Einführung und Kommentar zu

E. F. Codd: A Relational Model of Data for Large Shared Data Banks (1970)

von Holger Maaß, maass@maasster.de, 17.03.2011

Vorbemerkung

Datenbanken sind heute omnipräsent. Ein großer Teil davon - wahrscheinlich sogar der Hauptteil - sind relationale Datenbanken. Diese haben sich für verschiedenste Zwecke als besonders geeignet erwiesen. Für die Anordnung der Daten folgen solche Systeme einem ganz bestimmten Modell, das als relationales Modell bezeichnet wird.

Wer nun mehr über das relationale Modell wissen möchte, wird schnell auf einen Aufsatz des britischen Mathematikers und Informatikers E. F. Codd verwiesen, der als Gründungsdokument für dieses Datenbankmodell gilt:

> A Relational Model of Data for Large Shared Data Banks. Communications of the ACM Volume 13, Number 6 (June 1970), 377-387.

Dieser Aufsatz ist zu Recht ein Klassiker, und die Lektüre lohnt sich nach wie vor, wenn man Datenbanken nicht nur nutzen, sondern auch etwas tiefer verstehen will. Er ist nicht besonders lang (nur 10 Seiten) und frei im Internet zugänglich, so dass sich jeder Interessent diesem Text einfach so zuwenden kann. Der Link zum Text ist folgender:

http://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

Wer diesen Text das erste Mal zur Hand nimmt, merkt jedoch schnell, dass sich sein Inhalt nicht einfach so erschließt. Sicher gibt es andere Texte von Codd, die wesentlich schwieriger zu lesen sind, aber auch dieser hat es in sich. Wir wollen deshalb hier diesen Text Schritt für Schritt durchgehen, erläutern und kommentieren, um dem interessierten Leser einen leichteren Zugang zu ermöglichen.

Aufbau des Textes

Codds Aufsatz beginnt mit einem kurzen Abstract, wo der Inhalt in knapper Form vorweggenommen wird, und gliedert sich danach in zwei Hauptabschnitte auf. Der Text

enthält - wohl aufgrund seiner Kürze - kein explizites Inhaltsverzeichnis. Es gibt aber eine Unterteilung mit Überschriften, und das ist folgende:

Der *erste Hauptabschnitt* führt das relationale Modell ein und grenzt es gegen damals bereits in der Praxis existierende Datenmodelle ab. Darüber hinaus wird der Begriff der Normalform erläutert, die aufzeigt, wie eine Relation im Sinne des neuen Datenmodells im Normalfall, d.h. im Standardfall, aussehen sollte. Und drittens geht Codd im ersten Abschnitt auf die Frage der Sprache ein, in welcher eine Kommunikation mit der Datenbank erfolgen kann, z.B. um Informationen zu extrahieren oder Daten zu ändern.

Der *zweite Hauptabschnitt* erläutert zunächst das System der Operationen im relationalen Modell. Dies sind vor allem Mengenoperationen, weil Relationen - wie wir noch sehen werden - Mengen sind. Außerdem werden in diesem Abschnitt die Probleme der Redundanz

(mehrfaches Vorkommen derselben Information) und Konsistenz (Widerspruchsfreiheit) der Daten diskutiert, die miteinander in Zusammenhang stehen.

Was in den beiden Hauptabschnitten entwickelt wird, deutet in dieser Weise auch das vorgreifende Abstract des Textes an. Das Abstract führt aber noch eine wesentliche Unterscheidung ein, die wir nicht außer Acht lassen dürfen, nämlich die zwischen der *internen* Repräsentation der Datenstruktur durch das technische System (die Maschine) und der *externen* Repräsentation, wie sie dem Benutzer der Datenbank bzw. dem Anwendungsprogrammierer zugänglich gemacht wird. Der erste Satz des Abstracts und damit des gesamten Aufsatzes bringt die wesentliche Motivation dieser Unterscheidung schon deutlich zum Ausdruck:

> Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). (377)

Codd sieht klar, dass der Umgang mit großen Datenmengen in Datenbanken für Benutzer und Anwendungsprogrammierer nur dann noch effektiv bewältigt werden kann, wenn sie sich nicht um die technische Realisierung des Datensystems - z.B. die Speicherung im Dateisystem - kümmern müssen und die interne Repräsentation der Daten komplett vor ihnen verborgen wird. Nur die logische und inhaltliche Struktur der Daten soll dem Benutzer zugänglich sein, und nur auf dieser Ebene soll der Benutzer die Daten abfragen und verändern.

Es ist genau diese letzte Ebene, für die Codd das relationale Modell einführt, und deshalb ist sein hier besprochener Text auch heute noch vor allem für versierte Nutzer von Datenbanken interessant und vielleicht weniger für technische Entwickler von Datenbanksystemen. Dass das relationale Modell die Sicht der Nutzer auf die Daten beschreibt und regelt und damit die logische Struktur, in der sich der Nutzer denkend bewegt, ist bis heute eine wenig verbreitete Einsicht, wie mir scheint. Viele verorten nämlich die relationalen Strukturen noch auf der Ebene der technischen Realisierung des Datenbanksystems und sehen erst in einer soweit vereinfachten Benutzeroberfläche ihren Einstiegspunkt, dass kaum noch etwas von eigentlich im Modell vorgesehenen Kommunikationsmöglichkeiten übrig bleibt.

1. Relational Model and Normal Form

1.1. Introduction

Codd stellt zunächst heraus, dass er mit seinem Text das Ziel verfolgt, die elementare Theorie der Relationen (aus der Mathematik) auf den Bereich der strukturierten Datenbanken anzuwenden. Als direkte Vorläufer nennt er hier D.L. Childs mit einem 1968 erschienenen Aufsatz[1] sowie die Forschungen zu deduktiven Frage-Antwort-Systemen mit Verweis auf eine Schrift von Levien und Maron aus dem Jahr 1967.[2] Der Autor verschweigt also keineswegs, dass er an die Arbeiten von anderen anknüpft und seine eigenen Ideen nicht einfach vom Himmel fallen, und es wäre natürlich interessant, genauer zu verfolgen, wie er an seine Vorgänger einerseits anknüpft und andererseits über sie hinausgeht. Das würde aber unser Anliegen eines Kommentars überfordern und hier zu weit führen.

Nur soviel möchte ich an dieser Stelle anmerken, dass offenbar die Forschungen zu deduktiven Systemen, welche automatisch logische Folgerungen ziehen können (Deduktionen), ein entscheidender Ideengeber für das relationale Datenbankmodell gewesen sind. Es mag dabei vielleicht paradox erscheinen, dass Codd gerade die Idee eines automatisierten logischen Schließens nicht mit in sein Datenbankmodell übernimmt und damit das Hauptziel der deduktiven Systeme beiseite lässt. Die dort eher als Hilfsmodelle bemühten mengen- und logikorientierten Datenstrukturen werden dagegen in Codds relationalem Modell umso zentraler, wie wir noch sehen werden.

Der Autor betont hier noch einmal, dass es ihm in diesem Text um zwei Grundprobleme geht. Zum einen die Datenunabhängigkeit (*data independence*), wobei es um die Unabhängigkeit der logischen Datenstruktur von der maschinellen Repräsentation geht, und zum zweiten die Dateninkonsistenz (*data inconsistency*) als potentiell gravierendes Problem aller Datensysteme.
Für das eingeführte relationale Modell werden drei Aspekte genannt, worin es den bislang dominierenden graphen- oder netzwerkorientierten Modellen überlegen ist:

1. Die Daten können in ihrer natürlichen Struktur beschrieben werden („with its natural structure only“).

2. Das Modell bietet eine gesicherte Basis, um Ableitbarkeit, Redundanz und Konsistenz von Daten zu behandeln.

[1] Vgl. Childs 1968.
[2] Vgl. Levien & Maron 1967. Noch ausführlicher sind die Ideen dargestellt in Levien & Maron 1965.

3. Die relationale Sicht erlaubt eine klarere Einschätzung der logischen Möglichkeiten und Grenzen der bisherigen Systeme.

1.2. Data Dependencies in Present Systems

Codd gesteht den bisherigen Systemen durchaus Teilerfolge auf dem Weg zu einer logischen Datenstruktur zu, die weitgehend unabhängig von maschineller Repräsentation ist. Ein Maß an Unabhängigkeit wie es im relationalen Modell anvisiert wird, sieht er aber noch nicht erreicht. Vor allem drei Arten der Datenabhängigkeit sollen zurückgedrängt werden:

- bei der Sortierung bzw. Anordnung der Daten als Folge,
- bei der Indizierung und
- bei den Zugriffspfaden.

1.2.1. Ordering Dependence

Hier geht es vor allem darum, dass die logische Reihenfolge der Daten ganz unabhängig sein soll von der Reihenfolge in der maschinellen Repräsentation, z.B. von der Reihenfolge der Speicherung im Dateisystem. Laut Codd haben alle bisherigen Systeme diese Unabhängigkeit nicht erreicht.
Man mag sich fragen, warum dieser Punkt so bedeutsam sein soll. Meines Erachtens geht es hier darum, auch die sequenzielle Anordnung von Daten als Teil der logischen Datenstruktur zu begreifen und nicht von vornherein nur der Seite der maschinellen Repräsentation zuzuschlagen. Man neigt einfach leicht dazu, letzteres zu tun.

1.2.2. Indexing Dependence

Indizes sind in der Datenbankwelt so etwas wie Inhaltsverzeichnisse, in denen neben der eigentlichen Datenstruktur eine zusätzliche Ordnungsrelation aufgebaut wird, die die Performance von gewissen Abfragen in dramatischer Weise beschleunigen kann. Codd geht es darum, einen Index nicht als Teil der logischen Datenstruktur zu begreifen, sondern als Teil der maschinellen Repräsentation. Programme, die auf der logischen Ebene auf die Daten zugreifen, sollen funktionsfähig bleiben, auch wenn gewisse Indizes gelöscht oder verändert werden. Dass Indizes für eine akzeptable Performance unverzichtbar sein können, würde Codd nicht bestreiten.

Die Forderung des Autors nach Indexunabhängigkeit der logischen Datenstruktur ist sicher vernünftig, weil ein Endanwender zunächst vom Nachdenken über Indizes entlastet werden soll. Bei größeren Datenmengen kann man sich allerdings fragen, ob ein Index nicht doch Teil der logischen Datenstruktur werden kann, weil gewisse Abfragen gar nicht mehr funktionieren, wenn nicht ein bestimmter Index gesetzt ist. Dies sind aber sicher Grenzfälle, und Indizes sollten auf jeden Fall nicht im primären Fokus des Anwenders sein, der sich auf der logischen Ebene der Daten bewegt.

1.2.3. Access Path Dependence

Die Abhängigkeit bei den Zugriffspfaden ist für Codd meines Erachtens die kritischste und nimmt bei ihm auch den größten Raum ein. Die Überwindung dieser Art von Abhängigkeit zeigt wesentlich deutlicher als die beiden anderen, dass das relationale Modell über bis dahin gängige Datensysteme hinausgeht und führt uns bereits zu einem Kernaspekt, den man sich meines Erachtens genauer anschauen sollte.

Worum geht es bei den Zugriffspfaden? Die bislang dominierenden Datenmodelle bauen nach Codd entweder auf Baumstrukturen oder auf Netzwerkstrukturen auf. Verschiedene Datensätze können darin verknüpft werden, wobei bei den Netzwerkstrukturen auch m:n Beziehungen systematisch vorgesehen sind. Charakteristisch für die nicht-relationalen Modelle ist jedoch, dass die Verknüpfungen von verschiedenen Datensätzen in der Datenstruktur selbst verankert werden und damit essentieller Teil der Datenstruktur sind.

Wenn ein Datensatz mit einem anderen Datensatz verknüpft ist, enthält dieser Datensatz also einen Link oder Zeiger auf den Datensatz, mit dem er verbunden ist. Auf diese Weise enthält das Datensystem Pfade, die der Endanwender oder ein Programm benutzen kann, um sich zu orientieren und bestimmte Daten zu finden.[3] Für Codd besteht aber nun genau darin das Problem, denn Programme und Endanwender werden abhängig von der Existenz solcher Pfade. Das macht das ganze Datensystem einerseits unübersichtlich, denn die Menge solcher Pfade kann bei komplexen Datenstrukturen sehr groß werden, und andererseits unflexibel, denn jede Änderung an den Daten muss ggf. existierende Pfade auch mit ändern.

Die Abhängigkeit der Datenstruktur von den Zugriffspfaden ist nun nicht mehr eine Abhängigkeit der logischen Repräsentation der Daten von der maschinellen Repräsentation

[3] Charles W. Bachman, der Begründer des Netzwerkmodells, hat die Tätigkeit des Datenbankprogrammierers denn auch im wesentlichen als ein Navigieren beschrieben. Vgl. dazu die lesenswerte Rede von Bachman anlässlich der Verleihung des Turing Award 1973: *The Programmer as Navigator*, vgl. Bachman 1973.

wie bei den Indizes oder bei der Reihenfolge der Datensätze. Die Abhängigkeit liegt hier vielmehr auf der logischen Ebene selbst und zeigt an, dass die logische Datenstruktur im relationalen Modell ganz anders definiert wird als in den Vorgängermodellen und allen voran im Netzwerkmodell.

Zwar sind die logisch abgegrenzten Einheiten im relationalen Modell - die Tabellen - auch untereinander verbunden bzw. können verbunden werden, aber diese Verbindungen werden nur durch Werte und ihre Gleichheit im Datensystem festgelegt und nicht durch so etwas wie Verbindungsstäbe zwischen den Daten. Z.B. sind zwei Datensätze im relationalen Modell dadurch irgendwie verbunden, dass in beiden Datensätzen die Artikelnummer 456 auftaucht. Die Verbindung ist aber im Grunde nur virtuell bzw. potentiell und muss vom Anwender durch eine JOIN-Abfrage aufgerufen werden. Die Verbindung von Daten entsteht also durch Wertzusammenhänge, die in der Abfragesprache (heute meist SQL) hergestellt bzw. auf Herstellbarkeit abgefragt werden. Dadurch wird die Datenstruktur äußerst flexibel. In einem späteren Aufsatz schreibt Codd:

> Between tabular relations there are no structural links such as pointers. Associations between relations are represented solely by values. These associations are exploited by high-level operators.[4]

Der Verzicht auf fest verdrahtete Zeigerstrukturen ist zwar nicht der einzige Unterschied des relationalen Modells zu seinen Vorgängern, aber doch ein entscheidender. In gewisser Weise wird damit dem Endanwender mehr zugemutet, weil die Orientierung im Datensystem nicht mehr so einfach vorgegeben ist. Die Möglichkeiten, Beziehungen zwischen verschiedenen Daten herzustellen und auch zu erforschen, sind dadurch aber enorm erweitert worden und haben sicher dem relationalen Modell mit zu seinem breiten Erfolg verholfen.

1.3. A Relational View of Data

Nachdem Codd bis hierhin seinen Fokus auf die Kritik an bereits bestehenden Datenmodellen gelegt hat, führt er nun das relationale Datenmodell erst eigentlich ein. Er schreibt:

> The term *relation* ist used here in its accepted mathematical sense. Given sets $S_1, S_2, \ldots, S_n$ (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on. (379)

[4] Codd 1979, S. 400.

Und in einer Fußnote fügt er hinzu:

> More concisely, *R* is a subset of the Cartesian product $S_1 \times S_2 \times \ldots \times S_n$. (ebd.)

Wir sehen hier deutlich, dass der Begriff „Relation“ im relationalen Modell in einem ganz bestimmten Sinne verwendet wird, nämlich im Sinne der Mathematik. Genauer gesagt entstammt der Begriff der Mengenlehre, die sich in der zweiten Hälfte des 19. Jahrhunderts als Teilgebiet der Mathematik entwickelt hat. Nehmen wir z.B. folgende drei Mengen:

S_1 (Berlin, Paris, Rom, Madrid)
S_2 (Italien, Deutschland, Spanien, Frankreich)
S_3 (französisch, italienisch, deutsch, spanisch)

Dann können wir eine Relation bilden, indem wir aus jeder Menge jeweils ein Element entnehmen und diesen Vorgang mehrmals wiederholen. Als Ergebnis würden wir z.B. Folgendes erhalten:

1. Tupel: Rom, Italien, italienisch
2. Tupel: Paris, Frankreich, französisch
3. Tupel: Berlin, Deutschland, deutsch
4. Tupel: Madrid, Spanien, spanisch

Wir können dies nun in einer Gitterform veranschaulichen, die noch besser verdeutlicht, dass sowohl horizontal als auch vertikal bestimmte Zusammengehörigkeiten bestehen:

Stadt	**Land**	**Sprache**
Rom	Italien	italienisch
Paris	Frankreich	französisch
Berlin	Deutschland	deutsch
Madrid	Spanien	spanisch

Eine Relation lässt sich also sehr gut als Tabelle veranschaulichen, und Codd nutzt diese Veranschaulichungsform auch schon in seinem Aufsatz, obwohl er hier noch nicht explizit von Tabellen spricht, sondern von einer *array representation*, was aber auf das gleiche hinausläuft. Die obere Zeile kann man dabei der Veranschaulichung hinzufügen und damit die Mengen benennen, aus denen die Elemente jeweils entnommen sind. Für S_1 steht hier „Stadt“, für S_2 „Land“ und für S_3 „Sprache“.

Codd weist darauf hin, dass diese Veranschaulichungsform kein integraler Bestandteil des relationalen Datenmodells ist und auch durch andere Darstellungsformen ersetzt werden könnte. So wäre es z.B. möglich, unsere Tabelle zu drehen. Man würde dann Folgendes erhalten:

Sprache	italienisch	französisch	deutsch	spanisch
Land	Italien	Frankreich	Deutschland	Spanien
Stadt	Rom	Paris	Berlin	Madrid

Codds Bemerkung zielt darauf ab, eine Verwechslung zwischen Veranschaulichungsform und logischer Definition von Relationen zu vermeiden und ist insofern durchaus am Platze. Es hat sich jedoch gezeigt, dass die Darstellungsform als Tabelle mit den Mengennamen in der obersten Zeile äußerst zweckmäßig ist und von den meisten Menschen besonders leicht überschaut werden kann. Die Tabelle ist demzufolge auch zur Standardform der Veranschaulichung geworden, und Codd wird später selbst auf die Nützlichkeit dieser besonders leicht fasslichen Darstellung hinweisen.[5] Wir gehen also wieder auf unsere anfängliche Tabelle zurück:

Stadt	**Land**	**Sprache**
Rom	Italien	italienisch
Paris	Frankreich	französisch
Berlin	Deutschland	deutsch
Madrid	Spanien	spanisch

Jede Zeile repräsentiert ein Tupel unserer Relation. Codd stellt heraus, dass die Reihenfolge der Zeilen keine Rolle spielt. Es ist also ganz egal, ob wir die Zeile mit Rom, Italien und italienisch als erste Zeile aufschreiben oder als letzte. Die Relation bleibt dabei dieselbe.

Für Codd ist es jedoch ausgeschlossen, dass die gleiche Zeile in der Relation zweimal auftreten kann. Diese Forderung wurde später häufig diskutiert[6] und ist in vielen heute gängigen relationalen Datenbanksystemen nicht umgesetzt, weil man dort durchaus zweimal die gleiche Zeile in eine Tabelle einfügen kann. Über das Für und Wider in dieser Frage müssen wir hier nicht weiter sprechen. Meines Erachtens muss man diese Forderung nicht als essentiellen Part des relationalen Modells ansehen und kommt auch gut ohne sie aus.

Der Autor diskutiert nun noch die Frage, ob die Reihenfolge der Spalten für eine Relation wesentlich ist und bejaht dies zunächst auch selbst. Seine Begründung ist, dass dieselbe Spalte ja auch zweimal vorkommen kann, z.B. „Bauteil“, wobei der Wertevorrat der beiden

[5] "... tables are the most important conceptual representation of relations, because they are universally understood." Codd 1981, S. 395.
[6] Vgl. dazu z.B. Codd 1990, S. 18f.

Spalten der gleiche ist, aber die zweite Spalte z.B. anzeigt, dass es sich um eine Komponente des jeweiligen Bauteils der ersten Spalte handelt. Bauteile können also aus anderen Bauteilen zusammengesetzt sein.

Codd weist allerdings gleich im Anschluss selbst darauf hin, dass die Reihenfolge der Spalten in dem Moment unwesentlich wird, wo jede Spalte einen eigenen - in der Relation eindeutigen - Namen bekommt. So könnte man die zweite Bauteil-Spalte mit dem Namen „Bauteil-Komponente“ betiteln, und das Problem wäre gelöst.

Eine Datenbank ist dann für Codd also eine Sammlung von Relationen, die nicht starr, sondern veränderlich sind. Jede Relation kann neue Tupel aufnehmen, bestehende Tupel ändern, oder es können vorhandene Tupel entfernt werden. Die grundlegenden drei Datenmanipulationsarten des INSERT, UPDATE und DELETE sind also hier schon benannt.

Der Autor definiert noch einige Begriffe, die für den Aufbau des relationalen Modells eine Rolle spielen. So bezeichnet er die Ausgangsmengen, aus denen die Werte für die Tupel entnommen werden, als *domains*. Eine *domain* ist also die Menge der möglichen Werte, die in einer Spalte benutzt werden dürfen, z.B. die Menge der Dezimalzahlen oder die Menge aller Städte. Eine *domain* ist also nicht in Gänze aktual verwirklicht, sondern potentiell.

Die tatsächlich zu einem bestimmten Zeitpunkt aktual vorhandenen Werte einer Spalte aus einer bestimmten Relation bezeichnet Codd dagegen als *active domain*.

Weiter werden die Begriffe *primary key* (Primärschlüssel) und *foreign key* (Fremdschlüssel) definiert. Ein Primärschlüssel ist für Codd eine Spalte (oder die Kombination aus mehreren Spalten), deren Werte die Tupel der Relation eindeutig identifizieren. Wenn es genau eine Spalte ist, dann gibt es keine zwei Zeilen, die in dieser Spalte denselben Wert haben. Der Autor verwendet dabei den Begriff *primary key* so, wie man heute üblicherweise den Begriff „Schlüssel“ bzw. „Schlüsselkandidat“ verwendet, denn er betont ausdrücklich, dass es in einer Relation mehrere Primärschlüssel geben kann, was der heute gängigen Terminologie nicht entspricht. Er spricht dann aber auch davon, dass unter den evtl. existierenden verschiedenen Primärschlüsseln einer ausgewählt werden kann, der dann *der* Primärschlüssel („*the* primary key“) wird. Diese Verwendung des Begriffs entspricht dann wiederum genau der heute üblichen.

Im Unterschied zum Primärschlüssel ist ein Fremdschlüssel in einer Relation *R* gerade nicht der Primärschlüssel, sondern eine Spalte (oder Kombination aus mehreren Spalten), die in

einer anderen Relation *S* der Primärschlüssel ist. Dabei können nach Codd *R* und *S* auch dieselbe Relation sein, wobei dann natürlich die Spalte (oder Spalten) des Fremdschlüssels nicht dieselbe sein darf wie die des Primärschlüssels. Auf jeden Fall ist beim Begriff „Fremdschlüssel“ Codds Verwendung der heute üblichen durchaus konform.

Nachdem er mit der Definition der Begriffe fertig ist, macht Codd - fast nebenbei - noch eine wichtige Bemerkung:

> In previous work there has been a strong tendency to treat the data in a data bank as consisting of two parts, one part consisting of entity descriptions (for example, descriptions of suppliers) and the other part consisting of relations between the various entities or types of entities (for example, the *supply* relation). ... In the user's relational model there appears to be no advantage to making such a distinction ... (380)

Im relationalen Modell werden Entitäten auf der einen Seite und deren Beziehungen untereinander auf der anderen Seite nicht mehr unterschiedlich behandelt, sondern mit der gleichen Grundstruktur abgebildet, nämlich als Relation im Sinne des eben definierten relationalen Modells. Das relationale Modell ist also einerseits so simpel, dass es mit nur einer Grundstruktur auskommt (der Relation) und andererseits so flexibel, dass sowohl Entitäten als auch Beziehungen zwischen Entitäten in dieser Grundstruktur abgebildet werden können.

Natürlich kann man als Benutzer einer Datenbank auf konzeptueller Ebene an der für das alltägliche Denken hilfreichen Unterscheidung zwischen Entität einerseits und Beziehung zwischen Entitäten andererseits festhalten.[7] Die Überführung des konzeptuellen Modells in das Datenmodell auf relationaler Ebene tut dem ja keinen Abbruch. Es ist aber bis heute ein weit verbreitetes Missverständnis geblieben, dass das Attribut „relational“ bei Datenbanken meinen würde, es gäbe Tabellen als Abbildungen für Entitäten, die dann zueinander in Beziehung treten können, wodurch alles irgendwie „relational“ wird.

Die Tragweite der Einführung des relationalen Modells wird aber erst dann deutlich, wenn man sich vor Augen führt, dass mit der Relation eine Grundstruktur für das Modell definiert ist, die von der Mikroebene der kleinsten Tabellen bis hin zur Makroebene der wie immer komplex verschachtelten Relationen niemals verlassen wird. Man bewegt sich damit in einem komplett geschlossenen logischen Raum. Auch das logische Operationssystem des relationalen Modells - die relationale Algebra - erzeugt als Ergebnisse immer wieder nur

[7] Vgl. etwa Chen 1976.

Relationen, wie wir noch sehen werden, und führt damit ebenfalls nicht aus dieser Grundstruktur heraus.

Neben einer vereinheitlichten Grundstruktur wird mit der Einführung des relationalen Modells aber noch ein weiterer wesentlicher Schritt vollzogen, mit dem es von seinem Ansatz her über seine Vorgängermodelle hinausgeht, und das ist die *konsequente Mengenorientierung*. Wie wir gesehen hatten, sind Relationen Mengen von Daten, die man sich als Tabellen veranschaulichen kann bzw. als zweidimensionale Arrays. Relationen sind also in bestimmter Weise strukturierte Mengen. Die primären Objekte des Datensystems sind somit nicht mehr einzelne Daten oder Datensätze, sondern Mengen von Datensätzen. Die Hauptsicht des relationalen Modells - und das ist auch die Hauptsicht des Datenbanknutzers - setzt also von vornherein bei Datensatzmengen an, und auch die primäre Umgangsweise mit den Daten ist eine Bearbeitung und Verknüpfung von Mengen. Die Operationen des Datenbankbenutzers und -programmierers werden dann vor allem solche einer Mengenalgebra sein und nicht mehr das Bearbeiten von einzelnen Datensätzen oder Datenelementen. Auf der Abstraktionsebene des relationalen Modells kehrt sich die Betrachtungsrichtung von Einzelelement zu Menge gewissermaßen um. Es ist nicht mehr die Menge, die aus Einzelelementen zusammengesetzt wird, sondern der einzelne Datensatz wird als Teilmenge einer ihn umgreifenden Relation auffindbar.

Damit, dass die Mengenorientierung schon in der Grundstruktur des Datenmodells als primäre Sichtweise gesetzt wird, geht Codd im Grunde nur konsequent einen Weg weiter, der von der Daten*datei* zur Daten*bank* führt. Während in einer Datei vor allem einzelne Daten gespeichert wurden, um sie zu einem späteren Zeitpunkt für einen speziellen Zweck wieder auffinden zu können, werden in einer Datenbank große Datenbestände zusammengeführt und einem in der Regel auch großen Benutzerkreis zur Verfügung gestellt. Das Wiederauffinden einzelner Datensätze ist zwar auch dort noch ein wichtiges Ziel der Datenspeicherung, aber darin liegt nicht mehr der einzige Zweck der Datenspeicherung und tendenziell auch nicht mehr der Hauptzweck. Dieser besteht vielmehr darin, die Daten zu einem Datensystem zu organisieren und Informationen von höherem Abstraktionsgrad daraus zu entnehmen, die sich nur durch Bezüge zwischen ganzen Datenbeständen ergeben können. Solche abstrakteren Informationen lassen sich - wenn überhaupt - ohne eine primär mengenorientierte Sicht auf die Daten nur mit großem Aufwand extrahieren.

Codd weist nun darauf hin, dass er bis hierhin nur Beispiele für Relationen diskutiert hat, deren Spalten ausschließlich einfache Werte enthalten, also keine irgendwie zusammengesetzten Werte. Zusammengesetzt aufgebaute und damit zerlegbare Werte sind

aber denkbar und können im relationalen Modell nach einer bestimmten Vorgehensweise systematisch integriert werden. Als Beispiel nennt er das Attribut „Gehaltsentwicklung“ in einer Relation „Angestellter“. So ist zwar zu jedem Zeitpunkt jedem einzelnen Angestellten einer Firma nur ein Gehalt zugeordnet, aber zu verschiedenen Zeitpunkten können dies natürlich pro Mitarbeiter mehrere Gehaltsangaben sein, wenn es etwa zwischendrin eine Gehaltserhöhung gab. Jedem Mitarbeiter sind also in verschiedenen Zeiträumen verschiedene Gehaltswerte zugeordnet, so dass man das Attribut „Gehaltsentwicklung“ aufspalten und in eine separate Relation auslagern muss. Genau dies ist Teil des Prozesses, den Codd als *Normalisierung* bezeichnet und der auch heute noch so bezeichnet wird.

1.4. Normal Form

Der Prozess der Normalisierung führt dazu, dass alle Relationen in der Datenbank in einer kanonischen Form vorliegen, die man als *Normalform* bezeichnet. Heute unterscheidet man dabei verschiedene Stufen der Normalisierung und spricht etwa von erster, zweiter und dritter Normalform. Codd hat sich in späteren Arbeiten auch selbst im Detail dazu geäußert, aber im hier besprochenen Text diskutiert er nur diejenige Normalform, die heute als erste Normalform bezeichnet wird. Für das relationale Modell ist diese Normalform meines Erachtens auch die wichtigste, denn erst wenn ein Datenbestand in erster Normalform vorliegt, haben wir es mit vollgültigen Relationen im Sinne des relationalen Modells zu tun.

Die weiteren Normalisierungsschritte, die zur zweiten oder dritten Normalform führen, unterscheiden sich in ihrem Zweck deutlich von dem Schritt, der nicht-normalisierte Daten in die erste Normalform überführt. Der Hauptzweck der weiteren Normalisierungsschritte ist die Minimierung der Redundanz und Erhöhung der Konsistenz im Datenbestand. Der Zweck des ersten Normalisierungsschrittes dagegen ist die eigentliche Überführung des Datenbestandes in die relationale Struktur.

Natürlich sind auch die weiteren Normalisierungsschritte von großer Bedeutung, z.B. dann, wenn man ein solides Datenbank-Design entwerfen will. Nähere Informationen kann man sich dazu heute leicht beschaffen. Wenn man aber über die Grundlagen des relationalen Modells nachdenkt, ist ein besseres Verständnis der ersten Normalform vorrangig, und Codd behandelt ja auch im vorliegenden Text nur diese genauer. Wir konzentrieren uns daher hier auf diese Form.

Der Einfachheit halber greifen wir an dieser Stelle zur Erläuterung das Beispiel von der Gehaltsentwicklung eines Angestellten wieder auf, das Codd im vorigen Abschnitt angeführt

hatte. Codd selbst wählt im Abschnitt 1.4. zwar noch ein komplizierteres Beispiel mit mehrfacher Verschachtelung, aber das Prinzip sollte bereits an einem einfachen Beispiel deutlich werden. Nehmen wir an, die Daten über Angestellte einer Firma liegen zunächst in folgender Form vor:

Personennr	Name	Gehaltsentwicklung
1	Fesenmeyer	ab 15.01.2008: 2000 Euro, ab 01.01.2010: 2200 Euro, ab 01.01.2011: 2300 Euro
2	Heidemann	ab 01.01.2007: 2500 Euro, ab 15.06.2009: 2700 Euro
3	Seelig	ab 09.05.2005: 3400 Euro, ab 15.08.2010: 3700 Euro

Dann sind die Werte der Spalten „Personennr“ und „Name“ einfach bzw. atomar, während die Werte der Spalte „Gehaltsentwicklung“ zusammengesetzt sind. Um die Daten in die Normalform des relationalen Modells zu überführen, muss die Spalte „Gehaltsentwicklung“ in Einzelwerte zerlegt werden, die ihrerseits nicht weiter aufspaltbar sind. Die erste Idee, die man hierzu hätte, wäre vermutlich, dass man die Spalte Gehaltsentwicklung auf mehrere Spalten aufteilt. Das Problem dabei ist jedoch, dass die Anzahl der Spalten immer weiter anwachsen müsste, um im Verlaufe der Zeit immer mehr unterschiedliche Werte aufnehmen zu können. Im relationalen Modell wählt man daher einen anderen Weg, und dieser besteht darin, die vorliegenden Daten auf zwei verschiedene Relationen aufzuteilen.

Wenn man davon ausgeht, dass die Spalte „Personennr“ der Primärschlüssel für die Relation „Angestellter“ ist, dann kann man folgende zwei Relationen aufbauen:

„Angestellter“

Personennr	Name
1	Fesenmeyer
2	Heidemann
3	Seelig

„Gehaltsentwicklung“

Personennr	Von	Bis	Gehalt (Euro)
1	15.01.2008	31.12.2009	2000
1	01.01.2010	31.12.2010	2200
1	01.01.2011	31.12.9999	2300
2	01.01.2007	14.06.2009	2500
2	15.06.2009	31.12.9999	2700
3	09.05.2005	14.08.2010	3400
3	15.08.2010	31.12.9999	3700

Die erste Relation enthält also keine Spalte „Gehaltsentwicklung“ mehr und ist damit gegenüber den Ausgangsdaten verkleinert. Die zweite Relation dagegen enthält die Primärschlüsselspalte „Personennr“ der ersten Relation, welche dann in der zweiten Relation

zu einem Fremdschlüssel wird. Sie stellt den Bezug zur ersten Relation her. Der Zeitraum, in welchem ein bestimmtes Gehalt gilt, wird durch zwei Datumsspalten repräsentiert (von, bis), und das Gehalt bekommt auch eine eigene Spalte.

Im Ergebnis liegen die Daten nun in zwei Relationen vor, in denen keine Spalte mit nicht-atomaren, d.h. zusammengesetzten Werten mehr vorkommt. Das ist das entscheidende Ziel der Normalisierung, die zur heute als erste Normalform bezeichneten Datenstruktur führt.

Die in den Ausgangsdaten noch gebündelte Datenstruktur der Gehaltsentwicklung wird also durch den Normalisierungsschritt entfaltet, an die Oberfläche gehoben und ausgebreitet. Komplexität wird damit gerade nicht gebündelt und gekapselt, wie etwa im objektorientierten Modell, sondern auseinander gelegt und sichtbar gemacht. Charakteristisch für das relationale Modell ist also, dass komplexe Datenstrukturen nicht in einem hierarchischen Stufenmodell strukturiert werden wie etwa bei den Verzweigungen einer Baumstruktur. Ein damit verbundener Nachteil ist sicher ein geringeres Maß an Übersichtlichkeit. Kompensiert wird dieser Nachteil aber durch Vorteile, die die relationale Form mit sich bringt.

Wichtigster Vorteil ist das Vorliegen der Daten in der einfachen und immer gleichen Struktur der Relation, d.h. als Tabelle - oder wie Codd es in diesem Aufsatz nennt - als

> … two-dimensional column-homogeneous array … (381)

Die universale Struktur ermöglicht z.B. einen einfachen Austausch von Daten zwischen verschiedenen Datensystemen, z.B. in Form von Textdateien, wo die Werte einer Tabellenzeile als Textzeile erscheinen und voneinander durch ein Trennzeichen getrennt sind, z.B. durch ein Semikolon:

```
1;Fesenmeyer
2;Heidemann
3;Seelig
```

Die Vorteile für den Datenaustausch werden von Codd hier auch explizit erwähnt. Entscheidend ist dabei wieder, dass Relationen zwar untereinander in Beziehung stehen können, dass sie aber keine Zeiger enthalten und somit jede einzelne Relation aus dem Geflecht der Relationen herauslösbar ist, um sie etwa in ein anderes Datensystem zu exportieren und dort weiter zu verwenden. Jede einzelne Relation kann also rein für sich betrachtet werden, auch wenn die volle Semantik der Daten sich meist nur im System der

Relationen erschließt. Ein modularer Aufbau des Datensystems ist damit aber auch gesichert, was wiederum zur Übersichtlichkeit und Wartbarkeit beiträgt.

Neben der Vereinfachung des Datenaustauschs zwischen verschiedenen Systemen ist ein weiterer Vorteil der universalen Relationsstruktur die Ermöglichung einer mengenorientierten Verarbeitung der Daten auf allen Ebenen. Dadurch steht dem Nutzer und Programmierer im relationalen Modell ein mächtiges Instrument zur Verfügung, das wiederum sehr dabei hilft, der teilweise enormen Komplexität der Daten Herr zu werden. Damit sind wir bei der Frage angelangt, in welcher Sprache man mit den Daten im relationalen Modell kommunizieren kann bzw. wie man sich darin bewegt und Informationen extrahiert. Im nächsten Unterabschnitt kommt Codd darauf zu sprechen.

1.5. Some Linguistic Aspects

Der Autor beginnt hier wie folgt:

> The adoption of a relational model of data, as described above, permits the development of a universal data sublanguage based on an applied predicate calculus. A firstorder predicate calculus suffices if the collection of relations is in normal form. Such a language would provide a yardstick of linguistic power for all other proposed data languages, and would itself be a strong candidate for embedding ... in a variety of host languages ... (381)

Das Vorgehen, dass von einer *host language* (aufrufenden Programmiersprache) aus eine Datenbankabfrage in einer *sublanguage* (Dienstsprache) an die Datenbank gesendet und das Ergebnis der Abfrage weiterverarbeitet wird, ist heute gängige Praxis und den meisten Lesern sicher bestens bekannt. Z.B. sendet man aus Programmiersprachen wie Java oder PHP eine Abfrage in der Datenbanksprache SQL an eine relationale Datenbank und fängt das Ergebnis wieder auf, um damit in der aufrufenden Sprache weiterzuarbeiten. Man hat es also hier mit einer Schichtenarchitektur zu tun und Schnittstellen zwischen den Schichten, die es erlauben, Informationen auszutauschen.

Aus der Perspektive des Anwendungsentwicklers erscheint die Datenbank aber oft als bloßer Datenbehälter, aus dem man sich Daten herausholen oder auch welche hineinstecken kann. Die Datenbank ist aus dieser Perspektive nur insofern eine Weiterentwicklung der Datendatei, als die Informationen etwas besser geordnet und strukturiert vorliegen. Umso mehr mag es aus einer solchen Perspektive überraschen, dass die universale Datenabfragesprache auf der Grundlage einer *angewandten Prädikatenlogik* aufgebaut werden soll, denn war nicht die logische Programmierung mit ihrem vielleicht

bekanntesten Vertreter *Prolog* immer eine rein akademische Episode geblieben, die sich in der praktischen Programmierung nie durchsetzen konnte?

In der Tat ist dies etwas überraschend, gerade wenn man - wie ich selbst auch - ursprünglich aus der praktischen Anwendung der Informationstechnologie herkommt und von da aus ein tieferes Verständnis der Modelle sucht, die sich hinter den im Einsatz befindlichen Technologien verbergen. Die heute dominierende Abfragesprache SQL ist vielleicht die in der Praxis am weitesten verbreitete Programmiersprache, die durchaus dem logischen Programmierparadigma zugeordnet werden kann. Die Möglichkeit einer prädikatenlogisch orientierten Programmierung steht damit auf fast allen gängigen Datenbanksystemen einfach so zur Verfügung. Genutzt wird diese Möglichkeit freilich von nur wenigen Leuten, wie man schnell feststellt, wenn man sich anschaut, in welcher Weise SQL heute vorrangig verwendet wird.

Inwiefern eine auf der Prädikatenlogik aufgebaute Programmiersprache auch in der praktischen Anwendung ein sehr mächtiges Werkzeug sein kann, ist sicher ein Thema für sich und kann in diesem Kommentar nur angedeutet werden. Ich möchte an dieser Stelle aber auf einen Aspekt aufmerksam machen, der dabei eine wichtige Rolle spielt, und das ist die gewissermaßen natürliche Verwandtschaft von Prädikatenlogik und Mengenlehre, aus denen man dann ein konsequent mengenorientiertes Programmiermodell entwickeln kann. Wenn man sich nämlich den in der Prädikatenlogik zentralen Begriff des Prädikats näher anschaut, so wird schnell klar, dass man Prädikate als Werte*mengen* auffassen und auf diese Weise einen Bezug zur Mengenlehre herstellen kann. So kann man ein Prädikat wie „ist grün“ als Menge aller Werte auffassen, die die Aussageform „x ist grün“ zu einer wahren Aussage machen, und das sind - einfach gesprochen - alle grünen Gegenstände.

Ein Problem in diesem Zusammenhang liegt auch darin, dass die Benutzung mathematischer Modelle wie der Mengenlehre oder der Prädikatenlogik die Notwendigkeit mit sich bringt, die Modelle in einer formal korrekten Weise auszubuchstabieren, damit sie überhaupt maschinell verarbeitet werden können. Die formale Korrektheit ist damit essentiell, aber es ist nur die eine Seite der Medaille. Ein anderer wichtiger Aspekt ist immer auch die Abbildbarkeit unseres natürlichen Denkens in diesen formalen Modellen. In welcher Weise also ein mengenorientierter und prädikatenlogischer Ansatz dem natürlichen Denken nahekommt, müsste meines Erachtens noch viel stärker systematisch aufgezeigt werden als dies in der Regel geschieht. Auf jeden Fall hat sich auch Codd mit dieser Frage

auseinandergesetzt[8], und man würde ihm mit Sicherheit Unrecht tun, wenn man seinem Modell *in toto* so etwas wie *praxisfernen Formalismus* vorwerfen würde.

Wenn man eine Programmiersprache auf der Prädikatenlogik aufbaut, so unterscheidet sie sich von vielen herkömmlichen und heute gängigen Programmiersprachen auch darin, dass sie deklarativ und nicht imperativ ausgerichtet ist. Codd verweist darauf explizit:

> The universality of the data sublanguage lies in its descriptive ability (not its computing ability). (382)

Den bei Codd benannten Unterschied zwischen *descriptive* und *computing* gibt man heute meist als Unterschied zwischen *deklarativem* und *imperativem* Programmierstil wieder. Beim deklarativen Stil steht das *Was* der Beschreibung eines Problems im Vordergrund und nicht das *Wie* des Weges oder Prozesses zu seiner Lösung. Genau umgekehrt implementiert der imperative Stil vorrangig den Prozess in Form von Teilschritten. Auch hier wäre es freilich ein Thema für sich, zu untersuchen, wie beide Programmierstile unserem natürlichen Denken entsprechen bzw. nicht entsprechen.

Ein wesentliches Charakteristikum der Abfragesprache im relationalen Modell, auf das Codd in diesem Unterabschnitt noch hinweist, ist die prinzipielle Möglichkeit der *symmetrischen Erschließung* einer jeden Relation. Er schreibt:

> Once a user is aware that a certain relation is stored, he will expect to be able to exploit it using any combination of its arguments as “knowns” and the remaining arguments as “unknowns,” because the information (like Everest) is there. This is a system feature (missing from many current information systems) which we shall call (logically) *symmetric exploitation* of relations. (382)

und fügt hinzu:

> Exploiting a relation includes query, update, and delete. (ebd.)

Was damit gemeint ist, können wir uns an der Relation „Gehaltsentwicklung“ veranschaulichen, die wir weiter oben schon besprochen hatten:

[8] Vgl. dazu vor allem Codd 1981.

Personennr	von	bis	Gehalt (Euro)
1	15.01.2008	31.12.2009	2000
1	01.01.2010	31.12.2010	2200
1	01.01.2011	31.12.9999	2300
2	01.01.2007	14.06.2009	2500
2	15.06.2009	31.12.9999	2700
3	09.05.2005	14.08.2010	3400
3	15.08.2010	31.12.9999	3700

Wir könnten hier z.B. die Frage stellen, welche Gehälter zu einem bestimmten Zeitpunkt gezahlt wurden, um z.B. den Gehaltsdurchschnitt in der Firma zu einem bestimmten Zeitpunkt zu ermitteln. Wir würden dann das Attribut „Zeit“ als *bekannt* annehmen und das Attribut „Gehalt“ als *unbekannte* Größe, die ermittelt werden soll. Umgekehrt könnten wir aber auch danach fragen, zu welchen Zeiten bestimmte Gehälter gezahlt wurden. Dann würden wir das Attribut „Gehalt“ als *bekannt* ansetzen und das Attribut „Zeit“ als *unbekannte* Größe erfragen.

Der Unterschied zu vielen anderen Informationssystemen besteht hier vor allem darin, dass keine bestimmte Richtung mehr vorgegeben ist, wie man sich in den Daten bewegen muss, sondern dass man im Grunde von jedem beliebigen Datenpunkt in alle Richtungen gehen kann. Dies ist gerade in Baum- oder Netzwerkstrukturen nicht der Fall, weil man dort bestimmte Daten nur über festgelegte Pfade in vorgegebenen Richtungen erreicht. So gelangt man in einem typischerweise baumstrukturartig aufgebauten Dateisystem zu einer bestimmten Datei z.B. nur dadurch, dass man den Pfad zur Datei kennt, der einen durch bestimmte Verzeichnisse und Unterverzeichnisse führt.

Im relationalen Modell ist das anders. Dort kann man über Attributwerte und deren Beziehungen zu anderen Attributwerten, wie z.B. =, >, < (oder andere funktionale Beziehungen) verschiedenste Wege durch das Datensystem einschlagen. Das Bild des „Weges“ oder „Pfades“ passt hier im eigentlichen Sinne auch nicht mehr, weil im Normalfall keine einzelnen Datenpunkte mehr angesteuert werden, sondern Abfragen formuliert werden, die Ergebnis*mengen* zurückliefern. Welche möglichen Beschreibungen innerhalb der relationalen Abfragesprache zu einer bestimmten Ergebnismenge führen, muss im relationalen Modell nicht mehr vordefiniert sein, sondern liegt darin allenfalls implizit als Gesamtheit der möglichen Wertzusammenhänge im System.

Dass das relationale Modell hier eine höhere Flexibilität bietet als bis dahin gängige Datensysteme, liegt sicher auf der Hand, und die damit verbundenen Vorteile leuchten zunächst sofort ein. Es gab allerdings schon bald auch Kritiker, die in der höheren Flexibilität

eine semantische Unterspezifikation monierten, die zu Orientierungslosigkeit führe. So sind im relationalen Modell z.B. Vergleiche zwischen dem Gehalt eines Mitarbeiters und seinem Alter systematisch nicht ausgeschlossen, weil beides Zahlenwerte sind, die prinzipiell in Beziehung gesetzt werden können.[9]

Die partielle Berechtigung dieser Kritik wird man nicht ganz entkräften können, aber sie ist auch kein Einwand, der das relationale Modell im Kern treffen könnte. Die konkrete Implementierung einer Datenstruktur im relationalen Modell wird vielmehr ohne weitere semantische Spezifizierungen nicht auskommen können, die z.B. über Fremdschlüssel oder Data-Dictionaries eingebracht werden können. Meines Erachtens sollte man hier auch unterscheiden zwischen einerseits dem Bereich der Daten-*Produktion*, wo durchaus starke Restriktionsmaßnahmen am Platze sind, die die Produktion von Unsinn von vornherein minimieren, und andererseits der Daten-*Analyse* oder Daten-*Abfrage*, wo eine hohe Flexibilität geradezu zwingend nötig ist, um einen Datenbestand in seiner ganzen Tiefe erschließen zu können.

1.6. Expressible, Named, and Stored Relations

Eine Datenbank, die nach dem relationalen Modell aufgebaut ist, enthält Daten ausschließlich in Form von Relationen. Codd unterscheidet dabei nun zunächst zwischen den *benannten* und den *ausdrückbaren* Relationen.

> The named set is the collection of all those relations that the community of users can identify by means of a simple name (or identifier). (382)
>
> The expressible set is the total collection of relations that can be designated by expressions in the data language. (ebd.)

Die benannten Relationen sind diejenigen, die vermittels eines expliziten Relationsnamens angesprochen werden können. Die ausdrückbaren Relationen sind im Unterschied dazu diejenigen, die in der Datenabfragesprache auf Grundlage der benannten Relationen definiert werden können.

Die benannten Relationen zerfallen noch einmal in zwei Gruppen: die gespeicherten Relationen einerseits und die von den gespeicherten Relationen abgeleiteten, aber nicht

[9] Vgl. etwa die Kritik von Peter Chen, der als Erfinder des Entity-Relationship-Modells bekannt geworden ist, in Chen 1976, S. 27.

physisch gespeicherten. Die gespeicherten Relationen werden heute meist als Tabellen (tables) bezeichnet und die nicht gespeicherten als Sichten (views).

Die Inhalte einer Tabelle sind dabei physisch im Datenbanksystem hinterlegt. Der Benutzer der Datenbank kann im relationalen Modell nicht mehr hinter diese zurückgehen oder nach deren Grundlage suchen, sondern nur deren Daten abfragen, ändern oder löschen. Tabellen sind damit die Basis-Relationen des Systems. Die Inhalte einer Sicht dagegen sind nicht selbst physisch im Datenbanksystem gespeichert, sondern nur die Ableitungsvorschrift, durch die die Daten aus den zugrundeliegenden Tabellen abgeleitet werden. Der Benutzer der Datenbank kann also auch „hinter“ eine Sicht schauen - wenn er die dazu nötigen Berechtigungen besitzt - und Abfragen auf die der Sicht zugrundeliegenden Tabellen selbst formulieren. Eine Sicht unterscheidet sich aber dadurch von einer nur ausdrückbaren Relation, dass ihre Ableitungsvorschrift als benennbare Relation hinterlegt ist. Eine lediglich ausdrück*bare* Relation kann wiederum in eine Sicht umgewandelt werden, indem sie als solche definiert und im System abgelegt wird.

Wenn ein Datenbankbenutzer Abfragen formuliert, ist es aus seiner Perspektive im Grunde gleichgültig, ob die Abfragen Daten aus Tabellen oder Sichten entnehmen, denn beides sind Relationen und können in gleicher Weise als Grundlage für Abfragen dienen. Interessant ist hier jedoch der Punkt, dass eine Abfrage auf eine Sicht ja eigentlich eine Abfrage auf eine Abfrage - und damit eine *verschachtelte* Abfrage - ist. Abfragen in der Datensprache des relationalen Modells haben also nicht nur Relationen als *Quellen*, sondern *produzieren* auch wieder eine Relation als *Ergebnis*. Ein Charakteristikum der Datensprache im relationalen Modell ist also ihre *Geschlossenheit* in dem Sinne, dass sowohl der Ausgangspunkt als auch das Ergebnis einer Abfrage die Form einer Relation hat. Ausgangspunkte können dabei auch mehrere Relationen in einer Abfrage sein, Ergebnis ist jedoch immer genau eine Relation.

Die Möglichkeit, Relationen zu verschachteln, mag als besonders anspruchsvolles Feature des relationalen Modells gewertet werden. Mächtig ist dieses Feature aber in jedem Fall, denn es bietet die Möglichkeit, ganze Mengen zu verschachteln, und bis heute ist sicher nur wenigen Datenbankspezialisten klar, welche Tragweite dies hat.

Codd verweist noch darauf, dass große Herausforderungen zu bewältigen sein werden, ehe Datenbanksysteme zur Verfügung stehen, die Abfragen im relationalen Modell auch effizient beantworten können. Hierzu wäre zu sagen, dass Performance-Fragen in der Anfangszeit des relationalen Modells natürlich im Vordergrund stehen mussten, gerade weil es bereits performante Datenbanksysteme gab, die anderen Datenmodellen folgten. Heute jedoch ist

diese Frage nicht mehr so virulent, weil inzwischen auch performante Umsetzungen des relationalen Modells in Datenbanksystemen existieren.

2. Redundancy and Consistency

2.1. Operations on Relations

Codd beschreibt nun die Operationen, die auf Relationen angewendet werden können, im einzelnen etwas näher. Er verweist darauf, dass Relationen Mengen sind und demzufolge die üblichen Mengenoperationen angewendet werden können. Hierzu zählen vor allem die *Vereinigung* von Mengen, die Bildung des *Durchschnitts*, der *Differenz* und des *Kartesischen Produkts*. Allerdings führt z.B. nicht jede Vereinigung zweier Relationen wieder zu einer Relation - wie Codd anmerkt. Die Vereinigung einer binären und einer ternären Relation etwa ist keine Relation. Die im relationalen Modell zugelassenen Operationen müssen also die Bedingung erfüllen, dass sie als Ergebnis wieder eine Relation produzieren.

2.1.1. Permutation

Die erste beschriebene Operation ist die *Permutation*, die einfach in der Umordnung der Spalten einer Relation besteht. Diese Operation ist immer möglich, sie verändert eine Relation aber nicht, sondern liefert als Ergebnis wieder die Ausgangsrelation selbst. Sie kann jedoch aus Gründen der Übersichtlichkeit von Bedeutung sein.

2.1.2. Projection

Eine wichtige Operation ist die *Projektion*, die auch in der heute gängigen Terminologie noch so genannt wird. Sie besteht einfach darin, dass gewisse Spalten einer Relation ausgewählt und andere dabei weggelassen werden. Diese Operation ist besonders einfach, da die Auswahl der Spalten stets durch Aufzählung der Spaltennamen erfolgt.

Codd verweist darauf, dass eine Projektion erst dann korrekt abgeschlossen ist, wenn aus dem Ergebnis alle Zeilenduplikate entfernt sind. Dies ist in den heute gängigen Datenbanksystemen in der Regel nicht der Fall. Dort werden die Duplikate nicht automatisch entfernt. Dies muss - falls gewünscht - ausdrücklich angegeben werden. Die Zulässigkeit von Zeilenduplikaten ist - wie bereits erwähnt - ein Unterschied der meisten heutigen Systeme zum von Codd entworfenen relationalen Modell.

2.1.3. Join

Eine weitere Operation bezeichnet Codd als *Join*. Dies ist keine einstellige Operation, die aus einer Relation eine andere Relation formt, sondern eine mehrstellige Operation, wo aus zwei oder mehr Relationen eine neue Ergebnisrelation entsteht. Codd definiert den Join-Begriff an dieser Stelle jedoch anders als dies heute üblich ist, und er ist selbst auch schon kurze Zeit später zu einer anderen Definition übergegangen[10], die der heute gängigen sehr nahe kommt. Offenbar hat Codd selbst eingesehen, dass seine Definition aus dem vorliegenden Text in verschiedene Probleme führt, die er später umgehen wollte.

Warum nun ist Codds Join-Begriff im vorliegenden Text problematisch? Das Hauptproblem besteht meines Erachtens darin, dass der Join zweier Relationen hier nicht in eindeutiger Weise zu einer Ergebnisrelation führt. Das wird von Codd selbst auch ausdrücklich erwähnt, aber seine Versuche, die fehlende Eindeutigkeit doch noch herzustellen, sind m.E. wenig geradlinig und viel zu kompliziert. Schauen wir uns dazu Codds eigenes Beispiel näher an. Folgende zwei Relationen R und S seien gegeben:

R

supplier	part
1	1
2	1
2	2

S

part	project
1	1
1	2
2	1

Man sieht sofort, wie man die beiden Relationen durch einen Join verbinden kann, weil sie eine Spalte gemeinsam haben, nämlich *part*. So kann man alle Zeilen von R mit allen Zeilen von S kombinieren, wo in Spalte *part* derselbe Wert steht. Wenn man dies tut, führt man einen *natürlichen Join* aus, der von Codd im Text bereits erwähnt wird und auch heute noch so heißt. Die Spalte *part* wird in der Ergebnisrelation dabei nicht doppelt aufgeführt. Man erhält folgendes Ergebnis:

[10] Vgl. z.B. Codd 1972, S. 9f.

supplier	part	project
1	1	1
1	1	2
2	1	1
2	1	2
2	2	1

Die erste Zeile von R ist dabei mit der ersten und zweiten Zeile von S kombinierbar, weil *part* dort auch den Wert 1 hat. Die zweite Zeile von R ist ebenfalls mit der ersten und zweiten Zeile von S kombinierbar, und die dritte Zeile von R ist nur mit der dritten Zeile von S kombinierbar.

Soweit, sogut. Bis hierhin geht Codd noch mit dem heutigen Join-Begriff konform. Für ihn ist nun aber auch folgende Relation ein gültiger Join von R und S:

supplier	part	project
1	1	2
2	1	1
2	2	1

Wie kommt das? Es liegt an der speziellen Join-Definition, die Codd im vorliegenden Text benutzt. Für ihn ist nämlich hier ein Join diejenige Operation, aus deren Ergebnisrelation sich die Ausgangsrelationen mittels Projektion ohne Informationsverlust wieder rekonstruieren lassen. Wie man leicht überprüfen kann, lassen sich aus beiden oben aufgeführten Ergebnissen die Ausgangsrelationen R und S mittels Spaltenauswahl wieder rekonstruieren.

Offenbar führt die Fokussierung auf die Rekonstruierbarkeit der Ausgangsrelationen, die Codd hier vornimmt, auf einen Holzweg. Es mag zwar sein, dass es verschiedene Joins gibt, aus denen sich die Ausgangsrelationen wieder eindeutig rekonstruieren lassen, aber wenn man sich darauf konzentriert, führt man in die Operation des Joins eine unnötige Bidirektionalität ein, die unliebsame Komplikationen mit sich bringt.

Es mag sein, dass Codd hier zunächst eine Parallele zur oben bereits beschriebenen symmetrischen Auswertbarkeit von Relationen im Blick hatte, die ja als Stärke des relationalen Modells gelten darf. Während die bidirektionale Auswertbarkeit einer einzelnen Relation durchaus ein praktikables Charakteristikum des relationalen Modells ist, muss das Operationssystem offenbar gerade unidirektional aufgebaut sein, um überschaubar zu bleiben. Auf jeden Fall hat auch Codd selbst später die relationale Algebra genau in diesem Sinne weiterentwickelt.

2.1.4. Composition

Die Operation, die Codd *Composition* nennt, spielt heute keine entscheidende Rolle mehr. Da diese Operation auf dem Join-Begriff des vorliegenden Textes aufbaut, ist es nach meiner Auffassung wenig hilfreich, das ausführlich zu kommentieren, was Codd dazu schreibt. Es sei hier nur kurz Codds eigene Definition wiedergegeben, wonach die *Composition* eine Projektion ist, die nach dem Join ausgeführt wird und diejenige Spalte weglässt, auf welcher der Join beruht. Im oben genannten Beispiel würde man also die Spalte *part* ganz weglassen und erhielte das Ergebnis der *Composition*.

2.1.5. Restriction

Als Restriktion (oder Selektion) wird heute in der Regel die Operation der Zeilenauswahl in einer Relation bezeichnet. Die Zeilenauswahl erfolgt dabei normalerweise nicht durch ausdrückliche Aufzählung der Zeilen, sondern durch eine Bedingung, der die Werte einer oder mehrerer Spalten genügen müssen.

Codds Begriff der *Restriktion* im vorliegenden Text bewirkt zwar auch eine Zeilenauswahl in einer Relation, aber die Auswahl erfolgt durch den Bezug auf eine andere Relation und ist damit ein Join im heutigen Sinne. Sehen wir uns dazu Codds eigenes Beispiel näher an, bei dem eine Relation R durch eine andere Relation S restringiert wird:

R

s	p	j
1	a	A
2	a	A
2	a	B
2	b	A
2	b	B

S

p	j
a	A
c	B
b	B

Das Ergebnis der Restriktion ist folgendes:

s	p	j
1	a	A
2	a	A
2	b	B

Die Restriktion ist also nur möglich, wenn die zweite Relation S nur Spalten enthält, die auch in R vorhanden sind. Für die Werte der Spalten gilt das nicht, wie man z.B. an der zweiten Zeile von S sieht, deren Werte in R nicht vorhanden sind. Das Ergebnis der Restriktion im Sinne Codds sind nun diejenigen Zeilen von R, die in den zu S gleichnamigen Spalten (hier: p und j) mit einer Zeile von S übereinstimmen. Wenn wir die Zeilen von R einzeln durchgehen, sehen wir, dass die erste und zweite Zeile von R mit der ersten Zeile von S übereinstimmen, die dritte und vierte Zeile von R keine Entsprechung finden, und die fünfte Zeile von R zur dritten Zeile von S passt. So finden wir also die erste, zweite und fünfte Zeile von R im Ergebnis wieder.

In der Abfragesprache SQL könnte man heute Codds Beispiel so formulieren:

```
SELECT R.*
FROM R INNER JOIN S
ON
    R.p = S.p
AND R.j = S.j
```

2.2. Redundancy / 2.3. Consistency

In den Abschnitten 2.2. und 2.3. geht Codd kurz auf die Probleme ein, die mit den Begriffen *Redundanz* und *Konsistenz* benannt sind. Dieser Themenbereich ist normalerweise sehr umfangreich und wird heute oft unter dem Titel der *Normalisierung* ausführlich behandelt. Da Codd diese Probleme im vorliegenden Text im Grunde nur anreißt, ohne sie detailliert abhandeln zu können, werden auch wir hier nur ein Beispiel des Autors aufgreifen und daran erläutern, wie Redundanzen und Inkonsistenzen in Datenbanken zustande kommen können. Wir verzichten auch darauf, Codds Unterscheidung zwischen starker (*strong*) und schwacher (*weak*) Redundanz näher zu kommentieren.

Bevor wir Codds Beispiel erläutern wollen, seien an dieser Stelle die Begriffe *Redundanz* und *Konsistenz* noch einmal kurz definiert. Redundanz bedeutet, dass ein und dieselbe Information in einer Datenbank mehrfach abgelegt ist. Dies führt einerseits zu einem höheren Aufwand bei der Änderung dieser Information, weil sie dann an mehreren Stellen geändert

werden muss, und es beinhaltet zum anderen die Gefahr, dass Unstimmigkeiten entstehen, nämlich dann, wenn die redundant vorkommende Information nur an einer Stelle geändert wird. In diesem Fall entstehen Inkonsistenzen im System, d.h. Widersprüche. Konsistenz bedeutet folglich Widerspruchsfreiheit, d.h. dass solche Widersprüche nicht existieren.

Redundanz liegt übrigens nicht nur dann vor, wenn eine Information in identischer Weise mehrmals vorkommt, sondern auch, wenn eine Information mit Hilfe der relationalen Operationen vollständig aus anderen Informationen hergeleitet werden kann. Je nachdem wie komplex solche Herleitungen sind, kann man natürlich verschiedene Stufen von Redundanz unterscheiden, und in eine solche Richtung ging auch Codds Versuch, im vorliegenden Text zwischen starker und schwacher Redundanz zu unterscheiden.

Man muss jedoch immer im Blick behalten, dass reale Datenbanken in der Regel nie ganz frei von Redundanzen und Inkonsistenzen sind. In geringen und vor allem kontrollierten Maßen führen sie meist nicht zu gravierenden Problemen. Erst wenn sie Überhand nehmen und nicht mehr kontrollierbar sind, treten ernsthafte Schwierigkeiten auf. Redundanzen und Inkonsistenzen sollten trotzdem so gering wie möglich gehalten werden, und wenn sie vermeidbar sind, auch vermieden werden.

Auf Seite 386 gibt Codd zwei Beispiele, von denen wir das erste näher erläutern wollen. Das Beispiel ist eine Relation *employee*, die aus vier Spalten besteht:

employee (serial #, name, manager #, managername)

Mit den Rauten (#) kennzeichnet Codd Schlüsselspalten, wobei *serial* Primärschlüssel und *manager* Fremdschlüssel ist. Wir formulieren hier die Relation mit deutschsprachigen Begriffen um und befüllen sie gleich mit einigen Beispielzeilen, um das Ganze etwas anschaulicher zu machen:

Angestellter

Lfdnr	Name	Vorgesetzter	Name des Vorgesetzten
1	Lehmann	3	Hahn
2	Kluge	2	Kluge
3	Hahn	2	Kluge
4	Krause	3	Hahn
5	Schmidt	1	Lehmann

Die Spalte *Lfdnr* ist der Primärschlüssel der Relation, so dass jedem Angestellten eine Nummer eindeutig zugeordnet ist. Die Spalte *Name* enthält den Namen des Angestellten,

wobei es natürlich auch vorkommen kann, dass zwei verschiedene Angestellte den gleichen Namen haben. Die Spalte *Vorgesetzter* enthält die Information, wer der Vorgesetzte des jeweiligen Angestellten ist. Dies ist eine Fremdschlüsselspalte, weil die Werte, die darin vorkommen können, genau die laufenden Nummern aus der ersten Spalte sind. Vorgesetzter kann also nur sein, wer zugleich ein Angestellter ist.

An dieser Stelle entsteht natürlich die Frage, was mit denjenigen Angestellten ist, die keinen Vorgesetzten mehr haben, an den sie berichten müssen. Zwei Möglichkeiten wären denkbar: zum einen könnte dann die Spalte *Vorgesetzter* leer bleiben, zum anderen könnte die laufende Nummer des Angestellten wiederholt werden, und diesen zweiten Weg haben wir hier gewählt, wie man in der zweiten Beispielzeile sieht.

Zu guter Letzt haben wir die Spalte *Name des Vorgesetzten*, und hier sieht man leicht, dass diese Information redundant vorliegt, denn sie lässt sich aus den Informationen der ersten drei Spalten vollständig herleiten. Nehmen wir z.B. die erste Zeile. Dort sehen wir, dass dem Angestellten mit der Nummer 1 (Lehmann) der Angestellte mit der Nummer 3 als Vorgesetzter zugeordnet ist. Den Namen dieses Vorgesetzten können wir herleiten, indem wir die Zeile suchen, wo Lfdnr = 3 ist und dort aus der zweiten Spalte den Namen „Hahn“ entnehmen. Dass also „Hahn“ auch in der letzten Spalte der ersten Zeile vorkommt, ist eine Redundanz.

Die mit einer solchen Redundanz verbundene Gefahr erkennt man sofort, wenn man den Fall annimmt, dass der Angestellte Hahn seinen Namen ändert, z.B. durch Heirat. Von einem bestimmten Zeitpunkt an heißt der Angestellte mit der Nummer 3 dann nicht mehr Hahn, sondern z.B. Schmidt. Das wird dann vielleicht in der Datenbank auch geändert, aber es ist möglich, dass es nur in der dritten Zeile (Spalte *Name*) geändert wird und in der ersten Zeile (Spalte *Name des Vorgesetzten*) das veraltete „Hahn“ stehen bleibt. In diesem Fall entsteht eine Inkonsistenz in der Relation, die erst dann beseitigt ist, wenn der Name „Hahn“ auch in der letzten Spalte überall dort geändert wird, wo in Spalte *Vorgesetzter* eine 3 steht.

Um diese Probleme von vornherein zu vermeiden, könnte man z.B. in der Datenbank zunächst eine Tabelle *Angestellter* anlegen, wo man die letzte Spalte weglassen würde:

Angestellter

Lfdnr	Name	Vorgesetzter
1	Lehmann	3
2	Kluge	2
3	Hahn	2
4	Krause	3
5	Schmidt	1

In einem zweiten Schritt könnte man dann mit Hilfe einer Ableitungsregel eine Sicht definieren, die zusätzlich eine Spalte *Name des Vorgesetzten* enthält. In SQL würde die Definition z.B. so aussehen:

```
CREATE VIEW v_Angestellter
AS
SELECT
      t1.Lfdnr,
      t1.Name,
      t1.Vorgesetzter,
      t2.Name AS 'Name des Vorgesetzten'
FROM Angestellter t1 INNER JOIN Angestellter t2
ON
t1.Vorgesetzter = t2.Lfdnr
```

Die Redundanz wäre nun durch eine explizite Ableitung vermieden, und die Namensänderung müsste nur noch an einer Stelle erfolgen, nämlich in der Ursprungstabelle *Angestellter*. In der Sicht *v_Angestellter* würde die Namensänderung in der letzten Spalte automatisch wirksam werden.

Die Erläuterung dieses Beispiels mag genügen, um zumindest ansatzweise vor Augen zu führen, wie Redundanzen und Inkonsistenzen in relationalen Datenbanken entstehen und welche Probleme damit verbunden sein können. Das Thema ist in der Praxis so wichtig wie komplex, und man kann problemlos ganze Bände damit füllen, typische Arten von Redundanzen und Inkonsistenzen in Datenbanken zu klassifizieren und mögliche Wege zu deren Beseitigung und Vermeidung aufzuzeigen. Das können und wollen wir an dieser Stelle nicht tun. Der interessierte Leser kann sich hiermit tiefer befassen, indem er z.B. das Thema *Normalisierung* an anderer Stelle weiterverfolgt.

2.4. Summary

Die Zusammenfassung, die Codd am Ende seines Textes gibt, fällt sehr kurz aus. Er gesteht hier auch selbst ein, dass er viele Fragen und Probleme nur anreißen konnte, so dass vieles offen bleiben musste. Er ist sich jedoch darüber im klaren, dass er etwas Neues im Bereich der Datensysteme ins Feld geführt hat und schließt mit folgendem Satz:

> It is … hoped that this paper can contribute to greater precision in work on formatted data systems. (387)

Nachbemerkung

Wir sind am Ende des Textes angelangt, der als Gründungsdokument für das relationale Modell im Bereich der Datenbanksysteme angesehen wird.[11] Dass dieser Text auch heute noch lesenswert ist und einem helfen kann, die Grundidee der relationalen Datenbanken besser zu verstehen, liegt m.E. vor allem daran, dass Codd das relationale Modell auf einer logischen Ebene der Datenanordnung und -verarbeitung definiert, die unabhängig von ihrer physischen Umsetzung in einem technischen System ist. Diese logische Ebene des Datenmodells entwickelt sich damit auch weitgehend unabhängig von Hardware und Software zur technischen Umsetzung des Modells in einer Maschine, unterliegt also anderen Entwicklungszyklen als die Technik selbst. Die Betonung der Selbständigkeit der Datenschicht ist auf jeden Fall eine der Hauptthesen von Codds Text, vielleicht sogar *die* Hauptthese.

Interessant ist weiterhin die Auseinandersetzung des Autors mit anderen Datenbanksystemen, die 1970 bereits in relativ ausgereifter Form vorlagen, wie z.B. das Netzwerkmodell von Bachman. Wichtigster Kritikpunkt ist hier, dass diese Datensysteme die Unabhängigkeit der Datenschicht nicht konsequent verfolgen und dadurch mit verschiedenen Problemen konfrontiert werden.

Um die unabhängige Datenschicht des relationalen Modells genauer in ihrer Struktur zu charakterisieren, greift Codd besonders auf zwei Theorien aus der Mathematik zurück: auf die Mengenlehre und die Prädikatenlogik. Die Daten werden dabei von vornherein in Mengenform gedacht, so dass auch deren Verarbeitung in Mengenoperationen konzipiert wird. Später wird Codd explizit zeigen, dass im relationalen Modell Mengenlehre und

[11] Zwar gibt es schon ein Jahr zuvor, 1969, ein Forschungspapier von Codd, das viele wesentliche Gedanken des Textes von 1970 bereits enthält, aber dieses Papier ist nur einem sehr kleinen Kreis von Rezipienten bekannt geworden. Vgl. dazu Codd 1969. Erst der hier kommentierte Aufsatz hat breite Beachtung gefunden und zum Eingang von Codds Ideen sowohl im universitären als auch im industriellen Bereich geführt.

Prädikatenlogik ineinander verschmelzen, indem er das relationale Operationssystem einmal aus mengentheoretischen Operationen aufbaut und zum zweiten aus einem Prädikatenkalkül, und danach die Gleichmächtigkeit der Ausdruckskraft beider Systeme nachweist.[12]

Besonders der erste Hauptabschnitt von Codds Text hat m.E. nichts von seiner Aktualität eingebüßt, auch heute nicht. Etwas anders ist es natürlich mit dem zweiten Abschnitt, wo es um die konkrete Ausformung der relationalen Sprache sowie um die Probleme der Redundanz und Inkonsistenz geht. Dort ist vieles noch unausgereift und hat sich später teilweise in andere Richtungen entwickelt als Codd es 1970 im Visier hatte. Es ist jedoch verständlich, dass gerade die konkrete praktische Ausformung des Modells nicht gleich am Anfang in optimaler Weise in den Blick kommen konnte. In dieser Hinsicht ist also der Text zum Teil auch veraltet. Erst wenn man als Leser selbst die Spreu vom Weizen zu trennen vermag, wird man die bleibenden Einsichten des Textes erkennen und ein besseres Verständnis der Idee des relationalen Modells aus der Lektüre mitnehmen können. Dies ein Stück weit mit vorzubereiten, war das Hauptziel dieses Kommentars.

Literaturverzeichnis

Bachman, Charles W. (1973): *The Programmer as Navigator*. ACM Turing Award lecture, Communications of the ACM, Volume 16, Issue 11, 1973, S. 653-658.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

Chen, Peter (1976): *The Entity-Relationship Model - Toward a Unified View of Data*, ACM Transactions on Database Systems, Vol.1, No.1, March 1976, S. 9-36.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

Childs, D.L. (1968): *Feasibility of a set-theoretic data structure : a general structure based on a reconstituted definition of relation*, Technical Report, University of Michigan, 1968, 37 Seiten.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

Codd, E.F. (1969): *Derivability, Redundancy and Consistency of Relations Stored in Large Data Banks*, IBM, San Jose, California, IBM Research Report RJ599, 1969, S. 1-13.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

[12] Vgl. Codd 1972.

Codd, E.F. (1970): *A Relational Model of Data for Large Shared Data Banks*, Communications of the ACM Volume 13, Number 6 (June 1970), 377-387.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

Codd, E.F. (1972*): Relational Completeness of Data Base Sublanguages*, San Jose, IBM Research Report RJ987, March 1972, S. 1-36.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

Codd, E.F. (1979): *Extending the Relational Model to Capture More Meaning*, ACM Transactions on Database Systems, Vol.4, No.4, December 1979, S. 397-434.

Codd, E.F. (1981): *Relational Database: A Practical Foundation for Productivity*, in: Ashenhurst, Robert L. et al.: *ACM Turing Award Lectures. The First Twenty Years, 1966 - 1985*, ACM Press 1987, S. 391-410.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

Codd, E.F. (1990): *The Relational Model for Database Management. Version 2*, Addison-Wesley, Reading u.a. 1990, 538 Seiten.

Levien, R. & Maron, M.E. (1965): *Relational Data File: A Tool for Mechanized Inference Execution and Data Retrieval*, Technical report, The Rand Corporation Dec. 1965, 88 Seiten.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]

Levien, R. & Maron, M.E. (1967): *A computer system for inference execution and data retrieval*, Communications of the ACM, Volume 10 Issue 11, Nov. 1967, S. 715-721.
[Im März 2011 gab es eine frei zugängliche Online-Version dieses Textes.]