INFORMATIONS-TECHNIK
Ausstellungsstrasse 70
CH-8090 Zürich
Homepage www.tbz.ch
Telefon 044 446 96 60
Telefax 044 446 96 66
E-Mail admin.it@tbz.zh.ch

Modul 153

Themenübersicht

Gerd Gesell, 24. November 2010

TECHNISCHE BERUFSSCHULE ZÜRICH

Löschen in professionellen Datenbanken

Die Standardoperationen, die über SQL an Datenbankbeständen von den Anwendern ausgeführt werden, sind Daten erfassen, verändern, abfragen und löschen. In fast allen professionellen Applikationen ist die Löschfunktion – also der SQL-Befehl delete – jedoch tabu. Damit wäre ein Informationsverlust verbunden, der aus verschiedenen Gründen unerwünscht ist.

Ein typischer Denkfehler von Datenbankneulingen ist es zum Beispiel, einen austretenden Mitarbeiter aus der Personaltabelle zu löschen. Wenn ich diesen Datensatz aber wirklich entferne, gehen mir sämtliche Beziehungen verloren, die damit in Verbindung stehen. Ich kann also nicht mehr nachvollziehen, welche Aktionen dieser Mitarbeiter früher gemacht hat. Innerhalb von Banken werden zum Beispiel die Zugriffe auf Kontodaten genau protokolliert. Wenn der austretende Mitarbeiter also gelöscht würde, wären sämtliche Aktivitäten von ihm nicht mehr nachvollziehbar. Das könnte zum Teil sogar juristisch Probleme geben.

Die Lösung im Falle der Mitarbeiterdaten wäre also, dass ich keine Daten lösche, sondern meinen Informationsbestand um die Austrittsdaten erweitere und den Mitarbeiterdatensatz eventuell als inaktiv markiere, was aber aufgrund der Austrittsdaten genau genommen redundant ist.

Auch zeitliche Abläufe werden nicht durch überschreiben (update) bestehender Einträge abgebildet. Ein Gegenstand, der mehrmals hintereinander an verschiedene Personen verliehen wird hat nicht nur einen Fremdschlüssel, der auf die jeweils aktuelle Person zeigt, sondern ich benötige weitere Tabellen, um die zeitliche Dynamik statisch abzubilden. Schliesslich sollen von der Datenbank Informationen und Auswertungen auch aus der Vergangenheit gemacht werden (z. B. wieviel Prozent der Zeit war der Artikel x ausgeliehen?).

Betrachten wir als nächstes Beispiel ein Kassensystem. Wenn ein Einkauf getätigt und erfasst worden ist, merkt der Kunde, dass er zu wenig Bargeld dabei hat und möchte auf den Kauf verzichten. Wenn das Datenbanksystem jetzt ein einfaches Löschen der zugehörigen Datensätze erlaubt, wäre das gleiche Verhalten auch missbräuchlich verwendbar. Es können ein paar Datensätze gelöscht werden und schon wäre überzähliges Bargeld in der Kasse und jeden Abend Party in der Verkaufsabteilung. Andererseits kann ich den Kunden auch nicht zum Kauf verpflichten nur weil meine Datenbank keine Korrekturmöglichkeit hat. Dieses Problem ist aber über Stornierungen möglich. Dabei kann ich je nach Informationsstruktur spezielle Tabellen für die Stornierungsbuchungen haben oder ich speichere sie in den gleichen Tabellen wie die Buchungen ab, aber mit zusätzlichen Informationen für die Zeit, Stornierungsgrund usw.

Wenn die Datenbank mit referentieller Integrität erstellt wurde, werden die Beziehungen vom DBMS verwaltet und überwacht. In diesem Fall kann ich die Daten schon aus technischen Gründen nicht löschen, es sein denn ich lösche auch die in Beziehung stehenden Datensätze. Stellen Sie sich aber ein System wie Wikipedia vor. Der Benutzer hat Einträge und Veränderungen an diversen Orten vorgenommen. Diese Veränderungen sind dann von anderen Benutzern auch wieder verändert worden. Da in Wikisystemen alle Aktivitäten historisiert werden, kann ich aus dieser Veränderungshistorie nicht einfach Informationen löschen. Aus diesem Grund werden auf Wikipedia Benutzernamen auch dann nicht gelöscht, wenn sie zur Kategorie „bäh“ gehören. Sie werden bestenfalls gesperrt, damit unter diesen Namen keine weiteren Aktivitäten erfolgen können.

Mehrfachbeziehungen

Bei Mehrfachbeziehungen haben zwei Tabellen mehrere Beziehungen, die unabhängig voneinander sind und jeweils einen anderen Sachverhalt repräsentieren. Um die verschiedenen Beziehung eindeutig zu kennzeichnen, müssen sie möglichst aussagekräftig beschriftet werden. Die Kardinalitäten können natürlich verschieden sein, da die Beziehungen voneinander unabhängig sind. In den Beispielen unten gibt es drei unabhängige Beziehungen zwischen den Entitätstypen *tbl_Fahrten* und *tbl_Orte*. Eine dieser Beziehungen ist die Auflistung der Orte, die bei einer Tour angefahren werden können. Dies ist eine mc:mc-Beziehung und benötigt daher eine Transformationstabelle.

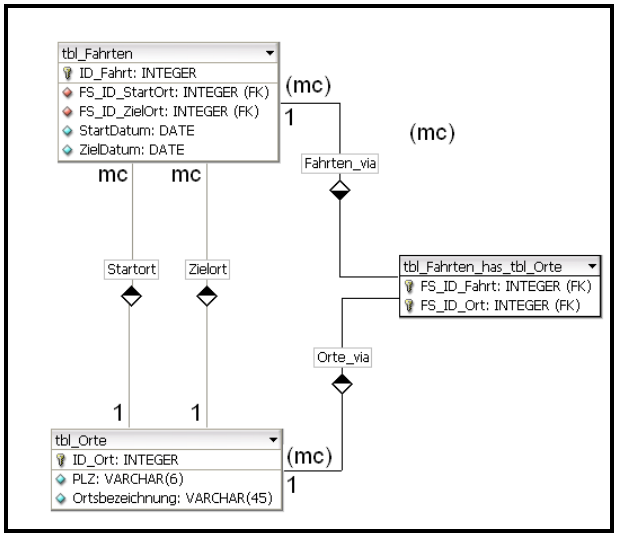


Abbildung 1: Darstellung mit DBDesigner4 (Kardinalitäten wurden nachträglich eingetragen)

Bei der Abbildung 2 ist der gleiche Sachverhalt dargestellt. Nur unterscheidet sich die Darstellung in MS-Access, da es nicht in der Lage ist mehr als eine Beziehung zwischen 2 Tabellen grafisch darzustellen. Das Problem wir gelöst, indem eine Tabelle „geclont“ wird (zu erkennen an der Bezeichnungserweiterung _1). Physikalisch sind *tbl_Orte* und *tbl_Orte_1* jedoch die gleiche Tabelle.

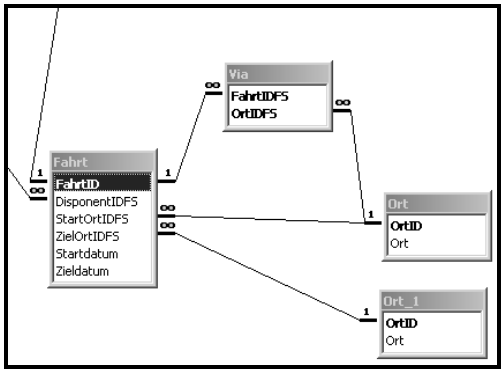


Abbildung 2: Darstellung im Beziehungsfenster von MS-Access

Generalisierung/Spezialisierung

Die Datenbankmodellierung, die wir hier betrachten beruht auf dem Attributkonzept. Wir definieren Attribute mit Attributsausprägungen und ordnen diese den Entitätstypen zu. Problematisch wird es, wenn jetzt mehrere Entitätstypen viele Attribute gemeinsam haben und einige nicht. Jetzt könnte Redundanz entstehen, wenn ein konkretes Objekt aus der realen Welt durch mehrere Entitätstypen beschrieben wird. In unserem konkreten Beispiel wären das Mitarbeiter, die auch als Kunden auftreten oder Fahrer, die auch als Disponenten arbeiten. In Anlehnung an [ZEHNDER, Carl August (1989): Informationssysteme und Datenbanken] dürfen „lokale Attribute“ nur einmal in einer Datenbank auftreten, da sonst Redundanz vorliegt. Die Lösung des Problems besteht darin, dass die gemeinsamen Attribute in einem allgemeinen Entitätstypen zusammengefasst werden (Generalisierung). Die nicht gemeinsamen Attribute verbleiben in den Entitätstypen (Spezialisierung). Um einen Informationsverlust zu vermeiden, müssen die Datensätze der spezialisierten Tabellen per Fremdschlüssel auf die generalisierten Tabellen verweisen. Diese Beziehung wird auch als „is-a“-Beziehung bezeichnet: eine Person *ist ein* Fahrer.

Diese Beziehungsform ist auch in der objektorientierten Modellierung bekannt und wird dort mittels Vererbung gelöst.

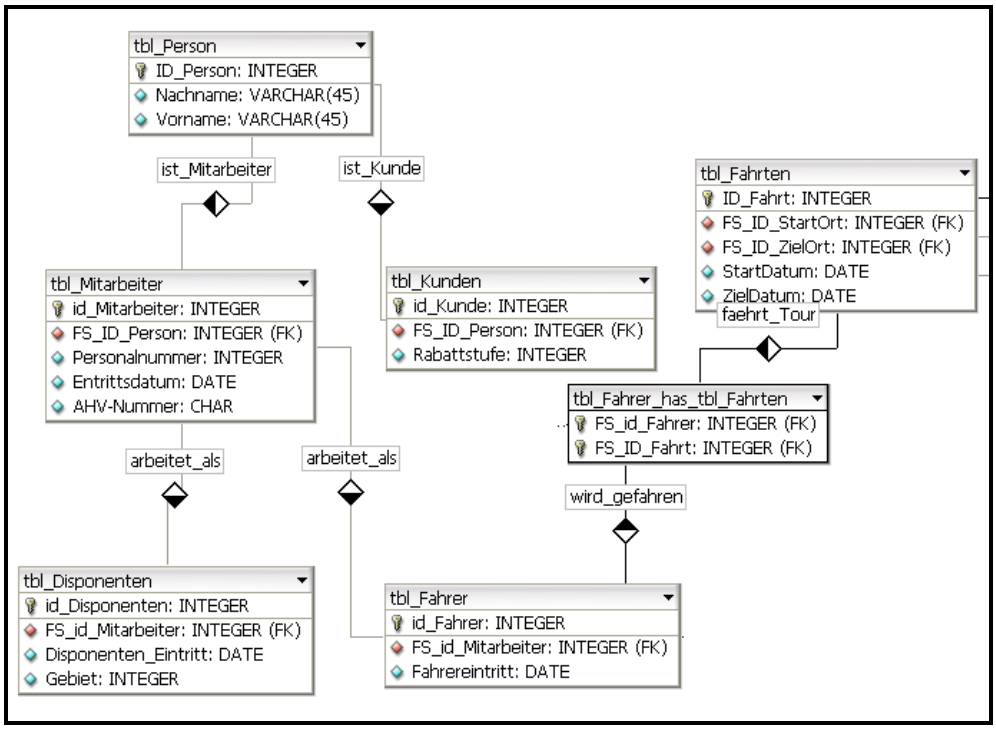


Abbildung 3: Beispiel für Generalisierung/Speziaisierung

Rekursion

Bis jetzt haben wir Beziehungen zwischen zwei verschiedenen Entitätstypen betrachtet. Es gibt aber auch die Möglichkeit, dass an einer Assoziation nur ein einziger Entitätstyp beteiligt ist. Konkret bedeutet das, dass ein Datensatz dieser Tabelle mit einem anderen Datensatz aus der gleichen Tabelle in Beziehung steht. Eine reine Hierarchie können wir innerhalb der gleichen Tabelle erledigen. Sie erhält also einen Fremdschlüssel, der auf den Identifikationsschlüssel der eigenen Tabelle verweist. Das klassische Beispiel hierfür ist eine Firmenorganisation, bei der jede Person genau eine andere Person als Vorgesetzten hat. Da die in der Hierarchie höchste Person natürlich keinen Vorgesetzten mehr haben kann, ist es eine c:mc Beziehung. Würden wir dieses Fremdschlüsselattribut auf „not null“ setzen (im Access: „Eingabe erforderlich“), könnten wir die „höchste Person nicht erfassen.

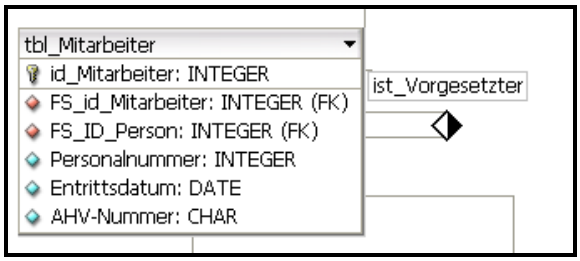


Abbildung 4: Darstellung einer einfachen Hierarchie im DBDesigner4

Mit dieser einfachen Variante kommen wir aber nicht weiter, wenn aus der klaren Hierarchie (einmal-Rekursion – nur ein Vorgesetzter ist möglich) eine Netzwerkstruktur wird. In diesem konkreten Beispiel bedeutet das, dass eine Person auch mehrer Vorgesetzte haben kann. Dann handelt es sich um eine mc:mc Beziehung und wir benötigen eine Transformationstabelle. Speziell daran ist, dass die beiden Fremdschlüssel der Transformationstabelle jeweils auf einen Identifikationsschlüssel der gleichen Tabelle zeigen, aber in unterschiedlichen Rollen. Im Beispiel sind das die Rollen „ist Vorgesetzter von“ und „ist_Mitarbeiter_von“.

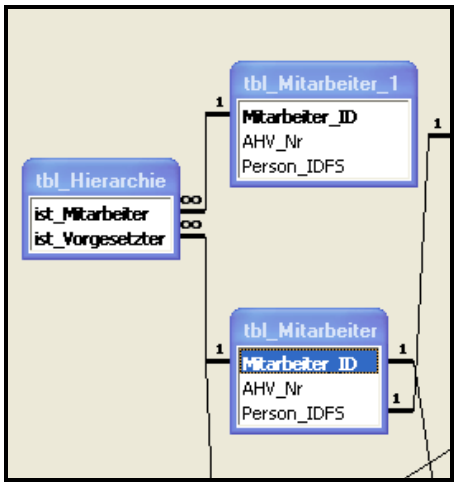
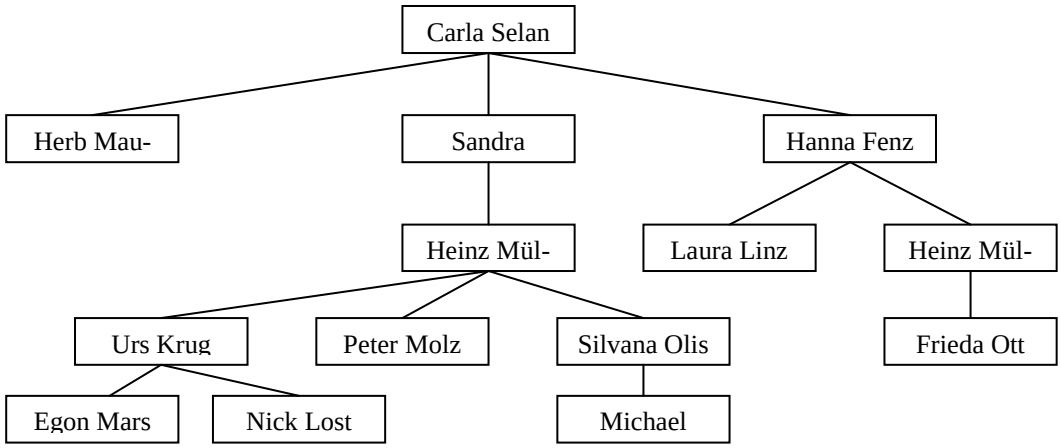


Abbildung 5: MS-Access-Darstellung einer netzwerkförmigen Struktur

einfache Hierarchie



Einfache Hierarchien können mit Rekursionen innerhalb der gleichen Tabelle abgebildet werden. Solange jedes Element nur maximal ein Oberelement hat, reicht ein Fremdschlüssel, der auf die Identifikationsschlüssel der eigenen Tabelle verweist. Lediglich das Wurzelelement oder Topelement hat einen NULL-Wert, da die Hierarchie ja irgendwo enden (oder anfangen) muss. Es handelt sich hierbei um eine c:mc Beziehung.

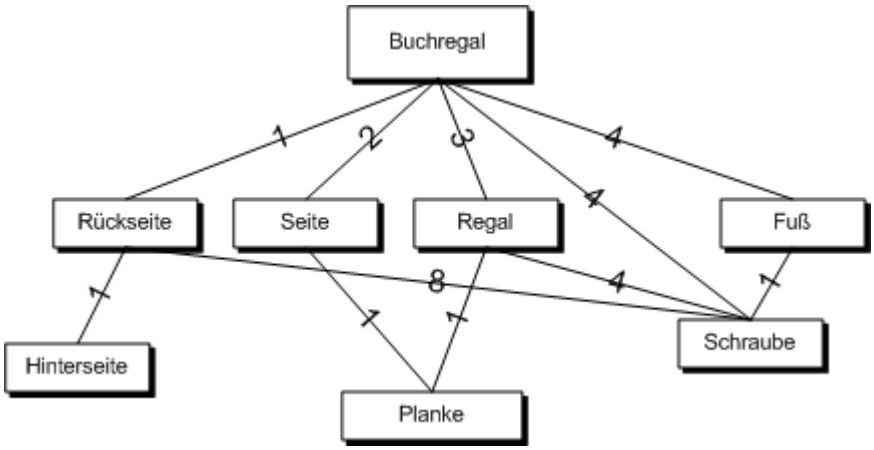
Wenn aber mehrere Oberelemente zugelassen sind (z. B. mehrere Projektleiter), dann handelt es sich um eine mc:mc-Beziehung und es wird eine Transformationstabelle benötigt, die für jeden Beziehungsgraphen einen Datensatz enthält, der angibt welches Oberelement zu welchem Unterelement gehört.

Stücklistenproblem

Eine klassische Anwendung der Rekursion in der Datenmodellierung ist die Stücklistenproblematik. Es gibt eine Tabelle mit Produkten und deren Eigenschaften. Jetzt kann sich aber auch ein Produkt aus mehreren anderen Produkten zusammensetzen (modulare Produkte). Beides – die zusammengesetzten Produkte (Aggregate) sowie die einzelnen Bestandteile – befindet sich in unserer Produkte- oder Artikeltabelle und wird auch einzeln verkauft. Das Problem besteht nun darin eine Liste mit Einzelteilen zu erzeugen, die alle Artikel auf elementarer Ebene enthält (die also nicht mehr aus mehreren Produktteilen bestehen).

Ein gutes Beispiel – sogar mit SQL-Code – finden Sie hier:

http://www.ianywhere.com/developer/product_manuals/sqlanywhere/1001/de/html/dbugde10/ug-parts-explosion-cte-sqlug.html



Das Beispiel beschreibt die Artikel einer Elementmöbelfabrik.

Lösung mit Rekursion: Da es sich im Modell ja nicht um eine einfache Hierarchie handelt (c:mc), reicht die Erstellung eines Fremdschlüsselattributs in der Produktetabelle nicht aus. Neben der Produktetabelle benötigen wir eine weitere Tabelle, die die Zusammensetzungen enthält. Also welche Komponente (Einzelteil oder Aggregat) fliesst in welches Aggregat und mit welcher Menge ein?

identifying relationship vs. non-identifying relationship

Zur Unterscheidung der identifying relationship von der non-identifying relationship gibt es extrem viel Schrott im Internet. Eine kurze einfache Erklärung gibt es hier: http://www.datenbank-grundlagen.de/beziehungen-datenbanken.html

Bei der identifying-Beziehung ist der Fremdschlüssel Bestandteil des Indentifikationsschlüssels. Die ID ist also eine aus mehreren (mindestens 2) Attributen zusammengesetzte Schlüsselkombination und der Datensatz der referenzierenden Tabelle („unten“) identifiziert sich teilweise durch den Datensatz, auf den er verweist. Ein Raum kann in der ID also unter anderem den Fremdschlüssel auf das Gebäude haben. Eine non-identifying Beziehung wäre hier nicht so sinnvoll, denn darin könnte ich jederzeit den Fremdschlüsselwert ändern und den Raum einem anderen Gebäude zuordnen.

Die Beziehung Mitarbeiter/Abteilung ist da ein besseres Beispiel für eine non-identifying-Beziehung. Der Fremdschlüssel auf den Arbeitgeber gehört nicht zur Identität einer Person, zumal der Inhalt – also die Firma, auf die referenziert wird – jederzeit wechseln kann.

Suchverfahren

Um in grossen unsortierten Listen einen gewünschten Wert zu finden müssen wir die lineare oder sequentielle Suche anwenden. Es bleibt uns nichts anderes übrig, als die gesamte Liste bis zum erfolgreichen Treffer oder gar bis zum Ende zu durchsuchen, wenn der Wert überhaupt nicht vorhanden ist. Aus diesem Grund ist es einfacher, wenn wir Listen – oder in Datenbanken dann Attribute – sortieren, um schnellere Verfahren anwenden zu können.

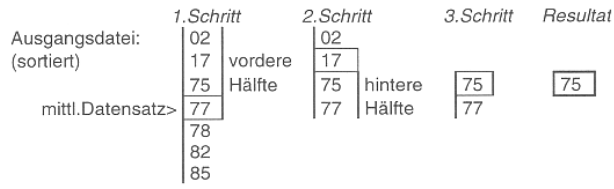
Bei der binären Suche oder der logarithmischen Suche wird eine sortierte Liste vorausgesetzt. Damit halbiert sich bei jedem Zugriff der Suchraum. Wir benötigen bei N Elementen $\log_2$ N Zugriffe. Wenn unser Suchraum aus 16 (=2^4)Werten besteht, benötigen wir also maximal 4 Zugriffe (vorherige „Glückstreffer“ ausgenommen). Mit nur einem Zugriff mehr, können wir bis zu 32 Werte abdecken. Allerdings benötigen wir auch für 17 Werte maximal 5 Zugriffe. Für dieses Suchverfahren wird auch noch der Begriff „logarithmische Suche“ verwendet.

Eine Erweiterung der binären Suche ist die Interpolationssuche. Bei der binären Suche wird der Suchraum jedes Mal genau in der Mitte geteilt. Die Interpolationssuche berücksichtigt noch die Grösse des gesuchten Wertes in Bezug auf den Suchraum. Es vergleicht also den grössten und den kleinsten Wert, geht von einer gleichmässigen Verteilung der Daten aus und teilt den Suchraum jetzt im Verhältnis zu dem Suchwert. Konkret greift es bei einem sehr hohen Wert bereits im „wahrscheinlicheren“ hinteren Bereich des Suchraumes zu. Jeder erfolgreiche Zugriff schränkt den Suchraum also um mehr als die Hälfte ein. Mit der zusätzlichen Berechnung des Zugriffselementes geht aber ebenfalls Rechenzeit verloren. Daher ist die Interpolationssuche nur bei grossen Suchräumen effizienter.

Ein leicht verständliches Zahlenbeispiel finden Sie übrigens bei Wikipedia:
http://de.wikipedia.org/wiki/Interpolationssuche

Die Beschreibung rechts ist aus dem Buch Informationssysteme und Datenbanken; ISBN3 7281 2019 7 von Carl August Zehnder;vdf Hochschulverlag Zürich 6. Auflage Seite 224

Nun brauchen wir noch ein Suchverfahren, um die richtige Seite im Buch, bzw. den richtigen Eintrag auf der Seite *möglichst schnell* finden zu können. Da die Datei sortiert ist, können wir auf beiden Stufen (richtige Seite und richtiger Eintrag) das sog. *binäre Suchverfahren* (auch „Intervallschachtelung“ genannt) einsetzen (Fig.5.3).



Figur 5.3: Das binäre Suchverfahren („Intervallschachtelung“) braucht für N=7 drei Schritte; gesucht ist der Datensatz mit Schlüssel '75'.

Das binäre Suchverfahren funktioniert im wesentlichen für eine geordnete Datei von N Datensätzen wie folgt:

> Jeder Arbeitschritt halbiert die zu untersuchende Datei (Fig.5.3). Geprüft wird dazu einzig der mittlere Datensatz. Liegt der Suchschlüssel vor (bzw. hinter) dem Schlüssel des mittleren Datensatzes, arbeitet man im nächsten Arbeitsschritt nur noch mit der vorderen (bzw. hinteren) Hälfte weiter. Das Halbieren erfolgt solange, bis nur noch der gesuchte Datensatz übrig bleibt.

Es ist leicht zu sehen, dass dieses Verfahren nur wenige Arbeitsschritte braucht; bei 1000 vorsortierten Datensätze lässt sich das gesuchte Element in jedem Fall mit 10 Halbierungen bestimmen (auf 500 - 250 -125 - 63 - 32 - 16 - 8 - 4 - 2 - 1); bei einer Million Datensätze mit 20 Halbierungen. Es gilt folgende logarithmische Regel:

Def.: Das *binäre Suchen* benötigt bei N Elementen $_2$log N Arbeitsschritte.

Indextabellen

Wenn wir grosse Datenbestände haben, ist es also nötig die Attribute zu sortieren, wenn häufiger danach gesucht wird. Eine Tabelle kann aber logischerweise nur nach einem Attribut sortiert werden. Eine schlechte Lösung wäre es jetzt die ganze Tabelle zu kopieren und in der 2. Version nach einem anderen Kriterium zu sortieren. Die bessere Variante sind Indextabellen. Diese sind eine Reduktion der Haupttabelle auf das Schlüsselattribut und das Indexattribut und sind nach dem Indexattribut sortiert. Bei einem Suchzugriff erfolgt der erste Suchvorgang in der Indextabelle und dann mit dem gefundenen Schlüssel der zweite Suchvorgang in der Haupttabelle. Damit sind SELECT-Operationen immer noch deutlich schneller, als eine Tabelle mit mehrere Millionen Datensätzen sequentiell zu durchsuchen.

Ein paar Nachteile bleiben – und deswegen ist es nicht ratsam einfach alle Attribute aus Prinzip zu indexieren:

- auch die Indextabellen benötigen Platz,

- DELETE- und INSERT-Operationen und UPDATE-Operationen auf Indexattribute benötigen mehr Ressourcen, da auch die Indextabellen angepasst werden müssen.

Die Indextabellen werden automatisch vom DBMS verwaltet. Allerdings können Sie die Anzahl und die Grösse der vorhandenen Indices anzeigen lassen.

Wenn Attribute auf unique gesetzt werden, wird automatisch eine Indextabelle erzeugt. Jeder Neueintrag muss auf Eindeutigkeit überprüft werden. Das entspricht einem SELECT mit dem Eingabewert auf das entsprechende Attribut und wird im positiven Fall eine leere Menge zurückgeben (der Wert ist erlaubt, da bis jetzt eindeutig) beziehungsweise im negativen Fall ein Ergebnis anzeigen (der Wert muss abgelehnt werden, da schon vorhanden).

Wenn eine Eindeutigkeit über mehrere Attribute erreicht werden soll, wird dies ebenfalls über eine Indextabelle realisiert. Sie können einen sogenannten „unique-Index“ unter Angabe aller notwendigen Attribute definieren. Damit ist gewährleistet, dass Werte innerhalb einzelner Attribute mehrfach vorkommen dürfen, aber die Kombination konkreter Werte über die ausgewählten Attribute nur einmal.