IN DIESEM KAPITEL

- Mit Dateien und Datenbanken arbeiten
- Erkennen, wie Datenbanken, Abfragen und Datenbankanwendungen zusammenhängen
- Verschiedene Datenbankmodelle kennenlernen
- Einen Überblick über die Entwicklung der relationalen Datenbanken erhalten

Kapitel 1
Relationale Datenbanken

SQL (ausgesprochen *ess kiu ell,* manchmal auch *se quel*) ist die internationale Standardsprache in Verbindung mit relationalen Datenbanken. Relationale Datenbanken sind die auf der ganzen Welt vorherrschende Form der Datenspeicherung. Um zu verstehen, *warum* relationale Datenbanken die wichtigsten Datenspeicher für kleine und große Unternehmen sind, müssen Sie zunächst die verschiedenen Arten der Speicherung von Computerdaten verstehen und wie diese Speichermethoden mit dem relationalen Datenbankmodell zusammenhängen. Um Ihnen dabei zu helfen, verbringe ich einen großen Teil dieses Kapitels damit, bis in die frühesten Tage der Computer zurückzublicken und die Geschichte der Datenspeicherung zu rekapitulieren.

Mir ist klar, dass große historische Übersichten nicht jedermanns Sache sind. Dennoch möchte ich aufzeigen, dass die verschiedenen Datenspeicherstrategien, die im Laufe der Jahre verwendet wurden, jeweils ihre eigenen Stärken und Schwächen haben. Letztlich überwogen die Stärken des relationalen Modells seine Schwächen, und es wurde zu der am häufigsten verwendeten Methode der Datenspeicherung. Kurz darauf wurde SQL die am häufigsten verwendete Methode für den Umgang mit Daten, die in einer relationalen Datenbank gespeichert sind.

Verstehen, warum heutige Datenbanken besser sind als frühere

In den Anfängen des Computings war das Konzept der Datenbank eher theoretisch als praktisch. Vannevar Bush, der Visionär des 20. Jahrhunderts, hatte die Idee einer Datenbank bereits 1945, noch bevor der erste Computer gebaut worden war. Praktische

Implementierungen von Datenbanken kamen jedoch erst einige Jahre später auf, wie beispielsweise das IMS (Information Management System) von IBM, in dem alle Komponenten der Apollo-Mondmission und ihrer kommerziellen Nachfolger erfasst wurden. Viel zu lange wurden Computerdaten in Dateien gespeichert, statt sie in Datenbanken zu überführen.

Komplexität

Jedes Softwaresystem, das eine nützliche Funktion ausführt, ist komplex. Je umfangreicher die Funktion ist, desto komplexer ist ihre Implementierung. Unabhängig davon, wie die Daten gespeichert werden, bleibt die Komplexität bestehen. Die einzige Frage ist, wo sich diese Komplexität befindet.

Jede nicht triviale Computeranwendung besteht aus zwei Hauptkomponenten: dem Programm und den Daten. Obwohl der Grad der Komplexität einer Anwendung von der auszuführenden Aufgabe abhängt, haben die Entwickler eine gewisse Kontrolle über den Ort dieser Komplexität. Die Komplexität kann hauptsächlich im Programmteil des Gesamtsystems oder auch im Datenteil angesiedelt sein. In den folgenden Abschnitten erläutere ich, wie sich der Ort der Komplexität in Datenbanken im Laufe der Jahre verschoben hat, als technologische Verbesserungen dies möglich machten.

Datenverwaltung mit komplizierten Programmen

Bei den frühesten Anwendungen von Computern zur Lösung von Problemen lag die gesamte Komplexität im Programm selbst. Die Daten bestanden aus Datensätzen fester Länge, die sequenziell in einer »flachen« Datei gespeichert wurden. Dies wird als Flat-File-Datenstruktur bezeichnet. Die Datendatei enthält nichts anderes als Daten. Die Programmdatei muss Informationen darüber enthalten, wo sich bestimmte Datensätze innerhalb der Datendatei befinden (eine Form von *Metadaten,* deren einziger Zweck es ist, die Primärdaten zu organisieren, die Sie *wirklich* interessieren). Bei dieser Art der Organisation liegt die Komplexität der Datenverwaltung also vollständig im Programm.

Hier ein Beispiel für Daten, die in einer flachen Dateistruktur organisiert sind:

```
Thomas Meier Siebenbürgerstr. 8 Neustadt BY 92683
Anne Schulz Küstenweg 373 Ehlingen BW 72705
Adrian Berger Bergstraße 37 Aberberg BW 92640
Arthur Maus Geheimrat-Ecker-Str. 2 Gartenstadt BW 72643
Johannes Sieber Fichtenweg 1 Berg BW 72715
Leonie Zuckerguss Hauptstraße 1243 Stanton BW 72610
Sebastian Gsangl Weidenstraße 44 Schafberg BW 72635
Matthias Holzer Kurze Lohe 292 Anaheim BW 72640
Felix Flatter Auenstraße 342 Burgstadt BW 72610
Juliane Jung Sonnenstraße 1257 Bad Bergen BW 72705
```

Dieses Beispiel enthält Felder für Name, Adresse, Stadt, Bundesland und Postleitzahl. Jedes Feld hat eine bestimmte Länge, und die Dateneinträge müssen so gekürzt werden, dass sie in diese Länge passen. Wenn die Einträge nicht den gesamten ihnen zugewiesenen Platz ausnutzen, wird Speicherplatz verschwendet.

Die Flat-File-Methode der Datenspeicherung hat mehrere Konsequenzen, einige davon vorteilhaft, andere nicht. Zunächst zu den positiven Merkmalen:

- ✔ **Der Speicherbedarf ist minimal.** Da die Datendateien nur Daten enthalten, benötigen sie nur ein Minimum an Platz auf Festplatten oder anderen Speichermedien. Der Code, der zu einem Programm hinzugefügt werden muss, das die Metadaten enthält, ist minimal im Vergleich zu dem Aufwand, der mit dem Hinzufügen eines Datenbankmanagementsystems (DBMS) auf der Datenseite des Systems verbunden ist. (Ein *Datenbankmanagementsystem* ist das Programm, das den Zugriff auf eine Datenbank – und die Operationen für diese – steuert.)

- ✔ **Operationen mit den Daten können schnell sein.** Da das Programm direkt mit den Daten interagiert, ohne dass ein DBMS dazwischengeschaltet ist, können gut konzipierte Anwendungen so schnell laufen, wie die Hardware es erlaubt.

Wow! Was könnte besser sein? Eine Datenorganisation, die den Speicherbedarf minimiert und gleichzeitig die Arbeitsgeschwindigkeit maximiert, scheint das Beste aus allen möglichen Welten zu sein. Aber Moment …

Flache Dateisysteme kamen in den 1940er-Jahren zum Einsatz. Wir kennen sie schon seit Langem, und dennoch werden sie heute fast vollständig durch Datenbanksysteme ersetzt. Woran liegt das? Vielleicht sind es die nicht ganz so vorteilhaften Merkmale:

- ✔ **Die Aktualisierung der Datenstruktur kann eine enorme Aufgabe sein.** Es ist üblich, dass die Daten eines Unternehmens von mehreren Anwendungsprogrammen mit unterschiedlichen Zwecken verarbeitet werden. Wenn sich die Metadaten über die Struktur der Daten im Programm befinden und nicht an die Daten selbst angehängt sind, müssen *alle* Programme, die auf diese Daten zugreifen, bei jeder Änderung der Datenstruktur angepasst werden. Dies verursacht nicht nur eine Menge redundanter Arbeit (weil in allen Programmen dieselben Änderungen vorgenommen werden müssen), sondern ist auch häufige Ursache von Problemen. Alle Programme müssen auf genau die gleiche Weise geändert werden. Wenn ein Programm versehentlich vergessen wird, wird es beim nächsten Start nicht mehr funktionieren. Selbst wenn alle Programme geändert werden, werden alle, die nicht genau so geändert wurden, wie es sein sollte, fehlschlagen oder, noch schlimmer, die Daten beschädigen, ohne dass es einen Hinweis darauf gibt, dass etwas nicht stimmt.

- ✔ **Flache Dateisysteme bieten keinen Schutz für einzelne Datenelemente.** Bei flachen Dateien haben Sie entweder Lese-/Schreibzugriff auf die gesamte Datei oder auf keinen Teil der Datei. Ein Flat-File-System verfügt nicht über ein Datenbankmanagementsystem, mit dem Sie den Zugriff auf die Daten auf autorisierte Benutzer beschränken können.

- ✔ **Die Geschwindigkeit kann beeinträchtigt werden.** Der Zugriff auf Datensätze in einer sehr großen flachen Datei kann tatsächlich viel langsamer sein als ein ähnlicher Zugriff in einer Datenbank, da flache Dateisysteme keine Indizierung unterstützen. Die Indizierung ist ein wichtiges Thema, das ich in Kapitel 9 behandle.

- ✔ **Die Portabilität wird zum Problem.** Wenn in jedem Programm fest kodiert ist, wie ein bestimmter Abschnitt der Daten von einem bestimmten Laufwerk abgerufen wird, was passiert dann, wenn Ihre Hardware veraltet ist und Sie auf ein neues System umsteigen müssen? Alle Ihre Anwendungen müssen geändert werden, um die neue Art des Datenzugriffs zu berücksichtigen. Diese Aufgabe ist so mühsam, dass sich viele Unternehmen dazu entschlossen haben, mit alten, schlecht funktionierenden Systemen weiterzuarbeiten, anstatt den Aufwand einer Umstellung auf ein System auf sich zu nehmen, das ihre Anforderungen viel effektiver erfüllen würde. Unternehmen mit Altsystemen, die aus Millionen von Codezeilen bestehen, sitzen sozusagen in der Falle.

In den Anfängen der elektronischen Datenverarbeitung war Speicherplatz relativ teuer, sodass die Systementwickler hoch motiviert waren, ihre Aufgaben mit so wenig Speicherplatz wie möglich zu bewältigen. Außerdem waren die Computer damals viel langsamer als heute, sodass die schnellstmögliche Erledigung von Aufgaben ebenfalls hohe Priorität hatte. Diese beiden Überlegungen machten flache Dateisysteme zur Architektur der Wahl, trotz der Probleme, die mit der Aktualisierung der Datenstruktur eines Systems verbunden sind.

Heute ist die Situation eine völlig andere. Die Kosten für die Datenspeicherung sind drastisch gesunken und sinken weiterhin exponentiell. Die Geschwindigkeit, mit der Berechnungen durchgeführt werden, hat exponentiell zugenommen. Infolgedessen sind die Minimierung des Speicherbedarfs und die Maximierung der Geschwindigkeit, mit der ein Vorgang ausgeführt werden kann, nicht mehr die primären Triebkräfte, die sie einst waren. Da die Systeme immer größer und komplexer geworden sind, hat sich auch das Problem ihrer Wartung vergrößert. Aus all diesen Gründen haben flache Dateisysteme an Attraktivität verloren, und Datenbanken haben sie in praktisch allen Anwendungsbereichen ersetzt.

Datenverwaltung mit einfachen Programmen

Der Hauptvorteil von Datenbanksystemen besteht darin, dass die Metadaten auf der Datenseite des Systems und nicht im Programm gespeichert werden. Das Programm muss nichts über die Details wissen, wie die Daten gespeichert sind. Das Programm stellt *logische* Abfragen, um Dateien zu erhalten, und das DBMS übersetzt diese logischen Abfragen in Befehle, die an die physische Speicherhardware weitergeleitet werden, um die angeforderte Operation durchzuführen. (Das bedeutet, die *logische Abfrage* fragt nach einer bestimmten Information, gibt aber nicht an, wo diese auf der Festplatte zu finden ist, beispielsweise in Stapeln, Spuren, Sektoren oder Bytes.) Hier die Vorteile dieser Organisation:

- ✔ Da Anwendungsprogramme nur wissen müssen, mit welchen Daten sie arbeiten wollen, und nicht, wo diese Daten genau abgelegt sind, sind sie nicht betroffen, wenn sich die physischen Details des Speicherorts der Daten ändern.

- ✔ Die Übertragbarkeit zwischen verschiedenen Plattformen ist einfach, selbst wenn diese sehr unterschiedlich sind, solange das DBMS, das auf der ersten Plattform verwendet wird, auch auf der zweiten verfügbar ist. Im Allgemeinen müssen Sie die Programme überhaupt nicht ändern, um sie an verschiedene Plattformen anzupassen.

Was sind die Nachteile? Unter anderem:

- ✔ Wenn ein Datenbankmanagementsystem zwischen das Anwendungsprogramm und die Daten geschaltet wird, verlangsamen sich die Operationen mit diesen Daten. Dies ist jedoch nicht mehr das Problem, das es früher war. Fortschritte, wie beispielsweise die Verwendung von Hochgeschwindigkeits-Cache-Speichern, haben dieses Problem erheblich entschärft.
- ✔ Datenbanken benötigen mehr Speicherplatz auf der Festplatte als die gleiche Datenmenge in einem flachen Dateisystem. Dies ist darauf zurückzuführen, dass zusammen mit den Daten auch Metadaten gespeichert werden. Die Metadaten enthalten Informationen darüber, wie die Daten gespeichert sind, sodass die Anwendungsprogramme sie nicht einbeziehen müssen.

Welche Art von Organisation ist besser?

Ich wette, Sie kennen meine Antwort bereits. Sie haben wahrscheinlich recht, aber ganz so einfach ist es nicht. Es gibt nicht die eine richtige Antwort, die für alle Situationen gilt. In den Anfängen der elektronischen Datenverarbeitung waren flache Dateisysteme die einzige praktikable Lösung. Um eine vernünftige Berechnung zeitnah und wirtschaftlich durchführen zu können, musste man das Verfahren verwenden, das am schnellsten war und den geringsten Speicherplatz benötigte. Da immer mehr Anwendungssoftware für diese Systeme entwickelt wurde, waren die Unternehmen, die diese Systeme besaßen, immer stärker an das gebunden, was sie hatten. Um auf ein moderneres Datenbanksystem umzusteigen, mussten alle Anwendungen von Grund auf neu geschrieben und alle Daten neu organisiert werden – eine gewaltige Aufgabe. Infolgedessen gibt es immer noch alte Flat-File-Systeme, die weiterbestehen, weil ein Umstieg auf eine modernere Technologie nicht machbar ist, sowohl wirtschaftlich als auch zeitlich.

Datenbanken, Abfragen und Datenbankanwendungen

Wie stehen die Chancen, dass eine Person tatsächlich eine Nadel im Heuhaufen findet? Nicht sehr gut. Die sprichwörtliche Nadel zu finden, ist deshalb so schwer, weil der Heuhaufen ein zufälliger Haufen Heu ist, mit einzelnen Halmen, die in alle Richtungen zeigen, und einer Nadel, die sich an einem zufälligen Ort zwischen all dem Heu befindet.

Ein flaches Dateisystem ist nicht wirklich mit einem Heuhaufen vergleichbar, aber es fehlt ihm an Struktur, und um einen bestimmten Datensatz in einer solchen Datei zu finden, müssen Sie Werkzeuge verwenden, die außerhalb der Datei selbst liegen. Das ist so, als würde man einen starken Magneten auf den Heuhaufen anwenden, um die Nadel zu finden.

Daten nützlich machen

Damit eine Datensammlung nützlich ist, müssen Sie die gewünschten Daten einfach und schnell abrufen können, ohne sich durch den Rest der Daten bewegen zu müssen. Eine Möglichkeit, dies zu erreichen, besteht darin, die Daten in einer logischen Struktur zu speichern. Flache Dateien haben keine großartige Struktur, Datenbanken dagegen schon. Historisch gesehen wurden das hierarchische Datenbankmodell und das Netzwerk-Datenbankmodell vor dem relationalen Modell entwickelt. Jedes dieser Modelle organisiert die Daten auf eine andere Weise, aber alle drei führen zu einem strukturierten Ergebnis. Aus diesem Grund wurden ab den 1970er-Jahren alle neuen Entwicklungsprojekte hauptsächlich unter Verwendung eines der drei oben genannten Datenbankmodelle durchgeführt: hierarchisch, netzwerkartig oder relational. (Ich gehe auf jedes dieser Datenbankmodelle später in diesem Kapitel im Abschnitt »Die konkurrierender Datenbankmodelle« näher ein.)

Abrufen der gewünschten Daten – und nur der gewünschten Daten

Von allen Operationen, die Menschen mit einer Datensammlung durchführen, ist der Abruf bestimmter Elemente aus der Sammlung die wichtigste. Dies liegt daran, dass Abrufe häufiger durchgeführt werden als jede andere Operation. Die Dateneingabe wird nur einmal durchgeführt. Änderungen an bestehenden Daten werden relativ selten vorgenommen, und Daten werden nur einmal gelöscht. Abrufe hingegen werden häufig durchgeführt, und dieselben Datenelemente können viele Male abgerufen werden. Wenn man also etwas optimieren sollte, dann den Datenabruf. Moderne Datenbankmanagementsysteme unternehmen daher große Anstrengungen, um Abfragen schnell zu gestalten.

Abrufe werden durch Abfragen durchgeführt. Ein modernes Datenbankmanagementsystem analysiert eine Abfrage, die ihm vorgelegt wird, und entscheidet, wie sie am besten ausgeführt werden kann. In der Regel gibt es mehrere Möglichkeiten, eine Abfrage auszuführen, einige davon viel schneller als andere. Ein gutes DBMS wählt stets einen möglichst optimalen Ausführungsplan aus. Natürlich ist es hilfreich, wenn die Abfrage von vornherein optimal formuliert ist. (Ich bespreche Optimierungsstrategien ausführlich in Teil VII, der sich mit Datenbank-Tuning befasst.)

Das erste Datenbanksystem

Das erste echte Datenbanksystem wurde von IBM in den 1960er-Jahren zur Unterstützung des Apollo-Mondlandungsprogramms der NASA entwickelt. Die Anzahl der Komponenten der Saturn V-Trägerrakete, für die Apollo-Befehls- und Servicemodule und die Mondlandefähre überstieg bei Weitem alles, was bis dahin gebaut worden war. Es musste gründlicher getestet werden als alles andere zuvor, denn jedes Bauteil musste den Unbilden einer Umgebung standhalten, die feindlicher und unnachgiebiger war als jede Umgebung, in der Menschen je zu arbeiten versucht hatten. Flache Dateisysteme kamen nicht in Frage. Die Lösung von IBM, die später in ein kommerzielles

Datenbankprodukt mit dem Namen IMS (Information Management System) umgewandelt wurde, zeichnete jede einzelne Komponente sowie ihre gesamte Historie auf.

Als der Hauptsauerstofftank der gescheiterten Apollo 13 auf dem Weg zum Mond riss, arbeiteten die Ingenieure fieberhaft an einem Plan, um das Leben der drei Astronauten auf dem Weg zum Mond zu retten. Sie waren erfolgreich und konnten den Astronauten einen funktionierenden Plan übermitteln.

Nachdem die Besatzung sicher zur Erde zurückgekehrt war, ergab eine Abfrage der IMS-Aufzeichnungen über den ausgefallenen Sauerstofftank, dass er irgendwo zwischen seiner Herstellung und dem Einbau in Apollo 13 zu Boden gefallen war. Die Ingenieure testeten erneut, ob er dem Druck standhalten würde, den er während der Mission aushalten musste, und legten ihn nach bestandenem Test wieder auf Lager. Es stellte sich jedoch heraus, dass der Test einen versteckten Schaden am Tank nicht aufgedeckt hatte, und die NASA hätte diesen Sauerstofftank bei der Apollo-13-Mission nicht verwenden dürfen. Die im IMS gespeicherte Historie zeigt, dass das Bestehen eines Drucktests nicht ausreicht, um zu gewährleisten, dass ein zu Boden gefallener Tank unbeschädigt ist. Bei den nachfolgenden Apollo-Missionen wurde dies berücksichtigt.

Konkurrierende Datenbankmodelle

Ein *Datenbankmodell* beschreibt einfach die Art und Weise, Datenelemente innerhalb einer Datenbank zu organisieren. In diesem Abschnitt stelle ich Ihnen die drei Datenbankmodelle vor, die als erste auf der Bildfläche erschienen sind:

- ✔ **Hierarchisch:** Organisiert Daten in Ebenen, wobei jede Ebene eine einzelne Datenkategorie enthält, und zwischen den Ebenen Eltern-Kind-Beziehungen hergestellt werden.
- ✔ **Netzwerk:** Organisiert Daten auf eine Weise, die einen Großteil der Redundanz des hierarchischen Modells vermeidet.
- ✔ **Relational:** Organisiert Daten in einer strukturierten Sammlung von zweidimensionalen Tabellen.

Nach der Einführung des hierarchischen, des Netzwerk- und des relationalen Modells haben Informatiker weitere Datenbankmodelle entwickelt, die sich in einigen Anwendungskategorien als nützlich erwiesen haben. Auf einige dieser Modelle und ihre Anwendungsbereiche werde ich später in diesem Kapitel kurz eingehen. Für allgemeine Geschäftsanwendungen wurden jedoch hauptsächlich die hierarchischen, netzwerkbasierten und relationalen Modelle verwendet.

Ein Blick auf den historischen Hintergrund der konkurrierenden Modelle

Das erste funktionierende Datenbanksystem wurde von IBM entwickelt und ging am 14. August 1968 bei einem Apollo-Vertragspartner in Betrieb. (Lesen Sie die ganze Geschichte im Einschub »Das erste Datenbanksystem« in diesem Kapitel.) Es wurde als IMS (Information Management

System) bezeichnet und ist (erstaunlicherweise) auch heute noch, über 50 Jahre später, im Einsatz, da IBM es zur Unterstützung seiner Kunden kontinuierlich weiterentwickelt hat.

Wenn Sie sich für ein Datenbankmanagementsystem entscheiden, sollten Sie in Erwägung ziehen, es von einem Anbieter zu kaufen, der es so lange unterstützen wird, wie Sie es nutzen wollen. IBM hat sich als ein solcher Anbieter erwiesen, aber natürlich gibt es auch andere.

IMS ist ein Beispiel für ein hierarchisches Datenbankprodukt. Etwa ein Jahr nach der Einführung von IMS wurde das Netzwerk-Datenbankmodell von einem Industrieausschuss beschrieben. Etwa ein Jahr später schlug Dr. Edgar F. »Ted« Codd, ebenfalls von IBM, das relationale Modell vor. Innerhalb weniger Jahre entstanden so die drei Modelle, die den Datenbankmarkt jahrzehntelang beherrschen sollten.

Es hat einige Jahre gedauert, bis das objektorientierte Datenbankmodell auf den Markt kam und sich als eine Alternative präsentierte, die einige der Mängel des relationalen Modells beheben sollte. Das *objektorientierte Datenbankmodell* ermöglicht die Speicherung von Datentypen, die sich nicht ohne Weiteres in die von relationalen Datenbanken verwendeten Kategorien einordnen lassen. Obwohl sie in einigen Anwendungen Vorteile bieten, haben objektorientierte Datenbanken keinen großen Marktanteil erobert. Das *objektrelationale Modell* ist eine Verschmelzung des relationalen und des objektorientierten Modells und wurde entwickelt, um die Stärken beider Modelle zu nutzen, während ihre größten Schwächen beiseitegelassen werden. Heute gibt es das sogenannte NoSQL-Modell, bei dem Daten in Form von Dokumenten statt in Tabellen gespeichert werden. Das bekannteste NoSQL-Modell ist das Datenbanksystem MongoDB. Da NoSQL Daten als Dokumente speichert, ist es vor allem auf die Arbeit mit nicht streng strukturierten Daten ausgelegt. Da es kein SQL verwendet, werde ich es in diesem Buch nicht behandeln.

Das hierarchische Datenbankmodell

Beim *hierarchischen Datenbankmodell* werden die Daten in Ebenen organisiert, wobei jede Ebene eine einzelne Datenkategorie enthält, und zwischen den Ebenen Eltern-Kind-Beziehungen bestehen. Jedes übergeordnete Element kann mehrere untergeordnete Elemente haben, aber jedes untergeordnete Element kann nur ein übergeordnetes Element haben. Mathematiker nennen dies eine *baumartige* Organisation, weil die Beziehungen wie ein Baum mit einem Stamm organisiert sind, der sich in Äste verzweigt, die sich wiederum in kleinere Äste verzweigen. Alle Beziehungen in einer hierarchischen Datenbank sind also entweder 1:1 oder 1:*n*. *N*:*n*-Beziehungen werden nicht verwendet. (Mehr zu dieser Art von Beziehungen in Kürze.)

Eine hierarchische Datenbank ist beispielsweise gut geeignet für eine Auflistung aller Teile, die in ein fertiges Produkt einfließen – auch als *Stückliste* bezeichnet. Eine ganze Maschine kann aus Baugruppen bestehen, die wiederum aus Unterbaugruppen bestehen, und so weiter, bis hin zu einzelnen Komponenten. Ein Beispiel für eine solche Anwendung ist die mächtige Saturn-V-Mondrakete, die in den späten 1960er- und frühen 1970er- Jahren amerikanische Astronauten zum Mond schickte. Abbildung 1.1 zeigt ein hierarchisches Diagramm der Hauptkomponenten der Saturn V.

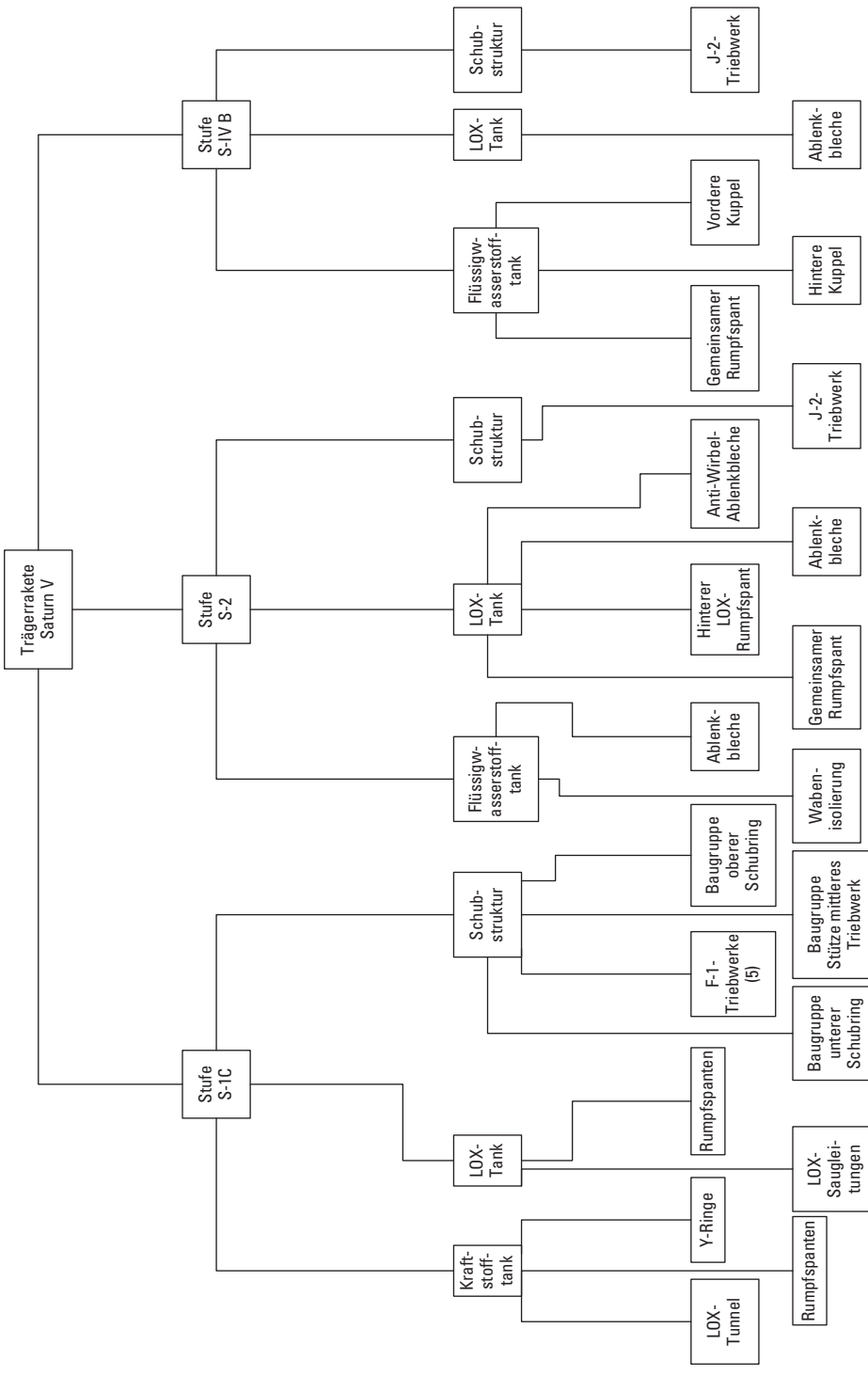

Abbildung 1.1: Ein hierarchisches Modell der Mondrakete Saturn V.

Zwischen Objekten in einer Datenbank können drei Beziehungen bestehen:

- **1:1-Beziehung:** Ein Objekt des ersten Typs ist mit einem und nur mit einem Objekt des zweiten Typs verbunden. In Abbildung 1.1 gibt mehrere Beispiele für 1:1-Beziehungen. Eines davon ist die Beziehung zwischen dem LOX-Tank der Stufe S-2 und dem hinteren LOX-Rumpfspant. Jeder LOX-Tank hat einen einzigen hinteren LOX-Rumpfspant, und jedes hintere LOX-Rumpfspant gehört zu einem einzigen LOX-Tank.
- **1:n-Beziehung:** Ein Objekt des ersten Typs ist mit mehreren Objekten des zweiten Typs verbunden. In der Stufe S-1C der Saturn V enthält die Schubstruktur fünf F-1-Triebwerke, aber jedes Triebwerk gehört nur zu einer einzigen Schubstruktur.
- **n:n-Beziehung:** Mehrere Objekte des ersten Typs sind mit mehreren Objekten des zweiten Typs verbunden. Diese Art von Beziehung wird von einer hierarchischen Datenbank nicht sauber gehandhabt. Versuche, dies zu realisieren, sind oft unübersichtlich. Ein Beispiel sind Sechskantschrauben mit zwei Zoll Durchmesser. Diese Schrauben werden nicht als eindeutig identifizierbar betrachtet, und jede dieser Schrauben ist mit jeder anderen austauschbar. In einer Baugruppe können mehrere Schrauben verwendet werden, und eine Schraube kann in mehreren verschiedenen Baugruppen eingesetzt werden.

Eine große Stärke des hierarchischen Modells ist seine hohe Leistungsfähigkeit. Da die Beziehungen zwischen den Entitäten einfach und direkt sind, können Abrufe, die so eingerichtet sind, dass sie die Struktur der Daten nutzen, sehr schnell sein. Abrufe, die die Struktur der Daten nicht nutzen, sind dagegen langsam und können manchmal gar nicht durchgeführt werden. Es ist schwierig, die Struktur einer hierarchischen Datenbank zu ändern, um neue Anforderungen zu erfüllen. Diese strukturelle Starrheit ist die größte Schwäche des hierarchischen Modells. Ein weiteres Problem des hierarchischen Modells ist die Tatsache, dass es strukturell sehr viel Redundanz erfordert, wie mein nächstes Beispiel deutlich macht.

Bleiben wir realistisch: Nicht viele Unternehmen entwickeln heute Raketen, die Nutzlasten zum Mond befördern Das hierarchische Modell kann jedoch auch für allgemeinere Aufgaben angewandt werden, zum Beispiel für die Verfolgung von Verkaufstransaktionen in einem Einzelhandelsunternehmen. Als Beispiel verwende ich einige Verkaufstransaktionsdaten von Gentoo Joyces fiktivem Online-Shop für Pinguin-Sammlerstücke. Er akzeptiert PayPal, MasterCard, Visa und Überweisungen und verkauft verschiedene Artikel mit Darstellungen von Pinguinen bestimmter Arten – Eselspinguine, Kinnstreifpinguine und Adeliepinguine.

Wie in Abbildung 1.2 dargestellt, tauchen Kunden, die mehrere Einkäufe getätigt haben, mehrfach in der Datenbank auf. Sie sehen zum Beispiel, dass Lynne mit PayPal, MasterCard und Visa eingekauft hat. Da es sich um eine hierarchische Datenbank handelt, tauchen Lynnes Informationen mehrfach auf, ebenso wie die Informationen für jeden Kunden, der mehr als einmal eingekauft hat. Auch die Produktinformationen werden mehrfach angezeigt.

Diese Organisation ist tatsächlich komplexer als in Abbildung 1.2 dargestellt. Zusätzliche »Bäume« würden die Details über jeden Kunden und jedes Produkt enthalten. Diese doppelten Daten sind eine Verschwendung von Speicherplatz, da eine Kopie der Kundendaten ausreicht, ebenso wie eine Kopie der Produktdaten.

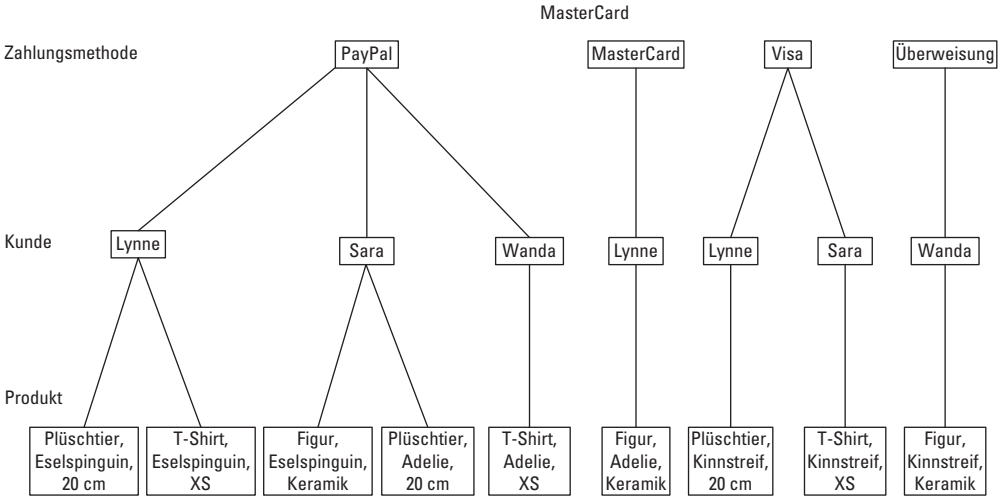

Abbildung 1.2: Ein hierarchisches Modell einer Verkaufsdatenbank für ein Einzelhandelsunternehmen.

Vielleicht noch ungünstiger als die Platzverschwendung durch redundante Daten ist die Möglichkeit der Datenbeschädigung. Wann immer mehrere Kopien derselben Daten in einer Datenbank vorhanden sind, besteht die Möglichkeit von Änderungsanomalien. Eine *Änderungsanomalie* ist eine Inkonsistenz in den Daten, nachdem eine Änderung vorgenommen wurde. Angenommen, Sie wollen einen Kunden löschen, der nicht mehr bei Ihnen kauft. Wenn mehrere Kopien der Daten dieses Kunden existieren, müssen Sie alle finden und löschen, um die Datenintegrität zu wahren. Nehmen wir an, Sie wollen nur die Adressdaten eines Kunden aktualisieren. Wenn mehrere Kopien der Kundendaten vorhanden sind, müssen Sie alle finden und auf exakt dieselbe Weise ändern, um die Datenintegrität zu wahren. Dies kann ein zeitaufwendiger und fehleranfälliger Vorgang sein.

Das Netzwerk-Datenbankmodell

Das Netzwerkmodell, das 1969 als Nachfolger des hierarchischen Modells auf den Markt kam, ist fast das genaue Gegenteil des hierarchischen Modells. In dem Bestreben, die Redundanz des hierarchischen Modells zu vermeiden, ohne allzu große Leistungseinbußen hinnehmen zu müssen, entschieden sich die Entwickler des *Netzwerkmodells* für eine Architektur, bei der die Elemente nicht dupliziert werden, sondern die Anzahl der mit einigen Elementen verbundenen Beziehungen erhöht wird. Abbildung 1.3 zeigt diese Architektur für die gleichen Daten, die in Abbildung 1.2 gezeigt wurden.

Wie Sie in Abbildung 1.3 sehen, weist das Netzwerkmodell nicht die Baumstruktur mit dem Verlauf in eine Richtung auf, die für das hierarchische Modell charakteristisch ist. Es zeigt sehr deutlich, dass beispielsweise Lynne mehrere Produkte gekauft hat, aber auch, dass sie auf mehrere Arten bezahlt hat. In diesem Modell gibt es nur eine Instanz von Lynne, im Vergleich zu mehreren Instanzen im hierarchischen Modell. Es gibt jedoch sieben Beziehungen, die mit dieser einen Instanz von Lynne verbunden sind, während es im hierarchischen Modell nicht mehr als drei Beziehungen gibt, die mit einer Instanz von Lynne verbunden sind.

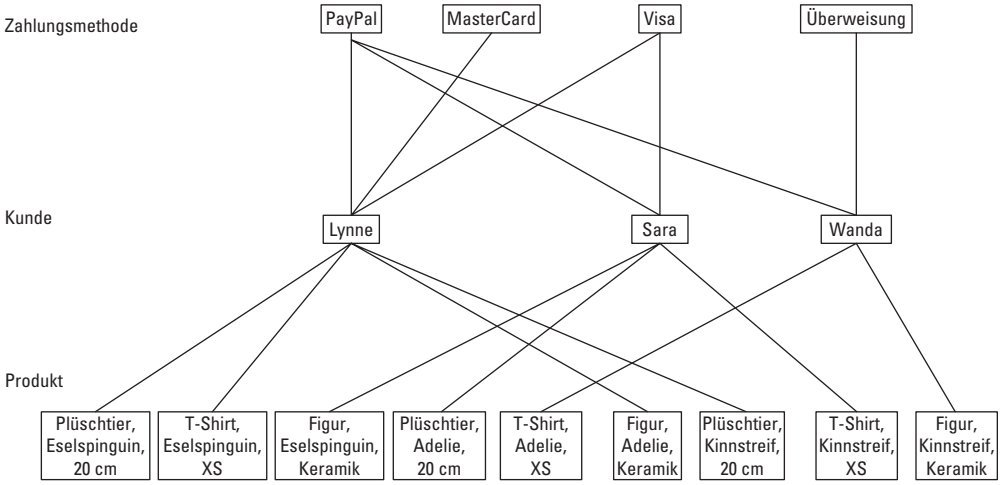

Abbildung 1.3: Ein Netzwerkmodell für Transaktionen in einem Online-Geschäft.

Das Netzwerkmodell beseitigt Redundanzen, allerdings auf Kosten komplizierterer Beziehungen. Dieses Modell kann für einige Arten von Datenspeicheraufgaben besser geeignet sein als das hierarchische Modell, für andere jedoch schlechter. Keines der beiden Modelle ist dem anderen durchweg überlegen.

Das relationale Datenbankmodell

1970 veröffentlichte Edgar Codd von IBM einen Artikel, in dem er das *relationale* Datenbankmodell vorstellte. Anfänglich wurde es von den Datenbankexperten kaum beachtet. Es hatte eindeutig einen Vorteil gegenüber dem hierarchischen Modell, da die Datenredundanz minimal war, und es hatte einen Vorteil gegenüber dem Netzwerkmodell mit seinen relativ einfachen Beziehungen. Allerdings hatte es einen als fatal empfundenen Makel. Aufgrund der Komplexität der dafür erforderlichen relationalen Datenbank-Engine würde jede Implementierung viel langsamer sein als eine vergleichbare Implementierung des hierarchischen oder des Netzwerkmodells. Daher dauerte es fast zehn Jahre, bis die erste Implementierung des relationalen Datenbankmodells auf den Markt kam.

Das mooresche Gesetz hatte die relationale Datenbanktechnologie endlich realisierbar gemacht. (1965 stellte Gordon Moore, einer der Gründer von Intel, fest, dass die Kosten für Computer-Speicherchips etwa alle zwei Jahre um die Hälfte sanken. Er sagte voraus, dass sich dieser Trend fortsetzen würde. Nach über 50 Jahren ist dieser Trend immer noch ungebrochen, und Moores Vorhersage wurde als empirisches Gesetz festgeschrieben.)

IBM lieferte 1978 ein in das Betriebssystem der Serverplattform System 38 integriertes relationales DBMS (RDBMS), und Relational Software, Inc. lieferte 1979 die erste Version von Oracle – dem Urgestein aller eigenständigen relationalen Datenbankmanagementsysteme.

Definieren, was eine Datenbank relational macht

Die ursprüngliche Definition einer relationalen Datenbank besagt, dass sie aus zweidimensionalen Tabellen mit Zeilen und Spalten bestehen muss, wobei die Zelle am Schnittpunkt einer Zeile und einer Spalte einen atomaren Wert enthält (wobei *atomar* bedeutet, dass er nicht in Teilwerte teilbar ist). Diese Definition wird üblicherweise so formuliert, dass eine Tabelle einer relationalen Datenbank keine *sich wiederholenden Gruppen* enthalten darf. Die Definition besagt auch, dass jede Zeile in einer Tabelle eindeutig identifizierbar sein muss. Man kann auch sagen, dass jede Tabelle in einer relationalen Datenbank einen *Primärschlüssel* haben muss, der eine Zeile in einer Datenbanktabelle eindeutig identifiziert. Abbildung 1.4 zeigt die Struktur einer Datenbank eines Online-Shops, die nach dem relationalen Modell aufgebaut ist.

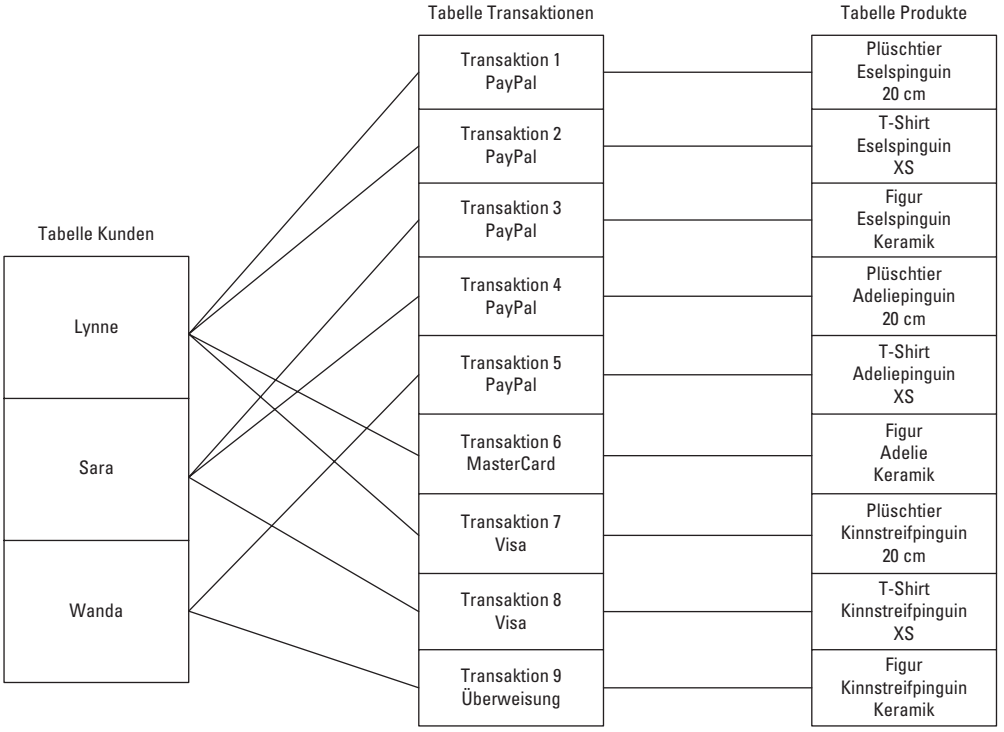

Abbildung 1.4: Ein relationales Modell von Transaktionen in einem Online-Geschäft.

Mit dem relationalen Modell wurde die Idee eingeführt, Datenbankelemente in zweidimensionalen Tabellen zu speichern. In dem in Abbildung 1.4 gezeigten Beispiel enthält die Tabelle »Kunden« alle Informationen über jeden Kunden, die Tabelle »Produkte« enthält alle Informationen über jedes Produkt, und die Tabelle »Transaktionen« enthält alle Informationen über den Kauf eines Produkts durch einen Kunden. Die Idee, eng verwandte Dinge von weiter entfernten Dingen zu trennen, indem die Dinge in Tabellen aufgeteilt werden, war einer der Hauptfaktoren, die das relationale Modell von den hierarchischen und Netzwerkmodellen unterscheiden.

Schutz der Definition von relationalen Datenbanken mit den Coddsch'en Regeln

Als das relationale Modell an Popularität gewann, begannen Anbieter von Datenbankprodukten, die nicht wirklich relational waren, ihre Produkte als relationale Datenbankmanagementsysteme zu bewerben. Um der Verwässerung seines Modells entgegenzuwirken, formulierte Codd zwölf Regeln, die als Kriterien dafür dienten, ob ein Datenbankprodukt tatsächlich relational war. Codds Idee war, dass eine Datenbank alle zwölf Kriterien erfüllen muss, um als relational zu gelten.

Die Regeln von Codd sind so streng, dass es auch heute noch kein DBMS auf dem Markt gibt, das sie vollständig erfüllt. Sie haben jedoch ein gutes Ziel vorgegeben, nach dem die Datenbankanbieter streben.

Hier die zwölf Regeln von Codd:

1. **Die Informationsregel:** Daten können nur auf eine Weise dargestellt werden, nämlich als Werte an Spaltenpositionen innerhalb von Zeilen einer Tabelle.

2. **Die Regel des garantierten Zugriffs:** Jeder Wert in einer Datenbank muss durch Angabe eines Tabellennamens, eines Spaltennamens und einer Zeile zugänglich sein. Die Zeile wird durch den Wert des Primärschlüssels angegeben.

3. **Systematische Behandlung von Nullwerten:** Fehlende Daten unterscheiden sich von bestimmten Werten, wie zum Beispiel null oder einer leeren Zeichenfolge.

4. **Relationaler Online-Katalog:** Autorisierte Benutzer müssen in der Lage sein, auf die Struktur der Datenbank (ihren *Katalog*) mit der gleichen Abfragesprache zuzugreifen, die sie für den Zugriff auf die Daten der Datenbank verwenden.

5. **Umfassende Datensprache:** Das System muss mindestens eine relationale Sprache unterstützen, die sowohl interaktiv als auch innerhalb von Anwendungsprogrammen verwendet werden kann und die Funktionen zur Datendefinition, Datenmanipulation und Datenkontrolle unterstützt. Heute ist diese eine Sprache SQL.

6. **Aktualisierung der Ansichten:** Alle Ansichten, die theoretisch aktualisierbar sind, müssen vom System aktualisiert werden können.

7. **Das System muss gleichzeitige Einfüge-, Aktualisierungs- und Löschoperationen unterstützen:** Dies bedeutet, dass das System in der Lage sein muss, Einfügungen, Aktualisierungen und Löschungen mehrerer Zeilen in jeweils einem einzigen Vorgang durchzuführen.

8. **Unabhängigkeit von physischen Daten:** Änderungen an der Art und Weise, wie Daten gespeichert werden, dürfen die Anwendung nicht beeinträchtigen.

9. **Logische Datenunabhängigkeit:** Änderungen an den Tabellen dürfen keine Auswirkungen auf die Anwendung haben. So darf beispielsweise das Hinzufügen neuer Spalten zu einer Tabelle eine Anwendung, die auf die ursprünglichen Zeilen zugreift, nicht unbrauchbar machen

10. **Integritätsunabhängigkeit:** Integritätseinschränkungen müssen unabhängig von den Anwendungsprogrammen festgelegt und im Katalog gespeichert werden. (Ich spreche in Kapitel 9 viel über Integrität.)

11. **Unabhängigkeit von der Verteilung:** Die Verteilung von Teilen der Datenbank auf verschiedene Standorte darf die Funktionsweise der Anwendungen nicht verändern.

12. **Nichtunterwanderungsregel:** Wenn das System eine Schnittstelle für Datensätze zur Verfügung stellt, darf es nicht möglich sein, diese zu verwenden, um die relationalen Sicherheits- oder Integritätsbeschränkungen zu umgehen.

Zusätzlich zu den ursprünglichen zwölf Regeln fügte Codd 1990 eine weitere Regel hinzu:

> **Regel null (Rule Zero):** Jedes System, das als relationales Datenbankmanagementsystem beworben wird oder von dem behauptet wird, dass es ein solches ist, muss in der Lage sein, Datenbanken ausschließlich über seine relationalen Funktionen zu verwalten, unabhängig davon, welche zusätzlichen Funktionen das System unterstützt.

Regel null war eine Reaktion auf die Werbestrategie einiger Anbieter von Datenbankprodukten, die behaupteten, ihr Produkt sei ein relationales DBMS, obwohl es in Wirklichkeit nicht vollständig relationale Fähigkeiten besaß.

Die inhärente Flexibilität des relationalen Datenbankmodells

Sie fragen sich vielleicht, warum relationale Datenbanken den Planeten erobert und hierarchische und Netzwerkdatenbanken in Nischen verbannt haben, in denen hauptsächlich Altkunden angesiedelt sind, die diese Datenbanken seit mehr als 40 Jahren verwenden. Dies ist umso erstaunlicher, als die meisten Experten bei der Einführung des relationalen Modells davon ausgingen, dass es weder mit dem hierarchischen noch mit dem Netzwerkmodell konkurrieren könne.

Ein Vorteil des relationalen Modells ist seine Flexibilität. Die Architektur einer relationalen Datenbank ist so beschaffen, dass es viel einfacher ist, eine relationale Datenbank umzustrukturieren als eine hierarchische oder Netzwerkdatenbank. Dies ist ein enormer Vorteil in dynamischen Geschäftsumgebungen, in denen sich die Anforderungen ständig ändern.

Der Grund, warum Datenbankexperten das relationale Modell ursprünglich ablehnten, war die Annahme, dass der zusätzliche Mehraufwand für die relationale Datenbank-Engine jedes Produkt, das auf diesem Modell basierte, so viel langsamer machen würde als hierarchische oder Netzwerk-Datenbanken, dass es nicht wettbewerbsfähig wäre. Im Laufe der Zeit hat das mooresche Gesetz diesen Einwand entkräftet.

Das objektorientierte Datenbankmodell

Objektorientierte Datenbankmanagementsysteme (OODBMS) tauchten erstmals 1980 auf. Sie wurden in erster Linie entwickelt, um nicht textliche, nicht numerische Daten wie grafische Objekte zu verarbeiten. Ein relationales DBMS kann in der Regel mit solchen

sogenannten komplexen Datentypen nicht gut umgehen. Ein OODBMS verwendet dasselbe Datenmodell wie objektorientierte Programmiersprachen wie Java, C++ und C# und funktioniert gut mit diesen Sprachen.

Obwohl objektorientierte Datenbanken für bestimmte Anwendungen besser geeignet sind als relationale Datenbanken, schneiden sie bei den meisten Mainstream-Anwendungen nicht so gut ab und haben die Vorherrschaft der relationalen Produkte nicht wesentlich beeinträchtigt. Aus diesem Grund werde ich mich nicht weiter zu OODBMS-Produkten äußern.

Das objektrelationale Datenbankmodell

Eine *objektrelationale Datenbank* ist eine relationale Datenbank, die es den Benutzern ermöglicht, neue Datentypen zu erstellen und zu verwenden, die nicht Teil des von SQL bereitgestellten Standardsatzes an Datentypen sind. Die Fähigkeit des Benutzers, neue Typen, sogenannte *benutzerdefinierte Typen,* hinzuzufügen, wurde in die SQL:1999-Spezifikation aufgenommen und ist in den aktuellen Implementierungen von IBMs DB2, Oracle und Microsoft SQL Server verfügbar.

Die derzeitigen relationalen Datenbankmanagementsysteme sind eigentlich objektrelationale Datenbankmanagementsysteme und keine rein relationalen Datenbankmanagementsysteme.

Das nicht relationale NoSQL-Modell

Im Gegensatz zum relationalen Modell findet das nicht relationale Modell immer mehr Anhänger, vor allem im Bereich des *Cloud Computings,* bei dem die Datenbanken nicht auf dem lokalen Computer oder im lokalen Netz verwaltet werden, sondern irgendwo im Internet gespeichert sind. Dieses Modell, das sogenannte NoSQL-Modell, eignet sich besonders für große Systeme, die aus Server-Clustern bestehen und auf die über das World Wide Web zugegriffen wird. CouchDB und MongoDB sind Beispiele für DBMS-Produkte, die diesem Modell folgen. Das NoSQL-Modell ist dokumentenbasiert und speichert alle zusammenhängenden Daten in demselben Dokument. Da alle zusammenhängenden Daten am selben Ort gespeichert werden, können Abfragen für große Datenmengen schneller erfolgen als in herkömmlichen relationalen Datenbanken. Das NoSQL-Modell ist nicht konkurrenzfähig mit dem SQL-basierten relationalen Modell für traditionelle Berichtsanwendungen.

Warum das relationale Modell gewonnen hat

Während der 1970er- und bis in die 1980er-Jahre hinein waren hierarchische und netzwerkbasierte Technologien die bevorzugten Datenbanktechnologien für große Unternehmen. Oracle, das erste eigenständige relationale Datenbanksystem, das auf den Markt kam, wurde erst 1979 eingeführt und war zunächst nur begrenzt erfolgreich.

Aus den folgenden Gründen, aber auch aus reiner Trägheit setzten sich relationale Datenbanken zunächst nur langsam durch:

- ✔ **Die ersten Implementierungen von relationalen Datenbankmanagementsystemen waren sehr langsam.** Das lag daran, dass sie mehr Berechnungen durchführen mussten als andere Datenbanksysteme, um dieselbe Operation auszuführen.
- ✔ **Die meisten Unternehmensleiter zögerten, etwas Neues auszuprobieren, wenn sie bereits mit der einen oder anderen älteren Technologie vertraut waren.**
- ✔ **Daten und Anwendungen, die bereits für ein bestehendes Datenbanksystem vorhanden sind, lassen sich nur sehr schwer für die Arbeit mit einem relationalen DBMS konvertieren.** Für die meisten Unternehmen mit einem bestehenden hierarchischen oder Netzwerk-Datenbanksystem wäre eine Umstellung zu kostspielig.
- ✔ **Die Mitarbeiter müssten eine völlig neue Art des Umgangs mit Daten lernen**. Auch das wäre sehr kostspielig.

Doch allmählich begannen sich die Dinge zu ändern.

Datenbanken, die nach dem hierarchischen oder Netzwerkmodell strukturiert sind, waren zwar sehr leistungsfähig, aber schwer zu pflegen. Strukturelle Änderungen an einer Datenbank erforderten ein hohes Maß an Fachwissen und viel Zeit. In vielen Unternehmen wuchsen die Rückstände bei den Änderungsanträgen von Monaten auf Jahre an. Abteilungsleiter begannen, ihre Arbeit auf PCs zu verlagern, anstatt sich an die IT-Abteilung des Unternehmens zu wenden, um eine Änderung der Datenbank zu beantragen. IT-Manager, die befürchteten, dass ihre Macht im Unternehmen schwindet, wagten den drastischen Schritt, die relationale Technologie in Betracht zu ziehen.

In der Zwischenzeit veränderte das mooresche Gesetz unaufhaltsam die Leistungssituation. Im Jahr 1965 stellte Gordon Moore von Intel fest, dass sich der Preis eines Bits in einem Halbleiterspeicher etwa alle 18 Monate bis zwei Jahre halbieren würde, und er sagte voraus, dass sich dieser exponentielle Trend fortsetzen würde. Eine logische Folge dieses Gesetzes ist, dass sich die Leistung von Prozessoren mit integrierten Schaltkreisen bei gleichen Kosten alle 18 bis 24 Monate verdoppeln würde. Beides hat sich seit mehr als 50 Jahren bewahrheitet, auch wenn das Ende des Trends in Sicht ist. Darüber hinaus haben sich auch die Kapazitäten und die Leistung von Festplattenspeichern exponentiell verbessert, parallel zu den Verbesserungen bei den Halbleiterchips.

Die Leistungsverbesserungen bei Prozessoren, Speichern und Festplatten führten zu einer drastischen Steigerung der Leistung von relationalen Datenbanksystemen und machten sie konkurrenzfähiger gegenüber hierarchischen und Netzwerksystemen. Wenn man diese verbesserte Leistung zusätzlich zum Vorteil der strukturellen Flexibilität der relationalen Architektur betrachtete, begannen relationale Datenbanksysteme, sehr viel attraktiver zu werden, sogar für große Unternehmen mit großen Investitionen in Altsysteme. In vielen dieser Unternehmen wurden zwar die bestehenden Anwendungen auf den bisherigen Plattformen beibehalten, aber neue Anwendungen und die Datenbanken, die ihre Daten enthielten, wurden mit der neuen relationalen Technologie entwickelt.