Datenbanken

Literatur

- Date C.J., An introduction to database systems, Addison-Wesley
- Rauh/Stickel, Konzeptuelle Datenmodellierung, Teubner-Verlag
- Chen P., The Entity-Relationship-Model towards a unified view of data

1. Datenbank-Architektur

-> Folie 1/3, Fig. 1.4

Das Ansi-Sparc-Modell beschreibt das grundlegende Modell einer Datenbank-Architektur als interne, konzeptionelle und externe Ebene.

-> Folie 1/5, Fig. 2.3

nicht Klausur relevant

- Es gibt datendefinierende (DDL = Data Definition Language) und datenmanipulierende Sprachelemente (DML = Data Manipulation Language).
- DDL für die interne Ebene heißt manchmal auch SDL (Storage definition Language)
- SQL ist im wesentlichen eine DDL für die konzeptionelle Ebene (z.B. create Table) und für die externe Ebene (z.B. create View). Es gibt aber auch einige wenige Befehle für die interne Ebene (z.B. create Index)
- DSL (Data sublanguage) wird vom Anwendungsprogrammierer benutzt (z.B. eingebettetes SQL)

1.1. Interne Ebene

- die tatsächliche physische Speicherung von Daten
- sollte für den Anwendungsprogrammierer unsichtbar sein

1.2. Konzeptionelle Ebene

- abstrahiert von der tatsächlichen physichen Speicherung der Daten
- Datenmodell für den gesamten Informationsbereich
- repräsentiert den gesamten Informationsgehalt der Datenbank
- Sicht der „Community of Users" - Datenbankadministratoren

1.3. Externe Ebene (Subschema, Sicht/View)

- Ebene der einzelnen Anwendung / des Endbenutzers
- Jeder Nutzer hat seine individuelle (externe) Sicht auf die Daten
- Beschreibung der Daten aus Sicht der einzelnen Anwendung

2. Entity-Relationship (ER)-Diagramme

- Eine Entität ist ein Datensatz — streng nur Datensatz
- Ein Entitätstyp besitzt Attribute
- Ein Attribut besitzt einen bestimmten Wertebereich Datum, Zahl, ect.
- Beziehungstypen (Relationstypen) stellen Verbindungen zu Entitätstypen her

z.B. ein Raum - B1 alle Räume

-> Folie 2/1

Definition der Entitäts-Integrität („Ein-Elementiger Unterschied"):

darf nicht zweimal vorkommen z.B. Daniel Düsentrieb darf nur einmal vorkommen. (mit identischen Daten)

- Sei E ein Entitätstyp mit Attributen $a_1, ..., a_n$
- Seien x,y Entitäten vom Typ E,

- dann gilt mit $x \neq y \Rightarrow$ für mindestens ein i ist $a_i(x) \neq a_i(y)$

2.1. *Kardinalität (Grad der Beziehung)*

2.1.1. m:n -Beziehung

- Ein Student besucht n Vorlesungen
- Eine Vorlesung wird von m Studenten besucht

2.1.2. 1:n - Beziehung und Min-Max-Notation

- Jede Rechnung besteht aus mindestens 1, max. n Positionen
- Jede Position gehört zu mindestens 1, max. 1 Rechnung

Bemerkung: Die Min-Max-Notation {(1:n), (1:1)} würde aufgelöst (1:n) lauten !!

-> Folie 2/2, 2/3, 2/4

2.1.3. Schlüssel- und Schlüsselkandidaten

Ein Schlüsselkandidat (möglicher Schlüssel) ist ein/sind Attribut(e) A vom Entity-Typ E, so daß jede Entität vom Typ E durch den Wert von A festgelegt ist.

Minimale Attributkombination:

- Ein Schlüssel ist nur dann ein Schlüssel, wenn er Schlüsselkandidat ist und bei Entfernung eines Attributes kein Schlüsselkandidat mehr ist.
- z.B. Student = @Matr.Nr + @Name + @Adresse + Fach + Semester
  (@Matr.Nr + @Name + @Adresse) ist zwar Schlüsselkandidat aber kein Schlüssel, da (@Matr.Nr + @Name) ebenfalls Schlüsselkandidat ist, etc.

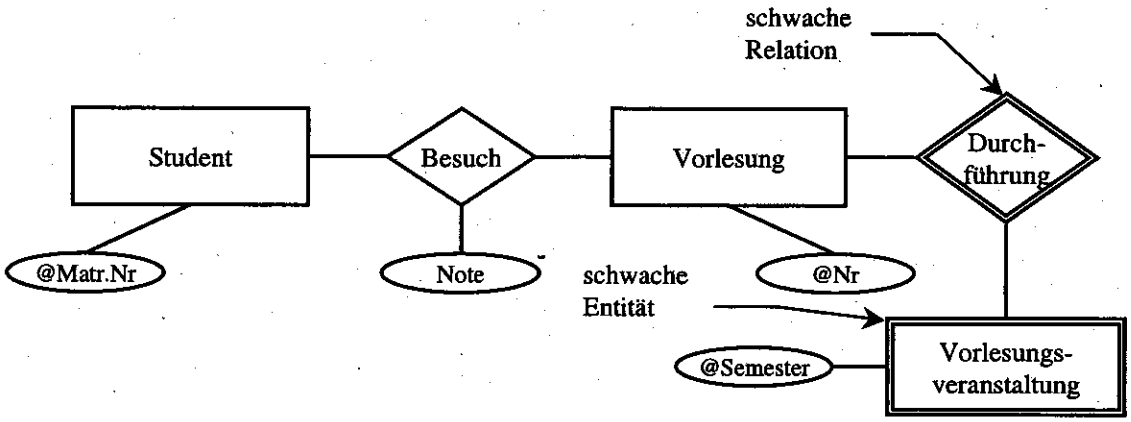
-> Folie 2/5, 2/6

2.1.4. Schwache/singuläre Entität

- Ein Entitytyp E heißt schwach oder singulär, wenn die Existenz einer Entität des Typs E von der Existenz einer anderen Entität abhängt
- Ein Relationstyp R (Beziehung zweier Entitätstypen) heißt schwach oder singulär, wenn einer der beteiligten Entitätstypen schwach/singulär ist

-> Folie 2/7, 2/8

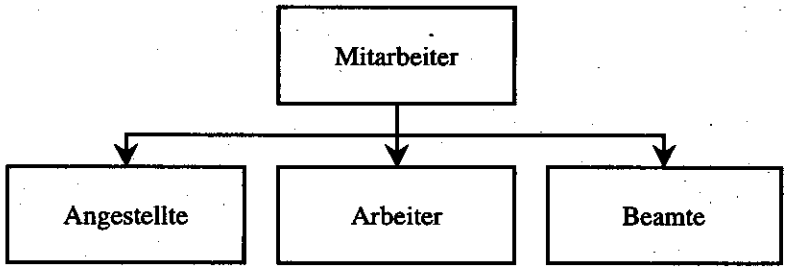
2.1.5. Primärschlüssel einer Relation



- Primärschlüssel von Besuch ist üblicherweise (@Matr.Nr, @VorlesungsNr) (Zusammenfassung der beteiligten Entitäten)

-> Folie 2/9, 2/10

2.1.6. Erweiterungen des ER-Modells



Spezialisierung/Generalisierung von Entitätstypen:

- der Typ „Arbeiter" ist ein Untertyp des Typs „Mitarbeiter" (Unterklasse oder abgeleitete Klasse)
- Ein Untertyp besitzt alle Attribute und Beziehungen des Obertyps (er erbt)

-> Übung: Entwurf eines ER-Modells

2.2. Datenbankmodelle

In der Anfangszeit der IT gab es überhaupt keine Datenbanken. Die Anwendungssoftware musste die Zugriffslogik auf die Daten noch selbst realisieren. Erst mit der wachsenden Zahl von Anwendungen und den immer größeren Datenmengen war eine Flexibilität erforderlich, die im DBMS durch die Trennung von Anwendung, Zugriffslogik und Datenbestand umgesetzt wurde.

2.2.1. Hierarchisch

Das hierarchische Datenmodell entstand aus sequenziellen Datensätzen, indem man Wiederholungsgruppen hierarchisch strukturierte. Die voneinander abhängigen Datensätze werden als Einheit behandelt.

Der Zugriff auf einzelne Daten muss über den Suchschlüssel der obersten Ebene erfolgen, dabei muss die Anwendungssoftware den Zugriffspfad kennen. Ist diese Voraussetzung erfüllt, lassen sich sehr gute Zugriffsgeschwindigkeiten realisieren.

Das Modell kann vor allem Eins-zu-n-Strukturen gut wiedergeben (z.B. Verzeichnisbäume von Dateisystemen), Beziehungen zwischen Datensätzen, die in verschiedenen „Ästen" oder Ebenen gespeichert sind, können jedoch nicht abgebildet werden.

Die Beziehungen zwischen Datensätzen müssen mit Zeigern - also physikalischen Speicheradressen - realisiert werden. Das heißt, daß die logische Struktur einer Datenbank auf der physikalischen Struktur des Dateisystems basiert, und nicht auf einer begrifflichen Abstraktion. Es gibt daher keine deklarative Abfragesprache. Um Informationen wiederzugewinnen, muß man entlang der Zeiger durch die Datenbank navigieren, was umständlich zu programmieren und fehleranfällig ist.

2.2.2. Relational

Der Vorteil dieser Systeme besteht darin, dass Abfragen relativ bequem mit SQL, also in einem standardisierten Sprachcode, formuliert werden können - zuvor musste jede Information durch aufwendige Programmierung abgerufen werden.

Mit relationalen DBMS war es erstmals möglich, situationsspezifische Berichte schnell und einfach zu generieren. Unternehmen konnten so schneller auf Änderungen in ihrem Markt reagieren, weil wesentlich detailliertere Auswertungen möglich waren. Im relationalen Modell werden die Daten in Tabellenstrukturen, die sich aus Spalten (Felder) und Zeilen (Datensätzen) zusammensetzen, gespeichert. Diese Tabellen dürfen keine redundanten Daten enthalten, so dass für jede Art von Informationen separate Tabellen angelegt werden müssen (für Kunden, Rechnungen, Rechnungspositionen etc).

Die verschiedenen Tabellen sind untereinander nur über Primary-Key- und Foreign-Key-Felder verbunden. Aus den gemeinsamen Spalten verschiedener Tabellen lassen sich dann spezielle Indextabellen erstellen, über die zur Laufzeit schnell Relationen zwischen den Tabellen gebildet werden können (logische Beziehungen, die den gesamten Komplex unterschiedlicher Tabellen für den Anwender als einheitlichen Datenpool darstellen).

Das hört sich nicht nur kompliziert an - es bedarf auch einer aufwendigen technischen Umsetzung. Aus diesem Grund kommt es sehr auf die Implementierung im jeweiligen DBMS an - ein weites Feld für den Wettbewerb der Anbieter von relationalen DBMS.

Das Modell hat Vor- und Nachteile. Zum einen ist es sehr flexibel, da die Zugriffswege nicht von vornherein fest programmiert werden müssen, sondern erst bei der Ausführung auf der Ebene des DBMS berechnet werden. Zum anderen begrenzt aber genau dieses Verfahren die Performance. Je mehr Tabellen beteiligt sind, desto komplexer sind die Berechnungen, die bei der Programmausführung immer wieder aufs Neue zur Verknüpfung der Daten vorgenommen werden müssen. Außerdem werden dabei hohe Ansprüche an Rechenleistung und Speicherkapazität gestellt.

Seinen bahnbrechenden Aufsatz „A Relational Model of Data for Large Shared Data Banks" hat Edgar Codd, der Erfinder des relationalen Datenbankmodells, zwar schon 1970 publiziert, aber erst Mitte der achtziger Jahre wurden kommerzielle RDB-Systeme angeboten. Das relationale Datenbankmanagementsystem DB2 von IBM zum Beispiel kam 1983 auf den Markt.

Das relationale Datenmodell ist relativ einfache Mengenlehre. Eine Relation R über W_1, ..., W_n besteht aus einem Relationenkopf (auch Relationsschema genannt) und einem Relationenkörper

- Der Relationenkopf ist eine feste Menge von Attributen $\{A_1, ..., A_n\}$ mit Wertebereich W_i (von A_i). n heißt Grad der Relation

- Der Relationenkörper ist eine endliche Menge von Tupeln über $A_1, ..., A_n$. Ein Tupel ist eine Menge $\{(A_1:w_1), (A_2:w_2), ..., (A_n:w_n)\}$ von Attribut-Wert-Paaren $(A_i:w_i)$ mit w_i aus W_i.

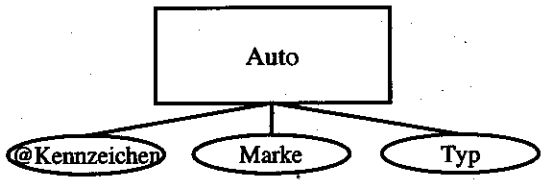
Relationenkopf →
Relationenkörper →

@Kennzeichen	Marke	Typ	Baujahr
PI-DR 346	VW	Bus	1984
B-UM 9	Landrover	109	1974
PI-DR 347	Ferrari	F40	1992

Grad der Relation

- Die ganze Tabelle ist die Relation (Reihenfolge der Spalten bei der Relation ohne Belang)
- ein Tupel entspricht einem Datensatz
- 2 Tupel mit denselben Attributwerten sind gleich

Auto als Entitätstyp:



Auto als Relation:

Auto = {(Kennzeichen : {N-ZZ-999, B-A-1}), (Marke: {VW, Opel}), ...}

In EBNF: (Erweitere Backus-Naur-Form)

Auto = @Kennzeichen + Marke + Typ

2.2.3. Netzwerk

In Netzwerkdatenbanken werden die Beziehungen zwischen den Sätzen verschiedener Tabellen nicht mit Indizes, sondern mit Pointern hergestellt. Diese werden schon beim Datenbankdesign festgelegt, also „fest verdrahtet". Der Zugriff auf Daten, das Navigieren in der Datenbank, kann auf diese Weise sehr schnell und ohne großen Speicherbedarf erfolgen. Der Pointer kennt den Zugriffsweg bereits, und das DBMS muss folglich beim Zugriff nichts berechnen. Zugleich wird Speicherplatz gespart, weil nicht für jede Tabelle separate Indexdateien angelegt werden müssen.

2.2.4. Gegenüberstellung

Die Datenbanktechnologie ist nicht eine stetige Fortentwicklung vom Schlechteren zum Besseren, sondern die Entwicklung von verschiedenen Modellen für verschiedene Aufgabenstellungen. Die simple Gleichung relational ist besser als hierarchisch ist Unsinn. Jede dieser Technologien hat ihren passenden Einsatzbereich, der je nach Verfassung der IT-Landschaft mal größer und mal kleiner ausfällt.

Der kurze Überblick über die beiden technologischen Richtungen zeigt, dass beide ihre Vor- und Nachteile haben. Hierarchische Systeme sind in Sachen Datenzugriff immer noch unerreicht schnell. Nicht zufällig sind nach wie vor die schnellsten Datenbanksysteme

nicht relationale, sondern hierarchische DBMS. Demgegenüber sind relationale DBMS unübertroffen in ihrer Flexibilität. Diese Unterschiede sind nicht Folge der mehr oder weniger guten Umsetzung des jeweiligen Konzepts, sondern grundlegender Natur.

Mit gleichem Aufwand wird eine hierarchische Datenbank immer schneller auf Daten zugreifen, eine relationale wird Daten immer effizienter speichern können. Ein wichtiger Faktor für die Dominanz der relationalen DBMS ist ohne Zweifel die Entwicklung der Hardware. Ist ein DBMS einmal installiert und vernünftig dimensioniert, so wird es immer eine Zeit dauern, bis Engpässe praktisch sichtbar werden. Zu diesem Zeitpunkt stehen zuletzt aber immer schon leistungsfähigere Prozessoren, schnellere und größere Platten bereit, die das Problem wieder auffangen können. Dass die Fortschritte der Hardwaretechnik demnächst an eine Grenze stoßen, ist derzeit nicht absehbar.

2.2.5. Datenbanken für Appliances

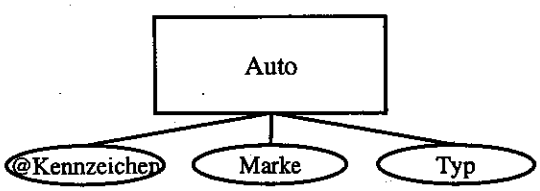
Der Anstoß zur „Revitalisierung" der hierarchischen DBMS - in einigen Bereichen waren sie ohnehin immer putzmunter - kommt daher auch aus einer ganz anderen Richtung. Die IT-Technik hat sich in letzter Zeit verstärkt ganz neue Einsatzgebiete erschlossen. Mikroprozessoren werden nicht nur immer leistungsfähiger, sondern auch immer kleiner und immer billiger. Sie finden sich daher nicht mehr nur in traditionellen Computersystemen, sondern in fast allen Geräten, ob im Haushalt wie Waschmaschinen, in Kleinstcomputern wie Handhelds oder Palmtops, in Telefonen, Spielautomaten, Kopiersystemen, PKWs, Uhren, Hifi-Anlagen und Waagen.

So klein und leistungsfähig die Prozessoren auch geworden sind, in solchen computerfernen Systemumgebungen herrscht absoluter Sparzwang. Der Arbeitsspeicher wird hier, wie in grauer IT Vorzeit, nach KB gezählt. Es gibt keine Festplatten, und es existieren weder ein Systemverantwortlicher noch ein Administrator.

Dennoch brauchen auch diese Systeme Datenbanken: Wenn sie nicht online sind, müssen sie Daten speichern, bis sie wieder ausgelesen werden können, und sie müssen Informationen für unterschiedliche Varianten der Programmausführung vorhalten. Für solche Aufgaben sind relationale DBMS meist nicht geeignet: Sie sind zu aufwendig und anspruchsvoll für diese Umgebungen. Andererseits wird die Flexibilität des relationalen Modells hier auch gar nicht benötigt, weil die Programmierer die Aufgaben der Geräte eng begrenzen können. Außerdem gibt es keine Anwender, die heute dies und morgen jenes von ihrer Datenbank wissen wollen.

2.3. Abbildung von ER-Modellen auf Relationen eines RDBMS

Umsetzung von Entitätstypen und Relationstypen (Beziehungen) in Tabellen.
Umsetzung der Entitätstypen:



=> Auto = @Kennzeichen + Marke + Typ + ...

Umsetzung der m:n- Beziehungen (anschaulich werden daraus 3 Tabellen !!):

- alle Attribute von R sind voll funktional abhängig von einem Primärschlüssel

Bemerkung:
- Alle Attribute innerhalb von geschweiften Klammern werden zu einer eigenständigen Relation und ein Attribut zum Primärschlüssel gemacht

-> Folie 3/4

3.2. Funktionale Abhängigkeiten

- b heißt funktional abhängig von a, falls für jedes Tupel x ∈ R gilt:
  ist der Wert a(x) von a für x bekannt, so ist dadurch der Wert b(x) von b für x eindeutig festgelegt.
- b heißt voll funktional abhängig von a, wenn b funktional abhängig von a ist, aber nicht von einem Teil von a (bei zusammengesetzten Attributen)

-> Folie 3/5

3.3. 2. Normalform

Aufbrechen der Entität in geeignete Teile und Verknüpfen dieser Teile über Beziehungen. Zusammengesetzte, mengenwertige Attribute werden zu eigenen Entitätstypen gemacht und über eine Beziehung mit dem gegebenen Entitätstyp verknüpft

- Eine Relation R heißt in 2NF, falls sie in 1NF ist und jedes Nicht-Schlüsselattribut voll funktional abhängig von allen Schlüsselkandidaten ist.
- Ein (zusammengesetztes) Attribut a heißt Nicht-Schlüsselattribut falls es nicht Teil eines Schlüssels ist

-> Folie 3/6

3.4. 3. Normalform

Eine Relation R ist in 3NF, falls sie in 2NF ist und kein Nicht-Schlüsselattribut funktional abhängig von einem anderen Nicht-Schlüsselattribut ist.

In der Praxis sollte bis zur 3NF normalisiert werden. Ausnahmen nur, wenn es zwingende Gründe gibt. Z.B. Effizienzprobleme beim Zugriff auf sehr große Datenmengen

-> Folie 3/7, 3/8, 3/9, 3/10

3.5. BCNF

Meist hat man bei Relationen in 3NF keine Speicher-Anomalie mehr. Besser ist die BCNF (Boyce-Codd-Normalform). Aus BCNF folgt 3NF.

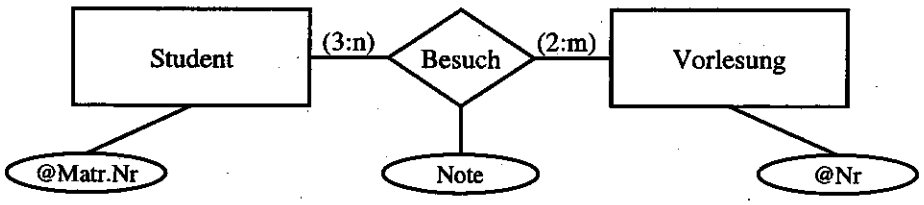
Bernstein-Algorithmus (für BCNF) leitet aus einer vorgeg. Menge von Attributen (betrachtet als universelle Relation) und einer vorgeg. Menge von funktionalen Abhängigkeiten entsprechende Relationen in BCNF ab.

3.5.1. Bernstein-Algorithmus

Def (Determinante):

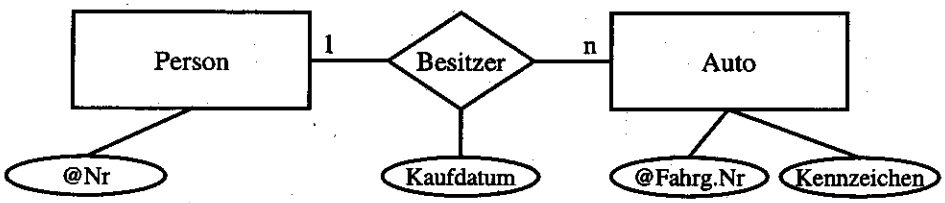
Sei a ein (evtl. zusammengesetztes) Attribut einer Relation R, dann heißt a Determinante in R, falls von a mindestens ein anderes Attribut voll funktional abhängig ist

Eine Relation R ist in BCNF, wenn jede Determinante in R ein Schlüsselkandidat ist.



- Besuch = @MatrNr + @<Vorlesung>Nr + Note
- Student = @MatrNr
- Vorlesung = @Nr

Umsetzung der 1:n - Beziehungen



1. Betrachte die 1:n- Beziehung als m:n- Beziehung (selten gewählte Möglichkeit):
   => Besitzer = @<Person>Nr + @Fahrgest.Nr + Kaufdatum
2. Beschreibe die Beziehung mit Hilfe eines Fremdschlüssels in der Zielentität (wo das n steht) - meist gewählte Möglichkeit:

   => Auto = @Fahrgest.Nr + Kennzeichen + <Person>Nr + Kaufdatum
   Fremd (<Person>Nr) Ref Person

   - <Person>Nr ist der Fremdschlüssel (muß nicht Primärschlüssel in Person sein, sondern drückt nur die Referenz aus; muß aber Schlüsselkandidat sein !)
   - Wenn <Person>Nr Primärschlüssel ist, darf er in der Spezifikation weggelassen werden

Vergleich der beiden Möglichkeiten:

- erst 2. Möglichkeit versuchen, dann nachdenken z.B. über eine ausreichende Zugriffsgeschwindigkeit
- z.B. Suchen eines Besitzers in der Tabelle Auto => sequentieller Durchlauf

3. Normalformen

Ziele:

- Vermeiden von Redundanzen im Datenbestand
- Vermeiden von Problemen beim Erzeugen, Löschen, Update von Daten.

Probleme:

- Einfüge-Anomalie
- Lösch-Anomalie
- Update-Anomalie

-> Folie 3/3

3.1. 1. Normalform

Eine Relation heißt normalisiert oder in 1. Normalform (1NF), wenn Sie keine Wiederholgruppen enthält.

- die Werte aller Attribute sind atomar (nicht weiter zerlegbar)

Schritte zur BCNF:

- Finde alle funktionalen Abhängigkeiten aller Attribute
- Mache aus den funktionalen Abhängigkeiten Relationen (ausgehend von den Determinanten)

Bemerkungen:

- Sei R eine Relation mit genau einem Schlüsselkandidaten, dann ist 3NF gleichwertig mit BCNF
- Eine Relation R ist nicht in BCNF, falls von irgendwelchen Attributen, die **allein** keine Schlüsselkandidaten sind, funktionale Abhängigkeiten ausgehen. In diesem Fall kann man ohne Informationsverlust in kleinere Relationen zerlegen.
- Es gilt: (5NF => 4NF =>) BCNF => 3NF => 2NF => 1NF
  Auch bei BCNF sind noch Speicheranomalien möglich (vgl. Stickel, Date für 4., 5.NF)

-> Folie 3/11

4. Datentypen

Bestimmen Sie auf der Basis der folgenden Überlegungen den geeigneten Datentyp für ein Feld (am Übungs-Beispiel Microsoft Access):

- Welcher Wertebereich soll im Feld zulässig sein? Sie können beispielsweise in einem Feld des Datentyps Zahl keinen Text speichern.
- Wieviel Speicherplatz möchten Sie für die Werte im Feld verwenden?
- Welche Operationen möchten Sie mit den Werten im Feld durchführen? Microsoft Access kann beispielsweise die Werte in Feldern des Datentyps Zahl oder Währung summieren, jedoch nicht die Werte in Feldern des Datentyps Text
- Möchten Sie ein Feld sortieren oder indizieren? Felder des Datentyps Memo, Hyperlink und OLE-Objekt können nicht sortiert oder indiziert werden.
- Möchten Sie ein Feld zum Gruppieren von Datensätzen in Abfragen oder Berichten verwenden? Felder des Datentyps Memo, Hyperlink und OLE-Objekt können nicht zum Gruppieren verwendet werden.
- Wie sollen die Werte in einem Feld sortiert werden? In einem Feld des Datentyps Text werden Zahlen als Zeichenfolgen (1, 10, 100, 2, 20, 200 usw.) und nicht als numerische Werte sortiert. Verwenden Sie deshalb ein Feld des Datentyps Zahl oder Währung, um Zahlen als numerische Werte zu sortieren. Außerdem werden viele Datumsformate nicht richtig sortiert, wenn diese in ein Feld des Datentyps Text eingegeben werden. Verwenden Sie ein Feld des Datentyps Datum/Uhrzeit, um die Werte richtig zu sortieren.

Die folgende Tabelle faßt die wichtigsten in Access verfügbaren Felddatentypen, ihren Zweck und den benötigten Speicherplatz zusammen:

Datentyp	Zweck	Größe
Text	Dieser Datentyp wird für Text oder Kombinationen aus Text und Zahlen verwendet, wie z.B. Adressen, aber auch für Zahlen, für die keine Berechnungen erforderlich sind, z.B. Telefonnummern, Artikelnummern oder Postleitzahlen.	Bis zu 255 Zeichen. Um die maximale Anzahl der zulässigen Zeichen festzulegen, stellen Sie die Eigenschaft Feldgröße ein.

Datentyp	Zweck	Größe
Zahl	Dieser Datentyp wird für numerische Daten verwendet, die für mathematische Berechnungen herangezogen werden. Eine Ausnahme bilden Berechnungen von Geldwerten (verwenden Sie hierfür den Datentyp Währung).	1, 2, 4 oder 8 Byte.
Währung	Dieser Datentyp wird für Währungswerte verwendet, um bei Berechnungen ein Abrunden zu verhindern. Der berechnete Wert ist auf bis zu 15 Stellen links und 4 Stellen rechts vom Dezimalkomma genau.	8 Byte.
Datum/Uhrzeit	Dieser Datentyp wird für Datums- und Zeitwerte verwendet.	8 Byte.
AutoWert	Dieser Datentyp wird für eindeutige fortlaufende Nummern (jeweils um den Wert 1 erhöht) verwendet, die beim Hinzufügen eines Datensatzes automatisch eingefügt werden.	4 Byte.
Ja/Nein	Dieser Datentyp wird für Felder verwendet, in denen nur einer von zwei Werten enthalten ist, wie z.B. Ja/Nein, Wahr/Falsch oder Ein/Aus.	1 Bit.
Memo	Dieser Datentyp wird für längere Texte und Zahlen, wie z.B. Notizen oder Beschreibungen, verwendet.	Bis zu 64.000 Zeichen.
OLE-Objekt	Dieser Datentyp wird für Objekte (z.B. Microsoft Word-Dokumente, Microsoft Excel-Kalkulationstabellen, Bilder, Klänge oder andere binäre Daten) verwendet, die unter Verwendung des OLE-Protokolls in anderen Programmen erstellt wurden und mit einer Microsoft Access-Tabelle verknüpft oder in eine Tabelle eingebettet werden können.	Bis zu 1 Gigabyte (durch Festplattenkapazität begrenzt).
Hyperlink	Dieser Datentyp wird für ein Feld zum Speichern von Hyperlinks verwendet. Ein Hyperlink kann ein UNC-Pfad oder ein URL sein.	Bis zu 64.000 Zeichen.

-> Übung: Umwandlung des ER-Modells in ein relationales Datenmodell