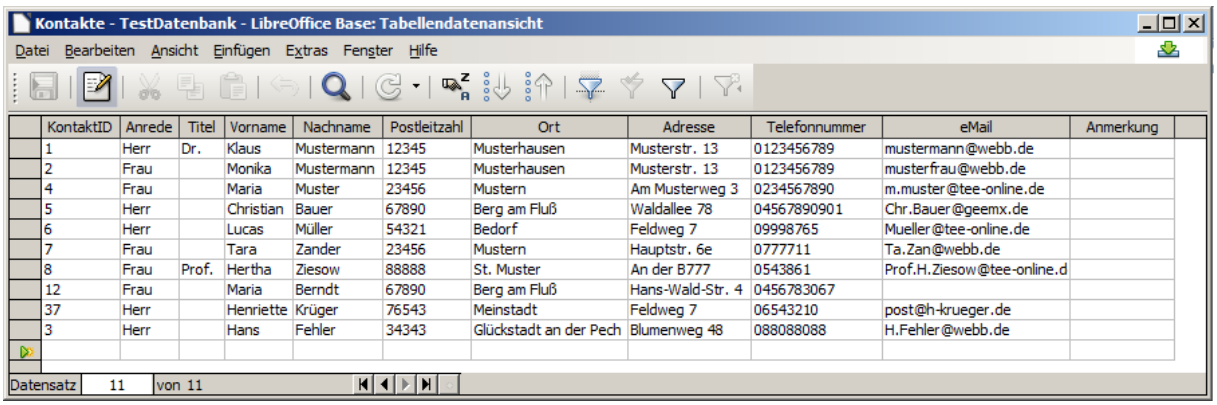


Der Stift – für den Eingabe-Modus – verschwindet und das aktivierte "Speichern"-Symbol (Diskette in der Symbolleiste) wird blaß. Beides bedeutet, dass die Daten - der gesamte Datensatz – wurde abgespeichert. Darum muss man sich in Datenbank-Programmen nicht kümmern.

Trotzdem kann es passieren – z.B. in ms ACCESS – dass man trotzdem zum Speichern aufgefordert wird. Dieses Speichern bezieht sich auf ev. Layout-Änderungen. Speichert man nicht, dann sind die Daten genau wie eingegeben gespeichert, nur das Layout wird nicht aktuallisiert.
Zum Daten-Eingeben reicht sonst auch einfach der Doppel-Klick auf die gewünschte Tabelle in der Datenbank-Übersicht.
Zu einer kritischen Situation kommt es, wenn man versucht beim Primär-Schlüssel einen doppelten Wert einzugeben. Die Fehler-Meldung ist "super" informativ:



Hier den doppelten Schlüsselwert als Fehler-Ursache zu identifizieren ist wohl nur Datenbank-Neerds zuzutrauen.
Fertig eingegeben – und korrigiert - könnte die Kontakt-Tabelle dann so aussehen.



KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerkung
1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	0123456789	mustermann@webb.de	
2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	0123456789	musterfrau@webb.de	
4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	0234567890	m.muster@tee-online.de	
5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	04567890901	Chr.Bauer@geemx.de	
6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	09998765	Mueller@tee-online.de	
7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	0777711	Ta.Zan@webb.de	
8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	0543861	Prof.H.Ziesow@tee-online.d	
12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	0456783067		
37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	06543210	post@h-krueger.de	
3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	088088088	H.Fehler@webb.de	

Aufgaben:

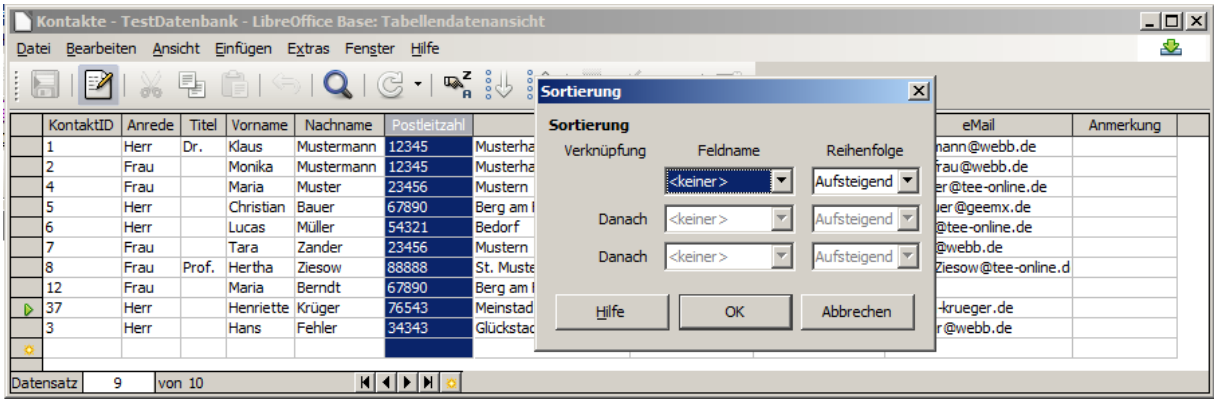
1. *Geben Sie die Daten aus der obigen Tabelle in die Tabelle "Kontakte! ein!*
2. *Erstellen Sie die Tabellen-Struktur der Tabelle "Institutionen" wie oben angegeben!*
3. *Füllen Sie die Tabelle "Institutionen" mit den folgenden Daten!*

InstitutionsID	Bezeichnung	Ort	eMail	Anmerkungen
1	Goethe-Gymnasium	Mustern	GoeGymn@webb.de	
2	Tanzverein "Polka"	Musterhausen	post@tv-polka.de	
3	Hilfe e.V.	Meinstadt	info@hilfe.info	
4	Traditionsverein	Cedorf	tradi@verein.de	
5	Let's share	Meinstadt	letsshare@verein.de	
6	Grundschule	Mustern	gs-mustern@webb.de	

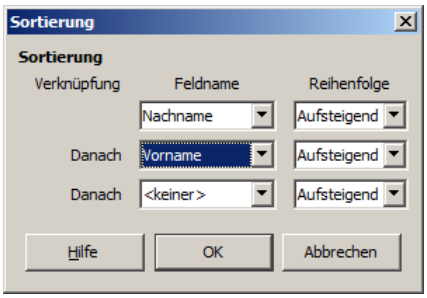
3.1.5.1.5. Filter und Sortierungen in Tabellen

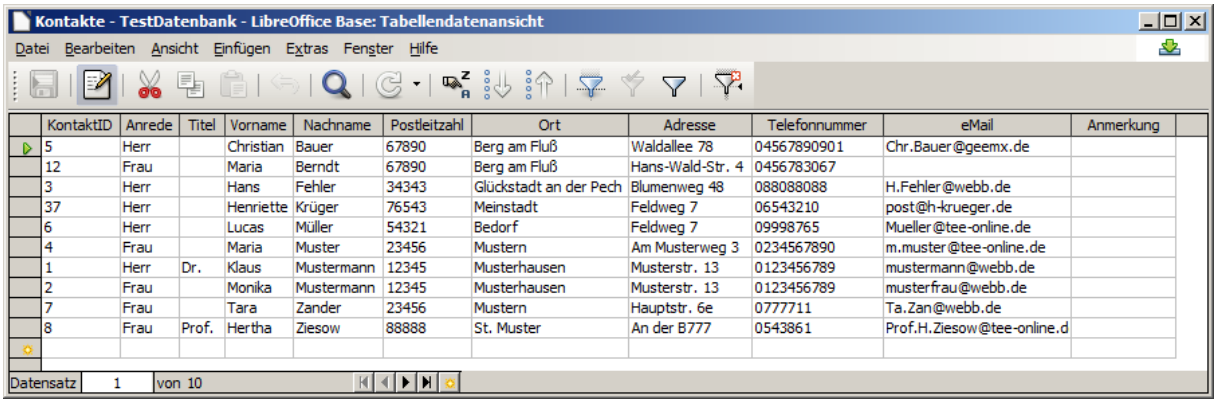
Bei großen Tabellen verliert man u.U. schnell mal die Übersicht, ob das Eine oder Andere schon eingegeben wurde. Hier wäre es schön, wenn man sich nur Teile der Tabelle heraus-picken könnte oder die Werte in einer bestimmten Reihenfolge angezeigt werden.
Das Auswählen bestimmter Teile einer Tabelle nennt man Filtern, ein in die Reihenfolge bringen wird als Sortieren bezeichnet.
Markiert man eine Spalte, dann kann diese schnell mittels der Pfeil-Schaltfächen aus der Symbolleiste realisiert werden.

Für mehrstufige oder komplexere Sortierungen gibt es eine andere Möglich-keit. Egal, ob man eine Spalte markiert oder nicht, mit dem Sortieren-Symbol (A-Z-Symbol) gelangt man immer zu einem Assistenten.

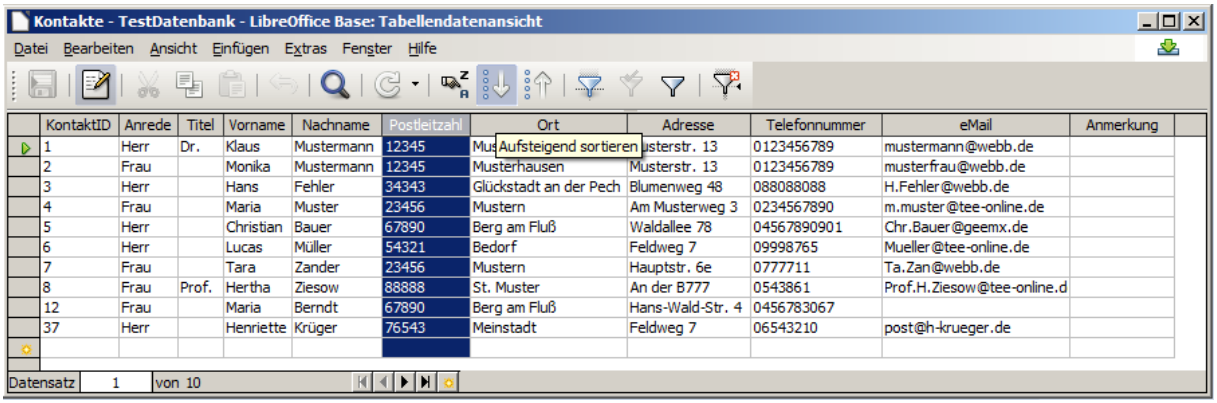


In diesem Sortierung's-Dialog lassen sich drei einzelne Sortierungen einrichten, die dann von oben nach unten abgearbeitet werden.
Im Beispiel (Abb. rechts) ist die klassische Sortierung zuerst nach dem Nachnamen und dann nach den Vor-namen eingerichtet.
Das Ergebnis ist wenig überraschend. An der Daten-Tabelle an sich ändert die Sortierung gar nichts. Es ist eine reine Darstellungsfrage und eigentlich schon einen Art Abfrage.





Will man nur nach einer Spalte sortieren, dann reicht das Markieren der Spalte und dann die Auswahl der Sortier-Richtung in der Symbolleiste.



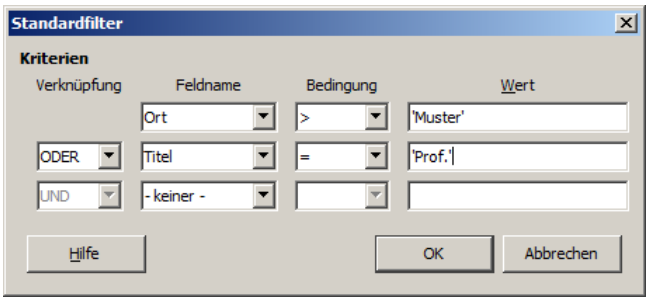
KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerkung
1	Herr	Dr.	Klaus	Mustermann	12345	Mus	usterstr. 13	0123456789	mustermann@webb.de	
2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	0123456789	musterfrau@webb.de	
3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	088088088	H.Fehler@webb.de	
4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	0234567890	m.muster@tee-online.de	
5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	04567890901	Chr.Bauer@geemx.de	
6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	09998765	Mueller@tee-online.de	
7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	0777711	Ta.Zan@webb.de	
8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	0543861	Prof.H.Ziesow@tee-online.d	
12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	0456783067		
37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	06543210	post@h-krueger.de	

Will man dann wieder den Grundzustand bzw. den letzten Filter- bzw. Sortier-Zustand, dann können die aktuellen Sortierungen und / oder Filterungen mit dem rot-geïxten Trichter-Symbol zurückgesetzt werden.

Filterung

Die erste Schaltfläche ("Autofilter") steht für eine Filterung nach dem markierten Begriff / Eintrag in der Tabelle. Ist eine Filterung aktiviert worden, dann wird auch das zweite Symbol ("Filter anwenden") nutzbar. Mit ihm kann zwischen gefilteter Ansicht und der vollständigen Tabelle gewechselt werden.

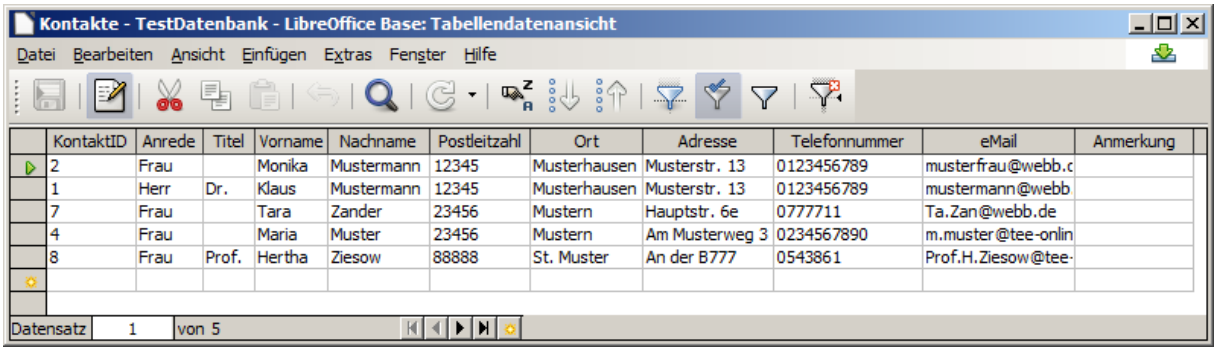
Hinter dem einfachen Trichter-Symbol (Standardfilter) versteckt sich ein kleiner Assistenten, der uns ein wenig an die mehrstufige Sortierung von oben erinnert.



Hier können wir nun Bedingungen für die Auswahl in freier Kombination eingegeben werden.

Als Ergebnis erhalten wir eine aufgekürzte Tabelle mit den betreffenden / gewünschten Datensätzen.

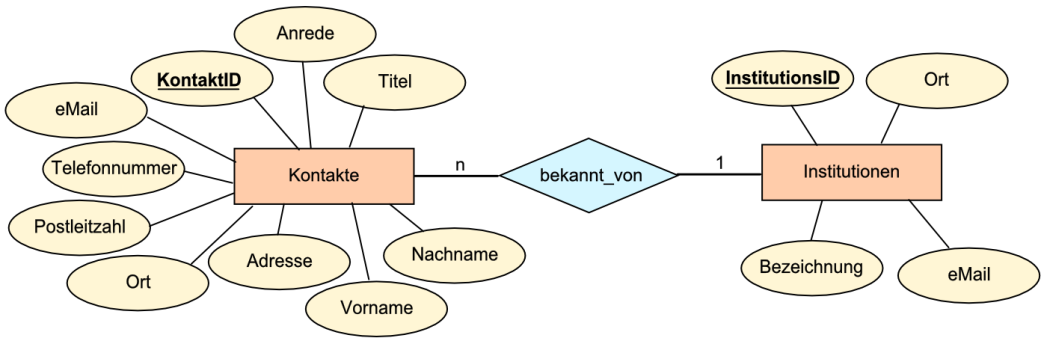
Es handelt sich praktisch um eine Selektion (→ Selektion / Restriktion).



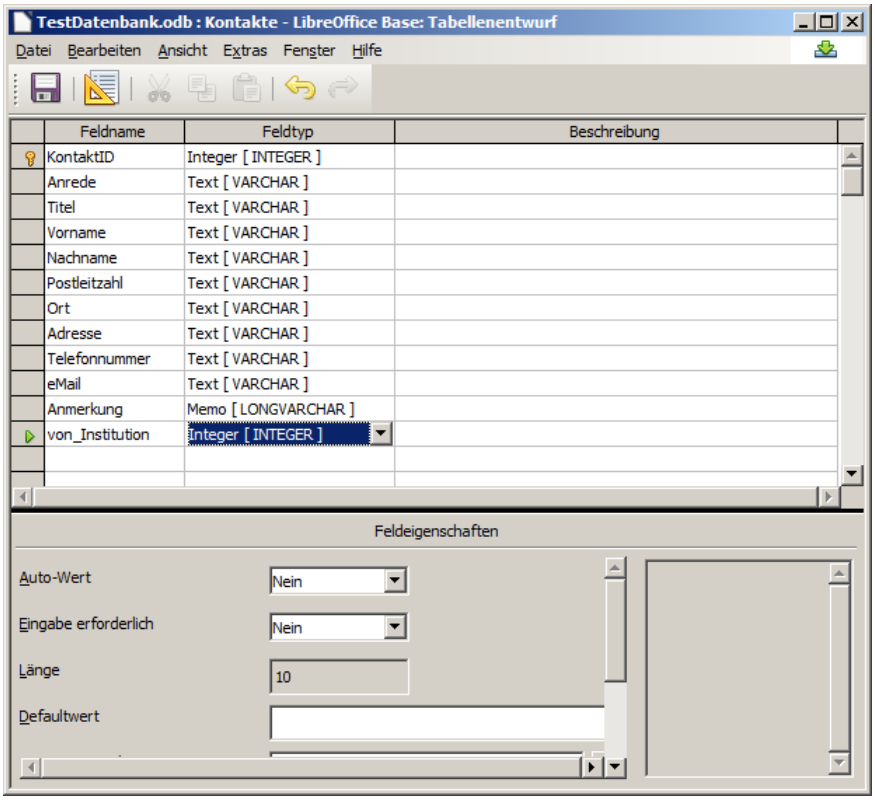
KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerkung
2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	0123456789	musterfrau@webb.c	
1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	0123456789	mustermann@webb	
7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	0777711	Ta.Zan@webb.de	
4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	0234567890	m.muster@tee-onlin	
8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	0543861	Prof.H.Ziesow@tee-	

3.1.5.1.6. Beziehungen zwischen Tabellen umsetzen

Die Tabellen "Kontakte" und "Institutionen" sind die Basis für die Beziehung "bekannt_von". In diese Tabelle bzw. die zusätzliche Spalte in einer der Basis-Tabellen werden nun keine echten Daten (Klar-Bezeichnungen) eingetragen, sondern nur Verweise auf die (beiden) aggregierte(n) Tabelle(n). Und was eignet sich dabei besser, als die Primär-Schlüssel der jeweils betroffenen Datensätze.
Solange wir noch keine "Formulare" zur Verfügung haben (→ 3.1.3.4. Formulare), ist die Dateneingabe (der Primär-Schlüssel) noch sehr mühselig.



Nach den Transformations-Regeln (→ 3.0.4. Transformation eines ERM (ERD) in eine relationale Datenbank) müssen wir an die Tabelle Kontakte ein Spalte "bekannt_von" oder so ähnlich anhängen, und diese dann passend mit den Primär-Schlüsseln der anderen Tabelle füllen.
Dazu wechseln wir in die Tabellen-Entwurfsansicht ("Bearbeiten" aus dem Kontext-Menü). Hier fügen wir dann an passender Stelle oder hinten an, ein neues Feld ein. Als Datentyp muss einer gewählt werden, der dem Primärschlüssel der anderen Tabelle entspricht.
Das Auffüllen mit den zugehörigen Daten ist dann mehr eine Finger-Übung.
Später kann man dafür dann auch spezielle Formulare nutzen, welche die Institutionen im Klarnamen angeben und dann die Umsetzung in den passenden Primär-Schlüssel selbstständig übernehmen.



3.1.4.1.6.1. Erstellen einer einfachen Beziehung zwischen zwei Tabellen

Was wir jetzt noch brauchen ist die Verbindungsliste zwischen den beiden Entitäts-Typen:
Als Erstes müssen wir ermitteln, um welchen Verknüpfungs-Typ es sich handelt.
In der Tabelle rechts sind die Beziehungen in Menschen-verständlicher Form zusammengetragen.
Zur Wahl stehen 1 : 1, 1 : n bzw. n : 1 oder eben n : m.
Da die Datensätze bei Kontakte immer nur einfach vorkommen (werden) und die Institutionen hier mehfach genannt sind, handelt es sich um eine "1 : n"-Beziehung.

Daten aus Tabelle …

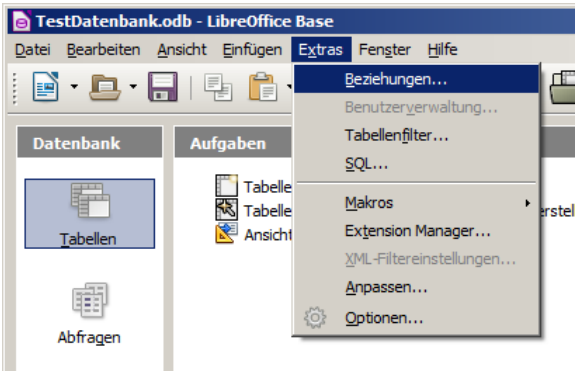
Kontakte		Institution
Name	**Vorname**	**bekannt_von**
Mustermann	Klaus	Goethe-Gymnasium
Mustermann	Monika	Tanzverein
Muster	Maria	Traditionsverein
Bauer	Christian	Let's share
Müller	Lucas	Goethe-Gymnasium
Zander	Tara	Grundschule
Ziesow	Hertha	Traditionsverein
Berndt	Maria	Hilfe e.V.
Krüger	Henriette	Goethe-Gymnasium
Fehler	Hans	Goethe-Gymnasium

Natürlich könnte man auch für eine Person mehrere Institutionen als Kontakt konstruieren. Dann wäre es eine "n : m"-Beziehung. Das geben unsere Daten aber nicht her. Trotzdem besprechen wir diese Möglichkeit weiter hinten.
Wie nun die Abbildung der Beziehungs-Typen in Relationalen Datenbanken erfolgt, haben wi schon vorgestellt (→ 2.1.6. Transformation eines ERM (ERD) in eine relationale Datenbank).

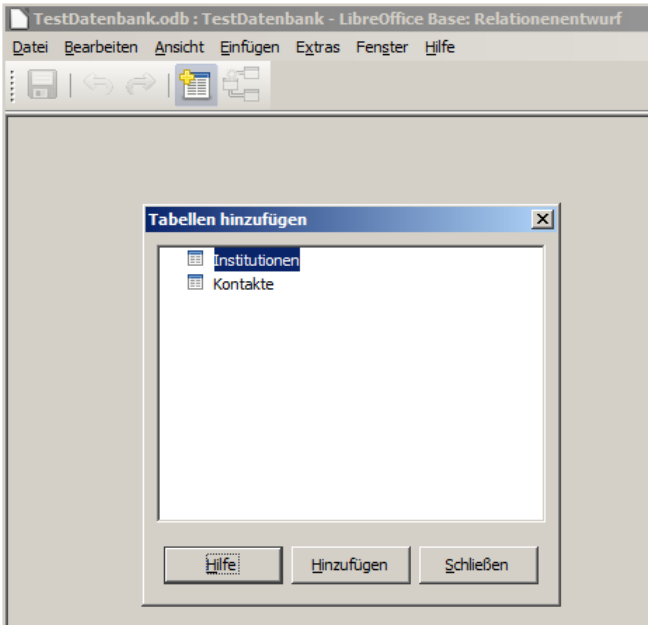
Aufgaben:

1. *Stellen Sie in einer Papiertabelle die Beziehung zwischen den Tabellen Kontakte und Institutionen über Fremdschlüssel her!*
2. *Ergänzen Sie eventuell fehlende Daten für einen Datenbank-Einsatz!*

Mit der obigen Aktion haben wir nun Daten-technisch ein Verbindung hergestellt, wir wissen um die Zusammenhänge – nicht aber das System.
Zwar könnte man jetzt schon spezielle Abfragen erzeugen (→ 3.1.5.2. Abfragen), die auf unserem Hintergrundwissen zu den Tabellen-Beziehungen basieren, aber wir wollen ja mehr dem DBS die Arbeit aufnacken. Außerdem würden fremde Nutzer kaum alle Beziehungen kennen können. Sie müssten sich dazu mit der Struktur der Datenbank beschäftigen. Diese ist ihnen aber nicht unbedingt zugänglich.
Das Erstellen der Beziehungen erfolgt graphisch und quasi Assistenten-gestützt.
Wir wählen dazu aus dem "Extras"-Menü den Punkt "Beziehungen …) aus.

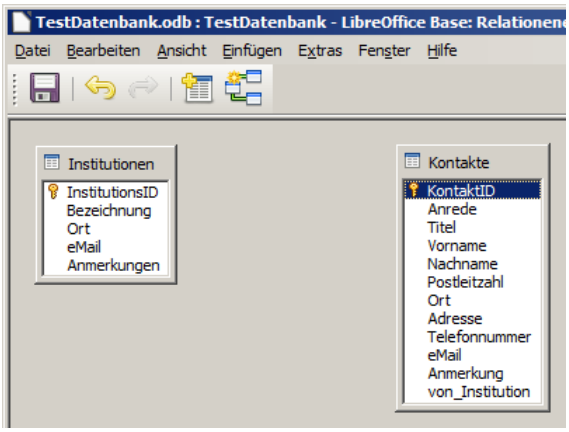


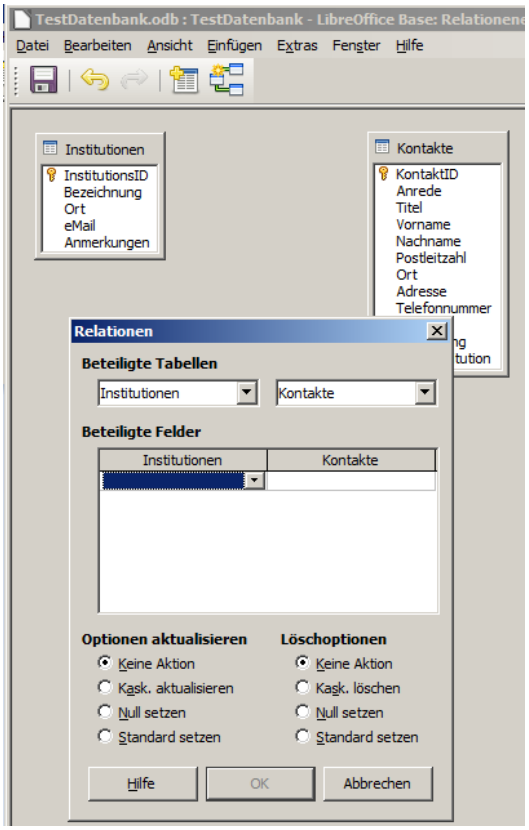
Es erscheint eine Arbeitsfläche (Relationsentwurf), auf der die Tabellen und ihre Beziehungen untereinander graphisch dargestellt werden können.
Hier sind dann später auch Änderungen möglich. Diese sollten aber frühzeitig erfolgen, da sonst Inkonsistenzen mit Abfragen usw. auftreten können, die auf "veraltete" bzw. "fehlerhafte" Beziehungen beruhen.
Zuerst müssen wir die benötigten Tabellen zur graphischen Arbeitsfläche "Hinzufügen".
Dabei sollte man zuerst einmal so sparsam wie möglich hinzufügen. Die Übersichtlichkeit kann sonst stark leiden.
Natürlich müssen aber später alle (für die aktuelle Aufgabe) benötigten Tabellen hinzugefügt werden.



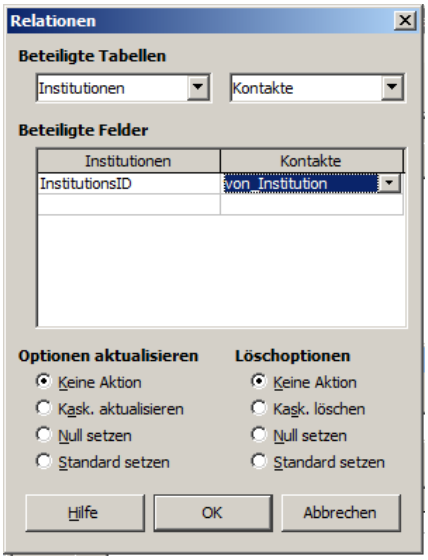
Sollten auf dem Relationsentwurf Tabellen fehlen, dann lassen sich diese mit der Schaltfläche "Tabelle hinzufügen".

Die Tabellen-Struktur-Tabellen (Tabellen-Bildchen) können frei auf der graphischen Oberfläche angeordnet werden. Schon vorhandene Beziehungen werden beim Umsortieren mit verschoben.
Über die Schaltfläche "Neue Relation" kann nun eine neue Beziehung eingerichtet werden.



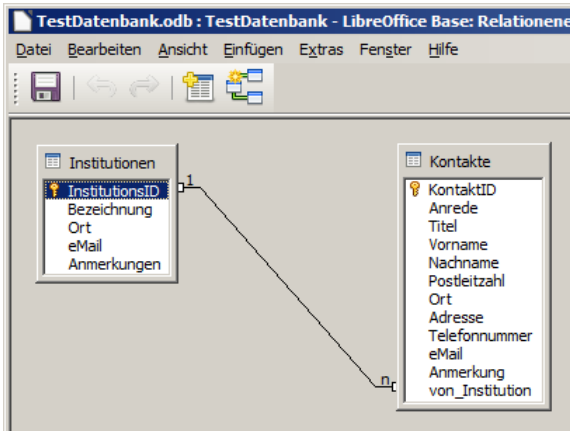


Zuerst wählt man die beteiligten Tabellen und dann die Felder aus. Die Datenliefernde Tabelle steht dabei links.
Nun werden die passenden Felder ausgewählt. Wie üblich erscheint der Auswähler erst, wenn man in die Zelle klickt.
Die verfügbaren Optionen lassen wir erst einmal unberührt. Sie können später geändert werden.



Die Optionen lassen wir vorerst unberührt. Die Konsequenzen der einzelnen Möglichkeiten müssen wir uns sehr gut überlegen.

Als Ergebnis erhalten wir eine Übersicht, die schon gut, die von uns erdachte, Beziehungen aus dem ERD abbildet.



Eine nicht-gebrauchte Beziehung (Relation) lässt sich durch Anklicken und drücken der "Entf"-Taste löschen. Bitte hier mit Bedacht vorgehen, denn die Relationen sind auch dann weg, wenn man versucht den Relationsentwurf ohne Abspeichern zu Schließen!

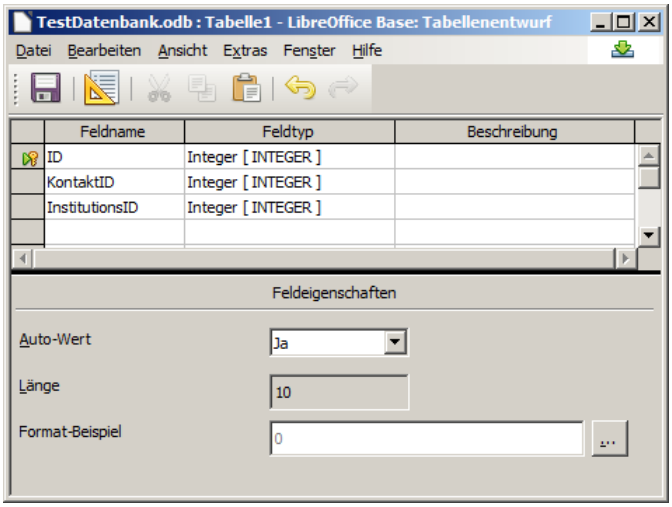
Aufgaben:

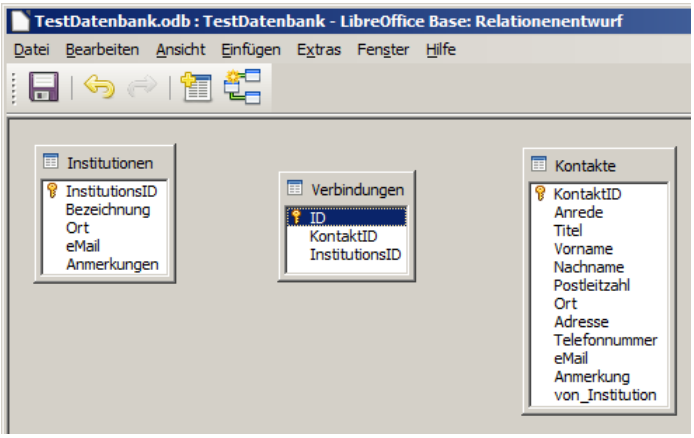
1. *Überlegen Sie sich, wie die Beziehung realisiert werden müsste, wenn es sich um eine "n : m"-Beziehung handeln würde! Bereiten Sie eine passende Realisierung vor! Diskutieren Sie Ihren Vorschlag mit anderen Kurs-Mitgliedern!*

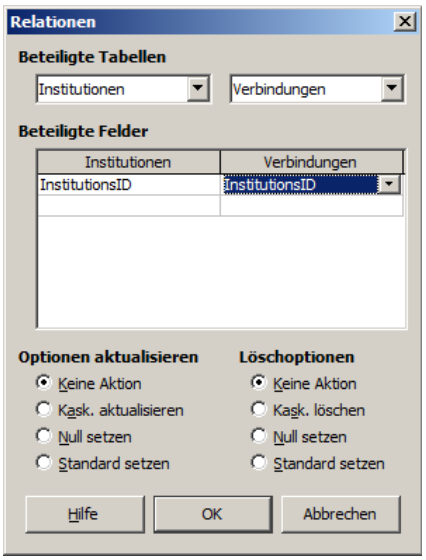
3.1.4.1.6.2. Erstellen einer "n : m"-Beziehung

Für "n : m"-Beziehungen brauchen wir ja eine (neue) Zwischentabelle (Beziehungs-Tabelle) (s.a. → 3.0.4. Transformation eines ERM (ERD) in eine relationale Datenbank). Die "n : m"-Beziehung wird dann in zwei Beziehungen zerlegt, welche die Kardinalitäten zwischen den Daten-Tabellen und der Beziehungs-Tabelle darstellen.

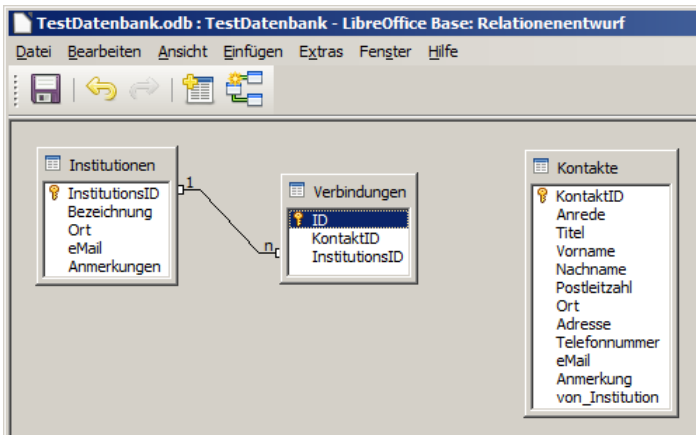
Nehmen wir an, wir wollten die "n : 1"-Beziehung zwischen "Kontakte" und "Institutionen" in eine "n : m"-Beziehung umwandeln, um alle Kontakt-Möglichkeiten abzubilden, dann könnte man etwa so vorgehen.
Zuerst wird die Beziehungstabelle erstellt und diese dann mit Daten gefüllt.
In dieser Umsetzung bleiben wir mal bei den ursprünglichen Beziehungen zwischen den Kontakten und den Institutionen, aus denen sie bekannt sind. Praktisch könnten wir jetzt beliebig viele weitere Kontakt-Beziehungen hinzugeben. Tara Zander könnte uns dann z.B. aus allen Institutionen bekannt sein.
Das Einfügen einer Tabelle in den Relationsentwurf sollte kein Problem mehr darstellen.
Im nächsten Schritt werden jetzt die Beziehungen zwischen den Daten-Tabellen und der Beziehungs-Tabelle hergestellt.
Auch das sollte für uns kein Hindernis mehr sein.



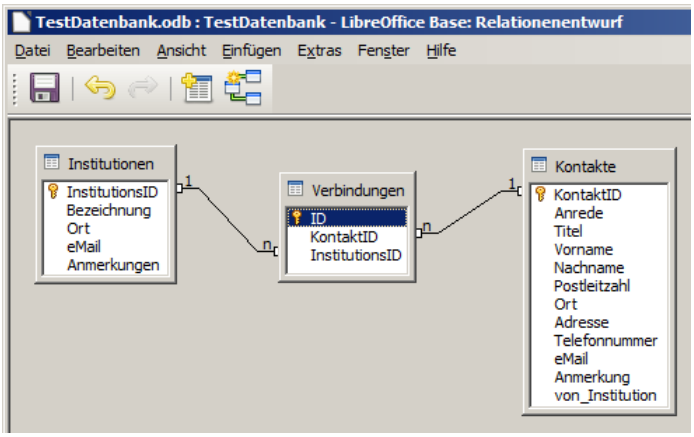




Das Ergebnis ist ebenfalls wenig überraschend (s. Abb. unten).



Die zweite Beziehung (Relation) wird genau so – wie vorne besprochen – erstellt.
Fertig ist unsere "n : m"-Beziehung. Aus der ursprünglichen "n : m"-Beziehung sind nun zwei "1 : n"-Beziehungen geworden.
Später könnte man z.B. die Tabelle "Verbindungen" um persönliche Verbindungsdaten, wie die individuelle eMail-Adresse für die konkrete Kontaktperson in dieser Institution, ergänzen.



Nun können neben den Erst-Kontakt-Beziehungen (gräulich unterlegt) auch noch diverse weitere Kontake auftauchen, so wie es in der Realität wohl wahrscheinlich ist:
Die Spalte "von_Institution" wird mit der n : m-Beziehung natürlich überflüssig und könnte / sollte entfernt werden, es sei den man will den Erst-Kontakt noch weiterhin als Information behalten.

Name	Vorname	bekannt_von
Mustermann	Klaus	Goethe-Gymnasium
Mustermann	Monika	Tanzverein
Muster	Maria	Traditionsverein
Bauer	Christian	Let's share
Müller	Lucas	Goethe-Gymnasium
Zander	Tara	Grundschule
Ziesow	Hertha	Traditionsverein
Berndt	Maria	Hilfe e.V.
Krüger	Henriette	Goethe-Gymnasium
Fehler	Hans	Goethe-Gymnasium
Berndt	Maria	Grundschule
Mustermann	Monika	Traditionsverein
Zander	Tara	Goethe-Gymnasium
Mustermann	Monika	Let's share
Mustermann	Klaus	Traditionsverein
Mustermann	Monika	Goethe-Gymnasium
Krüger	Henriette	Grundschule
Fehler	Hans	Hilfe e.V.

Aufgaben:

1. ***Setzen Sie nun den – im Gruppen-Gespräch – favorisierten Vorschlag (n : m-Beziehung) in BASE um! Nutzen Sie die Beziehungs-Informationen aus der obigen Tabelle!***

3.1.4.1.8. Erstellen von Tabellen mit SQL-Statement's

Die Erstellung von Tabellen und das Daten-Befüllen über SQL-Anweisungen ist möglich. Bei den von BASE gebotenen graphischen und Assistenz-basierten Möglichkeiten ist diese Variante eher zu aufwändig.
Anders sieht es aus, wenn man einen SQL-Text schon vorliegen hat oder diesen z.B. für einen universellen Einsatz auf unterschiedlichsten DBS schon vorbereitet hat. In diesem Fall ist eine SQL-basierte Definition und Daten-Eingabe eine echte Alternative.
Einige Datenbanken bieten einen SQL-Export ihrer Daten-Bestände an. Dies sind oft geeignete SQL-Texte für eine vollständige Einspielung in ein anderes DBMS.

3.1.5.2. Abfragen

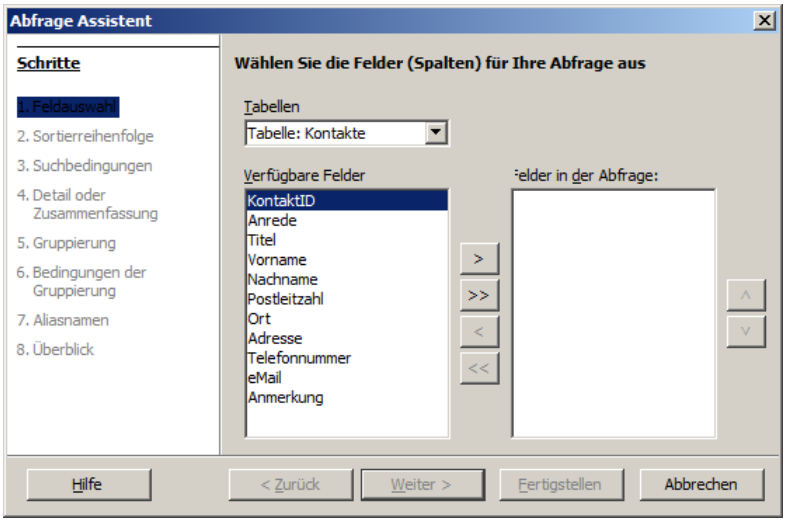
Die Daten-Menge großer Datenbanken übersteigen die Darstellungs-Möglichkeiten auf eine Bildschirm und die Verarbeitungs-Kapazität eines Menschen. Irgendwie müssen die Daten für die Anzeige reduziert werden. Auch gesetzliche Regelungen oder Verschwiegenheits-Vereinbarungen lassen nicht für jeden Nutzer einen Zugang zu allen Daten einer Datenbank zu. Über Selektionen (→ Selektion / Restriktion) und Projektionen (→ Projektion) werden die Daten auf das notwendige Maß zusammengestrichen und für die weitere Nutzung bereitgestellt. Die Auswahl der Daten und Bereitstellung in neuen (temporären) Tabellen nennt man Abfragen (Sichten / View's). Eine weitere Nutzung einer Abfrage kann dann in weiteren Abfragen, Berichten (→ 3.1.5.3. Berichte) oder Formularen (→ 3.1.5.4. Formulare) erfolgen.

3.1.5.2.1. Erstellen einer Abfrage mit dem Assistenten

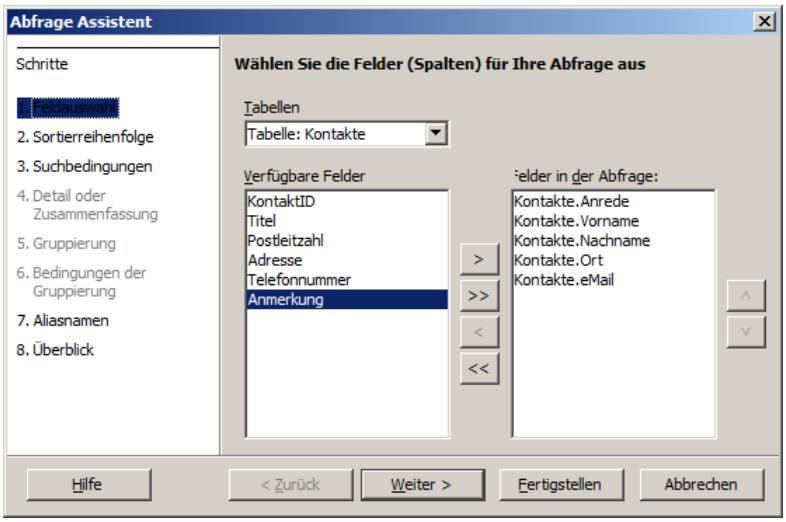
Für Anfänger ist die Assistenten-Methode wohl die einfachte Variante. Schrittweise klickt man sich die Elemente und Kriterien für die Abfrage zusammen. Man kann jederzeit wieder zurück bis hin zum Anfang, um notwendige Korrekturen vorzunehmen oder vernachlässigte Kriterien mit einzubauen.

Auswahl der Felder

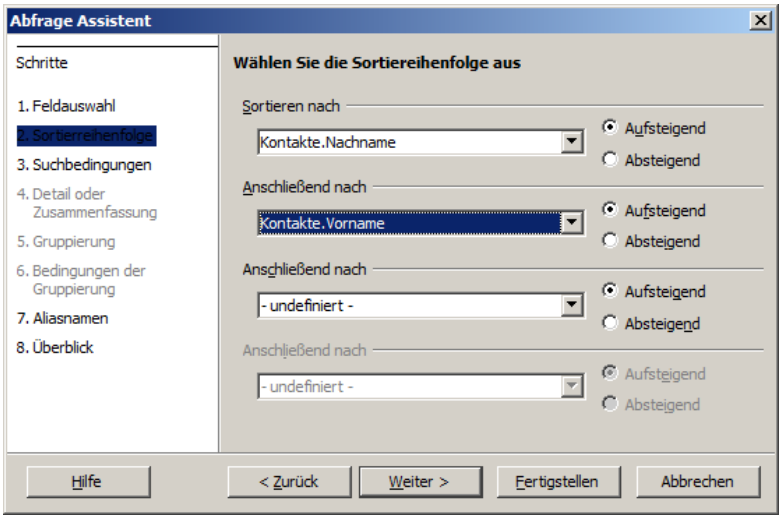
Durch die Wahl einzelner Felder aus der Gesamt-Tabelle beschränken wir also die Anzahl der bereitgestellten Spalten.
Hierbei handelt sich also praktisch um eine Projektion (→Projektion).



Bestätigung der Auswahl

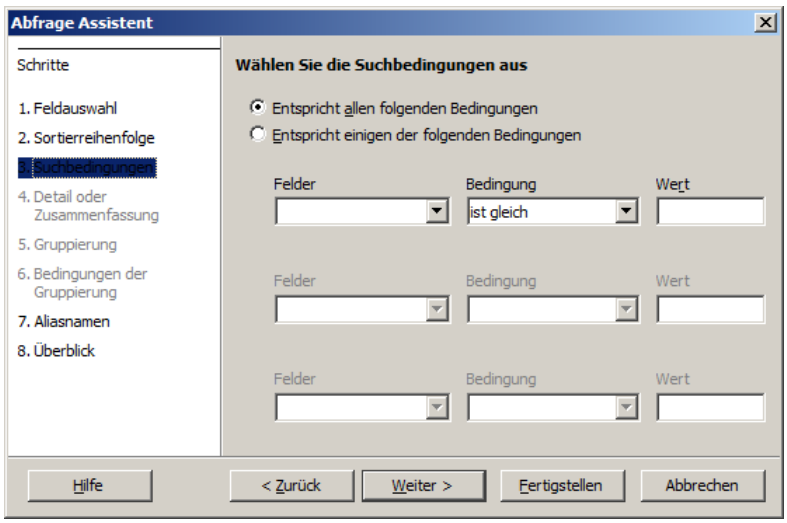


Sortierung



weitere Suchbedingungen

Durch weitere Auswahl-Kriterien beschränken wir die Anzahl der Ergebnis-Zeilen. Es werden nur noch ausgewählte Datensätze angezeigt.
Hier handelt es sich somit um eine Selektion (→Selektion / Restriktion).

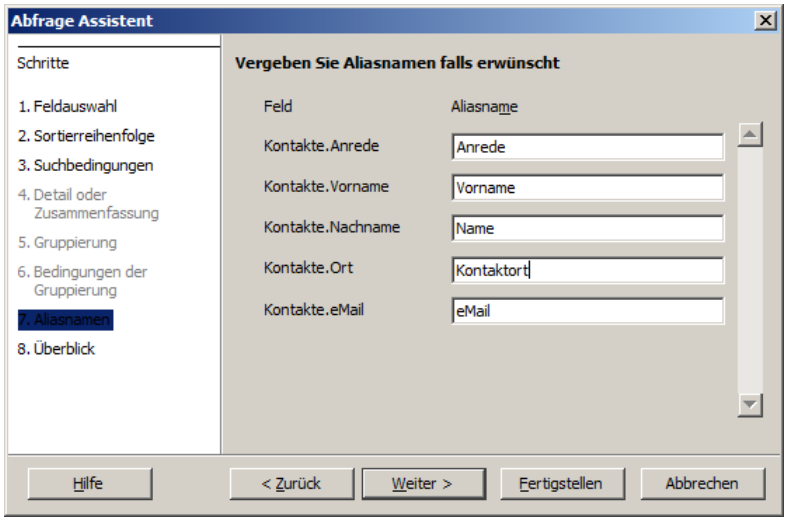


Wenn wir vorne schon eine Auswahl der Felder vorgenommen haben, dann liegt insgesamt eine Kombination aus Selektion und Projektion vor. Resultierende Tabellen sind dann immer deutlich kleiner als die ursprünglichen.

Aliasnamen – also spezielle Namen für die Abfragen

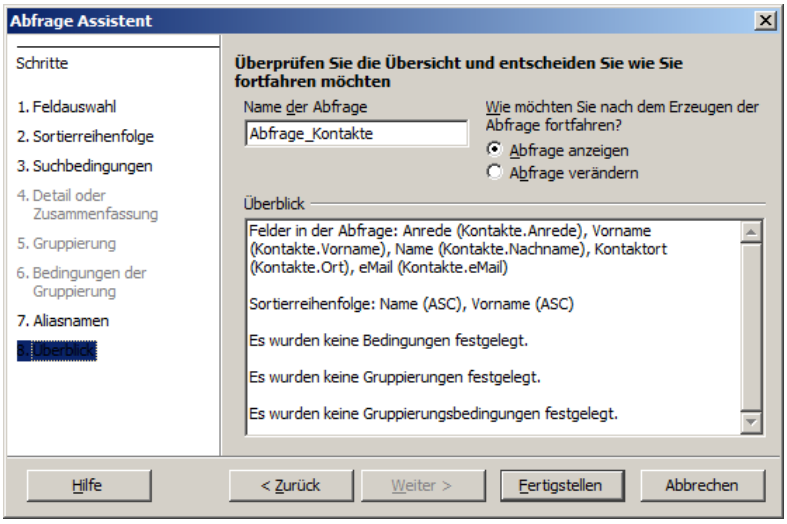
Verschleierung der Datenbank-Interna oder Verbesserung der Nutzerfreundlichkeit.

Alias-Namen können z.B. auch für die Erstellung einer Datenbank-Anwendung in einer anderen Sprache verwendet werden. Dafür möchte man nicht unbedingt die gesamte Struktur der Tabellen usw. überarbeiten.



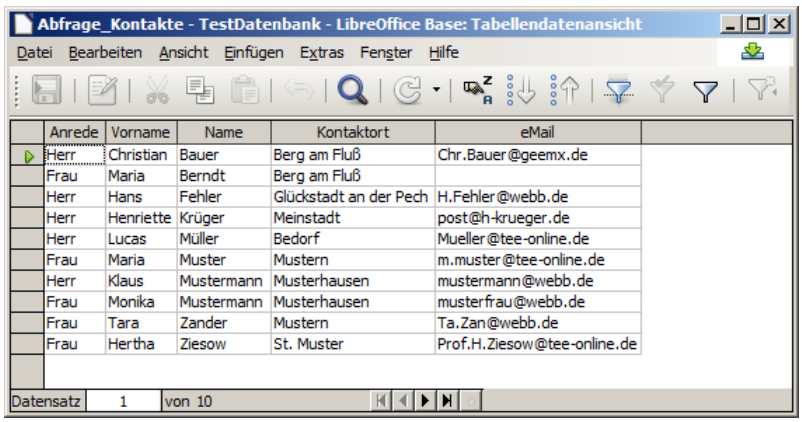
Überblick

Zusammenfassung des Assistenten



Ergebnis ist eine Tabelle

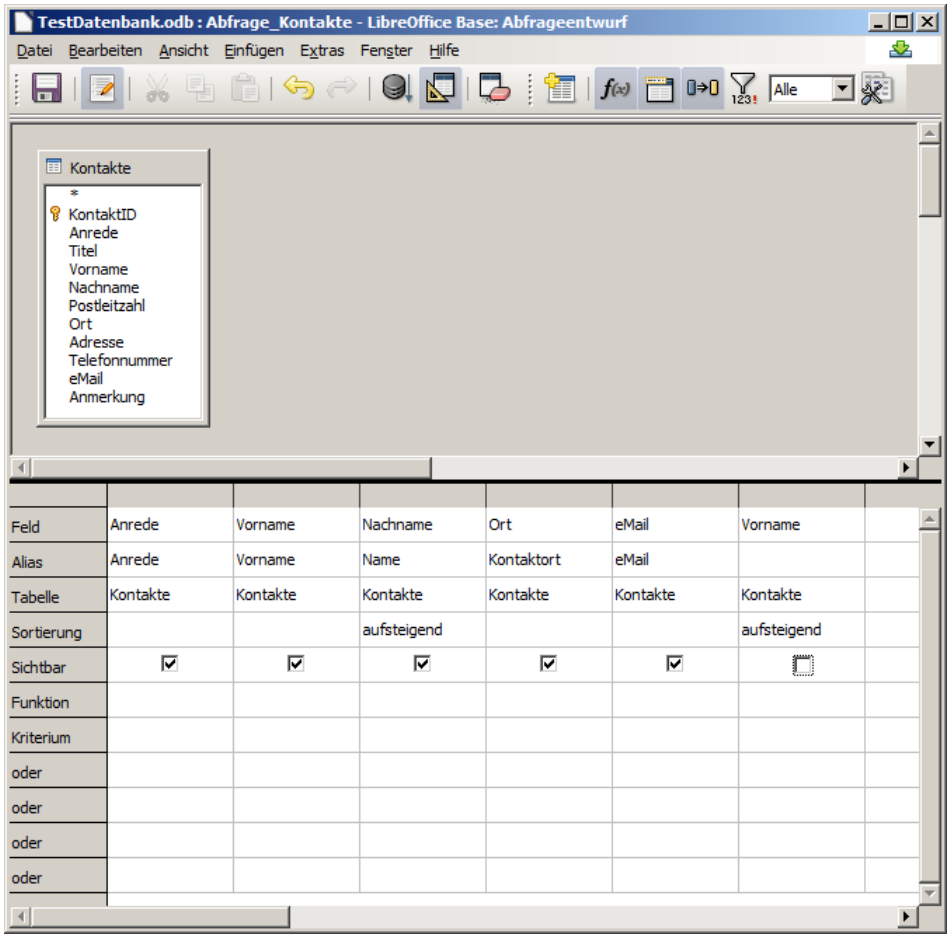
ist temporär, wird also nur erzeugt, wenn der Bedarf besteht, es werden dann die jeweils aktuellen Daten benutzt



Wer etwas über die Umsetzung der Assistenten-Schritte in eine Abfrage lernen möchte, kann sich die fertige Abfrage auch in der Entwurfs-Ansicht (für Abfragen; → 3.1.5.2.2. Erstellen einer Abfrage in der Entwurfs-Ansicht) anzeigen lassen.

Aufgaben:

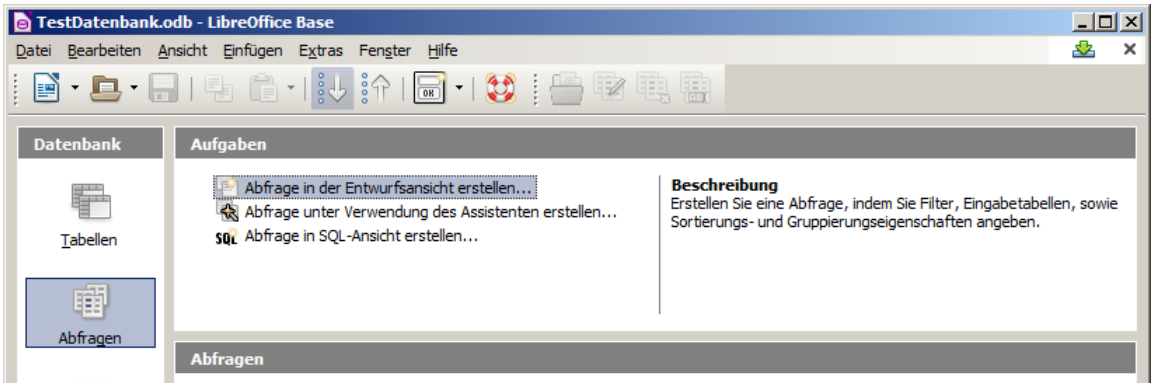
1. ***Prüfen Sie durch ein Experiment, ob eine Abfrage in der Form ihres letzten Aufrufs gespeichert wurde oder nur neu aktualisert in Ihrem DBS ("gespeichert") wird!***



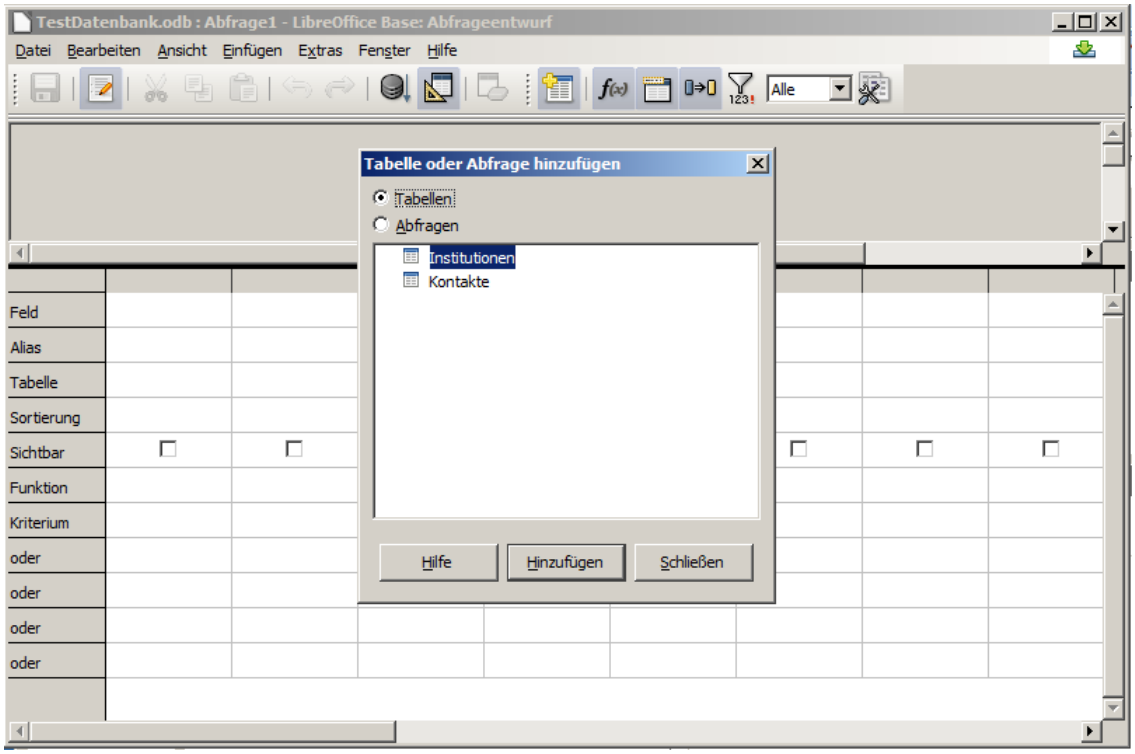
In Vorbereitung der Arbeit mit SQL ist es immer gut, sich Funktions-bekannte Abfragen usw. als fertiges SQL-Statement anzuschauen. Man erkennt dann relativ schnell das dahintersteckende Prinzip. Die SQL-Statements können dann als Vorlagen für eigene SQL-Anweisungen genutzt werden.

3.1.5.2.2. Erstellen einer Abfrage in der Entwurfs-Ansicht

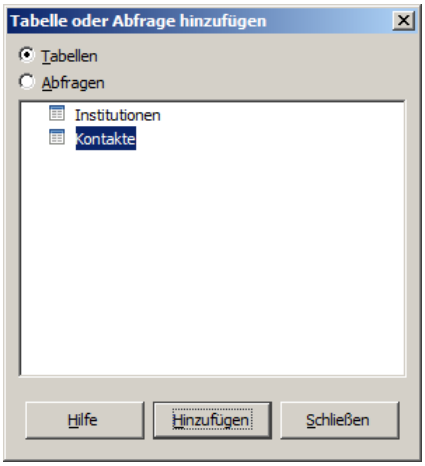
Diese Variante der Abfragen-Erstellung ist wohl die beliebteste. Es ist sehr einfach und schnell möglich, sich die Abfragen zusammenzuklicken und die Ergebnisse sind meist sofort zufriedenstellend.
Für viele Zwecke ist auch eine Vorbereitung über den Assistenten sinnvoll. Später kann die Abfrage kopiert werden und dann in einer Kopie mit den Feineinstellungen experimentiert werden.
Der Aufruf erfogt im oberen Teil der Teil der Abfrage-Übersicht.



Es erscheint die Entwurfs-Ansicht mit einem "Hinzufügen"-Dialog.



Für die weitere Nutzung von Daten müssen zuerst einmal die Tabellen oder andere – schon vorhandene – Abfragen hinzugefügt werden.
Wenn man genug Tabellen und / oder Abfragen hinzugefügt hat, verläßt man den "Hinzufügen"-Dialog mit "Schließen".



Zum Schluß wieder der Hinweis, sich das zugehörige SQL-Statement anzusehen.

Gespeichert werden bei Abfragen nur die Erzeugungs-Vorschriften, nicht aber die enthaltenen Daten. Diese werden bei jedem Aufruf / bei jeder Nutzung neu zusammengestellt.

Aufgaben:

1. ***Öffnen Sie sich die Entwurfs-Ansicht und fügen Sie sich die Tabelle "Kontakte" hinzu!***
2. ***Realisieren Sie eine "Abfrage2 Kontake", indem Sie die Abfrage aus der Assitenten-Nutzung*** (→ 3.1.3.1.1. Erstellen einer Tabelle mit Hilfe des Assistenten) ***nachbauen!***
3. ***Versuchen Sie die Abfrage2 so zu verändern, dass die nicht angezeigte Spalte "Vorname" (ganz rechts) nicht mehr benötigt wird, aber die Sortierung trotzdem erst nach der des Nachnamens (!) getätigt wird! (Hinweis!: Überlegen Sie sich, wie ein Computer(-Programm) die Entwurfs-Ansicht lesen / interpretieren wird!)***
4. ***Erstellen Sie eine Abfrage, die aus den Institutionen solche heraussucht, die alphabetisch nach H folgen! Handelt es sich bei dieser Abfrage um eine Selektion, Projektion, um beides und / oder einen Verbund*** *(wenn dieser schon besprochen wurde)*? ***Begründen Sie Ihre Meinung!***
5. ***Gesucht ist eine Übersicht aller Kontakt-Personen mit Name und Vorname, die in Orten wohnen, die nicht mit M beginnen! Handelt es sich bei dieser Abfrage um eine Selektion, Projektion, um beides und / oder einen Verbund*** *(wenn dieser schon besprochen wurde)*? ***Begründen Sie Ihre Meinung!***
6. ***Erstellen Sie eine Liste der Institutionen mit den Personen, die aus diesen bekannt sind! Handelt es sich bei dieser Abfrage um eine Selektion, Projektion, um beides und / oder einen Verbund*** *(wenn dieser schon besprochen wurde)*? ***Begründen Sie Ihre Meinung!***

Bedingungs-Operatoren in BASE

Bedingungs-Operatoren in Abfragen

Operator / Symbol(e)	Benennung	Bemerkungen / Bedeutung
=	(ist) gleich	in Abfrage-Feldern wird der "="-Operator nicht angezeigt; → wird kein Operator angegeben, dann wird automatisch "=" angenommen
<>	(ist) ungleich	alle anderen Werte werden akzeptiert
<	(ist) kleiner (als)	alle Werte, die kleiner als der angegebene sind, werden akzeptiert
<=	(ist) kleiner oder gleich	erfüllt, wenn der Wert kleiner als der angegebene oder gleich groß, wie dieser, ist
>	(ist) größer (als)	alle Werte, die größer als der angegebene sind, werden akzeptiert
>=	(ist) größer oder gleich	erfüllt, wenn der Wert größer als der angegebene oder gleich groß, wie dieser, ist

logische Operatoren in Abfragen

Operator / Symbol(e)	Benennung	Bemerkungen / Bedeutung
AND	logisches UND	beide Bedingungen (links und rechts) müssen den gleichen Wahrheits-Wert besitzen (, dann ist das Ergebnis auch wieder wahr)
OR	logisches ODER	es reicht, wenn einer der beiden Bedingungen (links und rechts) wahr ist (, dann ist auch das Ergebnis wahr)
NOT	logisches NICHT Negation	der Wahrheits-Wert der folgenden Aussage wird negiert / umgedreht (aus wahr wird falsch und umgekehrt)

Platzhalter / Joker-Symbole

Zeichen / Symbol	Bedeutung	Bemerkungen / Hinweise
***** **%**	entspricht einer beliebigen (auch leeren) Zeichenkette bzw. eines beliebigen Wertes	M* bzw. M% bedeutet somit, dass alle Texte / Zeichenketten gefunden werden / gemeint sind, die mit M beginnen (es müssen auch keine weiteren Zeichen folgen – nur M reicht)
? _	steht für (genau) ein Zeichen in einer Zeichenkette	M???? bzw. M_ _ _ _ bedeutet, dass alle Texte / Zeichenketten gefunden werden / gemeint sind, die mit M anfangen und dann folgend genau (exakt) noch 4 beliebige Zeichen enthalten (Leerzeichen werden nicht als Zeichen verstanden)

Filter-Bedingungen für Abfragen

Option	**Auswirkung / Bedeutung**	**SQL-Schlüsselwort**	**Bemerkungen Beispiel(e)**
ohne	*keine*		
IST LEER	ist erfüllt, wenn das Feld einen NULL-Wert besitzt	IS NULL IS LEER	bei Options-Feldern (JA / NEIN) wird der unbestimmte Fall genutzt (also weder Ja noch Nein)
IST NICHT LEER	Negation von "IST LEER"; ist erfüllt, wenn das Feld einen (von NULL abweichenden) Wert besitzt	IS NOT NULL IS NOT LEER	
LIKE **WIE**	Übereinstimmung; ist erfüllt, wenn das Feld den entsprechenden Wert besitzt	LIKE	WIE 'Muster*' oder WIE 'Muster????' (liefert: z.B. "Mustermann" und "Musterfrau" zurück) bei Zahlen besser **=** benutzen
ZWISCHEN *min* **UND** *max*	im Intervall; ist erfüllt, wenn der Feldinhalt zwischen den Angaben ***min*** und ***max*** liegt	BETWEEN *min* AND *max*	
IN (*Liste***)**	Entsprechung; Enthaltung ist erfüllt, wenn das Feld mit einem Wert aus der (Semikolon-getrennten) übereinstimmt		IN (3; 6; 12; 24) IN ('Moskau'; 'Berlin'; 'New York')
NICHT …	Negation; negiert den folgenden Ausdruck	NOT …	
= WAHR	Validierung; ist erfüllt, wenn der Feldinhalt ***wahr*** enthält	= TRUE	
= FALSCH	Falsifizierung ist erfüllt, wenn der Feldinhalt ***falsch*** enthält	= FALSE	
EXISTS(SELECT-Anweisung**)**	falls mind. ein Datensatz existiert, dann gibt die Funktion TRUE zurück, sonst FALSE		

3.1.5.2.3. Berechnungen in einer Abfrage

In Abfragen fehlen oft einige Zusatz-Daten, die uns das Leben unwahrscheinlich einfacher machen würden. Wieviele Datensätze sind denn nun in der Abfrage enthalten, welcher Durchschnitt ergibt sich z.B. für die Umsatz-Zahlen usw. usf.?
Dafür brauchen wir kleine Rechnungen / Funktionen, die quasi parallel oder im Hintergrund die abgefragten Daten auswerten.
Wenn wir Berechnungen zu Abfragen hinzufügen, dann handelt es sich sachlich eigentlich schon um einen Bericht (→ 3.1.5.3. Berichte). Heute wird die Trennung nicht mehr so hart gemacht.

statt des Feldnamens (Attributes) werden Formeln mit den Feldnamen (z.B. "Nettopreis") ohne führendes Gleichheitszeichen eingegeben (z.B. zur Berechnung des Bruttopreises: "Nettopreis" * 1,19)
es muss dann ein Alias vergeben werden, da sonst die berechnete Spalte keine Überschrift hätte, sie ist ja immer neu

Rechen-Operatoren in Abfragen

Operator / Symbol(e)	Benennung	Bemerkungen / Bedeutung
+	Addition	fortlaufende Addition z.B. über Konstrukt: Gesamt = Gesamt **+** Feldwert möglich
-	Subtraktion	fortlaufende Subtraktion z.B. über Konstrukt: Bestand = Bestand **-** Feldwert möglich
*****	Multiplikation	fortlaufende Multiplikation z.B. über Konstrukt: Produkt = Produkt ***** Feldwert möglich
/	Division	fortlaufende Divisiion z.B. über Konstrukt: Anteil = Anteil **/** Feldwert möglich

Funktionen in Abfragen

Option / Funktion	Auswirkung / Bedeutung	SQL-Schlüssel-wort	Bemerkungen
ohne	*keine*		
Durchschnitt	berechnet das (arithmetrische) Mittel (Mittelwert) des Feldes (Attri-butes)	AVG	
Anzahl	zählt die Datensätze, die in der Abfrage auftauchen	COUNT	COUNT(*) .. es werden alle Datensätze (in der Abfrage) gezählt COUNT(*Spalte*) .. es wer-den die Datensätze gezählt, bei denen die ***Spalte*** einen Nicht-NULL-Wert enthält
Maximum	ermittelt den größten Wert des Feldes (Attributes)	MAX	
Minimum	ermittelt den kleinsten Wert des Feldes (Attributes)	MIN	
Summe	berechnet die Summe der Feld-Werte (Attribut-Werte)	SUM	
Gruppiert	gruppiert die Datensätze nach den Werten im ausgewählten Feld (Attribut)	GROUP BY	

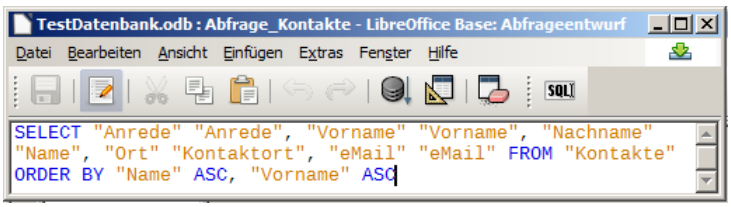
Verwendung z.B., um die Datensätze zu zählen oder den Gesamt-Wert der vorhandenen Waren für eine Inventur zu berechnen.

Verschiedene Waren-Gruppen, Jahrgänge oder die einzelnen Klassen lassen sich über die Gruppierung aufbröseln.

über Minimum und Maximum kann man sich Renner-Penner-Übersichten erstellen oder den besten oder schlechtesten Umsatz-Tag ermitteln

3.1.5.2.4. Erstellen einer Abfrage mittels SQL

Diese Methode zum direkten Erstellen einer Abfrage ist mehr was für Profis oder Programmierer. Für einfache Abfragen ist es aber ein gleichwertiges Mittel. Komplizierte Abfragen lassen sich besser über den Assistenten (→ 3.1.3.2.1. Erstellen einer Abfrage mit dem Assistenten) oder die Entwurfs-Ansicht (→ 3.1.3.2.2. Erstellen einer Abfrage in der Entwurfs-Ansicht) zusammenstellen und austesten.
Als Beispiel sei hier der Code für die mit dem Assisten erstellte Abfrage aufgezeigt.
Man kann den Text fast verstehen.
Mehr dazu im SQL-Teil dieses Skriptes (→ 5. Datenbanken - SQL; → 5.1.x. CREATE VIEW).
Zum langsamen Herantasten an SQL kann man sich zuerst einmal den SQL-Code von fertigen – funktionierenden – Abfragen ansehen.



Später ist man vielleicht schneller, wenn man nur noch den Code eingibt. Das setzt aber auch leider immer voraus, dass man die ganzen Namen von Tabellen, anderen Abfragen und den Attributen im Kopf oder auf einer Übersicht hat. Diese werden immer – exakt geschrieben – für einen funktionierenden SQL-Code gebraucht.
Letztendlich lesen sich SQL-Codes fast wie eine umgangssprachliche – wenn auch englische – Arbeits-Aufforderung. Dieses war ja auch ein erklärtes Ziel bei der Entwicklung von SQL.

Aufgaben:

1. ***Verändern Sie im obigen SQL-Statement das zweite Auftreten eines Attributes in die englische Bezeichnung (z.B. "Vorname" durch "Firstname")! Lassen Sie das Statement dann ausführen und schauen Sie sich das Ergebnis genauer an! Welche Funktion erfüllt die "Attribut-Wiederholung"?***
2. ***Überlegen Sie sich, ob man das erste Auftreten eines Attribut-Namens (innerhalb eines SQL-Statement's) ebenfalls ändern kann? Tragen Sie Ihren Standpunkt vor den anderen Kursteilnehmern vor! Diskutieren Sie den Vorschlag / die Vorschläge!***

komplexe Aufgaben (zu Abfragen):

1. *Erstellen Sie eine Abfrage, die zur Bezeichnung der Institution noch die verbundenen Kontakte mit Nachnamen und Vornamen zusammenstellt!*
   Geben Sie ob es sich um eine Selektion, Projektion oder einen Verbund bzw. um Kombinationen davon handelt! Begründen Sie Ihre Aussage!
   Notieren Sie sich die zugehörige SQL-Anweisung mit Platz zwischen den Zeilen! Kommentieren Sie die verschiedenen Teile der SQL-Anweisung!
2. *Stellen Sie eine Abfrage zusammen, die Titel, Vorname, Nachname und eMail-Adresse der Kontakte (aufsteigend) sortiert nach den Institutions-Bezeichnungen (mit anzeigen!) darstellt!*
   Notieren Sie sich die zugehörige SQL-Anweisung! Durch welchen SQL-Teil wird die Sortierung realisiert! Unterstreichen Sie diesen!
3. *Erstellen Sie eine neue Abfrage, wie bei 2., bei der eine 2. Sortierung (aufsteigend) nach dem Nachnamen erfolgt!*
   Notieren Sie sich die zugehörige SQL-Anweisung! Kennzeichnen Sie die verschiedenen Sortierungen mit unterschiedlichen Farben!
4. *Erstellen Sie eine weitere Abfrage, wie bei 3., in der nur Einträge angezeigt werden, bei denen der Textteil "Muster" im Nachnamen des Kontaktes auftaucht! (Testen Sie auch mal den Unterschied, wenn "muster" gesucht wird!)*
   Notieren Sie sich die zugehörige SQL-Anweisung!
5. *Kopieren Sie sich die Abfrage von Aufgabe 4 und verändern Sie diese so, dass alle Einträge herausgesucht werden, bei denen der Nachname auf "er" endet!*
6. *Erstellen Sie nur mit SQL eine Abfrage, die den Nachnamen, Vornamen und deren Telefonnummer sowie die eMail der Institution anzeigt!*

für die gehobene Anspruchsebene:

7. *Erweitern Sie die SQL-Anweisung von 6. um eine Sortierung nach den Nachnamen!*
8. *Ermitteln Sie, welche Kontakte eine Verbindung zu einer Institution haben, bei denen in den eMail-Adressen der Textabschnitt "web" vorkommt!*
9. *Stellen Sie eine vollständige SQL-Anweisung zusammen, die alle Institutionen mit all ihren Daten mit allen zugehörigen Kontakten (ebenfalls mit allen Daten) in einer Tabelle wiedergibt!*

Aufgaben für alle:

10. *a) Geben Sie die folgende SQL-Anweisung ein! Überlegen Sie sich dabei, was diese machen wird! Notieren Sie Ihre Vermutung!*

```
INSERT INTO "Institutionen" VALUES(7,'auto e.V.',NULL,'post@auto.ev','');
```

    b) Führen Sie die SQL-Anweisung aus! Prüfen Sie Ihre Vermutung anhand des neuen Datenbestandes (ev. gegen die Tabellen auf Papier)! Sollte Ihre Vermutung nicht richtig gewesen sein, dann erkunden Sie, warum die Vermutung falsch war!
11. *Geben Sie die folgende SQL-Anweisung ein! Überlegen Sie sich dabei, was diese machen wird!*

```
UPDATE "Kontakte" SET "Adresse"='Blumenweg 14' WHERE "Nachname"='Fehler';
```

12. *Überlegen Sie sich eine SQL-Anweisung, die Ihre Orts-Tabelle um einen Ort (45678 Neuberg) ergänzt! Sie können diese auch ausprobieren!*

3.1.5.3. Berichte

Unter Berichten verstehen wir Ergebnis-Auskopplungen meist als Ausdruck oder eben als PDF-Dokument). Dabei soll der Zustand der Datenbank – bzw. seiner Daten – zu einem bestimmten Zeitpunkt dokumentiert werden. Ein klassisches Beispiel ist eine Klassen-Liste zum Anfang des Schuljahres oder eine Umsatz-Zusammenstellung zum Quartals- oder Jahres-Ende.
Die in einem Bericht erzeugten Daten sollen im Allgemeinen dann später durch weitere Daten nicht verändert werden. Niemanden nützt ein Quartals-Bericht z.B. vom II. Quartal, in dem auch Daten vom III. mit eingeflossen sind. Auch nachträgliche Veränderungen (außer echte Fehler-Korrekturen) sollen dadurch ausgeschlossen werden.

Für viele Anwendungen bieten sich auch laufende Berichte an. Das könnten z.B. Zimmer-Auslastung in einem Hotel, Produktions-Übersichten, Lager-Bestände usw. sein.

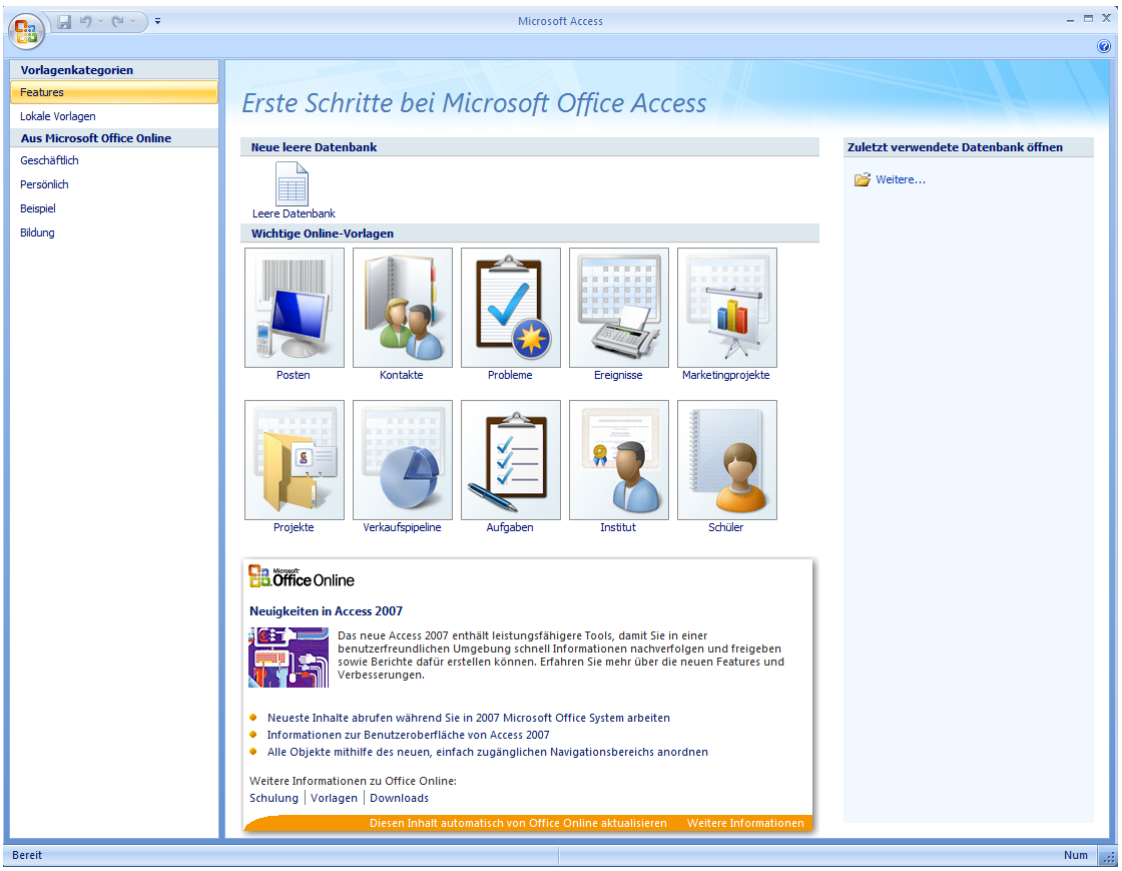
3.1.5.4. Formulare

Tabellen sind übersichtlich und stellen die Daten sehr kompakt dar. Für den normalen Nutzer sind sie aber eher schwierig zu benutzen. Schnell ist man in Zeile oder Spalte verrutscht und schon stehen die falschen Daten irgendwo.
Besonders schwierig ist die Arbeit mit Fremdschlüsseln. Der Nutzer müsste sie entweder alle kennen oder ständig irgendwo nachschlagen. Das wäre sehr Zeit-aufwändig und wohl auch Fehler-anfällig.
Nutzer-freundlicher sind die üblichen Formulare / Dialoge, die wir zur Genüge aus Windows kennen. Formulare sind die modernen Ein- und Ausgabe-Hilfen für die normalen Datenbank-Benutzer. Sie müssen auch nicht unbedingt verstehen, wie alles in der Datenbank abläuft. Solche Dinge, wie Primär- und Sekundär-Schlüssel können wir mit Formularen gut vor dem unbedarften Nutzer verbergen.
Fremdschlüssel werden z.B. in Form von Auswahl-Feldern bereitgestellt, die nicht den Fremdschlüssel selbst, sondern ein beschreibendes Attribut anzeigt. Damit können die Nutzer dann auch etwas anfangen.

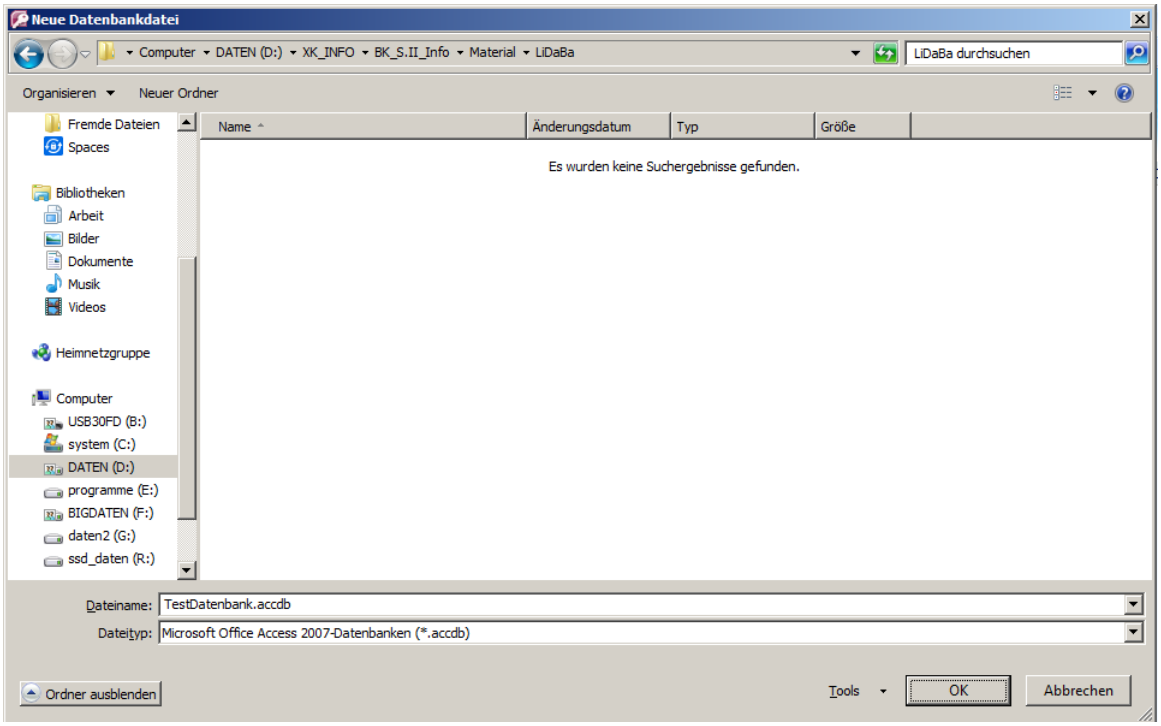
3.1.6. Datenbanken mit ACCESS

Die Nutzung von Datenbank-Komponenten aus Office-Produkten wird in einzelnen Fach-Rahmenplänen (bzw. Curricula) abgelehnt.
Für Einsteiger-Projekte sind sie aber sehr gut geeignet!

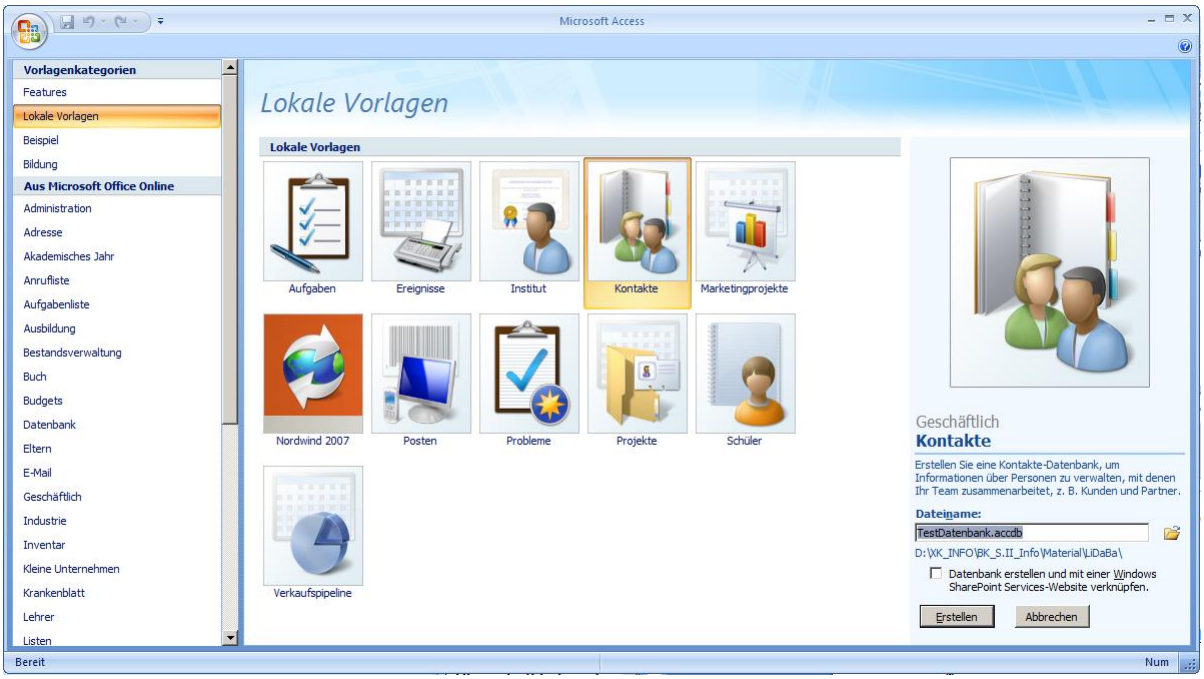
Die neueren Versionen (ab 2007) von ACCESS sind sicher für unbedarfte Nutzer ohne Datenbank-Theorie-Hintergrund ein nutzbares Tool. Viele Tools und Assistenten führen Anfänger bzw. Datenbank-Beginner gut durch das Programm. Allerdings muss man schon wissen, was man will. Ohne theoretische Vorkenntnisse kommt man auch in ACCESS nicht weit.
Wer aber strukturiert Datzenbank-Entwürfe umsetzen will, der braucht erst einmal viel Erfahrung und Pionier-Geist. So manche wichtige Funktion ist in die hintersten Menüband-Gefilde verstoßen worden. Auch die Benennung der Menü-Punkte ist sehr uneinheitlich und wenig hifreich, wenn man etwas sucht.
Nach dem Start von ACCESS erscheint ein graphisch aufgepeppter "Erste Schritte"-Dialog.



ACCESS bietet von sich aus nun die ersten Vorlagen für Datenbanken an bzw. ganz rechts die letzten benutzten DB-Dateien.
Um jetzt nicht gleich die Nachhause-Telefonieren-Funktion von ACCESS zu aktivieren und irgendwelche Vorlagen von Microsoft-Online zu beziehen, nutzen wir erst einmal den Bereich "Lokale Vorlagen".
Hier wählen wir "Kontakte" als gewünschte Vorlage und speichern in einem beliebigen Ordner mit einem passenden Namen ab. Die aktuelle Datei-Endung für ACCESS-Datenbanken lautet *.ACCDB.



Nun bleibt noch das eigentliche "Erstellen", d.h. von der Vorlage wird eine konkrete Kopie (Instanz) erstellt.



Ältere Versionen von ACCESS entsprechen dem Handling, wie wir es bei der BASE-Datenbank vorgestellt haben. Leider versucht ACCESS das Fahrrad neu zu erfinden und verkompliziert das Vorgehen gleich von Anfang an.

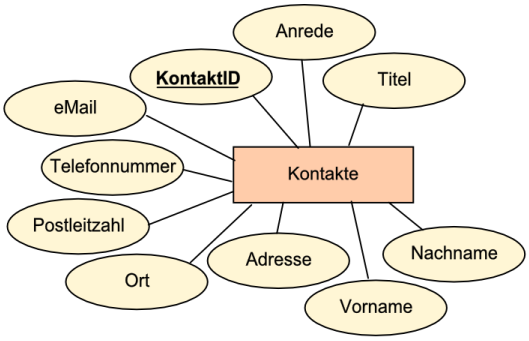
Tips und Hinweise für Arbeits-Erleichterungen in neuen ACCESS-Versionen

- wenn man zuerst die Detail-Tabellen anlegt, dann kann man schon im Entwurf der übergeordneten Tabellen Nachfrage-Spalten erstellen, die sich aus den Detail-Tabellen speisen, aber nur den (Fremd-)Schlüssel speichern
- Abfragen lassen auch Daten-Eingaben zu; damit lassen sich schnell Daten eingeben, vor allem, wenn nur bestimmte Felder benutzt werden sollen / ausreichen
-

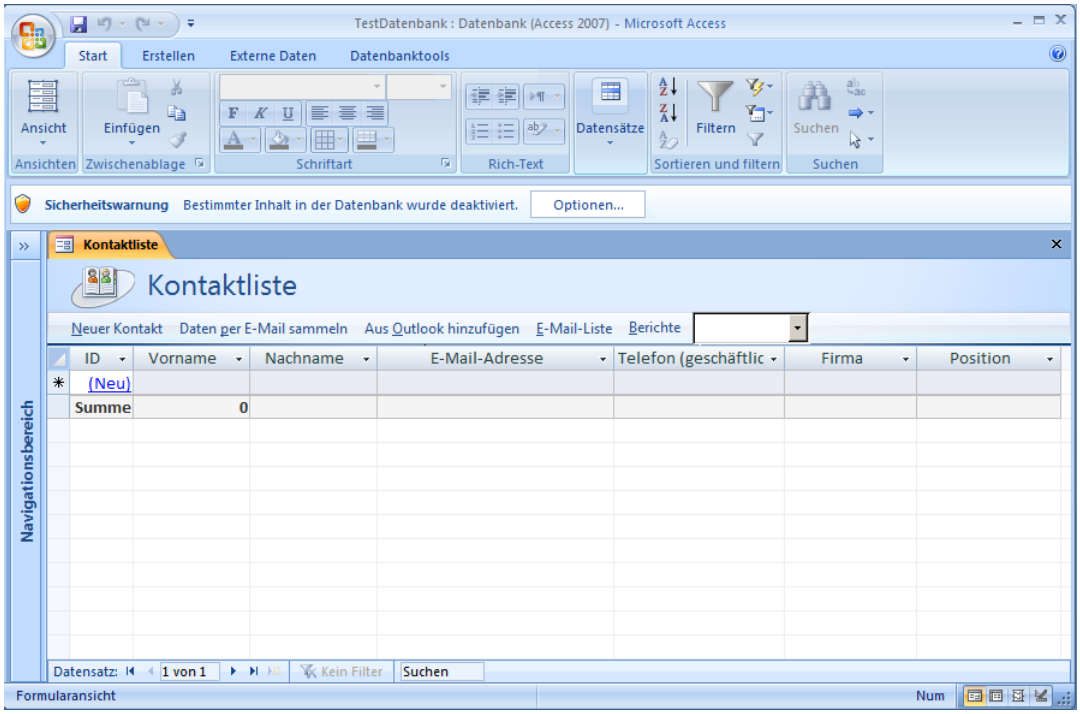
3.1.6.1. Tabellen

ACCESS leitet uns nach dem ersten Öffnen einer Datenbank gleich in die Daten-Eingabe. Das wollen wir auch gleich tun, um uns an die Arbeitsweise zu gewöhnen.

Ziel soll auch hier eine ähnliche Tabelle, wie in der BASE-Lösung (→ 3.1.5.1. Tabellen). Das nebenstehende ERD einer ersten Teil-Lösung ist die Orientierung für uns.

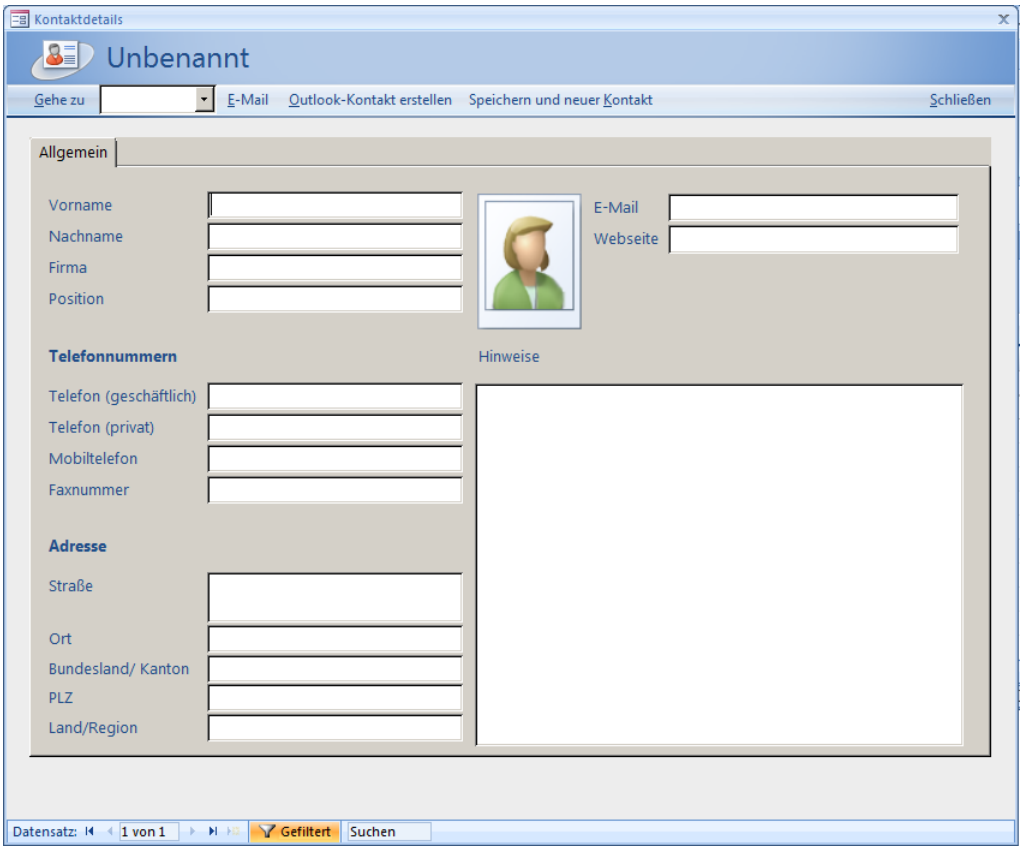


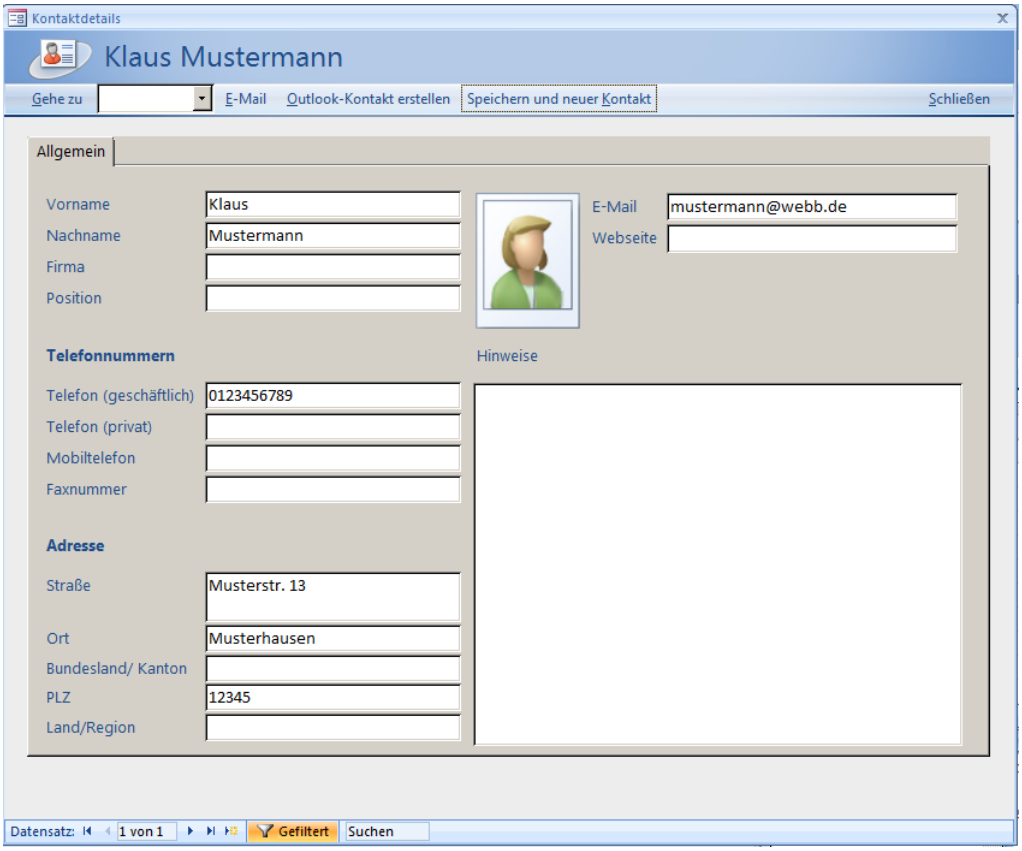
3.1.6.1.1. Eingeben von Daten in eine Tabelle



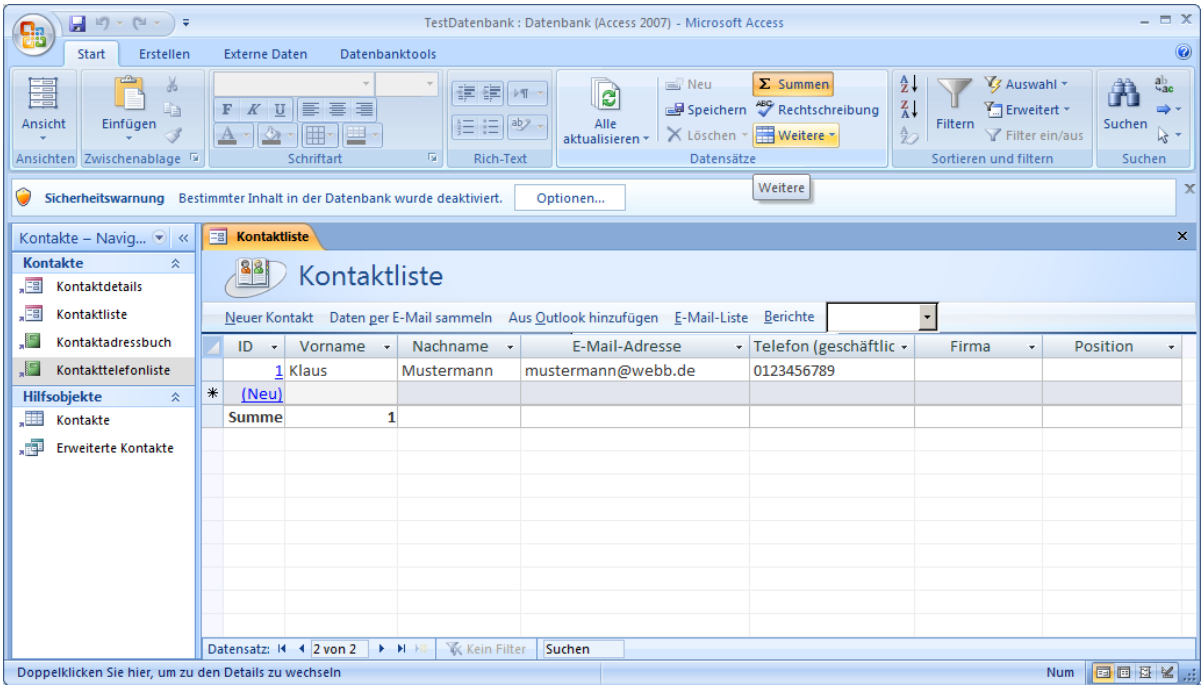
Zu existierenden Tabellen (oder auch Abfragen) lassen sich in ACCESS automatisch einfache Formulare erstellen (→ 3.1.6.4. Formulare). Mit diesen ist dann eine Nutzer-freundliche Dateneingabe – aber auch –Anzeige – möglich.

Die fertigen Beispiel-Datenbanken beinhalten auch schon Bedienungs-freundliche Formulare. Alle notwendigen Einstellungen sind getätigt und die elementaren Funktionen schon integriert.

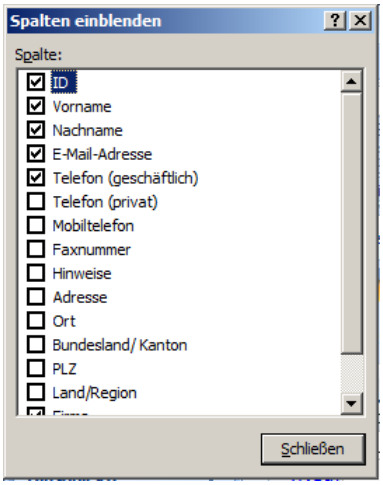


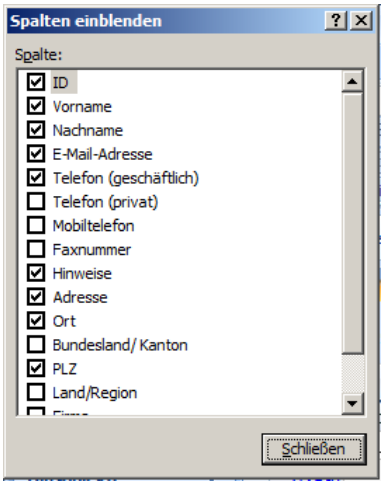


Jetzt können mit dem Formular alle Daten eingegeben werden, für die ein passendes Feld vorgesehen ist. Nach der Eingabe aller Daten eines Kontaktes geht man auf "Speichern und neuer Kontakt" oder "Schließen".
Wenn wir uns dann die Tabelle ansehen, sind die vielen eingebenen Daten trotzdem nicht sichtbar geworden. Das diese aber immer noch da sind, können wir bei der Nutzung des Formulares über die Datensatz-Navigation unten links einsehen.
Warum die Datenfelder nur eingeschränkt angezeigt werden, erschliesst sich wohl nur Microsoft. Wir können uns aber alle Felder einblenden – und natürlich auch wieder ausblenden – wenn im Menüband "Start" im Menübandbereich "Datensätze" "Weitere" ausgewählt wird.

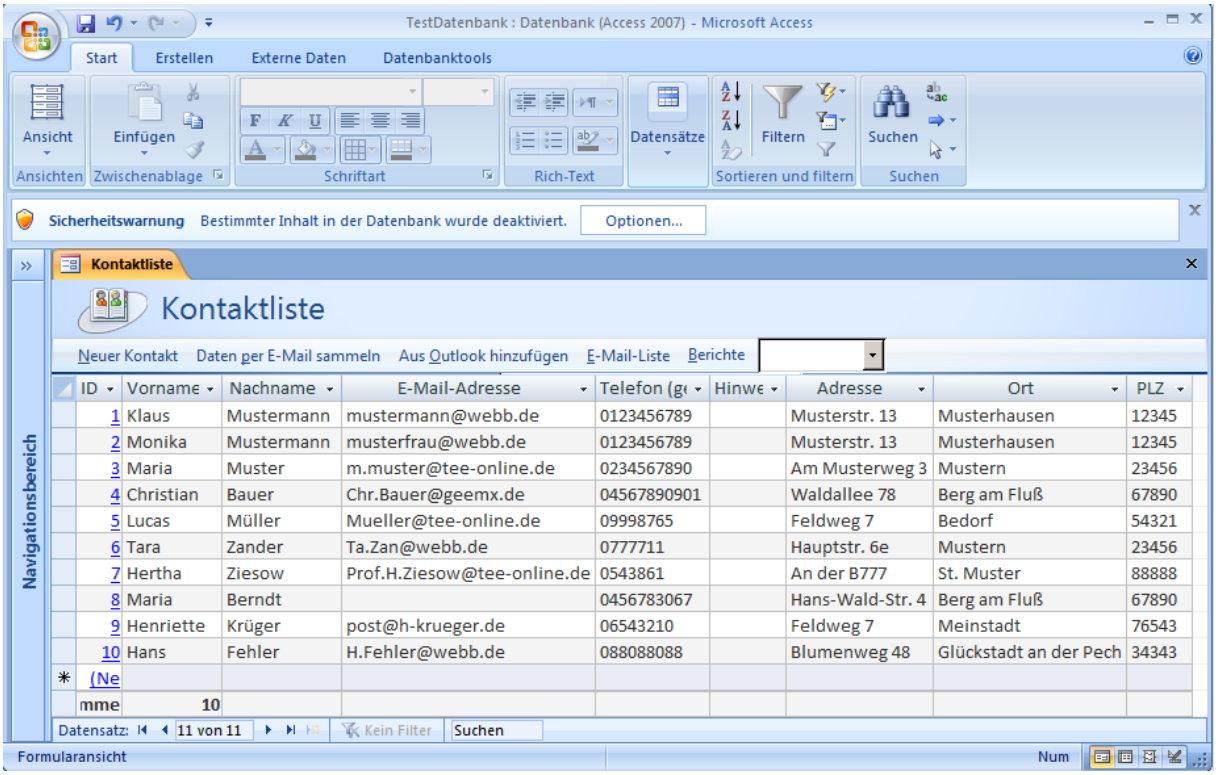


Über das Menü können nun Spalten (Felder, Attribute) eingeblendet werden. Zuerst werden nur die Spalten angezeigt, die derzeit in der Tabellen-Ansicht aktiv sind.
Durch hinzu- oder raus-klicken kann die Auswahl jetzt für unsere Bedürfnisse angepasst werden.
Zwar sind immer noch nicht alle Informationen untergebracht, aber dazu kommen wir später.





Nach dem "Schließen" ist die Tabellen-Ansicht um die gewünschten Spalten erweitert und auch die eingegebenen Daten werden angezeigt.
Ob man dann die Daten lieber über die Tabelle oder die Formular-Ansicht erledigt, ist Geschmackssache.
Erfahrungsgemäß geht die Arbeit in der Tabelle etwas zügiger von der Hand, dafür ist das Formular übersichtlicher.

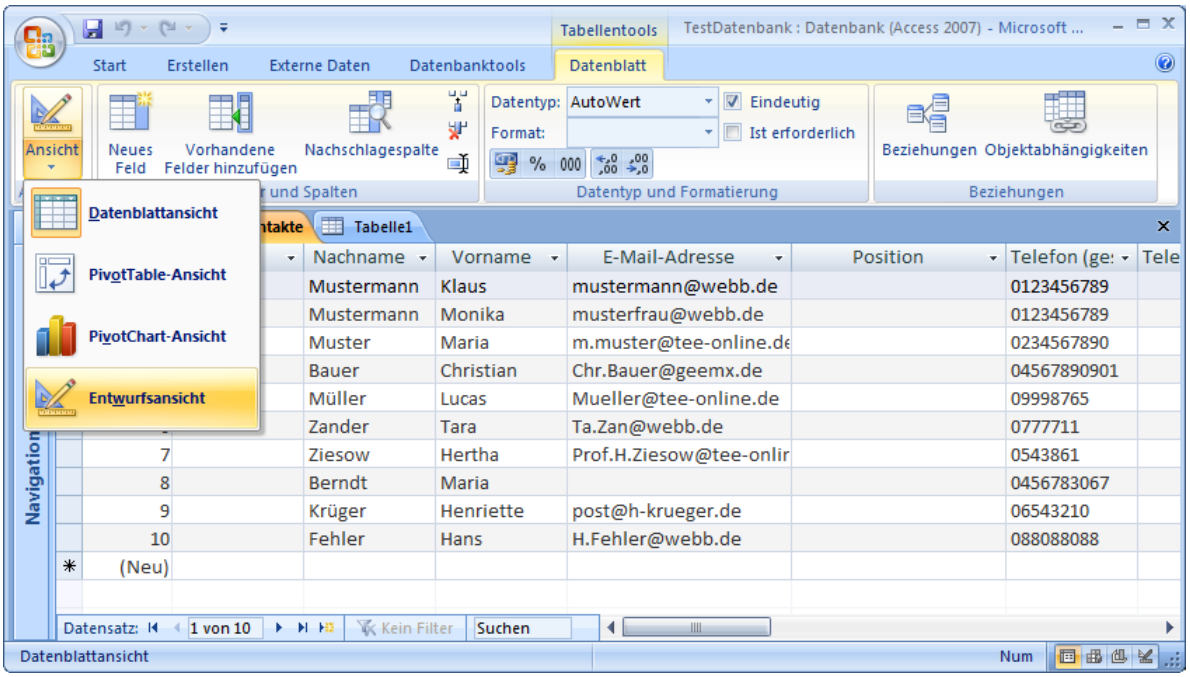


ID	Vorname	Nachname	E-Mail-Adresse	Telefon (g	Hinwe	Adresse	Ort	PLZ
1	Klaus	Mustermann	mustermann@webb.de	0123456789		Musterstr. 13	Musterhausen	12345
2	Monika	Mustermann	musterfrau@webb.de	0123456789		Musterstr. 13	Musterhausen	12345
3	Maria	Muster	m.muster@tee-online.de	0234567890		Am Musterweg 3	Mustern	23456
4	Christian	Bauer	Chr.Bauer@geemx.de	04567890901		Waldallee 78	Berg am Fluß	67890
5	Lucas	Müller	Mueller@tee-online.de	09998765		Feldweg 7	Bedorf	54321
6	Tara	Zander	Ta.Zan@webb.de	0777711		Hauptstr. 6e	Mustern	23456
7	Hertha	Ziesow	Prof.H.Ziesow@tee-online.de	0543861		An der B777	St. Muster	88888
8	Maria	Berndt		0456783067		Hans-Wald-Str. 4	Berg am Fluß	67890
9	Henriette	Krüger	post@h-krueger.de	06543210		Feldweg 7	Meinstadt	76543
10	Hans	Fehler	H.Fehler@webb.de	088088088		Blumenweg 48	Glückstadt an der Pech	34343
*	(Ne							
	mme	10						

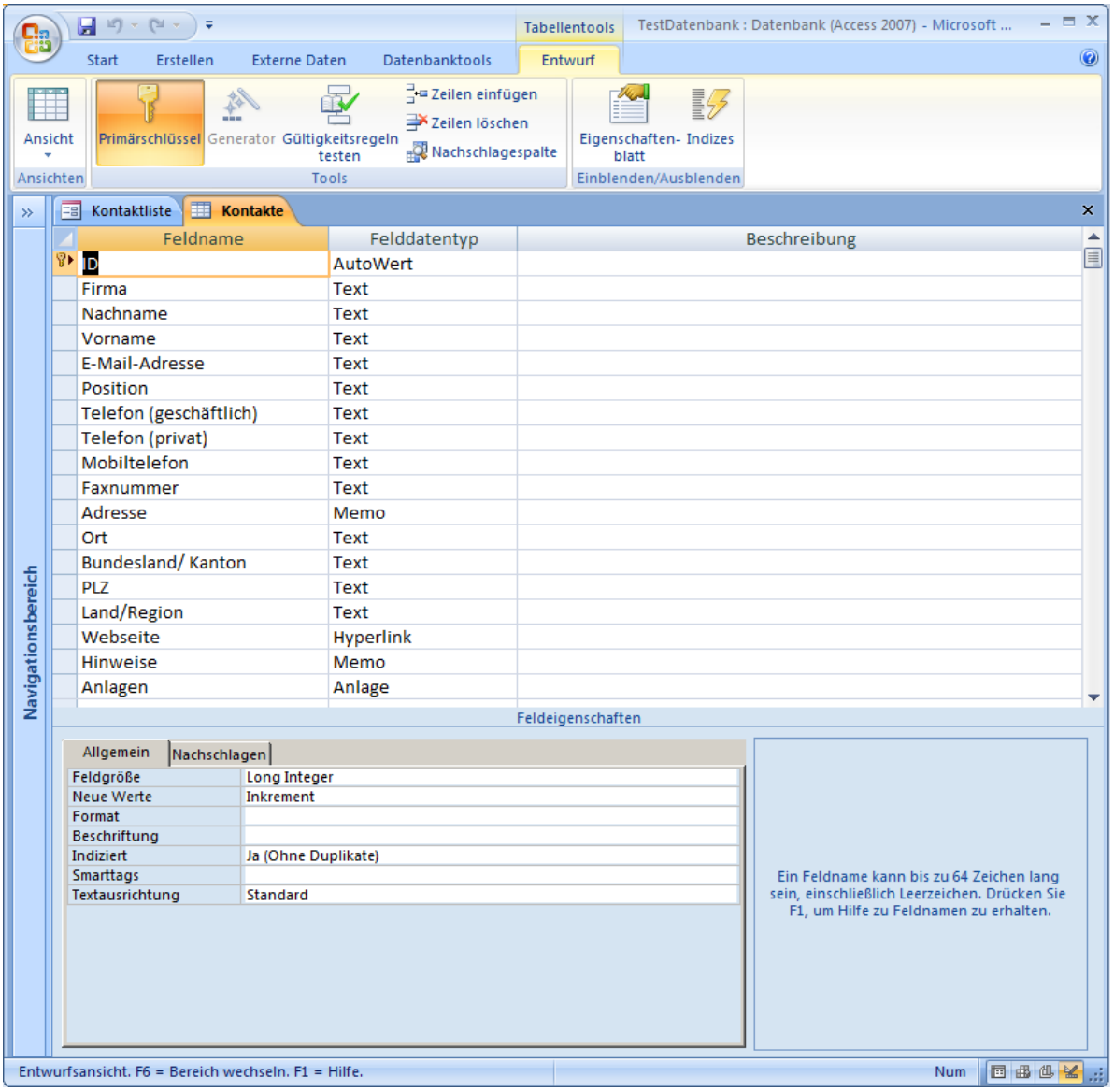
Letztendlich bleiben nun die fehlenden Felder "Anrede" und "Titel".
Wer sich mit der Datenbank BASE beschäftigt hat, weiss, dass es eine Entwurfs- oder Struktur-Editier-Ansicht gibt. Genau diesen werden wir jetzt auch in ACCESS nutzen, um die fehlenden Felder zu ergänzen und die weniger geeigneten Spalten-Namen korrigieren.

3.1.6.1.2. Ändern des Relationsschema einer Tabelle – Korrekturen am Entwurf

Nicht dass man diese Ansicht irgendwo gleich direkt und offensichtlich findet, sie ist im Menüband "Tabellentools" im Band "Datenblatt" unter "Ansicht" versteckt.



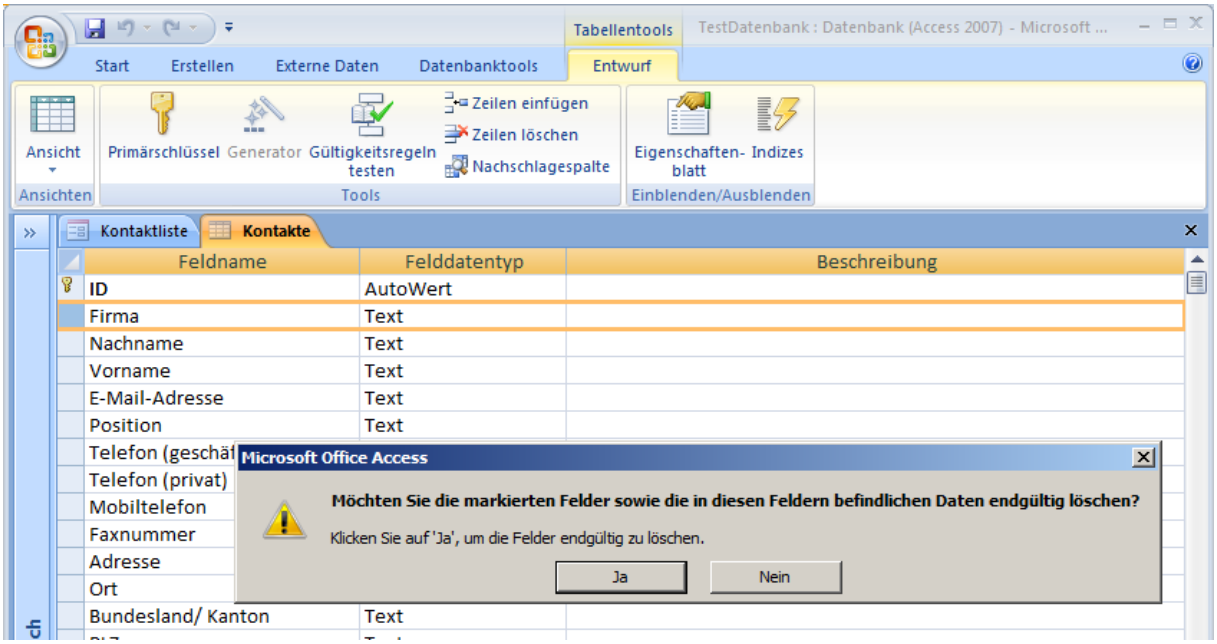
Die Tabellenstruktur ist klassisch aufgestellt. Neben dem frei wählbaren Feldnamen (Attribut, Spalten-Überschrift, Spaltenname) wird für jedes Feld noch der Datentyp gefordert. Die Beschreibung ist optional und kann bei kleinen, übersichtlichen und mit sprechenden Feldnamen versehenen Datenfeldern beruhigt weggelassen werden.



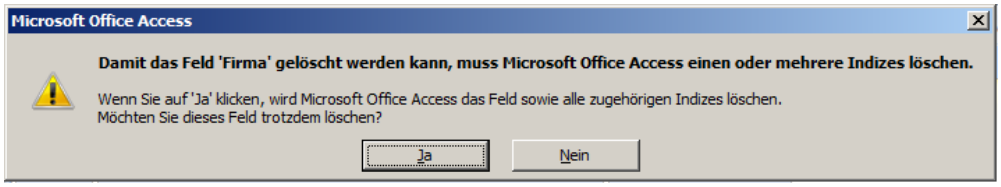
über die rechte Maustaste lässt sich eine unerwünschte Zeile löschen
aber Achtung!, dabei handelt es sich um die gesamte Spalte der Daten-Tabelle (mit all den Daten)
die nachfolgende Bestätigung sollte man sich immer gut überlegen und nicht routiniert auf "Ja" klicken
die Daten sind dann nämlich wirklich! gelöscht

Vorgehen bei Umgestalten / Neuerstellen von Spalten (aus alten)

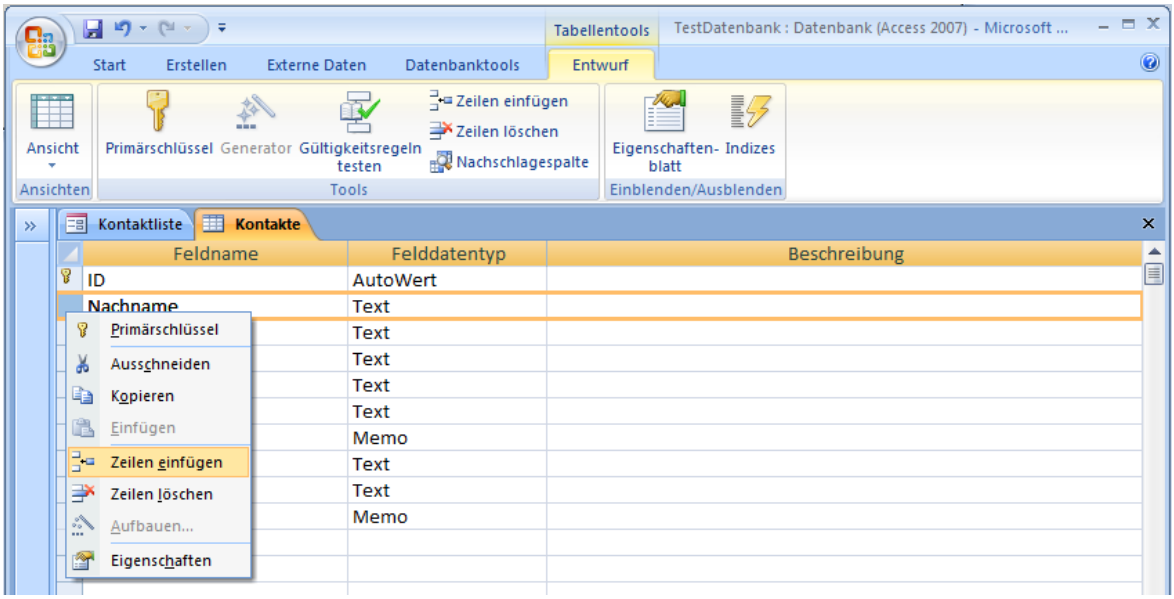
- Erstellen der neuen Spalte(n)
- Übernahme der Daten aus der alten Spalte in die neue(n) Spalte(n)
- Löschen der alten Spalte

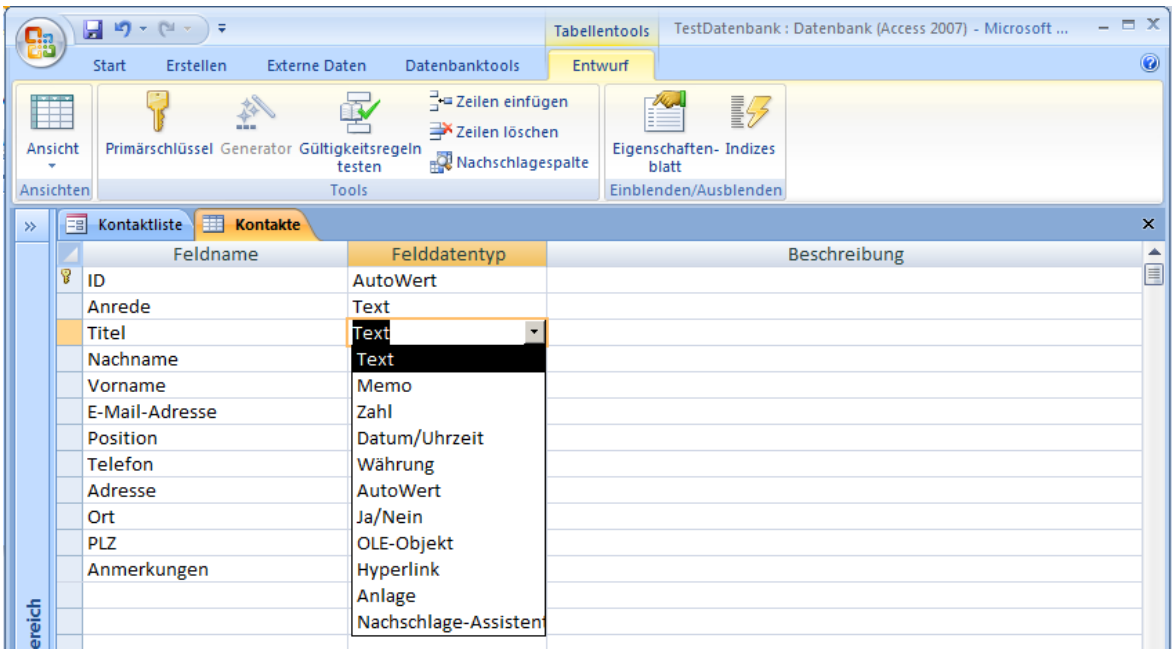


U.U. kommt noch eine 2. Bestätigungs-Anfrage nach einem zu löschenden Index, das kann man dann beruhigt bejahen.

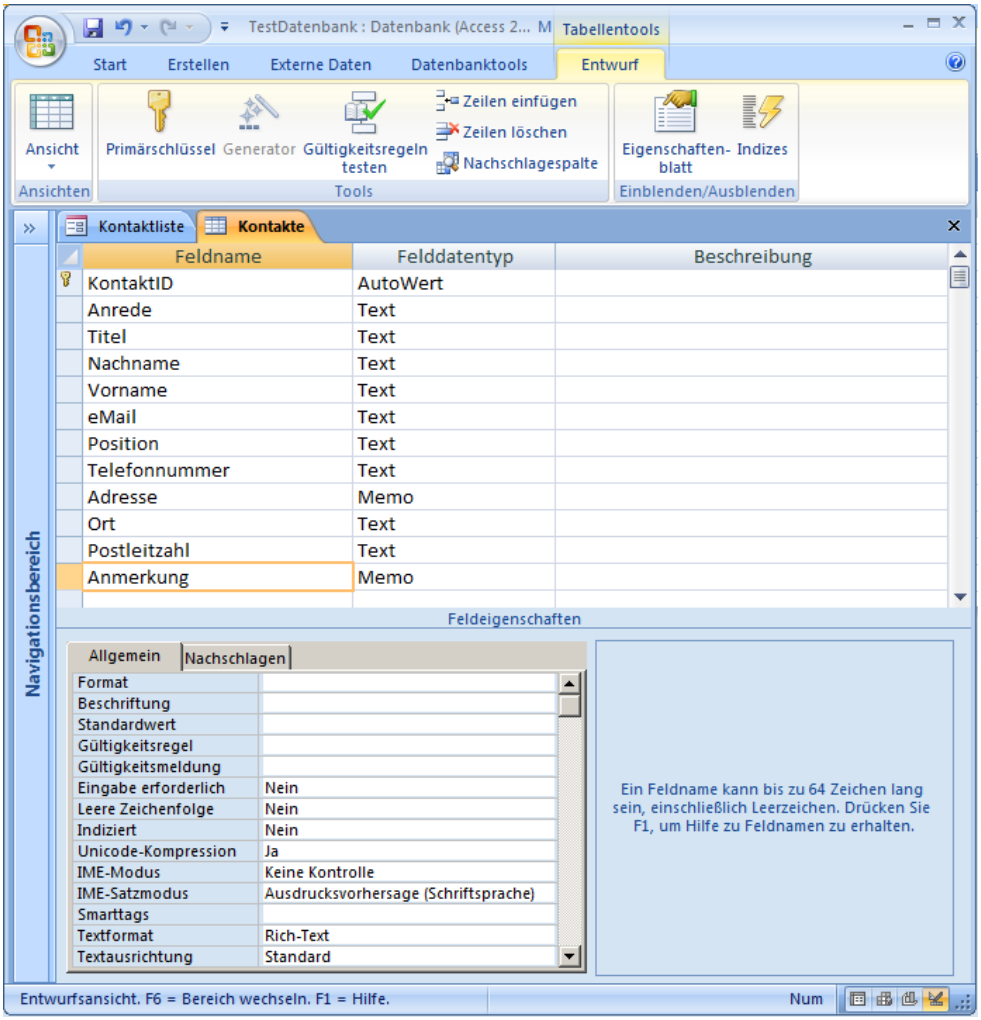


Die Indixes sind nur Dabenbank-interne Hilfs-Tabellen, um Zugriffe auf größere Datenbestände / Tabellen zu beschleunigen. Indixies werden entweder automatisch aktualisert oder neuaufgebaut. Bei selbst erstellten Indixes sollte man ev.die Aktualisierung oder den Neuaufbau händisch anstoßen.



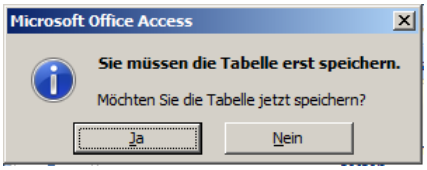


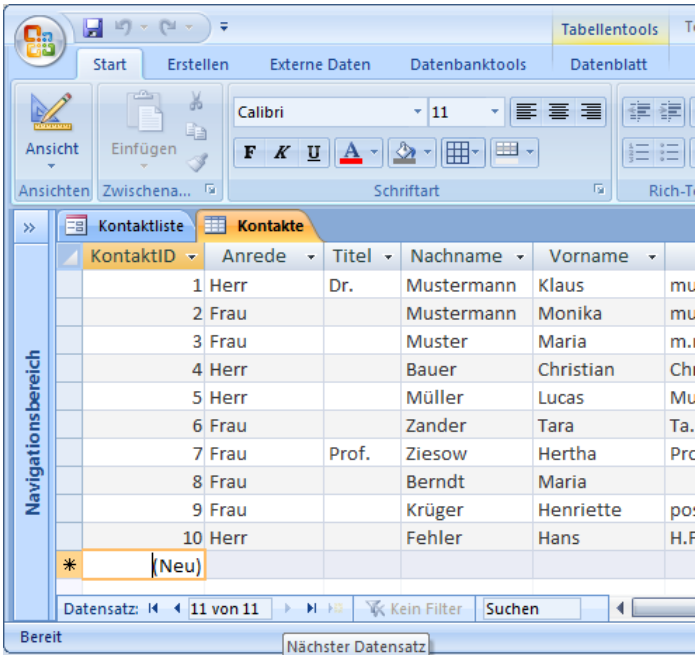
Feldnamen lassen sich direkt ändern
insgesamt sollte die Struktur dann wie folgt aussehen
(wir benötigen die gleiche Struktur in dieser ACCESS-Datenbank wie die in der BASE-Datenbank. Nur so können wir die späteren Zugriffe über SQL bzw. Programmier-Schnittstellen sinnvoll behandeln oder gegenüberstellen
ansonsten macht es praktisch nichts aus, ob ein Feld "Telefon" oder "Telefonnummer oder gar "Klingelling" heisst, für interne Zwecke sollte man nur verständliche Namen nutzen



Die im unteren Bereich sinchtbaren Feldeigenschaften (bei "Allgemein") sind für Spezialisten interessant. Für unsere Anwendungen sollte man dort nicht so viel herumschrauben. Ev. kann man die "Feldgröße" bei Text-Felder anpassen. Das spart u.U. Speicherplatz – ist aber bei dem Platz-Angebot moderner Datenträger eher nicht notwendig. Da ärgert man sich dann eher, wenn man den Nachnamen auf 30 Zeichen beschrägt hat und dann einer daherkommt, der einen Namen mit 31 Zeichen hat.

Der geänderte Entwurf muss nun noch gespeichert werden, um ihn zu aktivieren.
Als Letztes müssen noch die fehlenden Daten ergänzt werden.



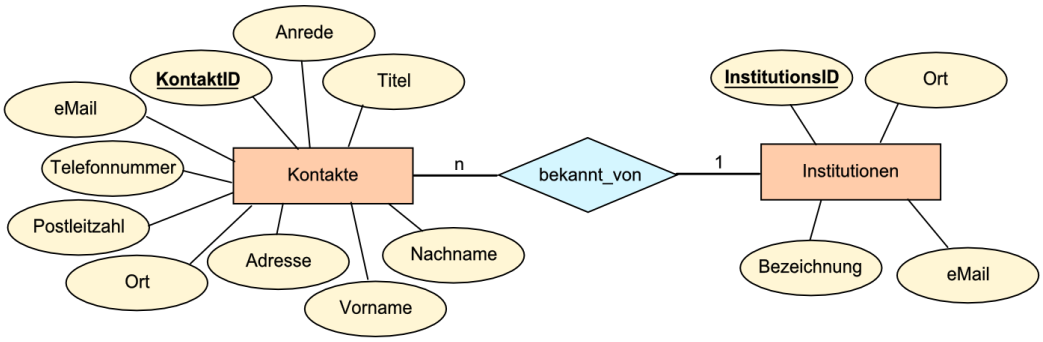


Aufgabe:

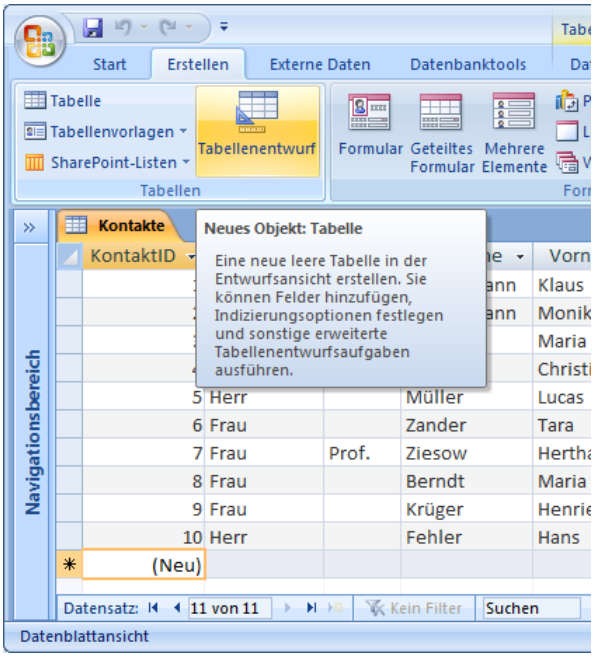
1. ***Geben Sie die Daten aus der obigen Tabellen bzw. Tabellen-Ausschnitten in die Tabelle "Kontakte! ein!***
   (Sie können auch die vollständige Tabelle aus dem Abschnitt 3.1.5.1.4. verwenden!

3.1.6.1.3. Erstellen einer neuen Tabelle über den Tabellenentwurf

Die Datenbank soll nun um die zweite Tabelle "Institutionen" erweitert werden.

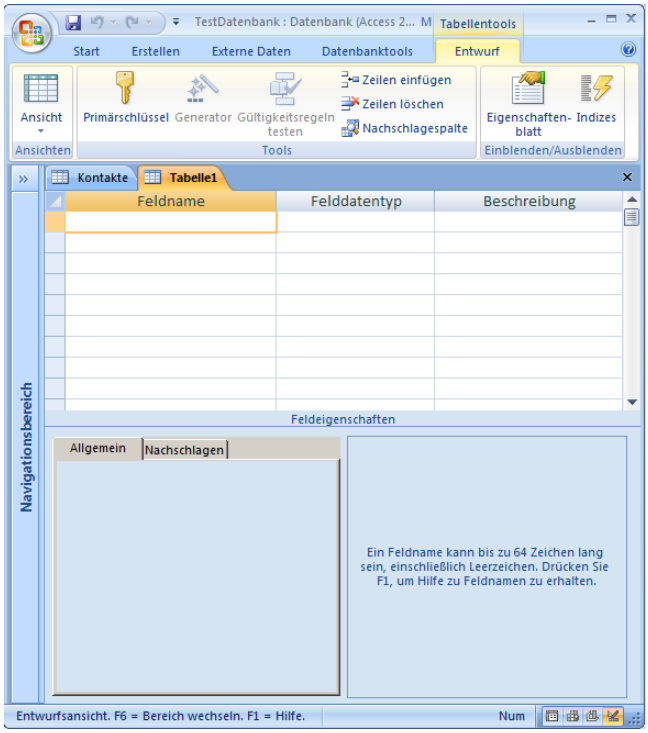


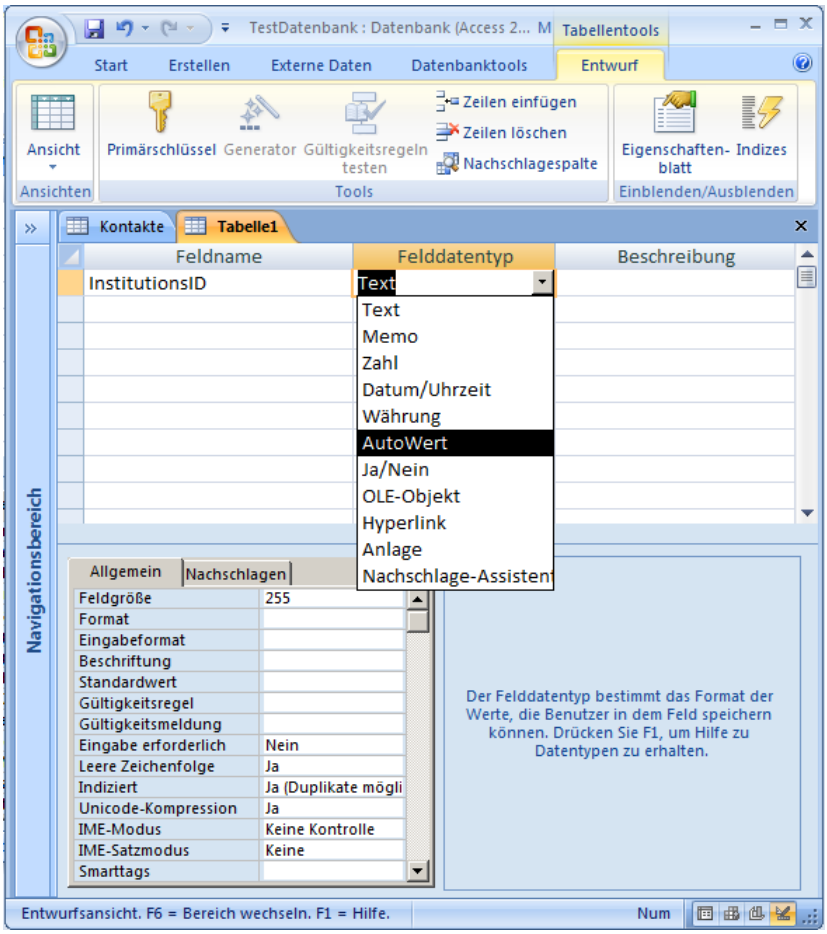
Zum effektiveren Arbeiten gehen wir gleich über den "Tabellenentwurf" aus dem Menüband "Erstellen" "Tabellen" an die Aufgabe heran.



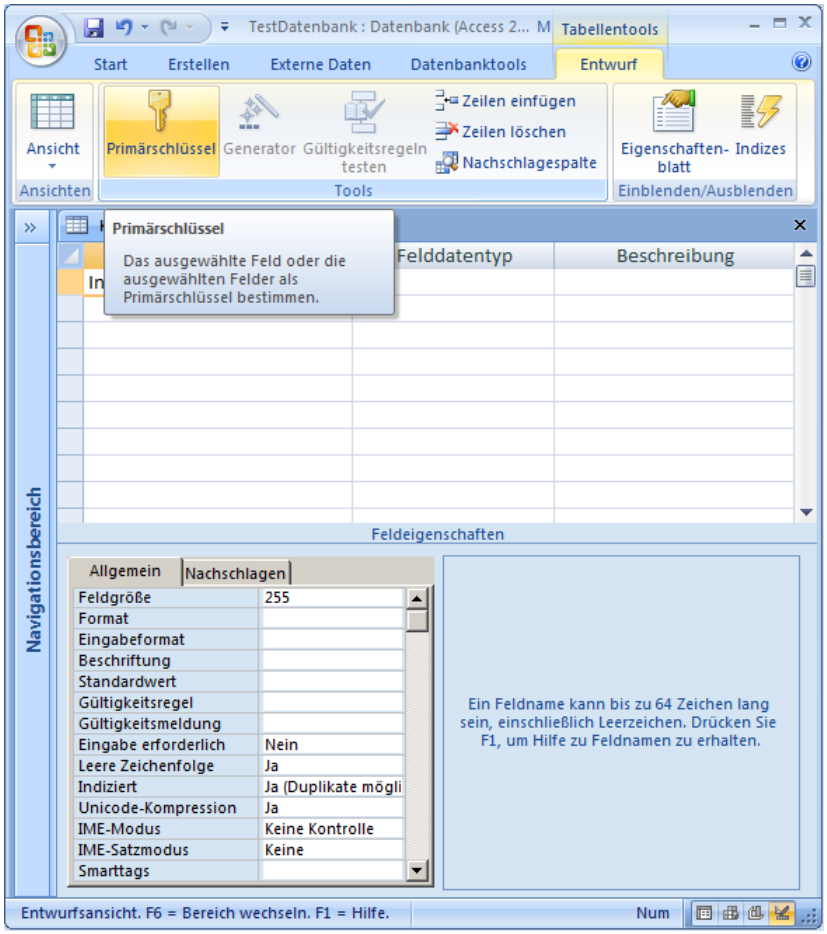
aus meiner Sicht ist die Arbeit mittels Tabellen-Editor die effektivste

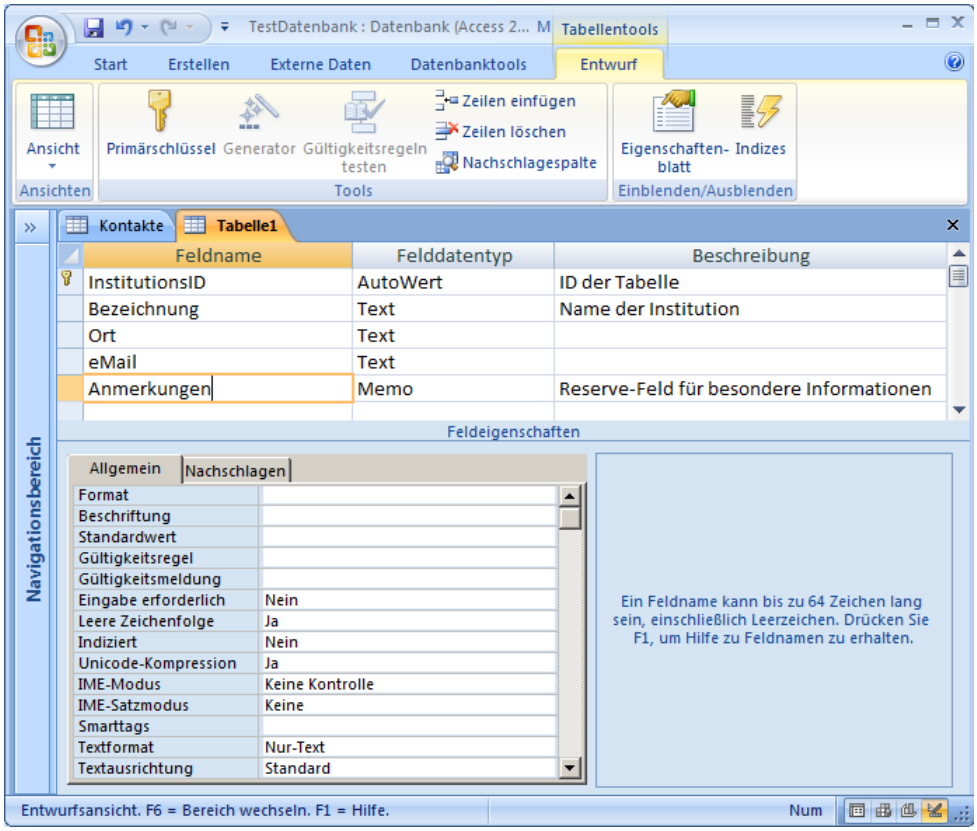
Die neue – zu erstellende – Tabelle "Institutionen" soll analog zur Besprechung in BASE erstellt werden.
Sachlich ist auch kein Unterschied zu erkennen. Es werden die gleichen Informationen zum Definieren einer Tabelle gebraucht.
Die Feldnamen leiten wir aus dem ERD ab und die Felddatentypen sind hier auch einfach definiert.
Für die ID benutzen wir "Autowert", da dies ja auch die Schlüssel-Spalte werden soll.
Alle anderen Spalten sind Texte. Wer clever vorausdenkt, fügt auch wieder eine Spalte "Anmerkungen" mit dem Typ "Memo" hinzu.



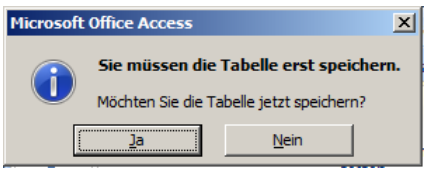


Den Primär-Schlüssel sollte man gleich zuweisen, dann vergißt man dies später nicht.
ACCESS legt sonst einen eigenen Schlüssel fest und das passt vielleicht nachher nicht immer in unsere Datenbank-Struktur mit den geplanten und vielleicht auch schon nummerisch festgelegten Verweisen.

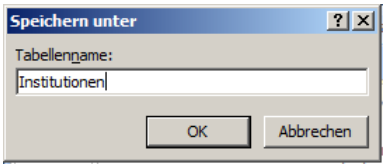




Beim Versuch die Entwurfsansicht zu verlassen oder bei einem ordentlichen Betätigen des "Speichern"-Symbols müssen wir uns entscheiden, ob der Entwurf so in die Datenbank aufgenommen werden soll Da meint hier wieder nur die Struktur der Tabelle. Die Daten sind ja noch gar nicht erfasst und werden ja auch dann – ganz Datenbank-typisch – immer gleich automatisch gespeichert.



Noch schnell den Namen für die Tabelle eingeben und die Entwurfsarbeit ist erledigt.



Aufgaben:

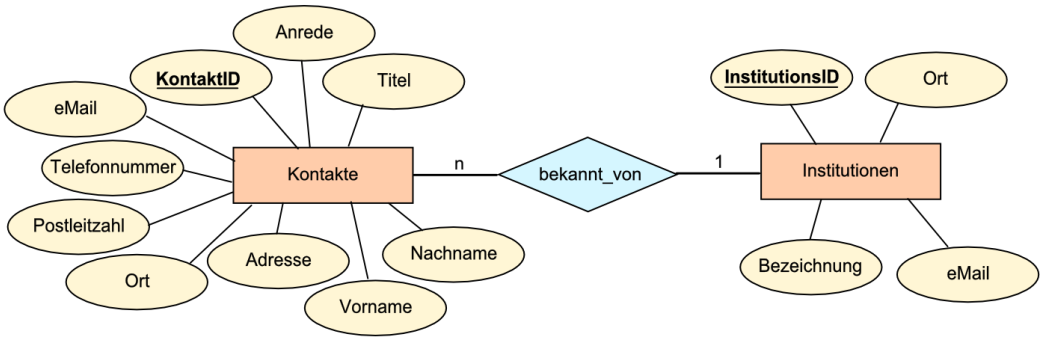
1. *Erstellen Sie die Tabellen-Struktur der Tabelle "Institutionen" wie oben angegeben!*
2. *Füllen Sie die Tabelle "Institutionen" mit den folgenden Daten!*

InstitutionsID	Bezeichnung	Ort	eMail	Anmerkungen
1	Goethe-Gymnasium	Mustern	GoeGymn@webb.de	
2	Tanzverein "Polka"	Musterhausen	post@tv-polka.de	
3	Hilfe e.V.	Meinstadt	info@hilfe.info	
4	Traditionsverein	Cedorf	tradi@verein.de	
5	Let's share	Meinstadt	letsshare@verein.de	
6	Grundschule	Mustern	gs-mustern@webb.de	

Übungs-Aufgaben zu ACCESS:

1. *Vollziehen Sie den Weg zur Erstellung einer Tabelle mit dem Assistenten nach!*
2. *Erstellen Sie mit Hilfe der Assistenten in einer neuen Datenbank ("CD-Sammlung") eine Tabelle für Ihre eigene CD-, Album-/MP3-Sammlung! (Benutzen Sie mindestens 7 Attribute!)*

3.1.6.1.4. Beziehungen zwischen Tabellen umsetzen



Die Tabellen "Kontakte" und "Institutionen" sind die Basis für die Beziehung "bekannt_von". In diese Tabelle werden nun keine echten Daten (Klar-Bezeichnungen) eingetragen, sondern nur Verweise auf die beiden aggregierten Tabellen. Und was eignet sich dabei besser als die Primär-Schlüssel der jeweils betroffenen Datensätze.
Solange wir noch keine "Formulare" zur Verfügung haben (→ 3.1.6.4. Formulare), ist die Dateneingabe noch sehr mühselig.

3.1.6.2. Abfragen

3.1.6.2.1. Erstellen einer Abfrage mit dem Assistenten

Für Anfänger ist die Assistenten-Methode wohl die einfachte Variante. Schrittweise klickt man sich die Elemente und kriterien für die Abfrage zusammen. Man kann jederzeit wieder zurück bis hin zum Anfang, um notwendige Korrekturen vorzunehmen oder vernachlässigte Kriterien mit einzubauen.

Auswahl der Felder

Bestätigung der Auswahl

Sortierung

weitere Suchbedingungen

Aliasnamen – also spezielle Namen für die Abfragen

Verschleierung der Datenbank-Interna oder Verbesserung der Nutzerfreundlichkeit

Überblick

Zusammenfassung des Assistenten

Ergebnis ist eine Tabelle

ist temporär, wird also nur erzeugt, wenn der Bedarf besteht, es werden dann die aktuellen Daten benutzt

Wer etwas über die Umsetzung der Assistenten-Schritte in eine Abfrage lernen möchte, kann sich die fertige Abfrage auch in der Entwurfs-Ansicht anzeigen lassen.

Aufgaben:

1.

2. Prüfen Sie durch ein Experiment, ob es sich bei den Abfragen (im Vergleich zu den Tabellen) nur um temporäre oder permanente Daten / Daten-Änderungen handelt!

3.

3.1.6.2.2. Erstellen einer Abfrage in der Entwurfs-Ansicht

Diese Variante der Abfragen-Erstellung ist wohl die beliebteste. Es ist sehr einfach und schnell möglich, sich die Abfragen zusammenzuklicken und die Ergebnisse sind meist sofort zufriedenstellend.
Für viele Zwecke ist auch eine Vorbereitung über den Assistenten sinnvoll. Später kann die Abfrage kopiert werden und dann in einer Kopie mit den Feineinstellungen experimentiert werden.
Der Aufruf erfolgt im oberen Teil der Teil der Abfrage-Übersicht.

Es erscheint die Entwurfs-Ansicht mit einem "Hinzufügen"-Dialog.

Für die weitere Nutzung von Daten müssen zuerst einmal die Tabellen oder andere – schon vorhandene – Abfragen hinzugefügt werden.
Wenn man genug Tabellen und / oder Abfragen hinzugefügt hat, verläßt man den "Hinzufügen"-Dialog mit "Schließen".

Aufgaben:

1. ***Öffnen Sie sich die Entwurfs-Ansicht und fügen Sie sich die Tabelle "Kontakte" hinzu!***
2. ***Realisieren Sie eine "Abfrage2 Kontake", indem Sie die Abfrage aus der Assitenten-Nutzung*** (→ 3.1.3.1.1. Erstellen einer Tabelle mit Hilfe des Assistenten) ***nachbauen!***
3. ***Versuchen Sie die Abfrage2 so zu verändern, dass die nicht angezeigte Spalte "Vorname" (ganz rechts) nicht mehr benötigt wird, aber die Sortierung trotzdem erst nach der des Nachnamens (!) getätigt wird! (Hinweis!: Überlegen Sie sich, wie ein Computer(-Programm) die Entwurfs-Ansicht lesen / interpretieren wird!)***

Prüfen, ob alle so auch für ACCESS gültig sind!?

Bedingungs-Operatoren in Abfragen

Operator / Symbol(e)	Benennung	Bemerkungen / Bedeutung
=	(ist) gleich	in Abfrage-Feldern wird der "="-Operator nicht angezeigt; → wird kein Operator angegeben, dann wird automatisch "=" angenommen
<>	(ist) ungleich	alle anderen Werte werden akzeptiert
<	(ist) kleiner (als)	alle Werte, die kleiner als der angegebene sind, werden akzeptiert
<=	(ist) kleiner oder gleich	erfüllt, wenn der Wert kleiner als der angegebene oder gleich groß, wie dieser, ist
>	(ist) größer (als)	alle Werte, die größer als der angegebene sind, werden akzeptiert
>=	(ist) größer oder gleich	erfüllt, wenn der Wert größer als der angegebene oder gleich groß, wie dieser, ist

Platzhalter / Joker-Symbole

Zeichen / Symbol	Bedeutung	
***** **%**	entspricht einer beliebigen (auch leeren) Zeichenkette bzw. eines beliebigen Wertes	
? **_**	steht für (genau) ein Zeichen in einer Zeichenkette	

Filter-Bedingungen für Abfragen

Option	Auswirkung / Bedeutung	SQL-Schlüsselwort	Bemerkungen Beispiel(e)
ohne	*keine*		
IST LEER	ist erfüllt, wenn das Feld einen NULL-Wert besitzt	IS NULL IS LEER	bei Options-Feldern (JA / NEIN) wird der unbestimmte Fall genutzt (also weder Ja noch Nein)
IST NICHT LEER	Negation von "IST LEER"; ist erfüllt, wenn das Feld einen (von NULL abweichenden) Wert besitzt	IS NOT NULL IS NOT LEER	
LIKE **WIE**	Übereinstimmung; ist erfüllt, wenn das Feld den entsprechenden Wert besitzt	LIKE	
ZWISCHEN *min* **UND** *max*	im Intervall; ist erfüllt, wenn der Feldinhalt zwischen den Angaben ***min*** und ***max*** liegt	BETWEEN *min* AND *max*	
IN (*Liste***)**	Entsprechung; Enthaltung ist erfüllt, wenn das Feld		IN (3; 6; 12; 24)

	mit einem Wert aus der (Semikolon-getrennten) übereinstimmt		IN ('Moskau'; 'Berlin'; 'New York')
NICHT …	Negation; negiert den folgenden Ausdruck	NOT …	
= WAHR	Validierung; ist erfüllt, wenn der Feld-inhalt ***wahr*** enthält	= TRUE	
= FALSCH	Falsifizierung ist erfüllt, wenn der Feld-inhalt ***falsch*** enthält	= FALSE	

3.1.6.2.3. Berechnungen in einer Abfrage

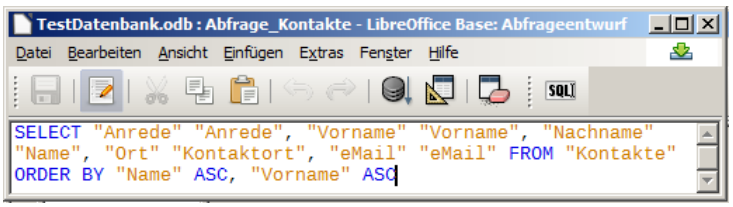
statt des Feldnamens (Attributes) werden Formeln mit den Feldnamen (z.B. "Nettopreis") ohne führendes Gleichheitszeichen eingegeben (z.B. zur Berechnung des Bruttopreises: "Nettopreis" * 1,19)
es muss dann ein Alias vergeben werden, da sonst die berechnete (temporäre) Spalte keine Überschrift hätte

Funktionen in Abfragen

Option / Funktion	**Auswirkung / Bedeutung**	**SQL-Schlüssel-wort**	**Bemerkungen**
ohne	*keine*		
Durchschnitt	berechnet das (arith-metrische) Mittel (Mittelwert) des Feldes (Attributes)	AVG	
Anzahl	zählt die Datensätze, die in der Abfrage auftauchen	COUNT	COUNT(*) .. es werden alle Da-tensätze (in der Abfrage) gezählt COUNT(*Spalte*) .. es werden die Datensätze gezählt, bei denen die ***Spalte*** einen Nicht-NULL-Wert enthält
Maximum	ermittelt den größten Wert des Feldes (Attributes)	MAX	
Minimum	ermittelt den kleinsten Wert des Feldes (Attributes)	MIN	
Summe	berechnet die Summe der Feld-Werte (Attribut-Werte)	SUM	
Gruppiert	gruppiert die Datensätze nach den Werten im aus-gewählten Feld (Attribut)	GROUP BY	

3.1.6.2.4. Erstellen einer Abfrage mittels SQL

Diese Methode zum Erstellen einer Abfrage ist mehr was für Profis oder Programmierer. Für einfache Abfragen ist es aber ein gleichwertiges Mittel. Komplizierte Abfragen lassen sich besser über den Assistenten (→ 3.1.3.2.1. Erstellen einer Abfrage mit dem Assistenten) oder die Entwurfs-Ansicht (→ 3.1.3.2.2. Erstellen einer Abfrage in der Entwurfs-Ansicht) zusammenstellen und austesten.
Als Beispiel sei hier der Code für die mit dem Assisten erstellte Abfrage aufgezeigt.
Man kann den Text fast verstehen.
Mehr dazu im SQL-Teil dieses Skriptes (→ 5. Datenbanken - SQL; → 5.1.x. CREATE VIEW).
Zum langsamen Herantasten an SQL kann man sich zuerst einmal den SQL-Code von fertigen – funktionierenden – Abfragen ansehen.



Später ist man vielleicht schneller, wenn man nur noch den Code eingibt. Das setzt aber auch leider immer voraus, dass man die ganzen Namen von Tabellen, anderen Abfragen und den Attributen im Kopf oder auf einer Übersicht hat. Diese werden immer – exakt geschrieben – für einen funktionierenden SQL-Code gebraucht.
Letztendlich lesen sich SQL-Codes fast wie eine umgangssprachliche – wenn auch englische – Arbeits-Aufforderung. Dieses war ja auch ein erklärtes Ziel bei der Entwicklung von SQL.

3.1.6.3. Berichte

In modernen Datenbanken verschmelzen Abfragen und Berichte immer stärker. Ursprünglich waren vor allem Berechnungen und Zusammenfassungen Elemente von Berichten.
Der ursprüngliche Gedanke eines berichtes ist es ja auch den Zustand einer Datenbank, oder einzelner Elemente davon, dauerhaft zu dokumentieren. Ein gutes Beispiel ist der Quartals-Bericht oder eine Jahres-Abschluß. Diese Berichte konnten dann ausgedruckt und abgeheftet werden. Schon einige Tage nach den Stichtagen waren die Datenbanken dann in anderen Zuständen. Heute lassen sich fast alle Aktionen auch Zeit-bezogen auswerten. Dadurch kann der Jahres-Bericht auch erst ein halbes Jahr später sicher und immer wieder gleich abgefragt werden.
Berichte eignen sich auch heute noch besonders gut, wenn man mehr oder wenige umfassende Auszüge / Übersichten seiner Daten benötigt.
Da in unserer einfachen Datenbank keine Zeit-basierte Erfassung und Protokollierung der Daten eingebaut ist, können wir uns die aktuelle Situation einer Tabelle oder eines Ausschnittes aus der Datenbank gut als Bericht zusammenstellen lassen.

Aufgaben:

1.
2. *Erstellen Sie einen Bericht der Kontakte-Datenbank, der eine Liste der Kontakte, alphabetisch geordnet und gruppiert nach den Einrichtungen zeigt! Am Ende jedes Einrichtungs-Blockes und am Ende des Berichtes sollen auch die Anzahlen der Kontakte auftauchen!*
3. *Erweitern Sie den Bericht von Aufgabe 2 noch um die Anzahl der Einrichtungen!*

3.1.6.4. Formulare

ACCESS zeichnet sich durch eine Menge praktischer Hilfmittel, Funktionen und Assistenten zur Erstellung und Benutzung von Formularen aus. Mit ein wenig Geschick und Planung gelingt einem schnell ein Konstrukt aus Formularen, die einem unbedarften Nutzer eine eigene Applikation vorgaukeln. Nicht in allen Fällen muss ein Nutzer wissen, das er jetzt mit seiner Heiligkeit – einer Datenbank – arbeitet. Die meisten Nutzer verbinden damit sowieso andere Vorstellungen, als ein (praktischer) Informatiker.

Aufgaben:

1.
2.
3.

Hilfe (Tutorial) zu microsoft ACCESS
https://www.access-tutorial.de/

3.1.7. Datenbanken mit SQLite

sachlich gesehen ist SQLite eine Bibliothek aus Programmen, die ein relationales Datenbank-System ermöglichen

Logo von SQLite
Q: de.wikipedia.org
(D. Richard Hipp)

besonders für eingebettete Datenbank-Systeme (embedded database (management system)) gedacht, also integriert in andere Programme oder Systeme
tritt praktisch nach außen nicht in Erscheinung, macht im Hintergrund die Arbeit

kann man sich wie eine Runtime-Version vorstellen, der Nutzer / Programmierer bekommt hoch-professionelle Programm-Routinen, die er in seine Projekte einbauen bzw. in diesen dann nutzen kann

von Richard HIPP (1961 -) in der Programmiersprache C (sehr Maschinen-nah) geschrieben, deshalb klein und schnell

Schnittstellen für Java und C++
z.B. in Python ab Version 2.5 schon enthalten
wird von vielen anderen System (Adobe AIR) oder Programmen (Apple Safari; mozilla Firefox; google Chrome; Skype) genutzt

mit PHP nutzbar

```php
$db = new SQLite3('test.db');
$result = $db->query('SELECT * FROM tbl2');

while($user = $result->fetchArray(SQLITE3_ASSOC)){
  var_dump($user);
}
```

ähnlich erfolgt Zugriff in anderen Programmier- bzw. Skript-Sprachen

Vorteile von SQLite:

einfach, schnell, klein
sehr kleine, kompakte Implementierungen, deshalb auch auf älterer, wenig Leistungs-fähiger Hardware nutzbar (es gibt auch Umsetzung, die auf einem Raspberry Pi funktioniert!)
erzeugt eine Datei pro Datenbank, kleine Dateien
für Windows, Linux, Unix, Android, OS X, Symbian OS X, iOS, Windows Phone, … verfügbar
Datei kann Betriebssystem-übergreifend genutzt werden
funktioniert / gibt es auch (in Verbindung mit bestimmten Fontends) als portable App (z.B. integriert im IoStick von Tino HEMPEL → www.tinohempel.de)
erfüllt Sprachstandard SQL92
schnell
von Konsole oder aus anderen Programmen heraus nutzbar
gemeinfreie Lizenz (Public domain)
Konfigurations-frei
Server-los (funktioniert ohne Server)
in sich geschlossenes / eigenständiges System (self contained), (netzunabhängig)
Transaktions-basiert
sehr verbreitet → viele Hilfen / Foren / …
es existiert eine ODBC-Schnittstelle

arbeitet auch mit Open-/Libre-Office zusammen
in-memory-Nutzung möglich (Datenbank nur im RAM; keine persistente Speicherung / Haltung der Daten)

Nachteile:

keine eigene Bedienoberfläche (nur Konsolen-Bedienung / -Benutzung möglich bzw. Zugriff über andere Programme)
Tabellenstrukturen lassen sich nachträglich nur begrenzt ändern, in neueren Versionen lassen sich aber schon Spalten hinzufügen und Tabellen und Spalten umbenennen
bestimmte Nutzer- / Zugriffs-Rechte sind nicht umgesetzt – also sind nicht nutzbar (diese Funktionen müssen durch das übergeordnete Programm realisert werden!)
fehlende Client-Server-Architektur (Schreib-Zugriffe verschiedener Prozesse / Threads / Programme / Nutzer können nur nacheinander abgearbeitet werden)
fehlende Typ-Sicherung (fehlerhafte Eingaben werden immer in Texte umgewandelt)
nicht in deutsch verfügbar
nicht für große Datenbestände

3.1.7.0. Voraussetzungen, Einschränkungen, Download, Installation

für schulische Zwecke (Arbeiten im Unterricht, Hausaufgaben, Projekte, …) bietet sich die fertige(n) Installation(en) auf dem IoStick von Tino HEMPEL (→ www.tinohempel.de) an

auf der Normal-Version des Io-Sticks gibt es die folgenden graphischen SQLite-Bedien-Programme (Fontend's) – schon fertig installiert und eingerichtet.
Man muss nur noch das passende Programm auswählen, und kann sofort anfangen zu arbeiten.

SQLite-Programme auf dem IoStick

• **SQLiteStudio**	Fontend mit dem größten Leistungs-Umfang sehr empfehlenswert → 3.1.7.1.3. Arbeiten mit dem SQLiteStudio
• **SQLite Database Browser**	Fontend zum Tabellen-Handling → 3.1.7.1.1. Arbeiten mit dem SQLite Database Browser
• **SQLiteman**	→
• **SQLite Manager**	→ 3.1.7.1.2. Arbeiten mit dem SQLite Manager

Sollten keine speziellen Argumente für ein anderes Programm sprechen, würde ich ab hier die Weiterarbeit mit dem SQLiteStudio empfehlen (→3.1.7.1.3. Arbeiten mit dem SQLiteStudio). In dessen Bespechung gehen wir auch auf weitere Konzepte der Datenbanken ein. Diese können teilweise auch von anderen Programmen realisiert werden.

Die Abitur-Version bietet nur die Programme:

SQLite-Programme auf dem IoAbiStick
(in Mecklenburg-Vorpommern zum Abitur zugelassen)

• **SQLiteStudio**	Fontend mit dem größten Leistungs-Umfang sehr empfehlenswert → 3.1.7.1.3. Arbeiten mit dem SQLiteStudio
• **SQLite Database Browser**	→ 3.1.7.1.1. Arbeiten mit dem SQLite Database Browser
• **SQLite Manager**	(nur auf älteren Versionen) → 3.1.7.1.2. Arbeiten mit dem SQLite Manager

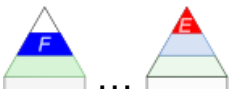
Der Vollständigkeit halber sei hier noch erwähnt, dass der Normal-Stick auch noch weitere Datenbank- und SQL-Prgramme bietet.

(weitere) Datenbank- und SQL-Programme auf dem IoStick

• **HeidiSQL**	graphisches Fontend für mySQL
• **Webserver mit mySQL**	fertig konfigurierter Webserver mit mySQL als Datenbank-System
• **LibreOffice BASE**	Datenbank-System / -Programm aus der Office-Suite von LibreOffice.org bzw. OpenOffice.org

Links:
http://www.sqlite.org
http://www.tinohempel.de
http://www.libreoffice.org
http://www.openoffice.org ; http://www.openoffice.org/de/

3.1.7.0.1. SQLite auf einem Raspberry Pi



Datenbanken sind nur was für große Rechner oder Server – könnte man denken: Aber da täuscht man sich gewaltig. Es geht auch im Mini-Maßstab. Gerade sogenannte eingebettete Datenbanken eignen sich auch für kleine Systeme und Anwendungen.

Der Mini-Rechner RaspberryPi ist ein gutes Beispiel für ein Arbeiten im Super-Kleinen. Wer sich intensiver mit dem Gerät beschäftigt, merkt schnell, dass der RaspberryPi ein Tausend-sassa ist.
Der RaspberryPi 2 hat mit 1 GB RAM und einem Vierkern-Prozessor schon einiges unter der Haube.
Der aktuelleste Pi (Version 4 mit 1 bis 8 GB RAM) ist noch Leistungs-stärker.
Mit ziemlich großer Sicherheit funktioniert aber auch der Raspberry Pi zero – ein echter Zwerg unter den Einplatinen-Rechnern.

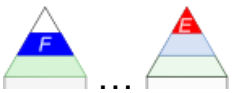
Eine Datenbank auf SQLite-Basis ist ebenso kein Problem, wie ein laufendes Libre- oder Open-Office mit BASE.

Natürlich merkt man die sparsam ausgestattete Hardware im Arbeits-Betrieb, aber nicht immer geht es um Schnelligkeit und riesige Datenmengen.
Gerade für schulische Experimente ist ein langsamerer Ablauf angenehmer.

Und gerade zum Ausprobieren ist Übersichtlichkeit und ein beobachtbares Tempo ein gutes Hilfsmittel.

da die typischen Betriebssysteme auf dem Raspberry Pi Linux-basiert sind, sind die Installation und der betrieb der Datenbank etwas spezieller
es gibt aber vile Hilfe im Internet (Tutorial's, F&A, Foren, …)

3.1.7.0.2. Daten-Typen in SQLite



Basis-Datentypen

• **NULL**	NULL-Wert (Attribut hat NULL-Wert)
• **INTEGER**	Vorzeichen-behaftete Ganzzahl 6 verschiedene Größen / Längen möglich für Primärschlüssel wird Standard-Typ empfohlen
• **REAL**	Fließkomma-Zahl / Gleitkomma-Zahl 8 Byte IEEE Fließkomma-Zahl
• **TEXT**	Text im Text-Typ der Datenbank (UTF-8, UTF-16BE, UTF-16LE)
• **BLOB**	variabler binärer Typ es wird das gespeichert, was eingegeben wurde

Hinweis:
Es gibt in SQLite keinen Datentyp BOOLEAN. Als Äquivalent wir empfohlen die Interger-Werte 0 für false und 1 für true zu benutzen.

spezielle Datentypen

• **Datum**	kann in verschiedenen Versionen gespeichert sein • als Text → ISO8601-String: YYYY-MM-DD HH:MM:SS.SSS • als Real-Zahl → Greenwich-Zeit im Julianischen Kalender nach dem 24.November 4714 v.Chr. als Integer-Zahl → UNIX-Zeit: Anzahl der Sekunden seit dem 01. Januar 1970 00:00 UTC
• **Numeric**	Datenbank-System wandelt Eingaben in ein passendes Zahlenformat um (auch Texte)
•	
•	

3.1.7.1. Nutzung von SQLite

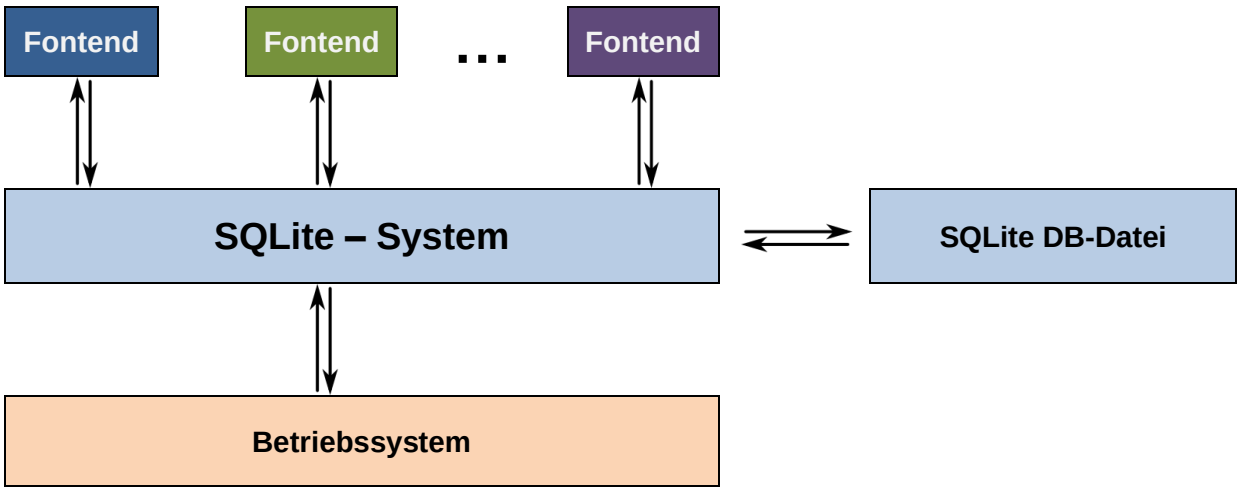
einfache Fontend's zum direkten Arbeiten
stammen alle von Dritt-Anbietern

Fontend's / Benutzeroberflächen für SQLite

• **SQLite Browser (DB Browser for SQLite; SQLite Database Browser)**	• graphische Oberfläche • für Win (inklusive portApps), MacOS, Linux • Source code verfügbar → praktisch auf weitere Systeme übertragbar → 3.1.7.1.1. Arbeiten mit dem SQLite Database Browser
• **SQLite Manager**	• Konsolen-orientierte Oberfläche • auch in deutsch verfügbar → 3.1.7.1.2. Arbeiten mit dem SQLite Manager
• **SQLite Expert**	• graphische Oberfläche • sehr umfangreich und Leistungs-stark • diverse Importe • SQL-Scripts- und PDF-Export • für Windows; freie Basis-Version, erweiterte professionelle Version erhältlich •
• **SQLite Adminstrator**	• sehr Leistungs-fähiges Tool • enthält guten SQL-(Text-)Editor (automatische Code-Komplementierung; Syntax-Highlighting) •
• **Navicat for SQLite**	• intuitiv, Praxis-orientiert • für Win, MacOS und Linux verfügbar; auch für andere Datenbanken verfügbar • kostenpflichtig (14tägiger Test möglich) •
• **SQLite Studio**	• sehr Leistungs-fähig • gut für die schulische Umgebung geeignet • für Win (inklusive portApps) → 3.1.7.1.3. Arbeiten mit dem SQLiteStudio
• **SQLite ???**	•

SQLite Expert wohl am umfangreichsten, aber direkte Installation notwendig, was in den meisten Fällen kein Problem sein sollte
in schulischen / Ausbildungs-Situationen sind die anderen Oberflächen vorteilhafter / unproblematischer
Meine Empfehlung geht dabei in Richtung SQLite Studio.

Man kann sich aber jederzeit umorientieren, da alle Fontends auf SQLite als Basis-System aufsetzen. Das Einzige, was sich ändert, ist die vielleicht etwas andere Bedienung und der Leistungs-Umfang des Fontends.

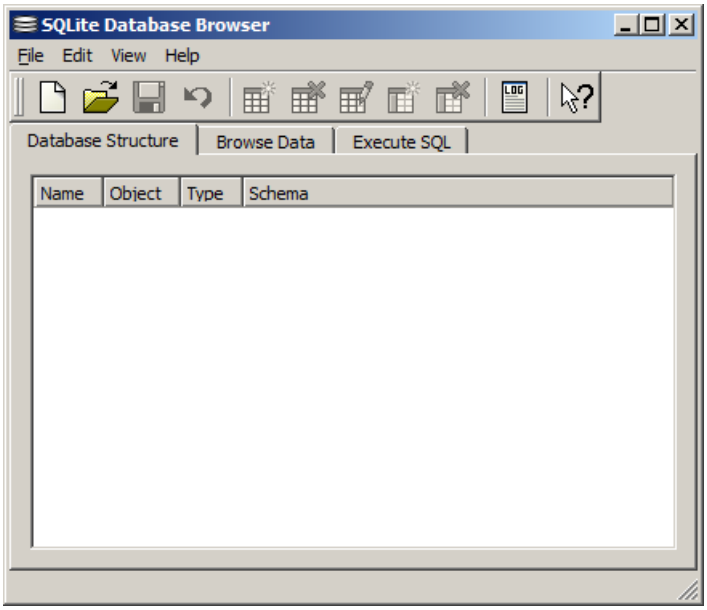


3.1.7.1.1. Arbeiten mit dem SQLite Database Browser

Der SQLite Database Browser ist ein recht einfach gestricktes Tool. Besonders geeignet ist der Browser für die Erstellung und das bestrachten von Datenbank-Strukturen.
Eine praktische Datenbank-Arbeit ist z.T. schwierig bis hin zu ganz unmöglich. Dafür ist der Browser einfach nicht gedacht.
Assistenten oder andere Hilfs-Tools sucht man im SQLite Database Browser deshalb auch vergeblich.

3.1.7.1.1.x. Erstellen einer Beispiel-Datenbank (Kontakte-Institutionen)

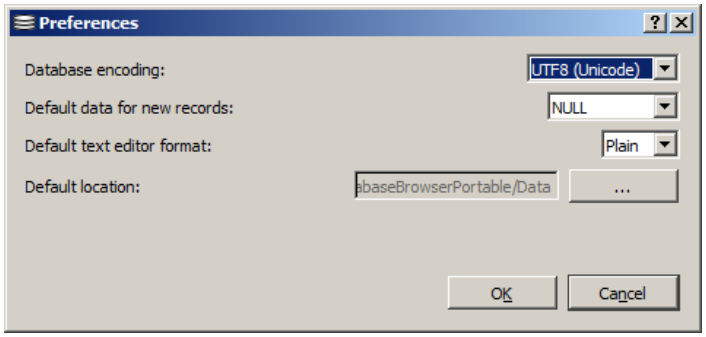
Oberfläche für SQLite



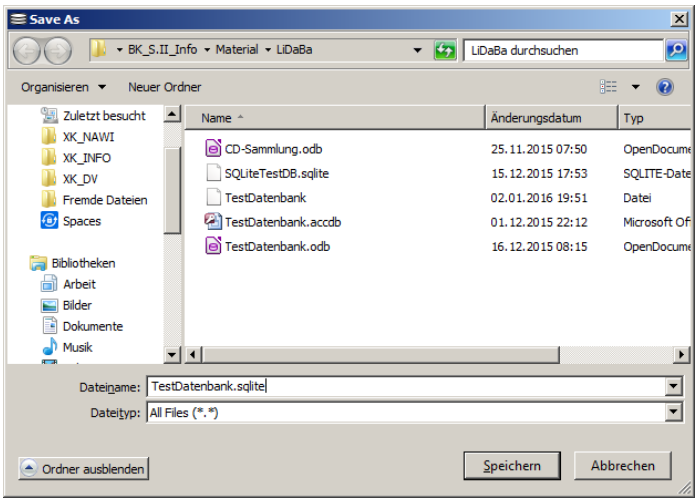
Der Reiter "Browse Data" zeigt den Daten-Inhalt einer auszuwählenden Tabelle (oben links). Neben einem einfachen Datensatz-Navigator gibt es sehr einfache Möglichkeiten zum Eingeben und Löschen von Datensätzen.

Bei "Executable SQL" können direkte SQL-Anweisungen eingegeben werden.

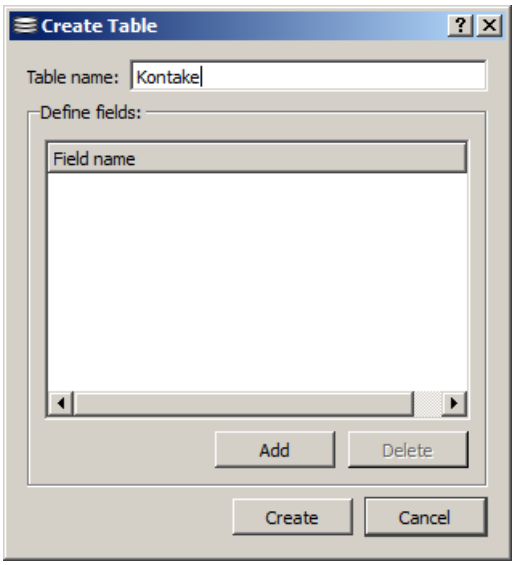
selbst die Benutzer-Oberfläche bietet nur sehr wenige Einstellungs-Möglichkeiten



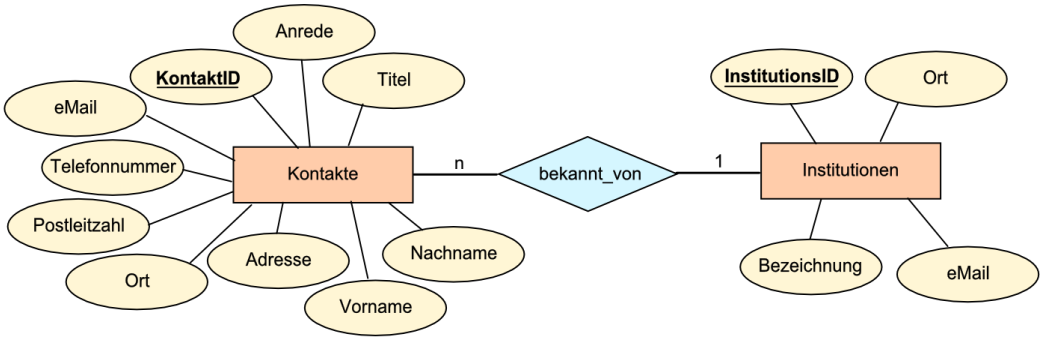
Erzeugen der neuen Datenbank(-Datei)

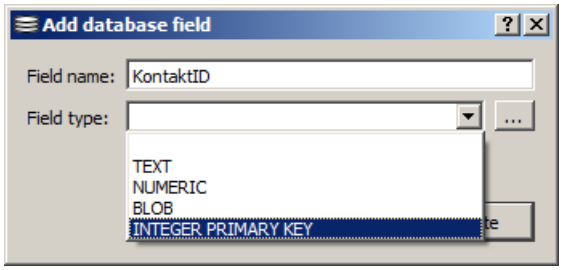


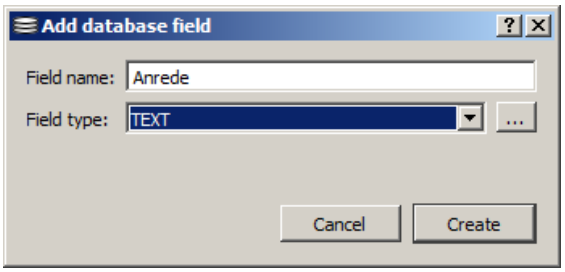
Erstellen einer Tabelle



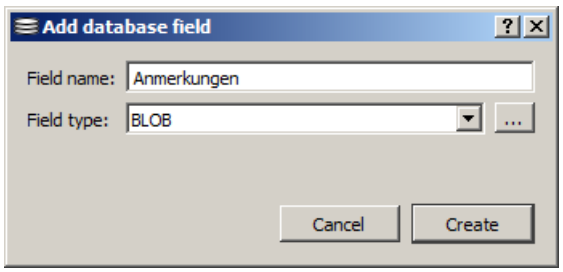
Als zu realisierende Tabelle wollen wir uns zuerst an die "Kontakte" machen. Grundlage ist das vorne besprochene und bei BASE und ACCESS ebenfalls umgesetzte folgende ERD:



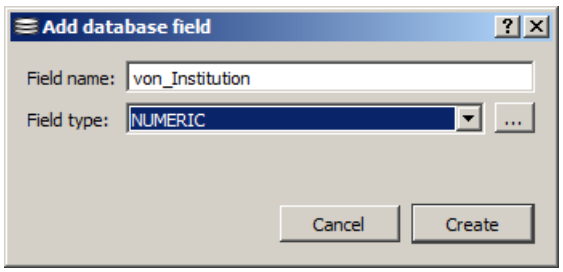




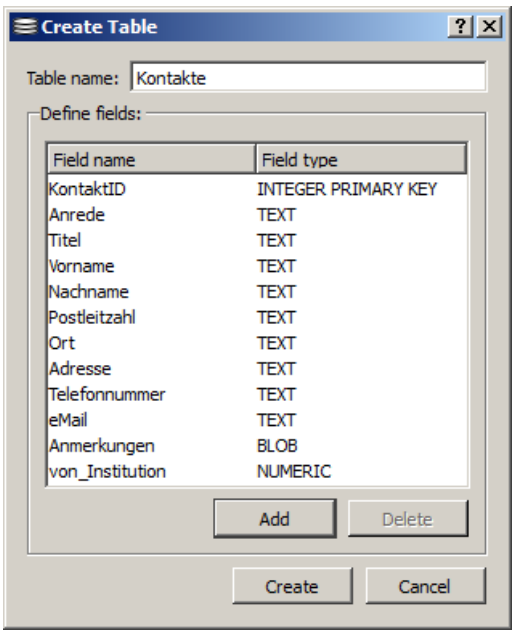
Das optionale "Anmerkung"s-Feld für "alle" Fälle kann hier natürlich auch weg gelasen werden. Das ERD gibt solche Informationen jedenfalls nicht her.



Wer die Kapitel zu ACCESS und BASE schon durchgearbeitet hat, weiss um die notwendige Spalte "von_Institution" für die Realisierung der Beziehung "bekannt_von" aus dem ERD.
Für die anderen Nutzer wird die notwendige Änderung / Ergänzung später ausführlich beschrieben (→ Tabellen-Struktur bearbeiten).



Nachdem alle Felder / Attribute defniert sind, kann die Tabelle kreiert ("Create") werden.



Am Schluß kann man sich das SQL-Äquivalent in der Spalte "Schema" ansehen. Wer es nicht mit der Maus und dem Zusammenklicken hat, könnte den SQL-Text auch gleich unter "Execute SQL" eingeben (→ Alles schneller mit SQL).

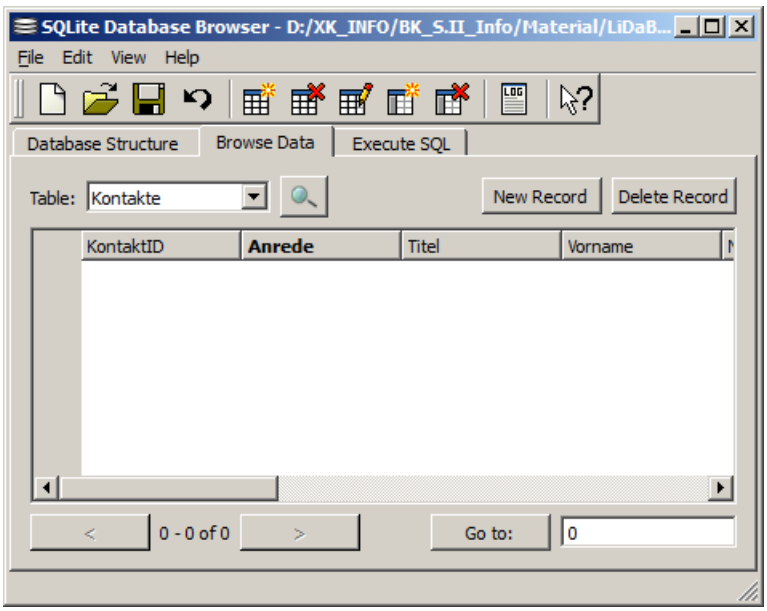
Im Database Browser können wir uns auch den SQL-Text unserer zusammengeklickten Tabellen-Struktur ansehen.

```
CREATE TABLE Kontakte (KontaktID , Anrede , Titel , Vorname , Nachname ,
Postleitzahl , Ort , Adresse , Telefonnummer , eMail , Anmerkung ,
von_Institution )
```

Die typisierte Tabellen-Erstellung und das Setzen des Primär-Schlüssels wird weiter hinten mit einer vollständigen SQL-Anweisung realisiert (→ Alles schneller mit SQL Erstellen der Tabellen).

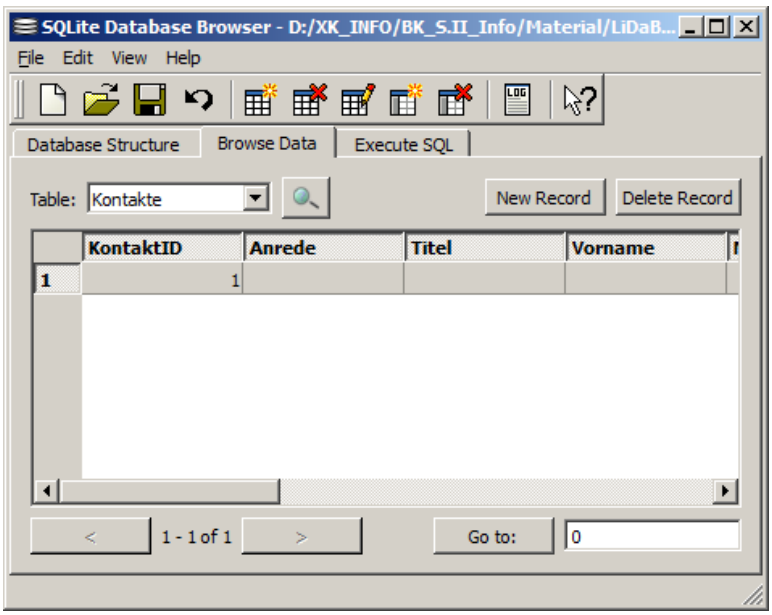
Daten eingeben

Der Reiter "Browse Data" zeigt den Daten-Inhalt einer auszuwählenden Tabelle (oben links). Neben einem einfachen Datensatz-Navigator (unten) gibt es sehr einfache Möglichkeiten zum Eingeben und Löschen von Datensätzen.



Mit "New Record" wird lediglich eine neue Zeile – also ein Datensatz – angelegt. Als einziger Wert ist der primäre Schlüssel vorgegeben.

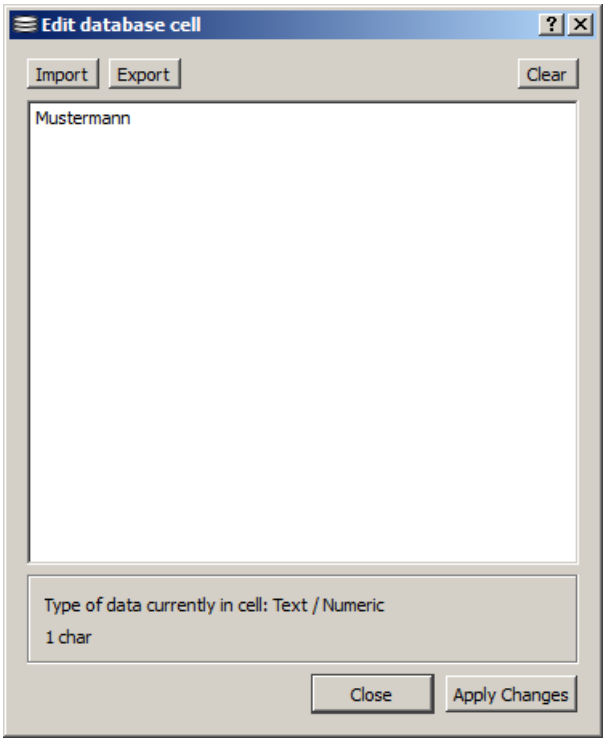
Die Eingabe einzelner Werte für die einzelnen Spalten (Attribute) ist eine mühselige Angelegenheit. Da lohnt es sich entweder mit einem Daten-Import oder mit einer Eingabe über SQL-Anweisungen nachzudenken.



im zentralen Text-Feld kann man die Eingaben schreiben

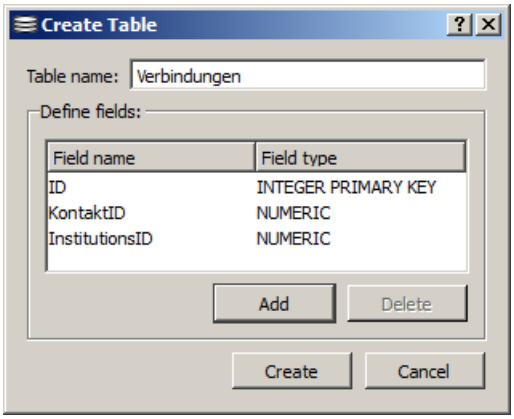
Die Eingabe wird nur übernommen, wenn man "Apply Changes" anklickt. "Close" entspricht einem Abbruch.

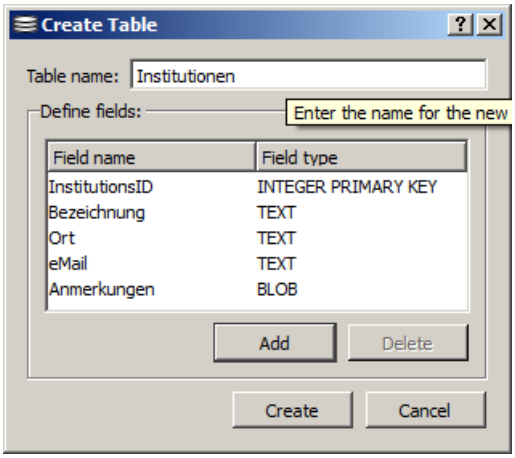
Wem das Eingeben auf diese Weise zu nervig wird, der sollte sich schon mal mit der Vorgehensweise mit SQL informieren (→ Alles schneller mit SQL) und vielleicht dahin wechseln.



Wichtig ist hier, dass man sich regelmäßig um das Speichern kümmert. Anders als bei den klassischen Datenbanken werden die Daten erst mit einem expliziten "Speichern" in die Datenbank-Datei geschrieben!

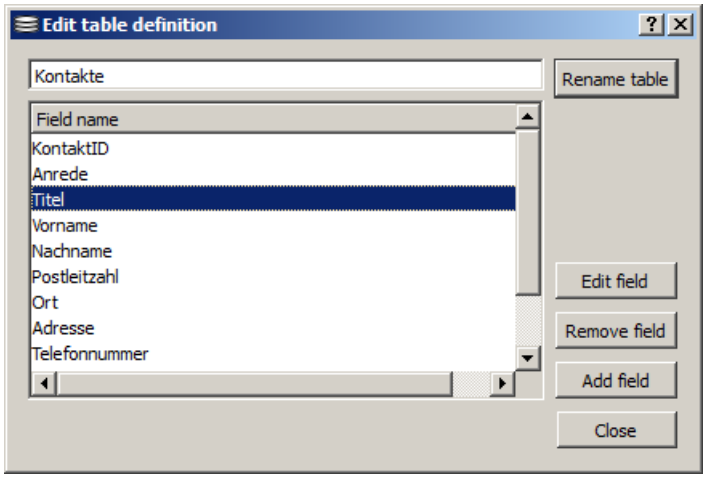
Genau so legen wir nun die beiden weiteren Tabellen "Institutionen" und "Verbindungen" an und füllen sie mit den notwendigen Daten.





Tabellen-Struktur bearbeiten

Der eine oder andere Fehler lässt sich bei " Modify Table" korrieren. Das Ändern funktioniert beim Namen der Tabelle, genauso wie bei den Feldern.
Aber Vorsicht, werden z.B. Spalten / Felder gelöscht, dann verschwinden auch die enthaltenen Daten. In so einem Fall muss man sich dann das "Speichern" genau überlegen, denn bis dahin ist alles nur im Speicher passiert.



Alles schneller mit SQL

Erstellen der Tabellen

```
CREATE TABLE Kontakte (KontaktID INTEGER PRIMARY KEY, Anrede TEXT, Titel
TEXT, Vorname TEXT, Nachname TEXT, Postleitzahl TEXT, Ort TEXT, Adresse
TEXT, Telefonnummer TEXT, eMail TEXT, Anmerkungen BLOB, von_Institution
NUMERIC);
```

```
CREATE TABLE Institutionen (InstitutionsID INTEGER PRIMARY KEY, Bezeich-
nung TEXT, Ort TEXT, eMail TEXT, Anmerkung BLOB);
```

```
CREATE TABLE Verbindungen (ID INTEGER PRIMARY KEY, KontaktID NUMERIC, In-
stitutionsID NUMERIC);
```

Eingeben von Daten

```
INSERT INTO Kontakte VALUES(1, 'Herr', 'Dr.', 'Klaus', 'Mustermann',
'12345', 'Musterhausen', 'Musterstr. 13', '0123456789',
'mustermann@webb.de', '', 1);
```

Zur Eingabe-Hilfe kann man sich ja die unveränderlichen SQL-Teile in die Zwischenablage kopieren und dann bei jedem Datensatz wieder einfügen. Dann müssen nur noch die eigentlichen Daten eingearbeitet werden.

```
INSERT INTO Kontakte VALUES(, '', '', '', '', '', '', '', '', '', );
```

Es macht zwar den Eindruck, man müsse unwahrscheinlich viel schreiben, aber beim genaueren Hinsehen fällt auf, dass man ja auch in der Einzeleingabe alles tippen musste. Dazu kommen dann noch die vielen Maus-Aktionen.
Hier noch die fehlenden SQL-Anweisungen für die restlichen Datensätze für die Tabelle "Kontakte".

```
INSERT INTO Kontakte VALUES(2,'Frau','','Monika','Mustermann',
'12345','Musterhausen','Musterstr. 13','0123456789',
'musterfrau@webb.de','',2);
INSERT INTO Kontakte VALUES(4,'Frau','','Maria','Muster',
'23456','Mustern','Am Musterweg 3','0234567890',
'm.muster@tee-online.de','',4);
INSERT INTO Kontakte VALUES(5,'Herr','','Christian','Bauer',
'67890','Berg am Fluß','Waldallee 78','04567890901',
'Chr.Bauer@geemx.de','',5);
INSERT INTO Kontakte VALUES(6,'Herr','','Lucas','Müller',
'54321','Bedorf','Feldweg 7','09998765',
'Mueller@tee-online.de','',1);
INSERT INTO Kontakte VALUES(7,'Frau','','Tara','Zander',
'23456','Mustern','Hauptstr. 6e','0777711',
'Ta.Zan@webb.de','',6);
INSERT INTO Kontakte VALUES(8,'Frau','Prof.','Hertha','Ziesow',
'88888','St. Muster','An der B777','0543861',
'Prof.H.Ziesow@tee-online.de','',4);
INSERT INTO Kontakte VALUES(12,'Frau','','Maria','Berndt',
'67890','Berg am Fluß','Hans-Wald-Str. 4','0456783067',
'','',3);
INSERT INTO Kontakte VALUES(37,'Herr','','Henriette','Krüger',
```

```
'76543','Meinstadt','Feldweg 7','06543210',
'post@h-krueger.de','',1);
INSERT INTO Kontakte VALUES(3,'Herr','','Hans','Fehler',
'34343','Glückstadt an der Pech','Blumenweg 48','088088088',
'H.Fehler@webb.de','',1);
```

Da wird schnell klar, warum viele Datenbank-Freaks gleich in SQL oder mit passenden Text-Editoren arbeiten. Einige der Editoren bieten sogar Syntax-Highlighting (Syntax-Hervorhebung).
Hier die SQL-Anweisungen für die restlichen zwei Tabellen-Inhalte mit so einem Syntax-Highlighting. Man erkennt deutlich die verschiedenen Syntax-Elemente und damit die Struktur der Anweisungen.

```
INSERT INTO Institutionen VALUES(1,'Goethe-Gymnasium','Mustern',
'GoeGymn@webb.de','');
INSERT INTO Institutionen VALUES(2,'Tanzverein "Polka"','Musterhausen',
'post@tv-polka.de','');
INSERT INTO Institutionen VALUES(3,'Hilfe e.V.','Meinstadt', 'in-
fo@hilfe.info','');
INSERT INTO Institutionen VALUES(4,'Traditionsverein','Cedorf', 'tra-
di@verein.de','');
INSERT INTO Institutionen VALUES(5,'Let''s share','Meinstadt',
'letsshare@verein.de','');
INSERT INTO Institutionen VALUES(6,'Grundschule','Mustern',
'gs-mustern@webb.de','');
```

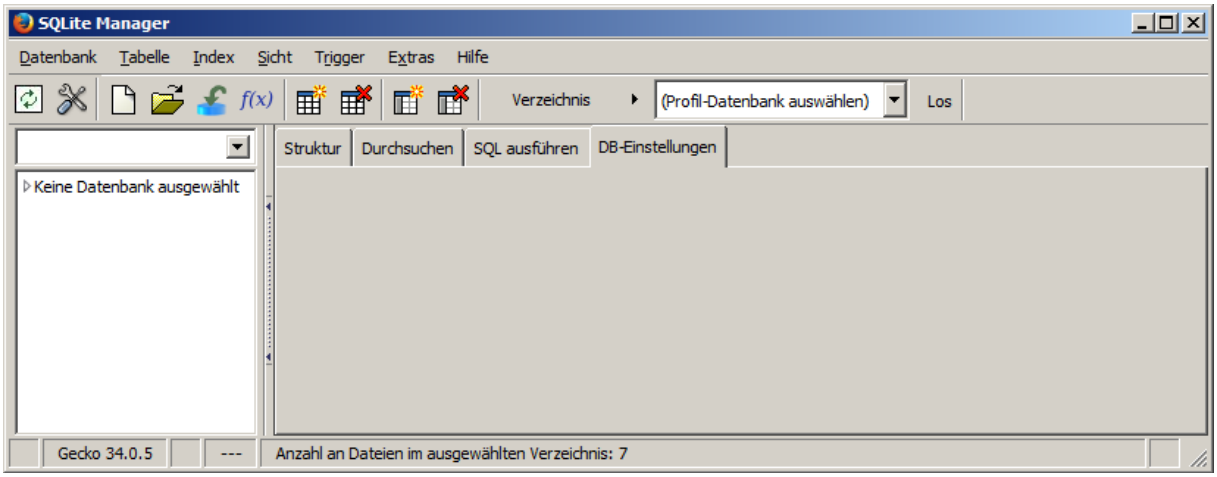
```
INSERT INTO Verbindungen VALUES(0,1,1);
INSERT INTO Verbindungen VALUES(1,2,2);
INSERT INTO Verbindungen VALUES(2,3,1);
INSERT INTO Verbindungen VALUES(3,4,4);
INSERT INTO Verbindungen VALUES(4,5,5);
INSERT INTO Verbindungen VALUES(5,6,1);
INSERT INTO Verbindungen VALUES(6,7,6);
INSERT INTO Verbindungen VALUES(7,8,4);
INSERT INTO Verbindungen VALUES(8,12,3);
INSERT INTO Verbindungen VALUES(9,37,1);
```

Fehler oder Probleme lassen sich so schnell finden und korrigieren.
So, oder so ähnlich sieht das dann in einem anderen Editor aus. Die abgespeicherten Text-Dateien (mit dem SQL-Anweisungen) können dann später importiert werden.
Der umgekehrte Weg – also ein Export des SQL-Anweisungen – ist möglich. Dazu müssen die SQL-Anweisungen nicht wirklich eingegebene worden sein. Der SQLite Database Browser exportiert eine ganze Datenbank auch als SQL-Quelltext.

Links:
Q: http://sqlitebrowser.sourceforge.net

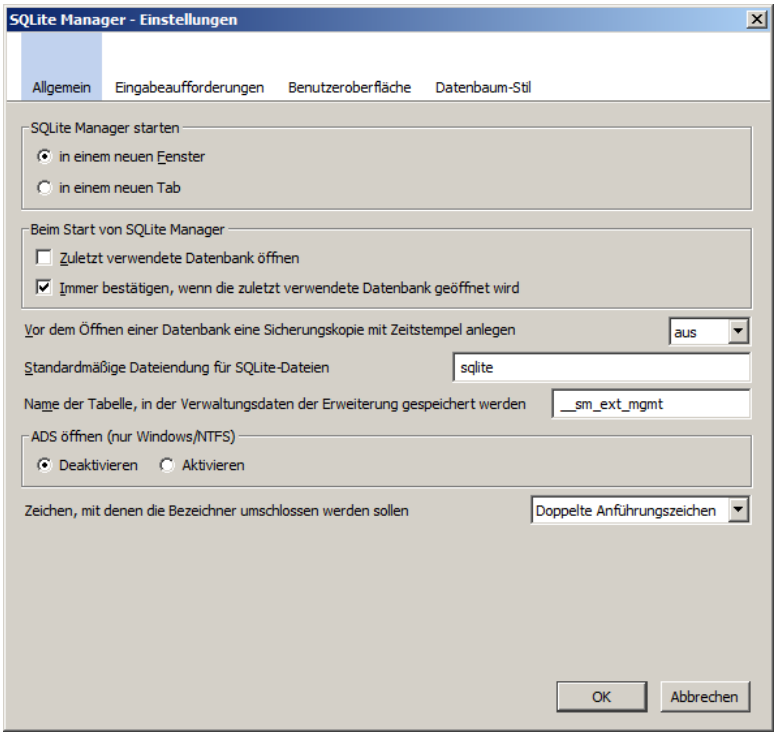
3.1.7.1.2. Arbeiten mit dem SQLite Manager

Die Oberfläche vom SQLite Manager erinnert in vielen Punkten an den SQLite Database Browser (→ 3.1.7.1.1. Arbeiten mit dem SQLite Database Browser). So unterschiedlich sind die beiden Programme auch gar nicht.



Der SQLite-Manager übernimmt nun auch wieder das automatische Speichern nach Eingaben. So erwartet man es ja auch bei Datenbank-Systemen.

Wer will, kann sich bei jedem Start eine Sicherungskopie seiner SQLite-Datei erstellen lassen. Dieses ist eine der Optionen, die im Bereich "Allgemein" eingestellt werden können.



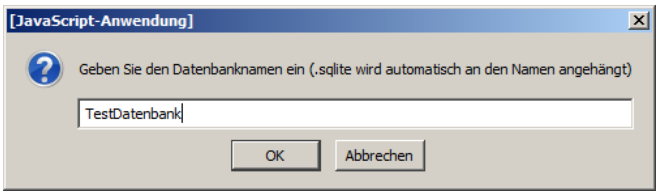
Auch sonst bietet der SQL-Manager einige Einstellungen mehr, als der einfacher gestrickte SQLite-Database-Browser.

Aber viele Einstellungen und viele Bedien-Knöpfe erhöhen das Risiko einer Fehlbedienung. Die Wahl der Bedien-Oberfläche für SQLite sollte also an den Aufgaben erfolgen.
Die erzeugten Datenbank-Dateien können gegenseitig verwendet / geöffnet werden.
Die hohe Kompatibilität der SQL-Daten-Dateien (komplette Datenbanken) ergibt sich aus dem aufgesetzten Anwendungen auf das elementare SQLite-System (s.a. → Fontend's / Benutzeroberflächen für SQLite).

3.1.7.1.2.x. Erstellen einer Beispiel-Datenbank (Kontakte-Institutionen)

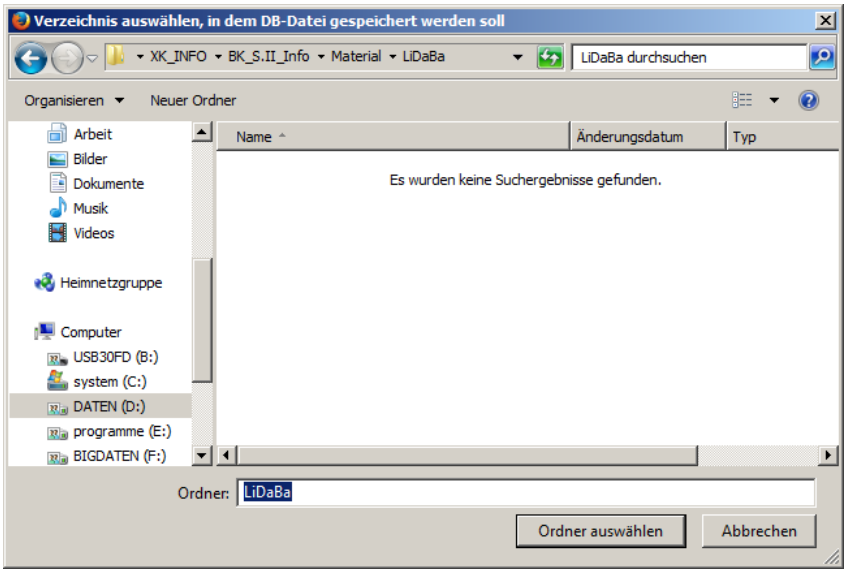
Erstellen einer neuen Datenbank

Namen für Datenbank vergeben

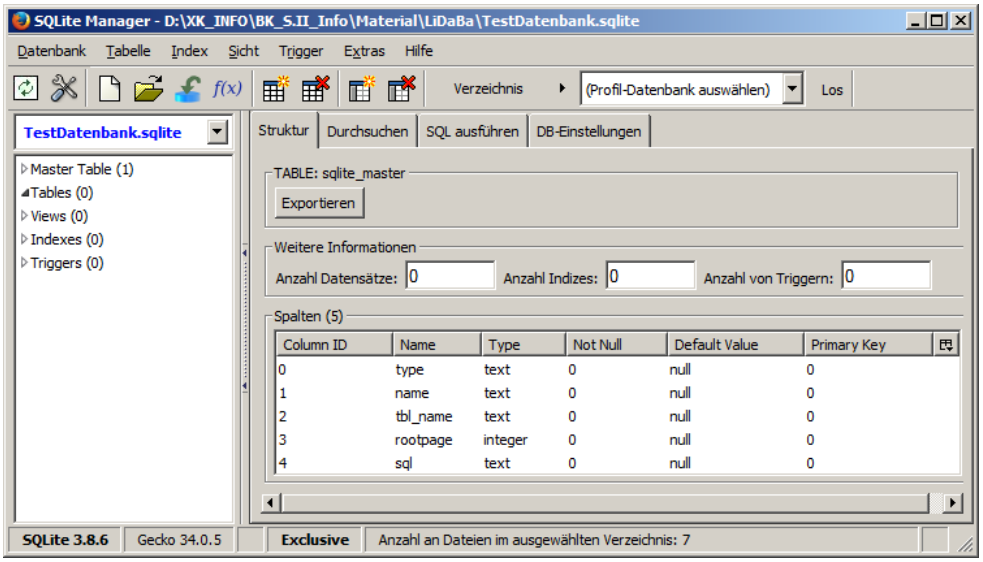


Ordner zum Speichern auswählen

der Name der Datei im gewählten Verzeichnis wurde ja schon vorher bestimmt



leere SQLite-Datenbank

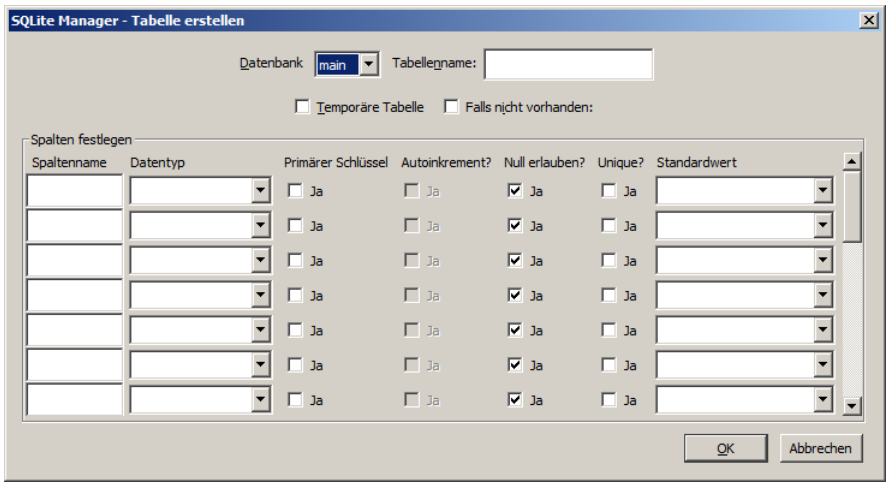


Tabellen

neue Tabelle erstellen

leeres Tabellen-Erstellungs-Raster

sehr übersichtliche und verständliche Möglichkeit die Struktur einer Tabelle zu entwickeln bzw. einzugeben

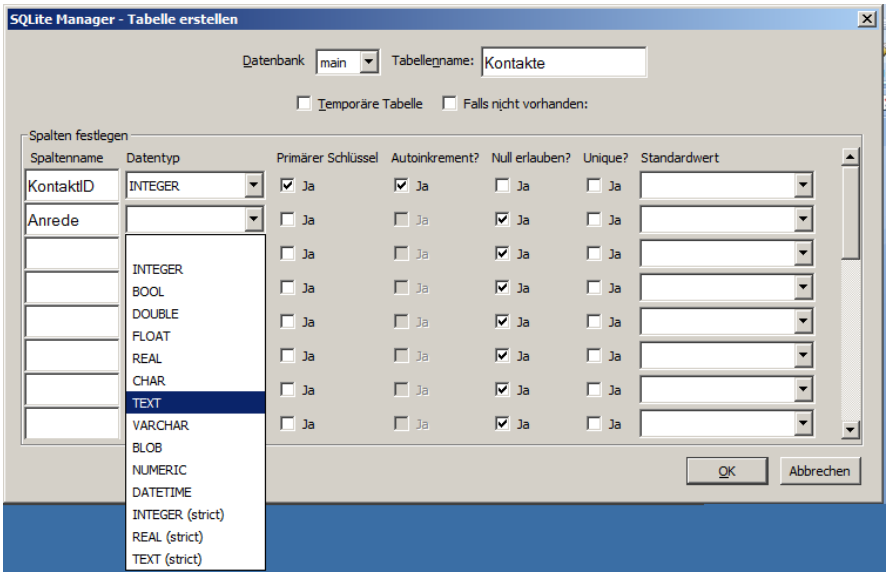


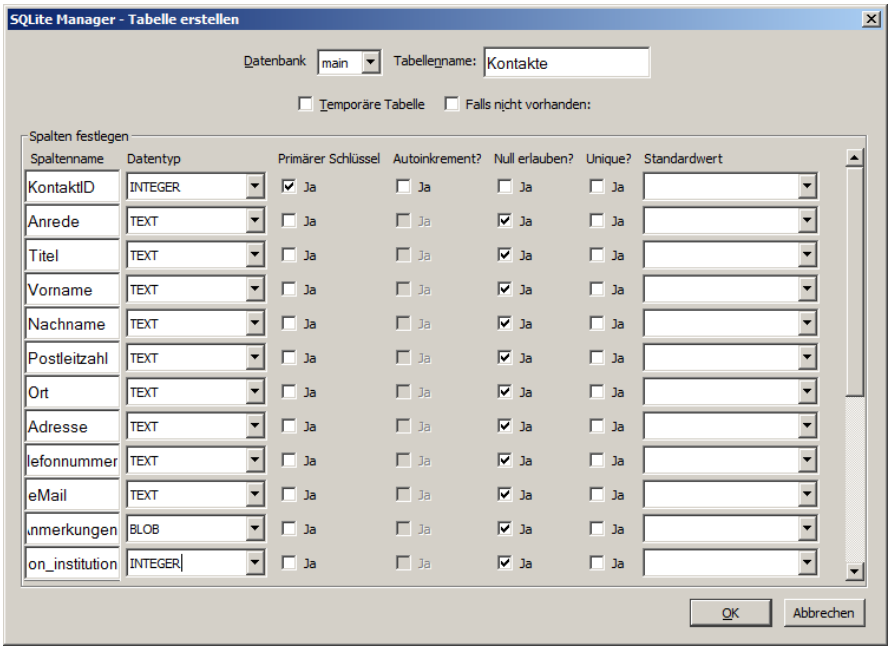
Tabellen-Struktur eingegeben

Primärer Schlüssel normalerwese mit Autoinkrement günstig

in unserem Fall wegen vorgegebener Schlüssel ungeeignet

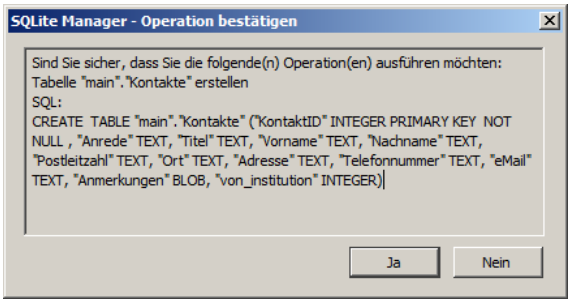
Auswahl der Datentypen für die Felder / Attribute umfangreicher als beim SQLite-Database-Browser





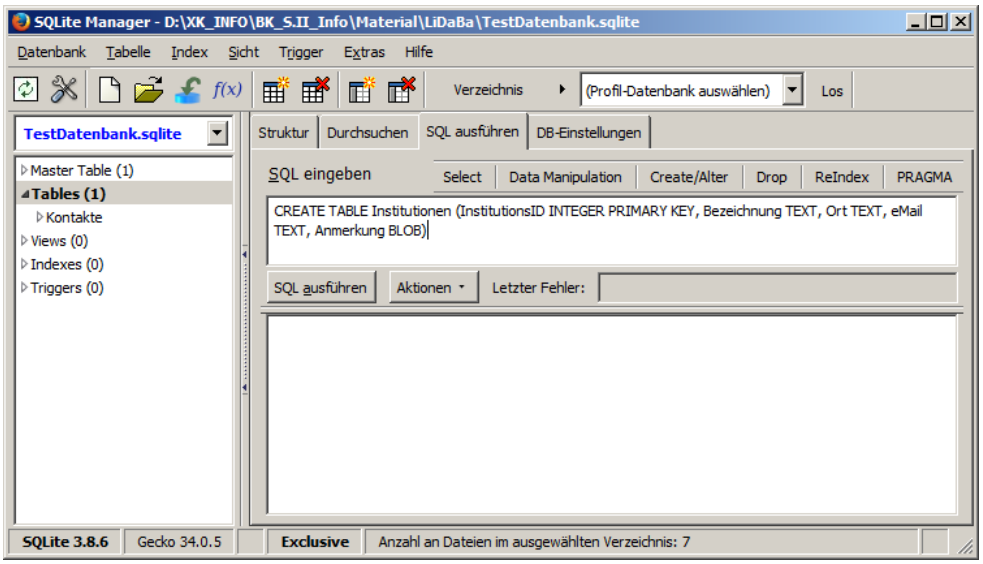
Bestätigung mit Anzeige des verwendeten SQL-Statement's

diesen sollte man sich immer anschauen, um ein immer besseres Gefühl für SQL-Ausdrücke zu bekommen



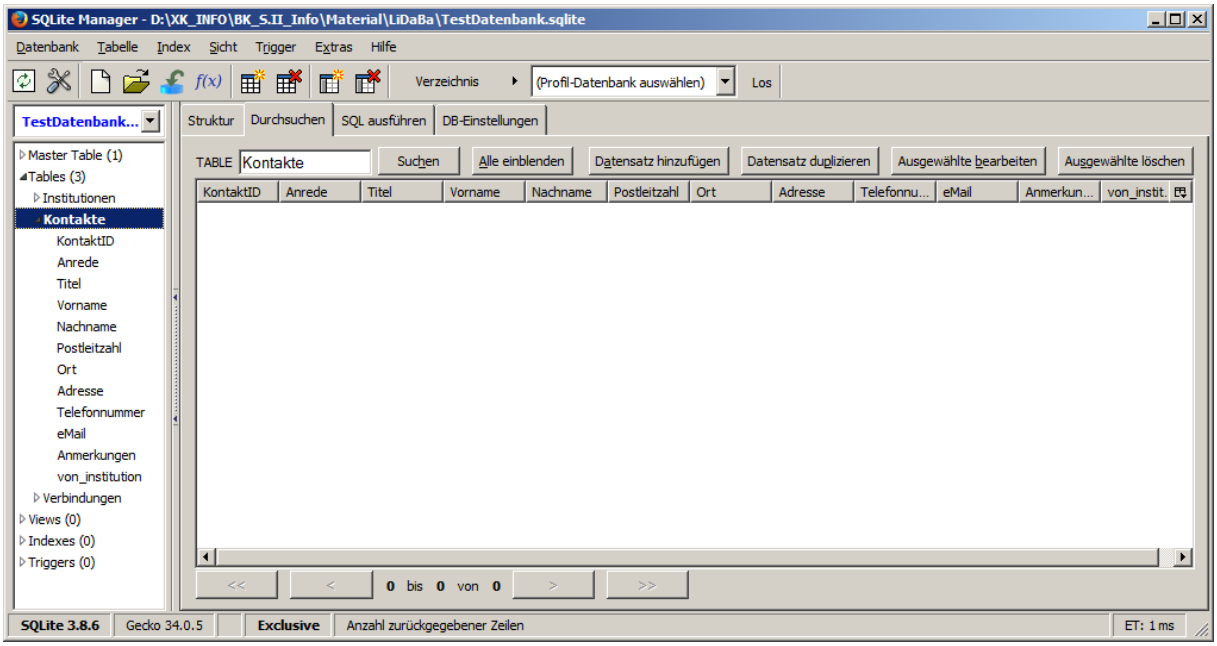
Institutionen über SQL-Quelltext-Eingabe

bei "letzter Fehler" sollte dann: "not an error" stehen

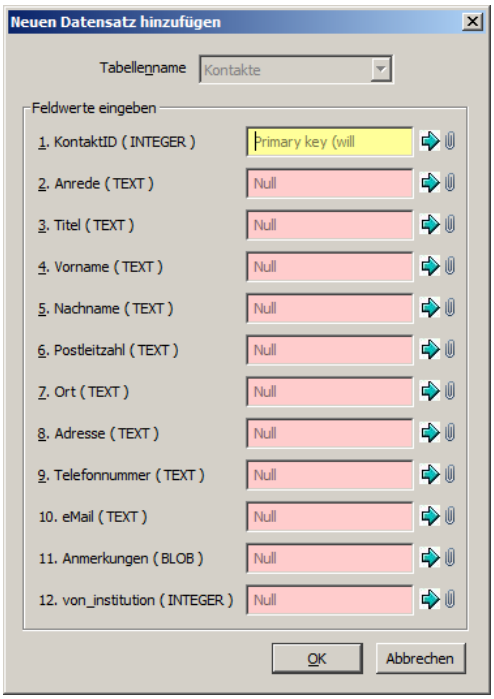


Daten eingeben

Reiter "Durchsuchen"



einen neuen Datensatz / eine neue Zeile mittels "Datensatz hinzufügen"



es folgt ein einfaches Formular mit farblichem Feedback

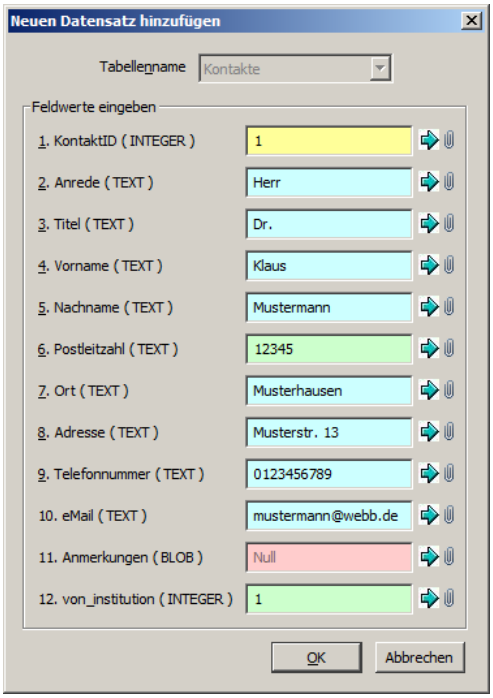
es werden die Felder mit Texten hellblau hinterlegt, die mit nummerischen Werten grünlich

Zellen mit NULL-Werten sind rötlich
auch wenn das so ein bisschen dramatisch aussieht, es ist in den meisten Fällen kein Problem, wenn die Felder frei bleiben

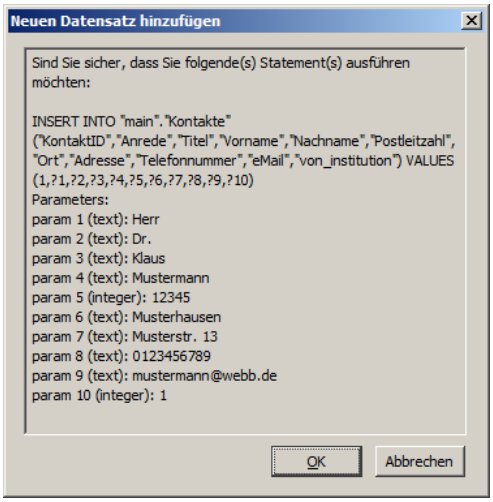
Was natürlich belegt werden muss (- zumindestens in unserem Fall -) ist der primäre Schlüssel bei "KontaktID", als Hinweis dient die gelbliche Feldfarbe

der fertig eingebene Datensatz kann hier schon gut kontrolliert werden

Trotzdem erfolgt nach dem "OK" eine nochmalige Rückfrage.



Übersetzung der im Formular gemachten Eingaben in eine SQL-Anweisung. Diese kann zwar nicht direkt geändert werden, aber bei einem "Abbrechen" gelangt man zum Formular zurück. Mit einem "OK" der resultierenden SQL-Anweisung wird die Datensatz-Eingabe bestätigt.



nach Bestätigung erfolgt scheinbar trotzdem Rückkehr zur (alten) Datensatz-Eingabe

es können jetzt aber die Daten des neuen (!) Datensatzes unter Verwendung des vorherrigen zusammengestellt werden
Achtung! Daten wirklich gründlich überprüfen, es schleichen sich durch die Vorbelegung schnell Übernahme-Fehler ein!

Hier noch mal die komplette Daten-Tabelle aus einer BASE-Datenbank:

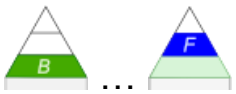
Kontakte - TestDatenbank - LibreOffice Base: Tabellendatenansicht

Datei Bearbeiten Ansicht Einfügen Extras Fenster Hilfe

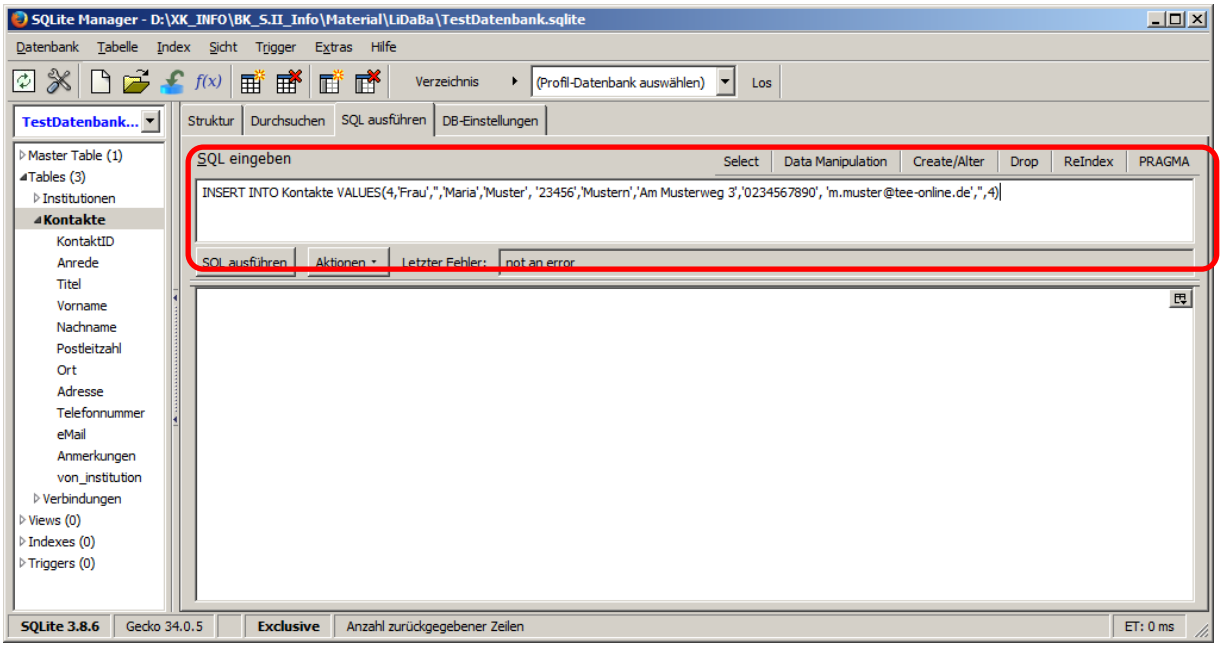
KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerkung
1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	0123456789	mustermann@webb.de	
2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	0123456789	musterfrau@webb.de	
4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	0234567890	m.muster@tee-online.de	
5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	04567890901	Chr.Bauer@geemx.de	
6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	09998765	Mueller@tee-online.de	
7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	0777711	Ta.Zan@webb.de	
8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	0543861	Prof.H.Ziesow@tee-online.d	
12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	0456783067		
37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	06543210	post@h-krueger.de	
3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	088088088	H.Fehler@webb.de	

Datensatz 11 von 11

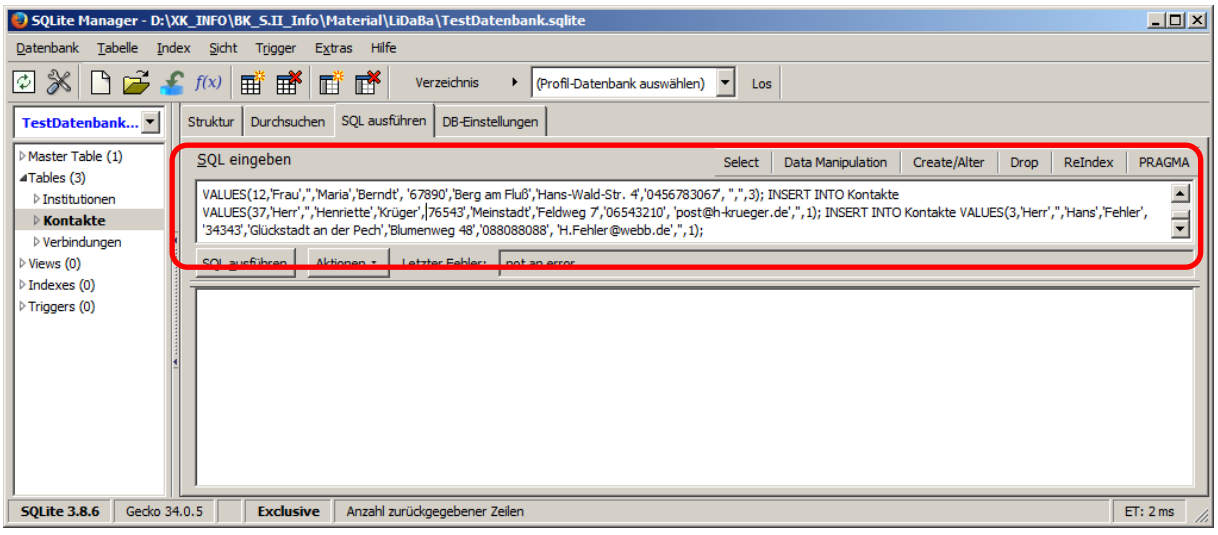
Arbeiten mit SQL



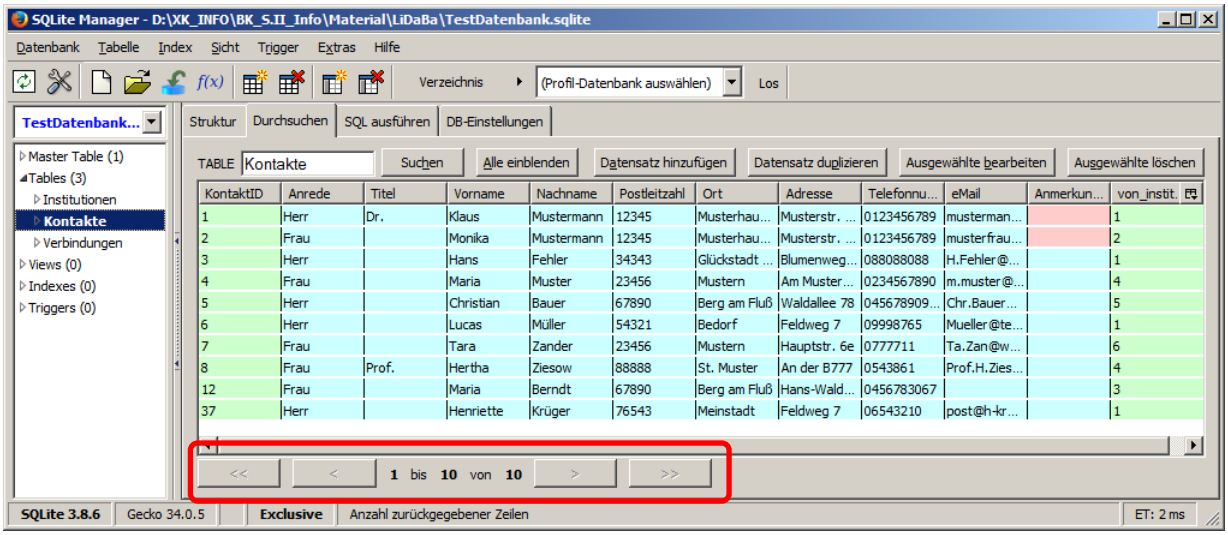
Obwohl im SQLite-Manager die Formular-Eingabe schon recht komfortabel ist, geht es mit einer direkten Eingabe unter "SQL ausführen" meist noch schneller



Das funktioniert auch für mehrere SQL-Anweisungen hintereinander. Die eingegebenen Anweisungen müssen dann Semikolon-getrennt notiert werden.

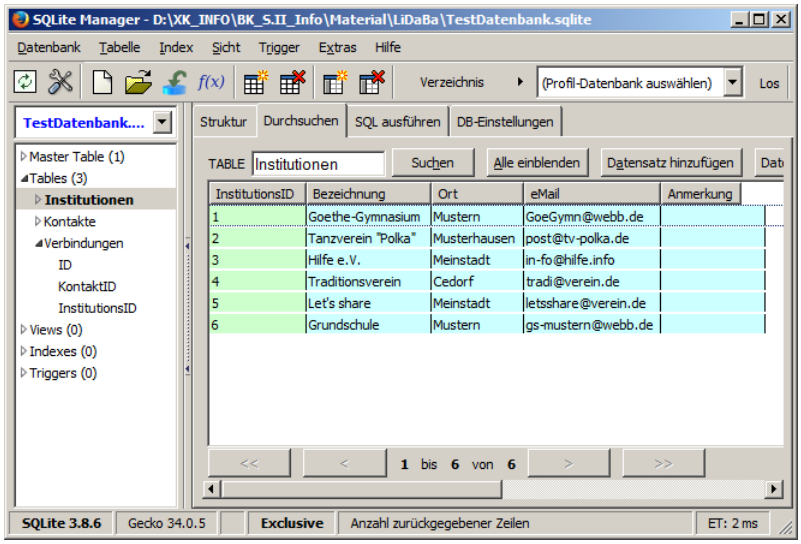


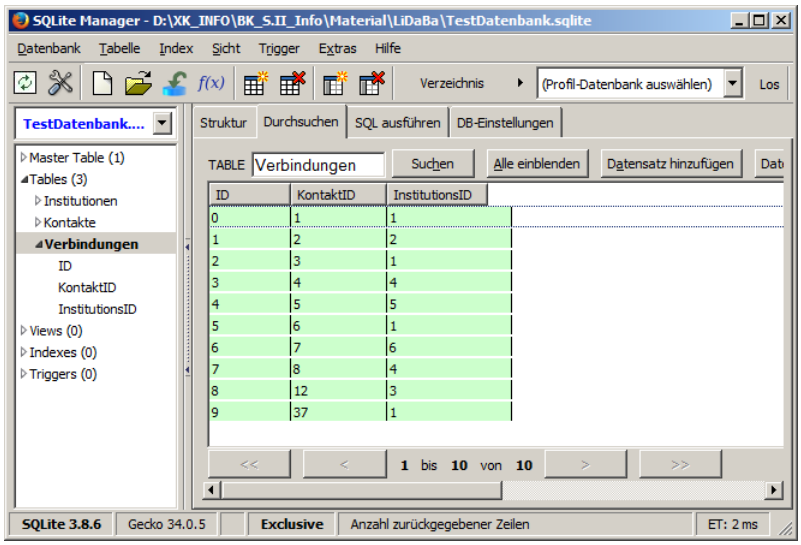
Am Schluss können wir uns die Tabelle mit den Daten nochmals ansehen. Der Reiter heißt etwas ungewöhnlich "Durchsuchen". Er bietet neben der reinen Tabelle auch einen kleinen Datensatz- / Seiten-Navigator unter der Tabelle.



Datensatz-Navigator unter der Datentabelle

Eingabe der Daten der anderen beiden Tabellen entweder über das Formular oder ein SQL-Statement





Aufgaben:

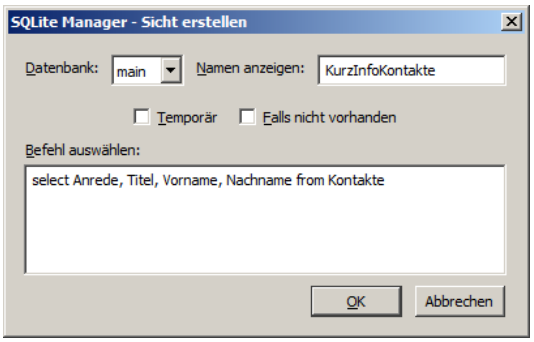
1. Vervollständigen Sie die Daten der "Verbindungen"-Tabelle!

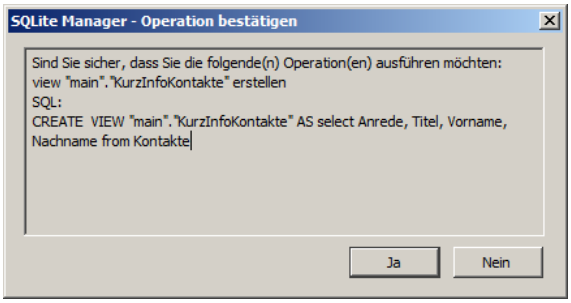
3.1.7.1.2.x. Arbeiten mit der Beispiel-Datenbank (Kontakte-Institutionen)

Abfragen

Neben den klasischen Tabellen und Indizies bietet der SQLite-Manager auch die Möglichkeit Abfragen zu erstellen. Sie heißen hier eingedeutscht "Sichten". In anderen Programmen oder Beschreibungen wird auch von "View"s gesprochen, was ja der wörtlichen Übersetzung entspricht.

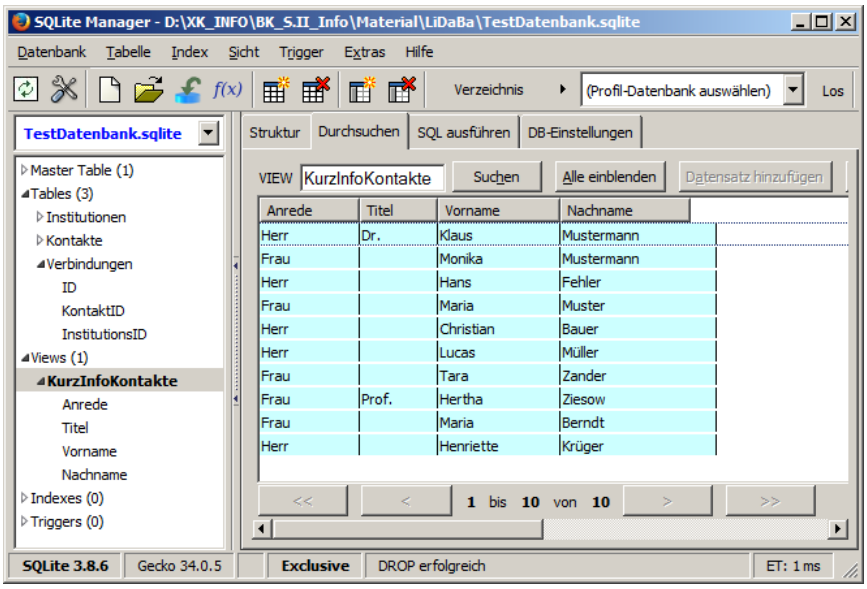
Eingeben als SQL-Anweisung





Bestätigung der übersetzten / korrigierten / angepassten SQL-Anweisung

Ergebnis taucht im Datenbank-Strukturbaum (links) unter "Views" auf und kann – wie üblich bei "Durchsuchen" angezeigt werden



Links:
Q: http://sqlite-manager.googlecode.com

3.1.7.1.2.x. Erstellen und Nutzen von Indizies

für SQLite Studio beschrieben → 3.1.7.1.3.4. Erstellen von Indizes

3.1.7.1.3. Arbeiten mit dem SQLiteStudio

Q: sqlitestudio.pl

SQLiteStudio ist ein Open-Source-Projekt (→ http://sqlitestudio.pl). Fertige ausführbare Dateien / Programme gibt es für die gängigen Betriebssysteme:

- Windows (9x → 7)
- Linux
- MacOS X
- UNIX

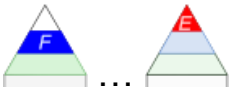
Das Studio ist eine graphische Benutzer-Oberfläche für das Datenbank-System SQLite.
Besonders für Nutzer von restiktiven Betriebssystem-Umgebungen ist interessant, dass SQLiteStudio eine portableApp ist. Das bedeutet, man muss das Programm nicht installieren. Es kann in der eingeschränkten Nutzer-Umgebung ausgeführt werden. Deshalb läuft es eben auch so gut vom IoStick oder auch als portable App im gleichnamigen Menü-System.
Wie bei Open-Source-Projekten üblich, ist auch der gesamte Quell-Code zugänglich. Man kann das Programm also auch für ein anderes Betriebssystem compilieren, wenn ein passender Compiler verfügbar ist.
Wem das Programm nicht genug leistet und der Programmierung mächtig ist, kann sich gerne an der Weiterentwicklung des Programm's beteiligen. Auch kleine Geldspenden helfen den Entwicklern weiter.
Das SQLite Studio erinnert schon sehr stark an das Datenbank-Händling in den verschiedenen Betriebssystemen. Wer schon mal eine ODBC-Schnittstelle einrichten musste, weiss, was ich hier meine.
Im folgenden Kapitel beziehen wir uns auf die Version 3.1.1. (vom IoStick (2017)).

3.1.7.1.3.0. SQLiteStudio "installieren"

Wer den IoStick (Quelle: → https://tinohempel.de/info/info/IoStick/index.html) benutzt, hat schon eine vorbereite Version dabei und kann sofort mit dem Arbeiten beginnen (→ 3.1.6.1.3.1. SQLiteStudio starten).

Ansonsten lädt man sich die aktuelle Version (→ https://sqlitestudio.pl/index.rvt?act=download) – bzw. wenn neue Versionen ganz anders aussehen sollte und man weiter bei diesem Script bleiben will, dann die Version 3.1.1. – aus dem ZIP-Archiv (→ https://sqlitestudio.pl/files/sqlitestudio3/complete/zip/) herunter.
Die ZIP-Datei wird in einem beliebigen Ordner entpackt. In dem neu entstandenen Ordner "SQLiteStudio" findet man die Datei SQLiteStudio.exe. Wer die Endung EXE nicht sieht, kann die ausführbare Datei aber am Symbol (siehe oben) erkennen. Es empfiehlt sich eine Verknüpfung auf dem Desktop anzulegen, damit man sich nicht jedesmal bis zur EXE durchklicken muss.
Eine schöne und komfortable Möglichkeit ist die Einbindung von SQLiteStudio in ein vorhandenes PortableApps-System (Download: →https://portableapps.com/). Die gedownloadete ZIP-Datei von SQLiteStudio wird dann in den PortableApps–Ordner entpackt. Weiter sind keine Arbeiten notwendig. Beim nächsten Start des PortableApps-System steht die App im Menü-System unter "Sonstige" bereit.

SQLite Studio auf einem Raspberry Pi



Das SQLiteStudio kann auch auf einem Raspberry Pi verfügbar gemacht werden. Im Raspian – einer gängigen Linux-Distrubution für den Rasp Pi – ist das Studio aber nicht enthalten. Es kann aber nachträglich installiert werden. Dazu benutzt man die nachfolgenden – teilweise auch kommentierten – Befehle in einer Konsole. Jede Zeile ist dabei ein Konsolen-Befehl, der i.A. auch immer irgendwelche Kontrollanzeigen generiert. Einige Befehle können auch langerfristige Aktionen / Downloads auslösen. Man sollte also etwas mehr Zeit einplanen.
Die erste – jetzt folgende – Konsolen-Kommando-Sequenz kann entfallen, wenn SQLite schon fuktioniert. Es schadet aber nicht die Kommando's nochmals einzugeben.

`$ sudo apt-get update`	
`$ sudo apt-get dist-upgrade`	
`$ sudo apt-get install build-essential tcl`	kann entfallen
`$ sudo apt-get install libreadline-dev ⮐ libncurses5-dev`	
`$ sudo apt-get install sqlite libsqlite3-dev`	kann entfallen, falls SQLite schon läuft

⮐ … direktes Weiterschreiben in der Kommando-Zeile

Die nächsten Kommando's laden die aktuelle Version von SQLite Studio herunter und installieren dann das Programm. Die Versions-Nummern müssen jeweils angepasst werden, da das Projekt ständig weiterentwickelt wird.

`$ sudo apt-get install qt5-default qtscript5-⮐ dev qttools5-dev libqt5svg5-dev`	
`$ mkdir -p ~/projects/sqlitestudio/`	
`$ cd ~/projects/sqlitestudio/`	
`$ wget http://sqlitestudio.pl/files/sqlite⮐ studio3/complete/tar/sqlitestudio-3.1.1.tar.gz`	Nummern ev. durch aktuelle / andere Version ersetzen
`$ tar xf sqlitestudio-3.1.1.tar.gz`	bei Problemen beim Übersetzen später, lieber auf die letzte verfügbare 3.0.x Version zurückgreifen
`$ mkdir -p output/build/`	
`$ cd output/build/`	
`$ qmake ../../SQLiteStudio`	
`$ make -j4`	-j4 für Rasp Pi3, sonst ohne
`$ make -j4 install`	

⮐ … direktes Weiterschreiben in der Kommando-Zeile

Starten von SQLiteStudio unter Raspian /

Nun kann eine neue Datenbank angelegt werden (→ 3.1.6.1.3.2. eine Datenbank erstellen).

Links / Quellen:
http://www.raspberry-pi-geek.de/Magazin/2016/05/Arbeiten-mit-SQLiteStudio

SQLite Studio auf einem Mac-Rechner

SQLiteStudio gibt es auch als Mac-Version. Den Download findet man unter → https://www.sqlite.org/download.html. Es wird eine ZIP-Datei bereitgestellt, die nach dem entpacken für die Mac-typische Installation genutzt werden. Das ist sicher auch die einfachste Variante.
Will man mit dem IoStick arbeiten tut sich ein großes Problem auf. Alle Programme des IoStick, wie auch das Menü-System sind nur für Windows-Rechner gedacht.
Es gibt aber verschiedene Möglichkeiten aus der Patsche zu kommen.
Man benötigt ein Installations-Medium für Windows und optimalerweise eine gültige Lizenz. Ansonste kann man nur die 6 Wochen Testzeitraum nutzen. Für einmal SQL sollte das reichen.

Möglichkeit 1: Nutzung einer virtuellen Maschine (virtueller Rechner)

Vorteile:
- leicht wieder zu entfernen (Löschen der virtellen Maschinen, Löschen des Managers für die virtuellen Maschinen → fertig)
- einfache Übertragung von Daten zwischen Mac und Win
- Naht-loses Umschalten zwischen dem Arbeiten in Mac und Win
-

Nachteile:
- Installation eines zusätzlichen Software-Paketes notwendig (Manager für die virtuellen Maschinen)
- großer Festplatten-Bedarf für die virtuellen Maschinen
- Leistungs-Verlust (Rechner wird langsamer; virtuelle Maschine ist langsamer und Leistungs-begrenzt)
-

Möglichkeit 2: Nutzung einer Parallel-Installation (Multiboot)

Vorteile:
-

Nachteile:
-

Möglichkeit 3: Installation / Nutzung von VirtualBox

Möglichkeit 4: Installation / Nutzung von Parallels Desktop

Vorteile:
-

Nachteile:
-

Möglichkeit 5: Installation / Nutzung von Vmware Fusion

Vorteile:

-

Nachteile:

-

Möglichkeit 6: Nutzung von Windows parallel zum MacOS (Apple Bootcamp)

Vorteile:

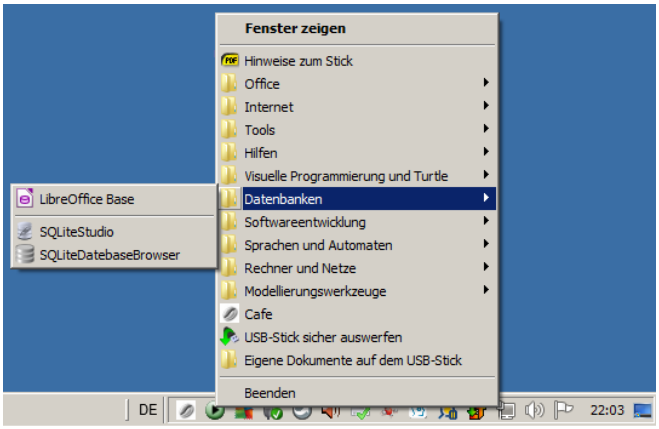
-

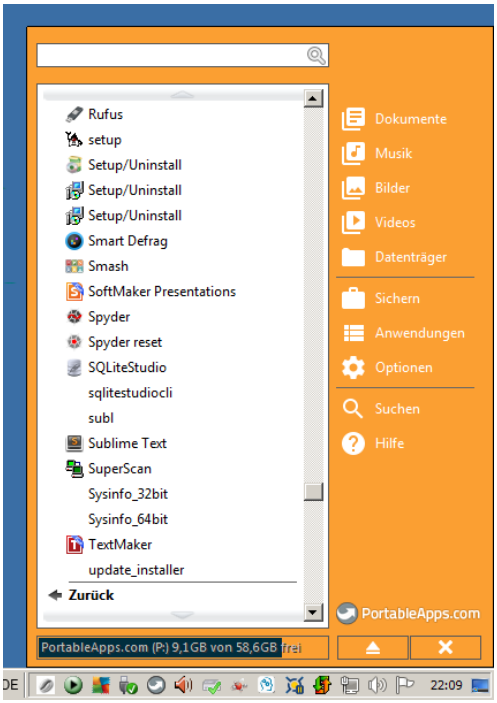
Nachteile:

-

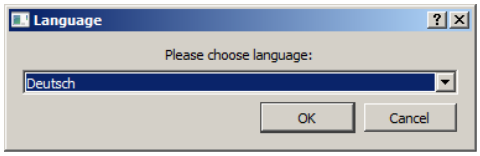
3.1.7.1.3.1. SQLiteStudio starten

Die IoStick-Benutzer finden einen Menü-Eintrag von SQLiteStudio" unter "Datenbanken" (s.a. Abb. rechts).
Wer die ZIP-Datei von SQLiteStudio in einen portApps-Ordner eines funktionierenden PortableApps-System entpackt hat, findet SQLiteStudio im Prgrammordner "Sonstige" (s.a. Abb. unten).

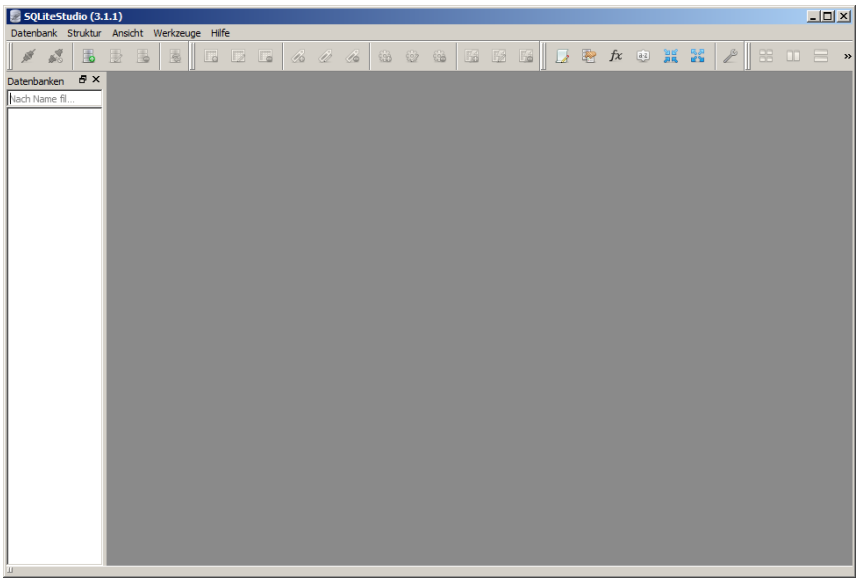




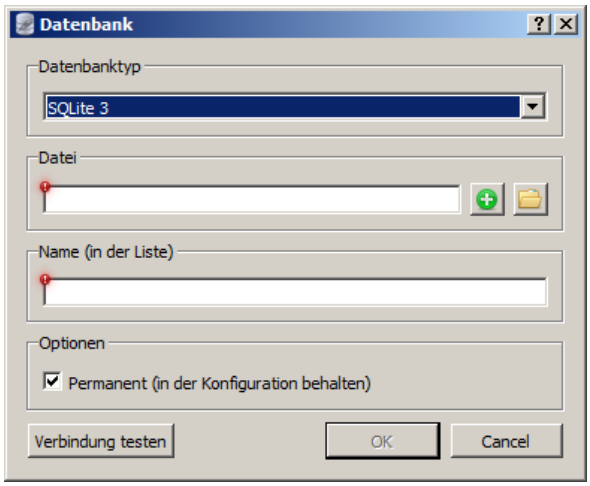
Bei der Erstbenutzung werden wir nach der gewünschte Sprache gefragt. Obwohl nicht alle Programm-Elemente übersetzt sind, hilft eine Sprachauswahl auf "Deutsch" bei der ersten Arbeit mit dem Programm. Später kann man dann bei "American English" bleiben.



Das leere Programm-Fenster zeigt uns den derzeit leeren Zustand unseres Datenbank-Management-System's an.



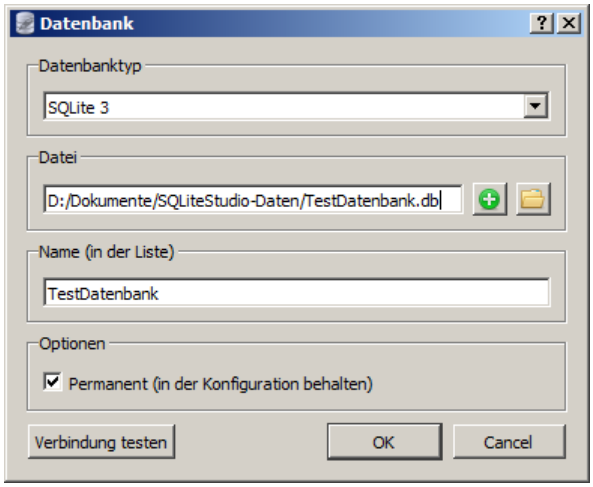
3.1.7.1.3.2. eine Datenbank erstellen



"Datenbank" "Datenbank hinzufügen"
oder [Strg] [o]

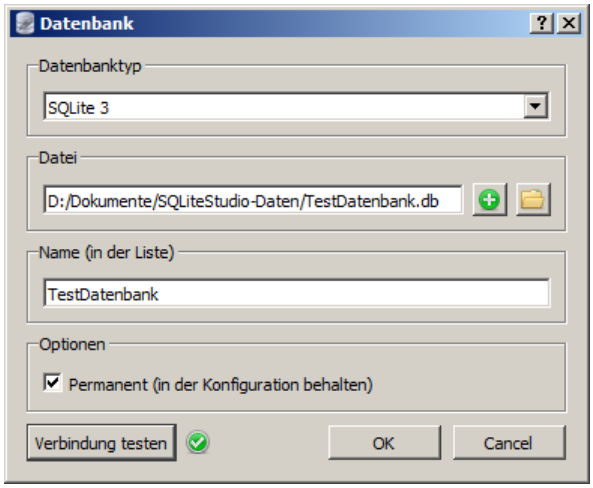
die kleinen roten Ausrufezeichen markieren die notwendigen Minimal-Eingaben

Die Datei, in der die gesamte Datenbank gespeichert werden soll, legen wir über die Schaltfläche mit dem "grünen Plus"-Zeichen fest
alternativ können wir eine bestehende Date über das "Ordner"-Symbol öffnen



aus dem Dateinamen leitet SQLiteStudio einen Datenbank-Namen ab, den man aber ändern kann

die Option "Permanent" bestimmt, ob unsere Datenbank dauerhaft in der Datenbank-Liste links aufgeführt werden soll
das ist für das Weiterarbeiten beim nächsten Mal sehr praktisch



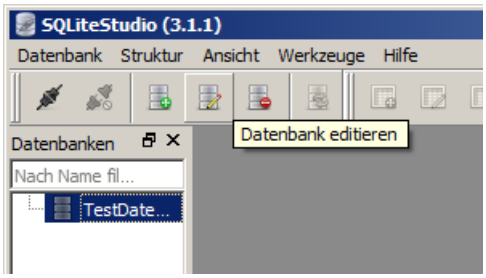
zu guter Letzt testen wir, ob die Verbindung zum internen SQL-Server steht

mit "Verbindung testen" geht das schon vor der abschließenden Bestätigung über die Anlage der Datenbank

das grüne Häkchen bestätigt die Funktionsfähigkeit

Sachlich kann eine SQL-Datenbank auch in anderen Anwender-Programmen genutzt werden. Das entspräche dann schon mehr einer Server-Client-Datenbank.

nachträgliches Ändern der Datenbank-Anbindung ist möglich

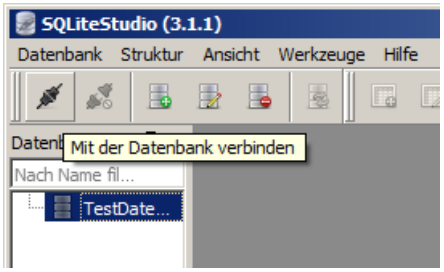


über Symbolleiste oder Menü "Datenbank" "Datenbank editieren"

ebenso das Löschen einer Datenbank
z.B. über "Datenbank" Datenbank entfernen"

Mit der Datenbank verbinden

zum Arbeiten mit der Datenbank muss man sich mit der Datenbank verbinden



wir arbeiten sozusagen mit unserem Programm als Client und brauchen eine Verbindung zum eigentlichen (hier Programm-internen) SQL-Server

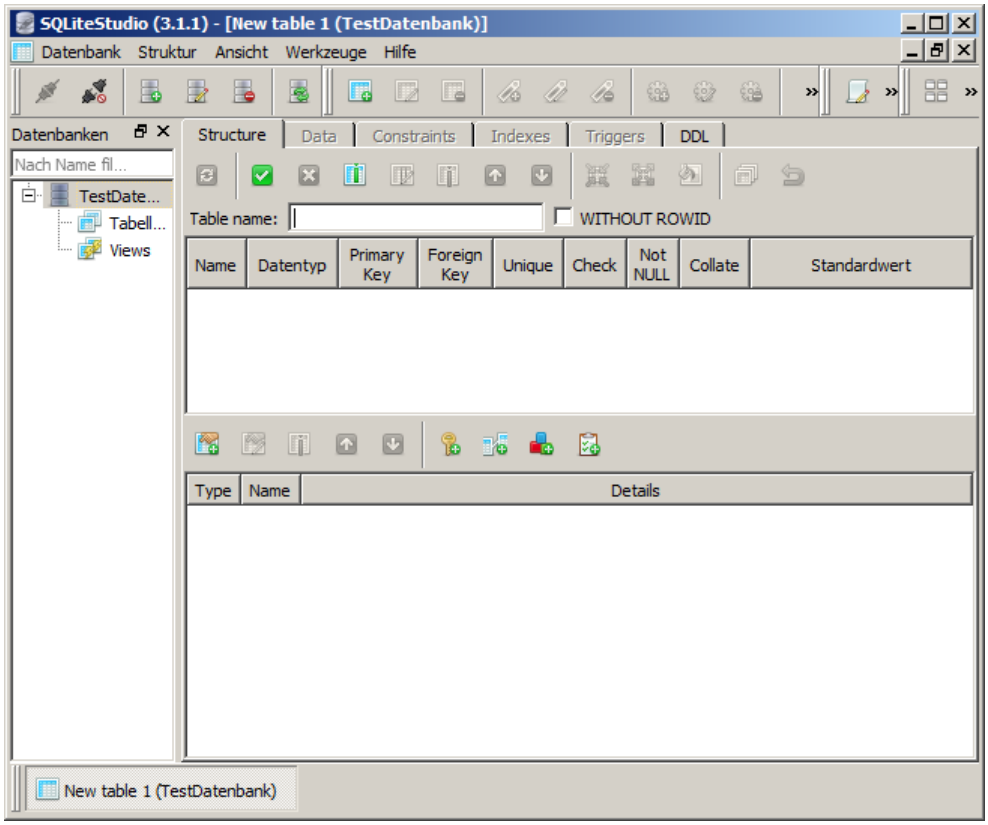
Stecker-Button in der Symbolleiste oder "Datenbank" "Mit der Datenbank verbinden"

Hat die Verbindung geklappt, dann sehen wir unter dem Datenbank-Eintrag in derListe die Struktur unserer Datenbank

es gibt zwei Bereiche, einmal die Tabelle und zum Zweiten die Views (Abfragen, Sichten)

die Tabellen enthalten die eigentlichen Daten

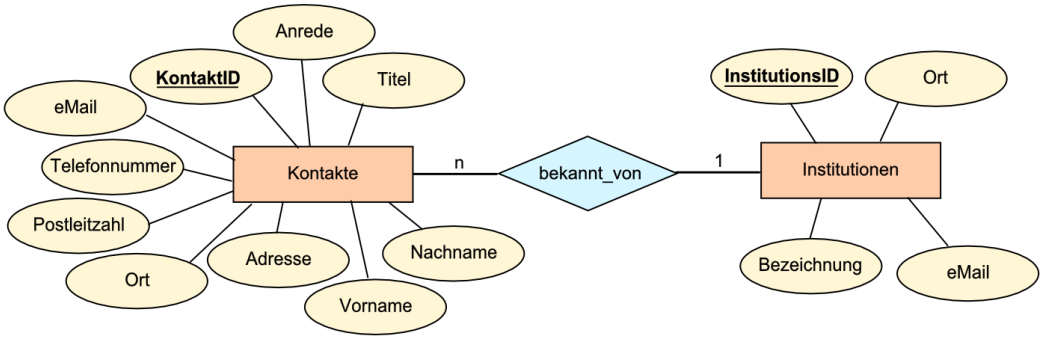
mit den Views bekommen wir bestimmte Sichten auf unsere Datenbank bzw. auf Teile / Ausschnitte davon.



Aufgaben:

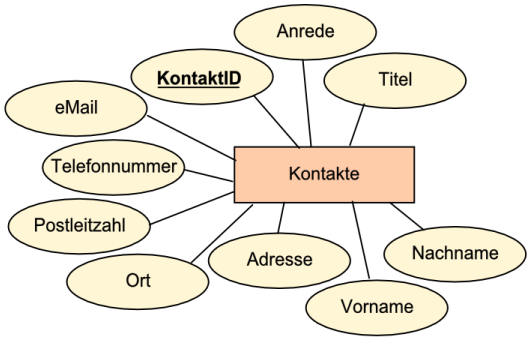
1. *Erstellen Sie sich im SQLiteStudio eine Datenbank "TestDatenbank", speichern die zugehörige Datei in Ihrem Arbeits- / Home-Ordner!*
2. *Verbinden Sie sich dann mit der Datenbank!*

Realisierung der gleichen Datenbank, wie bei den anderen Systemen. Das zu realiserende Entity-Relationship-Modell sieht so aus:



die erste zu erstellende Tabelle ist "Kontakte"

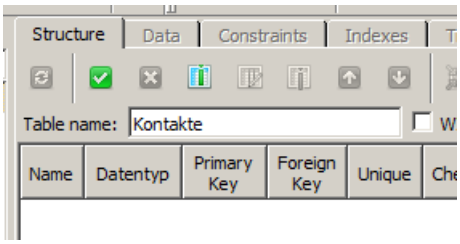
natürlich kann man auch erst mit der untergeordneten Tabelle "Institutionen" beginnen



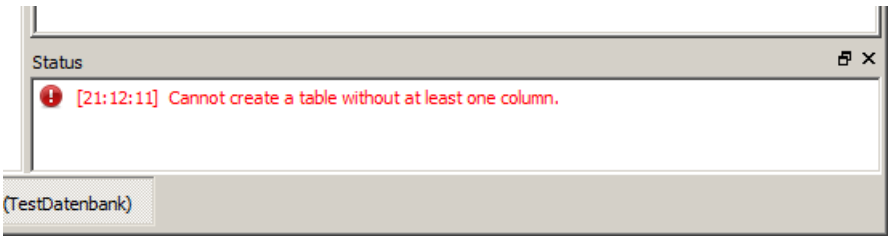
Unter dem Reiter "Structure" erstellen wir jetzt Spalte für Spalte unsere Tabellen-Struktur – also die Spalten-Überschriften (Attribute) und legen auch deren Datentyp fest.

Zuerst bestimmen wir den Namen der Tabelle. Laut derm ERD soll die Tabelle "Kontakte" heißen.

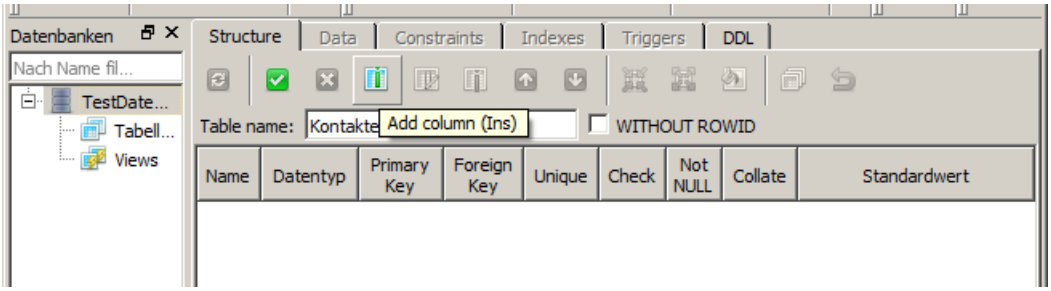
Erst, wenn alle Angaben zur Tabelle eingetragen sind, bestätigen wir die Struktur mit dem "Grünen Häkchen" ("Commit structure changes")



Geschieht die Bestätigung verfrüht, dann bekommt man i.A. eine Fehlermeldung im Fensterchen "Status".



Zum Anlegen einer Spalte wählt man das Symbol mit der Tabelle und der grünlichen Spalte. Der Hilfe-Text (in engl.) "Add column (Ins)" ist da sicher verständlich.



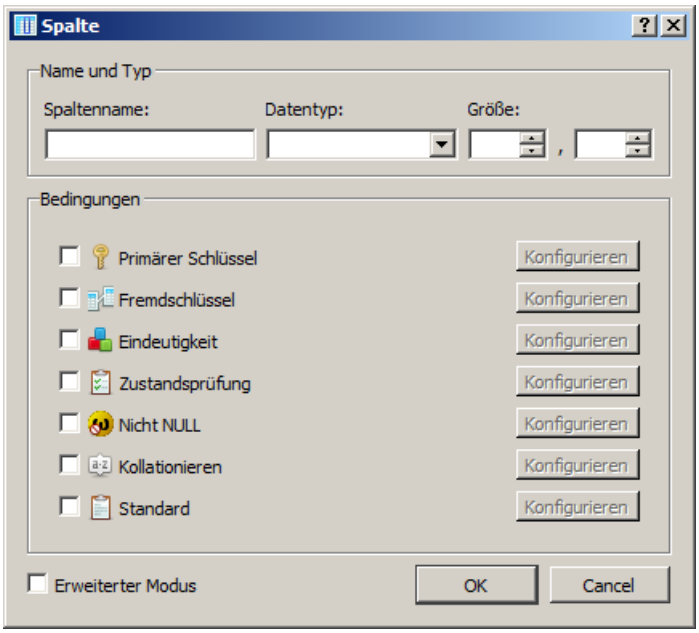
Für jede Spalte sind nun diverse Angaben zu machen. Dabei sind der Spaltenname und der Dateityp zwingend erforderlich.

bei anderen Angaben richtet sich die Angabe-Pflicht mehr nach dem Modell unserer Datenbank

ein planmäßiges, genaues und zielgerichtetes Arbeiten ist hier aber notwendig

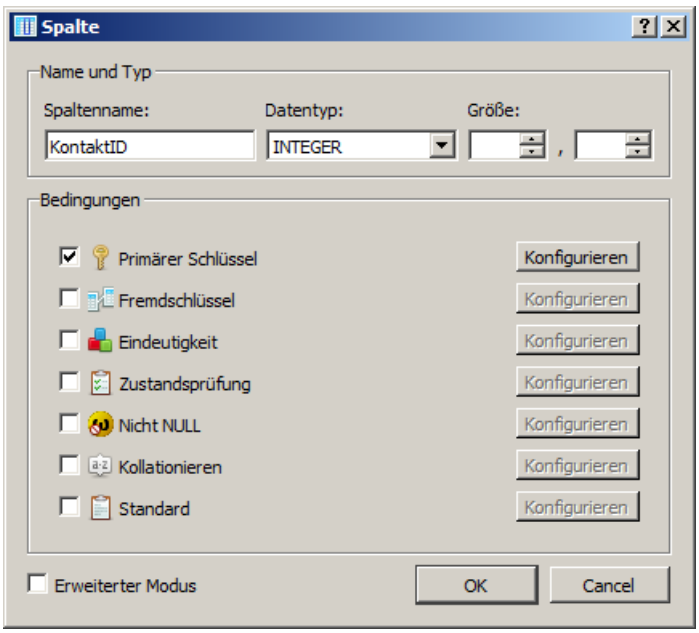
zwar sind einzelne Optionen auch nachträglich veränderlich, aber so etwas ist in Datenbanken allgemein nicht zu empfehlen

alle Bedingungen / Details lassen sich "Konfigurieren", d.h. hier sind zusätzlichen Einstellungen / Optionen möglich

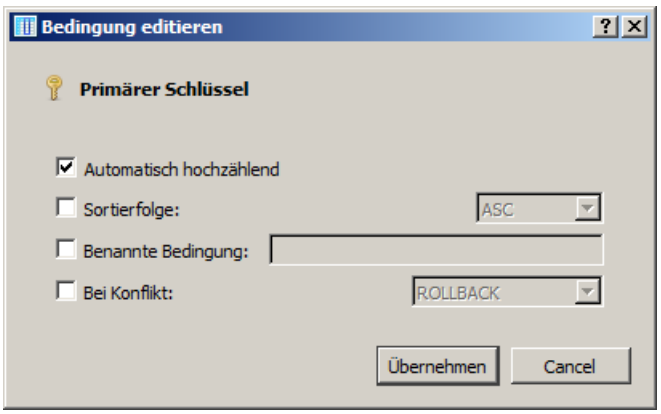


Bedingungen / Optionen

• **Primärer Schlüssel**	(Primary Key) Spalte enthält den Primär-Schlüssel dieser Tabelle
• **Fremdschlüssel**	(Foreign Key) Spalte enthält einen Fremd-Schlüsse
• **Eindeutigkeit**	(Unique) Werte in der Spalte müssen eindeutig sein und dürfen sich somit nicht wiederholen
• **Zustandsprüfung**	(Check condition) Festlegung von Prüf-Bedingungen für die Eingaben
• **NichtNULL**	(Not NULL)
• **Kollationieren**	(Collate)
• **Standard**	(Default) voreingestellter Belegungs-Wert



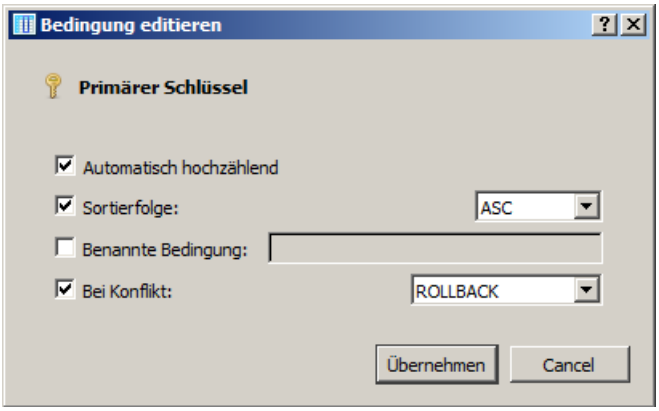
Konfiguration des primären Schlüssel's



Konfigurationen / Optionen für den Primär-Schlüssel

• **Automatisch hochzählend**	(Autoincrement) SQLite übernimmt das Erstellen neuer Primärschlüssel-Werte
• **Sortierfolge**	(Sort order) legt fest, wo der neue Datensatz eingeordnet werden soll
• **Benannte Bedingung**	(Named contraint)
• **Bei Konflikt**	(On conflict) Konflikt-Behandlung; Was soll Widersprüchen, fehlerhaften oder Grenz-überschreitenden Werten passieren?

minimal sollte man hier "Automatisch hochzählend" setzen, dann braucht man sich um den Primärschlüssel nicht mehr weiter kümmern.
recht praktisch ist auch das Setzen der Optionen "Sortierfolge:" und "Bei Konflikt:". Ein ROLLBACK verhindert die Aufnahme nicht regulärer Datensätze in die Tabelle



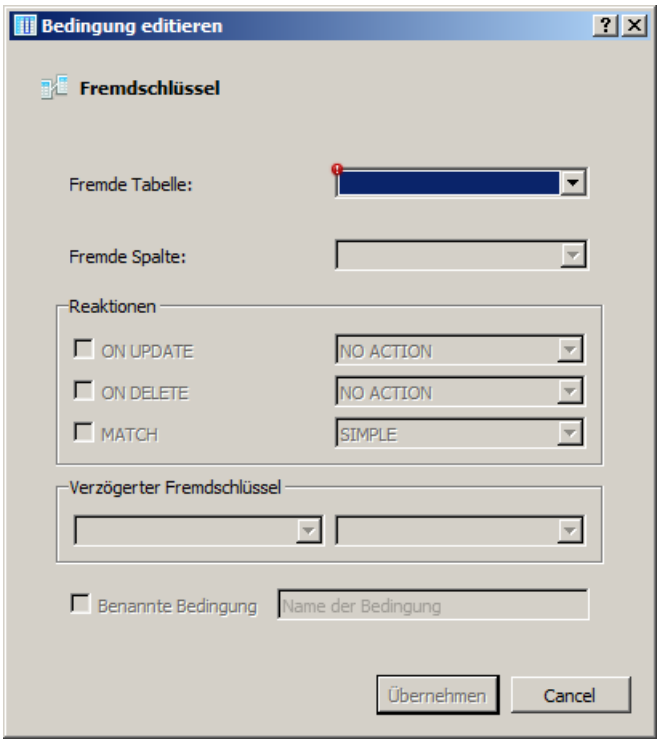
Hinweis: Für unsere Tabelle "Kontakte" mit vorgegebenen Schlüsselwerten geht das nicht!

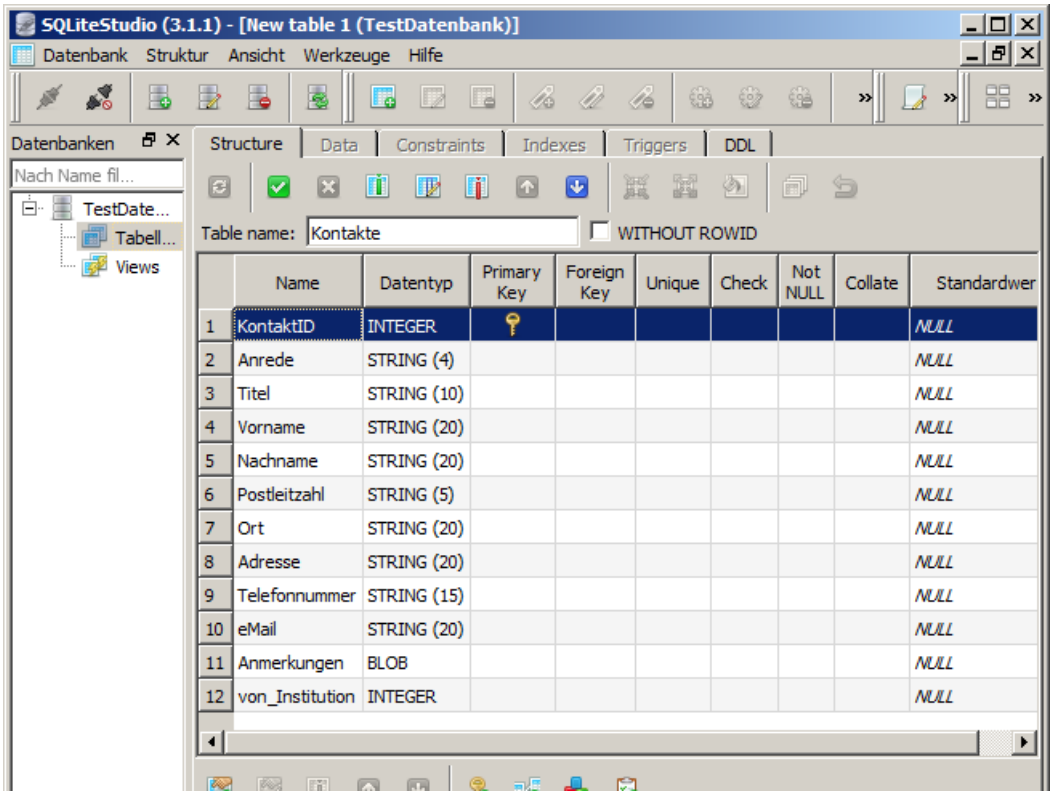
die letzte Spalte soll einen Fremdschlüssel

das bekannte rote Erinnerungs-Ausrufezeichen verweist uns auf die notwendige Konfiguration des Fremdschlüssels

auch hier zeigt das Ausrufezeichen auf eine notwendige Angabe. Und da haben wir ein Problem: Wir haben noch gar keine Tabelle zum Verweisen.
hier müssen wir die Konfiguration des Fremdschlüssels also abbrechen ("Cancel")
die Verbindung dieser Tabelle mit der noch zu erstellenden Tabelle "Institutionen" müssen wir später noch nachholen

Um also Tabellen insgesamt mit möglichst wenig Verweis-Problemen zu erzeugen, ist es sinnvoll die gesamte Tabellen-Struktur rückwärts anzulegen.
In unserem Beispiel also zuerst die Tabelle "Institution" und dann die eben besprochene Tabelle.

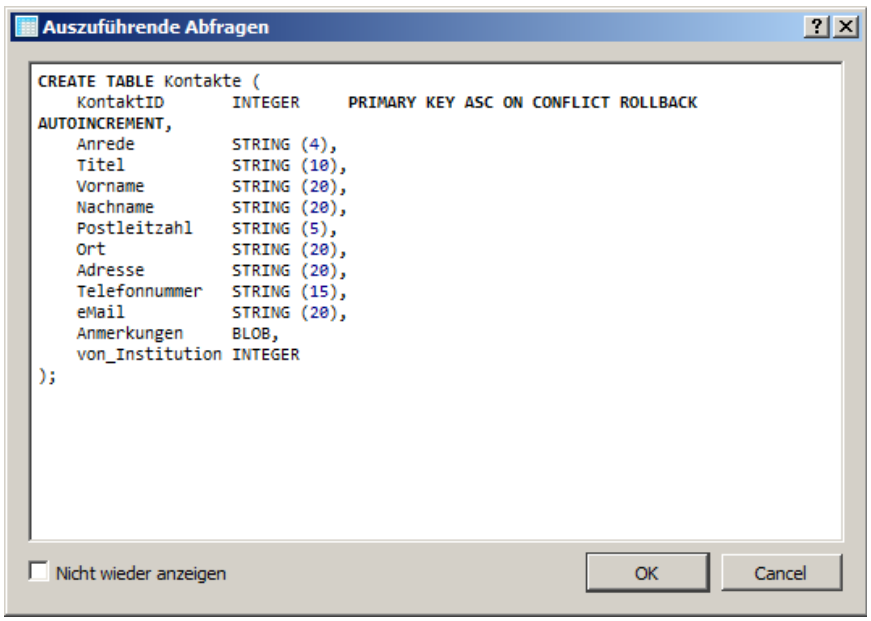




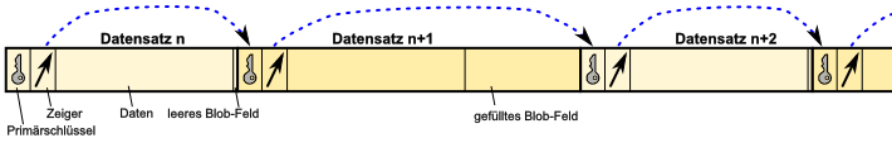
Dem aufmerksamen Leser ist sicher aufgefallen, dass ich hier wieder eine zusätzliche Spalte mit definiert habe – die "Anmerkungen". Die Anlage eine Spalte für irgendwelche Bemerkungen usw. würde ich immer empfehlen. In diese Spalte kann man für sich selbst oder andere Nutzer Hinweise hinterlassen oder Daten speichern, für es keine reguläre Spalte gibt. Der ungewöhnliche Datentyp "BLOB" steht für ein beliebig großes Textfeld. Steht nichts im Feld, dann wird auch praktisch kaum Speicherplatz gebraucht.

Auch später hat das Feld immer nur den Speicherplatz-Bedarf für soviele Textzeichen, wie wirklich eingetragen sind.

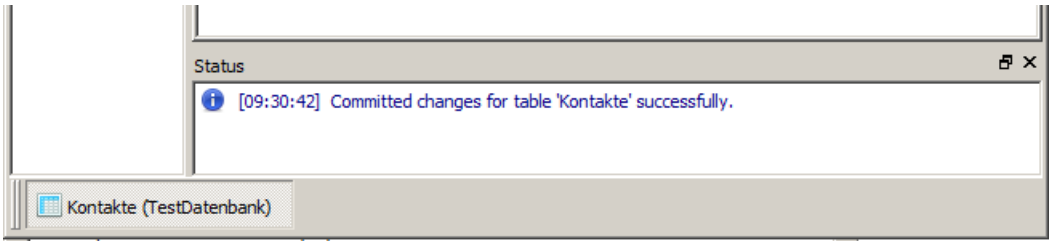
Auf den ersten Blick könnte man jetzt annehmen, dass Blob-Felder sehr günstig für Datenbanken sind. Das gilt aber nur für den reinen Speicherplatz. Durch sehr viele Blob-Felder könnte eine Tabelle aber langsamer durchsuchbar werden, da ja jedesmal der Beginn des nächsten Feldes / Datensatzes berechnet werden muss.



Die Erreichbarkeit der nachfolgenden Datensätze wird über Zeiger (Pointer, Sprung-Adressen) realisiert.

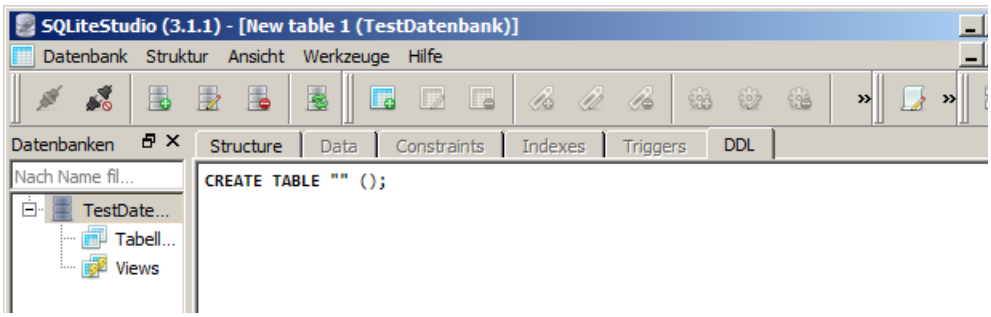


Deshalb ist es egal, ob ein Blob-Feld leer oder gefüllt ist.
Nach dem Bestätigen erhalten wir die Anzeige des kreierten SQL-Befehls für unsere zusammengeklickte Tabellen-Definition.
Die nachfolgende Status-Information ist unser Ziel.

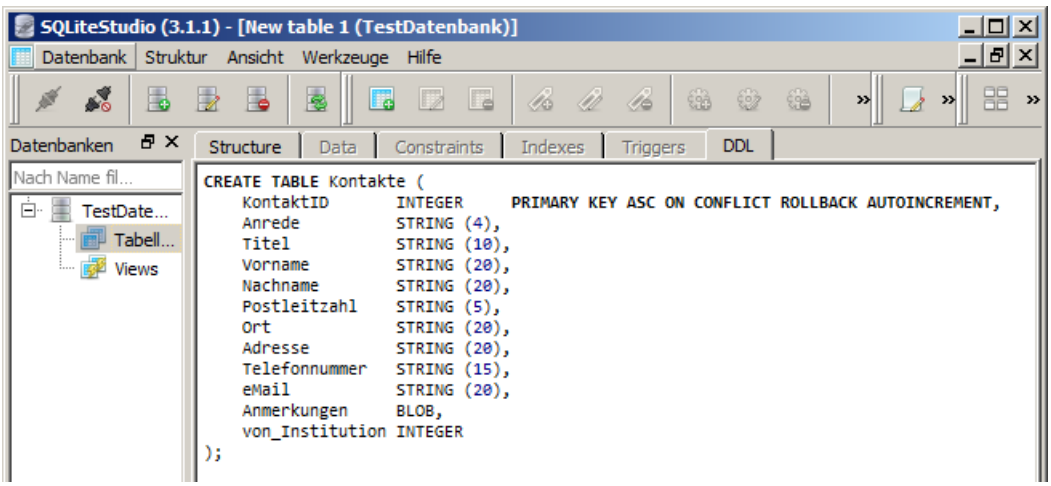


Dann ist alles ok. und wir können uns um das nächste Problem kümmern.

Unter dem Reiter "DDL" (Data Defination Language / Date Description Language) lässt sich die Struktur der Tabelle auch über einen SQL-Befehl erzeugen.



Jeder Definitions-Schritt wird gleich in SQL-Code umgesetzt, so dass am Ende der oben schon gesehene SQL-Befehl herauskommt. Hier könnten wir aber noch in die Befehls-Gestaltung eingreifen und Änderungen vornehmen.

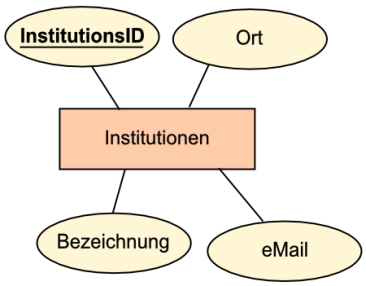


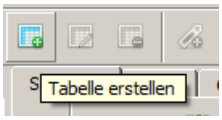
Aufgaben:

1. Erstellen Sie die Tabelle "Kontakte" mit den angezeigten Optionen!

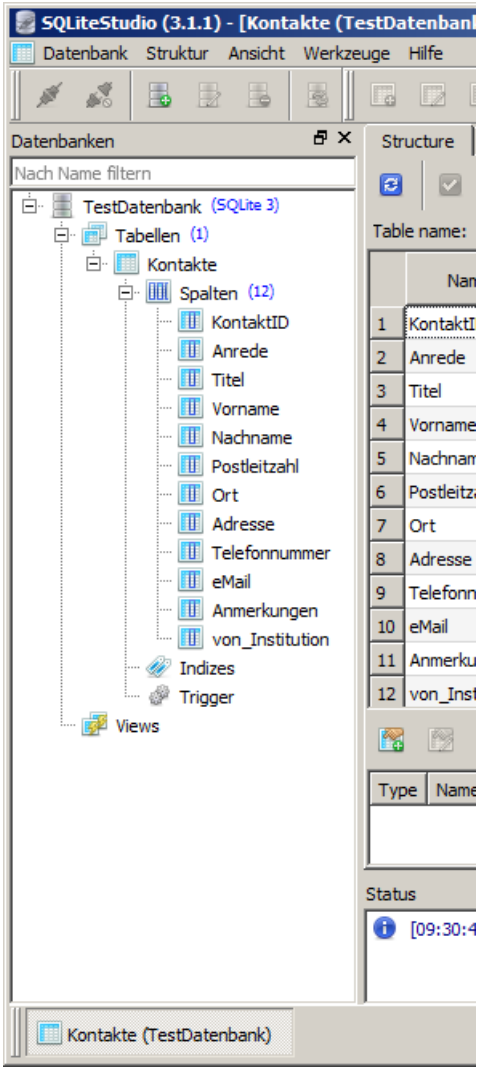
2. Erläutern Sie, was die Ausdrücke in der Zeile "KontaktID …" bedeuten!

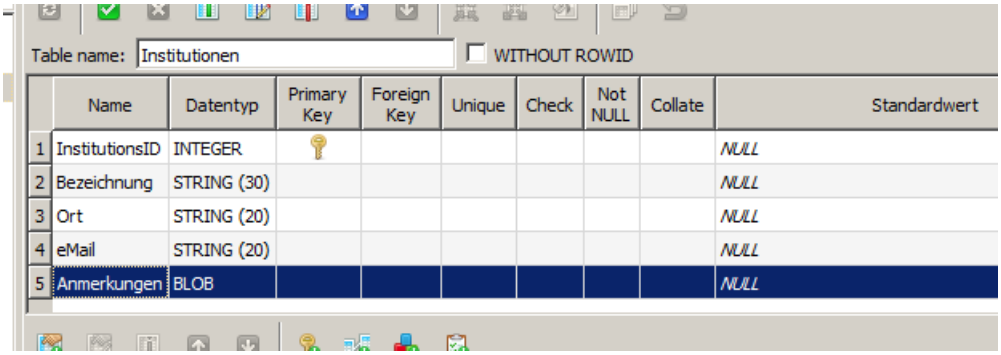
Die ordnungsgemäße Erstellung unserer Kontakte-Tabelle wird jetzt auch in der Datenbank-Struktur-Anzeige links sichtbar. Der Baum lässt sich an den Plus-Symbolen aufspreizen. Die Informationen sind dann schnell verfügbar und wir brauchen dann unsere Papier-Definition nicht mehr.
Als nächstes brauchen wir die Institutions-Tabelle. Das ERD sieht so aus:



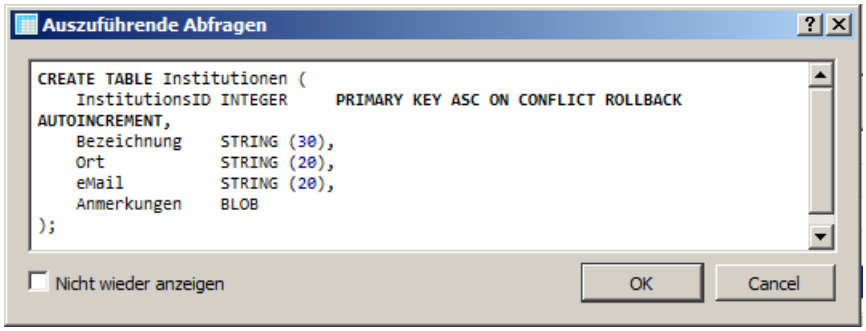


Mittels "Tabelle erstellen" wiederholen wir das eben besprochene Verfahren. Die Details sind hier weniger umfangreich.





Der SQL-Befehl wird uns wieder angezeigt. Praktisch kann man diese Anzeige zukünftig auch verhindern. Dazu muss man nur "Nicht mehr anzeigen" anklicken. In der Phase des Erlernen von SQL bzw. den ersten Arbeiten mit Datenbanken und SQL sollte man sich die Texte aber ruhig ansehen.



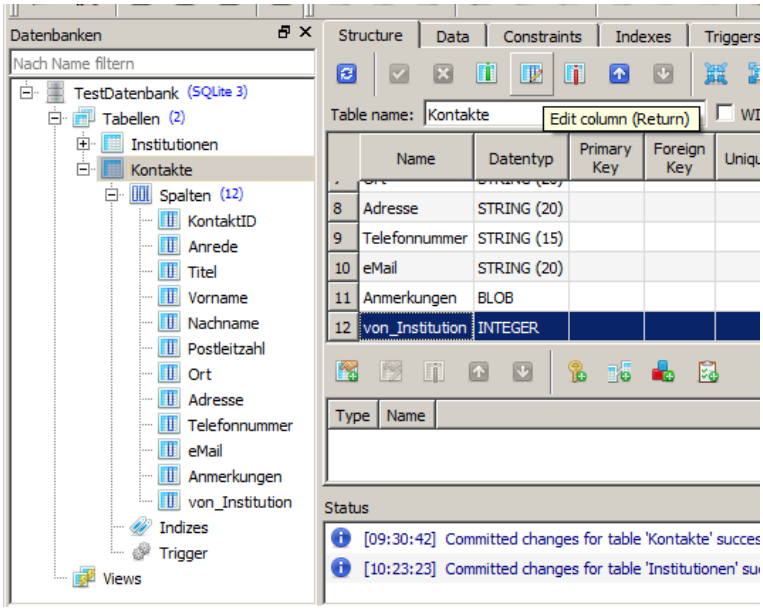
Man bekommt recht schnell ein Gefühl für die Struktur der Befehle und mit auch schon geringen Englisch-Kenntnissen kann man die Wirkung der erkennen. Das hilft auch bei der Suche nach Fehlern in SQL-Befehlen.

Aufgaben:

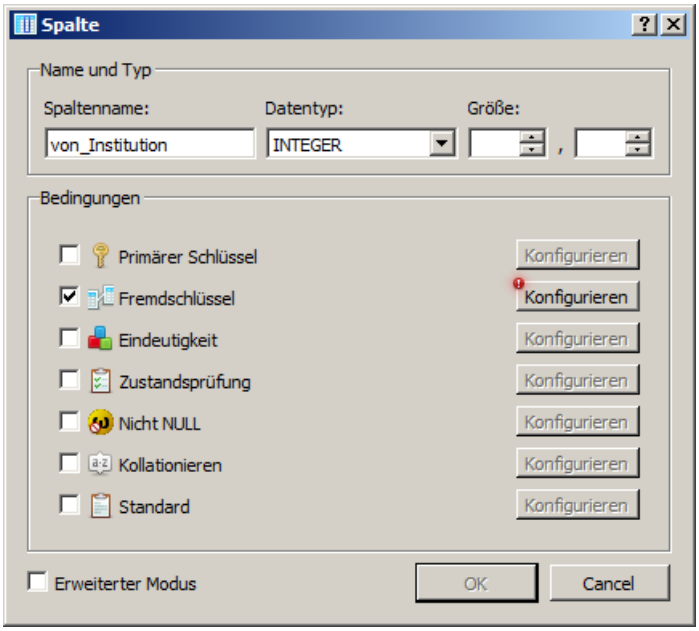
1. Erstellen Sie sich nun die Tabelle "Institutionen", wie oben gezeigt!

2.

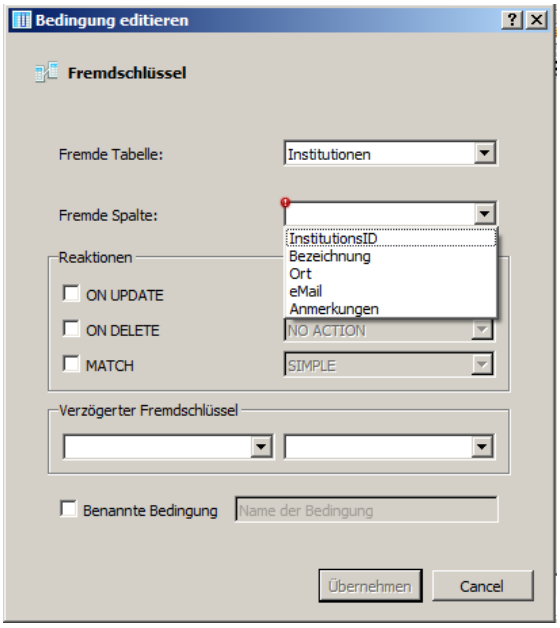
Tabellen verknüpfen – Fremdschlüssel (nachträglich) einrichten



Nach der Auswahl der (zukünftigen) Femdschlüssel-Spalte kommt man entweder über das Editier-Symbol ("") oder mittels [Enter]-Taste in die Detail-Anzeige.

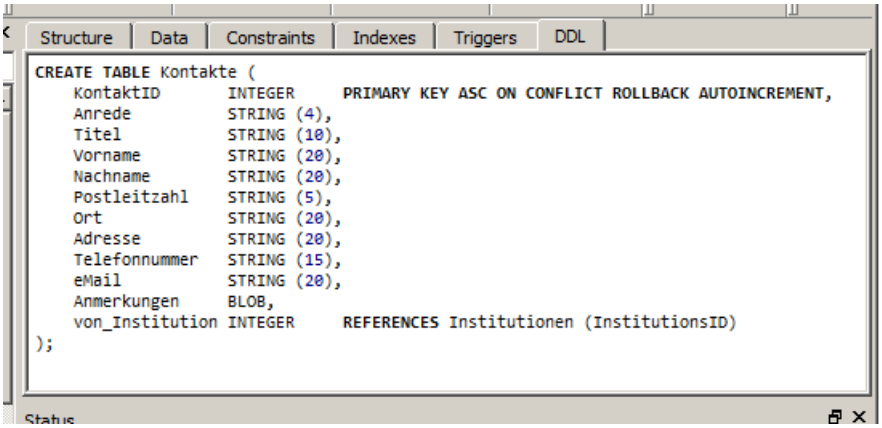


In der Konfiguration muss nun zwingend die verknüpfte Tabelle eingetragen werden. Aus dieser können wir dann die passende Primärschlüssel-Spalte auswählen. Ihre Werte werden später in der Kontakte-Tabelle als Verweis auf eine Institution verwendet.

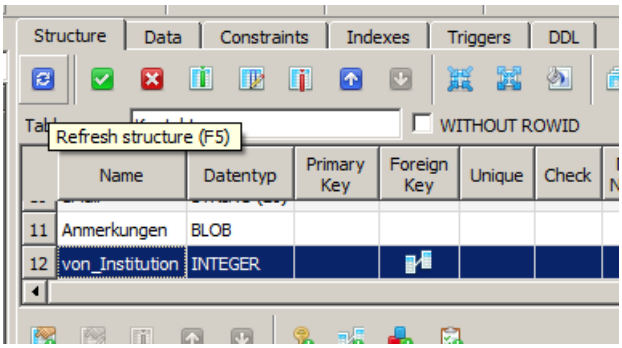


Interessant sind die Möglichkeiten, auf bestimmte Vorgänge / Veränderungen einzugehen. Dabei kann man definieren, was passieren soll, wenn die Daten aktualisiert werden (ON UPDATE), wenn sie gelöscht werden (ON DELETE) und / oder, wenn sie mit einem Wert übereinstimmen (MATCH).
Der Rest ist für die Profi's und für uns nicht wirklich wichtig.
Unter dem Reiter DDL sehen wir nun die Definition unserer Tabelle.

Allerdings ist das der SQL-Befehl für die "Ersterstellung" der Tabelle. Das haben wir ja schon erledigt. Hier soll ja "nur" die Struktur angepasst werden.
Ein Struktur-Refresh [F5] ist nicht die Lösung unseres Aktualisierungs-Problems!



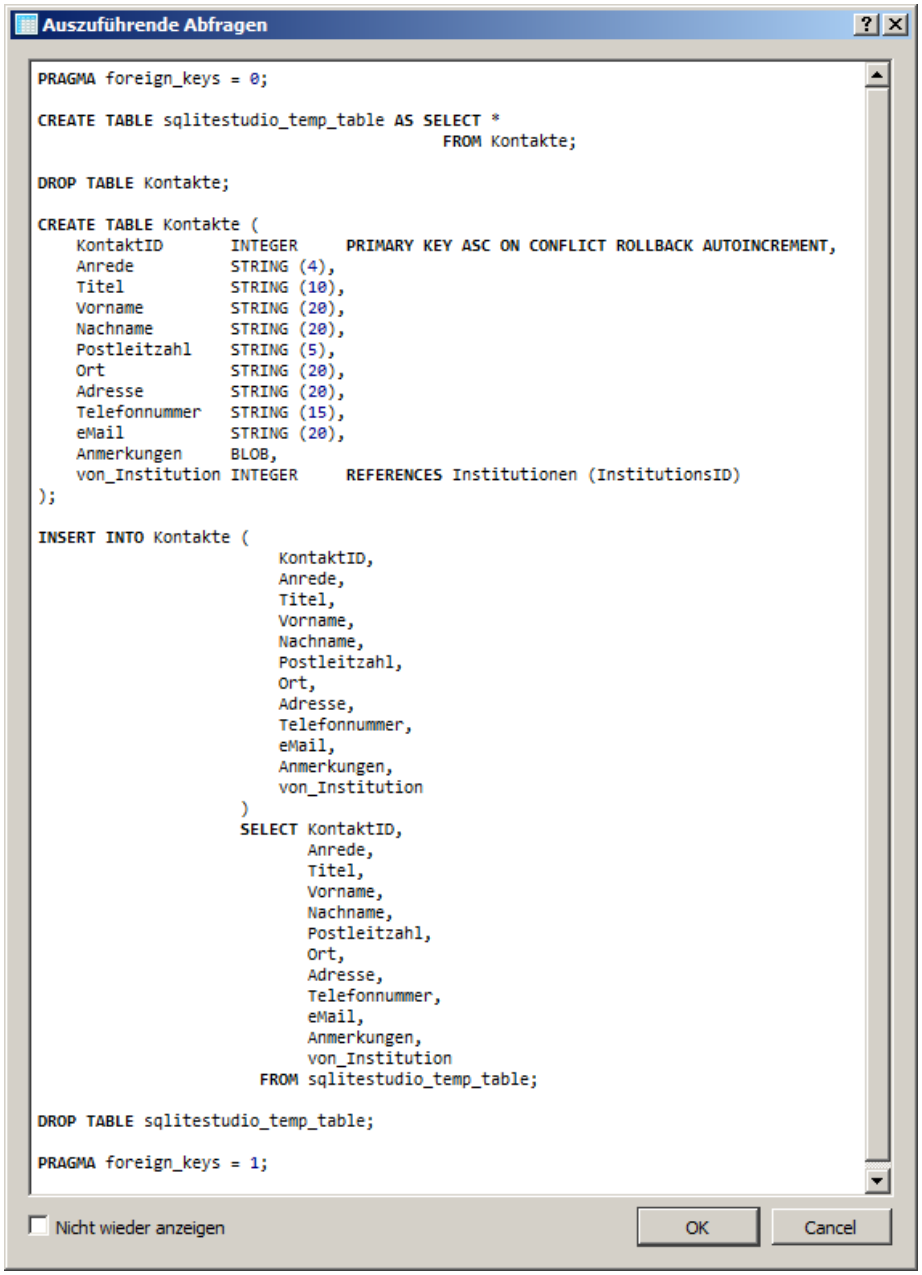
Beim einem Refresh wird die derzeit gültige Struktur in die Anzeige gebracht.
Also sollten wir das jetzt **nicht** machen, sonst müssen wir die Einrichtung des Fremdschlüssels noch einmal erledigen!
In unserem Fall ist eine Neudefinition der Tabelle "Kontakte" notwendig.
Wie aufwendig das ist, zeigt der SQL-Befehl, der nach dem Aktivieren des Grünen Häkchen vorgeschlagen wird.



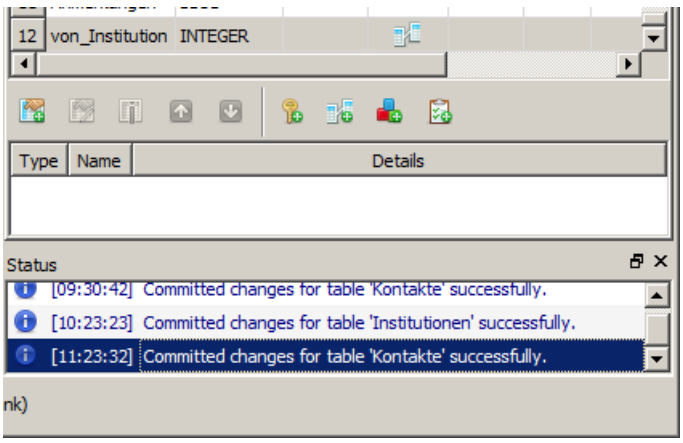
Aufgaben:

1. *Stellen Sie die Verknüpfung zwischen den Tabellen "Kontakte" und "Institutionen" her!*
2. *Warum benutzt man hier eigentlich eine zusätzliche Tabelle für die "Institutionen"? Man hätte die Daten doch gleich in die Tabelle "Kontakte" eintragen können, oder nicht? Erklären Sie ausführlich!*

Dieser SQL-Konstrukt ist wegen der lapidaren Änderung um die Erstellung eines Fremd-Schlüssels notwendig.
Was hier passiert ist auch ohne fortgeschrittene SQL-Kenntnisse interessant. Kurz erklärt geht das System so vor:
Es wird zuerst dafür gesorgt, dass die nachfolgenden Befehle ungestört hintereinander abgearbeitet werden können. Als Nächstes wird von der alten Tabelle eine Kopie als temporäre Tabelle erstellt. Sie dient dann später als Daten-Quelle für die neue Kontakte-Tabelle. Vorher wird die alte Tabelle Kontakte gelöscht.
Nach dem Erstellen der neuen Kontakte-Tabelle mit dem Fremdschlüssel folgt das Spalten-weise Kopieren der Daten aus der temporären Tabelle in die neue. Das in unserem Fall noch gar keine Daten enthalten waren spielt hier keine Rolle. Zum Schluss wird die temporäre Tabelle noch gelöscht.



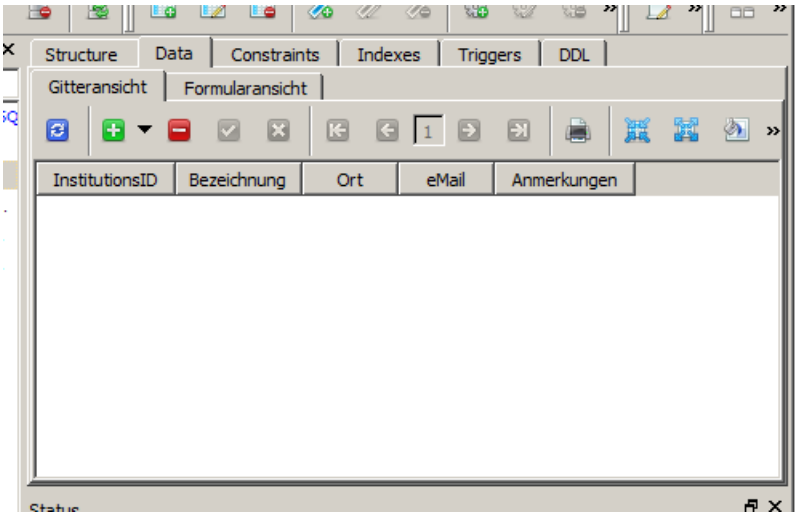
In der Status-Anzeige bekommen wir ein positives Feedback.



3.1.7.1.3.3. Daten in die Tabellen eingeben

manuelle Eingabe von Daten

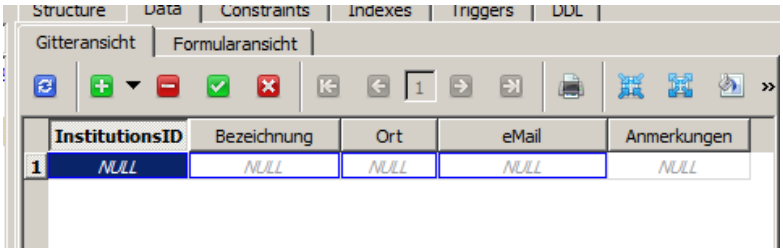
Jede Datenbank lebt von den Daten, die in den Tabellen vorhanden sind. Nur sie sind nachher für den Nutzer interessant.
Die genutzte Struktur ist dem Nutzer eigentlich völlig egal.
Wir beginnen hier bei der Dateneingabe mit der Institutions-Tabelle.
Den Zugriff auf die Tabellen-Inhalte erhält man über den reiter "Data".
Wir haben zwei Möglichkeiten Daten einzutragen. Die erste ist die sogenannte "Gitteransicht" ("Grid view"). Das ist praktisch eine ganz normale Tabellen-Ansicht.
Die zweite EingabeForm ist die "Formularansicht" ("Form view").



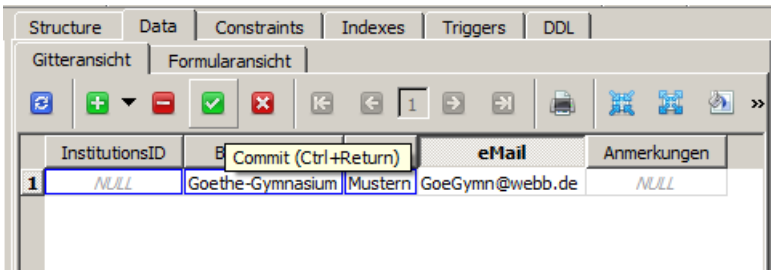
Diese besprechen wir gleich weiter hinten (→ Eingabe in der "Formularansicht").
Über das grüne Plus-Symbol legen wir einen neuen Datensatz (eine neue Zeile) in unserer Tabelle an.

Eingabe in der "Gitteransicht"

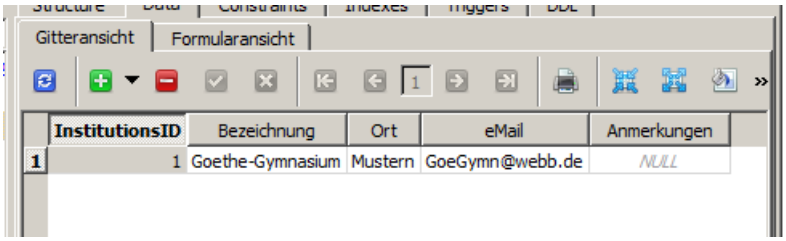
Die Eingabe-Zeile beginnt gleich mit der "InstitutionsID". Da ist man gleich verleitet, jetzt eine ID zu vergeben. Dies tun wir aber nicht, weil diese Spalte von uns als Primärschlüssel mit automatischer Hochzählung definiert wurde.

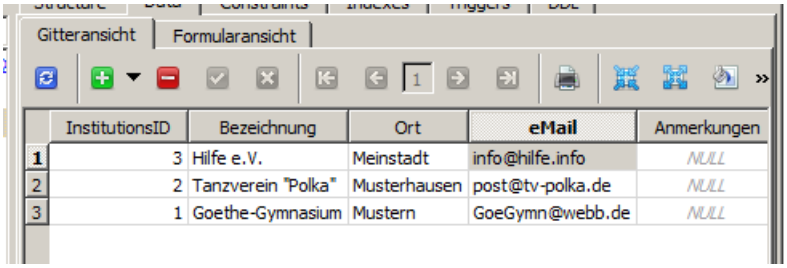


Diesen Sachverhalt muss man sich merken. Für einen Fremd-Nutzer ist das sicher ein Problem. Eine – einigermaßen gute - Verabredung ist es, die Primärschlüssel-Spalten immer mit ID enden zu lassen.
Wir beginnen also regulär mit der "Bezeichnung".

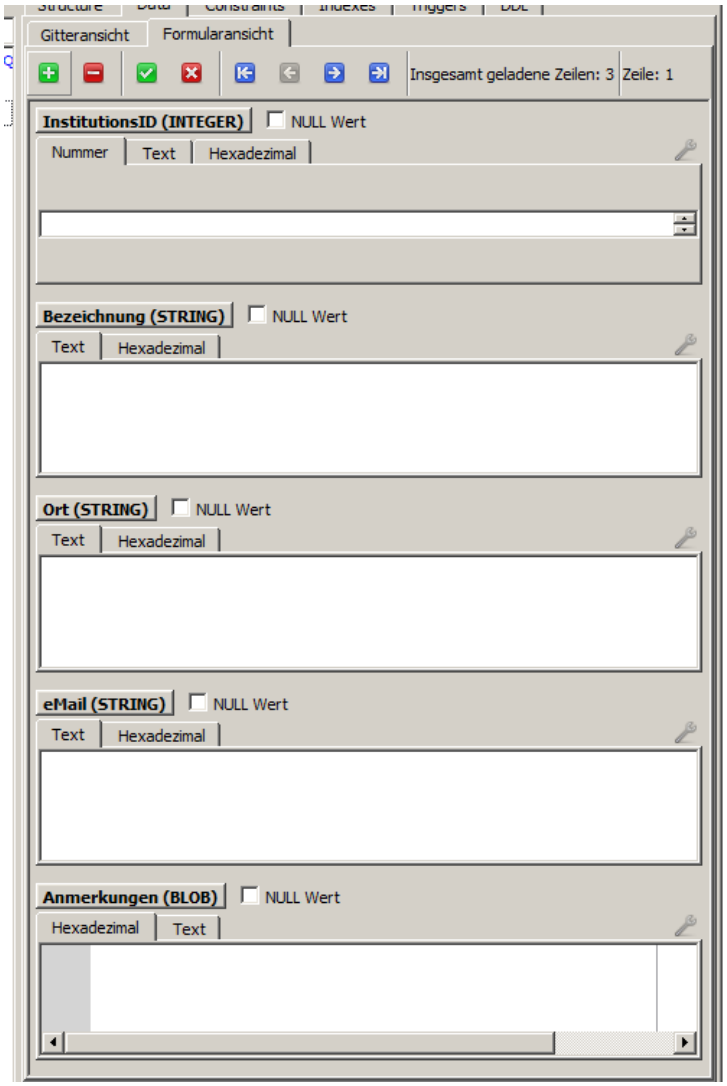


Wenn alle Daten (eines Datensatzes) eingetragen sind, bestätigen wir die Gültigkeit dieses Datensatzes durch einen Klick auf das grüne Häkchen. Wie im kleinen Hilfe-Text angezeigt geht auch die Tasten-Kombination [Strg] [Enter].
Der Datensatz bekommt dann automatisch die nächste Primärschlüssel-Nummer – hier eben die "1".
So kann man relativ schnell Tabellen mit Daten füllen.





Eingabe in der "Formularansicht"



Die Formularansicht verspricht eigentlich eine schönere Eingabe. In der schematischen Umsetzung innerhalb von SQLiteStudio ist sie aber eher unübersichtlich und nur für kleine Tabellen sinnvoll.
Ein Wechsel von Attribut zu Attribut mit [Tab] wird hier durch Zwischen-Stopp's bei den Formular-Element-Reitern aufwendiger.
Auch NULL-Werte muss man anlegen, was den Eingabe-Aufwand nochmals vergrößert.
Zu guter Letzt spricht auch noch die hier notwendige Angabe der ID für Verwirrung. Bei Nicht-Eingabe gibt es eine Fehler-Meldung im Status-Bereich. Der Nutzer kann aber kaum wissen, welche Nummer als nächstes benutzt werden kann.
Zu erwähnen ist aber der Datensatz-Navigator mit den blauen Sprung-Schaltflächen. Mit ihm kann man schön durch die einzelnen Datensätze durchnavigieren.
Wichtig ist auch hier, dass neue Datensätze oder die Änderungen in Felder zumindestens am Schluss über das grüne Häkchen (oder [Strg] [Enter]) bestätigt werden müssen!
Die Übernahme eines Datensatzes in die innere Datenbank (Database) erfolgt nur als Ganzes. Entweder wird die gesamte Transaktion ausgeführt oder eben nicht.

Aufgaben:

1. ***Geben Sie die angezeigten Daten manuell in die Tabelle ein!***
2. ***Probieren Sie auch die Formular-Ansicht aus, in dem Sie drei weitere "Institutionen" im obigen Stil (neue imaginäre Einrichtungen mit den vorgegebenen Orten!) ergänzen!***

Import von Daten

Vielfach stehen Daten schon in irgendeiner digitalen Form vor. Solche Daten möchte man nicht wirklich noch einmal abtippen. Die meisten Datenbanken bestehen ja nicht gerade aus einigen kleinen Tabellen mit wenigen Zeilen.
Die meisten modernen Programme bieten zum besonderen Speichern oder Exportieren bestimmte Datei-Formate an. Sehr verbreitet sind CSV-, XML- und JSON-Dateien. SQLiteStudio kann z.B. CSV-Dateien importieren. Diese lassen sich gut mit einem Editor (z.B. NotePad) oder Tabellenkalkulationen (wie microsoft EXCEL oder libreoffice / openoffice CALC) erzeugen, bearbeiten und kontrollieren.
Unsere Kontakte-Tabelle wollen wir nun per Import füllen. Voraussetzung ist eine CSV-Datei. Der grundsätzliche Aufbau ist leicht erklärt. Das CSV-Format beschreibt eine reine Text-Datei. Jede Zeile stellt einen Datensatz dar und muss mit einem Enter bzw. abgeschlossen werden. Die einzelnen Felder in einer Zeile sind durch ein definiertes Trennzeichen separiert. Klassischerweise benutzt man ein Komma (","), das Semikolon (";") oder einen Tabulator(-Sprung). Texte werden ev. in Anführungszeichen gesetzt, die Zahlen und z.B. Datum- und Zeit-Angaben werden direkt notiert. Komma-Zahlen (reele Zahlen) werden – wie in der IT meist üblich – in der der englischen Schreibweise (mit einem Punkt als Dezimal-Trenner) notiert.
Eine Komma-separierte CSV-Datei mit den Daten für die "Kontakte"-Tabelle könnte so aussehen:

kontakte.csv

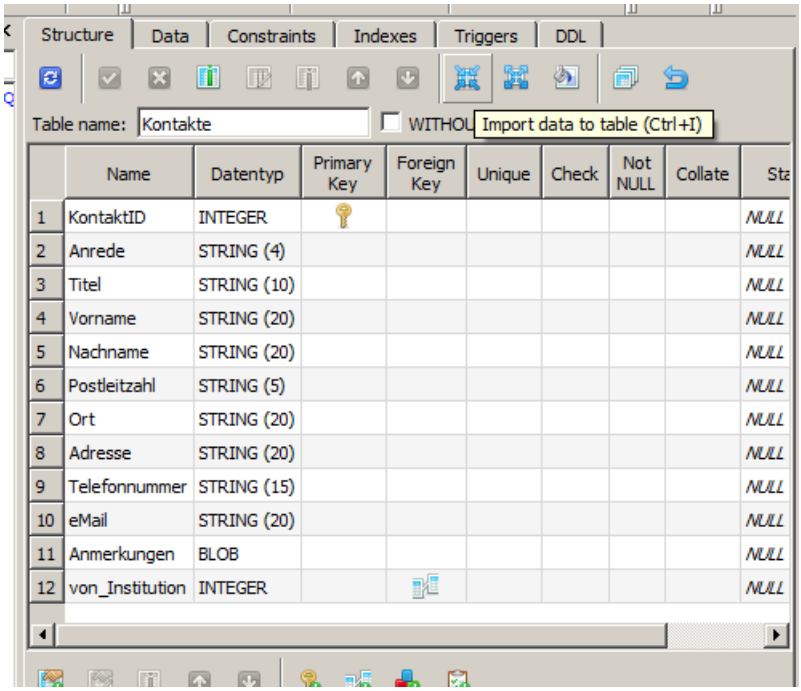
```
Kontak-
tID,Anrede,Titel,Vorname,Nachname,Postleitzahl,Ort,Adresse,Telefonnummer,eM
ail,Anmerkung,von_Institution✓
1,Herr,Dr.,Klaus,Mustermann,12345,Musterhausen,Musterstr.13,0123456789, mus
termann@webb.de,,1✓
2,Frau,,Monika,Mustermann,12345,Musterhausen,Musterstr. 13,0123456789, must
erfrau@webb.de,,2✓
3,Herr,,Hans,Fehler,34343,Glückstadt an der Pech,Blumenweg 48,088088088, H.
Fehler@webb.de,,1✓
4,Frau,,Maria,Muster,23456,Mustern,Am Musterweg 3,0234567890,m.muster@tee-
online.de,,4✓
5,Herr,,Christian,Bauer,67890,Berg am Fluß,Waldallee 78,04567890901, Chr.Ba
uer@geemx.de,,5✓
6,Herr,,Lucas,Müller,54321,Bedorf,Feldweg 7,09998765,Mueller@tee-online.de,
,1✓
7,Frau,,Tara,Zander,23456,Mustern,Hauptstr. 6e,0777711,Ta.Zan@webb.de,,6✓
8,Frau,Prof.,Hertha,Ziesow,88888,St. Muster,An der B777,0543861,Prof.H.Zies
ow@tee-online.de,,4✓
12,Frau,,Maria,Berndt,67890,Berg am Fluß,Hans-Wald-Str. 4,0456783067,,,3✓
37,Herr,,Henriette,Krüger,76543,Meinstadt,Feldweg 7,06543210,post@h-krueger
.de,,1✓
```

Zur Sicherheit habe ich die Positionen für das Enter-Zeichen mit einem besonderen blauen Zeichen versehen. Normalerweise ist das Zeichen nicht sichtbar. In einem Hex-Editor sieht man dann die Sequenz: **0D 0A**. Das steht für **CR** und **LF**.

Aufgaben:

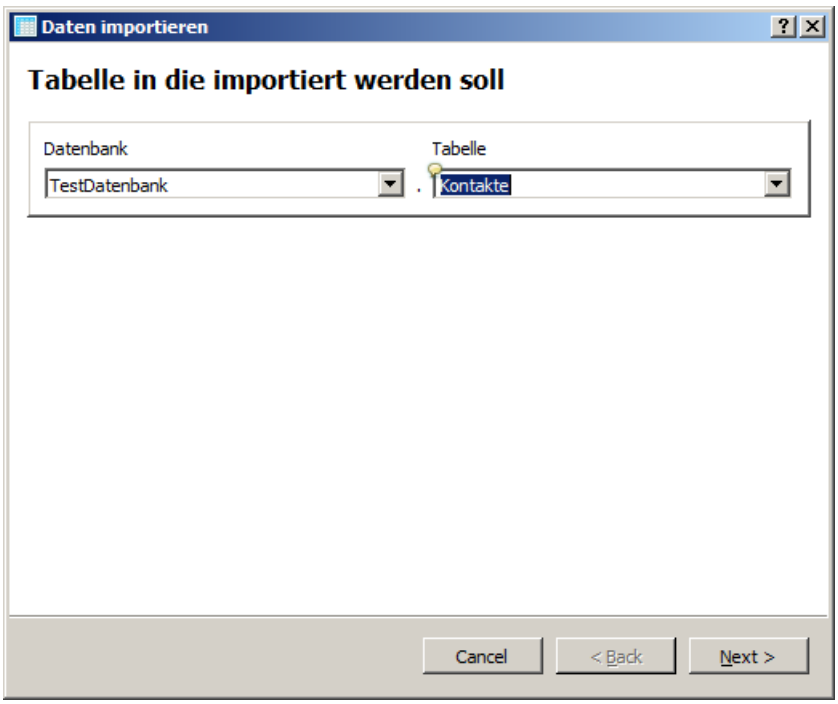
1. *Informieren Sie sich über die Bedeutung / ursprüngliche Verwendung von CR und LF!*
2. *Vergleichen Sie die Codierung von Texten mit ASCII- und Unicode-Zeichensatz!*
3. *Welche Bedeutung hat die Kenntnis über die Codierung einer Text-Datei für einen Import in Datenbank-Systemen? Beachten Sie dabei auch, dass die meisten Datenbank-Systeme auf unterschiedlichen Betriebssystemen laufen (müssen)!*

Die Symbolleiste der Struktur-Ansicht enthält den Import-Button. Die vier blauen nach innen gerichteten Pfeile sind als Symbol wohl sehr passend.

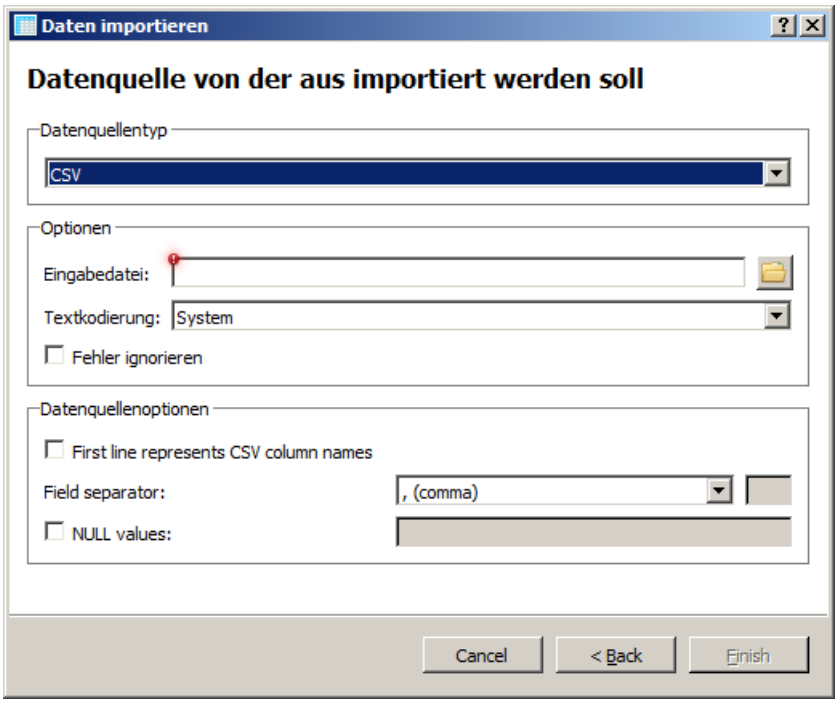


	Name	Datentyp	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Sta
1	KontaktID	INTEGER							NULL
2	Anrede	STRING (4)							NULL
3	Titel	STRING (10)							NULL
4	Vorname	STRING (20)							NULL
5	Nachname	STRING (20)							NULL
6	Postleitzahl	STRING (5)							NULL
7	Ort	STRING (20)							NULL
8	Adresse	STRING (20)							NULL
9	Telefonnummer	STRING (15)							NULL
10	eMail	STRING (20)							NULL
11	Anmerkungen	BLOB							NULL
12	von_Institution	INTEGER							NULL

Es startet ein kleiner Assistent, der vor und zurück durchgearbeitet werden kann.
Viele Daten sind notwendig. Als erstes müssen wir z.B. die Ziel-Tabelle spezifizieren. Im Normalfall ist das die, aus deren Struktur-Ansicht heraus man den Import-Assistenten aufgerufen hat.
Das gelbe Lämpchen ist nur noch mal ein Hinweis auf die Kontrolle durch den Nutzer.
Mittels "Next"-Schaltfläche geht es zum nächsten Assistenten-Schritt.

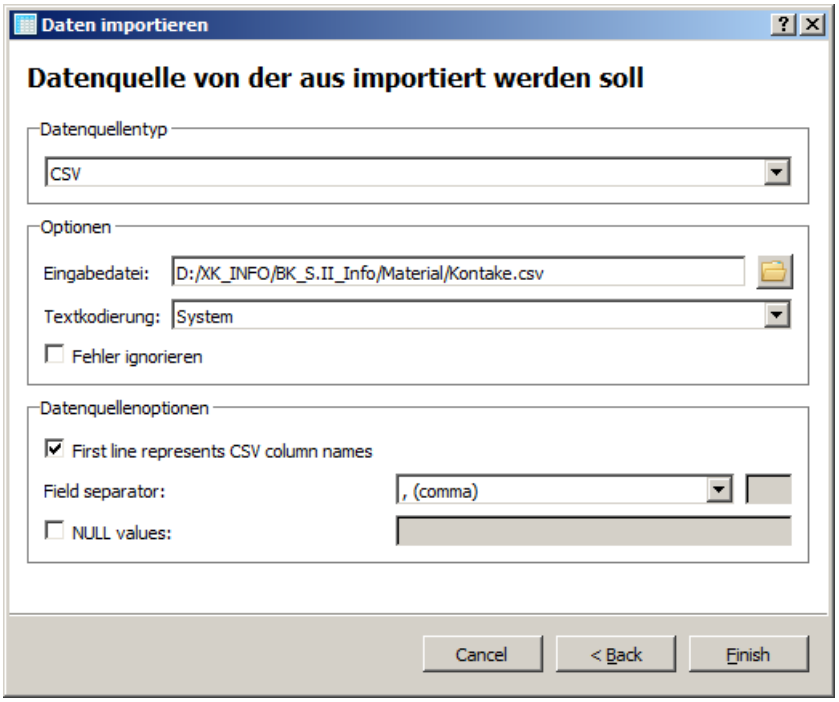


Hier stellen wir nun die Details zum Import ein.
Bei "Datenquellentyp" kann man zwischen einer reinen Daten-Datei im CSV-Format und einer SQL-Datei ("RegExp") auswählen.
Da unsere Daten – wie schon vorgestellt – im CSV-Format kommen, bleiben wir bei der Vorgabe.
Die "Eingabedatei" wird über das "Ordner"-Symbol rechts ausgewählt.
Die Textcodierung kann eine Herausforderung sein. Vor allem, wenn man Daten aus anderen Betriebssystemen importiert, dann lohnt hier eine andere Wahl.

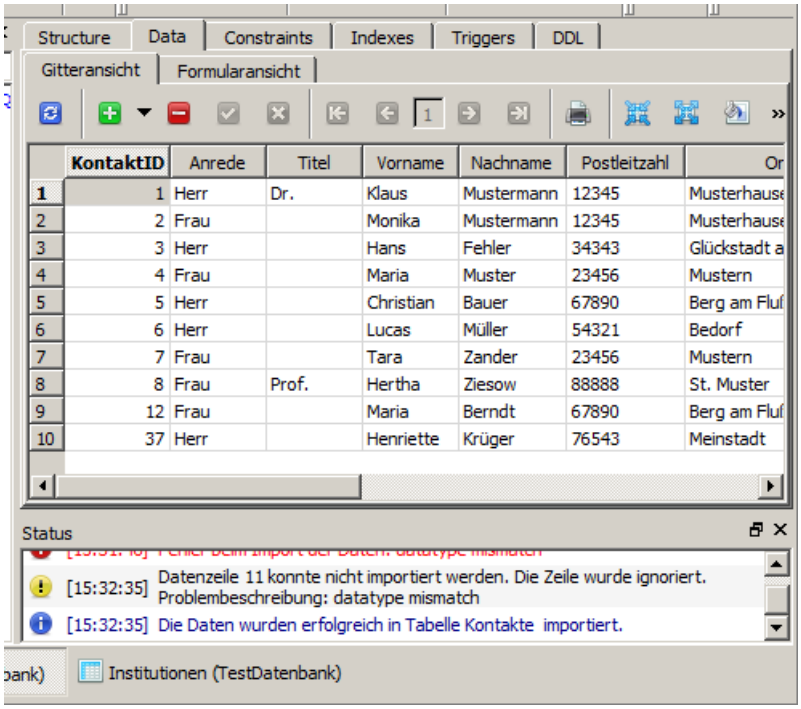


Mit "System" wird hier die Codierung genommen, die auf dem aktuellen Betriebssystem gültig ist. Da meine CSV auch im gleichen Betriebssystem erstellt wurde, bleibt die Vorgabe auch hier erhalten.

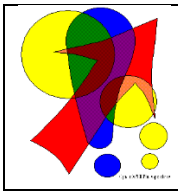
Bei den "Datenquelloptionen" muss man nun die CSV-Datei oder die Herstellungs-Optionen genauer kennen.
Aus dem Abdruck oben können wir erkennen, dass die erste Zeile Spalten-Namen enthält ("First line represnts CSV column names"). Der Feld-Seperator ist ein Komma.
Sollte andere Zeichen benutzt worden sein, dann wird hier die entsprechende Wahl getroffen.
Am Schluss bestätigen wir bei "Finisch" ("Beenden").
Sollte eine Fehler-Meldung auftauchen, dann empfiehlt sich eine Interpretation der Meldung und ev. die Überprüfung der Optionen.



Eine gute Möglichkeit ist die Option "Fehler ignorieren" zu setzen, um das System etwas Fehler-toleranter zu machen.
So sind z.B. Leer-Zeilen – wie sie häufig als letzte Zeile – in einer CSV-Datei vorkommen (abschließendes [Enter]) ein Problem für den Import-Filter von SQLiteStudio.
Wenn alles geklappt hat, dann erhalten wir im Status-Feld eine passende Meldung.
Die Daten stehen sofort für weitere Arbeiten zur Verfügung.



	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Or
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhause
2	2	Frau		Monika	Mustermann	12345	Musterhause
3	3	Herr		Hans	Fehler	34343	Glückstadt a
4	4	Frau		Maria	Muster	23456	Mustern
5	5	Herr		Christian	Bauer	67890	Berg am Flu
6	6	Herr		Lucas	Müller	54321	Bedorf
7	7	Frau		Tara	Zander	23456	Mustern
8	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster
9	12	Frau		Maria	Berndt	67890	Berg am Flu
10	37	Herr		Henriette	Krüger	76543	Meinstadt



Bemerkungen zur führenden Null in Texten (z.B. PLZ und Telefonnummern):
Sowohl der Import – als auch die manuelle Eingabe oder Korrektur – ergibt keine führenden Nullen in Text-Feldern. Dies ist ein bekannter Bug, der sicher in einer der nächsten Versionen behoben wird.
Als Alternativ-Eingabe schlagen die Entwickler einen Notierung in einfache Hochkommata vor.

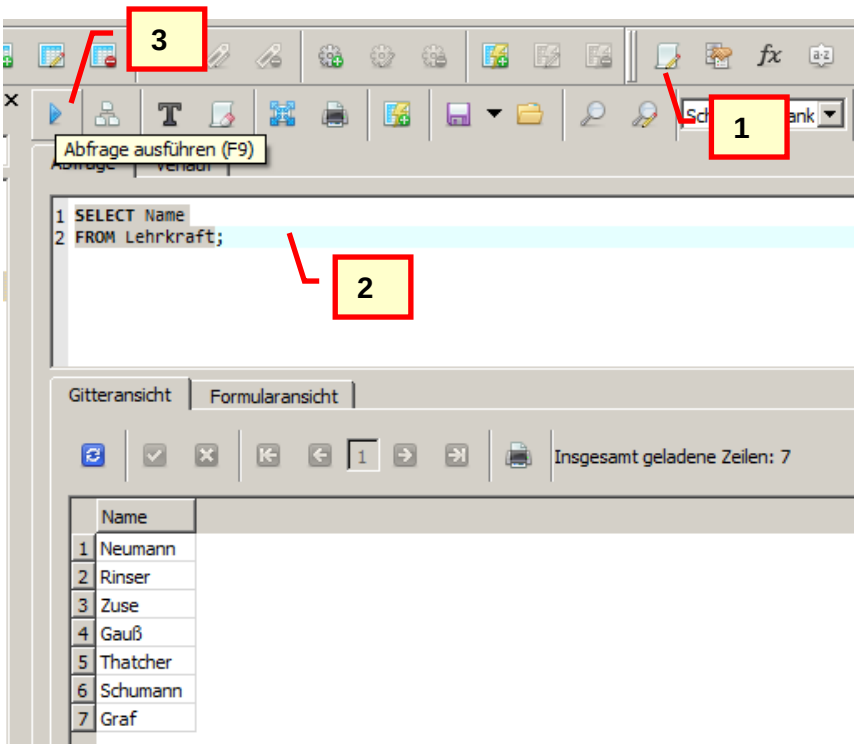
Aufgaben:

1. *Erstellen Sie sich in einem Editor eine CSV-Datei "Kontakte" mit den angegebenen Daten!*
2. *Importieren Sie diese dann in die zugehörige Tabelle im SQLiteStudio!*

3.1.7.1.3.4. Erstellen von Abfragen

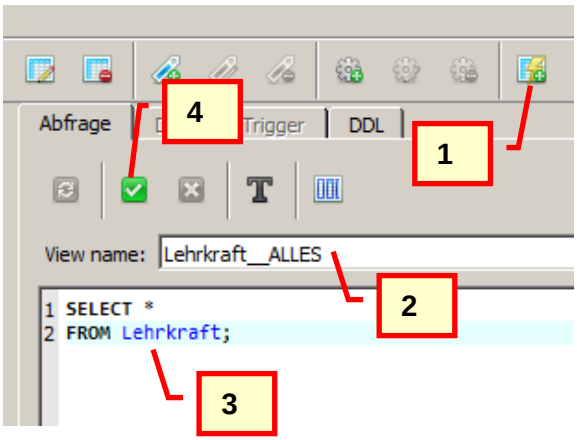
einfache Variante (temporäre Arbeits-Abfrage)

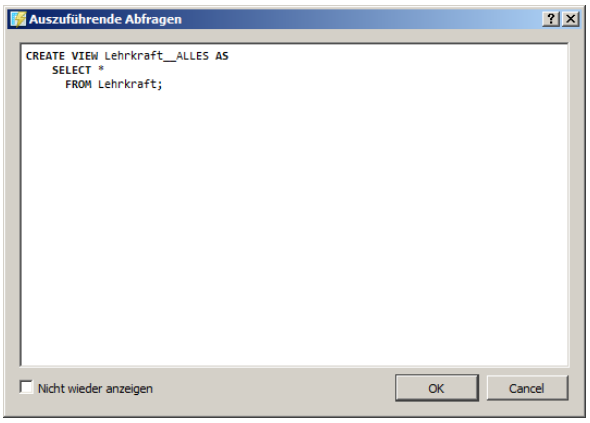
Dazu wählt man zuerst "Open SQL &editor" (**1**) aus. Unter dem Reiter "Abfrage" kann jetzt ein SQL-Statement eingetragen werden (**2**) und dann sofort beim blauen Dreieck (**3**; "Abfrage ausführen") abgearbeitet werden.
Das Ergebnis erscheint dann etwas tiefer in der Gitter- oder Formular-Ansicht. Letztere ist aber etwas gewöhnungsbedürftig.
Wer Tastatur-betont arbeiten möchte, benutzt zum Öffnen des SQL-Editors [Alt] + [E] und zum Ausführen [F9].



Variante mit Abspeichern / Sichern der Abfrage

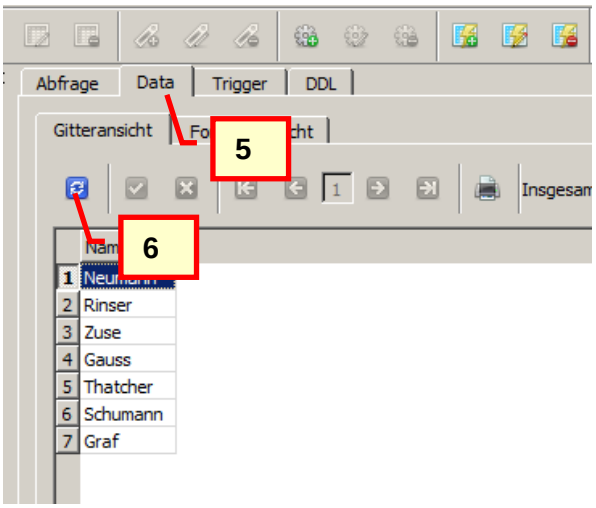
Hierfür wählen wir zuerst einmal "Create a view" (**1**). Die Sicht bekommt nun einen sinnvollen Namen (**2**).
Für den View-Namen sollte man nur Buchstaben, Ziffern und den Unterstrich verwenden.
Dann gibt man das SQL-Statement ein (**3**) und führt beim grünen Häckchen aus (**4**).





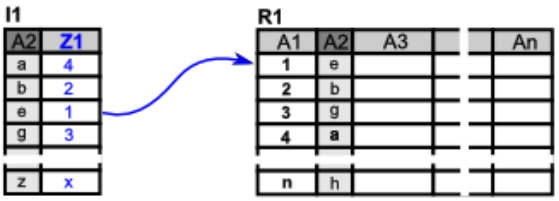
Unter "Data" sehen wir das Ergebnis.

Soll eine definierte Abfrage neu angewendet werden, dann wechselt man zum Reiter "Data" (**5**) und macht hier eine "Aktualisierung der Tabellendaten" (**6**; blaues Symbol mit Kreis-Pfeilen).
Das Tastatur-Kürzel hierfür ist [F5].



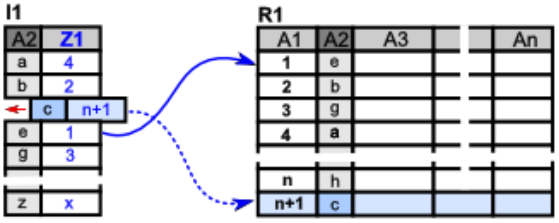
3.1.7.1.3.5. Erstellen von Indizes

Die Suche in großen Datenbanken kann schnell zum Problem werden. Man denke z.B. an eine Suche im Telefonbuch. Wir würden bei der Suche nach einem Eintrag sicher gleich viele Seiten überspringen, um dichter an den gesuchten Eintrag zu kommen. In einer Datenbank sind die Daten normalerweise ja nicht sortiert. Sie wurden dort eingetragen, wie sie kommen. Also müsste beim Suchen vorne anfangen und sich dann Datensatz für Datensatz durch die Tabelle hangeln, bis man den passenden Eintrag gefunden hat. In großen Datenbanken funktioniert dieses Vorgehen nur, wenn man beliebig Zeit hat, aber wer hat das schon.
Eine erste Möglichkeit zur Lösung des Problems wäre es, die Tabelle nach jedem neuen Eintrag neu zu sortieren. Auch dies wird bei großen Tabellen kaum realisierbar sein, weil es einfach zu viel Zeit benötigen würde. Außerdem ist nicht immer klar, nach welcher Spalte / nach welchem Attribut sortiert werden soll.
Abhilfe schaffen sogenannte Index-Strukturen. Sie stellen kleine Hilfs-Tabellen dar. Statt hier alle Daten zu speichern, werden außer dem Ordnungs- oder Sortier-Wert noch ein Verweis auf den regulären Datensatz gespeichert. Die Verweise heißen in der Informatik Zeiger. In der Datenbank-Praxis ist das der Primärschlüssel des verwiesenen Datensatzes.



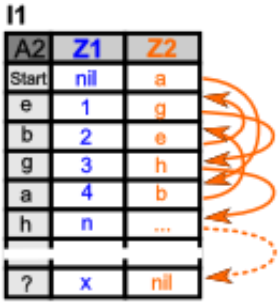
Prinzip eines tabellarischen Index

Vereinfacht kann man sich das so vorstellen: Für die Tabelle 1 (Relation1 = R1) wird z.B. ein tabellarischer Index 1 (I1) bezüglich des Attributes A2 erzeugt. Wir gehen vereinfacht davon aus, das sich die Attribut-Werte von A2 nicht wiederholen. In der Index-Tabelle ist jetzt jedem Attribut-Wert von A2 – also hier den Kleinbuchstaben von **a** bis **z** – ein Zeiger auf den zugehörigen Datensatz in Tabelle R1 zugewiesen.



Aktualisierung der Index-Tabelle nach Hinzufügen eines Datensatzes in der Verweis-Tabelle

Kommt nun ein neuer Datensatz in Tabelle R1 dazu, dann wird dieser im Normalfall angehängt.
Für den neuen Datensatz wird ein neuer Index-Eintrag erzeugt und – in einer einfachen Variante – an die richtige Sortier-Position eingefügt.
Noch effektiver wäre eine Zwei-Zeiger-Index-Tabelle. Der eine Zeiger verweist auf den Datensatz in R1 und der zweite Zeiger ist für die Gestaltung der Sortierreihenfolge in I1 verantwortlich. In so einer Tabelle bräuchten statt der aufwändigen Umsortierung ab Eintrag e in I1 nur noch zwei Zeiger umgestaltet ("verbogen") werden und schon wäre auch der Index fast sofort aktualisiert.
Diese kleinen Tabellen lassen sich sehr schnell durchsuchen, sortieren, umorganisieren und notfalls auch neu anlegen.

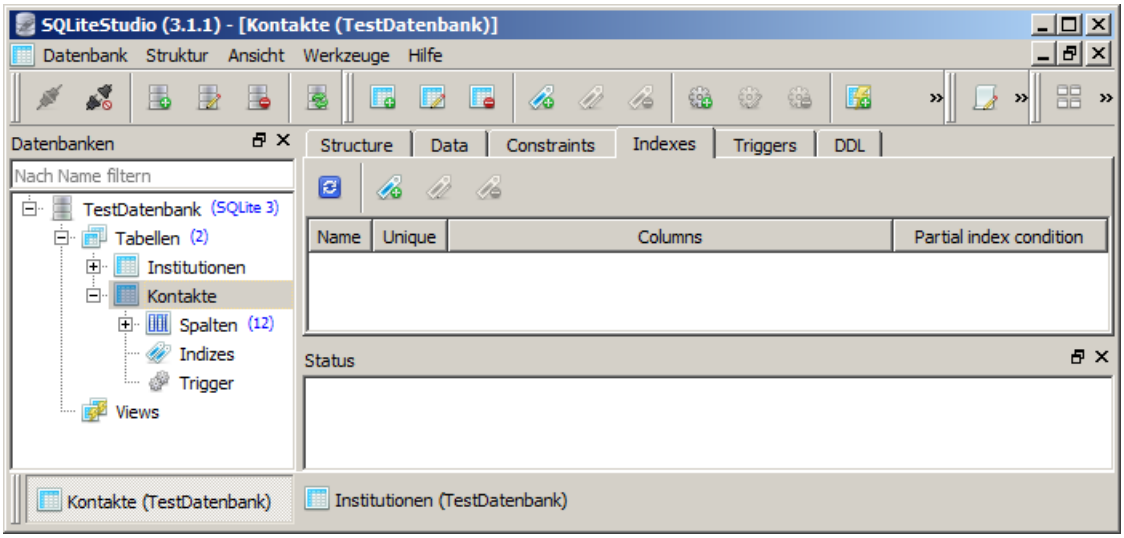


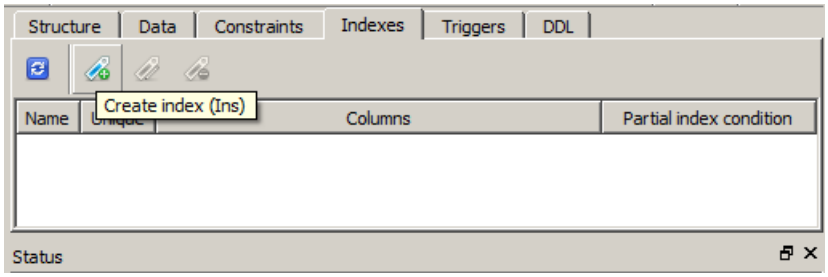
tabellarischer 2-Zeiger-Index

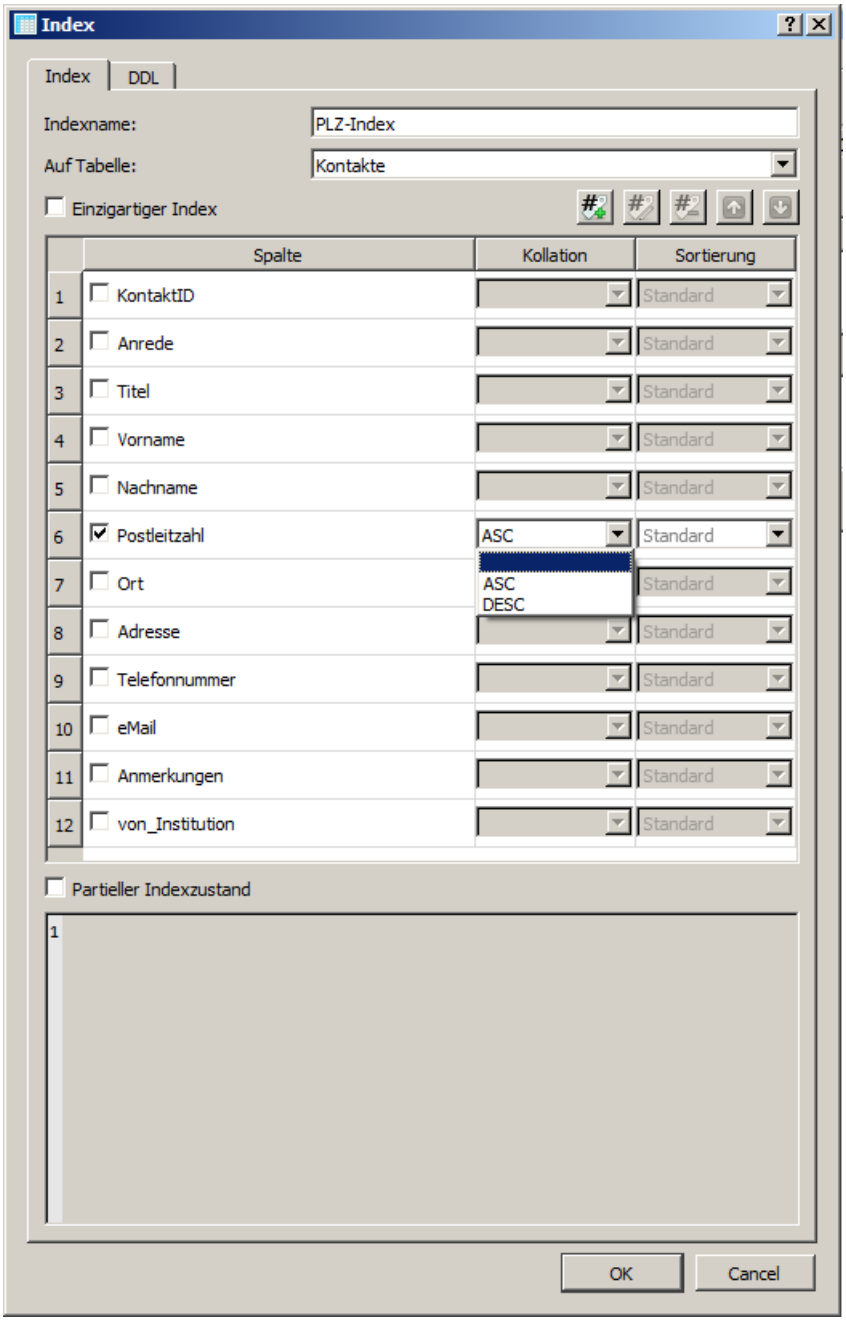
Aufgabe zwischendurch (für die gehobene Anspruchsebene):

1. ***Welche Veränderungen müssen am 2-Zeiger-Index vorgenommen werden, wenn ein neuer Datensatz (wie oben: n+1) in R1 aufgenommen wird? Schätzen Sie den Zeitaufwand im Vergleich zum 1-Zeiger-Index oben ab!***

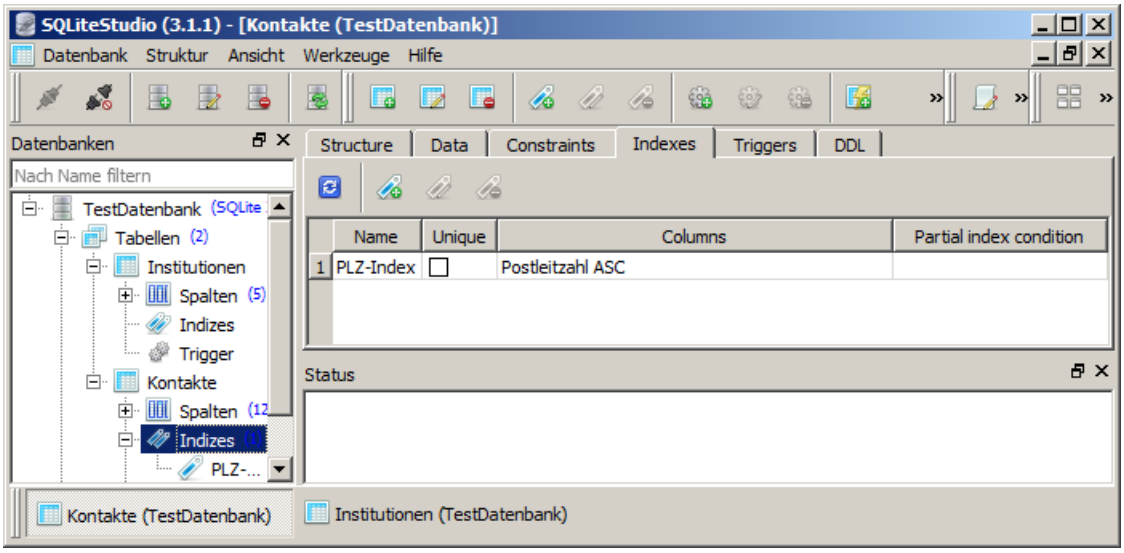
In modernen Systemen werden zudem statt Tabellen weitaus effektivere Datenstrukturen, wie z.B. Bäume (genau: B^+-Bäume), benutzt. Die Aktualisierung eines Index kann zudem parallel zur Eingabe / Veränderung anderer Einträge erfolgen. Dabei werden Stillstand- oder Nacht-Zeiten besonders effektiv ausgenutzt. Die zunehmende Hardware-Parallelisierung in der Verarbeitung von Daten verbessert die Arbeit mit Indizes nochmals. Je größer Daten-Tabellen sind und nach umso mehr Attributen sie durchsucht / sortiert werden sollen, umso effektiver sind Indizes.
Vorteilhaft ist, dass man sich zu einer Tabelle beliebig viele Indizes anlegen kann. So kann ein Index die Sortierung der Namen enthalten, während ein anderer die Sortierung der Telefonnummern verwaltet. Es lassen sich auch Spalten-Kombinationen (Attribut-Kombinationen) indizieren, was z.B. für Shop- und Personen-Suche-Systeme interessant ist.
Leider gibt es keine verbindlichen Standard's für Indizes in SQL. Jedes Datenbank-System hat da seine eigenen Vorgehensweisen. Was hier zählt ist ja auch die Effektivität bei der praktischen Arbeit. Nur ganz elementare Befehle haben sich als "Praxis-Übereinkunft" durchgesetzt. Dazu gehören CREATE und DROP INDEX.
Indizes haben aber auch Nachteile. Werden Datensätze verändert, dann müssen eben auch immer die Indizes angepasst werden. Das bedeutet zusätzlichen Rechenaufwand. Zudem sind Indizes oft nur im Speicher vorhanden. Dadurch wächst der Speicher-Bedarf solcher Systeme oft dramatisch an. Hier muss zwischen Aufwand und Nutzen abgewägt werden.






Index
Index
DDL
Indexname:
PLZ-Index
Auf Tabelle:
Kontakte
Einzigartiger Index
Spalte
Kollation
Sortierung
1 KontaktID
2 Anrede
3 Titel
4 Vorname
5 Nachname
6 Postleitzahl
7 Ort
8 Adresse
9 Telefonnummer
10 eMail
11 Anmerkungen
12 von_Institution
Standard
ASC
ASC
DESC
Partieller Indexzustand
1
OK
Cancel


Auszuführende Abfragen
CREATE INDEX [PLZ-Index] ON Kontakte (
Postleitzahl ASC
);
Nicht wieder anzeigen
OK
Cancel

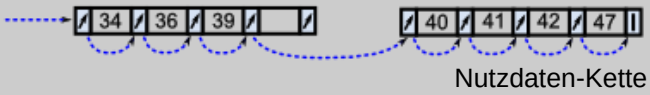


Aufgaben:

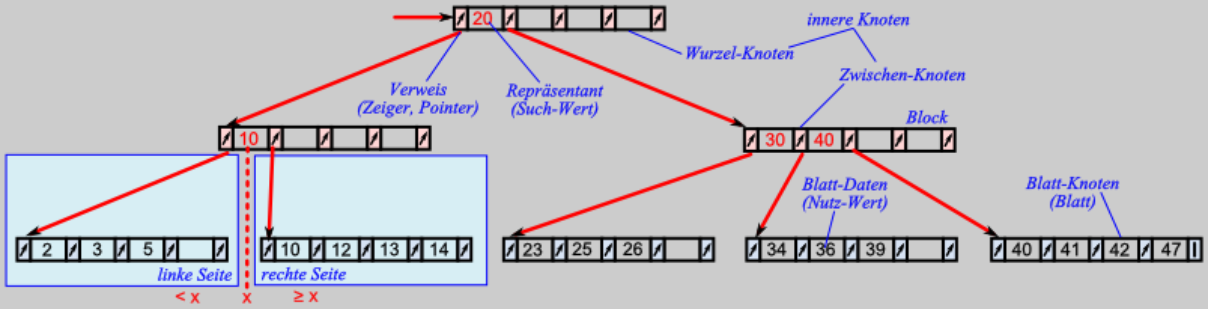
1. *Erstellen Sie den Index "PLZ-Index", wie oben vorgegeben!*
2. *Definieren Sie einen Index "Orts-Index" mit umgekehrter Sortier-Reihenfolge!*
3. *Erstellen Sie einen weiteren Index und erläutern Sie, warum dieser z.B. in einer großen Daten-Tabelle einen Sinn macht!*

Exkurs: B⁺-Bäume

B⁺-Bäume – früher auch B*-Bäume genannt – sind zuerst einmal typische Baum-Strukturen. Besonderheiten sind einmal die einheitliche Länge der Wege zu den Daten. Dabei wird auch auf eine möglichst geringe Verzweigungs-Tiefe Wert gelegt, um die Daten möglichst schnell zu erreichen. Es sind mehr als zwei Zweige möglich. Man spricht deshalb auch von einem Höhen-balancierten Mehrweg-Baum (Multiweg-Baum). Als zweite charakteristische Besonderheit gilt die ausschließliche Speicherung der Daten (Nutz-Werte) in den Blatt-Knoten (End-Blättern). Alle End-Blätter stellen zudem noch eine geordnete Liste dar. Blatt-Knoten haben somit auch keine Nachfolger mehr.
Diese interne Verketteung wird im folgenden Beispiel-B+-Baum aus Übersichtlichkeits-Gründen nicht mehr angezeigt.



Nutzdaten-Kette



Beispiel-B⁺-Baum (k=2, n=2k=4)
mit beschrifteten Teilen

Neben den Blatt-Knoten gibt es noch den Wurzel-Knoten und ev. weitere Zwischen-Knoten. Zusammen werden sie auch innere Knoten genannt. In einem kleinen B⁺-Baum kann der Wurzel-Knoten aber auch gleichzeitig das End-Blatt sein.
Wurzel- und Zwischen-Knoten enthalten Entscheidungs-Kriterien, die im Gegensatz zu den anderen Binär-Bäumen aber auch andere Werte, als solche aus der Daten-Liste enthalten können.
Ein Knoten kann mehrere Werte aufnehmen. Desweiteren sind im Knoten mehrere Zeiger (Referenzen, Pointer) auf andere Knoten-Elemente oder andere Knoten eingebaut. Man spricht bei solchen strukturierten Knoten auch gerne von einem Block.
B⁺-Bäume sind ausgesprochen dynamische Strukturen, die nach dem Lösen oder Hinzufügen von Werten immer mal wieder umgeordnet werden. Das oberste Ziel ist es dabei, die Verzeigungs-Länge immer klein und für alle Blatt-Elemente gleich lang zu gestalten.
Sie existieren nur während der Arbeitszeit eines Systems im Speicher.

allgemein gilt:

- Blatt-Knoten enthalten nur Nutz-Daten (Datenwerte), die in sich als sortierte Kette angeordnet sind
- innere Knoten (Wurzel- + Zwischen-Knoten) enthalten nur Such- / Schlüssel-Werte und zugehörige Verweise auf tieferliegende Zwischen-Knoten oder Blatt-Knoten
- der linke Verweis eines inneren Knotens zeigt auf innere Knoten oder Blätter, die kleinere Nutz-Werte repräsentieren
- der rechte Verweis eines inneren Knotens zeigt auf innere Knoten oder Blätter, die größere oder gleichgroße Nutz-Werte repräsentieren

für n=4 gilt
(n=2k)

- alle Knoten haben max. 4 Suchwerte (n) und max. 5 Verweise (n+1)
- die Wurzel hat mind. 1 Wert und 2 Verweise (Minimal-Index → ein Eintrag)
- Zwischen-Knoten haben mind. 1 Suchwert und 2 Verweise
- Blätter haben mind. 2 Datenwerte und 3 Verweise (Verkettung)

!!!Achtung, ab hier Datensammlung, z.T. auch über B-Bäume!!!

höhenbalancierter Mehrweg-Baum
sehr dynamische Struktur (nur im temporär im Arbeits-Speicher, solange mit der entsprechenden Tabelle gearbeitet wird)
durch wenige Höhenstufen ist eine minimaler Zugriffs-Aufwand garantiert
nur die Blatt-Knoten enthalten Nutzdaten und haben keinen Nachfolger
Zwischen-Knoten (innere Knoten und der Wurzel-Knoten) enthalten Hilfsdaten, um Zugriffe zu den Blatt-Knoten bzw. weiteren Zwischen-Knoten zu realisieren
Ordnung eines b-Baumes ist k
ein Baum der Ordnung 4 enthält maximal 3 Schlüsselwerte je Knoten
interne Knoten enthalten mindestens k-viele und (???) maximal 2k-viele Schlüsselwerte
Tiefe des Baumes (Baum-Höhe h) immer gleich für alle Zweige (Weg zu den Daten gleich lang)
Intervall-Grenzen werden durch die Schlüsselwerte bestimmt
es sind mehr als zwei Zweige möglich (Multiweg-Baum)

Teilbäume werden Seiten genannt

Wurzel ist entweder Blatt oder hat zwei Zweig-Knoten

Anzahl der Blätter an einem Knoten (auch Block genannt) wird durch k () begrenzt
Knoten hat neben dem Wert (Repräsentant) noch linken und rechten Zeiger
linker Zeiger eines inneren Knotens verweist auf ein Blatt mit Nutzdaten, in dem nur Nuzdaten mit kleinerem Schlüselwert enthalten sind
rechte Zeiger der inneren Knoten verweist auf ein Blatt mit einem größeren Schlüsselwert
rechte Zeiger der End-Knoten (Daten-Blätter) ergeben eine geordnete Liste der Elemente

Lesen:
Suche startet in der Wurzel; anhand von Vergleichen (kleiner des ersten Wurzel- oder Zwischen-Elementes / Zweigen oder zwischen zwei Elementen / Zweigen

Schreiben:
Suche, wie bei Lesen; im Fall eines Überlaufes wird Block gesplittet und der Index-Knoten um den Repäsentanten / Wert ergänzt

?Einfügen: (aus anderer Q)
wenn Platz vorhanden, dann an freier Stelle einfügen
wenn kein Platz (im Blatt-Knoten) vorhanden (→ Überlauf), dann wird Knoten geteilt
wenn kein Platz in Blatt und Index-Knoten, dann Aufteilen des Index-Knoten
dabei wird mittleres Index-Element in übergeordneten Index-Knoten eingetragen
falls dieser nicht existiert, dann wird neue Wurzel erzeugt

Löschen:
Suche, wie beim Lesen; im Fall eines Unterlaufes (Block mit zuwenig Elementen) werden benachbarte Knoten zusammengelegt

Anzahl der Zugriffe beim Lesen von der Höhe des Baumes bestimmt
Anzahl der Zugriffe beim Schreiben zwischen 1 und max. 2*h + 1
Anzahl der Zugriffe beim Löschen zwischen 1 und max. h + 1 bei durchlaufenden Unterlauf bis zur Wurzel

Speichereffizienz:
höherer Speicher-Bedarf, da Schlüsselwerte doppelt gespeichert werden
Füllgrad der Blätter zu 50% durch Struktur-Vorschriften gewährleistet

Links:
http://slady.net/java/bt/ (Java-Applet:B^+-Baum-Demo)

Aufgaben:

1. *Wieviele Nutz-Werte kann der im Exkurs als Beispiel angegebene Beispiel-B^+-Baum bei einer Höhe von 2 maximal aufnehmen? Erklären Sie!*

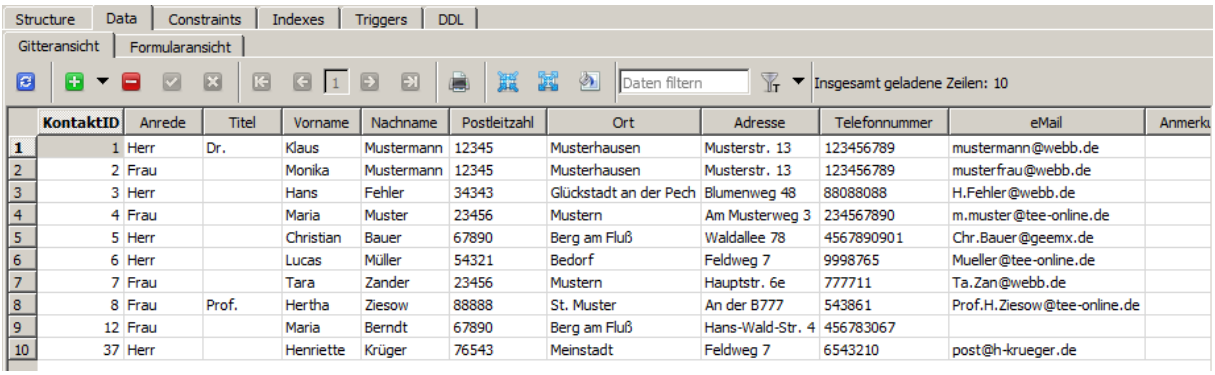
3.1.7.1.3.6. Erstellen von View's / Sichten / Abfragen

Programm-interne (Anzeige-)Möglichkeiten

betrifft nur die aktuelle Anzeige und nur den aktuellen Nutzer
in unserem Fall mach das SQLite-Studio für uns die Anzeige auf dem Bildschirm
Filter wirken nur innerhalb der Gitter-Anzeige
keine Weiternutzung in anderen Filtern etc. möglich
im Sinne von DBMS sind diese Ansichten keine echten Sichten (View's, Abfragen), weil sie wirklich nur der temporären Darstellung auf dem Bildschirm dienen
echte View's können als Defintion gespeichert werden und dann in der Datenbank für die weitere Nutzung aufgerufen werden, sie stellen praktisch Daten-Filter für die riesigen Daten-Bestände dar, die Filter können z.T. gut hintereinander kombiniert werden
die Daten können aus mehreren Quell-Tabellen stammen, werden unter bestimmten Regeln (Tabellen-Algebra) (neu) zusammengestellt und dann wiederum als (Ergebnis-)Tabelle bereitgestellt
ev. werden noch Kenngrößen, wie Mittelwerte, Summen, Anzahlen mitermittelt.
Berechnungen mit Daten aus den Datensätzen sind ebennfalls möglich (z.B. die Berechnung des Brutto-Preises aus dem gespeicherten Netto-Preis und dem gültigen Mehrwert-Steuersatz (aus einer anderen Tabelle))
sie sind temporär und nur für den aktuellen Aufruf eines View's gültig
für Nutzer ist nicht erkennbar, ob es sich um ursprüngliche oder um eine zusammengestellte Tabelle handelt
können als (eigene) Tabelle abgespeichert werden, trotzdem sind sie nur temporär und werden immer neu aus dem aktuellsten Daten-Bestand zusammengestellt
die Kenngrößen meist beim nächsten Aufruf der "gespeicherten" Tabelle neu ermittelt

bei den Programm-internen Ansichten spielt sich alles in der ursprünglichen Tabelle ab, wir sehen einfach nur einen Teil davon bzw. die Tabelle andes dargestellt

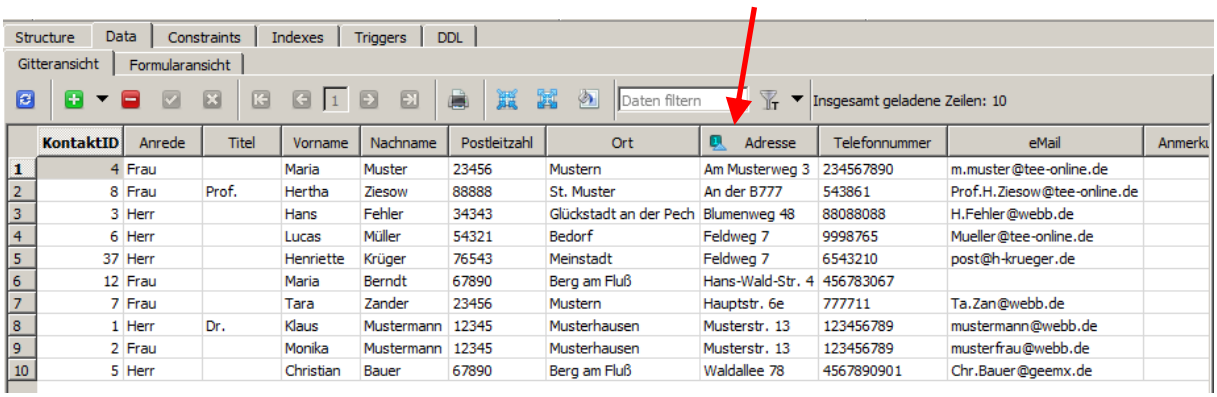
Ausgangs- / Beispiel-Tabelle für alle nachfolgenden Sortierungen usw.



	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerku
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
2	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
3	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
4	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
5	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	
6	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
7	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
8	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	
9	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
10	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	

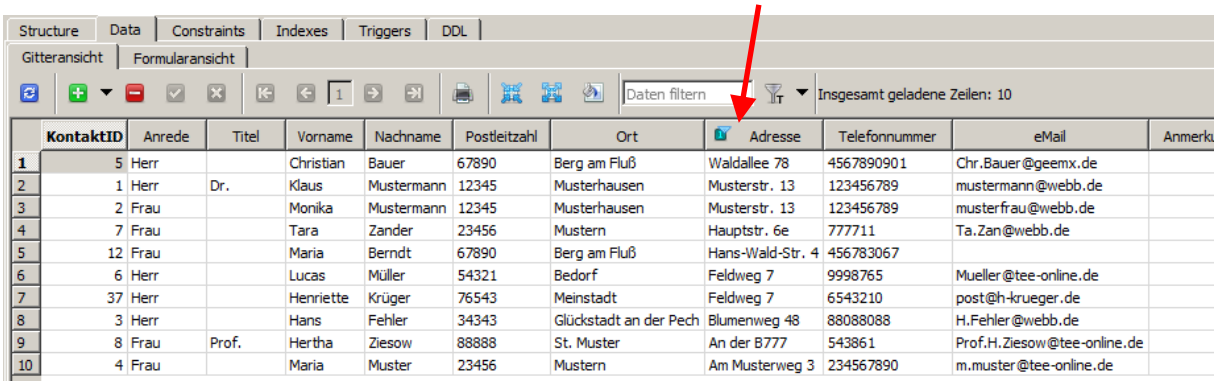
Sortierung(en)

dazu reicht ein Klick auf den Attribut-Namen (Spalten-Name)
zuerst aufsteigende Sortierung
ersichtlich am nach unten breiter werdenden grünen Dreieck in der Kopfzeile



	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerku
1	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
2	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	
3	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
4	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
5	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	
6	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
7	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
8	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
9	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
10	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	

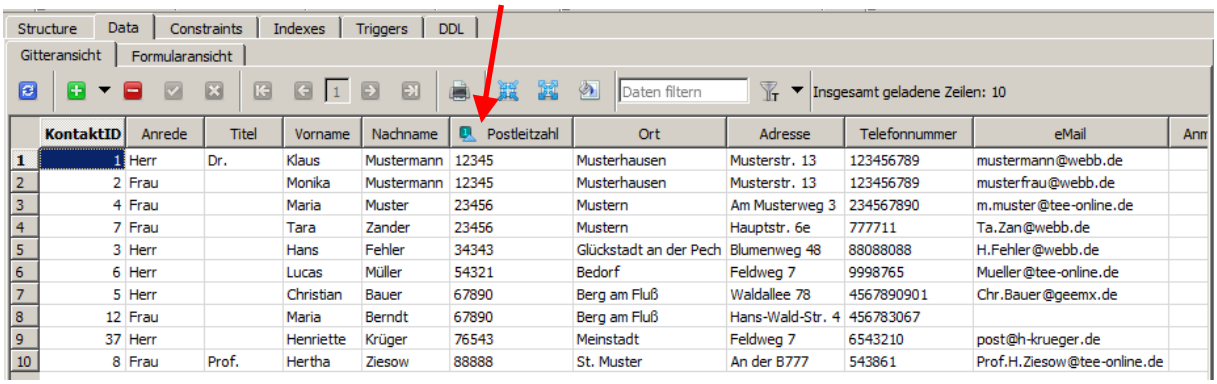
Mit einem weiteren Klick auf den Spalten-Kopf wird Sortier-Reihenfolge umgedreht.
Solche Sortierungen sind nur temporär für die Anzeige gemacht. Soll die Abfrage immer in einer sortierten Tabelle enden, dann kann mit dem SORT BY (→ 5.4.2.18. die Datensätze sortieren)



	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerku
1	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	
2	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
3	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
4	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
5	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
6	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
7	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	
8	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
9	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	
10	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	

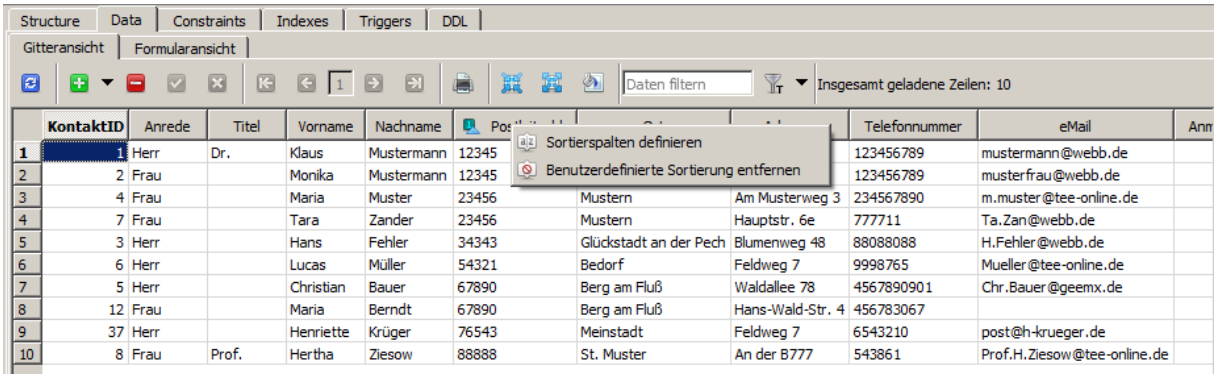
mehrfache / gestaffelte Sortierung

z.B. für eine "Adress-Tabelle" zuerst Sortierung nach Postleitzahlen



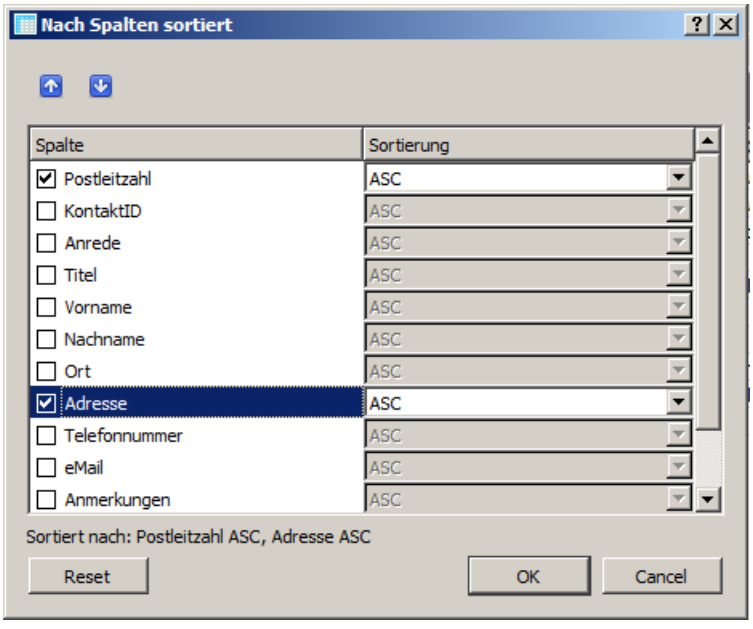
	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anm
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
2	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
3	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
4	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
5	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
6	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
7	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	
8	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
9	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	
10	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	

und dann noch nach den Straßennamen (Adressen)
über rechte Maustaste auf die Kopfzeile

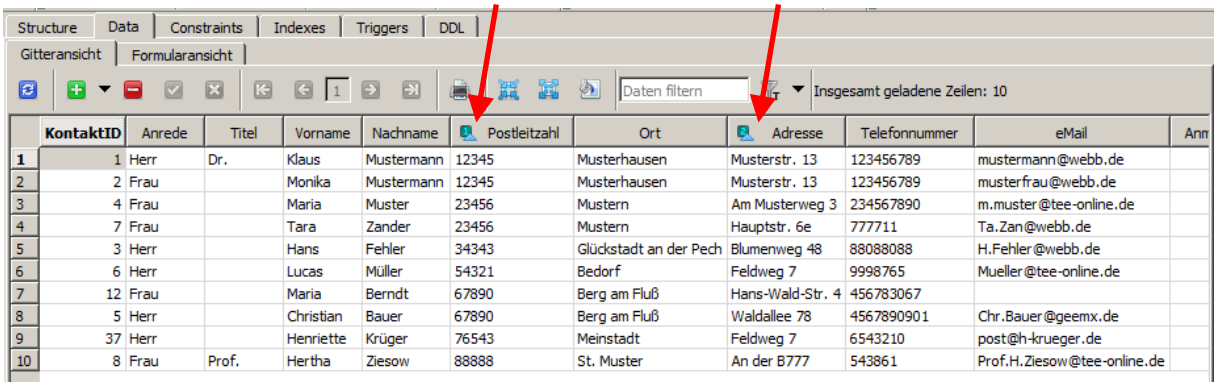


	KontaktID	Anrede	Titel	Vorname	Nachname	Pos			Telefonnummer	eMail	Anm
1	1	Herr	Dr.	Klaus	Mustermann	12345			123456789	mustermann@webb.de	
2	2	Frau		Monika	Mustermann	12345			123456789	musterfrau@webb.de	
3	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
4	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
5	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
6	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
7	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	
8	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
9	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	
10	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	

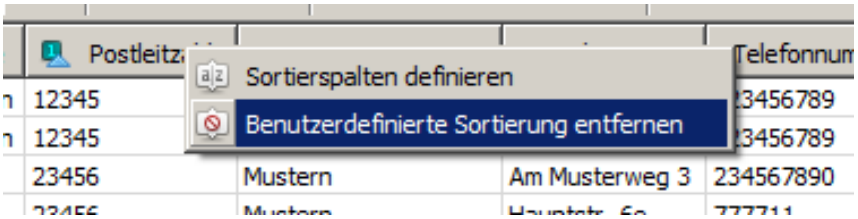
bei "Sortierspalten definieren" ruft man einen einfachen Dialog auf, in dem sowohl die Sortier-Richtung, als auch die Spalten-Reihenfolge (Hierarchie) festgelegt werden kann
Dazu wählt man die Spalte, deren Rangfolge verändert werden soll aus, und bewegt diese dann mittels der blauen Pfeile nach oben oder unten.



Sollte man mehrere Sortierungen irgendwie total verquer haben, dann kann man mit einem "Reset" wieder die Ausgangs-Situation herstellen und die Sortierungen neu organisieren.



	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anr
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
2	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
3	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
4	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
5	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
6	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
7	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
8	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	
9	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	
10	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	



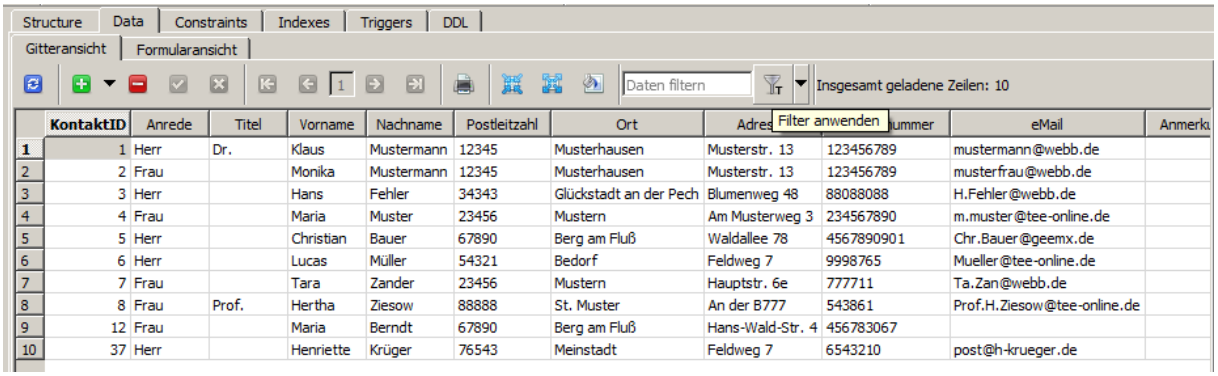
Die Sortierreihenfolge (Hierarchie) wird an den Nummern in den kleinen Sortier-Kennzeichnungs-Symbolen deutlich.
Werden die Sortierungen nicht mehr gebraucht, dann kann man diese wieder über einen Rechts-Klick in die Kopfzeile die "Benutzerdefinierte Sortierung entfernen".

Aufgaben:

1. ***Sortieren Sie Tabelle "Institutionen" alphabetisch nach den Orten!***
2. ***Wie lautet der DLL-Ausdruck für die Sortierung der Tabelle "Kontakte" zuerst nach der "Anrede" (unhöfliche Sortierung) und dann nach den Telefonnummern in umgekehrter Reihenfolge?***

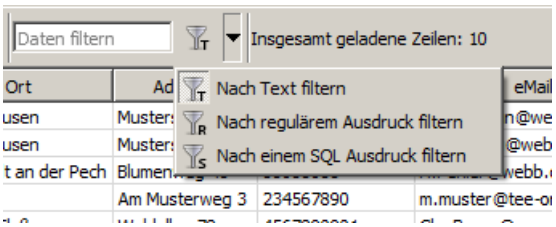
Filter-Methoden in Tabellen

Filterung dient der Auswahl der anzuzeigenden Datensätze (vor allem eben bei großen Tabellen)
gut für schnelle Datensuche; Heraussuche passender Datensätze

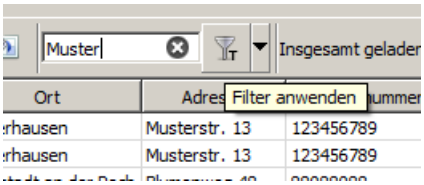


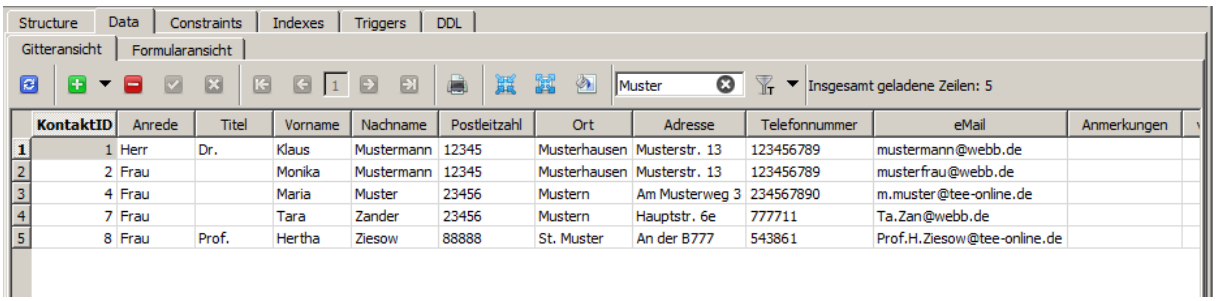
	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerkungen
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
2	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
3	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
4	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
5	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	
6	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
7	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
8	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	
9	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
10	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	

Nach Text filtern



Am Einfachten und Allgemeinsten ist die Text-Filterung. Dazu wählt man am Besten zuerst einmal die Filter-Methode aus. Beim Klick auf den Auswähler neben dem Trichter-Symbol erscheinen die drei Möglichkeiten.
Zur Textsuche ("Nach Text filtern") steht ein kleines T neben dem Trichter-Symbol.
Nun gibt man den Suchbegriff in das "Daten filtern"-Feld ein und klickt einmal auf das Trichter-Symbol.
Der Filter wird sofort ausgeführt und in den meisten fällen verkürzt sich die Tabelle.
Wie schon gesagt, bezieht sich dies nur auf die aktuelle Anzeige. An der Daten-Tabelle werden keine Veränderungen vorgenommen.



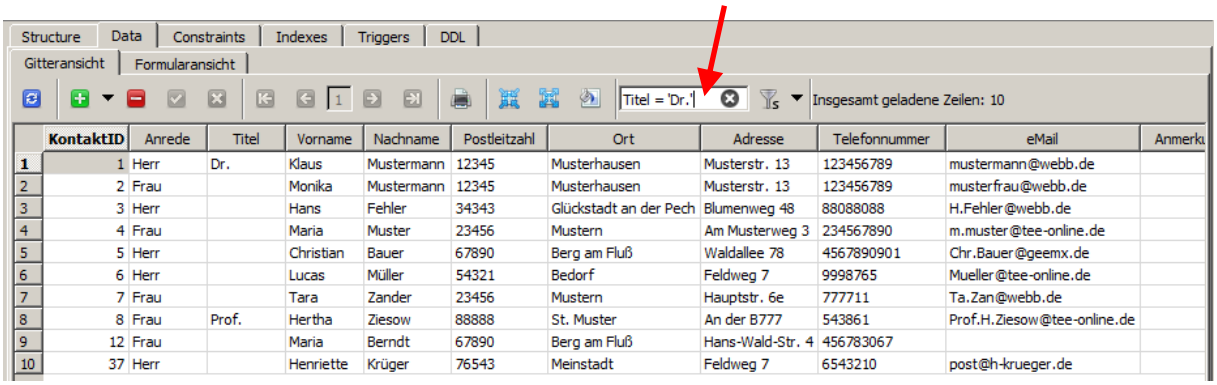


	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerkungen
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
2	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
3	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
4	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
5	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	

Das merkt man auch spätestens, wenn man die Filterung mittels des Lösch-Symbol's im Suchfeld wieder ausschaltet. Alle Datensätze sind unbeschadet wieder da.

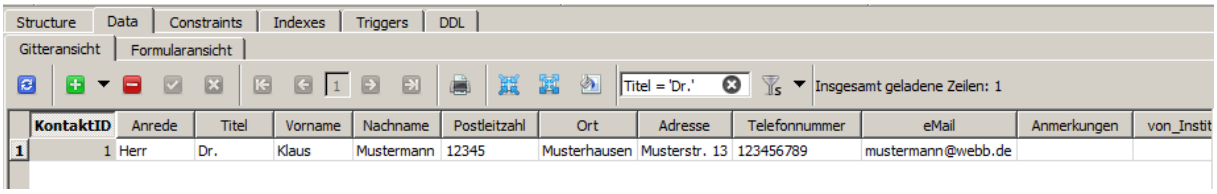
Suche mit SQL-Ausdruck

im Auswahl-Menü neben dem Trichter-Symbol wählt man "Nach einem SQL Ausdruck filtern"



	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerku
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de	
2	2	Frau		Monika	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	musterfrau@webb.de	
3	3	Herr		Hans	Fehler	34343	Glückstadt an der Pech	Blumenweg 48	88088088	H.Fehler@webb.de	
4	4	Frau		Maria	Muster	23456	Mustern	Am Musterweg 3	234567890	m.muster@tee-online.de	
5	5	Herr		Christian	Bauer	67890	Berg am Fluß	Waldallee 78	4567890901	Chr.Bauer@geemx.de	
6	6	Herr		Lucas	Müller	54321	Bedorf	Feldweg 7	9998765	Mueller@tee-online.de	
7	7	Frau		Tara	Zander	23456	Mustern	Hauptstr. 6e	777711	Ta.Zan@webb.de	
8	8	Frau	Prof.	Hertha	Ziesow	88888	St. Muster	An der B777	543861	Prof.H.Ziesow@tee-online.de	
9	12	Frau		Maria	Berndt	67890	Berg am Fluß	Hans-Wald-Str. 4	456783067		
10	37	Herr		Henriette	Krüger	76543	Meinstadt	Feldweg 7	6543210	post@h-krueger.de	

Hinter dem Kurz-Aufruf steckt die nebenstehende SQL-Anweisung, die wir im SQLite-Studio so allerdings nicht zu Gesicht bekommen. Wer sich den Text ansieht, wird schnell erkennen, wie eine Auswahl von Daten erfolgen kann.



	KontaktID	Anrede	Titel	Vorname	Nachname	Postleitzahl	Ort	Adresse	Telefonnummer	eMail	Anmerkungen	von_Instit
1	1	Herr	Dr.	Klaus	Mustermann	12345	Musterhausen	Musterstr. 13	123456789	mustermann@webb.de		

Aufgaben:

1. *Lassen Sie nur die Datensätze anzeigen, in denen der Nachname "Fehler" enthalten ist!*
2. *Gesucht sind die nach eMail sortierten Datensätze, die den Nachnamen "Mustermann" enthalten!*

Aufgaben zu Übungs-Datenbanken:

Geo-Datenbank ""

1. !

Geo-Datenbank ""

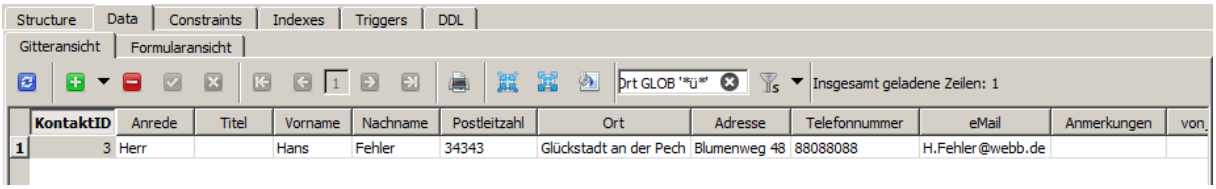
1. !

Suche mittels regulären Ausdrücken

reguläre Ausdrücke sind spezielle Formulierungen von Ausdrücken
man kann sie sich als Schablonen / Raster / Filter vorstellen, die nur bestimmte Inhalte zulassen / durchlassen
Beschreibung von zulässigen Worten (Menge von Zeichenketten, die einem vorgegebenem Syntax entsprechen / für die vorgegebene (reguläre) Grammatik- / Bildungs-Regeln gelten)
siehe auch "Theoretische Informatik → Sprachen und Grammatiken"; Sprache vom Typ 3 nach CHOMSKY
z.B. Suche mit Wildcards (Joker-Zeichen)
mehr was für Informatik-Greeks (im Skript 🕮 **Sprachen und Automaten** gehen wir auf solche grammatikalischen Strukturen genauer ein → KLEENE-Stern)

SQL-Suchanfrage nach allen Einträgen in der Spalte "spaltenname" mit einem "ü" würde lauten:

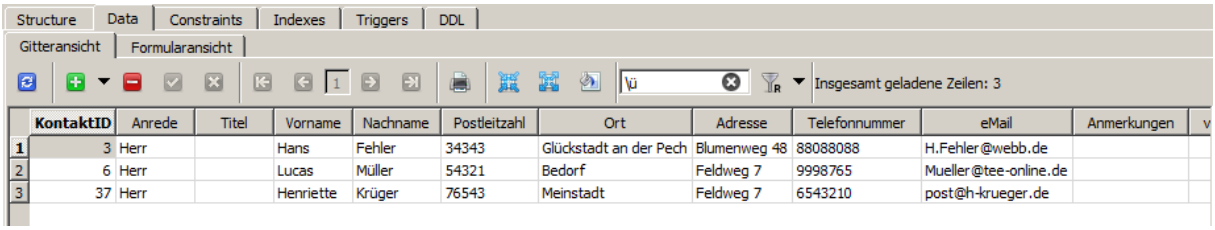
spaltenname GLOB '*ü*'



für alle "ü"s in der Daten-Tabelle "Kontakte":

als regulärer Ausdruck ist der Sucheintrag:

`\ü`



anderes Beispiel:
es sollen die Datensätze angezeigt werden, die nur bestimmte alphanumerische Zeichen enthalten

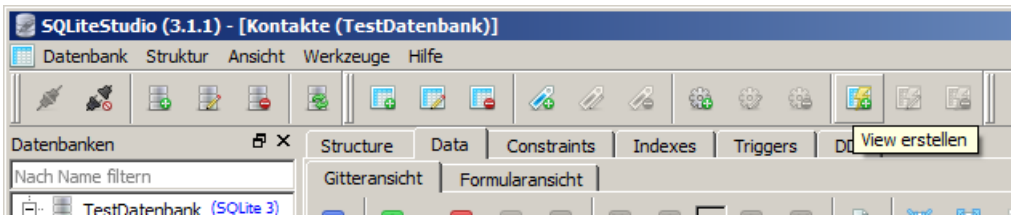
ltrim(lower(***spaltenname***), 'abcdefghijklmnopqrstuvwxyzäüö') = ''

als reguläre Expression:
'[a-zA-Z]*'

(???)

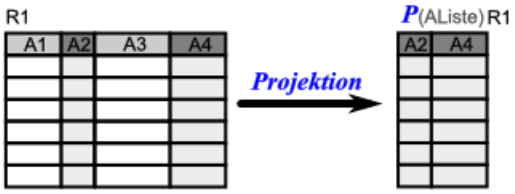
echte Abfragen / Views / Sichten (auf Datenbank-Ebene)

Die letzten beiden Ansichten des vorlaufenden Kapitels gehen schon stark in die Richtung, die wir uns jetzt ansehen wollen. Allerdings passierten die meisten Dinge (Sortierungen / Filterungen) nur innerhalb von SQLite-Studio. Wir brauchen aber auch Abfragen, die "exportierbare" Daten für andere Programme bereitstellen.
Viele Daten müssen und sollen für den Nutzer selektiv zur Verfügung stehen. Z.B. darf nicht jeder Nutzer einer Personen-Datenbank alle persönlichen Daten sehen und das ist immer ein deutliches Beispiel: Nicht jeder darf das Gehalt einer anderen Person auslesen dürfen.
Häufig ist es auch so, dass Daten-Tabellen Unmengen von Datensätzen enthalten, die z.Z. oder überhaupt nicht mehr relevant für die Arbeit sind. Die Datensätze dürfen oft erst nach Jahren gelöscht werden, weil immer noch irgendwelche Auskunfts- oder Prüfungs-Rechte oder –Pflichten bestehen (z.B. Datenschutz-Regeln, Behörden, Aufsichtsräte, …).
Durch fertige Filter sorgt man dafür, dass die Daten, die für den nächsten Arbeitsschritt gebraucht werden, möglichst gering gehalten werden. Diese Filter werden im Allgemeinen Views, Abfragen oder Sichten genannt. Man kann sie sich immer wie rosa Brillen vorstellen. Man sieht dann alles nur noch rosa. Die Farben wurden stark reduziert.
Mit "echten" View's / Abfragen können wir auch mehrere Tabellen verknüpfend bearbeiten. Man sucht z.B. die Datensätze einer Tabelle, die bestimmte Werte enthalten, wobei diese Objekte dann auch in einer anderen Tabelle mit einem weiteren Kriterium übereinstimmen müssen. So etwas geht bei den Programm-internen Anzeige-Abfragen eben nicht. Dort konnten wir nur eine einzelne Tabelle filtern. In der Tabellen-Algebra (Relationen-Algebra → 4.0. Relations-Algebra / Relations-Kalküle) wird das Verbinden von Tabellen JOIN genannt.
In modernen Views sind zudem noch Auswertungs-Möglichkeiten vorhanden. Früher war es oft so, dass diese Sichtweise auf die Datenbank als Bericht bezeichnet wurde.
Views lassen sich speichern und in / mit anderen Views / Abfragen / Sichten verknüpfen. I.A. sind die angelegten Views auch für andere Anwendungen / Nutzer zugänglich, wenn sie dann die notwendigen Rechte haben (Nicht jeder Angestellte darf überhaupt irgendwelche Personendaten sehen.). Die neuen View's werden aus den aktuellen Daten-Beständen zusammengestellt und als Tabelle zur Verfügung gestellt.
Eine weitere Nutzungs-Variante ist die Schaffung von Nutzer-spezifischen temporären Daten-Beständen, die nur in die Richtung von der Datenbank zum Nutzer funktionieren. Der Nutzer kann mit den Daten machen / probieren, was er will. Ein Zurückschreiben von Änderungen oder gar zerstörten Daten ist nicht möglich.
Zumindestens im SQLiteStudio benötigen wir für die Erstellung von Views auch einige Grundkenntnisse über die Formulierungs-Sprache SQL. Da hilft es uns schon, dass wir uns die SQL-Äquivalente der anderen – vorne besprochenen - Tätigkeiten auch mal angesehen haben. Da SQL ziemlich dicht an einem sehr abstrakten, aber einfachen, Englisch liegt, kommen wir schnell hinter die grundlegenden Prinzipien.



einfache Abfragen (Projektion(en))

In einfachen Abfragen werden nur bestimmte Spalten (Attribute) einer Tabelle ausgewählt. Diese Art der Abfrage nennt man **Projektion**. Sie gehört zu einem der Grund-Funktionen der Relationen-Algebra (Tabellen-Algebra).
Durch die Einschränkung der Felder werden die ursprünglichen Tabellen nun deutlich schlanker und übersichtlicher.

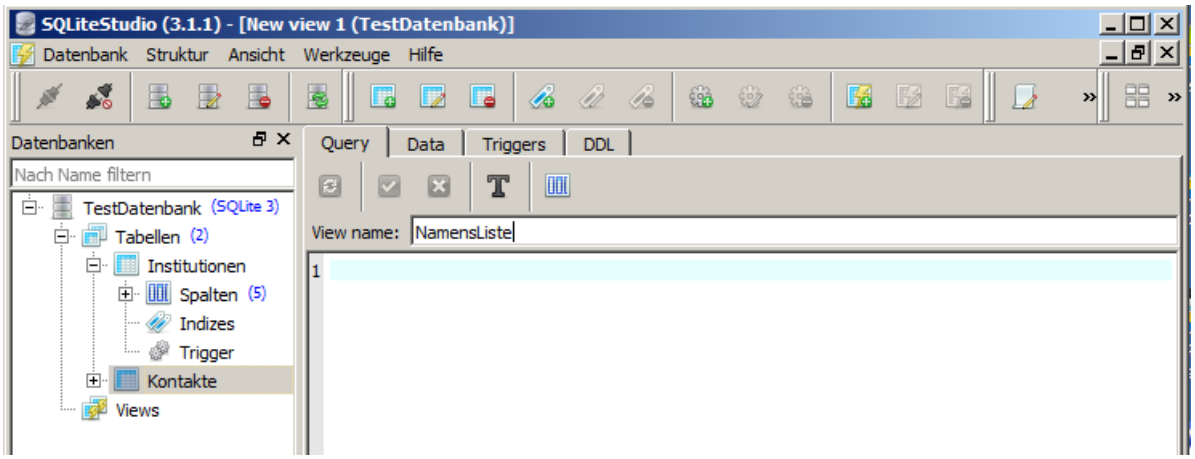


SQL-Repräsentation

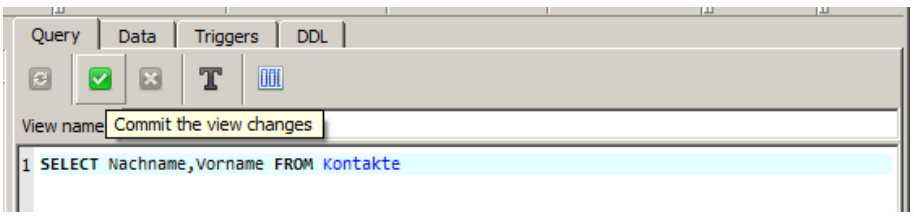
Für die tieferschürfende Arbeit mit Datenbanken kommen wir nicht um die Besprechung der SQL-Statement's herum. Eine ausführliche Darstellung der SQL-Sprach-Elemente gibt es im Kapitel **(→** 5.1.11.1. Projektion (Abbildung):**)**. Hier jetzt nur die einfachen Befehle bzw. Syntax-Ausdrücke:

SELECT *spalteX* **{ ,** *spalteY* **}** **FROM** *tabelle*;

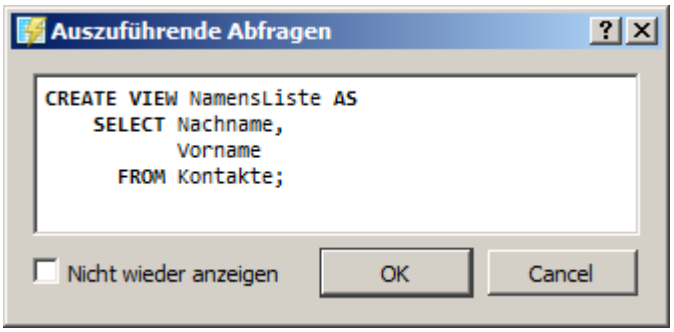
Die geschweiften Klammern (**{ }**) stehen für zusätzliche mögliche Objekte – meist im Sinne der erweiterten Wiederholung des umschlossenen Ausdruckes. Hier also der Aufzählung weiterer Spalten / Attribute. Wir stellen die Syntax-Hilfsstrukturen wie Alternativen, Optionen und Wiederholungen in roter Farbe dar, damit deren Bedeutung deutlicher wird. Die "Befehls-Wörter" der Sprache sind fett hervorgehoben. Die variablen Elemente – also bestimmte Platzhalter z.B. für Spalten- und Tabellen-Namen - sind kursiv dargestellt.

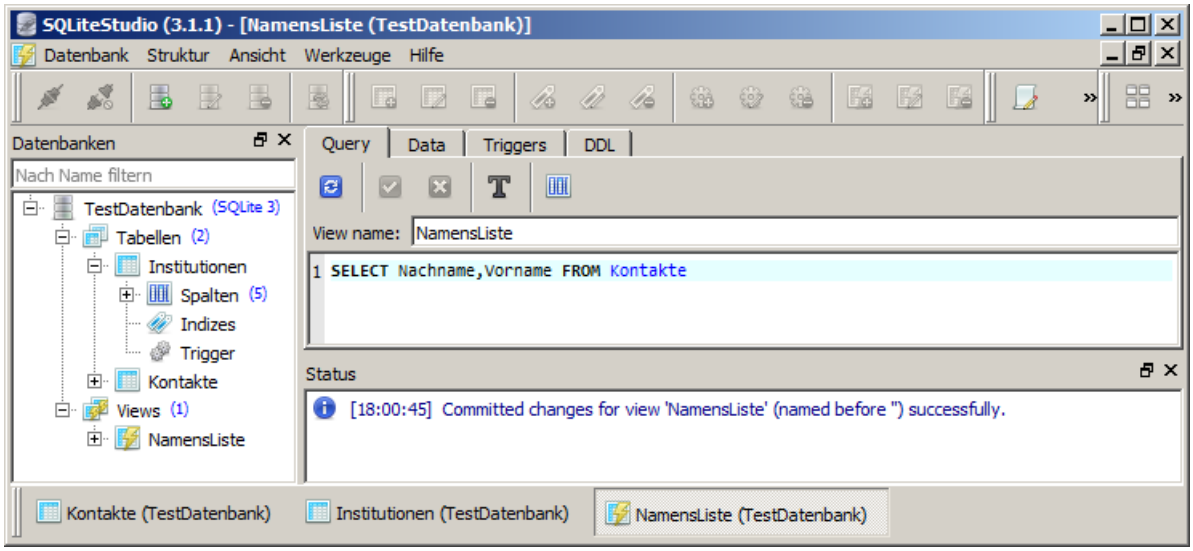


Im SQLite-Studio wird ein einfaches Syntax-Highligthning benutzt.

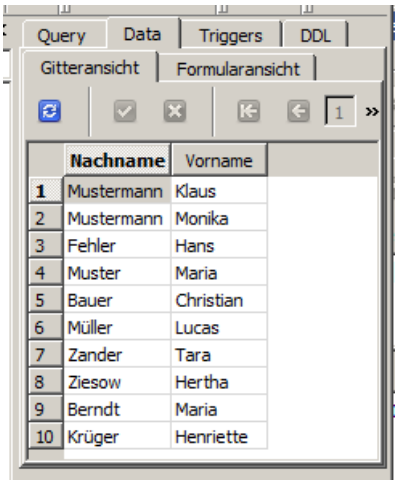


vor der eigentlichen Ausführung in SQLite bekommen wir den SQL-Ausdruck noch einmal – ev. etwas umformatiert – angezeigt



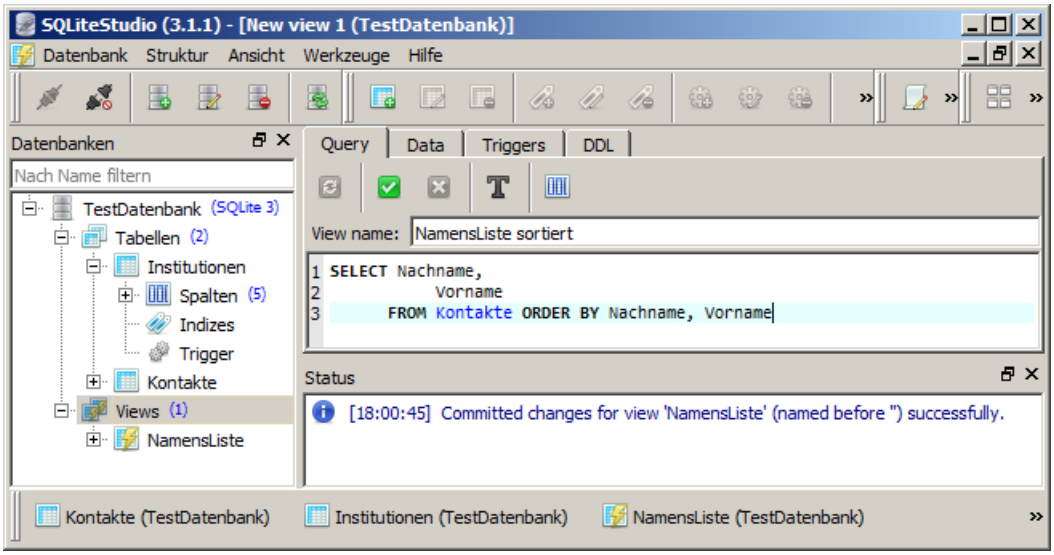


Unter dem Reiter "DDL" kann man sich auch später die zugrundeliegende SQL-Anweisung ansehen. Das ist ev. hilfreich, wenn man ähnliche Aufgaben erledigen soll und nur einzelne Sachverhalte – wie z.B. eine zusätzliche Sortierung – hinzufügen möchte.
Die SQL-Anweisung lässt sich Kopieren und z.B. in einem neuen View wieder Einfügen.
Für die Definition des View's brauchen wir nur den Teil ab SELECT und bis vor dem Semikolon. (Ein abschließendes Semikolon wird aber durch SQLiteStudio ignoriert.)
An die kopierte SQL-Teilanweisung fügen wir dann z.B. eine Sortierung hinzu.

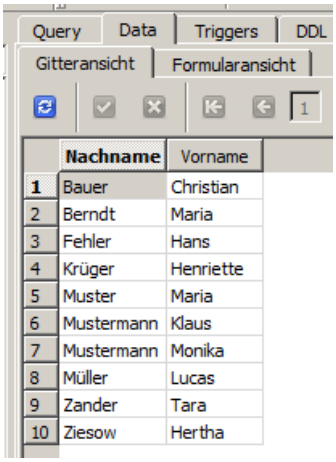


SQL-Repräsentation (→)

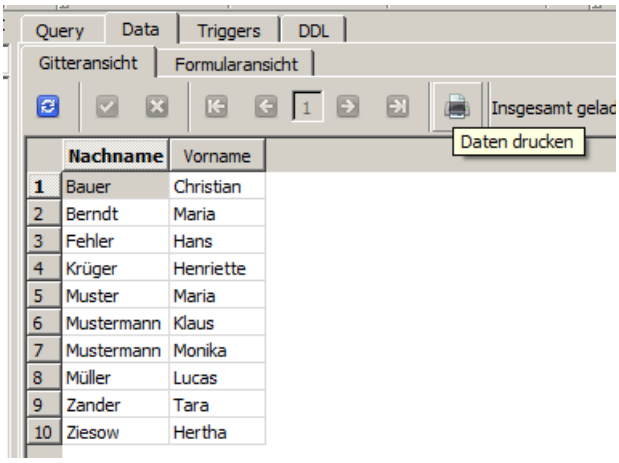
SELECT *spalteX* **{ ,** *spalteY* **}** **FROM** *tabelle* **ORDER BY** *spalteX* **{ ,** *spalteY* **};**



wie die Sortierung erfolgt, zeigen wir gleich



Irgendwann wird nun auch die praktische Nutzung als Papier-Ausdruck usw. interessant. Über das Drucker-Symbol in der Gitteransicht erhält man nach dem üblichen Drucker-Dialog den passenden Druck. Es entsteht ein einfacher Ausdruck der reinen Ansichten-Tabellen.
Leider scheint es derzeit noch keinen Export der Daten aus einem View zu geben.
Eine Ausnahme sind PDF-Dateien, wenn man einen PDF-Drucker installiert hat. Da es freie Versionen im Internet gibt, steht praktisch jedem diese Möglichkeit zur Verfügung.

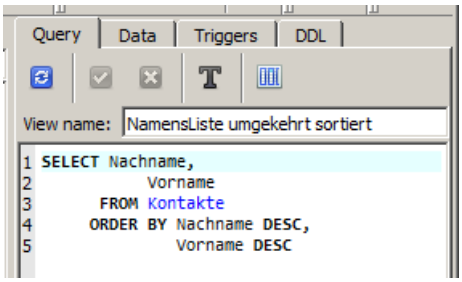


Nicht vollständig definierte View's sind manchmal intern noch präsent. Das merkt man dann, wenn auf einmal im DDL-Text zusätzlich eine DROP-Anweisung erscheint.

erweiterte Abfragen mit Sortierungen usw. usf. (Selektion(en))

Eine erste gestalterische Möglichkeit – die klassische Sortierung – haben wir eben schon bei der Erstellung eines einfachen View's besprochen. Das ORDER BY ist wohl auch nicht wirklich schwer zu verstehen. Etwas anders sehe ich hier die umgedrehte oder kombinierte Sortierung. Jetzt werden die SQL-Teil-Anweisungen – also die SELECT-Konstrukte schon komplexer und auch das eine oder andere Mal undurchsichtiger.

Aber fangen wir erst einmal mit einer umgedrehten Sortierung an. Praktisch soll die vollständig anders sortierte Liste von dem Beispiel aus dem letzten Abschnitt entstehen.



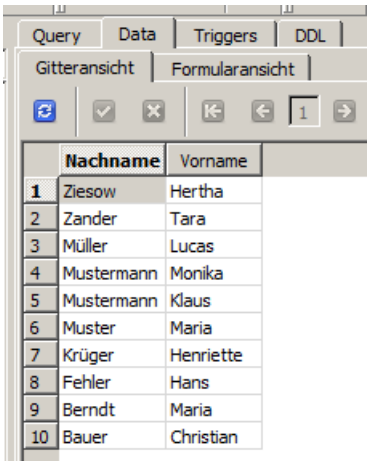
Zuerst verwenden wir den gleichen SELECT-Ausdruck, wie oben. Die umgedrehte Sortierung erreicht man mittels DESC (descending) hinter dem Spaltennamen. Bei der aufsteigenden Sortierung würde eigentlich dort ASC (ascending) stehen. Das wird aber als Standard vorausgesetzt und kann deshalb entfallen.

SQL-Repräsentation (→)

SELECT *spalteX* **{ ,** *spalteY* **}** **FROM** *tabelle*
ORDER BY *spalteX* **[ASC] | DESC] { ,** *spalteY* **[ASC] | DESC] } ;**

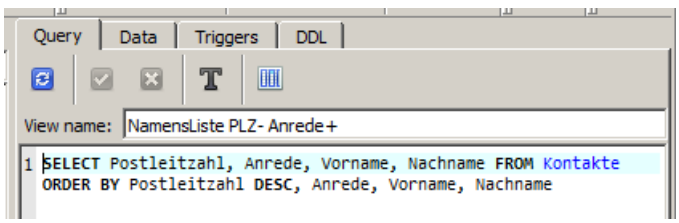
Die eckige Klammer (**[ ]**) steht optionale Bestandteile im Ausdruck. Sie können u.U. weg gelassen werden. Der senkrechte Strich (**|**) kennzeichnet Alternativen. Eine kann / muss dann ausgewählt bzw. benutzt werden.

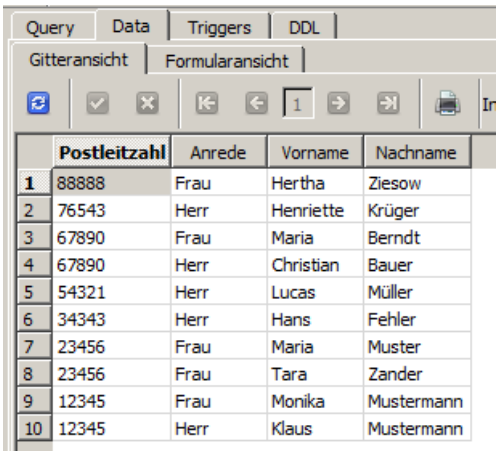
die fertige Liste ist nicht überraschend.



Im nächsten Beispiel wollen wir die Sortier-Richtungen kombinieren. Dazu gehen wir auf unseren originalen Daten-Bestand der "Kontakte"-Tabelle zurück.

Gesucht ist nun eine Ansicht (Abfrage-Tabelle), in der die Postleitzahlen aus irgendeinem Grund rückwärts sortiert sein sollen und dann die Frauen und Männer schön und höflich (Frauen zuerst) angezeigt werden sollen.



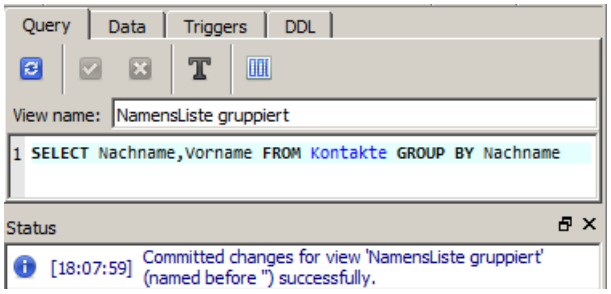


	Postleitzahl	Anrede	Vorname	Nachname
1	88888	Frau	Hertha	Ziesow
2	76543	Herr	Henriette	Krüger
3	67890	Frau	Maria	Berndt
4	67890	Herr	Christian	Bauer
5	54321	Herr	Lucas	Müller
6	34343	Herr	Hans	Fehler
7	23456	Frau	Maria	Muster
8	23456	Frau	Tara	Zander
9	12345	Frau	Monika	Mustermann
10	12345	Herr	Klaus	Mustermann

Aufgaben:

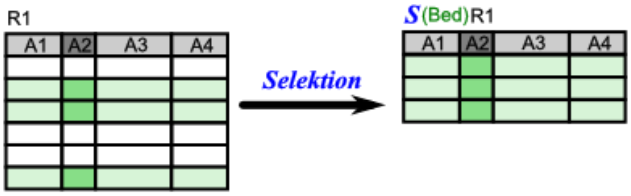
1. *Sortieren Sie die "Kontakte" in einer neuen Sicht "NamensListe umgekehrt höflich" umgekehrt nach den "Nachnamen" und dann nach "Anrede" in der höflichen Form!*
2. *Denken Sie sich für Ihren Nachbarn eine dreistufige Sortierung aus! Tauschen Sie die Arbeits-Aufträge aus und erledigen Sie diese!*
3. *Drucken Sie Ergebnisse einmal aus und übergeben Sie diese zur Kontrolle an den Nachbarn! Dieser vergleicht den Ausdruck mit seinem Arbeits-Auftrag!*

Gruppierungen



Bei einer größeren Anzahl von Datensätzen wünscht man sich oft ein zusätzliches Ordnungs-System. Das könnten z.B. Gruppierungen sein.
Im SQLiteStudio kann man sich zwar solche View's definieren – eine gruppierte Anzeige ist aber derzeit nicht möglich. Auch im Ausdruck bekommt man nur die übliche Tabelle.
Andere Programme bieten aber entsprechende Möglichkeiten.

auch als Auswahl (der Datensätze) verstanden



… WHERE *bedingung*

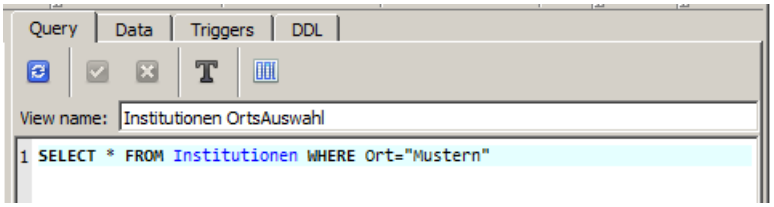
Ergebnis beinhaltet alle Tupel (also den vollständigen Datensatz), bei dem die Bedingung erfüllt ist
Wir sprechen dann von einer **Selektion**.

SQL-Repräsentation (→ 5.1.11.2. Selektion (Auswahl):):

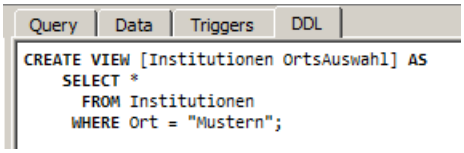
SELECT * FROM *tabelle* **WHERE *bedingung*;**
bedingung kann dabei ein einfacher oder komplexer – z.B. zusammengesetzter – (SQL-)Ausdruck sein

bedingung muss als Ausdruck immer ein WAHR oder FALSCH (TRUE / FALSE) ergeben.

Die Möglichkeiten für die Formulierung der Bedingungen sind sehr komplex und umfangreich. Sie müssen ja für eine perfekte Daten-Auswahl auch sehr Leistungs-fähig sein. Aus praktischen Gründen besprechen wir hier nur einige ausgewählte Möglichkeiten. Wenn man das Prinzip verstanden hat, dann ist das "Rumspielen" mit speziellen Funktionen, Optionen und Erweiterungen nicht mehr so ein großes Problem. Eine etwas umfangreichere Darstellung der Möglichkeiten von SQL findet der Leser weiter hinten (→ 5.1.11. SELECT … FROM). Ansonsten ist hier der Zugriff auf die SQLite-Beschreibung / -Dokumentation zu empfehlen. Man kann diese direkt aus dem SQLiteStudio über "Hilfe" und "SQLite Dokumentation" erreichen. Dabei gelangt man zur offiziellen Dokumentations-Seite von SQLite (→http://sqlite.org/lang.html)
Starten wir hier mit einer einfachen Selektionen.
Gesucht seien die Datensätze aus der Tabelle "Institutionen", die den Ort Meinstadt beeinhalten.
Hinter dem WHERE – was man der Einfachheit halber mit einem **wenn** übersetzen kann – folgt der Bedingungs-Teil.

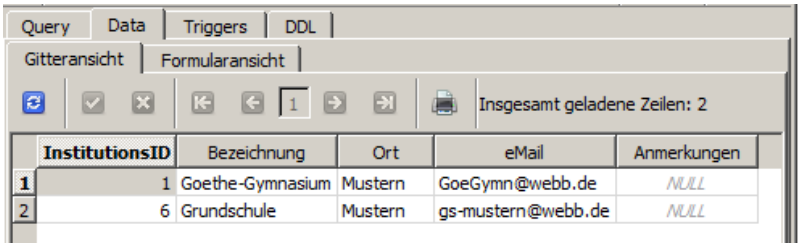


Als Beispiel soll der Ort einem vorgegebenen Text entsprechen. Das wird durch Gleichsetzung von Spaltenname und Suchtext erreicht.
Unter dem Reiter "DLL" oder vor dem Ausführen über das grüne Häkchen können wir uns den vollständigen SQL-Befehl ansehen.



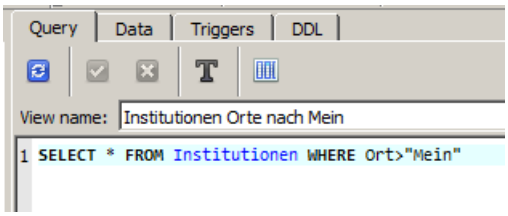
In den folgenden Beispielen werden wir darauf öfter verzichten, da die wesentlichen Teile schon in der Abfrage-Beschreibung ("Query") zu sehen ist.

Das Ergebnis ist eine auf zwei Zeilen reduzierte "Institutionen"-Tabelle.

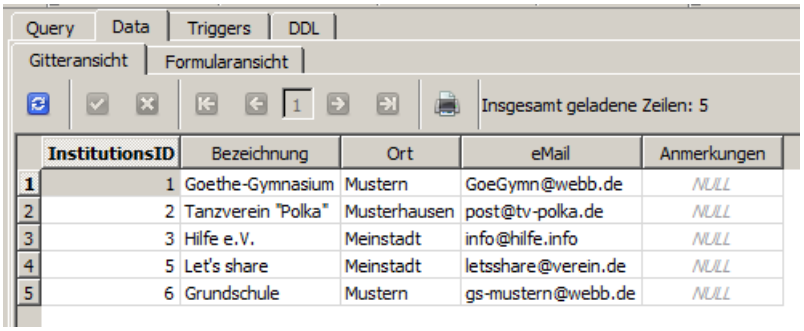


	InstitutionsID	Bezeichnung	Ort	eMail	Anmerkungen
1	1	Goethe-Gymnasium	Mustern	GoeGymn@webb.de	NULL
2	6	Grundschule	Mustern	gs-mustern@webb.de	NULL

Ähnliches hatten wir ja auch schon durch eine Programm-interne Filterung erreicht (→ Filterung mittels "Text"-Muster / Such-Text).
Das Wert-Suchen in Zahlen-Feldern unterscheidet sich nicht von der Text-Suche.
Da liegt die Vermutung nicht fern, dass man auch mittels "kleiner …", "kleiner-gleich …", "größer-gleich …" und "größer als …" abfragen kann. All diese Bedingungs-Operatoren sind natürlich wirklich möglich.
Bei Texten wird die lexikalische Reihenfolge verwendet.
Suchen wir also alle Datensätze mit Orten, die alphabetisch nach "Mein" folgen, dann formulieren wir das mittels **Ort > "Mein"**.
Es sind auch Teil-Texte oder Einzelzeichen zulässig.
Nach dem Klick auf das grüne Häkchen bekommen wir die – gleich unten abgebildete – Tabelle.



Aber – ist hier nicht ein Fehler passiert? Meinstadt taucht auch in der Abfrage-Tabelle auf.
Ursache für dieses "Missverständnis" ist die Reihenfolge von Texten / Worten wie in einem Lexikon. Die kurzen Wörter kommen zuerst. Somit steht "Meinstadt" hinter "Mein".



	InstitutionsID	Bezeichnung	Ort	eMail	Anmerkungen
1	1	Goethe-Gymnasium	Mustern	GoeGymn@webb.de	NULL
2	2	Tanzverein "Polka"	Musterhausen	post@tv-polka.de	NULL
3	3	Hilfe e.V.	Meinstadt	info@hilfe.info	NULL
4	5	Let's share	Meinstadt	letsshare@verein.de	NULL
5	6	Grundschule	Mustern	gs-mustern@webb.de	NULL

Das SQL-System hat also völlig korrekt gearbeitet. Ev. muss man als Anfragender seine Abfrage weiter präzisieren.
Will man nun nach Text-Abschnitten suchen, die innerhalb eines Wortes liegen, müssen wir auf Joker-Zeichen zurückgreifen. Man weiss ja nicht genau, wieviele und welche Zeichen vor oder hinter dem Such-Textabschnitt liegen.
Eine beliebig lange und auch frei zusammengesetzte Zeichenkette wird mit dem Prozent-Zeichen (%) codiert. Eine solche Zeichenkette darf auch leer sein. Will man die Anzahl der Zeichen spezifizieren, dann wird das Joker-Zeichen Unterstrich (_) für immer genau ein zeichen benutzt.
Einige kleine Beispiele zur Veranschaulichung.
Man erkennt schnell, dass die Joker-Zeichen sehr mächtig sind und viele Anwendungsfälle abdecken.
Die Joker-Zeichen sind z.B. für die Suche bei deutschen Namens-Variationen,

`%abc`
beschreibt alle Texte, die mit "abc" enden; dazu gehört auch "abc"

`_dorf`
entspricht z.B. "Adorf", aber nicht "Zweidorf"

`%geld%`
meint alle Texte, die irgendwo "geld" enthalten; "geld" kann auch am Anfang oder am Ende stehen

wie Meier, Maier, Meyer und Mayer hilfreich. Wer weiss schon immer exakt, wie wer genau geschrieben wurde. Eine Suche ohne die Joker-Zeichen würde immer nur einen Namen finden.

Selektion aus einem View

z.B. Suche nach "@verein" in den eMail-Adressen nur der Institutionen, die in Orten nach "Mein" kommen

Benutzung von auswertenden / berichtenden Funktionen

sum(spaltenname), max(spaltenname), min(spaltenname), … im SELECT-Teil

???trim(???), … im WHERE -Teil

statt des Feldnamens (Attributes) werden Formeln mit den Feldnamen (z.B. "Nettopreis") ohne führendes Gleichheitszeichen eingegeben (z.B. zur Berechnung des Bruttopreises: "Nettopreis" * 1,19)
es muss dann ein Alias vergeben werden, da sonst die berechnete Spalte keine Überschrift hätte, sie ist ja immer neu

Zusammenstellung ausgewählter / häufig verwendeter Operatoren usw. für SQLite

Bedingungs-Operatoren in Abfragen

Operator / Symbol(e)	Benennung	Bemerkungen / Bedeutung
=	(ist) gleich	geprüft wir exakte Identität
<>	(ist) ungleich	alle anderen Werte werden akzeptiert
<	(ist) kleiner (als)	alle Werte, die kleiner als der angegebene sind, werden akzeptiert
<=	(ist) kleiner oder gleich	erfüllt, wenn der Wert kleiner als der angegebene oder gleich groß, wie dieser, ist
>	(ist) größer (als)	alle Werte, die größer als der angegebene sind, werden akzeptiert
>=	(ist) größer oder gleich	erfüllt, wenn der Wert größer als der angegebene oder gleich groß, wie dieser, ist

logische Operatoren in Abfragen

Operator / Symbol(e)	Benennung	Bemerkungen / Bedeutung
AND	logisches UND	
OR	logisches ODER	
NOT	logisches NICHT Negation	

Platzhalter / Joker-Symbole / Wildcards

Zeichen / Symbol	Bedeutung	
%	entspricht einer beliebigen (auch leeren) Zeichenkette bzw. eines beliebigen Wertes	
_	steht für (genau) ein Zeichen in einer Zeichenkette	

ev. noch Spalten in den folgenden Tabellen / Übersichten tauschen / ersetzen

Rechen-Operatoren in Abfragen

Operator / Symbol(e)	Benennung	Bemerkungen / Bedeutung
+	Addition	
-	Subtraktion	
*****	Multiplikation	
/	Division	

Funktionen in Abfragen

Option / Funktion	Auswirkung / Bedeutung	SQL-Schlüssel-wort	Bemerkungen
ohne	*keine*		
Durchschnitt	berechnet das (arithmetrische) Mittel (Mittelwert) des Feldes (Attributes)	AVG	
Anzahl	zählt die Datensätze, die in der Abfrage auftauchen	COUNT	COUNT(*) .. es werden alle Datensätze (in der Abfrage) gezählt COUNT(*Spalte*) .. es werden die Datensätze gezählt, bei denen die ***Spalte*** einen Nicht-NULL-Wert enthält
Maximum	ermittelt den größten Wert des Feldes (Attributes)	MAX	
Minimum	ermittelt den kleinsten Wert des Feldes (Attributes)	MIN	
Summe	berechnet die Summe der Feld-Werte (Attribut-Werte)	SUM	
Gruppiert	gruppiert die Datensätze nach den Werten im ausgewählten Feld (Attribut)	GROUP BY	

Filter-Bedingungen für Abfragen

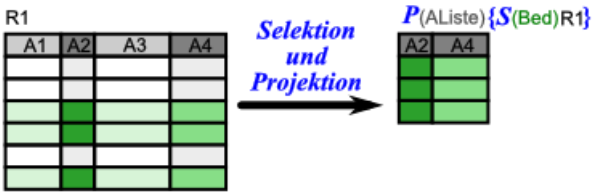
Option	Auswirkung / Bedeutung	SQL-Schlüssel-wort	Bemerkungen Beispiel(e)
ohne	*keine*		
IST LEER	ist erfüllt, wenn das Feld einen NULL-Wert besitzt	IS NULL IS LEER	bei Options-Feldern (JA / NEIN) wird der unbestimmte Fall genutzt (also weder Ja noch Nein)
IST NICHT LEER	Negation von "IST LEER"; ist erfüllt, wenn das Feld einen (von NULL abweichenden) Wert besitzt	IS NOT NULL IS NOT LEER	
LIKE **WIE**	Übereinstimmung; ist erfüllt, wenn das Feld den entsprechenden Wert besitzt	LIKE	WIE 'Muster*' oder WIE 'Muster????' (liefert: z.B. "Mustermann" und "Musterfrau" zurück)
ZWISCHEN *min* **UND** *max*	im Intervall; ist erfüllt, wenn der Feldinhalt zwischen den Angaben ***min*** und ***max*** liegt	BETWEEN *min* AND *max*	
IN (*Liste***)**	Entsprechung; Enthaltung ist erfüllt, wenn das Feld mit einem Wert aus der (Semikolon-getrennten) übereinstimmt		IN (3; 6; 12; 24) IN ('Moskau'; 'Berlin'; 'New York')
NICHT …	Negation; negiert den folgenden	NOT …	

	Ausdruck		
= WAHR	Validierung; ist erfüllt, wenn der Feldinhalt ***wahr*** enthält	= TRUE	
= FALSCH	Falsifizierung ist erfüllt, wenn der Feldinhalt ***falsch*** enthält	= FALSE	
EXISTS(SELECT-Anweisung**)**	falls mind. ein Datensatz existiert, dann gibt die Funktion TRUE zurück, sonst FALSE		

kombinierte Abfragen (Projektion(en) + Selektion(en))

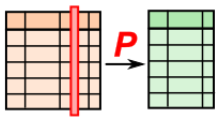
Verbindung von Selektion und Projektion (bezüglich einer Tabelle)
stellen die wirklichen praktisch nutzbaren Sichten in Datenbank-System und vor allem in / für Bedienoberflächen dar
zum Einen wird man so der Datenflut herr (Selektion) und zum anderen schränkt man die Daten auf den Teil ein, mit dem ein bestimmter Nutzer arbeiten darf (Projektion)

sachlich ist die Reihenfolge der Verbindung von Selektion und Projektion egal, es ergibt sich immer das gleiche Ergebnis
in SQL wird zuerst die **Projektion** formuliert (SELECT …) und dann folgt die **Selektion** (…WHERE …)



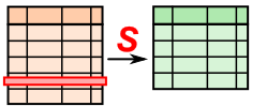
Projektion		Selektion
SELECT ***spaltenauswahl***	FROM *Datenbestand*	**WHERE** ***zeilenauswahl***

bei der Umsetzung in Datenbank-Kommandos (Datenbank-System-interne Befehle) ist eine vorlaufende Selektion meist sinnvoller, da so die Anzahl der weiter zu verarbeitenden Datensätze deutlich eingeschränkt wird. Dannach werden dann die notwendigen Projektionen vorgenommen (die Anzahl der Attribute in einem Datensatz meist deutlich kleiner als die Zahl der Datensätze einer Tabelle)
im SQL kann man schnell aus den vorher besprochenen Anweisungen komplexe Statement's zusammenzusetzen

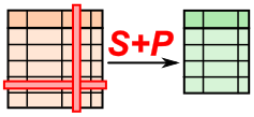


statt des * im SELECT-Teil der Projektionen werden nun konkrete Spalten (Attribute) genannt

oder bei vorhandenen (ausführlichen) SELECT-Teilen wird nun ein WHERE-Statement ergänzt (Selektion)



vorteilhaft ist, dass man die SQL-Befehle stückchenweise ausprobieren kann und sich passende Teile dann zusammenkopieren / zusammenstellen kann
trotz recht langer SQL-Anweisungen bleibt dann die Fehlerquote relativ gering



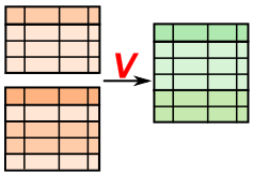
empfohlene Erstellungs-Schrittfolge
1. man startet also z.B. mit einem **SELECT * FROM *tabelle*** (→ praktisch Voll-Projektion)
2. erweitert dann um den **WHERE**-Teil (→ die Selektion)
3. und korrigiert dann den Stern (die Voll-Auswahl) durch die wirklich gebrauchten Spalten (→ (gewünschte) Projektion)

nach jedem Schritt kann man sich das Ergebnis anschauen und ev. gleich korrigierend eingreifen
meist kann auch bei Schritt **2** eine Zusammenstellung aus einzelnen Anweisungs-Teilen erfolgen (z.B. nach den Bedingungen noch Gruppierungen oder Sortierungen ergänzen)

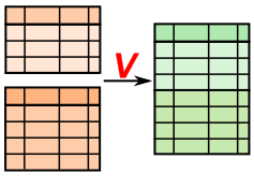
verknüpfte Abfragen (über mehrere Tabellen / Sichten hinweg) (Join('s))

Die Benutzung der Daten aus nur einer Tabelle ist eher die Ausnahme. Sehr viel häufiger werden Daten aus mehreren Tabellen gemeinsam für bestimmte Aufgaben gebraucht.
Um die Daten effektiver zu speichern, haben wir die Tabellen-Strukturen optimiert (→ Normalisierung) und dabei bestimmte Daten in andere Tabellen ausgelagert. In den ursprünglichen Tabellen stehen jetzt ja nur noch Verweise (Fremd-Schlüssel) zu den Datensäzen in den ausgelagerten Daten.
In der Praxis brauchen wir aber die kombinierten Daten. Dazu werden Tabellen wieder miteinander verbunden. Es gibt verschiedene Verbünde (Join's).
Gehen wir aber systematisch vor. Betrachten wir zuerst Tabellen mit vergleichbarer Struktur. In der Praxis könnten das Bestands-Daten aus verschiedenen Standort-Datenbanken, die nun zusammengefügt werden sollen. Solche Tabellen könnten z.B. aus Projektionen (→) stammen – also auch schon View's sein.

R1 **UNION** R2
fügt die Zeilen von zwei Tabellen mit gleicher Spaltenzahl (und Spalten-Struktur) zu einer neuen Tabelle zusammen
die Namen der Spalten müssen nicht gleich sein, die Datentypen müssen aber identisch sein!
doppelte Zeilen werden automatisch unterdrückt, ist dies nicht gewünscht, dann muss **UNION ALL** benutzt werden



Die "einfache" Vereinigung enthält also praktisch auch noch eine Selektion, was bei der vollständigen Vereinigung unterbleibt.



SQL-Repräsentation (→ 5.4.2.x. zwei Tabellen vereinen / einen inneren Verbund zwischen zwei Tabellen herstellen**)**

überarbeiten:

SELECT *spalteX* **{ ,** *spalteY* **}** **FROM** *tabelle*
ORDER BY *spalteX* **[[ASC] | DESC] { ,** *spalteY* **[[ASC] | DESC] } ;**

Aufgaben:

1. Vergleichen Sie die Programm-internen (An-)Sichten mit den echten Sichten (View's, Abfragen)!

3.1.7.1.3.6. Export von Daten

Daten austauschen ist heute eine der wichtigsten informatischen Aufgaben. Erst bei der Möglichkeit Daten in verschiedenen Programmen oder sie mehrfach zu nutzen, wird deren heutige Potenz wirksam.
Der Daten-Export aus dem einen Programm ist also genauso wichtig, wie der Import in dem anderen. Als Austausch-Formate haben sich einige sehr einfache Datei-Typen etabliert. Einige stellen wir hier im Rahmen der Export-Möglichkeiten des SQLiteStudio's vor.
Um die Daten-Repräsentation in den einzelnen Datei-Formaten deutlicher zu machen, zeigen wir den exportierten Daten-Bestand der "Institutionen"-Tabelle immer mit an. Da bieten sich schöne Vergleichs-Möglichkeiten.
Was wir in den nachfolgenden Abschnitten für den Tabellen-Export beschreiben, funktioniert auch für ganze Datenbanken und für Sichten (View's). Das Prinzip bleibt das Gleiche.

Export als CSV

Daten im CSV-Format sind sehr universell zu nutzen. CSV steht für Comma-separeted values (selten auch: Character-separeted values). Schon in einfachen Editoren lassen sie sich anzeigen und bearbeiten. Mehr Funktionen und Möglichkeiten hat man in Tabellenkalkulationen – wie z.B. microsoft®EXCEL® und / oder libreoffice / openoffice CALC). Auch für einen Austausch in andere Betriebssystem-Plattformen ist das CSV-Format hervorragend geeignet. Praktisch gibt es in jedem gängigen Betriebssystem mindestens ein Programm, dass mit CSV klar kommt. Die allgemeinste Codierung ist dann 7-bit-ASCII. Dessen Zeichen-Umfang ist aber sehr eingeschränkt (nur rund 100 Nutz-Zeichen).
Außer Komma-getrennt gibt es viele Spezial-Optionen. So sind heute viele andere Seperatoren möglich. Das ist besonders dann wichtig, wenn die Daten selbst Komma's enthalten (z.B. Texte oder deutsche Zahlen-Formate). Hier bieten sich dann Semikolon's oder Tabulatoren als Trenner an.
Die CSV-Dateien werden manchmal auch als TXT-Dateien abgelegt. Da man auch diese mit einem Editor ansehen kann, ist das Ermitteln des internen Datei-Formates meist nicht wirklich schwierig.
Die "Institutionen"-Tabelle sieht dann (in einem Text-Editor) so aus:

```
InstitutionsID,Bezeichnung,Ort,eMail,Anmerkungen
1,Goethe-Gymnasium,Mustern,GoeGymn@webb.de,
2,"Tanzverein ""Polka""",Musterhausen,post@tv-polka.de,
3,Hilfe e.V.,Meinstadt,info@hilfe.info,
4,Traditionsverein,Cedorf,tradi@verein.de,
5,Let's share,Meinstadt,letsshare@verein.de,
6,Grundschule,Mustern,gs-mustern@webb.de,
```

Aufgaben:

1. *Was ist eigentlich 7-bit-ASCII?*
2. *Informieren Sie sich in Wikipedia über die Definition des CSV-Format's!*
3. *Exportieren Sie die Tabelle "Kontakte" in eine CSV-Datei!*
4. *Überlegen Sie sich einen Algorithmus, der einfache Daten aus einer originalen CSV-Datei imortieren kann! Sie können sich bei den Befehls-Wörtern von Ihrer bevorzugten Programmiersprache leiten lassen.*

Export als PDF

Das PDF-Format (Printable Document Format) ist nicht bzw. nur teilweise Text-basiert. Im Allgemeinen kommt man an die Daten-Felder nicht heran. Dafür ist dieses Format ja auch nicht gedacht. Mit ihm sollen Daten in einer druckbaren Form weitergegeben werden, die beim Betrachter (End-Nutzer) nicht mehr verändert werden sollen / dürfen. Genaugenommen handelt es sich also auch nicht um ein Austausch-Format zum Hin- und Her-Tauschen, sondern nur zum eingeschränkten Informations-Austausch.
PDF ist ein klassisches Berichts-Format. Die ursprüngliche Ausrichtung von PDF war ja auch das einfache Drucken von Dokumenten.
Aber Achtung! Nicht signierte PDF-Dateien sind manipulierbar! Ihre Verwendung ist also immer mit Vorsicht zu genießen.

Ausschnitte der PDF-Export-Datei (Ansicht aus einem Editor)

```
%PDF-1.4
1 0 obj
…
…
>>
stream
xœÝZKo7¾ï¯à¹€ÖŸ@Vâ=0, ‡ ‡ÂiR──────────────────────────────────────────
VP7‡þýò±Ë¥4”L[´#G…
…
>>
startxref
16107
%%EOF
```

Export als HTML

HTML ist die Abkürzung von Hypertext Markup Language (Hypertext-Markierungs-Sprache). Auch hier steht der einseitige Informations-Austausch zum End-Nutzer im Vordergrund. Ein Import ist zwar möglich, dazu eignen sich ähnliche Formate, wie das XML deutlich besser.
HTML-Dateien sind reine Text-Dateien, in denen die Elemente strukturiert abgelegt werden. Hauptzweck ist die Darstellung von Texten, wie sie eben im Internet bei den verschiedenen Internet-Seiten zur Anwendung kommen. Das HTML-Format bestimmt deshalb auch vorrangig die allgemeine Darstellung auf dem Ziel-Rechner. Als Programm wird i.A. ein Browser (z.B. mozilla Firefox oder microsoft®Internet-Explorer®) benutzt.
Die Gestaltung der Daten-Anzeige wird von Tag's bestimmt, die im HTML in gewinkelten Klammern eingeschlossen sind. Bei den meisten Tag's gibt es einen einleitenden und einen beendenden Tag. Beide lauten gleich. Der beendende Tag hat zusätzlich noch einen Schrägstrich (Slash) vor dem Tag-Text. Die Tag's **<HTML>** und **</HTML>** bilden also ein Paar und umschließen z.B. den gesamten HTML-Text. Intern kommen dann geschachtelt weitere Tag's zur Anwendung. Einer der wenigen Tag's, die nur einzeln vorkommen, ist z.B. der Zeilenumbruch
 (für break). Die Groß- und Kleinschreibung der Tag-Texte ist egal. Die meisten Systeme benutzen aber Groß-Buchstaben zur auffälligeren Kennzeichnung. Innerhalb von einigen Tag's dürfen Optionen angegeben werden. Diese werden dann eher kleinbuchstabig geschrieben.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN" "http://www.w3.org/TR/REC-
html40/strict.dtd">
<html>
    <meta http-equiv="Content-Type" content="text/html; charset=System"/>
    <title>Exported table: Institutionen</title>
    <style type="text/css">
        table
        {
            border-style: solid;
            border-width: 1px;
            border-color: black;
            border-collapse: collapse;
…
…
        table tr td.rownum
        {
            padding: 0px 3px 0px 3px;
            border-style: solid;
            border-width: 1px;
            border-color: #666666;
            background-color: #DDDDDD;
            text-align: right;
        }

    </style>
    <body>
        <table>
            <tr class="title">
                <td colspan="6" align="center">Table: Institutionen</td>
            </tr>
            <tr class="header">
                <td align="right">
                    <b><i>#</i></b>
                </td>
                <td>
                    <b>InstitutionsID</b><br/>INTEGER
                </td>
                <td>
                    <b>Bezeichnung</b><br/>STRING
                </td>
                <td>
                    <b>Ort</b><br/>STRING
                </td>
                <td>
                    <b>eMail</b><br/>STRING
                </td>
                <td>
                    <b>Anmerkungen</b><br/>BLOB
                </td>
            </tr>
            <tr>
                <td class="rownum">
                    <i>1</i>
                </td>
                <td align="right">
                    1
                </td>
                <td align="left">
                    Goethe-Gymnasium
                </td>
                <td align="left">
                    Mustern
                </td>
                <td align="left">
                    GoeGymn@webb.de
                </td>
                <td align="left" class="null">
                    <i>NULL</i>
                </td>
```

```
            </tr>
            <tr>
                <td class="rownum">
                    <i>2</i>
                </td>
                <td align="right">
                    2
                </td>
                <td align="left">
                    Tanzverein &amp;quot;Polka&amp;quot;
                </td>
                <td align="left">
                    Musterhausen
                </td>
                <td align="left">
                    post@tv-polka.de
                </td>
                <td align="left" class="null">
                    <i>NULL</i>
                </td>
            </tr>
            <tr>
                <td class="rownum">
                    <i>3</i>
                </td>
                <td align="right">
                    3
                </td>
                <td align="left">
                    Hilfe e.V.
                </td>
                <td align="left">
                    Meinstadt
                </td>
                <td align="left">
                    info@hilfe.info
                </td>
                <td align="left" class="null">
                    <i>NULL</i>
                </td>
            </tr>
            <tr>
                <td class="rownum">
                    <i>4</i>
                </td>
                <td align="right">
                    4
                </td>
                <td align="left">
                    Traditionsverein
                </td>
                <td align="left">
                    Cedorf
                </td>
                <td align="left">
                    tradi@verein.de
                </td>
                <td align="left" class="null">
                    <i>NULL</i>
                </td>
            </tr>
            <tr>
                <td class="rownum">
                    <i>5</i>
                </td>
                <td align="right">
                    5
                </td>
                <td align="left">
                    Let's share
                </td>
```

```
                <td align="left">
                    Meinstadt
                </td>
                <td align="left">
                    letsshare@verein.de
                </td>
                <td align="left" class="null">
                    <i>NULL</i>
                </td>
            </tr>
            <tr>
                <td class="rownum">
                    <i>6</i>
                </td>
                <td align="right">
                    6
                </td>
                <td align="left">
                    Grundschule
                </td>
                <td align="left">
                    Mustern
                </td>
                <td align="left">
                    gs-mustern@webb.de
                </td>
                <td align="left" class="null">
                    <i>NULL</i>
                </td>
            </tr>
        </table>
        <br/><br/>
        <i>Document generated by SQLiteStudio v3.1.1 on Mi Dez 27 19:41:39
2017</i>
    </body>
</html>
```

In einem Browser sieht der Export dann natürlich viel besser aus – zumindestens für uns Menschen. Für den zweiseitigen Datenaustausch zwischen Rechnern ist das HTML-Format ja auch nicht wirklich gedacht.

Aufgaben:

1. *Exportieren Sie die Tabelle "Kontakte" in eine HTML-Datei!*
2. *Betrachten Sie diese Datei in einem Browser!*

für die gehobene Anspruchsebene:

3. *Erstellen Sie mit einem (HTML-)Editor eine minimale HTML-Datei, das eine ganz einfache Tabelle mit den exportierten Daten zeigt!*
4. *Übertragen Sie die Datei (z.B. mittels USB-Stick oder per eMail) auf ein Gerät mit einem anderen Betriebssystem und / oder anderem Browser! Betrachten Sie die Datei dann dort oder in einer passenden App (mit HTML-WYSIWYG-Ansicht's-Möglichkeit! Was stellen Sie fest?*
5. *Finden Sie im HTML-Text passende Entsprechungen für die im Browser dargestellte Datenstruktur (s. Aufg. 4)!*

Export als XML

XML ist stark mit HTML verwandt. Allerdings geht es beim XML (Extensible Markup Language) um die strukturierte Speicherung von Daten in einem universellen Austausch-Format. XML-Dateien sind also auch Text-Dateien, in denen Daten mittel Tag's vor allem Maschinenlesbar gemacht werden sollen. Eine gewisse Lesbarkeit für Menschen sollte aber erhalten bleiben.
Die Einfachheit von XML-Dateien lässt sie zu einem universellen Daten-Format auch für zukünftige Programme und Datenbanken werden. Import- und Export-Filter sind einfach zu programmieren. Die meisten Programmiersprachen bringen gleich von Haus aus Hilfsmittel zur effektiven Bearbeitung von XML-Dateien mit.
Applikationen dürfen eigene Tag's definieren und verwenden. So könnte ein spezielles Programm für die Verwaltung unserer "Institutionen"-Tabelle sich die Tag-Paare <InstitutionsID> </InstitutionsID>, <Bezeichnung> </Bezeichnung>, <Ort> </Ort>, <eMail> </eMail> und <Anmerkungen> </Anmerkungen> definieren. Die Definition wird in einer extra DTD-Datei gespeichert, auf die in der eigentlichen XML-Datei hingewiesen wird. Diese Art der XML-Dateien sind dann für Menschen deutlich besser zu lesen. Zeilenumbrüche verbessern ebenfalls die Lesbarkeit. In der reinen Anwendung sind sie nicht notwendig und fehlen entsprechend.

```
<?xml version="1.0" encoding="System"?>
<table>
    <database></database>
    <name>Institutionen</name>
    <ddl>CREATE TABLE Institutionen (InstitutionsID INTEGER PRIMARY KEY ASC ON
CONFLICT ROLLBACK AUTOINCREMENT, Bezeichnung STRING (30), Ort STRING (20), eMail
STRING (20), Anmerkungen BLOB);</ddl>
    <columns>
        <column>
            <name>InstitutionsID</name>
            <type>INTEGER</type>
            <constraints>
                <constraint>
                    <type>PRIMARY KEY</type>
                    <definition>PRIMARY KEY ASC ON CONFLICT ROLLBACK AUTOINCRE-
MENT</definition>
                </constraint>
            </constraints>
        </column>
        <column>
            <name>Bezeichnung</name>
            <type>STRING</type>
        </column>
        <column>
            <name>Ort</name>
            <type>STRING</type>
        </column>
        <column>
            <name>eMail</name>
            <type>STRING</type>
        </column>
        <column>
            <name>Anmerkungen</name>
            <type>BLOB</type>
        </column>
    </columns>
    <rows>
        <row>
            <value column="0">1</value>
            <value column="1">Goethe-Gymnasium</value>
            <value column="2">Mustern</value>
            <value column="3">GoeGymn@webb.de</value>
            <value column="4" null="true"/>
        </row>
```

```
        <row>
            <value column="0">2</value>
            <value column="1">Tanzverein &amp;quot;Polka&amp;quot;</value>
            <value column="2">Musterhausen</value>
            <value column="3">post@tv-polka.de</value>
            <value column="4" null="true"/>
        </row>
        <row>
            <value column="0">3</value>
            <value column="1">Hilfe e.V.</value>
            <value column="2">Meinstadt</value>
            <value column="3">info@hilfe.info</value>
            <value column="4" null="true"/>
        </row>
        <row>
            <value column="0">4</value>
            <value column="1">Traditionsverein</value>
            <value column="2">Cedorf</value>
            <value column="3">tradi@verein.de</value>
            <value column="4" null="true"/>
        </row>
        <row>
            <value column="0">5</value>
            <value column="1">Let's share</value>
            <value column="2">Meinstadt</value>
            <value column="3">letsshare@verein.de</value>
            <value column="4" null="true"/>
        </row>
        <row>
            <value column="0">6</value>
            <value column="1">Grundschule</value>
            <value column="2">Mustern</value>
            <value column="3">gs-mustern@webb.de</value>
            <value column="4" null="true"/>
        </row>
    </rows>
</table>
```

Export mit nachträglicher Hervorhebung bestimmter Struktur-Abschnitte

Aufgaben:

1. *Welche Funktion(en) haben die Struktur-Abschnitte in der Beispiel-Datei?*
2. *Finden Sie die Strukturfehler in der nachfolgenden abstrahierten XML-Datei!*

	Quell-Text	Kommentare
1	<Zoo>	
2	<Tiere>	
3	<Lurche>	
4	<Pfleger> Meier </pfleger>	
5	<Lurch> Moorfrosch </Lurch>	
6	<Lurch> Ringelnatter </Lurch>	
7	</Lurche>	
8	<Saeuger>	
9	<Betreuer> Dietrich </Betreuer>	
10	<Betreuer> Neumann <Betreuer>	
11	<Saeuger> Elephant </Saeuger>	
12	<Saeuger> Löwe </Löwe>	
13	<Tiere>	
14	<Gebaeude>	
15	Gehege: Rotwildgehege	
16	Käfig: Löwenkäfig	
17	<Gebaeude>	
18	<Personal>	
19	<Pfleger> Meier </Pfleger>	
20	<Pfleger> Dietrich </Pfleger>	
21	<Kassierer> Neumann </Kassierer>	
22	</Personal>	
23	</Direktor> <Direktor>	
24	>/Zoo>	

2. *Erstellen Sie mit einem Text-Editor eine XML-Datei mit den im Text erwähnten Tag's für eine "Kontakte"-XML-Datei!*

Export als JSON

JSON-Dateien sind sehr moderne, Text-basierte Austausch-Dateien. Auch für sie gibt es oft fertige Bibliotheken zu den verschiedenen Programmiersprachen. Ursprünglich stammt das Format aus der Programmiersprache JAVA. Der vollständige Name lautet: JavaScript Object Notation.
Beim JSON-Format sind die Daten vorrangig mittels Klammern und Komma als Separator strukturiert. Je nach Datenstruktur wird ein Eintrag z.B. mittels Name und Wert (Key-Value-Struktur) verwaltet. Beide sind durch einen Doppelpunkt getrennt. Für tabellarische Daten wird dann z.B. in Kopfzeile und Datenzeile(n) unterschieden. Das JSON-Format ist vor allem wegen seines sparsamen Einsatzes von Steuer-Zeichen beliebt. Im Allgemeinen sind JSON-Dateien kleiner als XML-Dateien – bei gleichem Inhalt.

```
{
    "type": "table",
    "database": null,
    "name": "Institutionen",
    "withoutRowId": true,
    "ddl": "CREATE TABLE Institutionen (InstitutionsID INTEGER PRIMARY KEY ASC ON
CONFLICT ROLLBACK AUTOINCREMENT, Bezeichnung STRING (30), Ort STRING (20), eMail
STRING (20), Anmerkungen BLOB);",
    "columns": [
        {
            "name": "InstitutionsID",
            "type": "INTEGER",
            "constraints": [
                {
                    "type": "PRIMARY KEY",
                    "definition": "PRIMARY KEY ASC ON CONFLICT ROLLBACK AUTOINCRE-
MENT"
                }
            ]
        },
        {
            "name": "Bezeichnung",
            "type": "STRING"
        },
        {
            "name": "Ort",
            "type": "STRING"
        },
        {
            "name": "eMail",
            "type": "STRING"
        },
        {
            "name": "Anmerkungen",
            "type": "BLOB"
        }
    ],
    "rows": [
        [
            1,
            "Goethe-Gymnasium",
            "Mustern",
            "GoeGymn@webb.de",
            null
        ],
        [
            2,
            "Tanzverein \\"Polka\\"",
            "Musterhausen",
            "post@tv-polka.de",
            null
        ],
        [
            3,
            "Hilfe e.V.",
            "Meinstadt",
            "info@hilfe.info",
            null
        ],
        [
            4,
            "Traditionsverein",
            "Cedorf",
            "tradi@verein.de",
            null
        ],
        [
            5,
            "Let's share",
```

```
            "Meinstadt",
            "letsshare@verein.de",
            null
        ],
        [
            6,
            "Grundschule",
            "Mustern",
            "gs-mustern@webb.de",
            null
        ]
    ]
}
```

Aufgaben:

1. *Informieren Sie sich in Wikipedia über die Definition des JSON-Format's!*
2. *Erstellen Sie auf der Grundlage des Wikipedia-Artikels eine rudimentäre JSON-Datei (handschriftlich auf dem Papier) zur / aus der "Kontakte"-Tabelle!*
3. *Vergleichen Sie das CSV-, XML- und JSON-Datei-Format in einer Tabelle! Kennzeichen Sie Gemeinsamkeiten und Unterschiede auch zwischen den verschiedenen Format-Paaren!*

Export als SQL

Die Datenbank-Sprache SQL kann neben der eigentlichen Arbeit an einem Datenbank-Management-System auch zum Datenaustausch benutzt werden. SQL ist ja quasi die Muttersprache der relationalen Datenbanken.
SQL-Dateien sind ebenfalls Text-basiert und sowohl Maschinen- als auch sehr gut von Menschen lesbar. Zeilen mit zwei Bindestrichen sind Kommentarzeilen und können auch weggelassen werden.
In Mehr-Nutzer-Systemen muss die spätere Einspeisung von SQL-Code sicherstellen, dass nicht ein anderer Nutzer dazwischenfunkt. Deshalb sind noch einige zusätzliche SQL-Anweisungen (SQL-Statement's) notwendig. Das sind z.B. BEGIN und COMMIT TRANSAKTION.
Interessant ist für den reinen Daten-Import später nur die Sequenz der INSERT-Anweisungen. U.U. muss eine SQL-Export-Datei für spezielle Zwecke in einem Editor angepasst werden, damit z.B. eine schon vorhandene Tabelle nicht zerstört wird.

```
--
-- File generated with SQLiteStudio v3.1.1 on Mi Dez 27 16:39:54 2017
--
-- Text encoding used: System
--
PRAGMA foreign_keys = off;
BEGIN TRANSACTION;

-- Table: Institutionen
CREATE TABLE Institutionen (InstitutionsID INTEGER PRIMARY KEY ASC ON CON-
FLICT ROLLBACK AUTOINCREMENT, Bezeichnung STRING (30), Ort STRING (20),
eMail STRING (20), Anmerkungen BLOB);
INSERT INTO Institutionen (InstitutionsID, Bezeichnung, Ort, eMail, Anmer-
kungen) VALUES (1, 'Goethe-Gymnasium', 'Mustern', 'GoeGymn@webb.de',
NULL);
INSERT INTO Institutionen (InstitutionsID, Bezeichnung, Ort, eMail, Anmer-
kungen) VALUES (2, 'Tanzverein "Polka"', 'Musterhausen', 'post@tv-
```

```
polka.de', NULL);
INSERT INTO Institutionen (InstitutionsID, Bezeichnung, Ort, eMail, Anmer-
kungen) VALUES (3, 'Hilfe e.V.', 'Meinstadt', 'info@hilfe.info', NULL);
INSERT INTO Institutionen (InstitutionsID, Bezeichnung, Ort, eMail, Anmer-
kungen) VALUES (4, 'Traditionsverein', 'Cedorf', 'tradi@verein.de', NULL);
INSERT INTO Institutionen (InstitutionsID, Bezeichnung, Ort, eMail, Anmer-
kungen) VALUES (5, 'Let''s share', 'Meinstadt', 'letsshare@verein.de',
NULL);
INSERT INTO Institutionen (InstitutionsID, Bezeichnung, Ort, eMail, Anmer-
kungen) VALUES (6, 'Grundschule', 'Mustern', 'gs-mustern@webb.de', NULL);

COMMIT TRANSACTION;
PRAGMA foreign_keys = on;
```

Aufgaben:

1. ***Exportieren Sie die Tabelle "Kontakte" in eine SQL-Datei!***
2. ***Erstellen Sie sich eine zweite Datenbank "ZweitDatenbank" im SQLiteStudio und importieren Sie die SQL-Datei später in diese Datenbank!***

3.1.7.1.3.7. Import der exportierten Daten in andere Programme

Alle oben gekennzeichneten Text-Datei-Formate lassen sich mit einfachen Editoren anzeigen. Nicht immer wird dabei die eigentliche Daten-Struktur erkennbar, da die Einträge in den Daten z.T. ohne Einrückungen und zusätzliche Zeilenumbrüche auskommen. Einige Spezial-Programme verfügen aber über Programm-interne Formatierungs-Hilfen, die recht ansehnliche Ergebnisse erzeugen.
Geht es nur um die Anzeige, dann helfen bei HTML- und XML-Dateien auch Browser (z.B. mozilla Firefox oder microsoft®Internet-Explorer®) weiter.
Tabellen-Kalkulationen sind ansosnten immer gute Import-Programme für Daten-Tabellen. Sowohl microsoft®EXCEL® oder libreoffice / openoffice CALC kommen sehr gut mit den exportierten CSV-Dateien klar.
Da gilt sicher auch für andere Office-Produkte (FreeOffice von SoftMaker) oder spezielle Tabellenkalkulations-Apps. Da sie aber im Schulbereich und in der freien Arbeits-Praxis eine geringere Rolle spielen, wurde die Funktionabilität nicht weiter geprüft.
Beim Import müssen zumeist die Textcodierung, der Seperator und das Vorhandensein der Kopf-Zeilen bzw. Defintions-Strukturen angegeben werden. Stellt man hier alles ordentlich ein, dann bekommt man meist sehr ordentliche Daten-Tabellen. Diese können dann in einem Programm-internen Tabellkalkulations-Dateityp abgspeichert werden. Es stehen dann auch alle Arbeits-Funktionen und Programm-Leistungen (z.B. Diagramme) zur Verfügung.

3.1.7.1.3.8. Nutzung der Datenbank über Programmiersprachen

Die Programmierung und "Fremd"-Nutzung von Datenbanken besprechen wir weiter hinten (→ 6. Datenbanken - Programmierung). Interessanterweise sind sich die üblichen Programmiersprachen im Umgang mit SQL-Datenbanken recht einheitlich.
An dieser Stelle soll nur mal kurz ein Beispiel aus der Programmiersprache Python aufgezeigt werden.

Datennutzung / Datenzugriff mit Python

Die Kommunikation zwischen Datenbank(-Server) und Client (unser Python-Programm) läuft über SQL-Code. Das funktioniert praktisch bei allen SQL-Datenbanken und vielen anderen Systemen mehr. Eine entsprechende SQL-Schnittstelle gehört heute zu den Minimal-Funktionen. Niemand will für ein neues Datenbanksystem alle seine Daten neu eingeben oder seine Hilfs-Scripte neu programmieren. Sachlich gibt es auch wenige Gründe, weshalb man kein SQL nutzen sollte. Diese Gründe sind vornehmlich für Profi's interessant und spielen in normalen Datenbank-Welten kaum eine Rolle.

```
import sqlite3

conn = sqlite3.connect('daten/Kontakte.dat')
curs = conn.cursor()
curs.execute("CREATE TABLE personen(PID, Vorname, Name, eMail)")

DatenListe=[(1,"Monika","Musterfrau","musterfrau@webb.de"),
            (2,"Klaus","Mustermann","muma@tee-online.de"),
            (3,"Prof. Lisa","Klug","Prof.L.Klug@uni-mustern.de)]

for Elem in DatenListe:
    curs.execute("Insert INTO personen VALUES (?,?,?,?)", Elem)

…

curs.close()
conn.close()
```

curs.fetchone() liefert eine Ergebnis-Zeile zur Anfrage als Tupel
curs.fetchmany(n) liefert n Ergebniszeilen zur Anfrage als n-Tupel von Tupeln
curs.fetchall() liefert alle Ergebniszeilen zur Anfrage als Tupel von Tupeln

zusätzliche Informationen und Programm-Beispiele gibt es im Programmierungs-Kapitel (→)
oder auch im Skript 🕮 **Python**

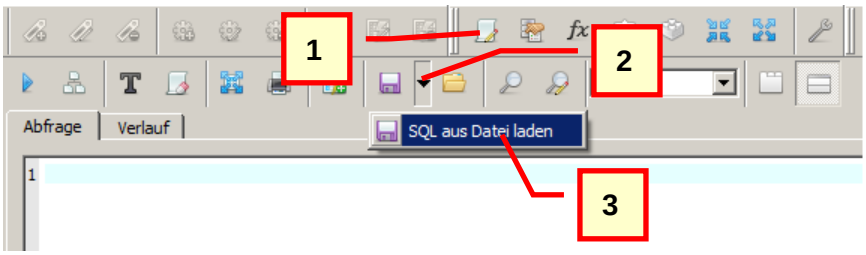
Links:
https://wiki.sqlitestudio.pl/index.php/User_Manual (online-Hilfe, Bedienungsanleitung)

3.1.7.1.3.9. SQL-Datenbanken importieren

ganze Datenbanken im SQL-Format werden am Besten direkt über sqlite importiert.
→ 3.1.7.1.4. SQL-Datenbanken in SQLite importieren

bei SQLiteStudio:
zuerst Erstellen einer Datenbank-Datei in SQLite-Studio

Scheinbar falsch! (auch schon falsche Symbolik) hier scheinbar speichern gemeint
Öffnen des SQL-Editors und dort neben dem Disketten-Symbol den Auswähler anklicken und "SQL aus Datei laden"
("Load SQL from file")

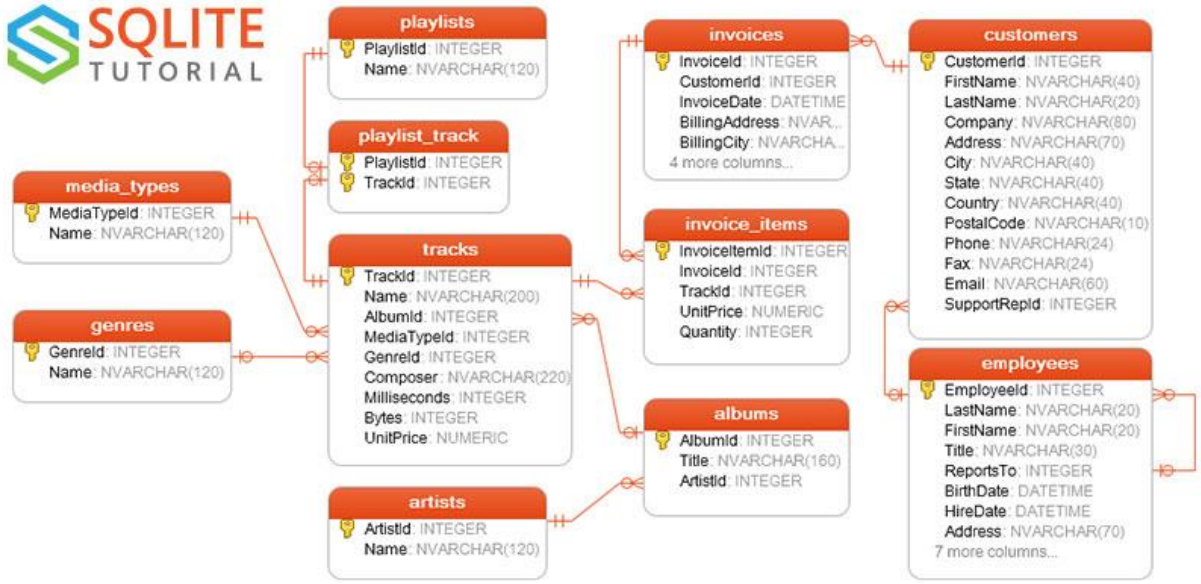


??? SQLiteStudio unterstützt nur Import von Daten in vorhandene Tabellen

Beispiel-Datenbanken:

von SQLite selbst bereitgestellt

chinook



Q: https://www.sqlitetutorial.net/sqlite-tutorial/sqlite-sample-database/

zum Integrieren mittel Kommandozeile ins Verzeichnis wechseln, in dem SQLite installiert ist dort dann die folgenden Eingaben machen:
…sqlite> **sqlite3 *Pfad*\chinook.db**

…sqlite> **.tables**
werden die importierten Tabellen angezeigt

SQLiteStudio - Tastatur-Kürzel (original (lassen sich bei den "Einstellungen" anpassen))

Aktion	Tastenkombination
Editor für Textwerte in Zellen	
Gewählten Text kopieren	Ctrl+C
Gewählten Text ausschneiden	Ctrl+X
Gewählten Text löschen	Del
Von der Zwischenablage einfügen	Ctrl+V
Wiederholen	Ctrl+Y
Rückgängig	Ctrl+Z
Ergebnisansicht	
Änderungen der Zellenninhalte speichern	Ctrl+Return
Kopiert Zelleninhalt(e) in die Zwischenablage	Ctrl+C
Markierte Datenzeile löschen	Del
Fügt einen leeren Wert in die selektierte(n) Zelle(n) ein	Alt+Backspace
Neue Datenzeile einfügen	Ins
Inhalt der markierten Zelle im separaten Editor öffnen	Alt+Return
Fügt Zelleninhalt(e) von der Zwischenablage ein	Ctrl+V
Änderungen der Zelleninhalte zurücknehmen	Ctrl+Backspace
Fügt den NULL Wert in die selektierte(n) Zelle(n) ein	Backspace
Ergebnisansicht (tabellarisch und Formular)	
Daten aktualisieren	F5
Zur Formularansicht wechseln	Ctrl+.
Zur tabellarischen Ergebnisansicht wechseln	Ctrl+,
Formularansicht der Ergebnisse	
Änderungen der aktuellen Zeile speichern	Ctrl+Return
Derzeitige Zeile löschen	Ctrl+Del
Springe zur ersten Zeile dieser Seite	Ctrl+Alt+PgUp
Neue Zeile einfügen	Ins
Springe zur letzten Zeile dieser Seite	Ctrl+Alt+PgDown
Springe zur nächsten Zeile	Ctrl+Alt+Right
Springe zur vorherigen Zeile	Ctrl+Alt+Left
Änderungen der aktuellen Zeile zurücknehmen	Ctrl+Backspace
Hauptfenster	
Statusfeld verbergen	Esc
Nächstes Fenster	Ctrl+PgDown
Konfigurationsdialog öffnen	F2
CSS Konsole öffnen	F11
Debug Konsole öffnen	F12
SQL Editor öffnen	Alt+E
Vorheriges Fenster	Ctrl+PgUp
Liste der Datenbanken	
Datenbank hinzufügen	Ctrl+O
Filter zurücksetzen	Esc
Gewählte Einträge kopieren	Ctrl+C
Gewählten Eintrag löschen	Del
Von der Zwischenablage einfügen	Ctrl+V
Schema aktualisieren	F5
Alle Schemas aktualisieren	Shift+F5
Alles auswählen	Ctrl+A
Neues Fenster zufügen	
Neuen Trigger zufügen	Ins
Gewählten Trigger löschen	Del
Gewählten Trigger bearbeiten	Return
Springe zum nächsten Reiter	Alt+Right
Springe zum vorherigen Reiter	Alt+Left
Aktualisiere View Triggerliste	F5

Aktion	Tastenkombination
Report-Verlaufsfenster	
Gewählten Eintrag löschen	Del
SQL Editor Eingabefeld	
Code-Assistenten anfordern	Ctrl+Space
Gewählten Text kopieren	Ctrl+C
Selektierten Textblock kopieren und unterhalb einfügen	Ctrl+Alt+Down
Selektierten Textblock kopieren und oberhalb einfügen	Ctrl+Alt+Up
Gewählten Text ausschneiden	Ctrl+X
Gewählten Text löschen	Del
Aktuelle Zeile löschen	Ctrl+D
Suche im Text	Ctrl+F
Nächster Fund	F3
Vorheriger Fund	Shift+F3
Format-Inhalte	Ctrl+T
Selektierten Textblock eine Zeile nach unten verschieben	Alt+Down
Selektierten Textblock eine Zeile nach oben verschieben	Alt+Up
Inhalte aus einer Datei laden	Ctrl+O
Von der Zwischenablage einfügen	Ctrl+V
Wiederholen	Ctrl+Y
Ersetze im Text	Ctrl+H
Inhalte in eine Datei speichern	Ctrl+S
Gesamten Editorinhalt auswählen	Ctrl+A
Toggle comment	Ctrl+/
Rückgängig	Ctrl+Z
SQL Editor-Fenster	
Abfrage ausführen	F9
Execute "EXPLAIN" query	F8
Tastatureingabe-Fokus in das obere SQL Editorfenster setzen	Alt+PgUp
Tastatureingabe-Fokus in das untere Ergebnisfenster setzen	Alt+PgDown
Wechsel von der aktuellen Datenbank zur nächsten in der Liste	Ctrl+Down
Wechsel von der aktuellen Datenbank zur vorherigen in der Liste	Ctrl+Up
Gehe zum nächsten Editor-Reiter	Alt+Right
Gehe zum vorherigen Editor-Reiter	Alt+Left
Tabellenfenster	
Neue Spalte zufügen	Ins
Neuen Index zufügen	Ins
Neue Tabellenbedingung zufügen	Ins
Neuen Trigger zufügen	Ins
Gewählte Spalte löschen	Del
Gewählten Index löschen	Del
Markierte Tabellenbedingung löschen	Del
Gewählten Trigger löschen	Del
Gewählte Spalte bearbeiten	Return
Gewählten Index bearbeiten	Return
Markierte Tabellenbedingung bearbeiten	Return
Gewählten Trigger bearbeiten	Return
Tabellendaten exportieren	Ctrl+E
Daten in die Tabelle importieren	Ctrl+I
Springe zum nächsten Reiter	Alt+Right
Springe zum vorherigen Reiter	Alt+Left
Aktualisiere Tabellenindexliste	F5
Aktualisiere Tabellenstruktur	F5
Aktualisiere Tabellentriggerliste	F5

Beispiel: Literatur-Datenbank (LiDaBa)

Hilfs-Quellen:
https://www.geschichte.tu-darmstadt.de/index.php?id=1123

Aufgaben-Beschreibung:

Für ein Forschungsgebiet und zu erstellende Dokumentationen (konkret: Facharbeit, Diplomarbeit, Hausarbeit, Bachelorarbeit, Promotion, …) soll eine Sammlung der verfügbaren Literaturquellen zusammengestellt werden. In der Datenbank soll später z.B. nach Autoren, Titeln, Schlüsselbegriffen, Verlagen, ISBN-Nummern usw. gesucht werden und die Daten strukturiert zusammengestellt werden.

Aufgaben:

1. *Analysieren Sie, welche Daten Sie für die Erstellung einer Literatur-Datenbank – wie oben beschrieben – benötigen!*
2. *Untersuchen Sie, welche Daten mit großer Wahrscheinlichkeit mehrfach in der Datenbank gespeichert werden!*
3. *Schlagen Sie ein ER-Modell für die LiDaBa vor (auf einem A4-Blatt)!*
4. *Diskutieren Sie die verschiedenen Vorschläge und leiten Sie aus den Vorschlägen eine (zumindestens theoretisch) optimale Lösung ab!*
5. *Erstellen Sie ein – für den Kurs gemeinsam akzeptiertes – Entity-Relationship-Diagramm in einem Zeichenprogramm! Exportieren Sie das Diagramm als nutzbare Graphik-Datei (z.B.: PNG, JPG, GIF oder in der Not als BMP)*
6. *Legen Sie eine Projekt-Dokumentation an! Übernehmen Sie Aufgabenstellung und dokumentieren Sie die Entwicklungsschritte Ihrer Modell-Erstellung und –Umsetzung! (Spätestens am Ende der Unterrichtstunden immer kurz aktualisieren!)*
7. *Setzen Sie die Datenbank vollständig in Ihr favorisiertes Datenbank-System um!*

3.1.7.2. SQL-Datenbanken in SQLite importieren

SQLite bietet viele Möglichkeiten
Voraussetzung ist aber eine echte SQLite-Installation

`sqlite` *DatenbankDatei* `-init` *SQL_Datei*

`sqlite3.exe` *DatenbankDatei* `".read` *SQL_Datei*`"`

von einem SQLite-Prompt (siehe dazu auch: → 3.1.7.3. Nutzung von SQLite mittels Kommandozeilen-Befehlen):

```
sqlite> .read datenbank.sql
```

oder

```
cat datenbank.sql | sqlite3 datenbank.db
```

! wichtig ist, dass die SQL-Statements mit einem Semikolon (**;**) enden

3.1.7.3. Nutzung von SQLite mittels Kommandozeilen-Befehlen

Voraussetzung ist aber eine echte / lokale SQLite-Installation
SQLite-Studio bringt eine sqlite3.dll mit, diese Datei enthält zwar die Funktionen
die Funktionen lassen sich aber nicht direkt starten

im Installations-Verzeichnis von SQLite gibt es die sqlite3.exe
diese Datei ist das Kommandozeilen-Programm

direkter Start:
`sqlite3`

Start mit Datenbank
`sqlite3`

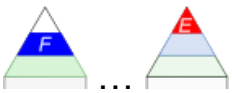
alle Befehle beginnen mit einem Punkt (**.**), deshalb auch dot-Kommando's genannt
dahinter ev. Optionen, die mit einem Bindestrich beginnen oder passende Argumente, wie z.B. die benutzte Datenbank usw. usf.
Anzeige der verfügbaren Kommando's mit
`.help`

zurück auf die "normale" Kommando-Ebene:
`.exit`

oder Beenden mit dem System-typischen End-of-File-Signal's
Windows: [Strg] + [Z]
Linux: [Strg] + [D]

weiterführende Links:
https://www.sqlite.org/draft/cli.html (engl. Beschreibung zu vielen sqlite-Kommando's)

3.1.8. Datenbanken mit PROLOG



PROLOG ist eigentlich ein System zur logischen Programmierung und Nutzen von Wissens-Datenbanken. Auch wenn der relationale Ansatz nicht ganz fern ist, sind Datenbanken in PROLOG doch ganz anders strukturiert.
Datenbanken – besser Fakten-Sammlungen – bestehen aus Atomen. Ein Atom stellt dabei einen Datensatz dar. Der Atom-Name ist als Tabellen-Name zu verstehen.

Tabelle				
Attribut1	Attribut2	Attribut3	**…**	AttributN

tabelle(attribut1, attribut2, attribut3, …, attributN**).**

Es fällt gleich auf, dass es keine Attribut-Namen gibt. Nur die Daten und der Tabellen-Name ist in PROLOG sichtbar. Die Attribut-Namen können wir natürlich in Kommentaren mit angeben. Das erhöht die Verständlichkeit der Fakten-Sammlung.
Der größte Unterschied zu relationalen Datenbanken ist die fehlende Schlüssel-Struktur. PROLOG unterstützt keine Primär-Schlüssel. Diese können wir zwar nachbilden, da aber jeder Nutzer die Daten-Basis jederzeit beliebig ändern kann, ist die Einmaligkeit von Primär-Schlüsseln nicht gewährleistet. Automatismen zum Vergeben von fortlaufenden oder einmaligen Schlüsseln gibt es in PROLOG nicht. Für die exakte Vergabe von Schlüsseln ist der Nutzer vollständig selbst verantwortlich.

```
%% Fakten → Schüler
%% schueler(Name,Vorname,weiblich,GebTag,GebMonat,GebJahr)
schueler("Bauer","Marie",true,10,5,2002).
schueler("Droste","Alex",false,1,08,2000).
schueler("Müller","Melanie",true,7,10,2001).
schueler("Müller","Guido",true,17,3,2002).
schueler("Zander","Erico",false,13,6,2001).
```

Eine **Projektion**, wie sie z.B. durch die SQL-Anweisung:

```
SELECT Name, GebJahr FROM Schueler;
```

realisiert werden würde, könnte in PROLOG durch:

```
?- schueler(Name, _, _, _, _, GebJahr).
```

dargestellt werden. Dadurch erhält man für die obige Daten-Basis:

Eine **Selektion**, die z.B. durch die folgende SQL-Anweisung repräsentiert werden würde:

```
SELECT * FROM Schueler WHERE GebJahr = 2001;
```

sollte dann in PROLOG durch:

```
?- schueler(Name, Vorname, Weiblich, GebTag, GebMonat, 2001).
```

oder:

```
?- schueler(Name. Vorname, Weiblich, GebTag, GebMonat, X), X = 2001.
```

ersetzbar sein. Das Ergebnis ist wenig überraschend:

Natürlich lassen sich auch **Projektion und Selektion** verknüpfen. Entsprechend zu:

```
SELECT Vorname FROM Schueler WHERE GebTag > 15;
```

ist der PROLOG-Konstrukt:

```
?- schueler(_, Vorname, _, X, _, _), X > 15.
```

Da ergibt sich für die obige "Tabelle":

Bei den Selektionen interessieren uns noch UND- bzw. ODER-Verknüpfungen, wie z.B.:

```
SELECT Name, Vorname FROM Schueler
       WHERE Weiblich = false AND GebJahr = 2002
```

Sie sind ebenfalls Problem-los in PROLOG abbildbar:

```
?- schueler(Name, Vorname, false, _, _, 2002).
```

Für ODER sähe die Umsetzung von:

```
SELECT Name FROM Schueler WHERE GebMonat = 7 OR GebMonat = 11;
```

in PROLOG so aus:

```
?- schueler(Name, _, _, _, 7, _); schueler(Name, _, _, _, 11, _).
```

Bleibt die Verknüpfung von zwei Tabellen in einem Join. Als 2. Tabelle sei hier die Wort-Ersetzung der Monate gegeben.

```
%% Fakten → Monate
%% monat(Monat,MonatsName).
```

```
monat(1,'Januar').        monat(2, 'Februar').      monat(3, 'März').
monat(4,'April').         monat(5, 'Mai').          monat(6, 'Juni').
monat(7,'Juli').          monat(8, August').        monat(9, September').
monat(10,'Oktober').      monat(11, 'November').    monat(12, 'Dezember').
```

Für eine Anzeige wollen wir nun den Geburts-Monat als Text angegeben haben. Die SQL-Anweisung könnte so aussehen:

```
SELECT schueler.Name, monat.MonatsName
FROM INNER JOIN schueler, monat
WHERE schueler.GebMonat=monat.Monat
```

Das PROLOG-Äquivalent ließe sich so ausdrücken:

```
schueler(Name, _, _, _, _X, _), monat(_X,GebMonat).
```

Ein weiteres Problem taucht auf, wenn wir unsere Daten-Basis ändern wollen. Die Daten werden beim sogenannten "Konsultieren" in den Speicher geladen und sind hier unveränderlich. Erst nach einem Ändern im Quell-Text und erneutem "Konsultieren" besitzen wir eine geänderte Daten-Basis im Speicher. Für rein lesenden Zugriff auf Daten ist das ok.
Es gibt aber die Möglichkeit in PROLOG dynamische Strukturen anzulegen. Dazu müssen die Atome entsprechend deklariert werden.

```
:- dynamic funktor/stelligkeit.
```

In unserem Schüler-Beispiel würde das so aussehen:

```
:- dynamic schueler/6
```

Die Angabe der Atome bleibt so, wie oben aufgezeigt. Mit dem **listing**-Befehl können wir uns den aktuellen Daten-Stapel (praktisch wohl eine Listen-Struktur) ansehen:

```
?- listing(schueler/6).
```

So ergibt sich die bekannte Daten-Basis:

PROLOG stellt noch weitere Funktionen bereit, die dem Handling der Daten-Struktur dienen:

Funktionen für dynamische Daten-Strukturen (in PROLOG)

- **asserta(fakt).** fügt den Fakt an den Anfang der dynamischen Daten-Struktur ein
- **assertz(fakt).** hängt den Fakt an das Ende der dynamischen Daten-Struktur an
- **retract(fakt).** löscht den angegebenen Fakt aus der dynamischen Daten-Struktur (quasi Löschen eines Datensatzes)
- **retractall(fakt).** löscht alle Fakten zum angegebenen Funktor (quasi Löschen der gesamten Tabelle)

Ergänzend kann eine eigene Regel zum Einfügen in die Daten-Struktur definiert werden:

```
einfuegen(N,V,G,T,M,J):- assertz(schueler(N,V,G,T,M,J)).
```

Diese macht aber nur dann Sinn, wenn der Quelltext für noch mehr Personen lesbar sein soll, oder die interne Fakten-Struktur gestaffelt oder stärker strukturiert ist.

Beispiel für strukturierte Fakten:

```
einfuegen(N,V,G,T,M,J):- assertz(schueler(N,V,G,gebDat(T,M,J))).
```

```
abfrage_NamensListe:- schueler(Name, Vorname, _, _, _, _),
                      write(Vorname), write(' '), write(Name), nl, fail.
```

Interessant ist die Verwendung von fail am Ende der Regel-Definition. Dieses fail sorgt dafür, dass die Daten-Basis nach weiteren passenden Fakten durchsucht wird und auch diese der Anzeige (/ der Regel) zur Verfügung gestellt werden. Fail schlägt immer fehl und erzwingt damit den nächsten Durchlauf.
Aufgerufen wird diese Regel durch:

```
?- abfrage_NamensListe.
```

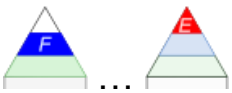
und liefert als Ergebnis eine übersichtliche Liste:
Nun fehlt noch eine Überschrift über die Liste. Auch das ist kein Problem:

```
abfrage_NamensListe:- write('Klassenliste'),nl,
                      ( schueler(Name, Vorname, _, _, _, _),
                        write(Vorname), write(' '), write(Name), nl, fail
                      ).
```

Aufgaben:

1.
2. *Was würde eigentlich passieren, wenn man statt "fail" das Schlüsselwörtchen "cut" in die Regel einbaut? Stellen Sie Voraussagen auf! Recherchieren dazu auch im Internet! Probieren Sie es dann aus!*
3. *Setzen Sie unsere Test-Datenbank (Tabellen "Kontakte" und "Institutionen" einschließlich Verknüpfung) in PROLOG um!*

3.1.8.1. fortgeschrittene Datenbank-Techniken mit PROLOG



Hier sind jetzt nicht nur die Datenabk-Operationen etwas anspruchsvoller, sondern auch die PROLOG-Umsetzungen.
Zuerst wollen wir bestimmte Attribute aus der Daten-Basis herausziehen und sie in einer Liste für die weitere Verwendung bereitstellen.
Als Beispiel wählen wir hier mal eine gewünschte Liste aus den Vornamen der Mädchen.
Das Prädikat zum Finden aller Passungen ist **findall/3**. Das erste Argument ist die Variable für das rauszuziehende Element gefolgt von dem Ausdruck, der ausgewertet werden soll. Als 3. Argument müssen wir noch eine Variable angeben, in der die Ergebnis-Liste aufgebaut werden soll.

```
?- findall(Elem, schueler(_, Elem, _, _, _, _), Liste).
```

Dieser Ausdruck liefert uns zuerst einmal alle Vornamen:

Damit eventuelle Bedingungen eingebaut werden können, muss das mittlere Argument um die Bedingung (weiblich) erweitert werden. Der gesamte (mittlere) Ausdruck muss in eine Klammer gefasst werden, da die Gesamt-Anzahl der Argumente für **findall** natürlich bei drei bleiben muss.

```
?- findall(Elem, (schueler(_, Elem, W, _, _, _), W = true), Liste).
```

So erhalten wir die gewünschte Liste:

Links:
https://www.tinohempel.de/info/info/prolog/datenbank.htm
https://www.philipphauer.de/info/info/datenbanken-in-prolog/

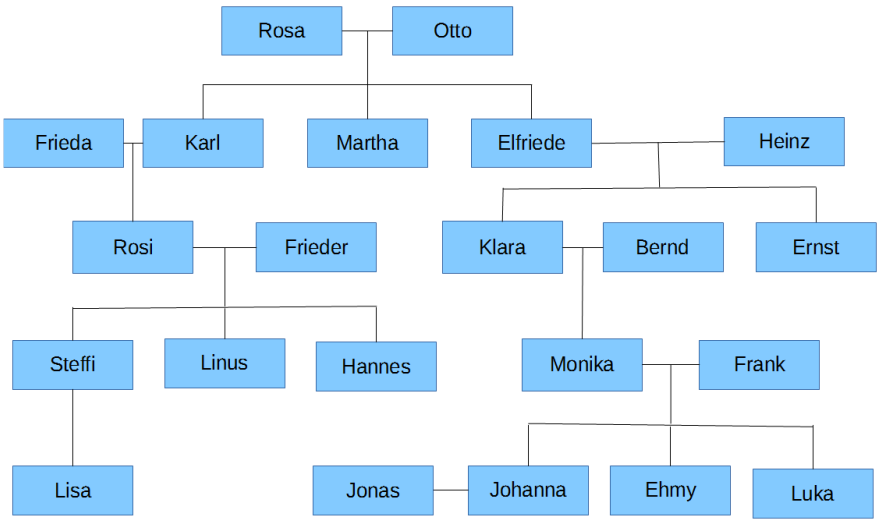
3.1.8.2. Wissens-Datenbanken mit PROLOG

PROLOG kann seine Vorteile besonders bei hierrarchischen, Baum-artigen, symbolischen und logisch verknüpften Daten ausspielen. Als Beispiel kann hier eine Datenbank zur Verwandschafts-Beziehungen gelten.

Bemerkungen / Legende:

- oben stehen die älteren Personen (Vorfahren)
- waagerechte Verbindungen zwischen zwei Personen sind Ehen
- die Kinder der Ehen stehen eine Schicht tiefer

Muster-Familienstammbaum



Umsetzung für SWI-PROLOG (z.B. auf IoStick)

Quelltext:

```
% Autor: Drews
% Datum: September 2015

% Datenbasis / Atome
weiblich(klara).
maennlich(bernd).
maennlich(ernst).
weiblich(monika).
maennlich(frank).
maennlich(jonas).
weiblich(johanna).
weiblich(ehmy).
maennlich(luka).

verheiratet(klara, bernd).
verheiratet(monika, frank).
verheiratet(johanna, jonas).

ist_elter_von(klara, monika).
ist_elter_von(bernd, monika).
ist_elter_von(monika, johanna).
ist_elter_von(frank, johanna).
ist_elter_von(monika, ehmy).
ist_elter_von(frank, ehmy).
ist_elter_von(monika, luka).
ist_elter_von(frank, luka).
```

Fortsetzung auf der nächsten Seite

```
% Regeln
ist_vater_von(X, Y):- maennlich(X), ist_elter_von(X, Y).
ist_opa_von(X,Y):- maennlich(X), ist_elter_von(X, Z),
                   ist_elter_von(Z, Y).
ist_nachkomme_von(X,Y):- ist_elter_von(Y, X) |
                         ist_elter_von(Y, Z), ist_nachkomme_von(X, Z).
```

diverse Regeln für Verwandtschaften:

Quelltext:

```
% Autor: Drews
% Datum: September 2015

weiblich(klara).
maennlich(bernd).
…
ist_nachkomme_von(X,Y):- ist_elter_von(Y, X) |
                         ist_elter_von(Y, Z), ist_nachkomme_von(X, Z).
```

```
verheiratet(X,Y):-verheiratet(Y,X).

ist_kind_von(X,Y):- ist_elter_von(Y,X).
ist_mutter_von(X,Y):- ist_elter_von(X,Y), weiblich(X).
ist_oma_von(X,Y):- ist_elter_von(X,Z), ist_elter_von(Z,Y), weiblich(X).

ist_bruder_von(X,Y):- ist_elter_von(Z,X), ist_elter_von(Z,Y),
                      X\=Y, maennlich(X).
ist_schwester_von(X,Y):- ist_elter_von(Z,X), ist_elter_von(Z,Y),
                      X\=Y, weiblich(X).

ist_schwiegermutter_von(X,Y):- ist_mutter_von(X,Z), verheiratet(Y,Z)
ist_schwiegervater_von(X,Y):- ist_vater_von(X,Z), verheiratet(Y,Z)
```

3.1.9. Datenbanken mit Snap!

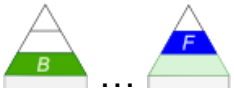
Hin und wieder taucht ein Artikel oder ähnliches zu diesem Thema auf. Praktisch ist Snap aber kein für uns gebräuchliches Datenbank-Management-System. Dadurch fliegt es an dieser Stelle aus der Kapitelfolge heraus.
Beim Programmieren von Datenbanken bietet es aber interessante Möglichkeiten. So kann man vor allem den Abfrage- / View-Bereich sehr schön abarbeiten.
Die Verbindung liegt hier zwischen einer visuell ansprechenden Block-orientierten Programmierung und der Nutzung einfacher SQL-Abfragen.

Die Block-Orientierung hilft sehr gut beim Zusammenstellen von Abfragen. Diese können immer direkt ausgeführt und geprüft werden. Übergibt man die Abfrage-Ergebnisse einer Variablen, dann können danach viele weitere Operationen darüber ausprobiert werden.
Super praktisch sind die einfachen Verknüpfungs-Möglichkeiten einfacher Programmiertechniken mit Datenbank-Abfragen. Da wird das Nutzungs-Potential von Programmierung und Datenbanken sehr schön sichtbar.
Auf einige Programmier-Möglichkeiten bezüglich von SQL-Abfragen gehen wir später noch etwas ausführlicher ein (→ 6.5. Datenbank-Programmierung mit Snap!).
Ein recht interessantes Projekt "Supermarkt" kann im Skript "MODROW: Der SQL-Supermarkt" (veröffentlicht: LOGIN 11/2015) weiterverfolgt werden. Natürlich bietet sich hierfür auch der Kurs "Informatik für Anfänger (…)" auf → https://open.sap.com an. In der dritten Woche wird in den Video's 5 und 6 auf Datenbanken eingegangen.
Abgelaufene Kurse können hier im Selbststudium nachverfolgt werden.

Links:
https://www.de.scratch-wiki.info/wiki/Publikation:Modrow2015-11-Im_Supermarkt_mit_SQLsnap
http://ddi-mod.uni-goettingen.de/Im%20Supermarkt.pdf (Skript)
http://snapextensions.uni-goettingen.de/ (direktes Arbeiten (online) in der Snap-Datenbank)

4. Datenbanken – fortgeschrittene Nutzung

4.0. Algebra über Relationen / Relationen-Kalküle



richtig **relationale Algebra** genannt
nicht richtig wäre "Relationen-Algebra", da diese zur abstrakten Algebra gehört und sich mit BOOLEschen Ausdrücken beschäftigt
im Datenbank-Jargon wird das manchchmal nicht sauber getrennt
unter einer Algebra versteht man ein Zusammensetzungs- bzw. Teilungs-System
Verknüpfung mathematischer Strukturen; Lehre von den Gleichungen
mehr Rechen-Vorschriften
Verallgemeinerung der Arithmetrik auf Zahlen und Zeichen, die Zahlen repräsentieren unter Verwendung definierter (arithmetrischer) Regeln / Vorschriften
z.B.: BOOLEsche Algebra für Wahrheits-Aussagen

Definition(en): Algebra
Eine Algebra ist eine (abgeschlossene) Menge von Regeln und Operationen zur Verarbeitung / Verknüpfung / Umwandlung von Zahlen, Variablen und anderen mathematischen Objekten.
.

Kalküle (bedeutet soviel wie Berechnungen bzw. Überlegungen)
System von Regeln aus der Logik und Mathematik zur systematischen Lösung von mathematischen Problemen
aus gegebenen Axiomen oder Aussagen werden weitere Aussagen gewonnen
ab- bzw. einschätzende Berechnungen (Wahrscheinlichkeits-behaftet; keine sichere / eindeutige Ergebnisse / Lösungen)
mehr Lösungs-Vorschriften / Sammlung von Regeln und deren Notation
z.B.: Notierungen mit logischen Symbolen

Tupel-Kalkül
Bereichs-Kalkül (Domänen-Kalkül)

auch nur Relationale Algebra genannt
betrachtet als Mengen-Element ein Tupel (Datensatz, Zeile)
je nach Anzahl der Spalten sprechen wir vom n-Tupel, also gilt für eine Tabelle mit 5 Spalten das 5-Tupel (auch: Quin-Tupel) als Mengen-Element
prozeduale Sprache

Definition(en): relationale Algebra / Relationen-Algebra

Die Relationen-Algebra ist eine (abgeschlossene) Menge von Regeln und Operationen zur Verarbeitung / Verknüpfung / Umwandlung von Tabellen (Relationen).
Das Ergebnis der Regeln und Operationen ist wiederum eine Tabelle.

.

Benennung der Tupel:

num. Schr.	Benennung
0-Tupel	leeres Tupel
1-Tupel	Singel
2-Tupel	Dupel; geord. Paar
3-Tupel	Tripel
4-Tupel	Quadrupel

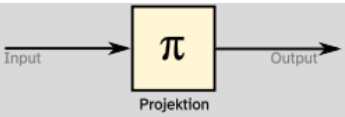
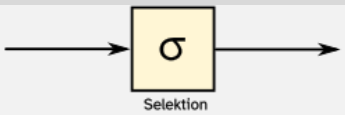
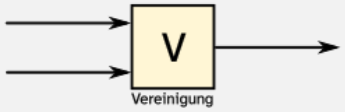
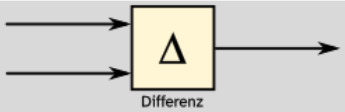
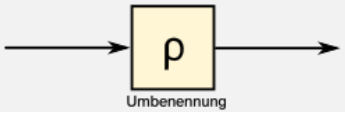
num. Schr.	Benennung
5-Tupel	Quintupel
6-Tupel	Sextupel
7-Tupel	Septupel
8-Tupel	Oktupel
9-Tupel	Nonupel

num. Schr.	Benennung
10-Tupel	Dekupel
20-Tupel	
100-Tupel	Centupel

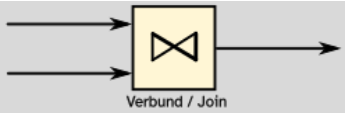
E.F. CODD (1923 - 2003) definierte Algebra für Relationen (1970), die auf 6 Operatoren basiert
die Zahl der Operatoren wird auch mal mit 5 oder 8 angegeben, insgesamt ist die Struktur und Zuordnung sehr undurchsichtig
aus einer oder mehrerer Tabellen (Relation(en)) entsteht immer wieder eine Tabelle (, die aber auch leer sein darf!)
oft werden 5 primitive Operatoren und drei komplexe Operatoren beschrieben worden
heute weiss man, dass bestimmte – wünschenswerte – Verknüpfungen mit den CODDschen Operatoren nicht realisert werden können
deshalb wurde die Relationen-Algebra später auch immer wieder erweitert
selbst CODD hat seine Algebra-Regeln und –Operatoren mehrfach korrigiert, dadurch fällt es schwer eine saubere Trennung vorzunehmen (Hierzu wäre eine tiefgreifende historische Aufarbeitung der Entwicklungs-Geschichte der CODDschen Algebra notwendig.).

#+#

(Mengen-)Operationen auf Relationen nach CODD

• **Projektion (π)**	unär	
• **Selektion (σ) (Restriktion)**	unär	
• **Kreuz-Produkt (x) (Kartesisches Produkt)**	binär	
• **Vereinigung**	binär	
• **Differenz (Δ)**	binär	
• **Umbenennung (ρ)**	unär	

weitere komplexe Operationen, die sich auf die Grund-Operationen zurückführen lassen:

• **Join (⋈) (Verbund)**	binär	
• **symmetrische Differenz**	binär	
• **Schnittmenge (Intersection)**	binär	
• **Division**	binär	

Symbole für spezielle Join's
??? ⋊ **Semi-Join** ⋉

• **Identität** **identische Relation**	gemeint ist hier die Objekt-Gleichheit, nicht die Gleichheit der Strukuren und Inhalte
• **Inversion** **Transponierung** **inverse Relation** **transponierte Relation**	Spiegeln einer Matrix / Tabelle entlang ihrer Diagonale
• **Relationen-Produkt**	

leere Relation ø
Relation, bei der alle Elemente miteinander in Beziehung stehen: 1 := A x A

minimale Menge der Operationen (Grundoperationen)

- **Projektion**
- **Selektion**
- **Kreuzprodukt**
- **Vereinigung**
- **Differenz**
- **Umbenennung**

mit diesen lassen sich alle Auswahlen treffen, sei entsprechen auch dem Algebra-System von CODD

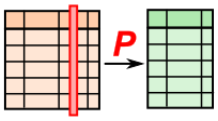
einige SQL-Operationen (z.B. GROUP BY oder HAVING) lassen sich durch algebrische Operationen nicht abbilden, deshalb werden manchmal einige Erweiterungen zur relationalen Algebra hinzugefügt → mehr theoretisches Problem

Erweiterung der klassischen Algebra um …:

- **Null-Werte**
- **Gruppierungs-Operatoren**
- **Aggregat-Funktionen**

Projektion

Ergebnis sind neue Tupel mit ausgewählten Attributen

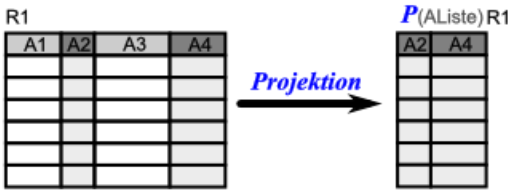


Projektionen werden in Termen mit **P** oder π verschlüsselt.

unäre Operation, die also nur mit einer Tabelle / Relation arbeitet

R1 sei eine Relation
mit den Attributen A1, A2 usw. usf. notieren wir dies in der Form:

R1={A1,A2,…,An}



Operatoren (Funktionen) notieren wir im Folgenden mit fetten Groß-Buchstaben oder griechischen Klein-Buchstaben bzw. den definierten Symbolen

R1

A	B	C
a	b	c
d	e	f
g	h	i

R2 = R1[A,C]

A	C
a	c
d	f
g	i

eine Projektion der Attribute A und C aus der nebenstehenden orangenen (Quell-)Tabelle R1 führt z.B. zur grünlichen (Ergebnis-)Tabelle R2, die nur die Attribute A und C enthält.

R2 = **P**(R1[A,C]) oder: R2 = π(R1[A,C])

In R3 möchten wir nur die Spalte B haben. Die passende Projektion sähe dann so aus:

od. z.B.:

R3= R1[B]

B
b
e
h

R3 = **P**(R1[B]) oder: R3 = π(R1[B])

In einigen Büchern usw. findet man auch die folgenden Notierung:

R3 = $\mathbf{P}_B$(R1) oder: R3 = π_B(R1)

oder eben für die mehr-attributige Projektion zu R2:

R2 = $\mathbf{P}_{A,C}$(R1) oder: R2 = $\pi_{A,C}$(R1)

Diese Notation ist bei einfacheren Ausdrücken übersichtlicher. Wenn aber die Anzahl der Attribute steigt und diese auch längere Namen haben, dann wird die Notation über mehrere Zeilen sehr schnell unübersichtlich. Wir bleiben deshalb bei obiger Notation im immer gleichgroß geschriebenen Symbolen.
Projektionen sind also praktisch immer Spalten-Auswahlen (Attribut-Beschränkung)

SQL-Repräsentation (→ 5.1.11.1. Projektion (Abbildung):):
SELECT *spalteX* **{** **,** *spalteY* **}** **FROM** *tabelle***;**

Hier ein **wichtiger Hinweis!**:
Der SQL-Befehl SELECT steht umgangssprachlich für "Auswahl" und nicht für "Selektion". Selektionen sind bestimmte algebraiische Operationen!

Aufgaben:

1. Geben Sie die folgenden Projektionen als Ergebnis-Tabellen an!

a) R4 = P(R1[A]) ***b)*** R5 = P(R1[B,C]) ***c)*** R6 = P(R2[C])

2. Gegeben sind die drei folgenden Tabellen.

R1

A	B	C
I	II	III
IV	V	VI
VII	VIII	IX

R2

D	E
z	y
x	w
v	u
t	s
r	q
p	o

R3

F	G	H	J
11	12	13	14
21	22	23	24

a) Geben Sie die folgenden Projektionen an!

aa) R4 soll die Spalten A und B von R1 enthalten

ab) die neue Relation R5 soll aus den Spalten D und E von R2 bestehen

ac) in die Relation R7 werden die Attribute H, G und F von R3 projiziert

b) Geben Sie für die Projektion aus der Teil-Aufgabe ac) die Ergebnis-Tabelle an!

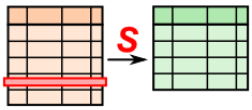
c) Handelt es sich bei den folgenden Ausdrücken um eine Projektion? Begründen Sie jeweils Ihre Entscheidung!

ca) R6 = P(R3[F,G]) ***cb)*** R5 = P(R1[B,A,A])

cc) R8 = P(R2[E],R3[G,J]) ***cd)*** R9 = P(R3[H,G,J,F])

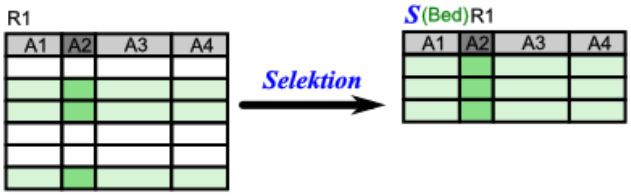
Selektion / Restriktion

auch als Auswahl (der Datensätze) verstanden



R WHERE Bedingung
Ergebnis beinhaltet alle Tupel, die die Bedingung(en) erfüllen

Selektionen werden in Termen durch
S oder σ (griech.: sigma) verschlüsselt.



R2 = **S**(R1[B=b]) oder: R2 = σ(R1[B=b])

Genauso könnte R3 durch andere Projektion gebildet werden

R3 = **S**(R1[A<g]) oder: R3 = σ(R1[A<g])
(bei Annahme geordneter Attribut-Werte von a nach j)

praktisch also Zeilen- bzw. Datensatz-Auswahl (bezüglich von Bedingungen) und Spalten-Auswahl (Einschränkung der Attribute)
die algebraiischen Ausdrücke werden ja üblicherweise von innen nach außen abgearbeitet

R1

A	B	C
a	b	c
d	e	f
g	b	h
i	j	b
b	b	b

od. z.B.:

R2= R1[A,B=b,C]

A	B	C
a	b	c
g	b	h
b	b	b

R3= R1[A<g,B,C]

A	B	C
a	b	c
d	e	f
b	b	b

SQL-Repräsentation (→ 5.1.11.2. Selektion (Auswahl):):
SELECT * FROM *tabelle* **WHERE *bedingung*;**
bedingung kann dabei ein einfacher oder komplexer – z.B. zusammengesetzter – (SQL-)Ausdruck sein

Hier ein **wichtiger Hinweis!**:
Der SQL-Befehl SELECT steht umgangssprachlich für "Auswahl" und nicht für "Selektion". Selektionen sind die gerade besprochenen algebraiische Operationen, bei denen eine Zeilen- / Datensatz-Auswahl erfolgt!

Aufgaben:

1. ***Prüfen Sie, ob die folgenden Selektionen zulässig sind und welche Ergebnisse entstehen!***

   a) S(R1[A>a]) ***b)*** S(R1[B=b,C=b]) ***c)*** S(R2[A<u,B=A,C>a])

2. ***Gegeben sind die drei folgenden Tabellen.***

R1

A	B	C
I	II	III
IV	V	VI
VII	VIII	IX

R2

D	E
z	y
x	w
v	u
t	s
r	q
p	o

R3

F	G	H	J
11	12	13	14
21	22	23	24

a) Geben Sie die folgenden Selektionen an!

aa) R4 soll die Datensätze aus R1 enthalten, bei denen C>II ist

ab) die neue Relation R5 soll aus R2 alle Datensätze enthalten, bei denen die Werte aus der Spalte E größer ist als die Werte aus E

ac) in die Relation R7 werden die Zeilen aus R3 herausgesucht, für die F gößer 11 und H kleiner gleich 23 ist

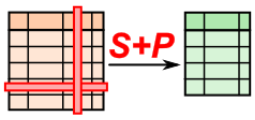
b) Geben Sie für die Selektion aus der Teil-Aufgabe ac) die Ergebnis-Tabelle an!

c) Handelt es sich bei den folgenden Ausdrücken um eine Selektion? Begründen Sie jeweils Ihre Entscheidung!

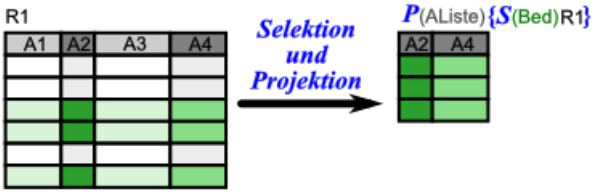
ca) R6 = S(R3[F=11]) ***cb)*** R5 = R(R1[B,A,A])

cc) R8 = S(R2[E>u],R3[G>10]) ***cd)*** R9 = S(R3[H=13,G=12,J=14,F=24])

Verbindung von Selektion und Projektion (bezüglich einer Tabelle)



serielle Abarbeitung von zwei unären Operation
kann als eine **unäre** Operation betrachtet werden



R2 = **P**(**S**(R1[B=b])[B,C]) oder: R2 = π(σ(R1[B=b])[B,C])

Es kommt zu einer Schachtelung, die zur Veranschaulichung durch verschiedene Farben gekennzeichnet wird:

R2 = **P**(**S**(R1[B=b])[B,C]) oder: R2 = π(σ(R1[B=b])[B,C])

Dabei ergibt die "blaue" Selektion eine imaginäre Zwischen-Tabelle Ri,

Ri = **S**(R1[B=b]) oder: Ri = σ(R1[B=b])

auf die dann die "rote" Projektion angewendet wird.

R2 = **P**(Ri[B,C]) oder: R2 = π(Ri[B,C])

R1

A	B	C
a	b	c
d	e	f
g	b	h
i	j	b
b	b	b

od. z.B.:

R2= R1[B=b,C]

B	C
b	c
b	h
b	b

R3= R[A<g]

A
a
d
b

Für die andere Ziel-Tabelle R3 kann die folgende Projektion genutzt werden:

R3 = **P**(**S**(R1[A<g])[A]) oder: R3 = π(σ(R1[A<g])[A])

(bei Annahme geordneter Attribut-Werte von a nach j)
sachlich ist die algebraische Abarbeitungs-Reihenfolge von Selektion und Projektion gleichwertig, so dass auch gilt:

R2 = **P**(**S**(R1[B,C][B=b]) oder: R2 = σ(π(R1[B,C] [B=b])

bzw.:
R3 = **P**(**S**(R1[A][A<g])) oder: R3 = σ(π(R1[A][A<g]))

An dieser Stelle zeigen wir noch einmal die Notation mit indizierten Attributen. Als Beispiel betrachten wir R2 von oben:

$R2 = P_{S_{R1B=b,C}}$

praktisch also Zeilen- bzw. Datensatz-Auswahl (bezüglich von Bedingungen) **und** Spalten-Auswahl (Einschränkung der Attribute)
die algebraischen Ausdrücke werden ja üblicherweise von innen nach außen abgearbeitet

SQL-Repräsentation (→ 5.1.11.2. Selektion (Auswahl):):
SELECT *spalteX* **{** **,** *spalteY* **}** **FROM** *tabelle* **WHERE** ***bedingung*****;**

die SQL-Systeme nehmen meist zuerst die Selektion und dann die Projektion vor, was bezogen auf das SQL-Statement bedeutet, dass zuerst der hintere Teil (z.B. WHERE) bearbeitet wird und dann der vordere (SELECT)

Aufgaben:

1.

2. Gegeben sind die drei folgenden Tabellen.

R1

A	B	C
I	II	III
IV	V	VI
VII	VIII	IX

R2

D	E
z	y
x	w
v	u
t	s
r	q
p	o

R3

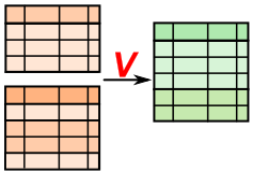
F	G	H	J
11	12	13	14
21	22	23	24

3.

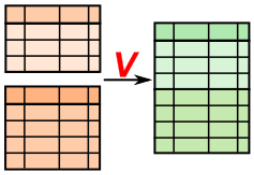
Kommen wir nun zu den **binäre**n Operationen. Diese Operationen arbeiten immer mit zwei Relationen. Dabei dürfen beide Relationen bei einzelnen Oprationen auch gleich sein. In einigen Fällen ist die Reihenfolge der Verwendung der beiden Tabellen wichtig (z.B. Differenz). Im Normal-Fall bearbeiten wir aber zwei unterschiedliche Tabellen. Ziel ist es, Informationen neu zu kombinieren oder zu erweitern.

Vereinigung / Union Join

R1 UNION R2
fügt die Zeilen von zwei Tabellen mit gleicher Spaltenzahl zu einer neuen Tabelle zusammen
die Namen der Spalten müssen nicht gleich sein, die Datentypen müssen aber identisch sein!



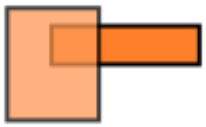
doppelte Zeilen werden automatisch unterdrückt, ist dies nicht gewünscht, dann muss UNION ALL benutzt werden



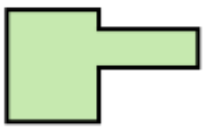
Die "einfache" Vereinigung enthält also praktisch auch noch Selektionen, was bei der vollständigen Vereinigung unterbleibt.

Mengen-mäßig betrachtet ist die Vereinigung die Menge, die alle Elemente aus den (Ursprungs-)Mengen M1 und M2 (orange). Die Ergebnis- bzw. Vereinigungsmenge ist grün ausgefüllt.

In der Mengen-Lehre ist eine Vereinigung, wie wir sie gerade mit UNION ALL vorgestellt haben, nicht möglich. In Mengen kommen – per Definition – Elemente nur einmalig vor.



Original- / Ursprungs-Mengen



Vereinigungs-Menge

Bei Relationen müssen die Datensätze (Zeilen, Tupel) als Element der Menge betrachtet werden. Entsprechend tauchen in der Ergebnis-"Menge" nun alle Datensätze einmal auf. Die doppelten (rot gekenzeichnet) aus der Menge 2 (hier R2) werden nicht noch einmal in die Ergebnis-Menge übertragen.

R1

A	B	C
a	b	c
d	e	f
g	h	i

R2

D	E	F
d	e	f
b	c	d
g	h	i
j	k	l
m	n	o

R3= R1 V R2

A	B	C
a	b	c
d	e	f
g	h	i
b	c	d
j	k	l
m	n	o

SQL-Repräsentation:
SELECT * FROM tabelle1 UNION SELECT * FROM tabelle2;

Aufgaben:

1.

2. Gegeben sind die folgenden Tabellen mit gleichen Datentypen in allen Spalten. Erstellen Sie die Vereinigungen von a) R1 und R2 als R4 ; b) R2 und R1 als R5; c) R2 und R3 als R6 und von d) R3 und R1 als R7!

R1

A	B	C
1	2	3
4	5	6
7	8	9
10	11	12

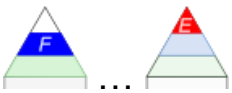
R2

D	E	F
1	2	3
2	3	4
3	4	5
4	5	6
3	8	9
10	12	13

R3

G	H	I	J
1	2	3	4
4	5	6	7
7	8	9	10

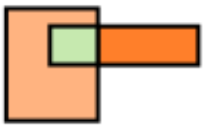
3. Erstellen Sie die vollständige Vereinigung (UNION ALL) für die Tabellen R2 und R1! (Probleme mit Schlüsseln übergehen wir hier!)



Schnitt(menge) / Durchschnitt / Intersection

R1 INTERSECT R2
Ergebnis sind nur die Tupel, die in beiden Tabellen vorkommen

In der Relationen-Algebra betrachten wir ja die Tupel als Mengen-Elemente. Somit müssen beim Ermitteln der Schnitt-Menge die Tabellen gleichviele Spalten enthalten.
Es werden alle Tupel ausgewählt, die sowohl so in der Relation R1 als auch in der Relation R2 vorkommen.


Schnitt(-Menge)

R1

A	B	C
a	b	c
d	**e**	**f**
g	**h**	**i**

R2

D	E	F
d	**e**	**f**
b	c	d
g	**h**	**i**
j	k	l
m	n	o

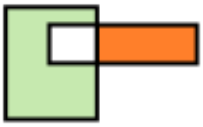
R3= R1 ∧ R2

A	B	C
d	e	f
g	h	i

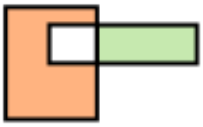
SQL-Repräsentation:
SELECT * FROM tabelle1 INTERSECT SELECT * FROM tabelle2;
SELECT * FROM tabelle1 WHERE spalte IN (SELECT spalte FROM tabelle2);

Differenz / Minus

R1 MINUS R2
Ergebnis sind alle Tupel aus R1, die nicht in R2 vorkommen
(aus Tabelle R1 werden alle Tupel aus R2 gelöscht)



Differenz(-Menge)
(M1 – M2)



Differenz(-Menge)
nach Operanden-Tausch
(M2 – M1)

R1

A	B	C
a	b	c
d	**e**	**f**
g	**h**	**i**

R2

D	E	F
d	**e**	**f**
b	c	d
g	**h**	**i**
j	k	l
m	n	o

R3 = R1 \ R2

A	B	C
a	b	c

R4 = R2 \ R1

A	B	C
b	c	d
j	k	l
m	n	o

SQL-Repräsentation:
SELECT * FROM tabelle1 MINUS SELECT * FROM tabelle2;
SELECT * FROM tabelle1 WHERE spalte NOT IN (SELECT spalte FROM tabelle2);

??? symetrische Differenz

Division / Quotient

R1 DEVIDEBY R2
Ergebnis enthält alle Attribute aus R1, die nicht in R2 vorkommen und die Tupel aus R1, die hinsichtlich eines gemeinsamen / gleichen Attributs-Wertes auch in R2 vorkommen

Nicht zu verwechseln mit der nummerischen Division von zwei Zahlen. Diese kann in SQL-Statement's mit einem / erledigt werden.
Hier ist die Division zweier Mengen bzw. der Quotient beider gemeint.

abgeleitete Operation:
R1 / R2 = P(R1) – P((P(R1) x R2) – R1)

gesucht sind bei der Division die (Teil-)Datensätzen aus R1, die für jeden Datensatz in R2 einen Eintrag haben

R1

A	B	C	D
a	b	c	d
e	**f**	g	**h**
i	j	k	l
a	m	c	n
e	**o**	g	**p**
i	q	k	r

R2

B	D
f	**h**
o	**p**

R3= R1 / R2

A	C
e	g

SQL-Repräsentation:
SELECT * FROM tabelle1 DEVIDEBY SELECT * FROM tabelle2;

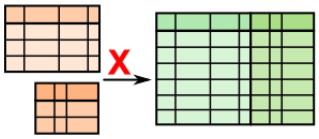
Aufgaben:

1.
2.
3.

Kreuz-Produkt / kartesisches Produkt

auch Cross Join

Kombination jeder Zeile der einen Tabelle (**R1**) mit jeder Zeile der anderen (**R2**)



Ergebnis besteht aus allen möglichen Tupel-Kombinationen von **R1** und **R2**
es entstehen sehr große Tabellen (neue_Zeilenzahl = ZeilenzahlTab1 * Zeilenzahl_Tab2)
Verwendung meist fraglich → ev. für DataMining
oft als Daten-Quelle für weitere Operationen benutzt

die etwas dunkleren grünen Felder in der Ergebnis-Tabelle R3 sollen nur die Wiederverwendung der ursprünglichen Datensätze in der Kombination zeigen

R1

A	B	C	D
a	b	c	d
e	f	g	h
i	j	k	l

R2

E	F	G
1	2	3
4	5	6

R3 = R1 X R2

A	B	C	D	E	F	G
a	b	c	d	1	2	3
e	f	g	h	1	2	3
i	j	k	l	1	2	3
a	b	c	d	4	5	6
e	f	g	h	4	5	6
i	j	k	l	4	5	6

SQL-Repräsentation:
SELECT * FROM *tabelle1* **CROSS JOIN** *tabelle2*;
SELECT * FROM *tabelle1, tabelle2*;

Aufgaben:

1. *In den Tabellen wurden vom Datenbank-Administrator die Spalten A und E als Schlüssel festgelegt. Welche Spalte(n) sind in der Tabelle R3 Schlüssel? Erklären Sie!*
2. *Prüfen Sie, ob R4 = R2 X R1 gleich der Tabelle R3 ist!*
3. *Was ändert sich in der Tabelle R3, wenn ein ev. unbedachter Datenbank-Administrator*
   a) die Spalten D und G als Schlüssel festgelegt hat
   b) die Schlüsselwerte in den Schlüsselspalten A und E gleich belegt hat.
4. *Gegeben sind die folgenden drei Tabellen. Stellen Sie die kartesischen Produkte für R1 X R3, R2 X R3 und R1 X R2 X R3 auf!*

R1

A	B	C
I	II	III
IV	V	VI
VII	VIII	IX

R2

D	E
z	y
x	w
v	u
t	s
r	q
p	o

R3

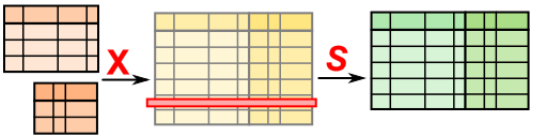
F	G	H	J
11	12	13	14
21	22	23	24

3.

Verbund / Verknüpfung (Join)

hintereinander ausgeführte Operationen "kartesisches Produkt" und "Selektion"

aus der Kreuz(-produkt)-Tabelle wird durch Selektion eine Auswahl von Datensätzen getätigt



als Sonderfall des Verbundes / Join's gilt der Equal-Join (Gleich-Verbund; → innerer / äquivalenter Verbund / Inner Join / Equivalent Join), bei dem in den Datensätzen der Kreuz-(produkt)-Tabelle nach bestimmten übereinstimmenden Feldern gesucht wird und diese dann eleminiert werden

R3 ist Kreuz-Produkt als Zwischen-Ergebnis und vorrangig gelblich dargestellt

R1

A	B	C	D
a	b	c	d
e	f	g	h
i	j	k	l

R2

E	F	G
1	2	3
4	5	6

R3 = R1 X R2

A	B	C	D	E	F	G
a	b	c	d	**1**	2	3
e	f	g	h	**1**	2	3
i	j	k	l	**1**	2	3
a	b	c	d	4	5	6
e	f	g	h	4	5	6
i	j	k	l	4	5	6

Selektion dahingehend, dass in E ein Wert z.B. kleiner als 3 (passende Werte sind rot und fett gekennzeichnet, wobei eben nur der 2. Datensatz die Bedingung erfüllt und damit aussortiert wird (rötlich unterlegt))

R4 = R3[A,B,C,D,E<3,F,G]

A	B	C	D	E	F	G
a	b	c	d	4	5	6
e	f	g	h	4	5	6
i	j	k	l	4	5	6

Definition(en): Verbund / Join
Ein Verbund (Join) ist eine (Datenbank-)Operation, bei der zwei oder mehr Tabellen miteinander so verknüpft werden, dass eine neue Relation (Tabelle) entsteht.
Ein Join ist eine Kombination der relationalen Operationen "kartesisches Produkt", "Selektion" und "Projektion".

Aufgaben:

1. *Erstellen Sie aus R1 und R2 eine Relation R5, die als kartesisches Produkt keine Datensätze enthält, bei denen F mit 2 belegt ist!*
2. *Gibt es eine Verbund-Relation aus R1 und R2, die in G Werte über 8 enthält? Erläutern Sie Ihre Antwort!*

innerer / äquivalenter Verbund / Inner Join / Equivalent Join

auch Equi-Join; Sonderfall des "normalen" Verbundes (Join, → Verbund / Verknüpfung (Join))

R1 **INNER JOIN** R2
Ergebnis sind alle möglichen Kombinationen der Tupel aus beiden Tabellen, die bestimmte Attribut(-Werte) gleich haben
(Verknüpfung über Fremdschlüssel-Primärschlüssel-Beziehung

R3 ist Kreuz-Produkt als Zwischen-Ergebnis und vorrangig gelblich dargestellt

R1

A	B	C	D
a	b	c	d
e	f	g	h
i	j	k	l

R2

E	F	G
x	b	c
y	f	c
z	j	c

R3 = R1 X R2

A	B	C	D	E	F	G
a	**b**	c	d	x	**b**	c
e	f	g	h	x	b	c
i	j	k	l	x	b	c
a	b	c	d	y	f	c
e	**f**	g	h	y	**f**	c
i	j	k	l	y	f	c
a	b	c	d	z	j	c
e	f	g	h	z	j	c
i	**j**	k	l	z	**j**	c

jetzt Selektion dahingehend, dass der Wert in B gleich dem Wert in F ist (passende Werte sind rot und fett gekennzeichnet, womit 3 Datensätze eleminiert werden (rötlich unterlegt)

R4 = R3[A,B=F,C,D,E,F,G]

A	B	C	D	E	F	G
a	b	c	d	x	b	c
e	f	g	h	y	f	c
i	j	k	l	z	j	c

SQL-Repräsentation:
SELECT * FROM tabelle1 **INNER JOIN** tabelle2 ON tabelle1.spalte = tabelle2.spalte;
ersatzweise auch möglich / Entsprechung:
SELECT * FROM tabelle1 **JOIN** tabelle2 ON tabelle1.spalte **=** tabelle2.spalte;
SELECT * FROM tabelle1, tabelle2 **WHERE** tabelle1.spalte **=** tabelle2.spalte;

Aufgaben:

1. *Gesucht ist eine Relation R6, die sich über einen inneren Verbund bildet, wobei in C und G auf gleiche Werte geprüft werden soll!*
2. *Wie sieht das Ergebnis für den Fall aus, dass das Kreuz-Produkt aus R2 X R1 gebildet wird und dann die Bedingung (C und G gleich) angewendet wird? Erläutern Sie Ihr Ergebnis!*

natürlicher Verbund / Natural Join

???: R1 JOIN R2
??? Ergebnis sind alle möglichen Kombinationen der Tupel aus beiden Tabellen, die bestimmte Attribut(-Werte) gleich haben

Natural Join ist ein Equi-Join (innerer Verbund) mit einer nachfolgenden Projektion (Spaltenausblendung)
natürlicher Verbund ist kommutativ und assoziativ
bei der Abrarbeitung können diese Eigenschaften genutzt werden, um die zu bearbeitende Datenmenge klein zu halten

(Verknüpfung über Fremschlüssel-Primärschlüssel-Beziehung mit Ausblendung () äquivalenter Spalten (z..B. Fremdschlüssel-Primärschlüssel-Verbindungen ())

R3 ist Kreuz-Produkt als Zwischen-Ergebnis und vorrangig gelblich dargestellt

R1

A	B	C	D
a	b	c	d
e	f	g	h
i	j	k	l

R2

E	F	G
a	b	c
i	j	k

R3 = R1 X R2

A	B	C	D	E	F	G
a	b	c	d	a	b	c
e	f	g	h	a	b	c
i	j	k	l	a	b	c
a	b	c	d	i	j	k
e	f	g	h	i	j	k
i	j	k	l	i	j	k

jetzt Projektion unter Ausblendung der zu A identischen Spalte E (rötlich unterlegt)

R4 = R3[A=E,B,C,D,F,G]

A	B	C	D	F	G
a	b	c	d	b	c
i	j	k	l	j	k

SQL-Repräsentation:
SELECT tabelle1.*, tabelle2.spalte INNER JOIN tabelle2 ON (tabelle1.spalteX = tabelle2.spalteX AND tabelle1.spalteY = tabelle2.spalteY);

Aufgaben:

1.
2.
3.

voller Inklusionsverbund / Full (Outer) Join

???: R1 FULL R2
Kombination aus linkem und rechtem Inklusionsverbund, leere Felder werden mit NULL belegt

R1

A	B	C	D
a	b	c	d
e	f	g	h
i	j	k	l

R2

E	F	G
a	2	3
i	5	6

R3 = (R1[A] = R2[E]), R1[B,C,D], R2[F,G]

A	B	C	D	F	G
a	b	c	d	2	3
e	f	g	h	NULL	NULL
i	j	k	l	5	6

SQL-Repräsentation:
SELECT * FROM tabelle1 FULL JOIN tabelle2 ON (tabelle1.spalteX = tabelle2.spalteX AND tabelle1.spalteY = tabelle2.spalteY);

/ Semi Join

???: R1 SEMI R2
Kombination aus einem natürlichen Verbund und einer anschließenden Projektion auf die Attribute der ersten Tabelle

R1

A	B	C	D
a	b	c	d
e	f	g	h
i	j	k	l

R2

E	F	G
a	2	3
i	5	6

R3 = (R1[A] = R2[E]), R1[B,C,D]

A	B	C	D
a	b	c	d
i	j	k	l

SQL-Repräsentation:
SELECT tabelle1.* FROM tabelle1 INNER JOIN tabelle2 ON (tabelle1.spalteX = tabelle2.spalteX AND tabelle1.spalteY = tabelle2.spalteY);

/ Theta Join / Non-Equivalent Join

???: R1 FULL R2
Verallgemeinerung des inneren Verbundes
neben der Gleichheit → innerer Verbund können auch andere Vergleiche / Formeln

SQL-Repräsentation:
SELECT * FROM tabelle1 INNER JOIN tabelle2 ON (tabelle1.spalteX [= | < | <= | > | >= | <>] tabelle2.spalteX AND tabelle1.spalteY = tabelle2.spalteY);

/ Self Join

???: R1 SELF R2
beliebiger Verbund einer Tabelle mit sich selbst

SQL-Repräsentation:
SELECT alias1.spalteX, alias1.spalteY, alias2.spalteX FROM tabelle alias1, tabelle alias2 WHERE alias1.spalteZ = alais2.spalteX (+);

linker Inklusionsverbund / Left (Outer) Join

???: R1 LEFT R2
die (ausgewählten) Daten aus der ersten (linken) Tabelle werden mit ev. vorhandenen Daten aus der zweiten (rechten) Tabelle ergänzt, leere Felder werden mit NULL belegt

SQL-Repräsentation:
SELECT * FROM tabelle1 LEFT JOIN tabelle2 ON (tabelle1.spalteX = tabelle2.spalteX AND tabelle1.spalteY = tabelle2.spalteY);
(SELECT * FROM tabelle1, tabelle2 WHERE (tabelle1.spalteX = tabelle2.spalteX (+) AND tabelle1.spalteY = tabelle2.spalteY (+));)

mit (+) sind die Spalten gekennzeichnet, bei denen Platz für NULL-Werte freigehalten werden muss (auf der rechten Seite)

rechter Inklusionsverbund / Right (Outer) Join

???: R1 RIGHT R2
die (ausgewählten) Daten aus der zweiten (rechten) Tabelle werden mit ev. vorhandenen Daten aus der ersten (linken) Tabelle ergänzt, leere Felder werden mit NULL belegt

SQL-Repräsentation:
SELECT * FROM tabelle1 RIGHT JOIN tabelle2 ON (tabelle1.spalteX = tabelle2.spalteX AND tabelle1.spalteY = tabelle2.spalteY);
(SELECT * FROM tabelle1, tabelle2 WHERE (tabelle1.spalteX (+) = tabelle2.spalteX AND tabelle1.spalteY (+) = tabelle2.spalteY);)

mit (+) sind die Spalten gekennzeichnet, bei denen Platz für NULL-Werte freigehalten werden muss (auf der linken Seite)

symmetrische Differenz

A	B	C	D
a	b	c	d
e	f	g	h
i	j	k	l

E	F	G
1	2	3
4	5	6

A	**B**	**C**	**D**	**E**	**F**	**G**
a	b	c	d	1	2	3
e	f	g	h	1	2	3
i	j	k	l	1	2	3
a	b	c	d	4	5	6
e	f	g	h	4	5	6
i	j	k	l	4	5	6

durch die referenzielle Integrität wird abgesichert, das zu jedem Fremd-Schlüssel auch ein Primär-Schlüssel einer anderen Tabelle vorhanden ist (/ passt)

Methoden der analytischen Informationssysteme

- **Online Analytical Processing (OLAP)** — Hypothesen-basierte Analyse von Daten (Bestätigung oder Ablehnung von vordefineirten Hypothesen)
- **Data-Mining** — Suche nach Zusammenhängen (Querverbindungen) zwischen Daten; Erzeugen neuer Daten(-Kombinationen); Ausgraben versteckter Daten
- **Data-Warehouse** — Zusammenfassung von Daten in einem einheitlichen Format (Datenauslage, Datenbereitstellung, Datenausbreitung)

kleine "Regel-Sammlung" zur Relationen-Algebra

"Regel"	Grob-Funktion
Projektion ::= Relation - Attribut(e) = TeilRelation Projektion ::= TeilTabelle = Tabelle - Spalten	Spalten-Auswahl
Selektion ::= Relation = TeilRelation1 + TeilRelation2 Selektion ::= Tabelle - Zeilen	Zeilen-Auswahl
Produkt = Relation X Relation KartesischesProdukt = Tabelle X Tabelle	Multiplikation
Join ::= Produkt + Selektion Verbund ::= Kartesisches Produkt - Zeilen	
EquiJoin = Join + Selektion GleichVerbund ::= KartesischesProdukt - Zeilen	
NaturalJoin = EquiJoin + Projektion natürlicherVerbund ::= GleichVerbund - Spalte(n)	

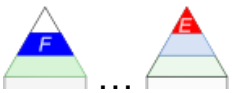
algebraische Darstellung der Relationen-Eigenschaften

R ist reflexiv $\iff$ $I \subseteq R$
R ist irreflesiv $\iff$ $I \cap R = \emptyset$
R ist symmetrisch $\iff$ $R = R^T$
R ist asymmetrisch $\iff$ $R \cap R^T = \emptyset$
R ist antisymmetrisch $\iff$ $R \cap R^T \subseteq I$
R ist transitiv $\iff$ $R \circ R \subseteq R$, also $R^2 \subseteq R$

Rechnen-Regeln der Relationen-Algebra

$R^{TT} = R$
$Q \circ (R \circ S) = (Q \circ R) \circ S$ (Assoziativität des Relations-Produktes)
$I \circ R = R \circ I = R$ (I als neutrales Element des Relations-Produkts)
$(R \circ S)^T = S^T \circ R^T$ ()

$R^T = R^{-1}$, wenn gilt: $(b,a) \in R^T \iff (a,b) \in R$



Wiederholung / Rückgriff

algebraïsche Elemente aus der Mengenlehre

Benennung	**Symbolik**	**Bedeutung / Beschreibung**
Leere Menge	{} Ø	Menge enthält keine Elemente
Menge	Großbuchstaben: A, B, …, M, N, … oder mit Zusatzziffern bzw. Indizies: M1, M2, … ; M_1, M_2, … für besondere Beziehungen zwischen Mengen: M_A, M_B, …	
Komplement	$\bar{A}_B$ $\bar{A}_B$	:= {x | x ∉ A ∧ x ∈ B}
Mächtigkeit	|M|	Anzahl der Elemente in der Menge
Potenz-Menge	$\mathcal{P}$(A)	:= {X | X ⊆ A}
Gleichheit	A = B	:⇔ $\forall x\,(x \in A \Leftrightarrow x \in B)$
Teil Teilmenge	A ⊆ B	:⇔ $\forall x\,(x \in A \Rightarrow x \in B)$
Disjunkte Mengen	A ∩ B = Ø	Mengen ohne gemeinsame Elemente
Vereinigung	A U B	:= {x | x ∈ A ∨ x ∈ B}
Schnitt Durchschnitt	A ∩ B	:= {x | x ∈ A ∧ x ∈ B}
Differenz	A \ B	:= {x | x ∈ A ∧ x ∉ B}
Symmetrische Differenz	A Δ B	:= {(x | x ∈ A ∧ x ∉ B) ∨ (x | x ∉ A ∧ x ∈ B)}
Kartesisches Produkt	A x B	:= {(a,b) | a ∈ A ∧ b ∈ B}

Beispiel:

Tabellen-Bestand (Datenbank):

Personal

PID	Name	Vorname	Ort	Vorgesetzter	Gehalt
1	Ahrend	Karl	Glücksort	3	2600
2	Koch	Anton	Adorf	9	2300
3	Zander	Michael	Hochberg	NULL	5000
4	Mayer	Frieder	Niederhagen	9	3100
5	Dietrich	Johannes	Brandhagen	9	2900
6	Meier	Claudia	Neu Schönau	3	1800
7	Bauer	Regina	Hochberg	9	2000
8	Tietze	Claudia	Brandhagen	9	2500
9	Lehmann	Monika	Lussow	NULL	5400
10	Schulz	Manne	Niederschönau	9	3400

Kunde

KID	Name	PLZ	Ort	Straße
1	Autohaus Günther	12345	Adorf	Hauptstr. 27
2	Autoverwertung 2000	23456	Niederburg	Kirchenweg 7
3	Autohandel Diamant	55555	Gleichstätt	Lessingstr. 4
4	Gebrauchtwagen 24/7	23456	Niederburg	Pappel-Allee 124
5	Zander-Gebrauchte	99999	Pechhagen	Nordstr. 23
6	Meier-Autohaus Adorf	12345	Adorf	Waldweg 9
7	Fahrzeuge Schmidt	12321	Mittelhausen	Goethe-Platz 2
8	Meier-Autohaus Gleichstätt	55555	Gleichstätt	Pechhagener Weg 33

Auftrag

AID	Auftragsdatum	KID	PID
1	03.06.2012	1	4
2	20.12.2012	5	1
3	30.08.2013	8	4
4	01.02.2014	2	1
5	23.11.2014	3	10
6	02.01.2015	1	4

5. SQL – die Datenbank-Sprache

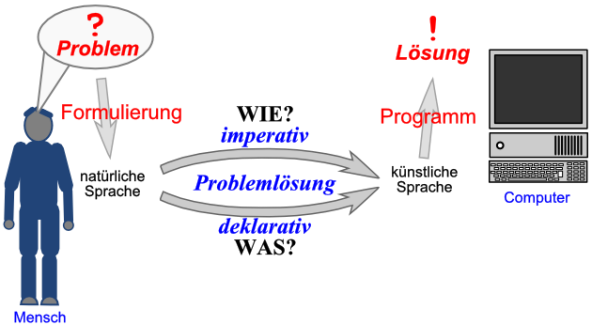
Structured Query Language
textorientierte Datenbank-Sprache zur Erzeugung und Manipulation von Tabellen und / oder Daten

Daten-Abfrage- und –Definitions--Sprache

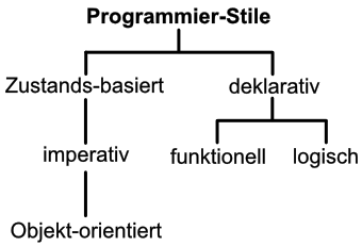
zur Verwaltung von Daten, Datenbanken und Zugriffsrechten und zur Datenmanipulation gedacht
sollte möglichst auch von weniger Programmier-affinien Personen nutzbar sein
herausgekommen ist die wohl leichteste Datenbank-Anfragesprache
für englisch-sprechende Personen mit geringen Computer- und Programmier-Kenntnissen lesbar und verständlich
mit wenig Übung anwendbar

gilt auch very high abstracly programming language (sehr hoch abstrahierende / abstrahierte Programmiersprache)

deklarative Sprache, d.h. wir müssen nicht spezifizieren, wie etwas gemacht werden soll – also den Algorithmus umsetzen – sondern wir sagen, **was** das Ergebnis sein soll



Bedeutung von Kenntnissen in SQL wird auf dem Job-Markt hoch eingestuft. Praktisch liegt die Häufigkeit von Bedarfen für einen angebotenen Job an zweiter Stelle direkt hinter JAVA.
Das liegt auch daran, dass viele Programmiersprachen SQL eingebettet benutzen. Man kommt also praktisch als Informatiker oder Programmierer nicht mehr an SQL und Datenbanken vorbei.



Übersicht über Programmier-Stile

Vorteile des deklarativen Konzepts:
- einfach zu erlernen, oft auch für weniger professionelle Nutzer gut nutzbar
- Computersystem selbst optimiert die Umsetzung und Ausführung des Programms → sind sehr schnell / effektiv
- i.A. wesentlich kleinerer Quelltext
-

Nachteile des deklarativen Konzepts:
- Daten-Zusammenhänge gehen für den außenstehenden Nutzer ev. verloren
-

DQL Data Query Language (Daten-Abfrage-Sprache)

DDL Data Description Language / Data Definition Language (Daten-Definitions-Sprache)
DML Data Manipulation Language (Daten-Manipilations-Sprache / Daten-Zugriffs-Sprache)
DCL Data Control Language (Daten-Kontroll-Sprache)
DRL Data Retrieval Language (Daten--Sprache)

Anfragen werden in sogenannte QEP's (sprich: quecks) zerlegt. Die **Q**uery **E**xecution **P**lan's stellen optimierte Arbeits-Schema's dar, umkomplxe Aufgaben überhaupt sinnvoll und zeitrelevant zu lösen

erste Version 1986 (ANSI-Standard)

SQL89

zwei mögliche Ebenen des umgesetzten / realisierten Sprachumfangs

- **Level 1**
- **Level 2**

SQL92 / SQL2

vier mögliche Ebenen des umgesetzten / realisierten Sprachumfangs

- **Entry Level**
- **Transitional**
- **Intermediate Level**
- **Full Level**

SQL2008 / SQL3

aktuelle Version SQL:2008 ISO/IEC 9075:2008

Ebene	typische SQL-Anweisungen (Datendefinitionen)	
extern	CREATE VIEW DROP VIEW	
konzeptionell	CREATE TABLE ALTER TABLE DROP TABLE CREATE DOMAIN ALTER DOMAIN DROP DOMAIN	
intern	CREATE INDEX ALTER INDEX DROP INDEX	

böse Frage zwischendurch:

Woher kommt die Unterscheidung der Ebenen in der obigen Tabelle und was beinhalten die Ebenen?

Alternativ zu diesem Skript kann man auch den SQL-Kurs (→ https://www.w3schools.com/sql/default.asp) auf der Entwicker-Lernplattform w3schools.com nutzen. Der Kurs ist teilweise übersetzt und führt durch die wesentlichen Inhalte. Die Test's sind anspruchsvoll, aber leider auch immer wieder mal in englischer Sprache. Am Schluß kann man auch ein Zertifikat erreichen.
w3schools wird auch von anderen Lernplattformen (z.B. AppCamps.de) und Tutorial's benutzt, um die praktischen Übungen zu realisieren. Auch über diese Wege ist ein anders orientierter Lern-Zugang möglich.
Im Abschnitt → 5.2. SQL lernen bei w3schools.comgehen wir auf einige Aspekte der Arbeit mit w3schools.com ein.

Daten-Typen in SQL

- **Integer** / **Int** — ganze Zahl
- **Smallint** — je nach DBMS verkleinerter Integer-Bereich
- **Numeric(*gs*,*ns*)** — Dezimal- / Gleitkomma-Zahl mit genau ***gs*** Stellen und mit ***ns*** Nachkomma-Stellen
- **Decimal(*ms*,*ns*)** — Dezimal- / Gleitkomma-Zahl mit mindestens ***ms*** Stellen und mit ***ns*** Nachkomma-Stellen
- **Real** — Dezimal- / Gleitkomma-Zahl mit sogenannter einfacher Genauigkeit
- **Double Precision** — Dezimal- / Gleitkomma-Zahl mit sogenannter doppelter / höherer Genauigkeit
- **Float(*st*)** — Dezimal- / Gleitkomma-Zahl mit einer Genauigkeit von ***st*** Stellen
- **Character** / **Char** — einzelnes Zeichen
- **Character(*n*)** / **Char(*n*)** — Zeichenkette aus genau ***n*** Zeichen
- **Varchar(*n*)** — Zeichenkette mit maximal ***n*** Zeichen
- **Boolean** — Wahrheitswert
- **Date** — Kalender-Datum
- **Time** — (Uhr-)Zeit

Links:

https://www.w3schools.com/sql/default.asp (Lern-Plattform / relativ gut teil-übersetztes Tutorial zu SQL mit Test's und Zertifikat)

https://appcamps.de/ (freie Lern-Plattform für Schulen zu vorwiegend informatischen Themen (durchgehend deutsch))

5.1. wichtige SQL-Anweisungen

Abarbeitung der wichtigsten in der üblichen Gebrauchs-Reihenfolge
z.T. Gruppierung nach theoretischen Konzepten dieses Skriptes
von einfachen Nutzungen zu immer spezielleren
keine Unterscheidung nach Sprachteil-Gruppen
nicht vollständig, nur die üblichen Schul-relevanten Anweisungen werden besprochen
Hier geht es noch viel weniger um das Er- oder Auswendig-Lernen einer Sprache, als beim Erlernen einer Programmiersprache im schulischen Kontext.
Elementare Anweisungen mit typischen Anwendungen.

Hier wird jetzt vielleicht auch deutlich, warum wir uns immer die SQL-Repräsentationen von irgendwelchen Operationen auf unseren Datenbanken angesehen haben. Da haben wir schon ein Gefühl für die Ausdrucks-Stärke von SQL-Anweisungen bekommen.
Besonders praktisch ist die quasi parallel nutzbare graphische Anwendung in Datenbanken, wie ACCESS und BASE und die Text-basierte Notation der Anweisungen in klassischen SQL-Datenbanken (wie z.B. SQLite →). Vielfach kann man die Arbeits-Veriante wählen, die einem am Meisten zusagt oder die das Problem am Schnellsten lösen hilft.

Verbindung zum Datenbanksystem / Datenbank-Konsole
z.B. für eine mySQL-Installation:

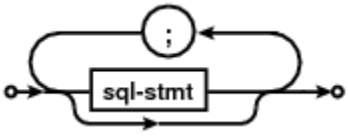
```
> mysql –h localhost –u root –p
```

Legende / Schreib-Konventionen:

Notation	Beschreibung / gemeintes Objekt	Beispiel(e)	Hinweise / Bemerkungen
fett	SQL-Anweisungen oder Syntax-Elemente Terminal	**CREATE** … **;**	
fett, rot	Sytax-Strukturen Wiederholung Alternative Option	**{ }** **|** **[ ]**	
fett, kursiv	Platzhalter für SQL- bzw. untergeordnete Strukturen Nichtterminal		
kursiv	Platzhalter für Elemente der Datenbank Variable		
fett, blau	besondere Strukturen, …		

Die Syntax-Diagramme (nebenstehender Stil) sind von der Webseite von SQLite (→ https://www.sqlite.org/syntaxdiagrams.html). Im folgenden nur noch mit der verlinkten Kurz-Referenz /###/ gekennzeichnet.
Was bedeutet ein solches Diagramm? Wie liest man es?
Oben steht der Name des SQL-Ausdrucks. Dieser heißt hier **sql-stmt-list**, was für SQL-Anweisungs-Liste steht. Beim

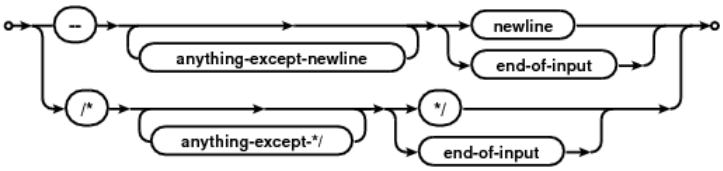
sql-stmt-list:



/###/

Lesen beginnt man am linken Punkt, den man sich als Verbinder / Kontaktstelle vorstellen kann.
Man folgt dann der Pfeilrichtung. Hier gibt zwei Möglichkeiten. Entweder man geht zu einem **sql-stmt** (SQL-Statement (SQL-Anweisung)) oder direkt zum Endpunkt rechts. Hier folgt dann ein anderes Syntax-Diagramm mit einem linken Verbinder.
Da sql-stmt von einem Rechteck umrandet ist, handelt es sich hier um einen Namen für ein weiteres folgendes Syntax-Diagramm mit dem Namen **sql-stmt** (SQL-Anweisung).
Nach einem **sql-stmt** kann man aber auch zum Anfang zurückkehren. Um z.B. ein weiteres **sql-stmt** folgen zu lassen ist aber zwangsläufig ein Semikolon (**;**) notwendig. Solche notwendigen Zeichen – auch Terminale genannt, werden in abgerundeten (ovalen) Rechtecken oder Kreisen notiert. Solche Namen, wie **sql-stmt** sind nur Bezeichner (Platzhalter, Variablen) für andere Syntax-Diagramme. Sie heißen auch Nicht-Terminale. In SQL-Anweisungen kommen sie so nicht vor. Sie müssen immer durch Terminale und / oder Variablen und / oder Namen (z.B. von Tabellen usw.) ersetzt werden.
Diese konkreten Namen für Tabellen usw. werden in den Syntax-Diagrammen *kursiv* gesetzt.
Syntax-Diagramme basieren auf den sogenannten BACKUS-NAUR-Formen zur Darstellung von Sprachen (s.a. → 🕮 **Sprachen und Automaten**).

comment-syntax:



/###/

5.1.1. CREATE DATABASE – Erstellen einer Datenbank

Erzeugen einer neuen Datenbank:
CREATE DATABASE *datenbankname*;

:

/###/

weiter z.B.:
→ 5.1.6. CREATE TABLE – Erstellen einer Tabelle
→ 5.1.2. CREATE USER – Erstellen eines Nutzers

5.1.2. CREATE USER – Erstellen eines Nutzers

Erstellen eines neuen Nutzers:
CREATE USER *nutzername'@'localhost* **IDENTIFIED BY** *anmeldename*;

:

/###/

weiter z.B.: (Voraussetzung sind vorhandene Tabellen usw. in der Datenbank)
→ 5.1.6. CREATE TABLE – Erstellen einer Tabelle
sonst:
→ 5.1.3. GRANT – Setzen von Zugriffsrechten
→ 5.1.4. REVOKE – Entziehen von Zugriffsrechten

5.1.3. GRANT – Setzen von Zugriffsrechten

Setzen von Zugriffs-Rechten auf die Datenbank oder Teile oder der aufgerufenen Funktionabilität dazu:

GRANT ALL ON *datenbank* **TO** *nutzername*;
statt ALL können auch die zulässigen Befehle angegeben werden, also z.B.: SELECT, DELETE, UPDATE, REFERENCES
Gegenstück ist **REVOKE** (Entzug der Zugriffs-Rechte)

GRANT SELECT, DELETE, UPDATE, REFERENCES(*nummer***) ON** *tabelle* **TO** *nutzername*;

5.1.4. REVOKE – Entziehen von Zugriffsrechten

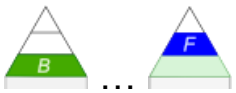
REVOKE DELETE ON *tabelle* **FROM** *nutzername*;

die Vergabe von Zugriffs-Rechten erfolgt mit **GRANT**

⋮

/###/

5.1.5. USE DATABASE – Verbindung zu einer Datenbank herstellen



USE DATABASE *datenbank*;

⋮

/###/

5.1.6. CREATE TABLE – Erstellen einer Tabelle

Erstellen einer Tabelle einschließlich ihrer Struktur (ohne eigentliche Daten):
CREATE TABLE *tabelle* **(*attributTyp* { , *attributTyp* } , PRIMARY KEY (*attribute*)) { , FORREIGN KEY (***attribut***) REFERENCES** *referenz_tabelle* **(***referenz_attribut***) });**

CONTRAINT …		
… PRIMARY KEY	setzt Spalte als Primär-Schlüssel	
… REFERENCES	definiert Spalte als Fremd-Schüssel	
… NOT NULL	erzwingt Daten-Eingabe (Wert kann nicht leer sein!)	
… CHECK	realisiert eine Prüfung der Eingabe	

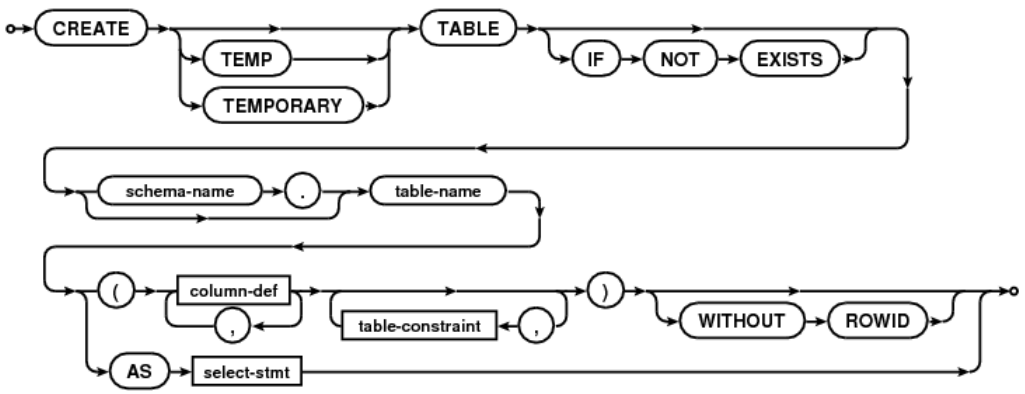
mit **UNIQUE(**spalte**)** wird festgelegt, das die Werte in der Spalte einmalig sein müssen
bei **UNIQUE(**spalte1, spalte2**)** muss die Kombination aus beiden Spalten-Werten einmalig sein

CREATE SEQUENCE variable **INCREMENT BY** 1 **MINVALUE** 1;
INSERT INTO *tabelle* **(***spalte1* { , *spalteN* }) **VALUES** (*variable***.NEXTVAL** { , wertN } **)**;

komplexes Beispiel:

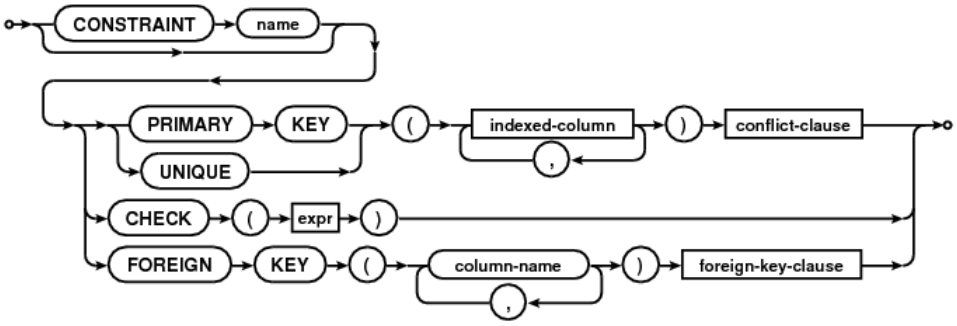
```
CREATE TABLE Nutzer ( NID INT CONSTRAINT Loginname PRIMERY KEY,
   Nachname VARCHAR(30) CONSTRAINT Nutzer_Name NOT NULL,
   Vorname VARCHAR(40) CONSTRAINT Nutzer_Vorname NOT NULL,
   GeburtsJahr INT,
   Geschlecht CHAR(1) CONSTRAINT Nutzer_Geschlecht CHECK ( TYP IN ('w', 'm') ),
   GebLand VARCHAR(20) ….. CHECK(GebLand="Deutschland")
   Groesse INT ….. CHECK((Groesse>=40) AND (Groesse<=250))
   Konto VARCHAR(25) CHECK(Konto LIKE "DE%")
   UNIQUE( NID ),
   UNIQUE( Nachname, Vorname ) );
```

create-table-stmt:



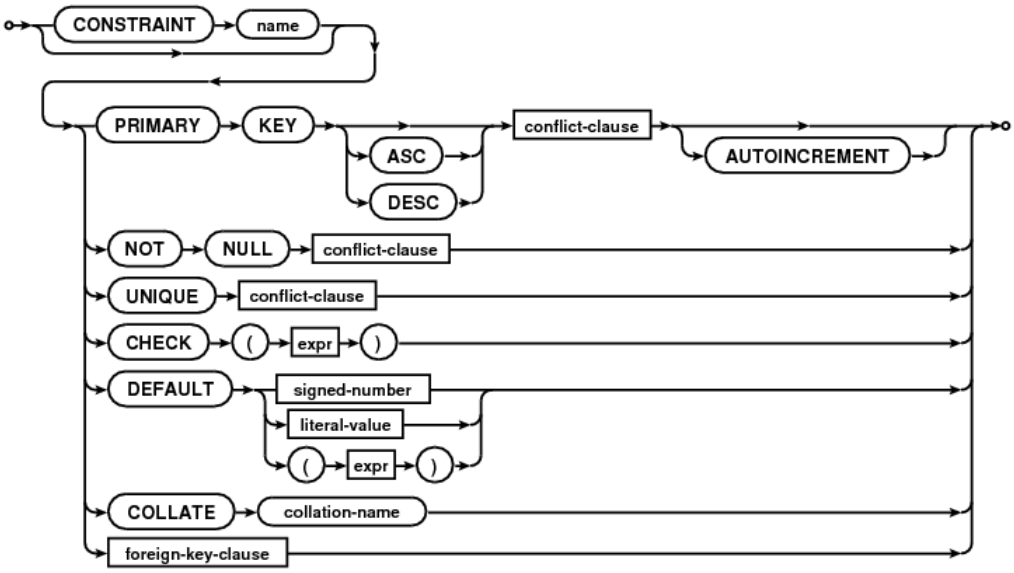
/###/

table-constraint:



/###/

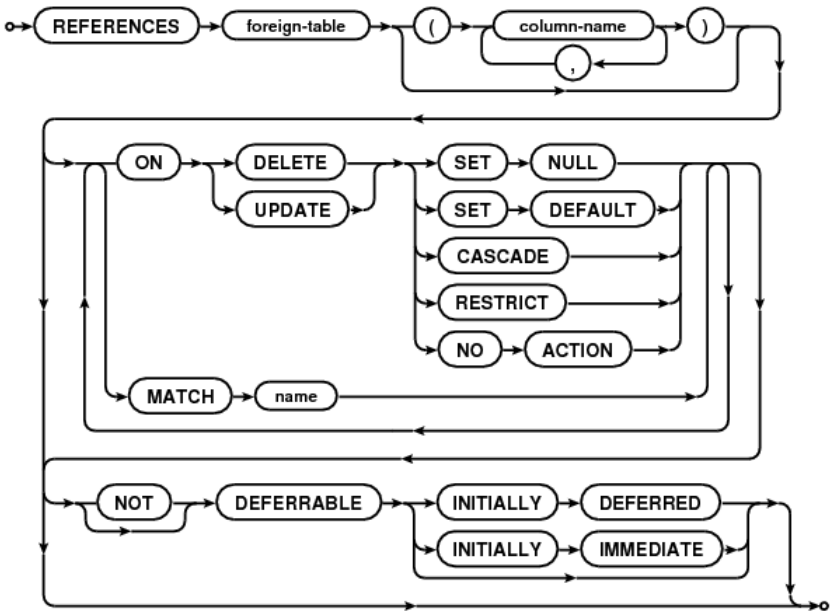
column-constraint:



/###/

5.1.6.1. Tabellen mit Fremdschlüsseln erstellen

foreign-key-clause:



/###/

Beispiel-Konstrukt:
Artikel:=(…; ↗ ArtikelGruppe; …)
ArtikelGruppe:=(AGrID; …)

Optionen bei der Definition von Fremd-Schlüsseln:

ON DELETE RESTRICTED:
ON UPDATE RESTRICTED:
es werden alle Lösch- bzw. Aktualisierungs-Anweisungen in der verknüpften Tabelle (hier: ArtikelGruppe) verweigert, solange es noch es noch Datensätze in der "übergeordneten" Tabelle (hier: Artikel) gibt, die den zu löschenden Fremd-Schlüssel enthalten

ON DELETE CASCADE:
ON UPDATE CASCADE:
alle Lösch- oder Aktualisierungs-Anweisungen (also Primär-Schlüssel-Aktualiserungen) in der verknüpften Tabelle (hier: ArtikelGruppe) wird an die übergeordnete Tabelle (hier: Artikel weitergegeben und alle Datensätze in diese Tabelle gelöscht / aktualisiert

ON DELETE SET NULL:
ON UPDATE SET NULL:
Löschung oder Aktualisierung eines Primär-Schlüssels in der verknüpften Tabelle (hier: ArtikelGruppe) bewirkt einen NULL-Wert in der übergeordneten Tabelle (hier: Artikel)

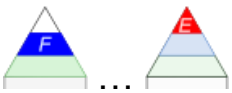
ON DELETE SET DEFAULT:
ON UPDATE SET DEFAULT:
Löschung oder Aktualisierung eines Primär-Schlüssels in der verknüpften Tabelle (hier: ArtikelGruppe) bewirkt ev. eine Neu-Setzung des Schlüssel-Wertes auf den / einen Vorgabe-Wert in der übergeordneten Tabelle (hier: Artikel)

die beiden letzten Varianten zerstören die referenzielle Integrität
ON CASCADE sollte nur mit Bedacht benutzt warden, da so schnell viele Kategorien / … schnell aus der datenbank verschwinden (und bei Bedarf wieder neu eingegeben werden müssen)

Beispiel aus der Warenhaus-Welt:

```
CREATE TABLE Artikel
(       AID             Interger        NOT NULL,
        ANummer         VarChar(15)     NOT NULL,
        ABezeichnung    Var0Char(30)    NOT NULL,
        …
        APreis          Float(2)        NOT NULL DEFAULT 0,

        PRIMARY KEY(AID),

        FOREIGN KEY(LID) REFERENCES Lieferant
        ON UPDATE CASCADE
        ON DELETE NO ACTION,

        …

        CHECK(APreis >= 0)
)
```

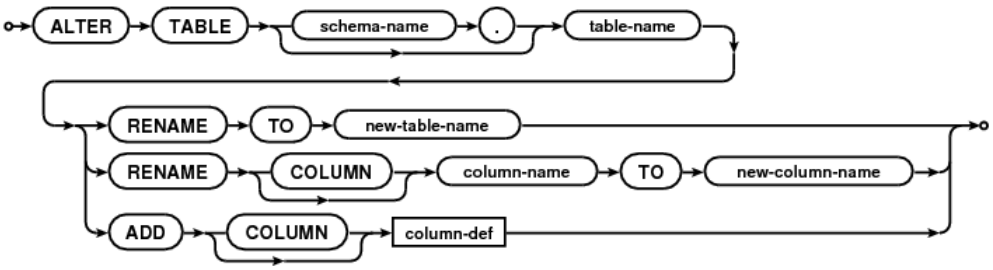
5.1.7. ALTER TABLE – Ändern der Tabelle(n-Struktur)



verändern der Tabellen-Struktur, z.B. Verändern der Feld-Größe:
ALTER TABLE *tabelle* **MODIFY (***attribut* **NUMERIC(***neueLänge***));**

oder Hinzufügen einer neuen Spalte:
ALTER TABLE *tabelle* **ADD (***attribut* **CHAR(***länge***));**

alter-table-stmt:

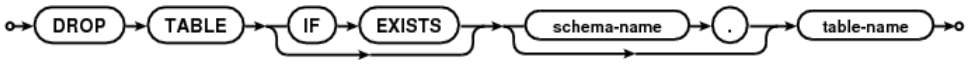


/###/

5.1.8. DROP TABLE – Löschen einer Tabelle

löschen einer ganzen Tabelle (einschließlich des Inhalts):
DROP TABLE *tabelle* **INCLUDING CONTENTS;**

drop-table-stmt:



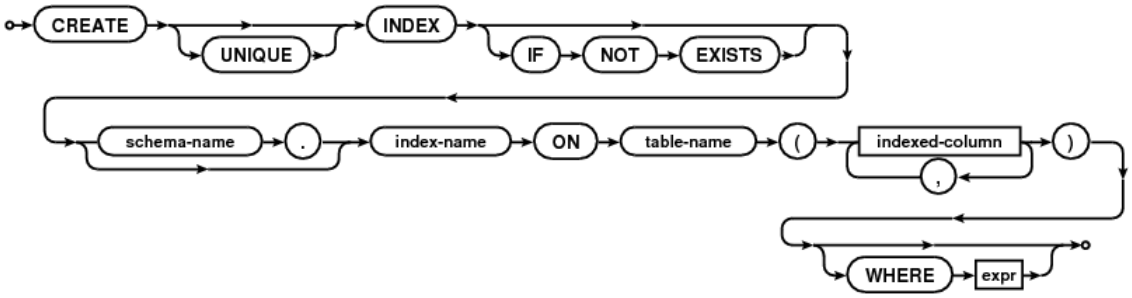
/###/

5.1.9. CREATE / ALTER / DROP INDEX – Erstellen, Ändern und Löschen eines Index

Es gelten die selben Prinzipien, wie bei **TABLE.**

Erstellen eines neuen Index:

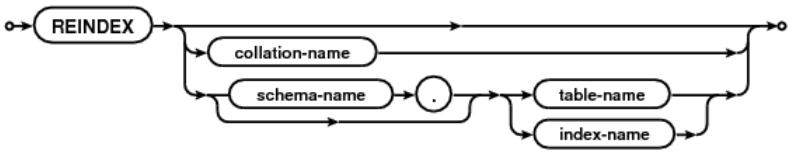
create-index-stmt:



/###/

Neuerstellen / Aktualisieren eines Index

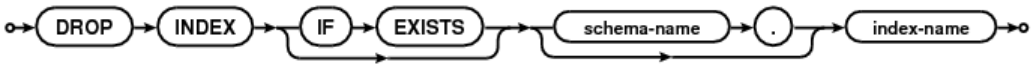
reindex-stmt:



/###/

Löschen eines Index

drop-index-stmt:



/###/

5.1.10. INSERT – Einfügen eines Datum's

Einfügen von Daten in eine Tabelle:
INSERT INTO *tabelle* **(***attribut* **{ ,** *attribut* **}) VALUES (***wert* **{ ,** *wert* **})**

:

/###/

5.1.11. SELECT … FROM – Auswählen von Spalten und / oder Zeilen

spalten ::= attribut { , attribut }
aggregation ::= aggregationsfunktion(spalte)
bedingung ::= vergleich

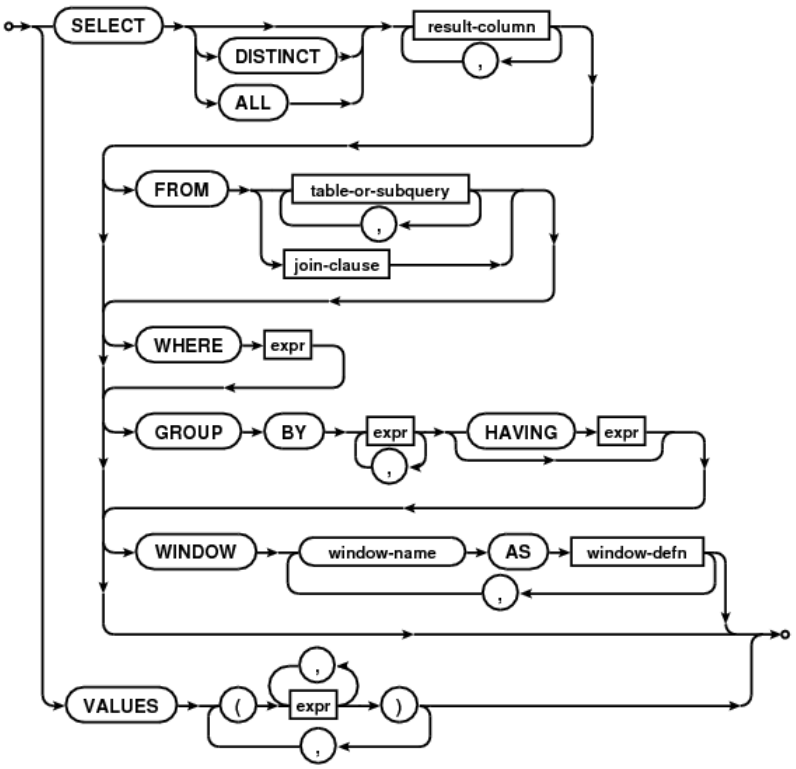
bedingung mus als Ausdruck immer ein WAHR oder FALSCH (TRUE / FALSE) ergeben.

Syntax:
SELECT ***spalten*** **|** ***aggregation*** **FROM** ***tabelle*** **[WHERE** *bedingung* **] [GROUP BY** *spalten* **|** *index* **] [HAVING** *bedingung* **] [ORDER BY** *spalten* **[ASC] | DESC]**

ASC .. (ascending) aufsteigende / ansteigende Reihenfolge / Sortierung
DESC .. (descending) absteigende Reihenfolge / Sortierung

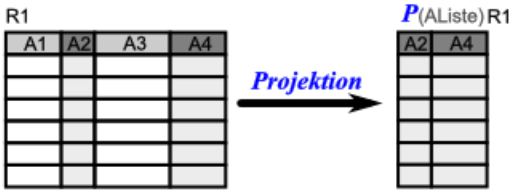
Darstellung einer Tabelle / Anzeige aller Daten einer Tabelle
SELECT * FROM *tabellenname*;

select-core:



/###/

5.1.11.1. Projektion (Abbildung):



Bei der Projektion werden von der Original-Tabelle oder einer Sicht nur bestimmte Spalten abgebildet. Eine Auswahl der Datensätze erfolgt nicht.
Es werden praktisch Sichten erzeugt.
Projektionen dienen z.B. dazu die angezeigte / verfügbare Daten-Menge zu reduzieren oder den Datenschutz-Bedingungen anzupassen.

SELECT *attribut* **{** **,** *attribut* **}** **FROM** *tabelle***;**

Anzeige ausgewählter Spalten einer Tabelle
SELECT *spaltenname1, spaltenname2* **{**, *spaltennameN***}** **FROM** *tabellenname*;

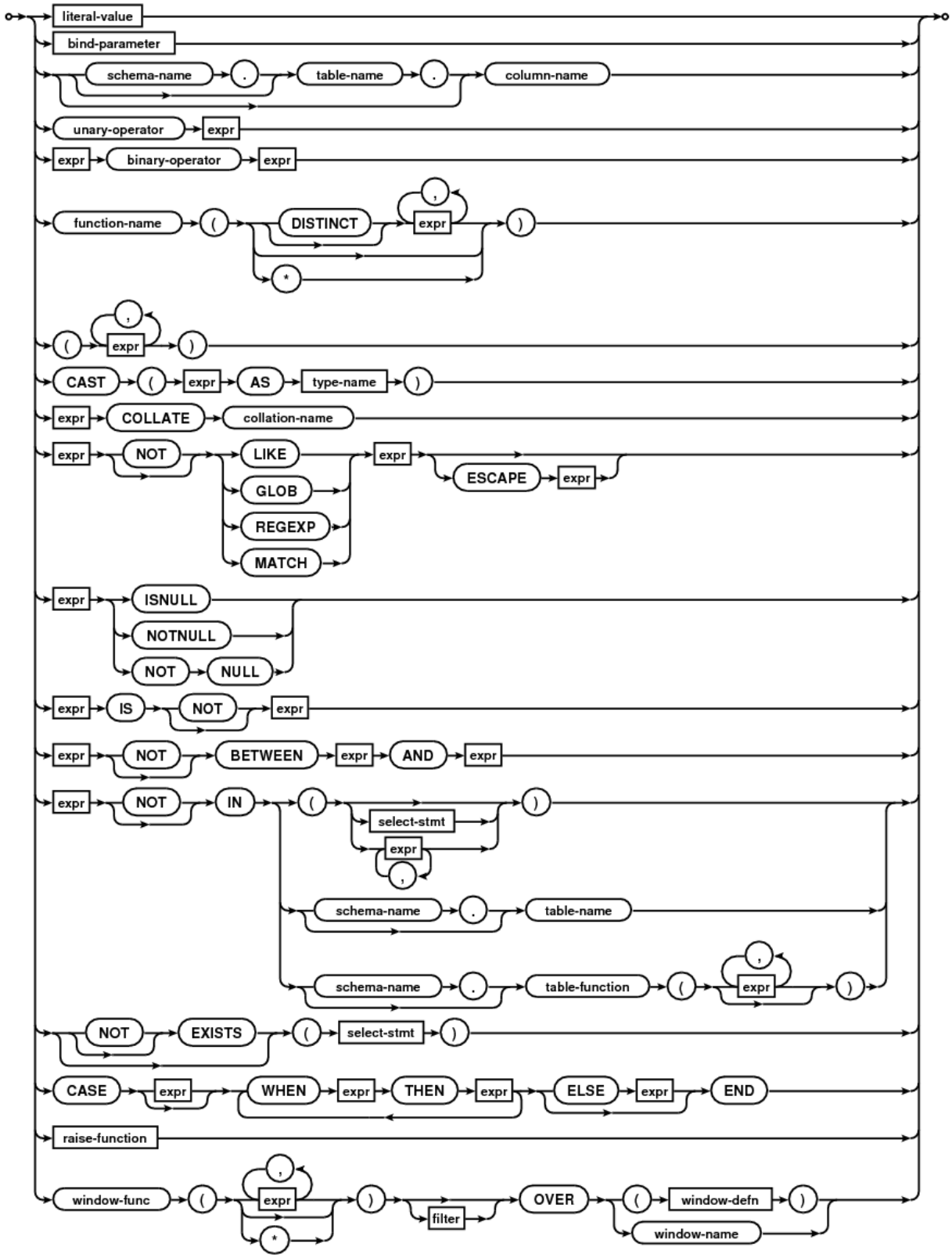
Anzeige ausgewählter Spalten (nach ihrer Positionsnummer in der Tabelle)
Zählung beginnt bei 1!
SELECT 1, 2 **{**, N**}** **FROM** tabellenname;

nach bestimmten Spalten sortierte Ausgabe ausgewählter Spalten einer Tabelle (meint standardmäßig aufsteigende Sortierung)
SELECT *spaltenname1, spaltenname2* **{**, *spaltennameN***}** **FROM** *tabellenname* **ORDER BY** spaltennameX **{**, spaltennameZ**}**;

… betont aufsteigende Sortierung
SELECT *spaltenname1, spaltenname2* **{**, *spaltennameN***}** **FROM** *tabellenname* **ORDER BY** *spaltennameX* **{**, *spaltennameZ***}** **ASC**;

… mit absteigenden Sortierung:
SELECT *spaltenname1, spaltenname2* **{**, *spaltennameN***}** **FROM** *tabellenname* **ORDER BY** *spaltennameX* **{**, *spaltennameZ***}** **DESC**;

expr:


literal-value
bind-parameter
schema-name
.
table-name
.
column-name
unary-operator
expr
expr
binary-operator
expr
function-name
(
DISTINCT
expr
,
*
)
(
expr
,
)
CAST
(
expr
AS
type-name
)
expr
COLLATE
collation-name
expr
NOT
LIKE
GLOB
REGEXP
MATCH
expr
ESCAPE
expr
expr
ISNULL
NOTNULL
NOT
NULL
expr
IS
NOT
expr
expr
NOT
BETWEEN
expr
AND
expr
expr
NOT
IN
(
select-stmt
expr
,
)
schema-name
.
table-name
schema-name
.
table-function
(
expr
,
)
NOT
EXISTS
(
select-stmt
)
CASE
expr
WHEN
expr
THEN
expr
ELSE
expr
END
raise-function
window-func
(
expr
,
*
)
filter
OVER
(
window-defn
)
window-name

/###/

5.1.11.1.1. Projektion für Fortgeschrittene:

Projektion nur eines Teils aus der Spalte (Teilstring):
SELECT SUBSTR(*spaltenname, startPosition, anzahlZeichen*) **FROM** *tabellenname***;**

Projektion mit Verbindung (Konkatenation) von Spalten zu einer neuen (hier mit Komma-Trennung):
SELECT *spaltenname1* **| |** ', ' **| |** *spaltenname2* "*neuerSpaltenname*" **FROM** *tabellenname***;**

Zusammenfassung gleicher Attribut-Werte (Spalten-Inhalte) zu einer Zeile:
SELECT DISTINCT *spaltenname* **FROM** *tabelle***;**
damit verhandert man das mehrfache Aufführen gleicher Attribut-Werte im Ergebnis

Nutzung von Aggregat-Funktionen (); hier: Zählung mit **COUNT()**
SELECT COUNT(*) "*Anzahl*" **FROM** *tabelle***;**

weitere Aggregat-Funktionen sind:

Aggregat-Funktion	Aufgabe	Bemerkungen
MIN()	Minimum	
MAX()	Maximum	
SUM()	Summe	
AVG()	Durchschnitt	

Gruppierung gleicher Werte einer Spalte (hier Spalte1) und gruppierte Summierung der Werte aus Spalte2:
SELECT *spaltenname1,* **SUM(***spaltename2***) FROM** *tabelle* **GROUP BY** *spaltenname1***;**

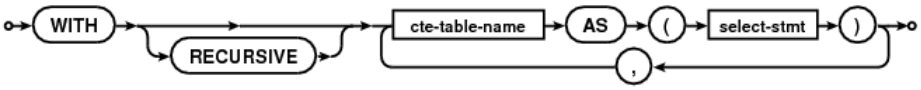
begleitete Daten-Typen-Konvertierung
SELECT TO_CHAR(*spaltenname, format***)** "Anzahl" **FROM** *tabelle***;**
SELECT TO_CHAR(*Datum, 'YYYY'***)** FROM tabelle;
weitere Daten-Typen-Konvertierungen: **TO_NUMBER()**, **TO_DATE()**

NULL-Werte sollen durch spezielle Texte etc. ersetzt werden:
SELECT *spaltenname1,* **NVL(** *spaltenname2, '???'***) FROM** *tabelle***;**

SELECT *Name*, **NVL(** *GebDatum, '???'***) FROM** *tabelle***;**

die ersten n Zeilen einer sortierten Auswahl anzeigen:
SELECT * FROM (SELECT ROWNUM *nummer, resultatTabelle.** **FROM (SELECT * FROM** *tabelle* **[WHERE** … **] ORDER BY** …) *resultatTabelle*) **WHERE** *nummer* = *n***;**

with-clause:



/###/

Beispiel(e) aus der Warenhaus-Welt:

den billigsten Artikel(-Preis) anzeigen:

```
SELECT MIN(APreis) FROM Artikel
```

die Daten für den billigsten Artikel anzeigen:

```
SELECT * FROM Artikel
WHERE APreis = (    SELECT MIN(APreis)
                    FROM Artikel
                )
```

den billigsten Artikel(-Preis) in jeder ArtikelGruppe anzeigen:

```
SELECT AGrID, MIN(APreis)
FROM Artikel
GROUP BY AGrID
```

den billigsten Artikel in jeder ArtikelGruppe (mit seinen Daten) anzeigen:

```
SELECT Tab1.*                                              Tab1 ist ein Alias
FROM Artikel Tab1                                          hier Def. des Alias
WHERE Tab1.APreis = (     SELECT MIN(Tab2.APreis)                Tab2 ist Alias
                          FROM Artikel Tab2                      Def. Alias
                          WHERE Tab1.AGrID = Tab2.AGrID
                    )
```

die Anzahl der Artikel ermitteln:

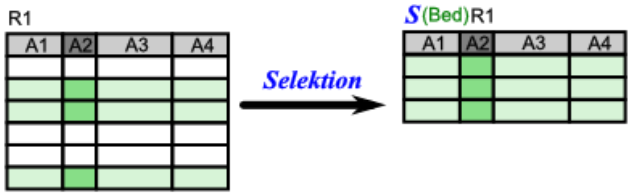
```
SELECT COUNT(*)
FROM Artikel
```

den durchschnittlichen Preis in jeder ArtikelGruppe berechnen:

```
SELECT AGrID, AVG(APreis)
FROM Artikel
GROUP BY AGrID
```

5.1.11.2. Selektion (Auswahl):

Aufgrund bestimmter Kriterien, Bedingungen, Werte, … werden bestimmte Datensätze ausgewählt.
Besonders typisch ist dazu der WHERE-Teil / -Anhang in SELECT-Anweisungen. Aber es gibt auch andere Möglichkeiten Datensätze auszuwählen.

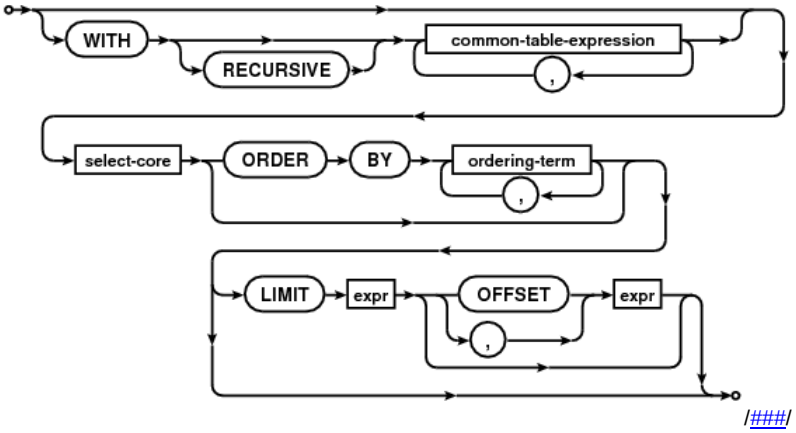


SELECT * FROM *tabelle* **WHERE *bedingung*;**
bedingung kann z.B. der Vergleich mit einem Wert sein; bedingung muss dann den spaltennamen ein Vergleichzeichen und den Vergleichswert enthalten
z.B.: spaltenname = wert oder spaltenname < wert oder spaltenname >= wert bzw. spaltenname <> wert usw. usf.

weiterhin geht für ***bedingung*** auch:
spaltenname **BETWEEN** *kleinsterWert* **AND** *größterWert*
wir suchen also aus einem Bereich heraus

Texte werden in einfache Hochkommata (**' '**) eingeschlossen
um z.B. die Groß- und Klein-Schreibung bei Texten nicht einzeln zu prüfen können Texte z.B. auf Groß-Buchstaben umgestellt werden mit **UPPER()**
(bei ACCESS wird die Funktion UCASE() benutzt)

simple-select-stmt:



/###/

SELECT * FROM *tabelle* **WHERE** *bedingung1* **{ AND** *bedingungN* **};**

SELECT * FROM *tabelle* **WHERE** *bedingung1* **{ OR** *bedingungN* **};**

SELECT * FROM *tabelle* **WHERE** *spaltenname* **LIKE** *suchausdruck***;**
suchausdruck kann z.B. 'M%' sein, dann werden alle Ausdrucke mit einem beginnenden großen B akzeptiert
enthält der suchausdruck Unterstriche, dann stehen diese für genau ein beliebiges Zeichen

SELECT * FROM *tabelle* **WHERE** *spaltenname* **IN** *suchmenge***;**
suchmenge wird in runden Klammern angegeben; also z.B.: (1, 3, 5, 7)

SELECT * FROM *tabelle* **WHERE** *spaltenname* **IS NULL;**

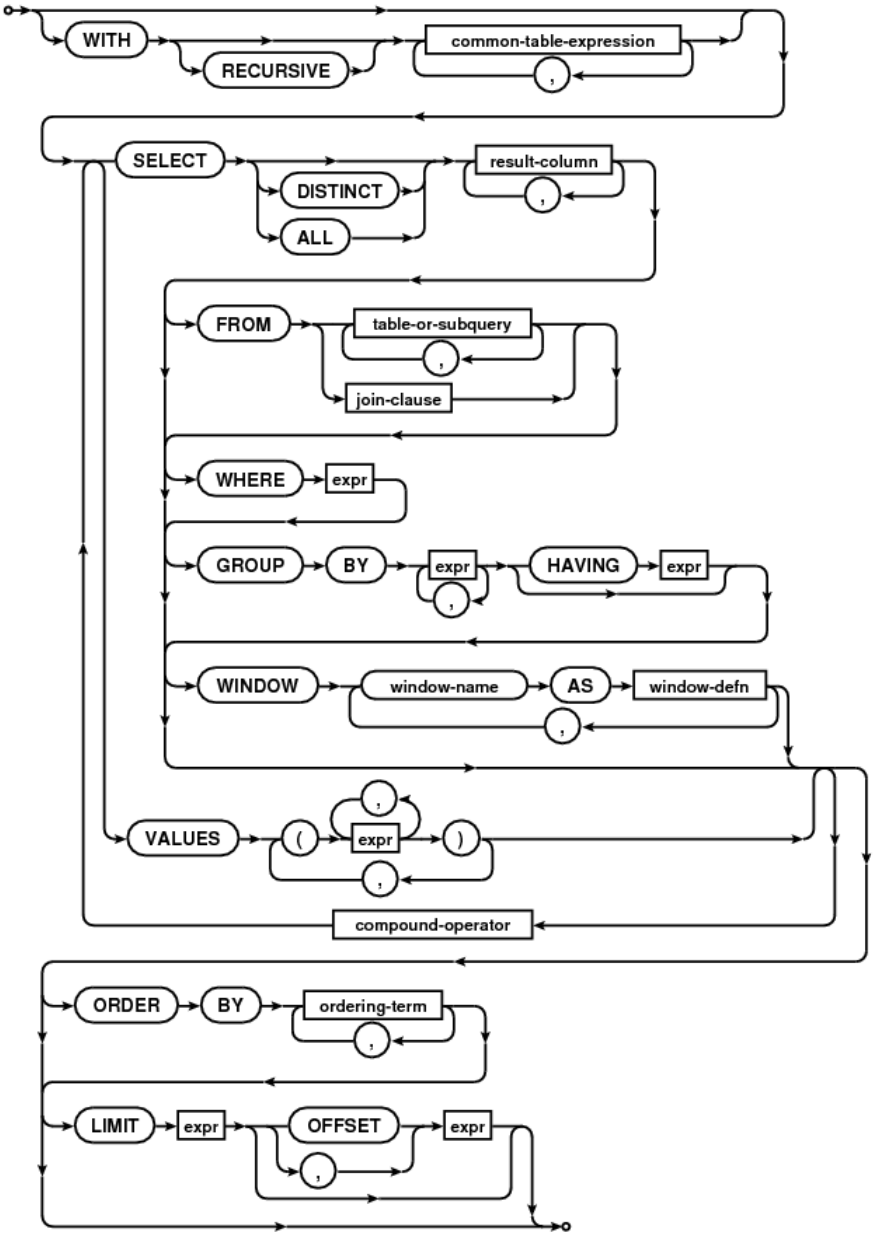
SELECT * FROM *tabelle* **WHERE** *spaltenname* **IS NOT NULL;**

5.1.11.2.1 Selektion für Fortgeschrittene:

Datum-bezogene Auswahl (hier bestimmte Anzahl Tage zurück):
SELECT * FROM *tabelle* **WHERE** *spaltenname* **>=** (**SYSDATE** – *anzahlTage*)**;**
SELECT * FROM *tabelle* **WHERE** *spaltenname* **>=** (**NOW()** – *anzahlTage*)**;**
die System-Datums-Funktion ist in den DB-Systemen unterschiedlich realisiert

SELECT * FROM *tabelle* **WHERE** *spaltenname1* **IS NOT NULL AND** *spaltenname2* **IS NULL;**

select-stmt:



/###/

das Schlüsselwort **DISTINCT**

Syntax
SELECT DISTINCT *spalte* **{**, *spalte***}** **FROM** *tabelle* **WHERE** ….

verhindert Wiederholung gleicher Ergebnisse / Ergebnis-Zeilen bezüglich der aufgezählten Attribute (Spalten)

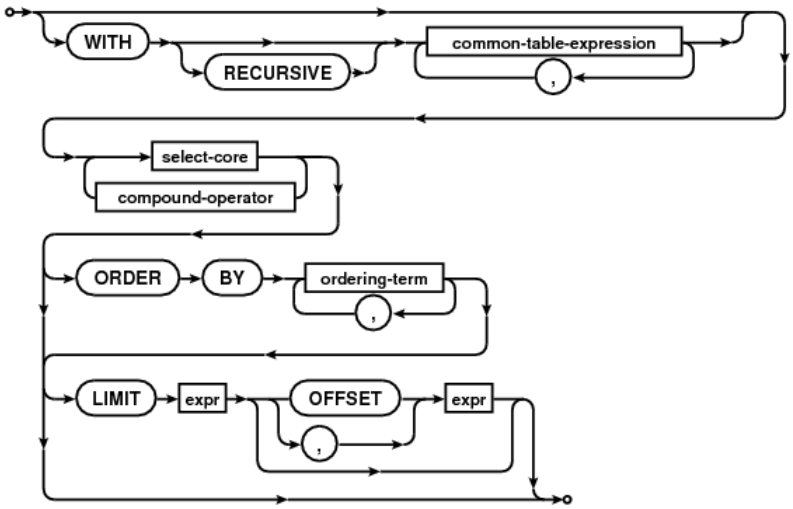
oft gebraucht ist das Auswerten der so kompromierten Tabelle, z.B. dahingehend, wieviel verschiedene Einträge (Werte) nun auftauchen
SELECT COUNT(DISTINCT *spalte* **)** **FROM** *tabelle* **WHERE** ….

in ACCESS kann dieser Konstrukt nicht funktionieren. Dann bietet sich der folgende Ausdruck an:
SELECT COUNT(*) AS *hilfstabelle* **FROM (SELECT DISTINCT** *spalte*
FROM *tabelle* **WHERE** …**)**.

:

/###/

factored-select-stmt:



/###/

mit LIMIT begrenzt man die Anzahl der Tupel (Datensätze) z.B., wenn es darum geht die 10 besten Schüler oder die 3 schnellsten Auto's oder die 5 größten Länder usw. usf.
natürlich lassen sich bei umgekehrter Sortierung auch die schlechtesten, langsamsten und kleinsten finden

komplexere Beispiel(e) aus der Warenhaus-Welt:

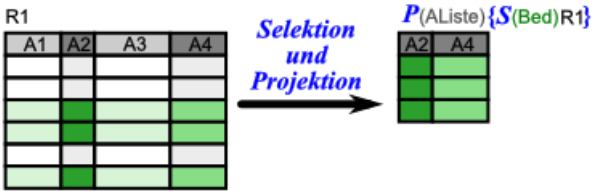
die ArtikelGruppe mit der Preis-Summe anzeigen, bei denen die Preis-Summe einen bestimmten minimalen Wert hat:

```
SELECT AGrID, SUM(APreis)
FROM Artikel
GROUP BY AGrID
HAVING SUM(APreis) >= 10
```

für jeden Kunden die Anzahl seiner Bestellungen und die Anzahl der bestellten Artikel anzeigen:

```
SELECT KID, COUNT(*) AS AnzahlBestellungen, SUM(Menge) AS ArtikelAnzahl
FROM Bestellungen
GROUP BY KID
```


5.1.11.3. Verbund (Join):



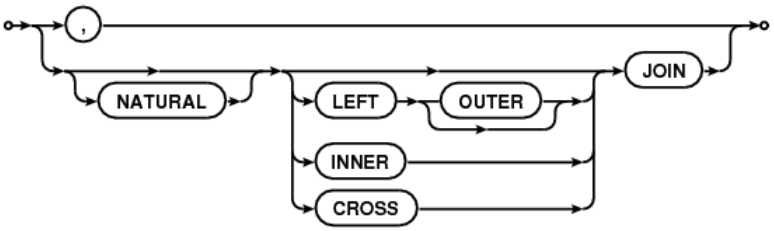
Verbindung von Projektion und Selektion. Hauptsächlich wird dabei in der **SELECT**-Anweisung der Teil vor dem **FROM** dazu benutzt, um die Spalten abzubilden (auszuwählen). Mittels **WHERE**-Abschnitt wird die eigentliche Auswahl vorgenommen.
Hierbei werden echte Sichten z.B. für Formulare / Berichte usw. erzeugt. Sie enthalten nur noch kleinste Daten-Mengen, aber geanu die, die gebraucht / gewünscht / angefordert wurden.

SELECT *
FROM Tabelle1, Tabelle2

liefert das relationale Produkt

Tabelle1 **X** Tabelle2

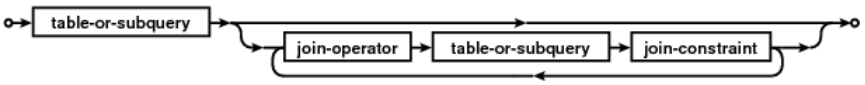
join-operator:



/###/

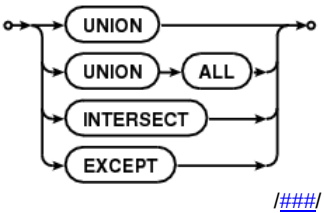
das entspricht einem CROSS JOIN (→ Kreuz-Produkt / kartesisches Produkt)
auch als kartesisches bzw. Kreuz-Produkt bezeichnet

join-clause:



/###/

compound-operator:



/###/

Beispiel(e) aus der Warenhaus-Welt:

:
SELECT
FROM

5.1.11.3.1 Verbund für Fortgeschrittene:

praktisch alle Kombinationen von einfachen und speziellen Projektionen und Selektionen möglich

hohe Kunst der Daten-Auswahl

SQL-Anweisungen werden dann auch schnell unübersichtlich und teilweise auch schwer lesbar, da jetzt die Syntax-Vorschriften dominieren und die Nutzer-Verständlichkeit zurückbleibt.
ev. durch Aufeinanderfolge von einzelnen SQL-Anweisungen / -Sequenzen zu vereinfachen

⋮

/###/

5.1.11.4. Verbund (Equi Join):

SELECT * FROM *tabelle* **WHERE** *spaltenname* **IN (*selektion*);**
selektion ist dabei wieder ein eigenständiger vollständiger Selektions-Befehl (aber ohne inneres, abschließendes Semikolon
quasi kaskadierte Selektion

SELECT * FROM *tabelle1, tabelle2* **WHERE** *tabelle1.attribut = tabelle2.attribut*

⋮

/###/

5.1.11.5. Tabellen-Erstellungs-Abfrage

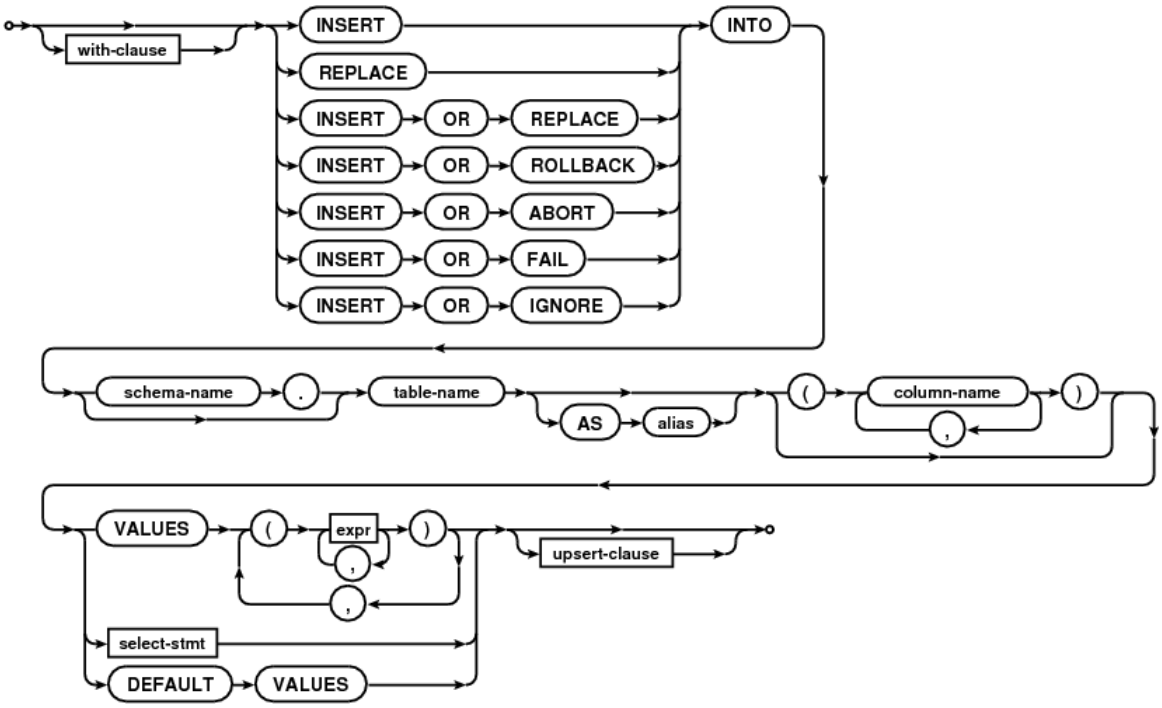
Syntax
SELECT *spalten* **INTO** *neue_tabelle* **FROM** *tabelle* **[WHERE *bedingung*];**

⋮

/###/

5.1.12. INSERT INTO – Einfügen von …

insert-stmt:



/###/

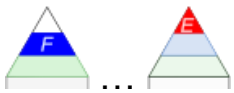
5.1.12.1. Anfüge-Abfrage

Syntax
INSERT INTO *tabelle* **{** ***spalten*** **}** **VALUES** **{** ***werte*** **}** **;**
INSERT INTO *tabelle* **{** ***spalten*** **}** **SELECT** ***spalten*** **FROM** *quell_tabelle* **[** **WHERE** ***bedingung*** **]** **;**

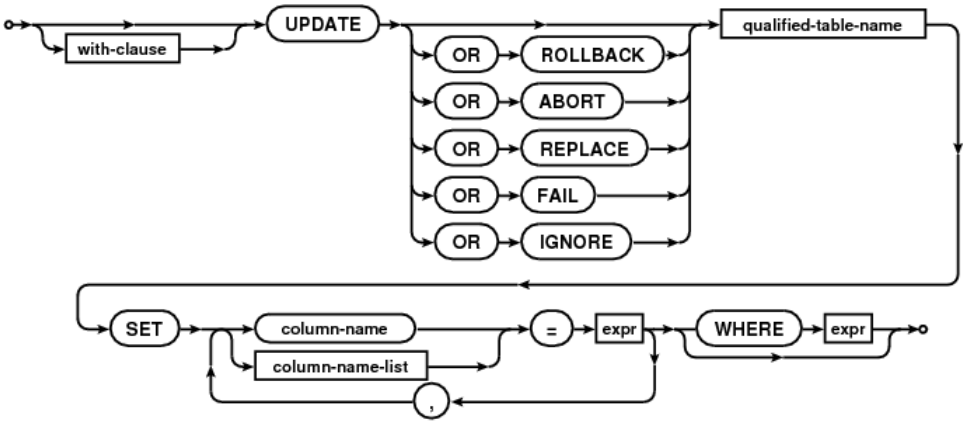
:

/###/

5.1.13. UPDATE – Aktualisieren von …



update-stmt:



/###/

5.1.13.1. Aktualisierungs-Abfrage

Ändern von Inhalts-Werten augrund einer zutreffenden Bedingung:
UPDATE *tabelle* **SET** *attribut = neuer_wert* **{**, *attribut = neuer_wert***}** **[WHERE *bedingung*];**

Syntax
UPDATE *tabelle* **SET** *spalte = ausdruck* **|** *wert* **WHERE *bedingung*;**

:

/###/

5.1.14. DELETE FROM – Löschen von …

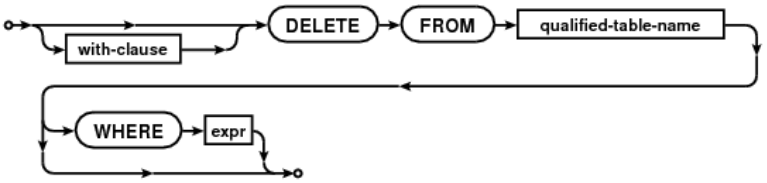
5.1.14.1. Lösch-Abfrage

Löschen von ausgewählten Datensätzen aus einer Tabelle:
DELETE FROM *tabelle* **WHERE** ***bedingung*;**

DELETE *tabelle*
→ **DROP** *tabelle*

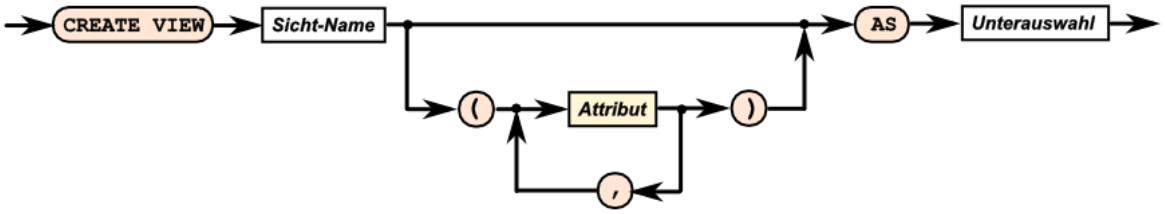
ganze Tabelle löschen:
DROP TABLE *tabelle*

delete-stmt:



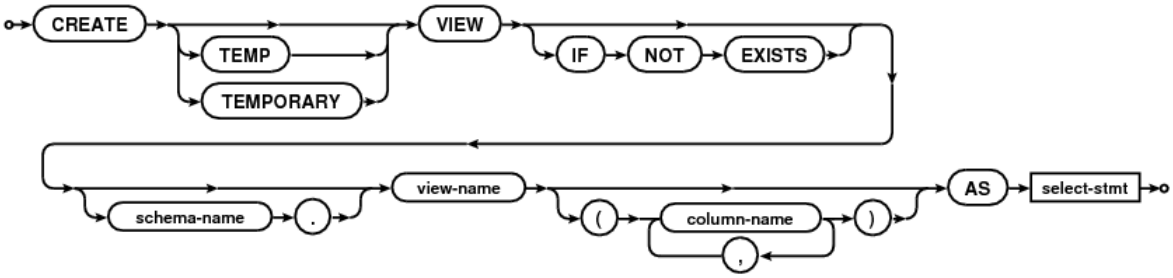
/###/

5.1.15. CREATE / ALTER / DROP VIEW – Erstellen, Ändern und Löschen einer Sicht (Abfrage)



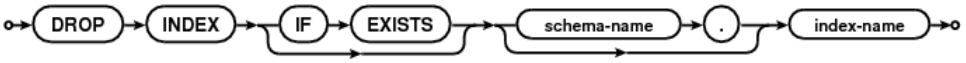
Syntax-Diagramm für CREATE VIEW (!!! noch prüfen!)

create-view-stmt:



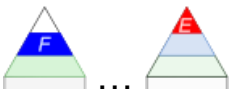
/###/

drop-view-stmt:

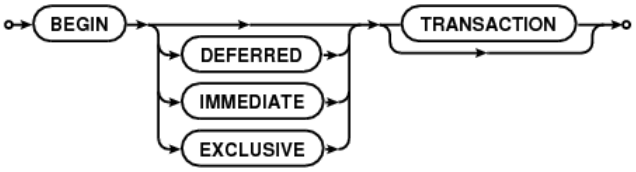


/###/

5.1.16. Transaktionen



begin-stmt:



/###/

commit-stmt:

/###/

wirkliche Ausführung aller DML-Anweisungen seit der letzten COMMIT-Anweisung
COMMIT;

Rücknahme aller DML-Anweisungen seit der letzten COMMIT-Anweisung
ROLLBACK;
immer dann notwendigm wenn eine Transaktion nicht vollständig durchgeführt werden konnte

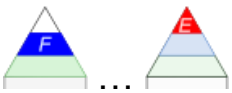
rollback-stmt:

/###/

Sperrung (Einstellung der Unveränderbarkeit) einer Tabelle bis zum nächsten COMMIT bzw. ROLLBACK
LOCK TABLE *tabelle* **IN EXCLISIVE MODE NOWAIT;**

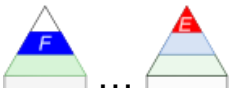
Links:
https://www.sqlite.org/syntaxdiagrams.html (Quelle der SQLite-Syntax-Diagramme in Kapitel 5.x und fehlende Diagramme (verw. Kurz-Referenz: ###))

5.1.17. Berechnungen in Datensätzen



```
SELECT ArtikelID, ArtikelName, BestellMenge, NettoEinzelPreis
                (NettoEinzelPreis)*1,19 AS BruttoEinzelPreis,
                (NettoEinzelPreis)*1,19*(BestellMenge) AS GesamtPreis
FROM Bestellung;
```

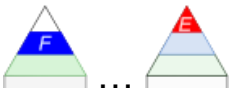
5.1.18. Berechnungen über Tabellen und / oder Abfragen



entspricht einer Berichts-Erstellung
Veränderung zu erwarten, wenn sich ein Wert in der Tabelle bzw. den betreffenden Tabellen (für eine Sicht / Abfrage) geändert hat

```
SELECT KundenID, BestellID,
                SUM(GesamtPreis) AS RechnungsPreis
FROM Bestellung;
```

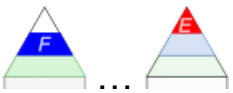
5.1.19. Abfrage von Meta-Daten



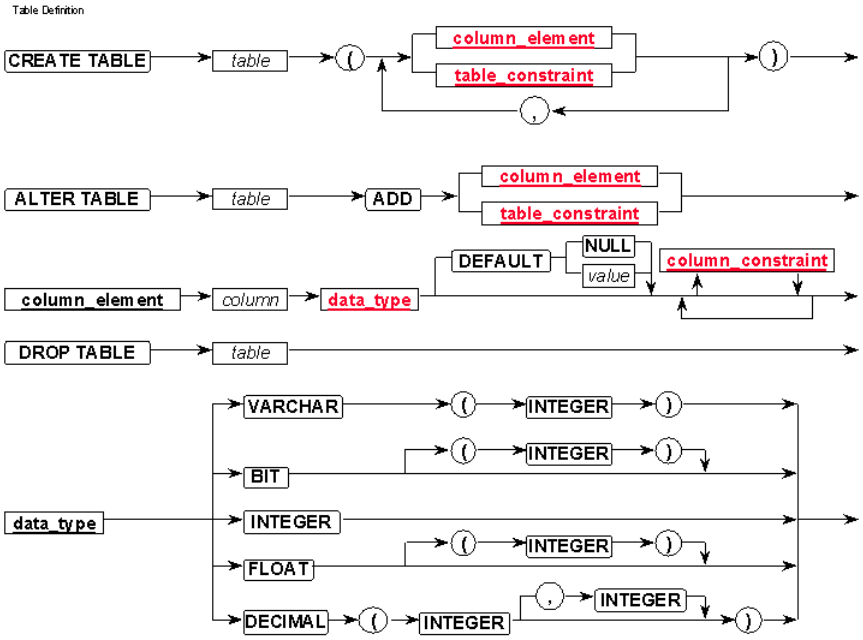
```
SELECT COLUMN_NAME, DATA_TYPE, CHARACTER_LENGTH
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME=TabellenName;
```

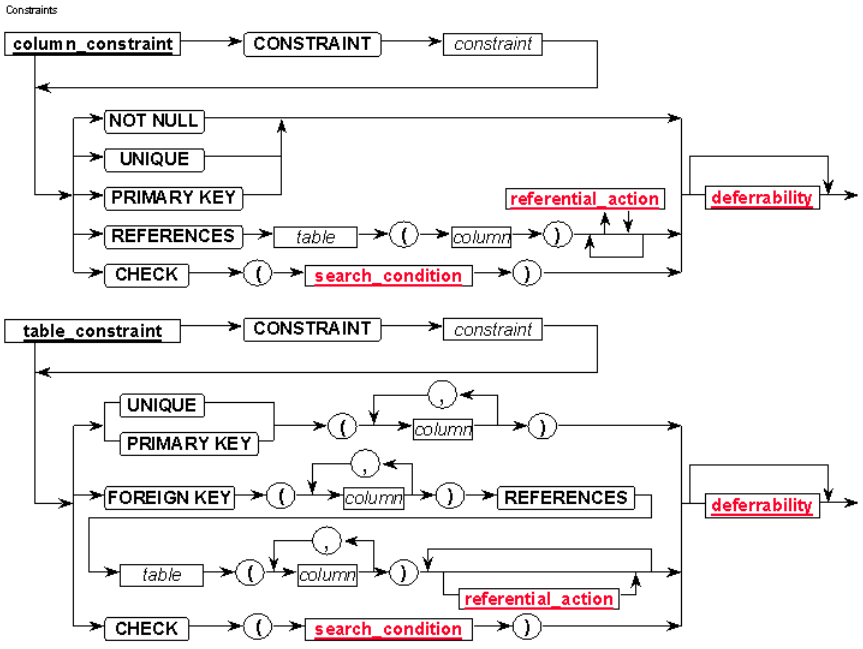
Syntax-Diagramme für SQL'92

(Q: http://www.dbs.ethz.ch/education/infosys/syntax_diagramme/sld001.html)

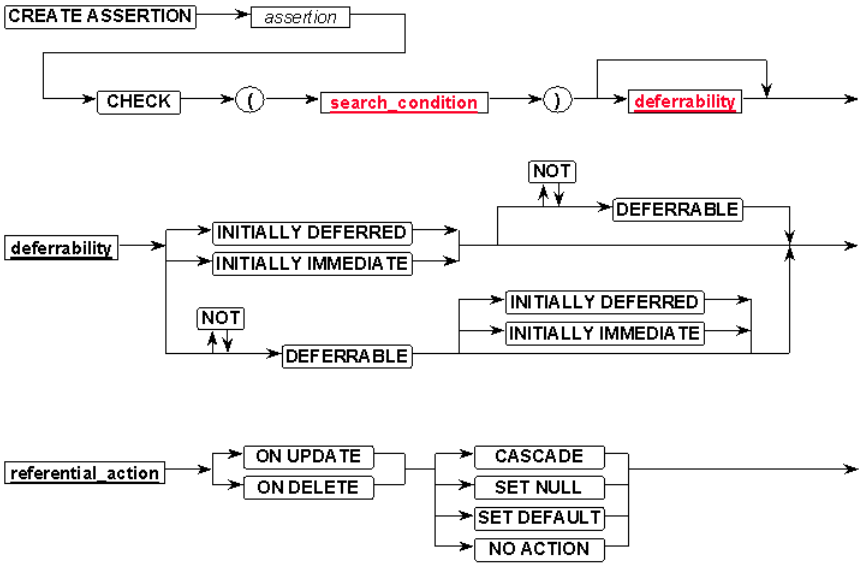


Dieses Syntax-Diagramm ist mit Absicht aufgenommen, um ein zusammenhängendes System für Aufgaben, als Hilfs-Blatt usw. usf. zu haben. Sachlich entspricht das weitesgehend den detaillierten Diagrammen aus SQLite, die wir in den vorlaufenden Abschnitten benutzt haben.

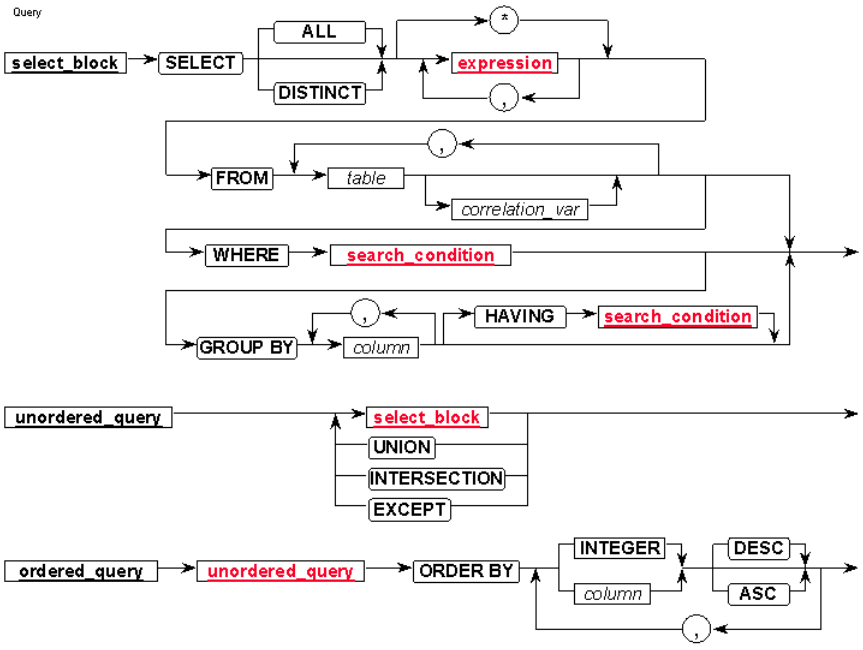




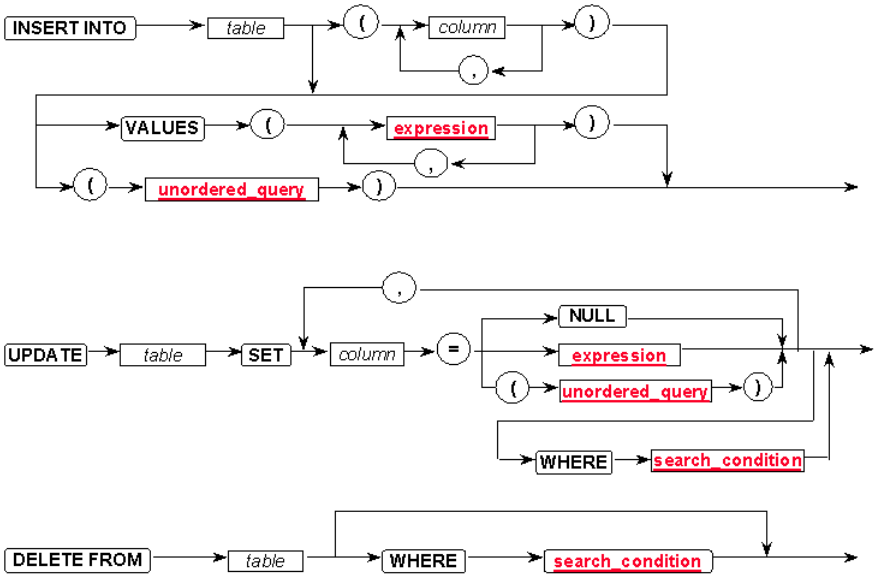
Assertion, Deferrability,
Referential Action



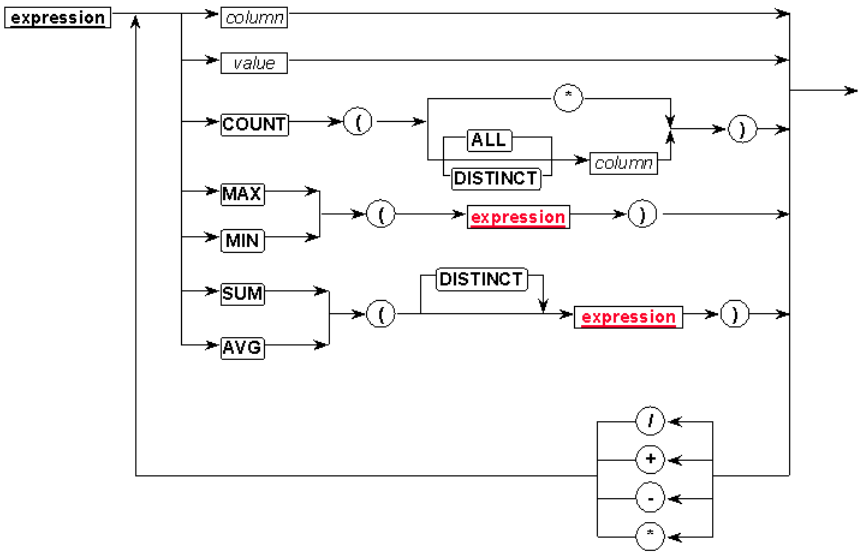
Query



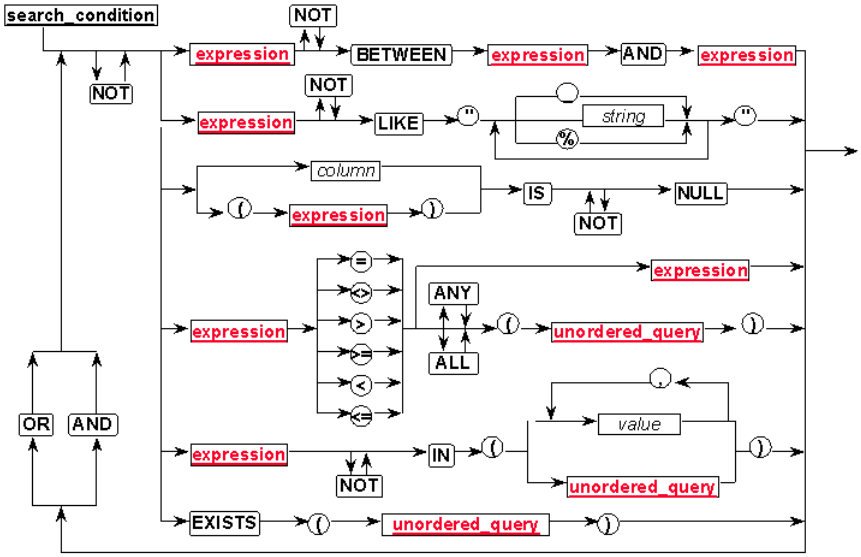
Data Manipulation



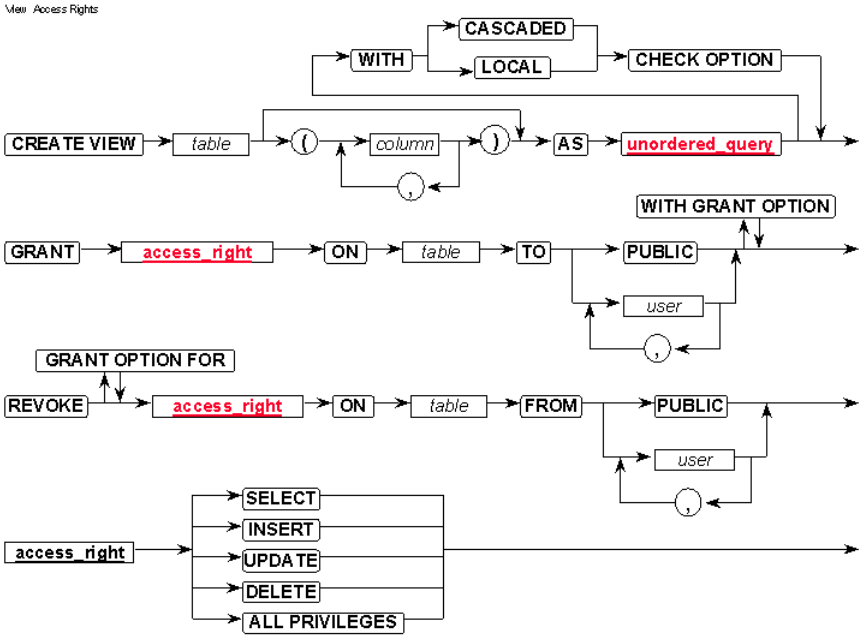
Expression

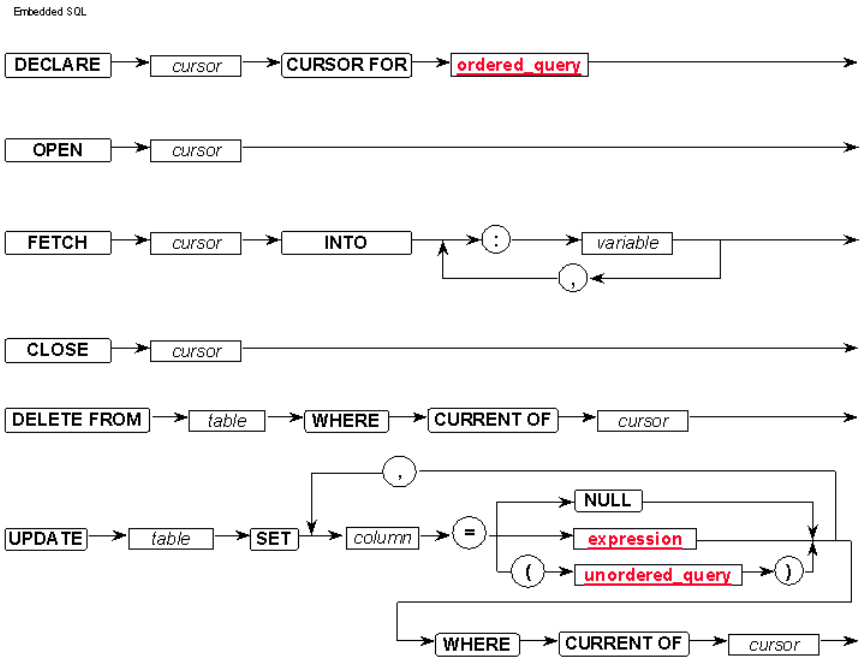


Search Condition



View Access Rights





Aufgaben:

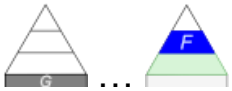
1.

2. Prüfen Sie anhand des Syntax-Diagramms, ob die folgenden Ausdrücke als SQL-Anweisungen zulässig sind!

a) CREATE TABLE Schuhe;
b) CREATE DATABASIS Webshop;
c) WHERE CREATE tabelle FROM attribut1 ORDER BY attribut2;
d) Fahrraeder SELECT Radgroesse FROM "26" WHERE Color="blau";
e)
f)
g)
h)

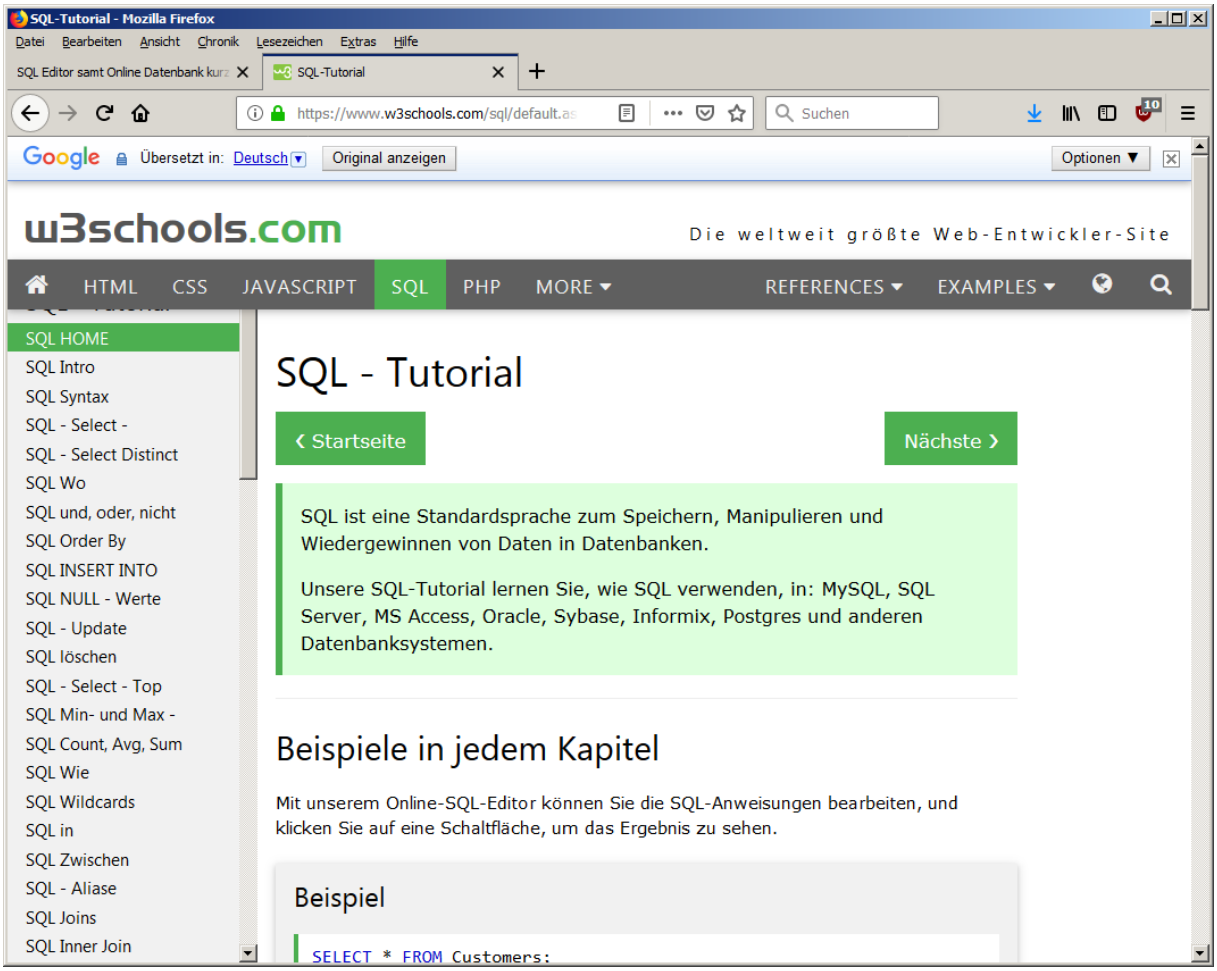
3.

5.2. SQL lernen bei w3schools.com



Wenn der Arbeitsrechner kein Datenbank und / oder SQL-System bereitstellt, dann kann man getrost auf eine online-Plattformen zurückgreifen.
Für Kurse, die Abitur-orientiert haben, können onlinePlattformen nur nebenläufig genutzt werden. In der Prüfungs-Situation steht ja praktisch keine Internet-Verbindung zur Verfügung.
Hier wird meist eine Datenbank mit vielen Daten(-sätzen) und ein SQL-Editor zur Verfügung gestellt.
Somit lassen sich Datenbanken und SQL mit sehr einfachen Smartphone's oder Tablet's erkunden und ausprobieren.
Zerschossener oder fehlerhafter SQL-Code ist kein Problem, man kann i.A. immer wieder von vorne anfangen. Die digitalen Spuren bleiben im Internet.
Der Start-Zustand der Datenbank lässt sich jederzeit wiederherstellen.

online Tutorial bei w3schools.com (→ https://www.w3schools.com/sql/default.asp)
Registrierung nicht notwendig
Arbeits-Stand wird über Cookies gespeichert

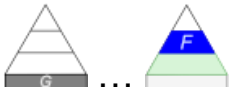


vorrangig englisch; teilweise übersetzt – auch deutsch verfügbar (nach Klick auf das Globus-Symbol erscheint ein Menü mit möglichen Sprachen. in die die Webseite übersetzt wird.
Es wird der google-Übersetzer verwendet. Der ist befriedigend bis gut, kann aber auch verführerisch oder verwirrend sein

Links:
http://www.inf-schule.de/information/datenbanksysteme (Tutorial mit eigenem einfachen Editor)

https://appcamps.de/unterrichtsmaterial/unterrichtsmaterial-zu-datenbanken-und-sql/ (Tutorial und Material von AppCamps)

5.3. Arbeiten mit Übungs-Datenbanken



5.3.1. Nutzung von Datenbanken aus online-Quellen

Prinzipiell haben wir verschiedene Möglichkeiten online-Datenbanken zu nutzen. Die erste Möglichkeit ist eine direkte Nutzung der online-Datenbank über das Internet. Voraussetzung ist natürlich eine ausreichend Leistungs-fähige Internet-Leitung und die Nutzungs-Möglichkeit des Internet's. Z.B. kann die Nutzung während der Durchführungs von Prüfungen eingeschränkt sein.
Meist ist in den online-Datenbanken nur eine lesende Nutzung möglich. Wir können also i.A. nur Anfragen an die Datenbank stellen. Die Datenbanken sind gegen Überschreiben und Löschen von Daten abgesichert. Die online-Nutzer haben keine entsprechenden Rechte. Man braucht sich keine Sorgen machen, dass man durch fehlerhafte SQL-Anfragen einen Schaden anrichtet. Dies darf aber nicht als Aufforderung verstanden werden, die Grenzen der Sicherheit auszutesten.
Die Nutzung selbst hat schon etwas Magisches, wenn man bedenkt, dass man gerade die Daten von einem ev. weit entfernten Rechner runterholt.
Sicherer in der Handhabung sind lokale Datenbanken. Dies setzt aber voraus, dass man die online-Datenbank irgendwie downloaden konnte. Leider sind viele der Datenbestände dann auch wieder sehr gross und unübersichtlich.
Ein entscheidender Vorteil ist, dass man eine zerstörte oder nachhaltig veränderte Datenbank einfach löschen kann und durch die downgeladte originale Datenbank ersetzen kann. Da ist man i.A. innerhalb von Sekunden wieder auf dem Ausgangs-Zustand. Das ist im schulischen Umfeld von großem Vorteil.
Viele Bereitsteller von Datenbanken bieten diese in verschieden Formaten an. Für uns sind die SQLite-Versionen (Datei-Endungen *.db und *. am Besten geeignet.
Natürlich ist die Bereitstellung einer Datenbank als SQL-Anweisungs-Sammlung die Lösung, die am universellsten ist. Schließlich können die von uns besprochenen Programme ja gerade SQL. Die meisten Programme bieten auch eine Import-Möglichkeit an.
Ein Import der SQL-Datenbanken setzt i.A. voraus, dass man im Ziel-Programm zuerst einmal eine neue, leere Datenbank angelegt hat. Erst danach können die Tabellen importiert werden.

Links:

http://www.oberstufeninformatik.de/Datenbanken/ (u.a. SQL-Datenbanken zum Downloaden)

https://dbup2date.uni-bayreuth.de/blocklysql/index.html (Wetter- und Fußball-Bundesliga-Datenbank + Anleitung(en))

5.3.2. Nutzung von Datenbanken aus Abitur-Aufgaben

aus rechtlichen Gründen wird hier nur kurz erläutert, wie man ev. an die Datenbanken kommt
Lehrer aus MV können in einer Moodle-Gruppe auf dem Landes-eigenem Bildungs-Server ua. auch auf die Abitur-Daten zurückgreifen
die Links in diesem Abschnitt funktionieren nur lokal. D.h. man muss diese Datenbanken in dem Ordner dieses Skriptes gespeichert haben. Ich selbst kann diese Datenbanken nicht über den eigenen Schulgebrauch hinaus verteilen!

aus Abitur Mecklenburg-Vorpommern 2017

praktikum.sqlite ("Praktikum")
zu Aufgabe 5

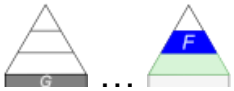
aus Abitur Mecklenburg-Vorpommern 2018

b1.sqlite ()
generiert mit einem Fake-Name-Generator (→ www.fakenamegenerator.com) (Lizenz für die Daten)

aus Abitur Mecklenburg-Vorpommern 2019

Surfen.sqlite (Surfen)
zu Aufgabe A1

5.4. "How to" für SQL / SQL-Koch-Anleitungen



Vielfach habe ich beobachtet, das war die einzelnen SQL-Anweisungen bekannt sind, aber das sich beim Zusammenstellen einer Abfrage ein riesiges Problem auftut.
Da Abfragen auch die Hauptnutzung von SQL in den Datenbank-Kursen (in Schulen) ist, beschränke ich mich hier zuerst einmal auch nur auf die Abfragen. Trotz alledem empfehle ich jedem Anfänger sich einen eigenen SQL-Spicker zu erstellen. Je nach Kurs-Art und Bewertung-Anforderungen ist der Spicker dann vielleicht auch durch den Kurs-Leiter für Lern-Kontrollen freigegeben.
Das "How to" ist für Anfänger / Einsteiger gedacht. Es werden nur typische und häufig benutzte Problemstellungen beachtet. Für komplexere Probleme bietet sich die Beschreibung der SQL-Befehle als Informations-Quelle an. Im "How to" werden die SQL-Anweisungen Tätigkeits- / Handlungs- / Ziel-orientiert vorgestellt. Eine systematische angelegte Vorstellung der SQL-Anweisungen erfolgte bereits in der Referenz (→ 5.1. wichtige SQL-Anweisungen).
U.U. hilft ein Nachlesen / Informieren an dieser Stelle weiter, als die Vorstellung im "How to".
Ein ausführlicheres Beispiel zur Konstruktion eines SQL-Statement's wird bei → 5.4.1.3. Erstellen einer (einfachen Daten-)Tabelle (Beispiel) aufgezeigt. Ev. kann auch der → EXKURS am Ende des "How to" genutzt werden.

Legende: **Beispiel(e)**

SQL-Bestandteile		
müssen exakt so geschrieben oder angeordnet werden (entspricht Terminalen)		
SELECT	obligatorische(r) SQL-Abweisung(steil); für das Problem notwendig	**CREATE TABLE**
SELECT	vor- oder nachlaufender SQL-Anweisung(s-Teil); zeigt die Lage einer obligatorischen Struktur in einem größeren Ausdruck an	
Variablen / Platzhalter		
müssen aus der Datenbank-Struktur stammen und wie dort geschrieben werden		
Tabelle	ein Objekt (Tabelle, Attribut, …) aus der genutzten Datenbank	
Ausdrücke / Formeln		
Meta-Symbole (Nicht-Terminale), die als Variablen / Konstruktions-Vorschriften von Ausdrücken dienen; müssen aus SQL-Bestandteilen, Variablen und / oder Werten zusammengestellt werden		
Bedingung	ein Objekt (Tabelle, Attribut, …) aus der genutzten Datenbank oder ein zusammengesetzter Ausdruck (aus Operation und Objekten)	**BETWEEN** 1 **AND** 100 ***Gehalt*** > 3000
Metazeichen		
dienen nur der verkürzten Schreibweise in den Ausdrücken; sie stellen Optionen, Wiederholungen oder Alternativen dar; sie werden nicht mitgeschrieben		
[…]	es kann weitere Teile der SQL-Anweisung geben; optionale Teile	
{…}	Wiederholungen (z.B. mehrere Tabellen)	***Tabelle*** **{** **,** ***Tabelle*** **}**
… **|** …	Alternative (entweder der linke oder der rechte Ausdruck	

5.4.1. Erstellen von Datenbanken, Tabellen und Indizes sowie Daten-Eingabe

5.4.1.1. Erstellen einer Datenbank

CREATE DATABASE *Datenbankname* [...]

Erläuterung:
Die ***Datenbank*** wird dem Datenbank-Management-System bekanntgegeben (Deklaration).

5.4.1.2. Verbinden mit … / Nutzen einer Datenbank

USE DATABASE *Datenbankname* [...]

Erläuterung:
Die angegebene ***Datenbank*** wird zum Arbeiten geöffnet.

5.4.1.3. Erstellen einer (einfachen Daten-)Tabelle

CREATE TABLE *Tabellenname* (*Struktur*)

CREATE TABLE *Tabellenname* (*Attribut* { , *Attribut* } PRIMARY KEY (*Spaltenname*))

CREATE TABLE *Tabellenname* (*Attribut* { , *Attribut* }
PRIMARY KEY (*Spaltenname* { , *Spaltenname* }))

Erläuterung:
CREATE TABLE erzeugt die ***Tabelle*** mit der in Klammern angegebenen ***Struktur***. Die ***Struktur*** ist eine Liste von ***Spaltennamen*** und ev. zugehörigen ***Bedingungen***.

typische Struktur (Attribut-Liste) ist:

- ***Attribut* { , *Attribut* }**

jedes ***Attribut*** besteht aus:

- ***Spaltenname Datentyp* [*Bedingung*]**

gültige ***Datentypen*** sind:

- **INT** für ganze Zahlen (z.B. gut für eine (zusätzliche) Schlüssel-Spalte geeignet)
- **DOUBLE oder REAL** für Zahlen mit Nachkommastellen (Gleitkomma-Zahlen)
- **DATE / TIME** für Kalender- bzw. Zeit-Daten
- **VARCHAR(*Zeichenanzahl*)** für Texte mit der Anzahl von Zeichen
- **BOOLEAN** für logische / Wahrheits-Werte (**TRUE** und **FALSE**)
- **BLOB** für zusätzliche – nicht genau spezifizierte – Daten (!! ohne feste Speicher-Reservierung)
- **…**

mögliche ***Bedingungen*** sind:

- **PRIMARY KEY** definiert die Schlüssel-Spalte(n)
- **NOT NULL** bestimmt, das in der Spalte immer Daten angebenen werden müssen (ein Leer-Lassen ist nicht zulässig)
- **FORREIGN KEY … REFERENCES** definiert einen Fremd-Schlüssel (siehe 5.4.1.4.) aus / zu einer anderen Tabelle
- **…**

Hinweis(e):
Bei der Definition des Primär-Schlüssels (Schlüssel-Attribut) sowie der Attribute können diverse weitere Bedingungen festgelegt werden. Siehe dazu in der SQL-Referenz.
Für praktische Tabellen emphiehlt sich die Anlage eines eigenständigen Primär-Schlüssel.
Die Änderung einer vorhandenen Tabellen-Struktur ist mit der Anweisung **ALTER TABLE** realisierbar (→ SQL-Referenz).
Oft wird empfohlen für Geld-Beträge den Daten-Typ **DECIMAL(8,2)** zu verwenden, um Rundungs-Problemen aus dem Weg zu gehen. DECIMAL(8,2) bedeutet, dass hier eine Festkommazahl mit insgesamt 8 Ziffern-Stellen bestimmt wird. Von den 8 Ziffernstellen sind 6 Vorkomma- und 2 Nachkomma-Stellen.

Beispiel für die schrittweise Zusammenstellung einer SQL-Anweisung (in diesem "How to"):

Ziel-Tabellen-Struktur:
Personen = (**PID**, Name, Vorname, GebDatum)

nur die schwarz gedruckten Teile des SQL-Ausdrucks werden in den Editor eingegeben!
man kann zu Anfang alle Ausdrucksteile auf einem Zettel mit Beistift schreiben, wegradieren und durch Inhalte ersetzen; gut funktioniert dies, wenn man für jeden Ausdruck (mit zugehörigen Meta-Zeichen) eine Zeile einplant (s.a. EXKURS am Ende des "How to")

CREATE TABLE *Tabellenname* (*Struktur*)

ersetzen der Variable "Tabellenname" durch den (Ziel-)Tabellen-Namen "Personen" und weiter beim Ausdruck "Struktur"

CREATE TABLE Personen **(*Struktur*)**

ersetzen des allgemeinen Ausdruck's "Struktur" durch den verfeinerten Ausdruck "Attribut {, Attribut }"

CREATE TABLE Personen **(*Attribut* { , *Attribut* })**

ersetzen des erstenHilfs-Ausdruck's "Attribut" durch eineSpezifikation aus "Spaltenname", "Datentyp" und ev. noch notwendigen "Bedingungen"

CREATE TABLE Personen **(*Spaltenname Datentyp* [*Bedingung*] { , *Attribut* })**

ersetzen der Ausdrücke "Spaltenname", "Datentyp" und ev. noch notwendigen "Bedingungen" durch die konkreten Angaben für die erste Spalte (1. Attribut)

CREATE TABLE Personen **(** PID INT PRIMARY KEY **{ , *Attribut* })**

nun weiter solange noch Attribute notwendig sind; der Ausdruck "Attribut" wird gleich durch die Konkreten Angaben ersetzt!

CREATE TABLE Personen **(** PID INT PRIMARY KEY, Name VARCHAR(30), **{ ,** ***Attribut*** **})**

CREATE TABLE Personen **(** PID **INT PRIMARY KEY,** Name **VARCHAR(30),**
Vorname **VARCHAR(30), { ,** ***Attribut*** **})**

CREATE TABLE Personen **(** PID **INT PRIMARY KEY,** Name **VARCHAR(30),**
Vorname **VARCHAR(30),** GebDatum **DATE)**

da der Nutzer jetzt beliebig viele leere Datensätze ohne Name, Vorname und Geburtsdatum erzeugen kann, sollte über eine Zwangs-Angabe von ev. Name und Vorname mit NOT NULL nachgedacht werden

CREATE TABLE Personen **(** PID **INT PRIMARY KEY,** Name **VARCHAR(30) NOT NULL,**
Vorname **VARCHAR(30) NOT NULL,** GebDatum **DATE)**

5.4.1.4. Erstellen einer Tabelle mit Verknüpfung zu einer anderen Tabelle / Erstellen einer Tabelle mit einer Referenz

```
CREATE TABLE Tabellenname ( Struktur )
```

```
CREATE TABLE Tabellenname (Attribut { , Attribut }
   PRIMARY KEY ( SpaltennameX )
   FOREIGN KEY ( SpaltennameY )
      REFERENCES ReferenzTabellenname( ReferenzSpaltenname )
)
```

```
CREATE TABLE Tabellenname (Attribut { , Attribut }
   PRIMARY KEY ( SpaltennameX )
   FOREIGN KEY ( SpaltennameY )
      REFERENCES ReferenzTabellenname( ReferenzSpaltenname )
   { , FOREIGN KEY ( Spaltenname? )
      REFERENCES ReferenzTabellenname( ReferenzSpaltenname ) }
)
```

Erläuterung:
Die Erzeugung einer Referenz (Fremdschlüssel) zu einer anderen (Referenz-)Tabelle erfolgt immer als Erweiterung einer typischen Tabellen-Definition (→ 5.4.1.3.; einschließlich eines Primärschlüssel's).

Hinweis(e):
Die Fremd-Schlüssel-Spalte sowie die Primär-Schlüssel-Spalte der referenzierten Tabelle müssen den gleichen Datentyp haben.
Die referenzierte Tabelle muss vorhanden sein. Für die Erstellung der Tabellen einer Datenbank ist also mit einer "Bottom up"-Strategie zu arbeiten.

5.4.1.x. einen Datensatz in eine Tabelle eingeben / importieren

5.4.1.x. einen Datensatz aus einer Tabelle entfernen / löschen

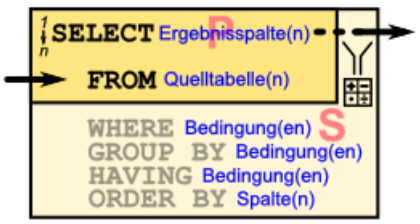
5.4.1.x. einen Datensatz in einer Tabelle verändern / modifizieren / einzelne Daten verändern

5.4.1.x. Erstellen eines Index

5.4.2. Abfragen

5.4.2.0. Grund-Schema für eine Abfrage

minimales Abfrage-Schema	komplexeres Abfrage-Schema
SELECT … **FROM …**	**SELECT …** **FROM …** **WHERE …** **GROUP BY …** **ORDER BY …**



Das minimale Schema kann immer gleich in das Eingabe-Feld für die SQL-Anweisung reingeschrieben oder reinkopiert werden.
Es empfiehlt sich für Anfänger anhand eines Synthax-Diagramm's (z.B.: → 5.1.11.2.1 Selektion für Fortgeschrittene:) die Wege durchzugehen und die notwendigen Terminale exakt zu übernehmen. Abweichungen von der Pfeil-Richtung sind nicht zulässig.
Die nachfolgende Reihenfolge ist Aufgaben-orientiert. Wenn man mit dem minimalen Abfrage-Schema beginnt, macht man nichts falsch.

Übungen zu den einzelnen Abfrage-Strukturen lassen sich z.B. an der Web-Datenbank Terra (→ https://www.sachsen.schule/~terra2014/) durchführen (ist mitlerweile auch aktualisiert worden!)

→ https://www.sachsen.schule/~terra2014/sql_abfragen.php

mySQL-Datenbank auch downloadbar:
https://www.sachsen.schule/~terra2014/terra2019oA.sql

Struktur der TERRA-Datenbank

Tabelle	Spalten
BERG	B_NAME GEBIRGE HOEHE JAHR LAENGE BREITE
GEO_BERG	id L_ID LT_ID B_NAME
EBENE	E_NAME HOEHE FLAECHE
GEO_SEE	id LT_ID L_ID S_NAME
FLUSS	F_NAME FLUSS SEE MEER LAENGE LAENGEU BREITEU LAENGEM BREITEM
SEE	S_NAME TIEFE FLAECHE

Tabelle	Spalten
LAND	L_ID L_NAME EINWOHNER FLAECHE HAUPTSTADT LT_ID
GEO_FLUSS	id L_ID LT_ID F_NAME
GEO_EBENE	LT_ID L_ID E_NAME
HAT_SITZ_IN	ST_NAME LT_ID L_ID ABKUERZUNG
LANDTEIL	id LT_ID L_ID LT_NAME EINWOHNER LAGE HAUPTSTADT
MEER	M_NAME TIEFE
GEO_WUESTE	LT_ID L_ID W_NAME

Tabelle	Spalten
STADT	ST_NAME L_ID LT_ID EINWOHNER BREITE LAENGE
GEO_MEER	id LT_ID L_ID M_NAME
GEO_INSEL	LT_ID L_ID I_NAME
UMFASST	id L_ID K_NAME PROZENT
LIEGT_AN	id ST_NAME LT_ID L_ID F_NAME S_NAME M_NAME
KONTINENT	K_NAME FLAECHE
aufgaben	id aufgabe loesung schwierigkeit hinweis

Tabelle	Spalten
IST_BENACHBART_ZU	id LAND1 LAND2
GEHT_UEBER_IN	MEER1 MEER2
INSEL	I_NAME INSELGRUPPE FLAECHE LAENGE BREITE

Tabelle	Spalten
IST_MITGLIED_VON	L_ID ABKUERZUNG ART
WUESTE	id W_NAME FLAECHE WUESTENART
ORGANISATION	id O_NAME ABKUERZUNG

Q: https://www.sachsen.schule/~terra2014/terra-struktur.php

5.4.2.1. Daten aus einer Tabelle verwenden

SELECT … FROM *Tabellenname* […]

Erläuterung:
Hinter **FROM** wird die ***Tabelle*** genannt, aus der die Daten für die Selektion stammen sollen.

notwendiger Vorlauf:
Auswahl von Tabellen-Spalten (Attributen)
- → 5.4.2.3. alle Spalten einer Tabelle auswählen / anzeigen ,
- → 5.4.2.4. einzelne Spalten einer Tabelle auswählen / anzeigen (Projektion) ,
- → …

mögliche Fortsetzungen:
- → 5.4.2.7. identische Ergebniszeilen verhindern (Pseudo-Selektion) ,
- → 5.4.2.8. bestimmte Datensätze / Zeilen auswählen (Selektion) ,
- → 5.4.2.14. die Datensätze / Zeilen gruppieren ,
- → 5.4.2.18. die Datensätze sortieren
- → …

5.4.2.2. Daten aus mehreren Tabellen verwenden

SELECT … FROM *Tabellenname* { , *Tabellenname* …} […]

Erläuterung:
Hinter **FROM** folgt eine Komma-getrennte Liste von ***Tabellen***, genannt, aus denen die Daten für die Selektion stammen sollen.

Hinweis(e):
Im **SELECT**-Teil müssen die ***Spalten*** (***Attribute***) eindeutig bezeichnet werden! Zu empfehlen ist immer die Schreibung: ***Tabellenname.Spaltenname*** für jedes einzelne ***Attribut***.

5.4.2.3. alle Spalten einer Tabelle auswählen / anzeigen

SELECT * **FROM** *…*

SELECT ALL FROM *…*

Erläuterung:
Das Sternchen (*) hinter **SELECT** umfasst alle (beliebige) Spalten der hinter **FROM** aufgezählten Tabelle. Das Schlüsselwort **ALL** umfasst ebenfalls alle Spalten.

5.4.2.4. einzelne Spalten einer Tabelle auswählen / anzeigen (Projektion)

SELECT *Spaltenname* { , *Spaltenname* } FROM *…*

Erläuterung:
Das Sternchen (*) hinter **SELECT** umfasst Komma-getrennten aufgezählten ***Spalten*** der hinter **FROM** aufgezählten Tabelle.

5.4.2.5. einzelne Spalten aus mehreren Tabellen auswählen / anzeigen (Projektion)

SELECT ***Tabellenname.Spaltenname*** **{** **,** ***Tabellenname.Spaltenname*** **}** **FROM** ***…***

Erläuterung:
Das Sternchen (*) hinter **SELECT** umfasst alle (beliebige) ***Spalten*** (in Punktschreibweise mit Angabe der ***Tabelle***) der hinter **FROM** aufgezählten ***Tabellen***.

5.4.2.6. Spaltennamen in der Ergebnis-Tabelle anpassen

SELECT ***Spaltenname*** **AS** ***neuerName*** **{** **,** ***Spaltenname*** **AS** ***neuerName*** **}** **FROM** ***…***

Erläuterung:
Mit dem Schlüsselwort **AS** kann jeder ***Tabellenspalte*** ein neuer ***Name*** vergeben werden.

Hinweis(e):
Die Hinweise bezüglich der eindeutigen Auswahl von Spalten (s.a. 5.4.5.) gelten weiterhin.
Eine Verwendung der Umbenennung mit AS ist nicht mit der vollständigen Auswahl aller Spalten mittels Sternchen (*****) möglich.

5.4.2.7. identische Ergebniszeilen verhindern (Pseudo-Selektion)

SELECT DISTINCT *Tabellenname.Spaltenname* **{ ,** *Tabellenname.Spaltenname* **}**
FROM *…*

SELECT DISTINCT *Spaltenname* **{ ,** *Spaltenname* **}**
FROM *…*

SELECT DISTINCT * **FROM** *…*

SELECT DISTICT ALL FROM *…*

Erläuterung:
Kommen in der Ergebnis-Tabelle Zeilen mit identischen Inhalten heraus, dann werden die Wiederholungen mittels **DISTINCT** unterdrückt und eine Zeilen-reduzierte Ergebnis-Tabelle ausgegeben.

Hinweis(e):
Bei Angabe von ***Spaltennamen*** müssen diese eindeutig einer ***Tabelle*** zuordnetbar sein! Es wird empfohlen bei der Nutzung mehrerer ***Tabellen*** die ***Spalten*** immer mittels Punkt-Schreibweise eindeutig zu charakterisieren.

5.4.2.8. bestimmte Datensätze / Zeilen auswählen (Selektion)

… **FROM** *…* **WHERE** ***Bedingung*** **[…]**

… **FROM** *…* **WHERE** ***Bedingung*** **{ ,** ***Bedingung*** **} […]**

… **FROM** *…* **WHERE** ***Bedingung*** **{** ***logischeOperation Bedingung*** **} […]**

Erläuterung:
Mittels **WHERE** wird eine Auswahl der Datensätze vorgenommen. Die Datensätze müssen die ***Bedingung(en)*** erfüllen.

typische ***Bedingungen*** sind:

- Vergleiche (z.B.: ***Spaltenname*** **=** 'Meier'); weitere Vergleichszeichen: **<**, **<=**, **>=**, **>** bzw. **!=** (für ungleich)
- Identität-Suche mit **IS** ***Wert*** (z.B.: ***Spaltenname*** **IS** "Meier")
- Bereichs-Suche mit **BETWEEN** ***Bereichsausdruck*** (z.B.: ***Spaltenname*** **BETWEEN** 100 **AND** 1000)
- Suche in einer Menge mit **IN** ***Menge*** (z.B.: ***Spaltenname*** **IN (**'Meier'**,** 'Meyer'**)**
- Ähnlichkeiten-Suche mit **LIKE** ***Filterausdruck*** (z.B.: ***Spaltenname*** **LIKE** "M??er")

logische Operatoren / Verknüpfungen sind:

- **AND** logische UND-Verknüpfung (beide Bedingungen müssen wahr sein)
- **OR** logische ODER-Verknüpfung (mindestens eine Bedingung muss wahr sein)
- **NOT** logische Negation der nachfolgenden Bedingung / Aussage (aus wahr wird falsch, aus falsch wird wahr)

Filterausdrücke können die folgenden Joker-Zeichen enthalten:

- **%** steht für beliebige Zeichen und Zeichenketten
- **?** steht für genau ein beliebiges Zeichen

Wert kann:

- ein beliebiger Daten-Eintrag (Feld) sein (sachlich ähnlich zu Gleichheit)
- eine leere Zelle / ein leeres Feld sein → **NULL**

ein ***Bereichsausdruck*** ist durch zwei ***Werte*** als Grenzen und das dazwischenliegende **AND** charakterisiert

5.4.2.9. Datensätze zählen (Aggregation)

```
SELECT COUNT(*) FROM …
SELECT COUNT( Spaltenname ) FROM …
SELECT COUNT( Spaltenname ) AS neuerSpaltenname FROM …
```

Erläuterung:
Es werden die Zeilen der Ergebnis-Tabelle gezählt. Im Allgemeinen ist die Angabe eines Namens für die Zähl-Ergebnis-Spalte sinnvoll.

5.4.2.10. das Minimum in einer Spalte ermitteln

```
SELECT MIN( Spaltenname ) FROM …
SELECT MIN( Spaltenname ) AS neuerSpaltenname FROM …
```

Erläuterung:
Es wird der kleinste Wert / Inhalt / das **Minimum** in der ***Spalte*** der Ergebnis-Tabelle ermittelt. Im Allgemeinen ist die Angabe eines Namens für die Ergebnis-Spalte (mit dem Minimum) sinnvoll.

5.4.2.11. das Maximum in einer Spalte ermitteln

```
SELECT MAX( Spaltenname ) FROM …
SELECT MAX( Spaltenname ) AS neuerSpaltenname FROM …
```

Erläuterung:
Es wird der grösste Wert / Inhalt / das **Maximum** in der ***Spalte*** der Ergebnis-Tabelle ermittelt. Im Allgemeinen ist die Angabe eines Namens für die Ergebnis-Spalte (mit dem Maximum) sinnvoll.

5.4.2.12. den Durchschnitt / Mittelwert einer Spalte ermitteln

```
SELECT AVG( Spaltenname ) FROM …
SELECT AVG( Spaltenname ) AS neuerSpaltenname FROM …
```

Erläuterung:
Es wird der **Durchschnitt** / **arithmetrische Mittelwert** der ***Spalte*** der Ergebnis-Tabelle ermittelt. Im Allgemeinen ist die Angabe eines Namens für die Ergebnis-Spalte (mit dem Durchschnitt) sinnvoll.

5.4.2.13. die Werte einer Spalte summieren

```
SELECT SUM( Spaltenname ) FROM …
SELECT SUM( Spaltenname ) AS neuerSpaltenname FROM …
```

Erläuterung:
Es wird der **Durchschnitt** / **arithmetrische Mittelwert** der ***Spalte*** der Ergebnis-Tabelle ermittelt. Im Allgemeinen ist die Angabe eines Namens für die Ergebnis-Spalte (mit der Summe) sinnvoll.

5.4.2.14. die Datensätze / Zeilen gruppieren

```
… FROM […] … GROUP BY Spaltenname […]
… FROM […] … GROUP BY Spaltenname { , Spaltenname } […]
```

Erläuterung:

Dannach können z.B. noch Sortierungen folgen → .

5.4.2.15. die Datensätze / Zeilen in einer Gruppe zählen

```
SELECT COUNT(*) FROM … [ WHERE … ] GROUP BY Spaltenname
SELECT Spaltenname { , Spaltenname }  COUNT(ZählSpaltenname)
FROM … [ WHERE … ] GROUP BY ZählSpaltenname
```

Erläuterung:
Mit **COUNT()** gleich hinter **SELECT** wird eine Ergebnis-Tabelle für alle (wegen Sternchen (*****)) bzw. die angegebenen ***Spalten*** mit einer Zähl-Spalte angelegt angelegt. Diese ist wird nach gleichartigen Einträgen in der genannten ***Spalte*** (hinter **GROUP BY**) gruppiert.

5.4.2.16. die Werte einer Gruppe summieren

```
SELECT COUNT(*) FROM … [ WHERE … ] GROUP BY Spaltenname
```

```
SELECT Spaltenname { , Spaltenname } COUNT(ZählSpaltenname)
FROM … [ WHERE … ] GROUP BY ZählSpaltenname
```

Erläuterung:
Mit **COUNT()** gleich hinter **SELECT** wird eine Ergebnis-Tabelle für alle (wegen Sternchen (*****)) bzw. die angegebenen ***Spalten*** mit einer Zähl-Spalte angelegt angelegt. Diese ist wird nach gleichartigen Einträgen in der genannten ***Spalte*** (hinter **GROUP BY**) gruppiert.

5.4.2.17. in einer Gruppe von Datensätzen / Zeilen den Mittelwert ermitteln

```
… FROM […]  …
```

Erläuterung:

5.4.2.18. die Datensätze sortieren

```
SELECT … FROM … WHERE … SORT BY Spaltenname Bedingung
```

```
SELECT … FROM … WHERE …
   SORT BY Spaltenname Bedingung { , Spaltenname Bedingung }
```

Erläuterung:
Die Definition einer Sortierung erfolgt praktisch am Ende der SQL-Anweisung. Neben der ***Spalte*** – nach der die Ergebnis-Tabelle sortiert werden soll – kann als ***Bedingung*** die Sortier-Richtung angegeben werden. Es sind mehrere (gestaffelte) Sortierungen möglich.

Hinweis(e):
Bedingung: Es wird eine aufsteigende Sortier-Richtung angenommen (, diese kann aber auch mit angehängtem **ASC** extra bestimmt werden). Die umgekehrte Sortier-Richtung erreicht man mit angehängtem **DESC**.

HAVING

als "Alternative" zu WHERE

5.4.2.x. zwei Tabellen vereinen / einen inneren Verbund zwischen zwei Tabellen herstellen

JOIN

Exkurs: SQL-Anweisungen erstellen für Nicht-Affine

Dieser Exkurs ist vorrangig für solche SQL-Anwender gedacht, denen die hintereinander weggeschriebenen Statements zu unübersichtlich sind. Mit der folgenden Arbeitstechnik kann man sich als nicht so Affiner das Leben etwas leichter machen. Wir greifen auf die gleichen Definitionen zurück, die wir gerade im "How to" aufgezeigt haben. Als erstes Arbeitsmittel verwenden wir ein Blatt Papier, einen Radierer und einen Bleistift. Wer will kann am Schluß die fertigen Statements mit einem kräftigeren Stift nachschreiben.
Zuerst suchen wir uns die Definition für unser zu bearbeitendes Problem heraus. So könnte unser Problem sein, Name und Vorname aus einer Personen-Tabelle herauszusuchen, wobei nur die Namen zwischen M und R dabei sein sollen. Die Ergebnistabelle soll umgekehrt nach den Namen sortiert werden.
Statt es nun so auch auf's Papier zu schreiben, zerlegen wir es in sinnvolle / syntaktisch passende Zeilen.

Unser erstes Problem ist also das Anzeigen Daten aus einer Tabelle → 5.4.2.1. Wir übernehmen also die dort vorgeschlagene Struktur (s.a. Abschnitts-Anfang) Zeilen-weise auf's Papier.	SELECT FROM ***Tabellenname***
Tabellenname ist von uns zu ersetzen, also radieren wir Tabellenname weg und notieren dafür: Personen Als nächstes war angegeben, dass nur einzelne Spalten angezeigt werden sollen. Dafür ist 5.4.2.4. geeignet.	SELECT FROM Personen
Natürlich könnte man auch gleich bei 5.4.2.4. starten. Die notwendigen Änderungen liegen zwischen SELECT und FROM, also tragen wir sie dort ein. Zuerst einmal als Meta-Ausdruck.	SELECT ***Spaltenname*** **{** **,** ***Spaltenname*** **}** FROM Personen
Diesen können wir wegradieren und durch die geforderten (anzuzeigenden) Spalten ersetzen. An dieser Stelle können wir das SQL-Statemant auch schon mal ausprobieren.	SELECT Name, Vorname FROM Personen
Ev. offenbaren sich hier Tipp-Fehler oder falsche Bezeichnungen.	
Die Auswahl von Datebnsätzen wird im Punkt 5.4.2.8. beschrieben. Es handelt sich um eine WHERE-Erweiterung hinter dem FROM-Abschnitt. Diesen Konstrukt übernehmen wir wieder.	SELECT Name, Vorname FROM Personen WHERE ***Bedingung***
Nun suchen wir etwas in einem Bereich. Dazu kann man z.B. den Bedingungs-Ausdruck mit BETWEEN benutzen. Dieser ersetzt den rauszuradierenden Bedingungs-Ausdruck. Das Einrücken dient einer übersichtlicheren Darstellung der zusammengehörenden Strukturen und einem leichteren Herausradieren.	SELECT Name, Vorname FROM Personen WHERE ***Spaltenname*** **BETWEEN** ***Wert*** **AND** ***Wert***
Auch die beiden Wert-Audrücke müssen noch konkretisiert werden. D.h. wir geben die untere und obere Grenze – also M und R an. →	SELECT Name, Vorname FROM Personen WHERE Name BETWEEN ***Wert*** **AND** ***Wert***

Da bis hierhin alle Ersatz-Ausdrücke aufgelöst sind, kann man auch wieder einen Versuch und das Statement testen. Würde das Zwischen-Ergebnis nicht passen, dann wäre ein neuer Ansatz zu empfehlen.	SELECT Name, Vorname FROM Personen WHERE Name BETWEEN 'M' AND 'R'
Bleibt als letzte Forderung die Sortierung nach dem Namen. Der Blick in die Lösungs-Vorschrift 5.4.2.18. besagt ein Anhängen der SORT BY-Anweisung an den WHERE-Bedingungs-Teil. Die Bedingung gehört zwar syntaktisch gleich hinter den Spaltennamen. Dann würden wir aber zwei zu radierende Ausdrücke gleich hintereinander haben. Da könnte es Überschneidungen geben.	SELECT Name, Vorname FROM Personen WHERE Name BETWEEN 'M' AND 'R' SORT BY ***Spaltenname*** ***Bedingung***
Zuerst ersetzen wir Spaltenname. Da nach den (Personen-)Namen sortiert werden soll ist das schnell realisiert. SORT BY lässt auch noch Bedingungen zu. Das sind die Sortier-Reihenfolgen / -Richtungen. Eine normale – aufsteigende – Sortierung bräuchten wir nicht weiter zu spezifizieren. Wer will kann auch betont ein ASC angeben.	SELECT Name, Vorname FROM Personen WHERE Name BETWEEN 'M' AND 'R' SORT BY Name ***Bedingung***
Die umgekehrte – absteigende – Sortierung muss aber unbedingt durch DESC gekennzeichnet werden. Somit wären wir fertig. Das Statement kann original so in den SQL-Editor übertragen werden. Diesen stören Zeilen-Umbrüche und Einrückungen nicht. Der hintereinanderweg geschriebene Text funktioniert genauso, ist aber deutlich schwerer zu lesen.	SELECT Name, Vorname FROM Personen WHERE Name BETWEEN 'M' AND 'R' SORT BY Name DESC
SELECT Name, Vorname FROM Personen WHERE Name BETWEEN 'M' AND 'R' SORT BY Name DESC	
Auch weitere Änderungen / Verbesserungen / Erweiterungen lassen sich an dem Zeilen-strukturierten Ausdruck deutlich besser vornehmen.	

5.5. Anleitung zum Lesen / Interpretieren von SQL-Anweisungen

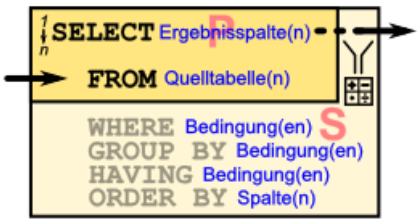
Eigentlich sollte die Struktur von SQL auch wenig geschulte (englisch-sprachige) Anwender in die Lage versetzen, den Inhalt bzw. den Zweck eines Ausdruck's zu verstehen. Das sollte zum Einen für das Lesen von SQL-Statement's gelten, als auch für die Erstellung. Eine vorgebildete (englisch-sprachige) Sekretärin oder Sachbearbeiterin sollte SQL-Ausdrücke formulieren können. Mit der immer größer werdenden Komplexität heutiger Datenbanken und der Sprache SQL hat dieser Anspruch schon deutlich gelitten. Mit den "Problemen" der Erstellung haben wir uns ja schon im Abschnitt (→) beschäftigt.
Hier wollen wir nun einen Ansatz liefern, wie man SQL-Statement's in verständliche – mehr oder weniger umgangs-sprachliche – Aussagen umsetzen kann.

> Wenn manche Lehrer wissen würden,
> wie schwer ihre Aufgaben für Schüler sind
> und wenn man diese ohne ihr eigenes Zusatzwissen
> bzw. vorgegebene Lösungs-Blätter lösen muss,
> dann würden sie diese nicht so stellen.

Besteht die Möglichkeit, ein SQL-Statement an der originalen Datenbank ablaufen zu lassen, dann hilft das ungemein. Aus dem Vergleich der Ausgangs-Tabellen und der Ergebnis-Tabelle (der Abfrage) lassen sich viele Wirkungen von speziellen SQL-Ausdrücken viel leichter erkennen.
Ein SQL-Statement ohne Datenhintergrund zu verstehen ist z.T. wirklich schwierig.
Praktisch tauchen Probleme mit (normalen) SQL-Ausdrücken nur im Zusammenhang mit Abfragen auf. Andere Anweisungen sind i.A. gut zu lesen und zu verstehen.
Wir beschränken uns hier deshalb auf die Zerlegung / Analyse von Abfragen. Das Grund-Prinzip lässt sich aber auch auf andere SQL-Strukturen übertragen.

5.5.1. Strukturieren der SQL-Anweisung / Abfrage

Zuerst schreiben wir die SQL-Anweisung so in mehrere Zeilen um, dass immer ein Schlüsselwort vorne steht. In guten SQL-Programmen und von umsichtigen SQL-Anwendern erstelle SQL-Statements sind das vielleicht schon. I.A. kann man im Editor (der DB-Management-Software) die SQL-Anweisungen mit [Enter] ohne Gefahr "umgestalten".
Solche Strukturierungen ignoriert die Datenbank und für uns Menschen werden die Ausdrücke deutlich besser lesbar. Besonders bei **AS**-Ausdrücken ist das zuerste sehr hilfreich. Sie müssen vielfach erst ganz zum Schluß mit einbezogen werden.
Auch Einrückungen mit Leerzeichen für untergeordnete Strukturen werden von den meisten Editoren "überlesen".



originaler Beispiel-SQL-Ausdruck:

SELECT FROM WHERE

umstrukturierter Beispiel-SQL-Ausdruck:

SELECT
FROM
WHERE

5.5.2. "Übersetzen" der Ausdrücke

Im zweiten Schritt übersetzen wir die einzelnen Zeilen so, dass die Funktion verständlich wird. Vielfach ist das ja schon von sich aus leicht möglich. Bei einigen Strukturen müssen wir aber Obacht geben.

Übersetzungs-Hilfen /-Floskeln / …

SQL-Ausdruck	Übertragung / Übersetzung	Bemerkungen

Übersetzung des Beispiel-SQL-Ausdruck's

Versuchen wir nun diese allgemeinen Formulierungen auf unser konkretes Beispiel anzuwenden.

SQL-Ausdruck	Übertragung / Übersetzung	Bemerkungen
SELECT		
FROM		

5.5.3. Abgleich der Angaben mit der Datenbank-Struktur / Ableiten der Teil-Struktur der Datenbank

5.5.4. Erstellen einer fach- oder umgangs-sprachlichen Beschreibung

Literatur und Quellen:

/1/
ISBN

/2/
ISBN

/3/
ISBN

/4/
ISBN

/5/ GIERHARDT, Horst:
Datenbanken: Entity-Relationship-Model.-Bad Laasphe, Städtisches Gymnasium, 2014.-horst@gierhardt.de
(http://www.oberstufeninformatik.de/Datenbanken/ERMTheorie.pdf)

/10/ SELLE, Stefan:
Künstliche Neuronale Netzwerke und Deep Learning.-Saarbrücken (2018)

Die originalen sowie detailliertere bibliographische Angaben zu den meisten Literaturquellen sind im Internet unter http://dnb.ddb.de zu finden.

Kurz-Referenz auf Quelle

/###/ Syntax-Diagramme für SQLite
https://www.sqlite.org/syntaxdiagrams.html

Internet-Seiten, etc.

/A/ Wikipedia
http://de.wikipedia.org

Abbildungen und Skizzen entstammen den folgende ClipArt-Sammlungen:

/A/ 29.000 Mega ClipArts; NBG EDV Handels- und Verlags AG; 1997

/B/

andere Quellen sind direkt angegeben.

Alle anderen Abbildungen sind geistiges Eigentum:

/I/ lern-soft-projekt: drews (c,p) 1997 - 2024 lsp: dre

verwendete freie Software:

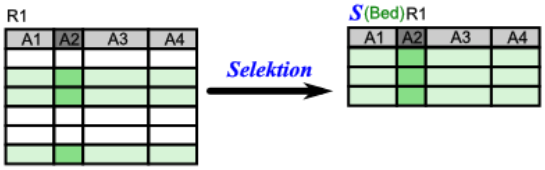
- **Inkscape** von: inkscape.org (www.inkscape.org)
- **CmapTools** von: Institute for Human and Maschine Cognition (www.ihmc.us)

⊞- (c,p) 2015 - 2024 lern-soft-projekt: drews -⊟
⊞- drews@lern-soft-projekt.de -⊟
⊞- http://www.lern-soft-projekt.de -⊟
⊞- 18069 Rostock; Luise-Otto-Peters-Ring 25 -⊟
⊞- Tel/AB (0381) 760 12 18 FAX 760 12 11 -⊟

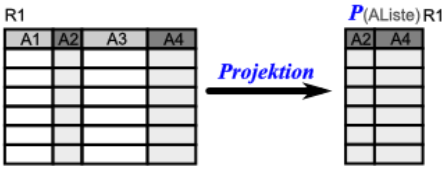
derzeit Aussortiertes

Methoden für eine Tabelle (oder Abfrage)

- **Selektion** — Auswahl von Zeilen (Tupeln) aus einer Relation aufgrund einer oder mehrerer Bedingungen

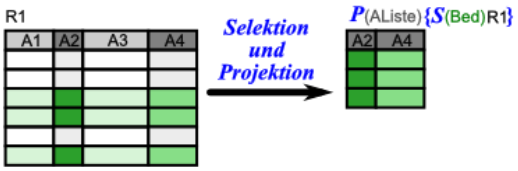


- **Projektion** — Auswahl (/ Einschränkung) von Spalten (Attribute) aus einer Relation aufgrund einer Auswahlliste



möglich auch Kombinationen, praktisch sinnvoll um die anzuzeigende / bereitgestellte Daten-Menge noch weiter zu reduzieren:

- **Selektion und Projektion** — Auswahl von Zeilen (Tupeln) und auch (Einschränkung) von Spalten (Attributen) aus einer Relation

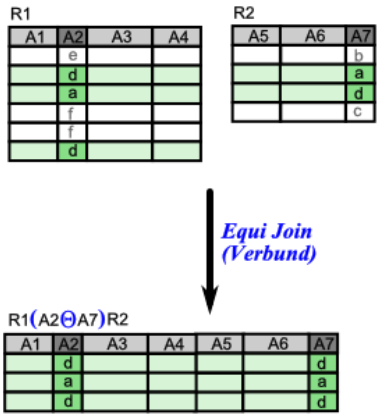


-
-

Methoden für mehrere Tabellen und / oder Abfragen

- **Verbund**
  Join
  (Equi Join)

  Verknüpfung von Relationen (Tabellen) aufgrund von Attributs-Beziehungen



- **Mengen-Operation**

  Bildung von Vereinigung, Differenz und / oder Durchschnitt auf Relationen mit gleicher Struktur