Heutige IT-Datenstrukturen

Aktuelle Zusammenstellung

28.Juli 2008

Autor:
Franz Plochberger
Freier Ingenieur
Huebwiesenstrasse 36/11
CH-8954 Geroldswil
0041(0)786 73 19 89

plbg@bluewin.ch
http://www.plbg.ch

1 Inhaltsverzeichnis

2 Abstrakt

Mit der Entstehung der Objekt-Orientierten (OO) Entwicklungsphilosophie und der daraus folgenden eigenständigen OO Applikationen ist eine Schnittestellen-Problematik mit vorhandenen Datenmengen in tradierten Strukturen wie Dateien, Hierarchischen und Relationalen Datenbanken entstanden.

Die etablierten IT-Zentralen großer Firmen sind aus personellen und kommerziellen Gründen nicht in der Lage, diese IT-Daten-Umwandlung mitzumachen. Es entstehen also parallele Bestände. IBM zeigt z.B. kein Interesse, Übergangssoftware für die tradierte Hostumgebung in Assembler, Cobol oder PI1 zu entwickeln.

Nur völlig neu entstehende Firmen können ihre IT OO gestalten.

Nebenbei bilden sich WWW – Datei – Standards im Internet, die wieder das Augenmerk auf den Inhalt von Texten, einfache Datenstrukturen und leicht zu erstellende Applikationen legen (HTML, XML).

Für große Datenmengen, wie sie in Industriezentren vorkommen, werden mathematisch durchkonstruierte Daten - Analysemethoden und Suchalgorithmen entwickelt (Datawarehouses, Data Mining).

Von USA schwappen seit etwa 5 Jahren Suchmaschinen für das gesamte Internet auf Europa. Derzeit wird deren Bedeutung für den E-Commerce echt erkannt (z.B. GOOGLE).

Internetportale mit gezielten Wissensforen (z.B. Wikipedia) boomen derzeit ebenfalls.

3 Gängige Datenstrukturen

Die derzeitige unüberschaubare Menge an Daten hat eine bereits unübersichtliche Menge verschiedenster Strukturierungen zur Folge. Diese Strukturen aus den Formalen Sprachen bleiben in der OO Denkweise erhalten. Es kommen nur neue hinzu.

3.1 Grundbegriffe

Um möglichst eindeutige Definitionen zu verwenden, will ich einige festhalten, wie ich sie mir geläufig sind. Wer genaueres darüber wissen will, möge in den einschlägigen Firmen-Publikationen nachschlagen. Heute kann man unter Verwendung dieser Wörter als Suchbegriffe im Internet bereits vielfältige und zum Teil ausführliche fachlich korrekte Informationen finden.

3.1.1 Die Menge (set)

ist eine Zusammenfassung von einzelnen Elementen. Eine Menge hat einen Namen, der dann für alle diese Elemente gilt.

Die Zuordnung zu dieser Menge kann entweder

- durch bloße Aufzählung oder
- durch Festlegen von bestimmten Eigenschaften der Elemente dieser Menge erfolgen.

Ein beliebiges Element kann also zu einer Menge gehören (= ein Element einer Menge sein) oder nicht.

3.1.2 Folge und n-Tupel

Rein mathematisch ist eine Folge eine bloße Aneinanderreihung von Elementen. Jedes Element hat eine bestimmte Position in dieser Folge, es gibt ein erstes und ein letztes Element und wahlweise auch eine Ordnung aller Elemente.

Ein n-Tupel ist eine Anordnung von jeweils n Elementen, zwischen denen zusätzlich eine oder mehrere „Relationen“ bestehen können.

3.1.3 Relation

Darunter verstehe ich eine definierte Beziehung zwischen zwei oder mehreren Elementen. Diese können Elemente der gleichen Menge oder auch verschiedener Mengen sein.

Das wesentliche ist die bestehende eindeutig festgelegte Beziehung.

3.1.4 Vektor

Ist eine Zusammenfassung von Komponenten. Das Wesentliche ist, dass ein Vektor immer mehrere Komponenten hat.

Durch Verwendung aller dieser und fester mathematischer Regeln der Vektorrechnung kann ein neues definiertes System (=Vektorraum) gebildet werden.

3.1.5 Automatendaten

Hier sind allgemein die Daten eines Datenautomaten, einer Datenmaschine gemeint. Diese werden als jene Daten separiert, die in „die verarbeitende Box“ = „das verarbeitende System“ eingegeben werden oder aus diesem ausgegeben werden.

Das Wort „Automat“ kommt aus dem Griechischen und kann sinngemäß als „sich selbst bewegendes Etwas“ übersetzt werden.

Michael Angelo verstand darunter ein „feinadriges Gebilde, das Tätigkeiten von Tieren und Menschen nachahmte“. Diese Definition hat heute mehr Gültigkeit denn je.

Hier konkret verstehe ich darunter, ein IT-System, das definierte Leistungen erbringt. Der Mensch kann die Leistungen dieser Box mit Hilfe definierter Steuerdaten variieren. Für ihn ist es ein Werkzeug oder Hilfsmittel, um eine einzelne vorbestimmte Aufgaben zu lösen (Buchhaltung, Kontoführung, Börsenhandel usw.).

In dieser Arbeit ist dieses System nicht der Kernpunkt, sondern nur die Daten, die es steuern oder die ein- und ausgegeben werden.

3.2 Gängige Datenstrukturen im Einzelnen

3.2.1 Das (Daten-) Feld (field)

Darunter verstehe ich einen **eindimensionalen Datenbereich eines einheitlichen Datentyps**.
Es gibt auch andere Festlegungen. Aus praktischer Erfahrung und pädagogischen Gründen will ich mich aber so festlegen.

Unter **Datentyp** werden bestimmte Zeichenmengen zusammengefasst.

„Typische" Beispiele dafür sind etwa:

- numerische Zeichen(numeric) für die Zahlen 0 bis 9,
- Buchstaben (character) für A bis Z und Sonderzeichen,
- Binäre Zeichen(binary) für rein binäre Darstellungen beliebiger Zeichen.

Einzelne Programmiersprachen haben genau festgelegte Regeln dafür (=Typdefinitionen).

Ein Feld hat einen Namen, den **Feldnamen** und hat eine bestimmte Länge (=Anzahl Zeichen) in Bits oder Bytes, die **Feldlänge.**

Bisher übliche Arten eines Feldes sind:

3.2.1.1 Ein Bit

Ist ursprünglich ein rein mathematischer Ausdruck, der eine Ziffer (=Stelle) im binären Zahlensystem darstellt.

Es ist die kleinste logische Einheit und die Basis für die gesamte zweiwertige (=binäre) Mathematik, die „BOOL'sche Algebra".

Konkret meine ich hier damit das Datenfeld, das diese binäre Zahl beinhaltet, also den Inhalt = Wert 0 oder 1 haben kann.

3.2.1.2 Ein Byte

Ist heute meistens die Zusammenfassung von 8 Bits.

Diese Anzahl 8 hat sich im Laufe der Zeit in der Praxis durchgesetzt und gilt heute allgemein.

Es kann in **zwei Halbbytes** von je 4 Bit Länge unterteilt werden, in Assembler spricht man vom Zonenteil und Ziffernteil.

Ein Byte gilt auch als Zähleinheit bei der Speicherung. Auf einem Speicherplatz von 1 Byte wird allgemein 1 Zeichen (char) gespeichert.

Es wird als Längenangabe (=Anzahl Zeichen) für größere Daten-Einheiten verwendet.

Es liegt aber kein dekadisches sondern ein binäres System zugrunde:

1 Byte 1 Stelle
1 kByte 1024 Stellen = 2 hoch 10
1 MByte 1024x1024 Stellen = 2 hoch 20
1 GByte 1024x1024x1024 Stellen = 2 hoch 30 ……

3.2.1.3 Das (Daten-) Wort (word)

Es ist meistens ein Feld aus 4 Bytes (= 32 Bit).

Dieses Wort besteht nun wieder aus zwei **Halbwörtern** von je 2 Byte (= 16 Bit) Länge.

Zwei Wörter werden zu einem **Doppelwort** von 8 Byte (=64 Bit) Länge zusammengefasst.

Es gibt auch noch andere Definitionen, die will ich aber aus pädagogischen Gründen hier nicht erwähnen.

3.2.1.4 Feld variabler Länge

Die Länge eines Datenfeldes muss nicht fix sein, wie oben. Theoretisch ist jede jeweils zweckmäßige Länge möglich.

Die notwendige Anpassung an Standardlängen (Byte, Halbwort, Wort) wird großteils automatisch durchgeführt. Man spricht dann von **Bytegrenzen**, **Wortgrenzen.** Weitere Themenbezeichnungen sind **Bündigkeit, Datenausrichtung oder Alignment.**

Felder variabler Länge haben am Beginn immer ein kennzeichnendes 2 oder 4 Byte langes Längenfeld, mit dem Betrag der Länge als Inhalt.

Die jeweilige Hardware ist allerdings auf Standardlängen fixiert. Es gibt folglich Halbwort-, Wort-, Doppelwort-Maschinen.

Der laufende Fortschritt macht manchmal auch schon noch größere Hardware-Längen möglich.

3.2.2 Die Menge (set)

Damit wird exakt der mathematische Grundbegriff von oben angewendet.

Entscheidend für die Menge sind deren Elemente. Eine Menge ist also ein logischer Sammelbegriff, eine logische Zusammenfassung mehrerer Elemente unter einem Namen.

Die einfachsten Form einer Menge entsteht durch **Aufzählung** gemäß seinem natürlichen Vorkommen, seinem physischen Auftreten**.**

Schon selektiver ist die Denkweise mit Elementen und jeweiligen Eigenschaften dieser. Alle Elemente einer Menge können auch eine kennzeichnende **gemeinsame Eigenschaften (=das Kriterium für die Zusammenfassung)** haben.

Eigenschaften eines Elementes ist **eine oder mehrere besondere Kennzeichnungen eines Elementes**.

Eine Aussage ist per definitionem ein Satz, von dem es sinnvoll ist zu behaupten, dass er wahr oder falsch ist.

Um beliebige Objekte zu Elementen einer Menge zusammenfassen zu können, muss ich also etwas über die Objekteigenschaft aussagen können.

Wenn also ein Objekt eine entscheidende Eigenschaft hat, ist sie ein Element einer Menge, sonst nicht.

3.2.2.1 Fuzzy - Menge (Fuzzy Set)

Wurde erstmals 1965 definiert von Lotfi A. Zadeh.

Er fordert, dass diese Menge neben den binären Zahlen 0 und 1 auch alle Zwischenwerte aus dem abgeschlossenem reellen Zahlenintervall 0,…,0.5555,…,0.999999, …,1 enthalten kann.

Damit wird die bekannte Fuzzy Logik begründet.

3.2.3 Feldstrukturen

3.2.3.1 Statische Feldstrukturen

3.2.3.1.1 Die Struktur schlechthin (structure)

Eine definierte Struktur von Feldern ist Voraussetzung für eine automatische Verarbeitung. Es wird in fast jeder Programmiersprache als Schlüsselwort verwendet (structure).

Eine statische Struktur ist dadurch gekennzeichnet, dass

- die Felder in Größe und Anordnung einmal festgelegt werden (Deklaration) und
- diese Struktur während der gesamten Bearbeitung gleich bleibt.

Diese Reihung der einzelnen Strukturelemente legt bei der Deklaration eine feste **Ordnung** unter den Feldern durch einfache Aufzählung fest. Alle Elemente werden zu einer Struktur unter **einem Namen**, dem Strukturnamen zusammengefasst.

Bei der Festlegung dieser Strukturen zeigt sich die Erfahrung und menschliche Intelligenz des Entwicklers (Datenmodellierung).

3.2.3.1.2 Der Datensatz (record)

ist nach seinem Aufbau eine typische und viel verwendete Datenstruktur.

Er hat eine besondere Verwendung in Zusammenhang mit Lesen aus Daten-Dateien oder Schreiben in solche.

Die Anzahl Bytes dieses Satzes ist fix oder variabel, sie ist die (maximale) **Satzlänge**.

Oft werden mehrere Sätze zu einem Lese- oder Schreib-**Block** zusammengefasst, die dann in **Dateien** gespeichert werden.

3.2.3.1.3 Die Tabelle (array)

Darunter versteht man ursprünglich eine Aneinanderreihung von Strukturen gleicher Bauart (Tabellensegmente).

Tabellensegmente können

- Felder, Strukturen (=eindimensionale Tabellen) oder
- wieder Tabellen sein (=mehrdimensionale Tabellen).

Um diese gleichartigen Elemente unterscheiden zu können wird eine laufende Nummerierung durchgeführt und diese als **Index** bezeichnet. Dieser wird dann als ein Teil des Namens den einzelnen Feldern hinzugefügt: Feldname(Index).

Weitere wichtige Größen einer Tabelle sind

- der **Maximale Index** für die größte Anzahl Elemente und
- die **Tabellengröße,** die die Anzahl Bytes einer Tabelle festlegt.

Man kann die gesamte Tabelle **sortieren**, was die Verarbeitung (suchen, löschen, einfügen) der einzelnen Tabellenelemente erleichtert. Eine kennzeichnende Größe dafür ist dann der **Sortierbegriff oder Suchbegriff**, der bestimmte Felder (einzelne oder mehrere) kennzeichnet.

Nach dem jeweiligen Inhalt dieser Begriffe wird die Tabelle dann in Programmschleifen bearbeitet.

3.2.3.1.4 Die Matrix (matrix)

Eine Matrix ist die ideale Struktur, um die **Beziehung zweier n-Tupel** übersichtlich und zergliedert darzustellen.

Dazu gleich ein Beispiel:

2 n-Tupel A und B lassen sich in die einzelnen Komponenten zerlegen A = (a1,a2,a3,....an) und B = (b1,b2,.....bm).

Die Matrix bildet sich wie folgt:

A	a1	a2	a3	....an	B
	r11	**r12**	**r13**	**....r1n**	b1
	r21	**r22**	**r23**	**....r2n**	b2
	.....				..
	rm1	**rm2**	**rm3**	**....rmn**	bm

Die einzelnen Elemente r11 bis rmn definieren die Beziehung in der **Matrix**.

Sie werden dabei übersichtlich in **Zeilen** und **Spalten** geordnet.

3.2.3.1.5 Der Würfel (cube) und komplexere Formen

Dieselbe Bildungsregel lässt sich konsequent auch **auf n n-Tupel erweitern.**

Es entsteht dann **der Würfel** für drei oder der **n-dimensionale Würfel** für n n-Tupel.

Das sich vorzustellen, wird natürlich ab 4 Vektoren schon schwierig. Man muss aber nur die Bildungsregeln konsequent erweitern.

3.2.3.2 Dynamische Feldstrukturen

Wenn die Anzahl der zu erfassenden Felder nicht bekannt ist, sehr wohl aber deren einzelne Typisierungen, wird eine dynamische (=bewegliche) Zuordnung unter einem Struktut-Namen definiert.

Dabei wird eine konstruktive Zwischenstufe verwendet, in der zusätzliche Datenfelder gefüllt werden, die nicht die eigentlichen Daten selbst, sondern reine **Strukturdaten oder Ordnungsdaten** enthalten.

Dafür sind die Bezeichnungen **Adressfelder oder Zeiger** gebräuchlich. Diese sind reserviert für die Hauptspeicheradressen der Feldstruktur.

In jüngster Zeit (ab der Programmiersprache JAVA) kommt man aber von diesen Zeigern, oder Adress-Feldern wieder ab, weil die Verwendung dieser Adressen zum Entwicklungszeitpunkt zu früh ist. Dies sollte erst durch die Kompilierung geschehen. Auch ergeben sich komplexe Zeiger-Korrelate, die zu einem späteren Wartungszeitpunkt sehr schwierig (wieder) zu verstehen sind.

Diese **Dynamisierung** will ich daher genauer untersuchen:
Was steckt dahinter? Warum entsteht dieser Aufwand? Dazu gehen wir ein wenig in die Programmiersprachen-Philosophie.

Ein Zeiger beinhaltet zuviel, er erfordert schon bei der Datendeklaration Aufgaben, die eigentlich erst in der letzten Stufe der Umwandlung eines Computer-Programms, bei der echten Verarbeitungsadresszuweisung erledigt werden sollten.

Das ist zum Zeitpunkt der Deklaration (=Strukturfestlegung) der Daten (-Klassen) viel zu früh. Zu diesem Zeitpunkt soll wirklich nur die Struktur und nicht auch der Hauptspeicheradressinhalt (= Ort im Hauptspeicher) zum Laufzeitpunkt angesprochen werden.

Erst auf der Assembler-Ebene sollte die Adressrechnung und Beachtung der Speicherung der Programmteile zum Laufzeitpunkt in den Sourcecode miteinfließen.

Der Umweg über die Speicheradressen der Daten zur Festlegung einer Beziehung, der sich ab der Sprache C, C++ etabliert hat, ist also nicht glücklich. Er wurde ab der Sprache JAVA wieder aufgegeben.

Ich will hier hinweisen auf meinen **Flachzeiger (flat pointer),** den ich rein theoretisch entwickelt habe, um schon ab sofort den bisherigen Zeiger nicht verwenden zu müssen.

Dazu siehe unter Bisherige Arbeiten, Einheitliche Datenstrukturen.

3.2.3.2.1 Gängige Arten Dynamischer Feldstrukturen

3.2.3.2.1.1 Der Stapel (stack)

Er ist den amerikanischen Großküchen entnommen, in denen die Esstabletts zur ebenen Entnahme in einem nach unten gefederten Stapel vorbereitet wurden und heute noch werden.

Er hat ein Beginnfeld und ein Endefeld analog dem Beginn-Tablett und dem Ende-Tablett. Die Dynamische Strukturierung ist aber insofern eingeschränkt, als nur am oberen Ende des Stapels sequentiell hinzugefügt und entnommen werden kann.

Diese Vorgänge sind sogar mit eigenen Namen versehen worden:

- **push** für oben hinzufügen, also wörtlich „von oben stoßen“ und
- **pop** für „auftauchen, (nach oben) knallen lassen“ beim Entnehmen.

Eine wichtige Größe dabei ist die maximale Anzahl der aufnehmbaren Felder, also die Stack-Tiefe oder Stack-Größe.

Die algorithmische Umsetzung in einem Programm kann auch mit meinem Flachzeiger geschehen.

Wenn alle Felder von gleichem Typ sind, ist die statische Struktur durchaus einer „auf den Kopf gestellten Tabelle “ vergleichbar, eine Tabelle also, bei der immer nur das höchste Tabellenelement hinzugefügt oder entfernt wird.

Im deutschen Sprachraum habe ich auch oft einen synonymen Ausdruck **„Keller - Speicher“** dafür gefunden.

3.2.3.2.1.2 Die Queue

Das ist ein „eingedeutschter Ausdruck“, Neudeutsch oder humorvoll „Deunglisch“.

Damit ist ein Stack oder eine Keller gemeint, der von beiden Seiten ge“pushed“ oder ge“popped“ werden kann.

Es ist also eine Segment-Folge, die am Beginn und am Ende ergänzt oder verkürzt werden kann.

Geringe Varianten sind oft auch vorhanden.

3.2.3.2.1.3 Die Kette (chain)

Die Verbindung e i n e s Feldes, einer Struktur oder eines Objektes mit e i n e m weiteren.

Dabei entsteht, wie der Name schon sagt, eine Kette mit den einzelnen „Gliedern“, die in e i n e r Reihe sich bilden. Eine Kette hat also ein Beginnfeld, beliebig viele „Glieder“-Felder und ein oder mehrere Endefelder.

Die einzelnen Elemente können durch die dynamische Strukturbildung (ev. gleich mit Flachzeigern) beliebig umgereiht werden.

Es können beliebige Elemente herausgelöscht werden oder neue eingefügt werden.

Die Kette als Gesamtheit bleibt aber nach dem jeweiligen Änderungsvorgang immer eine ununterbrochene Struktur.

3.2.3.2.2 Dynamische Feldstrukturen mit mehr als zwei Relationen (Verbindungen)

3.2.3.2.2.1 Der Haufen (heap)

ist durch seinen Namen schon fast erklärt.

Es können Einfach- oder Mehrfachverbindungen zwischen den verbundenen Feldern bestehen.

Mathematisch ist er ein Menge von Feldern m i t festgelegten Relationen der einzelnen Felder untereinander.

Typisch ist, dass diese Felder ganz beliebig miteinander verbunden werden können. Es gibt sonst keine durchgehende Ordnung , außer der, dass eine Verbindung besteht oder nicht.

Er hat also ein Beginnfeld, viele mögliche Verzweigungen und viele Endefelder.

Um meinen Flachzeiger anwenden zu können, müsste dieser vom Dupel auf ein n-Tupel erweitert werden, sonst ändert sich nichts.

3.2.3.2.2.2 Der Graph

Er kann wie ein Haufen gesehen werden, der aber auch **eine Ordnung der einzelnen Felder untereinander** festgelegt hat.

Besondere Kennzeichen eines Graphen sind ein Beginnfeld, ein oder mehrere Endefelder, detaillierte Mehrfachverbindungen mit festgelegter Flussrichtung.

Diese Mehrfachverbindungen werden neben der reinen Zuordnung mit weiteren Parametern ausgestattet, wie mengenmäßige **Über- oder Unterordnung** oder **gegenseitige Beeinflussung oder Steuerung**.

Ein Graph besteht aus Knoten (einzelne Elemente) und Kanten (Verbindungslinien).

Ein besonderes Augenmerk wird dabei auf seine optische und graphische Darstellbarkeit gelegt, daher auch der Name.

Er ist das heute am meisten verwendetet Konstruktionselement jedes Informationsmanagers.

Seine Komplexität kann von einer Handskizze bis zu einem - mittels eines Software-Tools erstellten - umfangreichen Diagramms gehen.

3.2.3.2.2.3 Das Netzwerk (net)

ist eine Vermischung von Haufen und Graph.

Ein natürliches Muster dafür wäre das Netz einer Spinne.

Dem Graphen (=Schreiber) entspricht das Spinnen und Fortbewegen der natürlichen Spinne. Die öfter vorkommenden Endpunkte sind die Befestigungspunkte – wie im Spinnen-Netz. Die Verknüpfungspunkte (Knoten) sind die einzelnen Datenelemente.

Die rein technische Realisierung wird allerdings selten Rücksicht nehmen auf eine rein physische geometrische Form des Netzwerkes sondern nur auf das **Herstellen einer Verbindung selektiver Punkte und die Findung der kürzesten Verbindung zu ihnen.**

Man spricht auch konkret von **Knoten** dieses Netzes. Die Verbindungen werden ebenfalls als **Kanten** bezeichnet.

Ein sehr aktuelles modernes Beispiel dafür ist eben das Internet. Die Verbindungspunkte dieses Netzes sind die einzelnen Netzwerkserver und die Verbindungen sind die echt physischen Leitungen oder neuerdings die „Funkverbindungen".

Das Netz hat deswegen eine wichtige bleibende Bedeutung, weil die Hirnforschung festgestellt hat, dass das menschliche Gehirn ebenfalls ein räumliches Netzwerk von Neuronen-Knoten und Nerven(-Kanten) ist. Daher bleibt die Netz-Struktur ein wichtiges dynamisches Strukturelement der Informatik-Gegenwart.

3.2.3.2.2.4 Der Baum (tree)

hat ein streng hierarchisches Prinzip.

Beginnend vom ersten Feld (**root**) wird Stufe für Stufe durch jeweilige „Verzweigung " – wie bei einem auf den Kopf gestellten echten Baum - ein immer vielfältigeres Gebilde geschaffen.

Wichtig dabei ist, dass immer nur von der Root weg nach unten verzweigt werden kann, eine Mehrfachverbindung im Sinne einer „Vernetzung innerhalb der Baumäste" wird in der Verarbeitung bewusst ausgeschlossen. Verzweigungshierachien (Astgabeln) haben dabei eine wichtige Rolle.

Es haben sich für die Beziehung der hierarchischen Ebenen auch die Fachausdrücke **Parent** und **Child** bzw **Twin** für gleiche Felder auf gleicher Ebene durchgesetzt.

Wenn immer nur e i n e Gabelung in jeder Stufe vorgesehen wird, also ein Parent immer nur zwei Childs hat spricht man auch von einem „**binären Baum** ".

3.3 Probleme aus der bisherigen Feldstrukturbildung

3.3.1 Kardinalität bei mehreren Feldern der gleichen Relations-Ebene

ist eine rein mathematische aber sehr nützliche Darstellungsform mit Hilfe von Zahlen und den Zeichen „(„ „„“, „:“ und „)“.

Dabei wird festgelegt, wie viele Felder mit wie vielen Feldern in Beziehung stehen.

Dabei markant ist, dass auch die Null mitgedacht werden kann, also das eventuelle **Nicht-Auftreten** eines Feldes.

Typische Fälle sind:

3.3.1.1 (0,1:0,1)

Kein oder 1 Feld steht in Beziehung zu einem anderen Feld.

3.3.1.2 (1:1)

sehr häufige, weil eindeutige „eins zu eins“ Beziehung

3.3.1.3 (1:n) oder (n:1)

ist selbstsprechend

3.3.1.4 (m:n)

ist ebenfalls sehr häufig.

Dazu will ich anmerken, dass diese Beziehung vor der Umsetzung in eine reelle Strukturform möglichst in mehrere 1:n- Formen umgewandelt werden soll, weil sie sonst sehr aufwendig wird.

Wenn dies nicht möglich ist, schlage ich vor **eine Relationsebene mit einer Relationsmatrix** zu definieren, in der jede einzelne Relation festgelegt ist.

3.4 Das „Routing “- Problem allgemein

Es ist derzeit in der Hardware sehr dominant. Die Geräte, die es lösen, heißen „Router“.

Die Aufgaben dieser Geräte sind:

- Herausfinden der optimalen Verbindung vom Sender zum Empfänger einer Nachricht.
- Übertragen einer Nachricht.
- Überwachen der Vollzugsmeldung

Zum Strukturproblem, das die **algorithmische Verarbeitbarkeit** mittels Software zum Ziel hat, kommt dabei noch **die Zeit,** in der dies geschehen soll, und die **Nachvollziehbarkeit** im Falle einer Unterbrechung einer Verbindung (=Fehlerfall).

4 Strukturen in mehreren Ebenen

Alle oben angeführten Strukturen lassen sich nach dem „Layer"- oder Schichtenprinzip stufenweise auf tiefere Ebenen verfeinern. Traditionell spricht man auch von Hierarchien.

Ein weiteres Synonym ist auch die Bezeichnung „**mehrdimensional**" in mehrdimensionalen Tabellen oder mehrdimensionalen Strukturen. Damit ist eine in mehreren Ebenen verschachtelte Struktur gleicher Unterstrukturen gemeint**.**

Der Algorithmus der dimensionalen Erweiterung ist gedanklich einfach:

Jedes Feld einer Struktur kann als Name einer weiteren hierarchisch tieferen Struktur betrachtet werden.

Die höchste Ebene ist also eine Struktur beliebiger Form wie bisher. Jedes Feld (Element) dieser obersten Struktur wird als Textfeld mit dem Namen der nächsten Ebene einer beliebigen tieferen Struktur gesehen. Die Form der jeweiligen Struktur wird jeweils nach Bedarf festgelegt. So kann eine Einheit vielfältigster Form entstehen.

Markant dabei ist, dass Strukturen auf mehreren Ebenen erst auf der letzten (=untersten, tiefsten) Stufe die echten Daten beinhalten können. Alle Stufen davor sind dann vor allem Textfelder mit Namen, Textfelder mit eventuellen Strukturparamtern.

In der Denkweise der Flachzeiger würde ein solcher Strukturparameter dann eben aus einem Feldnamen und einem Flachzeigernamen bestehen, der in diesem Fall nur den Namen des folgenden Feldes eine Stufe tiefer beinhalten müsste. Die Strukturform auf mehreren Ebenen wäre damit sehr einfach dynamisierbar.

Ein praktisches Anwendungsbeispiel für diese Strukturform wäre auch **die Liste bzw. die Stückliste** in der Leiterplattentechnik der Hardwareentwicklung.

5 Strukturumwandlungen von Dateien und Datenbanken

Aus gegebenem Anlass versuche ich ein zunächst eher abstraktes Konzept zu entwickeln, nach dem man Datenstrukturen der Formen **Datei und Hierarchische, Relationale oder Objektorientierte Datenbanken** ineinander überführen kann.

5.1 Die Datei

Hat sich historisch aus den **Lochkartenstapeln** entwickelt. Mit Erfindung der bi-stabilen Speichermedien in den 70er Jahren des 20.Jhdts. konnten die Daten, die am Beginn des 20.JHdts in Lochkarten gestanzt waren, Zeichen für Zeichen auf einen elektronischen Datenträger (Platte, Band) gespeichert werden.

Die wichtigste Unterstruktur einer Lochkarte war **die Zeile**. Die Länge dieser Zeile war fixiert auf 80 Zeichen oder Byte. Pro Karte wurden 9 Zeilen gespeichert. Die Zeile wurde eine Daten-Struktur und als solche **Satz (record)** genannt (siehe oben).

Mit der neuen Datenspeicherform „Datei" kam man von der zeilenweisen Speicherung zu einer mengenmäßigen Speicherung. Es wurden so viele Zeilen wie möglich hintereinander gereiht, der Länge auch variabel gestaltet und die Gesamt-Speichergröße eines Datenträgers als Kenngröße fixiert.

Dateien, in denen hintereinander geschriebener Text steht nennt man **Textdateien**. In diesen kann natürlich ein Satz über mehrere **Druckzeilen** gehen, wenn die Papierbreite des Druckpapiers nicht mit der Textdateizeilenlänge übereinstimmt.

Die Formen **Satz - Textzeile - Druckzeile** können also längenmäßig differieren.

Die Satzlänge, Satzanzahl, Dateigröße und Dateitype sind fixe Strukturparameter jeder Datei.

Eine Datei lässt sich als eine Zusammenfassung mehrerer Sätze gleicher Strukturform kennzeichnen.

Die Organisationsform einer Datei und Zugriffsform auf eine Datei sind wichtige Kenngrößen für einen Dateityp.

Sie legen fest, ob auf einen Satz sequentiell oder direkt zugegriffen werden kann. Bei direktem Zugriff ist ein separat geführter Index (Adresstabelle) notwenig.

5.2 Die hierarchische Datenbank

Eine **Datenbank** im Allgemeinen ist strukturell feiner gegliedert als eine Datei.

Wenn z.B. nur ein einzelnes Feld in einem Satz geändert werden muss, ist zunächst der einzelne Satz zu finden.

Bei einer Bandspeicherung war jedes Mal ein aufwendiger Suchvorgang notwendig.

Nach dem Finden des gesuchten Satzes wird der ganze Satz gelesen, das oder die Felder geändert und der ganze Satz wieder zurück geschrieben.

Man begann daher bald die starre Satzstruktur in kleinere Segmente (=**Datenbanksegmente**) zu zerlegen. Um einen gezielten Zugang zu diesen zu bekommen, setzte sich eine **hierarchische Anordnung** dieser einzelnen Segmente durch.

Die Sätze mit nur kleinen Unterschieden werden dabei rationeller gespeichert. Gleichbleibende Daten eines Satzes werden nur einmal gespeichert. Nur die Segmente mit verschiedenem Inhalt werden mehrfach gespeichert.

Ein so genannter **Datenbanksatz** ergibt sich in der Form einer Segment-Hierarchie mit Root, Child = Parent, ev. Twin, Child = Parent usw.

Ein DB-Satz kann also noch mehrere Hierarchiestufen haben. Die einzelnen Segmente haben die Kardinalität (1:n), dh**.** die Anzahl der Segmente, die zu einem DB-Satz gehören, kann variieren.

Das kennzeichnende (=oberste) Segment heißt „ROOT"-Segment. Die kennzeichnenden Felder (=Schlüsselfelder, Keys) dieses Segmentes kennzeichnen den gesamten Datenbanksatz und sind einmalig.

Entsprechend dem Hierarchischen Pfad kann eine vorgegebene Strukturform dieser Satzstruktur festgelegt werden. Die Bildungsregel lautet: Segmente von oben nach unten (bei Childs) und von vorn nach hinten (bei Twins).

Auf diese Weise können alle Segmente, die zu einem Root gehören hintereinander gereiht werden. Die Hierarchie der Bedeutung kann sich direkt in der Datenstruktur abbilden.

5.3 Die relationale Datenbank

geht von der hierarchischen Strukturform ab.

Eine bloße Aneinanderreihung der einzelnen Felder zu einfachen Satzstrukturen (=Relations-Tupeln, **Rows**) reicht. Diese Tupeln werden themenmässig als Tabellenelemente zu einer **Tabelle (=Relation)** zusammengefasst. Mehrere thematisch zusammenhängende Tabellen können miteinander verknüpft werden.

Das wesentliche ist nicht mehr die Hierarchie der einzelnen Segmente und Felder, sondern die nach den mathematisch Regeln der Mengenlehre gebildeten Gruppierungen dieser Felder und deren **Relationen**.

Die starre Hierarchie wird also durch flache logische Bildungsregeln ersetzt. Zum Auffinden der einzelnen Feldinhalte werden Schlüsselfelder nach der BOOL'schen Algebra verknüpft.

Für den Entwurf dieser Tabellenstrukturen und der einzelnen Felder werden eigene

Regeln vorgegebenen, die

5.3.1 Normalformen nach CODD.

1. Normalform: Eine Relation = Tabelle darf nicht in einer anderen Relation vorkommen.

Eine flache Hierarchie wird angestrebt oder die Schachtelungstiefe wird durch mengenmäßig breite aber logisch scharfe Verbindungen ersetzt.

2. Normalform: Ist eine Verschärfung der 1. Die 1.Normalform gilt weiter, erhält aber den Zusatz:

Jede Zeile (Tupel, row) einer Tabelle hat einen Schlüssel (=Keyfeld), der zu jedem Feld der Tupelstruktur in Beziehung steht. Er ist „zeilen-, row- und tupel-identifizierend".

Dieser Schlüssel (Primärschlüssel) soll also ermöglichen, dass jedes Tupel direkt selektiv aufgefunden werden kann.

3.Normalform: 1. und 2. gelten wieder weiter. Zusätzlich wird verlangt:

Nicht-Schlüssel- Attribute eines Tupels sind wechselseitig unabhängig (disjunkt). Es besteht keine hierarchische Abhängigkeit der einzelnen Felder.

Bestimmte Felder können Schlüsselfelder zu weiteren Tabellen sein (Fremdschlüssel).
Das ist die klare Aufforderung eine flache, aber breite und logisch sauberere Strukturbeziehung Tabelle→ Tupel→Feld zu bilden.

Die inhaltliche und verarbeitungsorientierte Speicherung kann also hiermit logisch differenziert nachvollzogen werden, das ist der Fortschritt.

Es muss erwähnt werden, dass die reine 3.Normalform in der Praxis selten erreicht werden kann.

5.3.2 Schlüsselfelder (keys)

Eine besondere Bedeutung haben die **Schlüsselfelder (Keys)**. Dies sind besondere Datenfelder, deren Inhalt eine ganzes Tupel repräsentieren. Die Auswahl dieses Schlüssel-Feldes ist also für die Effizienz einer relationalen Datenbanktabelle sehr entscheidend.

Zu jeder Relation = Tabelle gehört auch **der Index dieser Tabelle.** Dies ist eine Tabelle, die die zwei Felder **Tupel-Nummer** eines Tupels in der Relationentabelle und **Schlüssel-Feld** des tatsächlich vorhandenen Tupels mit einander in Beziehung bringt.

Es gibt Primär- und Sekundär- oder Fremdschlüssel und die jeweiligen Indextabellen dazu.

Die **Primärschlüssel** ermöglichen mit Hilfe des Primärindex den gezielten Zugriff auf ein bestimmtes Tupel.

Der **Sekundärschlüssel** ermöglicht eine zweite, variierende Zugriffsmöglichkeit mit Hilfe des Sekundärindex.

Der **Fremdschlüssel** ist ein Feld innerhalb des Tupels, das auf ein Tupel einer fremden Tabelle verweist.

Die Umwandlung einer Relation in einen hierarchischen DB-Satz und in einen Satz einer Datei und umgekehrt ist auf der Strukturstufe Tupel(Zeile, Row) der Relationalen Datenbank, Satz und Segment einer Hierarchischen Datenbank und Satz einer Datei eindeutig durchführbar.

5.3.3 Kriterien für die Umwandlung Datei, Hierarchische DB → Relationale DB

Das **Problem der Mehrfachspeicherung** von sich nicht oft verändernden Datenfelder der jeweiligen Strukturen ist sowohl bei einer Datei als auch einer Relationaler Datenbank vorhanden. Es gibt dann immer viele Felder, die keinen „Neuwert" haben, weil sie sich nicht ändern. Sie werden aber mitgepflegt.

Die **Segment-Hierarchie** Hierarchischer Datenbanken wird in Relationalen DBen bewusst abgebaut. Sie wird durch logische Zusammenhänge (Relationen) vieler Daten ersetzt.

Die **Mengenbildung der Relationalen DB-Felder** erfordert eine **selektive Umreihung der einzelnen Felder.** Diese legt die neuen logischen Relationen im Sinne von Beziehungen fest. Erst darnach kann eine eventuelle Umspeicherung von einer Datei oder Hierarchischen Datenbank auf eine Relationentabelle vorgenommen werden.

Jede einzelne Umwandlung verlangt einen speziellen **Umwandlungsalgorithmus**. Es ist also notwendig, dass ein Mensch diesen Algorithmus von Fall zu Fall **neu entwickelt**. Von allgemein gültigen Bildungsregeln kann hier nur Gebrauch gemacht werden, wenn immer wiederkehrende Strukturregeln feststellbar sind.

Die Datei wird durch **Sortieren nach selektiven Sortier-Feldern** leichter in eine Relationale DB umwandelbar sein.

Bei der Hierarchischen DB sind diese bereits durchgeführt. Es kann gleich der **Algorithmus der Umreihung der einzelnen Felder** entwickelt werden.

5.4 Objektorientierte Datenbanken

Es gilt in der Fachwelt eine Maxime, die scheint von den Pfadfindern übernommen worden zu sein: „Einmal objektorientiert - immer objektorientiert".

Die OO ist zu einem dominierenden Paradigma der gegenwärtigen IT geworden. Die gezielte Zusammenfassung von Daten und Funktionen oder Routinen zu einer Einheit fördert die prinzipielle logische Erfassbarkeit von Software. Unter einem Objektnamen werden alle dazugehörigen Aktivitäten und Daten zusammengefasst.

Der Typ jedes Objektes wird durch seine Klasse deklariert. Einzelne Klassen können ihre Elemente anderen Klassen vererben – es bildet sich also wieder eine Art Hierarchie, aber aus allgemeineren Gründen als in der reinen Datenhierarchie.

Die einzelnen Klassen sind also das neue Kern-Element. Die Zusammenhänge (Vererbungen) werden in eigenen Diagrammen festgelegt. Jede Klasse besteht aus einem reinen Datenteil und reinen Software-Routinen für einzelne Aufgaben innerhalb einer Klasse. Dadurch entsteht eine thematische Überschaubarkeit, die bisher nicht möglich war.

Zu den wichtigsten Objektorientierten Prinzipien:

Die Grundidee ist, gebrauchte Daten und Software, die diese bearbeiten (=Funktionen, Routinen, Methoden) in eine **„Kapsel" (= Objekt)** zu verschließen. Damit soll das Vorhandensein eines Objektes (einer Klasse) sagen, dass man damit alles machen kann, was zu Lösung der gestellten Aufgabe notwendig ist.

Der Zugriff auf Objekte geschieht mit **„Botschaften"(messages).** Jede Botschaft wird mit einer **„Antwort"** reflektiert.

Der innere Aufbau eines Objektes ist bei der Verarbeitung nicht mehr interessant (=Prinzip der Verkapselung, encapsulation), es wird nur noch auf die Durchführung und die Ergebnisse geachtet.

Objekte gleichen Typs haben dieselbe **Klasse.** Der Typ eines Objektes ist also seine Klasse**.** Diese wird bei der Entwicklung eines Objektes erstellt, ist also der eigentliche Kern. Mit Namen und ID versehen wird daraus ein wirksames (instanziiertes) Objekt.

Alle Klassen werden in einer Klassenbibliothek gespeichert. Diese ist meist relational, reine Objektspeicherformen haben sich nicht durchgestzt.

Für unsere Zwecke ist also diese Klasse ein wichtiges Element. Wir müssen ihre Struktur genau kennen.

Jetzt kommen wir in Konflikt mit dem Verkapselungsprinzip. Das hat ja die Absicht, nicht in das Objekt hineinschauen zu müssen.

Es soll eigentlich eine Schnittstelle (der Aufruf) reichen. Für den effizienten Einsatz unseres Objektes brauchen wir aber unabdingbar **eine detaillierte Beschreibung der Daten und Funktionen unseres Objektes d.h. eine vollständige „Bedienungsanleitung".**

Wir kommen also nicht umhin, diese Klassen, die aus Strukturen und Funktionen bestehen genauer zu erlernen, um ein System zu verstehen.

Die Branche hat sich ein Umwandlungsproblem - formal auf objektorientiert - leicht gemacht. Sie hat sie einfach ausgeschlossen und eine Kompatibilität aus allen Denkprinzipien gestrichen.

Das hat anfangs eine klare Entwicklungslinie vorgegeben, aber umgekehrt auch eine reine generelle Revolution ja sogar Löschung aller etablierten formalen Softwaresysteme gefordert.

Heute, etwa 10 Jahre nach Einführung des OO-Paradigmas existieren beide Systeme nebeneinander. Die tradierten formalen Systeme haben sich behauptet, weil sie funktionierten und tägliche Aufgaben zuverlässlich erfüllten. Warum sollte oder soll man sie löschen? Neuentwicklungen wurden aber generell immer OO durchgeführt, aber die tradierten formalen Systeme liefen und laufen weiter.

Festhalten will ich:

Dieser Bruch in der Softwareentwicklung verursacht derzeit einen immensen Aufwand und hat viele Jobs und Karrieren gekostet. Die einzige Lehre, die wir daraus ziehen können, ist solche Brüche zu vermeiden. Nachfolgende neue Prinzipien gleicher Größenordnung, wie etwa die von mir persönlich propagierte Subjektorientierung (SO) oder Humanorientierung (HO) darf also keinen Bruch mehr machen, sondern muss aus beiden Systemen (formal und OO) kontinuierlich weiterentwickelt werden können.

5.5 Zusammenfassung Strukturumwandlungen

⇒ Die große Softwarekrise der 70er und 80er-Jahre des 20.Jhdts ist heute **theoretisch gelöst**.
⇒ Bisherige IT-System laufen in der formalen Denkweise weiter.
⇒ Eine **Umwandlung von formal auf OO wird praktisch nie durchgeführt**.
⇒ Neue Systeme werden generell alle OO gestaltet.
⇒ Bei neuen Paradigmen (zB. HO) sollte diese Inkompatibilität vermieden werden.