Dateiverwaltung ist nicht gleich Datenbank

Alle reden über Dateiverwaltung. Über die vielfältigen Anwendungsmöglichkeiten und über Vor- und Nachteile der verschiedenen Programme. Wenig verbreitet ist jedoch das Wissen darum, was eine Dateiverwaltung eigentlich ist und welche Unterschiede zu Datenbanken bestehen.

Zuerst sei folgendes erklärt: eine Datenbank ist nicht einfach eine Art »bessere Dateiverwaltung«. Zwischen Datenbank und Dateiverwaltung bestehen nicht nur graduelle Unterschiede. Beiden gemeinsam ist die Aufgabe, eine Menge gleichartiger Daten, das heißt Dateien zu verwalten, ob es sich bei diesen Daten nun um Artikel, Schallplatten oder Videotitel handelt. Das Verwalten besteht im sogenannten »Änderungsdienst«, dem Ändern und Löschen von Datensätzen, dem Sortieren von Dateien und der möglichst komfortablen Beantwortung von Anfragen des Benutzers, also dem Selektieren bestimmter Datensätze aus der Gesamtmenge.

Mit einer leistungsfähigen Dateiverwaltung läßt sich nicht nur eine spezifische Datei verwalten, sondern der Benutzer kann beliebig viele verschiedene Dateien aufbauen. Mit einer Datenbank ist es jedoch möglich, mehrere Dateien miteinander zu verknüpfen (siehe Übersicht Tabelle 1).

Ein Beispiel: Stellen Sie sich vor, Sie seien der Inhaber eines Zeitungsvertriebs und seit kurzem stolzer Besitzer eines Computers. Diesen wollen Sie dazu verwenden, das Schreiben von Rechnungen an die Abonnenten zu automatisieren. Sie erwerben eine Datenbank wie zum Beispiel dBase II und bauen zwei Dateien auf: Eine Kundendatei, in der Sie Ihre Abonnenten führen (Kundennummer, Anschrift des Kunden, abonnierte Zeitschrift) und eine Zeitschriftendatei (Zeitschrift, Abonnentenpreis). Bisher verlief das Schreiben von Rechnungen so: Sie suchen in dem Karteikasten »Kunden« den jeweiligen Abonnenten, tragen dessen vollständige Anschrift in die Rechnung ein, merken sich den Namen der abonnierten Zeitschrift, suchen in dem Karteikasten »Zeitschriften« nach der entsprechenden Karteikarte und tragen den Abonnentenpreis in die Rechnung ein. Ein sehr zeitaufwendiger Vorgang, wenn die Dateien groß sind.

Der Datenbank müssen Sie nur angeben: »Suche in der Kundendatei nach dem Kunden xyz, merke dir die abonnierte Zeitschrift und suche in der Datei »Zeitschriften« nach diesem Titel. Die Datenbank verfügt nun über alle Informationen, um eine vollständige Rechnung auszudrucken.

Die gestellte Aufgabe erfordert die Verknüpfung von Kunden- und Zeitschriftendatei. Gute Datenbanken sind sogar in der Lage, mehr als zwei Dateien gleichzeitig zu bearbeiten und somit noch weitaus komplexere Anfragen zu beantworten.

Sollten Sie im Besitz einer Dateiverwaltung sein, müssen Sie auf diese Möglichkeiten der Verknüpfung von Dateien leider verzichten. Sie müssen ebenfalls darauf verzichten, sich mit Hilfe einer Datenbanksprache ein Abfrageprogramm zu schreiben, das genau auf Ihre spezielle Anwendung zugeschnitten ist. Jede Datenbank bietet eine solche Datenbanksprache. Mit Ihrer Hilfe lassen sich immer wiederkehrende Abläufe fest einprogrammieren. In unserem Beispiel könnte ein solches Programm so aussehen: die Datenbank wartet auf die Eingabe der Kundennummer, durchsucht daraufhin selbständig Kunden- und Zeitschriftendatei nach dem vorgegebenen Schema und druckt die Rechnung aus; anschließend wird auf die Eingabe der nächsten Kundennummer gewartet. Die Arbeit mit einer Dateiverwaltung hingegen geschieht im Dialog mit dem System. Der Benutzer muß sich auch bei immer wiederkehrenden Abläufen durch eine Vielzahl von Menüs hindurcharbeiten, um eine bestimmte Funktion anzuwählen.

Diese beiden Merkmale einer Datenbank, das Verknüpfen von Dateien, und die feste Programmierung immer gleicher Arbeitsabläufe sind jedoch nur für den kommerziellen Anwender interessant. Der private Benutzer, der seine Adressen oder Schallplatten verwaltet, wird mit einer leistungsfähigen Dateiverwaltung wohl immer zufrieden sein.

Sollten Sie aufgrund der Tatsache, daß Dateiverwaltungen nicht so leistungsfähig wie Datenbanken sind, nun der Ansicht sein, daß die Programmierung einer Dateiverwaltung wohl recht einfach sein müßte, muß ich Sie jedoch enttäuschen. Es gibt unzählige Möglichkeiten, eine Dateiverwaltung zu erstellen, die vor allem von den gewählten Datenstrukturen abhängen; und die Auswahl an Datenstrukturen ist fast unbegrenzt. Es gibt Listenstrukturen, Baumstrukturen, Hashing-Verfahren etc. Die Wahl der geeigneten Datenstrukturen entscheidet über die spätere Leistungsfähigkeit einer Dateiverwaltung und kann im Nachhinein kaum geändert werden. Es ist meist leichter, ein Programm neu zu schreiben, als die verwendeten Datenstrukturen eines fertigen Programms zu ändern, weil Sie sich als nicht leistungsfähig genug erwiesen haben.

Die meist verwendeten Datenstrukturen will ich kurz näher erläutern (einige zum Verständnis nötige

Fähigkeit	Dateiverwaltung	Datenbank
Suchen	ja	ja
Eintragen	ja	ja
Löschen	ja	ja
Ändern	ja	ja
Sortieren	ja	ja
Verknüpfen von Dateien	nein	ja
eigene Programmiersprache	nein	ja

Tabelle 1. Unterschiede zwischen Dateiverwaltung und Datenbank

Begriffe wie »binär«, »Zeiger« etc. sind im Lexikon in dieser Ausgabe erläutert). Sollte dieser Artikel bei Ihnen Appetit auf »mehr« wecken, so möchte ich Sie auf das Buch »Alles über Datenbanken und Dateiverwaltung mit dem C 64«, erschienen bei Data Becker, hinweisen. Kommen wir nun zu den versprochenen Datenstrukturen (Tabelle 2).

Listenstrukturen können Sie sich als eine Aneinanderreihung von Datensätzen vorstellen, vergleichbar mit der Anordnung von Daten in einem Telefonbuch. Diese Aneinanderreihung kann ungeordnet oder geordnet sein (meist alphabetisch). Geordnete Listen haben den Vorteil, daß sie schneller durchsucht werden können als ungeordnete. Sie können die Suche in einer geordneten Liste mit der Suche in einem Telefonbuch vergleichen, bei der Sie wohl kaum auf der ersten Seite mit der Suche nach »Maier« beginnen, sondern wesentlich effizienter suchen, eben (fast) binär. Problematisch an geordneten Listen ist das Eintragen und Löschen von Datensätzen. Um einen neuen Datensatz einzutragen, müssen alle alphabetisch nachfolgenden Datensätze verschoben werden, um Platz zu schaffen. Aus diesem Grund wird die Ordnung bei verketteten Listen nicht durch die Reihenfolge, sondern durch Zeiger hergestellt. Jedes Element der Liste besitzt einen solchen Zeiger, der auf den Ort weist, an dem der (alphabetisch) nachfolgende Datensatz abgespeichert ist. Da die Reihenfolge der Elemente unerheblich ist, kann ein neu einzutragender Datensatz einfach an das Ende der Liste angehängt werden. Er muß allerdings einen Zeiger auf den alphabetisch nachfolgenden Datensatz bekommen, und der vorhergehende Datensatz einen Zeiger auf den gerade neu eingetragenen. Eine verkettete Liste läßt sich leider nicht binär durchsuchen, sie muß sequentiell, also Element für Element, durchsucht werden. Dafür erlaubt sie das schnelle Eintragen und Löschen von Datensätzen. Da sie geordnet ist, kann sich der Benutzer einer solchen Dateiverwaltung jederzeit den alphabetisch nächsten oder vorhergehenden Datensatz zeigen lassen.

Baumstrukturen sind im Grunde genommen nur eine besondere Form von geordneten Listen, bei der die einzelnen Elemente, allerdings durch mehr als einen Zeiger, miteinander verkettet werden. Zum Einfügen eines neuen oder Löschen eines alten Elementes müssen nur wenige Zeiger verändert werden; das Verschieben von Elementen ist nicht nötig. Die Suche nach bestimmten Elementen ist schnell, da der Baum binär durchsucht wird. Aufgrund Ihrer Vorteile gehören Baumstrukturen zu den beliebtesten Datenstrukturen.

Hashing: Das sogenannte »Hash-Verfahren« hat mit den bisher geschilderten Datenstrukturen keinerlei Gemeinsamkeiten. Es beruht darauf, daß mit Hilfe eines geeigneten Algorithmus direkt aus dem abzuspeichernden Datensatz die Adresse gewonnen wird, an dem dieser abgelegt wird. Ein Beispiel: der einfachste denkbare Hash-Algorithmus bestünde darin, die Quersumme der einzelnen ASCII-Werte für ein einzelnes Datensatzfeld, zum Beispiel »maier«, zu bilden. Die ermittelte Zahl (366) könnte nun als Recordnummer oder als Index für einen String verwendet werden (a$(366)="maier;georg;münchen"). Bei der Suche nach einer bestimmten Adresse wird nun dieser Algorithmus auf den Namen angewandt, und der String mit diesem Index oder aber der jeweilige Record eingelesen. Der Vorteil des Hash-Verfahrens liegt in der Geschwindigkeit: in den meisten Fällen ist keinerlei Suche nach bestimmten Daten mehr nötig, sondern es kann direkt auf den gewünschten Datensatz zugegriffen werden. Nachteilig ist, daß die Daten völlig ungeordnet sind.

Diese kurze Vorstellung verschiedener Datenstrukturen sollte vor allem eines zeigen: Die Wahl einer bestimmten Datenstruktur hängt von der jeweiligen Aufgabenstellung ab. Wenn der entscheidende Gesichtspunkt bei der Verwaltung von Daten die Zugriffsgeschwindigkeit ist, empfiehlt sich das Hash-Verfahren. Muß die Datei jedoch unbedingt alphabetisch durchblättert werden können, so bieten sich Listen- oder Baumstrukturen an.

Struktur	Vorteile	Nachteile
ungeordnete Liste	einfacher Änderungsdienst (Blockoperationen nur zum Löschen nötig)	langsame sequentielle Suche, kein geordnetes Blättern
geordnete Liste	schnelle binäre Suche, geordnetes Blättern	aufwendiger Änderungsdienst (Blockoperationen)
verkettete Liste	geordnetes Blättern, einfacher Änderungsdienst (keinerlei Blockoperationen)	langsame sequentielle Suche
Baumstrukturen	schnelle binäre Suche, geordnetes Blättern, einfacher Änderungsdienst (keinerlei Blockoperationen)	–
Hashing	extrem schnelle Suche über Hash-Algorithmus, einfacher Änderungsdienst (keinerlei Blockoperationen)	kein geordnetes Blättern
geordnetes Blättern:	ausgehend von einem bestimmten Datensatz kann (alphabetisch/numerisch) vor- bzw. zurückgeblättert werden	
Blockoperationen:	Verschieben ganzer Datenblöcke, zum Beispiel zum Eintragen eines neuen Datensatzes	

◄ **Tabelle 2. Vor- und Nachteile der wichtigsten Datenstrukturen**

Dateiform	Vorteile	Nachteile	benötigte Zugriffsart
sequentielle Datei	einfache programmtechnische Realisation	Dateigröße von Rechnerspeicherkapazität abhängig	sequentieller Zugriff
index-sequentielle Datei	Dateigröße nur von der Kapazität des externen Speichers abhängig (Floppy, Kassette)	schnelle Suche nur über den Index	direkter Zugriff
Hashing-Datei	Dateigröße nur von der Kapazität des externen Speichers abhängig, extrem schnelle Suche über Hash-Feld	schnelle Suche nur über Hash-Feld, Beschränkung bei der Wahl der Datenstruktur auf Hash-Organisation	direkter Zugriff

▲ **Tabelle 3. Die wichtigsten Dateiformen**

Handicap für Kassettenrecorder

Kurz erwähnt werden sollte noch die Art des Zugriffs auf Daten. Man unterscheidet hauptsächlich den direkten und den sequentiellen Zugriff (Tabelle 3). Mit einem Kassettenrecorder geht es nur sequentiell: Um einen bestimmten Datensatz zu finden, müssen alle in der Datei vorhergehenden Elemente gelesen werden. Damit ist weder die index-sequentielle Datei, bei der über einen Zeiger direkt auf den gewünschten Datensatz zugegriffen werden kann, noch die Hashing-Datei, die ebenfalls direkten Zugriff voraussetzt, möglich.

Bei der Floppy gibt es diese Einschränkungen nicht. Sie hat nur zwei Grenzen; zum einen die Speicherkapazität und zum anderen die Geschwindigkeit, mit der Daten übertragen werden. Und diese beiden Punkte sind es auch, die die Verwendung von Datenbanken auf dem C 64 fraglich machen. Die Verwendung mehrerer Dateien ist sehr speicherintensiv, und die Verknüpfung dieser Dateien erfordert einen häufigen Zugriff auf die Diskette und ist daher auch ein Zeitproblem. Interessant dürften Datenbanken eher beim neuen PC 128 werden. Warten wir's ab.

(Said Baloui/gk)