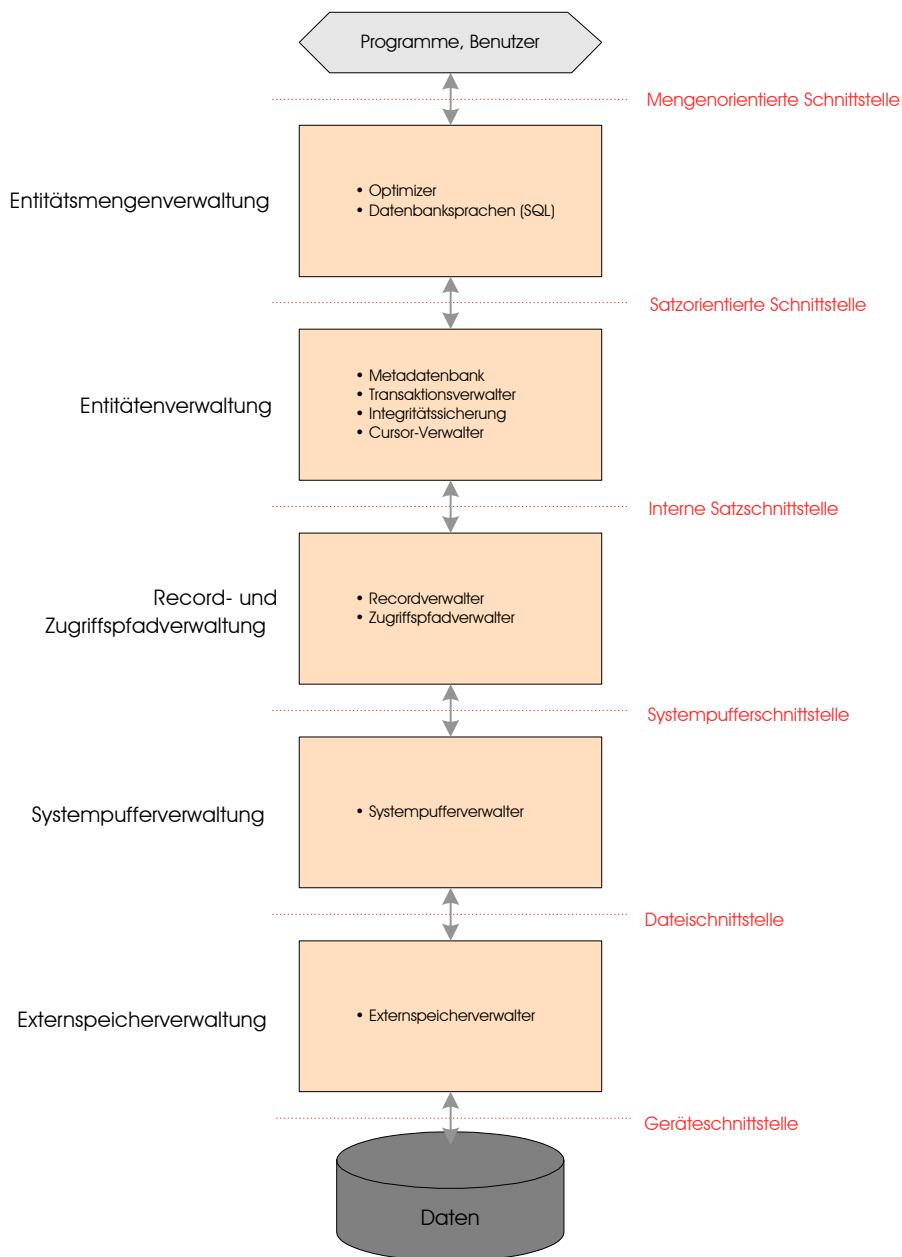


9. Datenbanktechnik, Softwarekomponenten eines Datenbanksystems

In diesem Kapitel muss vorerst ein rudimentäres Verständnis für die Arbeitsweise von Datenbanksystemen vermittelt werden. Nur mit Kenntnis der internen Softwarekomponenten und deren Lösungskonzepten kann ein effizientes und auf das Datenbanksystem ausgerichtetes internes Datenmodell erstellt werden, können Programme mit optimalem Zugriffsverhalten gestaltet werden.

Ein noch so komplexes und leistungsstarkes Datenbanksystem besteht letztendlich nur aus Software. Entsprechend jeder anderen Software lassen sich Datenbanksysteme ebenfalls in Software-Module gliedern. Zwar existiert für bestehende Datenbanksysteme keine allgemeingültige Gliederung in Module, doch lässt sich durch eine verallgemeinerte Gliederung die Arbeitsweise anschaulich erklären. Im Folgenden werden wir ein Schichtenmodell, angelehnt an Senko [Senko73], verwenden, welches das Datenbanksystem in fünf Schichten (Module) gliedert (siehe auch [Härder 83]).



Figur 34: Schichtenmodell eines Datenbanksystems

Jede Schicht verwendet die Funktionen der unteren Schicht, um ihre jeweils spezifischen Funktionen zu realisieren, welche diese Schicht wieder der nächsthöheren Schicht zur Verfügung stellt. Dadurch wird das Datenbanksystem hierarchisch strukturiert. Die von den Schichten den untenliegenden Schichten zur Verfügung gestellten Funktionen werden auch als Schnittstelle bezeichnet.

9.1. Externspeicherverwaltung

Die Software-Komponente Externspeicherverwaltung hat zur Aufgabe, die unterschiedlichen Geräteeigenschaften (z.B. Spurenanzahl, Blockgrösse etc.) der verwendeten Speichermedien der nächsthöheren Komponente (der Systempufferverwaltung) zu verstecken und stattdessen eine dateiorientierte Sichtweise auf die Speichermedien zu gewähren. Diese Schicht ist im Normalfall Teil des Betriebssystems. Diese Schicht stellt in der Dateischnittstelle Funktionen zur Verfügung, welche ein Verarbeiten von Dateien erlauben: Erzeugen, Löschen, Öffnen und Schliessen von Dateien, Lesen und Schreiben eines Dateiblocks.

9.2. Systempufferverwaltung

Die Systempufferverwaltung vermittelt der übergeordneten Schicht den Eindruck, sämtliche Daten seien im Systempuffer des Hauptspeichers gespeichert. Die Systempufferverwaltung übernimmt dabei die Aufgabe, Daten im Systempuffer zur Verfügung zu stellen und geänderte Daten wieder auf die Speichermedien zu übertragen. Die im Systempuffer verarbeiteten Einheiten werden Segmente oder Seiten genannt.

Natürlich haben nicht sämtliche Daten im Systempuffer Platz. Der Systempufferverwalter muss daher für neu angeforderte Segmente entscheiden, welche Segmente auf das Speichermedium ausgelagert werden sollen. Für diesen Entscheid sind eine ganze Reihe von Strategien denkbar. Durch die geeignete Wahl der Ersetzungsstrategie kann die Leistung des Datenbanksystems beträchtlich gesteigert werden. Sinnvollerweise unterstützt der Systempufferverwalter unterschiedliche Segmenttypen, um einen differenzierten Einsatz unterschiedlicher Strategien je Datentyp (Benutzerdaten, DBMS-Daten etc.) zu ermöglichen. Die Konfiguration des Systempufferverwalters (z.B. auch dessen Grösse) kann an dieser Stelle nicht weiter beschrieben werden und es muss daher auf weitere Ausführungen verzichtet werden.

Der Systempufferverwalter selbst kann noch immer als Teil des Betriebssystems realisiert sein. In diesem Fall sollten aber bestimmte Funktionen vom Systempufferverwalter dem Datenbanksystem zur Verfügung gestellt werden, ansonsten können wünschenswerte Funktionen des Datenbanksystems nicht oder nur ineffizient realisiert werden.

9.3. Record- und Zugriffspfadverwaltung

Die in dieser Schicht wahrgenommenen Aufgaben können leicht in zwei Aufgabenbereiche gegliedert werden:

1. Recordverwalter bzw. Satzverwaltung: Der Recordverwalter übernimmt die physische Abspeicherung von Datensätzen mit entsprechenden Funktionen wie Lesen, Einfügen, Modifizieren und Löschen.
2. Zugriffspfadverwaltung: Für eine praktikable Verarbeitung müssen Datensätze mit bestimmten gesuchten Eigenschaften effizient gefunden werden. Hierfür stellt das Datenbanksystem spezielle Zugriffshilfen zur Verfügung. Diese Zugriffshilfen werden allgemein als Zugriffspfade bezeichnet.

9.3.1. Recordverwaltung

Der Recordverwalter hat die Aufgabe, die physischen Datensätze des Datenbanksystems in den Segmenten des Systempuffers abzulegen. Bei der Definition des Datensatzformates werden dessen Felder sowie deren Datenformate festgehalten. Dadurch ist die Gesamtlänge des Datensatzes gegeben. Im günstigen Fall haben pro Segment ein oder auch mehr Datensätze Platz. Ist der Datensatz grösser als

ein einzelnes Segment, muss dieser auf mehrere Segmente verteilt werden, was zu einer deutlichen Verschlechterung der Zugriffszeiten führt.

Zur besseren Nutzung des verfügbaren Speichermediums drängt sich die Komprimierung (allenfalls mit gleichzeitiger Chiffrierung) der physischen Datensätze auf. Der erhöhte Rechenaufwand hält sich dabei die Waage mit den Zeitersparnissen beim reduzierten Ein- und Ausgabeaufwand. Der Datensatz hat dann keine feste Länge mehr, sondern diese variiert je nach Dateninhalt. Messungen zeigen, dass durch Komprimierung der physische Speicherbedarf für die Daten im Durchschnitt auf ca. 50% der ursprünglichen Grösse reduziert werden kann.


Dadurch wäre es dem Datenbanksystem grundsätzlich auch möglich, die Feldlänge der einzelnen Datensätze variabel zu gestalten. Die meisten Datenbanksysteme bieten diese Möglichkeit nicht an, sondern haben aus Effizienzgründen feste Feldlängen. Diese stellen dafür andere Konstrukte zur Verfügung, um Felder variabler Länge (z.B. für Textdokumente, Grafiken) im Datenbanksystem abzulegen.

9.3.2. Zugriffspfadverwaltung

Bis jetzt können Daten im Datenbanksystem zwar abgelegt werden, doch muss zum Auffinden eines Datensatzes mit bestimmten Eigenschaften der gesamte Datenbestand sequentiell durchsucht werden. In einem ersten Lösungsansatz könnten die Datensätze nach einem einzigen Kriterium physisch sortiert werden (z.B. nach dem Entitätsschlüssel Kundennummer), doch sind dann andere Zugriffspfade (z.B. nach dem Kundennamen oder nach Ort und Kundename) noch immer nicht möglich. Mittels Hilfsstrukturen sollen derartige Zugriffe nach beliebig kombinierten Attributen möglichst effizient gestaltet werden können.

Die physische Sortierung der Daten auf dem Speichermedium wird physischer Zugriffspfad genannt. Das physische Sortierkriterium entspricht dem Primärschlüssel. Der Primärschlüssel muss nichts mit dem Entitätsschlüssel (oder auch Identifikationsschlüssel) der Entitätsmenge gemeinsam haben (meist verwendet das DBMS einen eigenen Primärschlüssel, die Datensatznummer). Der sekundäre Zugriffspfad erlaubt den Zugriff mit einem frei definierbaren Sekundärschlüssel. Im Gegensatz zum Primärschlüssel dürfen für einen Sekundärschlüssel bei einem bestimmten Schlüsselwert beliebig viele Datensätze auftreten (z.B. 20 Entitäten mit dem Kundennamen 'Müller'). Im Folgenden wird nicht mehr zwischen physischem und sekundärem Zugriffspfad differenziert, da die Realisierung der Zugriffspfade auf inhaltlich identischen Algorithmen basiert.

Zur Optimierung der Zugriffszeiten müssen für sekundäre Zugriffspfade zwingend redundante Informationen durch den Zugriffspfadverwalter (z.B. ein Index nach Kundennamen) bereitgestellt werden. Es werden damit gezielt redundante Informationen eingeführt, um die Leistung des Datenbanksystems zu erhöhen. In der Theorie wurde eine Vielzahl von Algorithmen für Zugriffsverfahren entwickelt. In der Praxis sind allerdings nur wenige dieser Verfahren mit leichten Abweichungen und allerlei Mischvarianten anzutreffen.

 Mittels Hilfsstrukturen wird der effiziente Zugriff nach einem festgelegten Kriterium möglich. Doch lässt sich der Zugriff weiterhin beschleunigen, falls die Datensätze nach dem am häufigsten verwendeten Kriterium eines Sekundärschlüssels physisch sortiert abgelegt werden und zur Sortierung nicht die Datensatznummer verwendet wird. Diese gezielte physische Anordnung der Datensätze wird Clustering genannt. Können z.B. 20 Datensätze pro Segment gespeichert werden, so müsste im günstigsten Fall erst nach jeweils 20 Datensätzen wieder ein neues Segment gelesen werden. Wird kein Clustering eingesetzt, sind die Datensätze unsortiert und es würde allenfalls pro Zugriff ein Segment gelesen werden. Mittels Clustering kann die Zugriffshäufigkeit auf das Speichermedium daher im angesprochenen Fall bis zum Faktor 20 reduziert werden. Dies bedingt allerdings, dass die Entwickler über das Clustering informiert sind und dieses gezielt nutzen, oder dass das Datenbanksystem selbständig den optimalen Zugriffspfad wählt.

9.3.2.1. Zugriffsverfahren mit invertierten Listen

Bei der invertierten Liste wird für das gewünschte Zugriffskriterium ein Index gebildet in welchem die auftretenden Werte sortiert abgelegt werden. Zusätzlich wird für jeden Indexeintrag eine Zeigerliste geführt, welche die betreffenden Datensätze referenziert (z.B. mittels interner Datensatznummer). Für einen Zugriff auf den bzw. die Datensätze mit dem gewünschten Kriterium wird zunächst im Index die Position der Datensätze bestimmt und anschliessend direkt auf die entsprechenden Datensätze zugegriffen. Eine Zugriffsbeschleunigung erfolgt aus zwei Gründen: Erstens ist der Index nach dem gewünschten

Zugriffskriterium sortiert und ermöglicht damit binäres Suchen. Zweitens benötigt der Index selbst deutlich weniger Platz als die eigentlichen Daten, so dass pro Segment beträchtlich mehr Indexeinträge als Datensätze gespeichert werden können.

Index	Anz. Zeiger	Zeigerlisten mit Record-Id		
Schluep	2	2'345	321	
Schlüer	1	9'452		
Schlumsberger	4	16	341	389 1'532
Schlünz	1	734		
...	...			

Figur 35: Index und Zeigerlisten bei der invertierten Liste

Da die Länge der Zeigerlisten zu Beginn nicht bekannt ist und die Realisierung variabel langer Zeigerlisten aufwendige Algorithmen bedingt, werden die Zeigerlisten meist separat zur Indexstruktur abgelegt. In der Indexstruktur wird lediglich auf die betreffende Zeigerliste referenziert (wieder mittels Zeiger), welche separat hiervon verwaltet wird.

Trotz der einfachen Struktur und Algorithmen sind invertierte Listen in Reinform kaum anzutreffen. Das Aufsuchen von Datensätzen ist zwar äusserst effizient, aber beim Einfügen und Löschen neuer Datensätze ergeben sich Schwierigkeiten. Grundlage der invertierten Listen ist ein sortierter Index, dessen Einträge physisch sortiert sind. Dies bedingt, dass beim Einfügen für den neuen Indexeintrag Platz geschaffen werden muss und sämtliche nachfolgenden Indexeinträge daher nach hinten geschoben werden. Entsprechend müssen beim Löschen die Lücken wieder geschlossen werden. Zwar können durch Überlaufbereiche die Mängel etwas gedämpft werden, sie gewähren aber nur einen mangelhaften Kompromiss.

Um diesen Mangel zu beseitigen liegt die Idee nahe, den Index in kleinere Einheiten zu gliedern und zum Auffinden des korrekten Indexteiles wiederum einen Index zu verwenden. Dadurch würde ein hierarchischer, zweistufiger Index gebildet. Natürlich kann dieser Unterteilungsschritt beliebig oft wiederholt werden und dadurch ein mehrstufiger Index gebildet werden. Dieser Vorgang wird sinnigerweise so lange wiederholt, bis die grössten Indexteile physisch exakt die Grösse eines Segmentes haben. Verfeinert man diesen Algorithmus noch in der Art und Weise, dass die einzelnen Indexteile auf allen Hierarchiestufen möglichst gleichmässig und vollständig gefüllt sind, dann handelt es sich annähernd um einen baumstrukturierten Zugriffspfad.

Interessant ist natürlich die Frage, wie viele Zugriffe sind auf den Systempufferverwalter nötig, bis die gesuchten Daten gefunden werden. Die Antwort ist nicht ganz einfach zu geben. Geht man von einem einfachen Index aus, in welchem mittels binärer Suche die Zeigerreferenz eruiert wird und bei welchem für jeden Suchschritt ein Segment angefordert werden muss (was nicht ganz korrekt ist), so müssen etwa $\log_2(D)$ Zugriffe auf den Index erfolgen (D entspricht der Anzahl der Datensätze). Bei 1'000'000 Datensätzen sind damit ca. 17 Zugriffe notwendig. Dies ist ein durchaus befriedigendes Ergebnis, allerdings treten beim Einfügen und Löschen von Daten beträchtliche Probleme auf. So muss für diese Operationen im Schnitt jeweils die Hälfte aller Datensätze verschoben werden, für ein Datenbanksystem keine akzeptable Lösung.

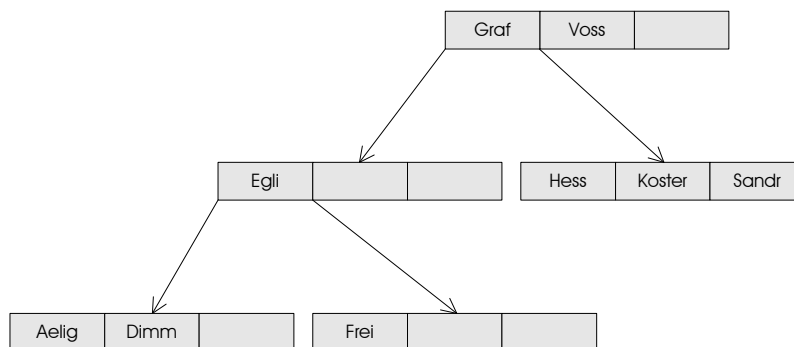
9.3.2.2. Baumstrukturierte Zugriffsverfahren, B- und B*-Baum

Für baumstrukturierte Zugriffsverfahren wurde eine Vielzahl unterschiedlicher Algorithmen entwickelt. Im Folgenden werden zwei wichtige Vertreter vorgestellt, welche sich speziell für den Einsatz in Datenbankumgebungen eignen und daher auch eine entsprechende Verbreitung gefunden haben. Dabei handelt es sich bei Ersterem um sogenannte B-Bäume und beim Zweiten um eine Variante hiervon, dem B*-Baum (siehe auch [Wirth 83]). Das B in der Bezeichnung bringt zum Ausdruck, dass es sich um balancierte Bäume handelt, es darf nicht mit Binär- (für Datenbanksysteme ungeeignet) oder Bitbäumen (nur in Ausnahmefällen geeignet) in Verbindung gebracht werden. Folgende Begriffe werden im Zusammenhang mit Bäumen verwendet:

1. Knoten: Die Knoten sind die Grundkomponenten des Baumes.
2. Zeiger: Mittels der Zeiger (physische Adressverweise, Pointer) werden die Knoten des Baumes miteinander verknüpft.
3. Wurzel: Die Wurzel ist der oberste Knoten des Baumes.
4. Blätter: Die Blätter sind jene Knoten, welche selbst keine untergeordneten Knoten mehr haben.

In B-Bäumen werden die Daten baumartig in den Knoten angeordnet, welche mittels Zeigern verknüpft sind. Die Grösse der Knoten entspricht dabei i.d.R. der Grösse der durch den Systempufferverwalter verarbeiteten Segmente. Die Knoten des B-Baumes beinhalten:

1. Mehrere Schlüsseleinträge
2. Die jeweils zugehörigen Daten selbst oder als Variante einen Zeiger auf die eigentlichen Daten
3. Mehrere Zeiger, welche auf je einen untergeordneten Knoten verweisen, in welchem sämtliche Schlüsselwerte grösser sind als der links vom Zeiger liegende Schlüsselwert, aber in welchem alle Schlüsselwerte kleiner sind als der rechts vom Zeiger liegende Schlüsselwert (dies gilt auch für alle wiederum untergeordnete Knoten)



Figur 36: Baum für das Attribut Name (ohne Daten)

Damit ist es möglich einen gewünschten Schlüsselwert gezielt zu suchen. Bei optimaler Verteilung und Belegung des Baumes ist das Suchen eines Schlüsselwertes sogar effizienter als die binäre Suche, und das ganz ohne dass die Daten physisch sequentiell geordnet sein müssen. Lediglich in den einzelnen Knoten selbst müssen die Daten sortiert sein.

Durch geeignete Algorithmen muss nun sichergestellt werden, dass die Knoten immer mindestens bis zu einem bestimmten Grad gefüllt sind und dass der Baum mehr oder weniger gleichmässig balanciert ist. Werden diese Forderungen nicht aufgestellt, kann ein Baum im schlimmsten Fall zu einer linearen Liste verkommen, der Zugriff zu gesuchten Daten entspricht dann dem sequentiellen Durchsuchen der Daten. In der Regel werden folgende Bedingungen für B-Bäume definiert:

1. Alle Knoten, mit Ausnahme der Wurzel, müssen mindestens je zur Hälfte mit Daten gefüllt sein.
2. Die Weglänge von der Wurzel bis zu sämtlichen Blättern des B-Baumes ist identisch.