IMS - Hierarchischen Datenbankmodell

IMS, die Kurzform von Information Management System, wurde um 1968 veröffentlicht und basiert auf einem hierarchischen Datenbankmodell. Das Grundkonzept besteht darin, dass die Ablage von Daten in Segmenten erfolgt. Diese Segmente werden in physischen hierarchischen Baumstrukturen angeordnet.

Die wichtigsten Merkmale eines hierarchischen Datenbankmodells werden in der folgenden Auflistung angeführt:

- Entitäten und die Beziehungen zwischen einzelnen Entitäten werden auf der Grundlage der Graphentheorie dargestellt, wobei die Entitäten den Knoten zugeordnet werden und die Beziehungen zwischen einzelnen Entitäten den Kanten.
- Alle Datensätze werden in einer strengen Baumstruktur abgebildet.
- Das hierarchische Datenbankmodell ist auf einen Schlüssel beschränkt.
- Ein beliebiger Datensatz ist ausschließlich über genau einen Vorgänger erreichbar.
- Das externe Schema ist nicht vorhanden, somit liegt nur eine Zwei-Ebenen Architektur vor.
- IMS bietet keine Datenunabhängigkeit, da die interne Ebene ebenfalls im Zugriff des Anwendungsprogrammierers steht.
  Die verwendete Datenbanksprache DL/1, die Kurzform von Data Language One, bietet keine interaktive Version und ist direkt in PL/1, Cobol oder System/370 Assembler eingebettet.
- Der Zugriff auf ein hierarchisches Datenbankmodell erfolgt mittels navigierender Operationen.

Der wesentliche Vorteil des hierarchischen Modells besteht darin, dass in der Umwelt häufig hierarchische Strukturen vorliegen, wie zum Beispiel ein Organigramm der Aufbauorganisation eines Unternehmens. Daraus resultiert, dass bei den Anwendern eine gewisse Vertrautheit und Gewohnheit mit hierarchischen Strukturen von Grund auf vorhanden ist und daher die Modellierung und Benutzung von hierarchischen Datenbankmodellen für den Anwender einfach zu erlernen ist.

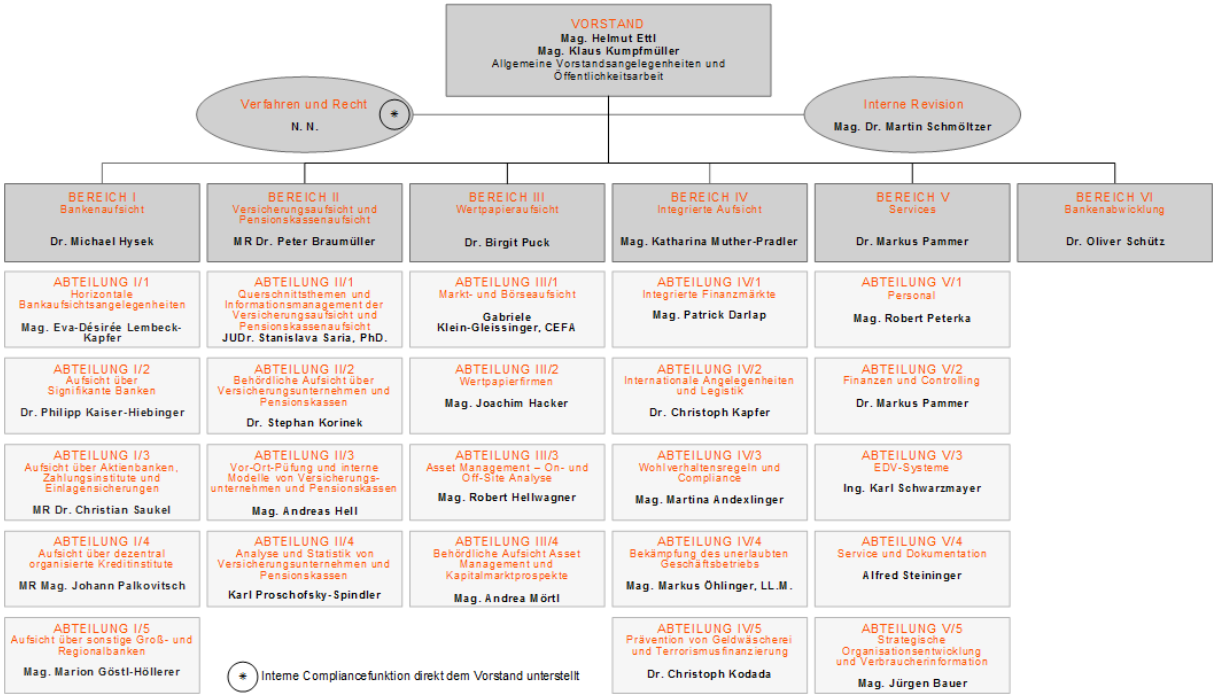

Abbildung 1: Organigramm der Österreichischen Finanzmarkt Aufsicht (FMA)

Aus heutiger Sicht überwiegen die Nachteile von hierarchischen Datenbankmodellen bei weiten deren Vorteile:

- Der Anwender muss die Struktur des Baumes und somit die interne Struktur der Datenbank kennen, um sinnvoll bei Operationen navigieren zu können.
  Jede Entität ist ausschließlich über die ihm zugewiesene Wurzel-Entität, die den Einstiegspunkt in die Hierarche darstellt, sowie über einen gerichteten Pfad erreichbar.
- Änderungen am Datenbankdesign sind nur schwer durchführbar, da die Beziehungen zwischen Entitäten bereits in der Entwurfsphase festgelegt werden müssen.
- Eine m:n-Beziehung zwischen Entitäten ist nur mit erheblichem Aufwand zu realisieren und führt gegebenenfalls zu unerwünschten, redundanten Informationen.
- Die strenge hierarchische Ordnung der Datensätze in einem Baum führt gegebenenfalls zu unerwünschten Auswirkungen bei den Operationen Löschen und Einfügen.
- Im Falle einer Speicherüberschreitung sind aufwendige Reorganisationen der Datenbank nötig.

Das nachfolgende Beispiel eines relationalen Schemas zeigt die Schwierigkeiten einer IMS Implementierung, falls die Beziehung zwischen Entitäten nicht zwangsläufig in einer logischen Baumstruktur abzubilden ist.

Supplier	(	sno,	sname,	scity,	sstate	)
		16	General Supply	Boston	Ma	
		24	Special Supply	Detroit	Mi	
Part	(	pno,	pname,	psize,	pcolor	)
		27	Power saw	7	silver	
		42	bolts	12	gray	
Supply	(	sno,	pno,	qty,	price	)
		16	27	100	$20.00	
		16	42	1000	$0.10	
		24	42	5000	$0.08	

Abbildung 2: Detailbeispiel Relationales Modell

Die Beziehung der Entität Lieferant zur Entität Ersatzteil kann in einer zweistufigen Hierarchie auf zwei unterschiedliche Weisen abgebildet werden:

- Schema 1: Die Wurzel-Entität wird dem Lieferanten zugeordnet.

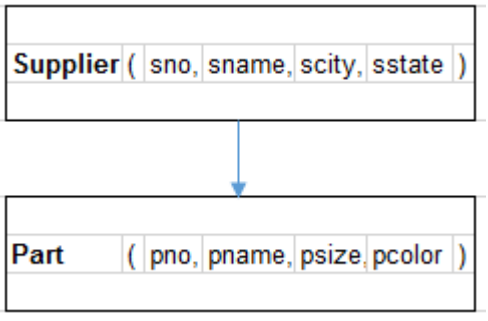

Abbildung 3: Detailbeispiel IMS, Schema 1

- Schema 2: Die Wurzel-Entität wird dem Ersatzteil zugeordnet.

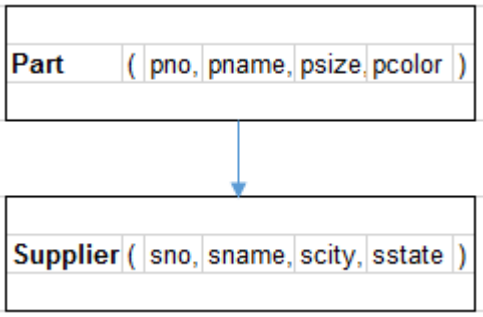

Abbildung 4: Detailbeispiel IMS, Schema 2

Die Datensätze zu Schema 1 sehen demnach wie folgt aus:

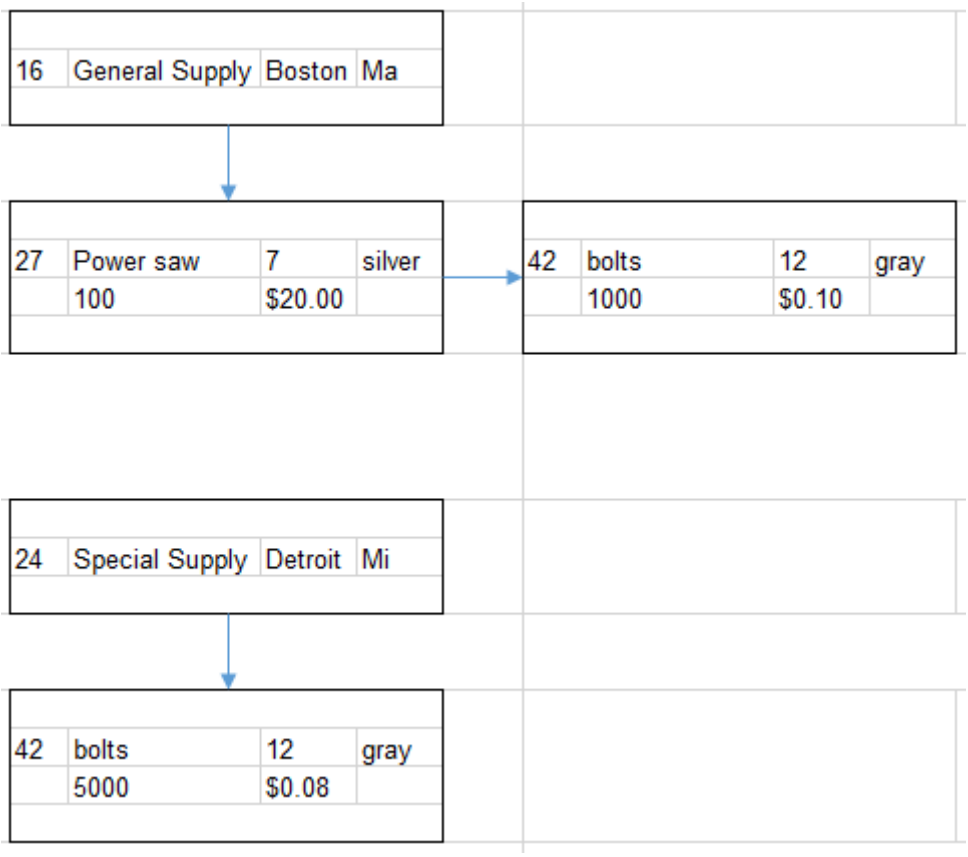

Abbildung 5: Detailbeispiel IMS, Schema 1 - Datensätze

Im zuvor angeführten Beispiel treten zwei unerwünschte Eigenschaften zu Tage:

- Redundante Informationen werden mehrmals in der Datenbank gespeichert.
  Im Schema 1 werden redundante Informationen zu den Ersatzteilen mehrmals gespeichert, da jeder Lieferant aufgelistet werden muss, der das gesuchte Ersatzteil in seinem Produktkatalog führt.
  Im Schema 2 trifft die mehrfache Speicherung an redundanter Information den Lieferanten. Die Informationen des Lieferanten werden für jedes Ersatzteil, das in seinem Produktkatalog geführt wird, redundant gespeichert.
  Redundante Informationen in Datenbanken sind unbedingt zu vermeiden, da sie unweigerlich über die Nutzungsdauer der Datenbank zu inkonsistenten Daten führen. Wird zum Beispiel fälschlicher Weise im zweiten Schema nur bei einigen Ersatzteilen die Adresse (scity oder sstate) des Lieferanten korrigiert und bei den restlichen Ersatzteilen des zu korrigierenden Lieferanten vergessen, ist die Datenbank bereits inkonsistent.

- Ein Element in einer unteren Hierarchieeben kann nicht ohne zugehöriges Element in der oberen Hierarchieebene existieren.
  Daraus folgt, dass es im Schema 1 nicht möglich ist ein Ersatzteil zu speichern, das zum Zeitpunkt der Erfassung von keinem Lieferanten bezogen werden kann.
  Wobei wiederum im Schema 2 keine Möglichkeit besteht einen Lieferanten in der Datenbank zu führen, der vorübergehend kein Ersatzteil liefert.
  In einem strengen hierarchischen System ist keine Unterstützung für diese Art von Sonderfällen vorgesehen.

Alle Datensätze einer IMS-Datenbank besitzen einen hierarchischen Sequenzschlüssel (Kurzform: HSK). Ein hierarchischer Sequenzschlüssel setzt sich aus einer Verknüpfung aller Schlüssel der vorangegangenen Datensätze und dem Schlüssel des aktuellen Datensatzes zusammen und definiert somit eine Reihenfolge aller Datensätze einer IMS-Datenbank.

Bei der Art der Speicherung von Datensätzen wird zwischen Wurzel- und Folgedatensatz unterschieden:

- Wurzel-Datensätzen können bei IMS sequentiell, indiziert (B+ Baum), oder gehasht verspeichert werden. Bei Indizierung und Hash wird jeweils der Schlüssel des Datensatzes zur Anwendung der Funktion herangezogen.
  Die Wahl der Speicheroption beschränkt den Umfang der Befehle der Datenbanksprache DL/1:

  - Bei der Verwendung der sequentiellen Option ist ein Einfügen von neuen Datensätzen nicht mehr unterstützt.
    Somit wird diese Option nur bei Batch-Umgebungen herangezogen. Zuerst wird eine Änderungsliste in der Reihenfolge des hierarchischen Sequenzschlüssels erzeugt, anschließend die Datenbank einmal verarbeitet, wobei die neuen Datensätze sofort an der richtigen Stelle eingefügt werden und anschließend wird das Ergebnis in eine neue Datenbank geschrieben. Diese Abfolge wird meist als „Old Maser, new Master“ bezeichnet.

  - Bei der Verwendung der Hash Option wird der Befehl „get next“ nicht mehr unterstützt, da keine Möglichkeit besteht den nachfolgenden Datensatz in der Reihenfolge des hierarchischen Sequenzschlüssels einfach auszuwerten.

  Die Einschränkungen wurden getroffen, um Designfehler auszuschließen, die zu einer schlechten Performance der Datenbank führen.

- Folgedatensätze werden sequentiell gespeichert.

Beim Navigieren durch den Baum gilt folgende Regel:

- Als erstes erfolgt die abwärts Navigation, anschließend wird von links nach rechts gesprungen.

Diese Regel wird in der Datenbanksprache DL/1 konsequent umgesetzt:

- Der Befehl „get next“ liefert den nachfolgenden Datensatz in der Reihenfolge des hierarchischen Sequenzschlüssels.
- Der Befehl „get next within parent“ listet den verblieben Baum unterhalb des aktuellen Datensatzes auf.

Im Schema 1 können somit alle roten Ersatzteile des Lieferanten #16 mittels folgender Abfrage aufgefunden werden:

```
Get unique Supplier (sno = 16)
Until no-more {
      Get next within parent (color = red)
}
```

Der erste Befehl lokalisiert den Lieferanten #16, danach wird unterhalb des gefundenen Datensatzes in der Reihenfolge des hierarchischen Sequenzschlüssels nach Ersatzteilen mit dem Attribut Rot gesucht. Hat man mit der Abfrage das Ende des verbliebenen Baumes erreicht, wird eine Fehlermeldung returniert.

Programmierer ist bei der Datenbanksprache DL/1 verantwortlich für die richtige Wahl des Abfragealgorithmus. IMS führt den gewählten Abfragealgorithmus lediglich aus ohne jegliche Optimierungen.

Die Abfrage nach dem roten Ersatzteil im Schema 1 könnte ebenfalls lauten:

```
Until no-more {
      Get next Part (color = red)
}
```

Bei einer Vielzahl an Lieferanten wäre die zweite Abfrage viel langsamer als die erste Abfrage. Jedoch im speziellen Fall, dass nur ein einziger Lieferant vorhanden ist, würde die zweite Abfrage eine bessere Performance liefern. Ein auf die Datenbanksprache DL/1 spezialisierter Programmierer muss mit diesen Optimierungsmaßnahmen bestens vertraut sein. Diese erweiterte Programmierkenntnis schlug sich ebenfalls auf einen höheren Marktwert für IMS Programmierer nieder.

CODASYL - Netzwerkorientiertes Datenbankmodell

CoDaSyL steht für Conference on Data Systems Languages. Die ursprüngliche Konferenz fand am 28. und 29.06.1959 in der USA statt. Anwesend waren neben Regierungs- und Militärvertretern auch die damals führenden Computerhersteller, um über die Entwicklung einer gemeinsamen Programmiersprache zu beraten, die eine weitreichende Hardwarekompatibilität bieten sollte. Die erste Errungenschaft von CODASYL war somit die Entwicklung der Programmiersprache COBOL. Ein weiterer bedeutender Meilenstein wurde mit den sogenannten CODASYL-Datenbanken oder Netzwerk-Datenbanksystemen gelegt, indem folgende standardisierte Verfahren von der Data Base Task Group (DBTG) beschlossen wurden:

- Definition der Struktur: Data Definition Language
- und der Bearbeitung: Data Manipulation Language

Auf weitere wesentliche Datenbankeigenschaften, wie die Datenunabhängigkeit wurde jedoch zum damaligen Zeitpunkt noch kein Augenmerk gelegt.

Die wichtigsten Merkmale eines netzwerkorientierten Datenbankmodells werden in der folgenden Auflistung angeführt:

- Die Ablage von Daten erfolgt in Satzarten (record types).
- Satzarten können in beliebigen Beziehungen zueinander stehen (set type, owner-member).

Diese Strukturierungsmerkmale bilden die Grundlage zur Abbildung von komplexen Netzwerkstrukturen.

Folgende Vorteile ergeben sich aus der Sicht der Datenmodellierung:

- Netzwerkorientierte Datenbankmodelle bieten eine redundanzfreie Abbildung von Datenstrukturen.
- Unterschiedliche Möglichkeiten der Strukturierung stehen zur Verfügung.
- Beliebige Beziehungen zwischen Satztypen sind modellierbar. Für die Modellierung von m:n Beziehungen wurden eindeutige Regeln definiert.
- Ein performanter Zugriff wird mit Hilfe von eingerichteten Pfaden realisiert.

Die folgenden Nachteile müssen bei der Verwendung von netzwerkorientierten Datenbankmodellen in Kauf genommen werden:

- Die Darstellung beliebig komplexer Strukturen ist unübersichtlich.
- Der Anwendungsprogrammierer muss über Kenntnisse des Zugriffspfades verfügen und der Anwender über Wissen der internen Datenbankstruktur.
- Die Beziehungen zwischen den einzelnen Entitäten sind bereits beim Entwurf der Datenbank festzulegen. Eine bereits vorhandene Beziehung ist schwierig zu adaptieren.
- Die Datenbank erfordert eine aufwendige Reorganisation im Falle einer Speicherüberschreitung.

UDS

UDS, die Kurzform von Universelles Datenbank System, wird als Beispiel einer netzwerkorientierten Datenbank angeführt. Die Entwicklung von UDS erfolgte im Kontext des Betriebssystems BS2000 und ist weitgehend konform zur ANSI-SPARC Architektur. Die Datenbankabfragesprache ist eingebettet in Cobol, Assembler, Fortran und Pascal. UDS besitzt eine interaktive Abfragesprache IQS, die Kurzform von Interactive Query System, jedoch keinen eigenen Query-Optimierer. Nachfolgend wird das Detailbeispiel des Relationalen Schemas in Abbildung 2 herangezogen, um die Modellierung eines netzwerkorientierten Datenbankmodelles zu veranschaulichen.

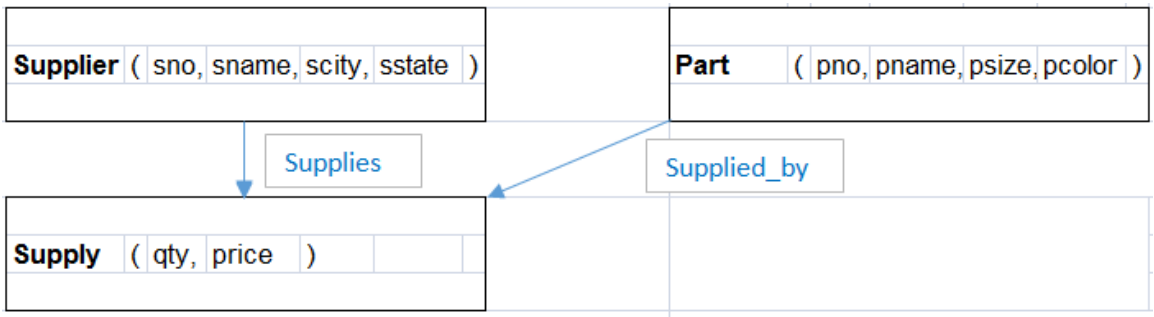

Abbildung 6: Detailbeispiel CODASYL

Das netzwerkorientierte Datenbankmodell organisiert eine Sammlung von Datensatztypen, die jeweils einen eindeutigen Schlüssel aufweisen, mit Hilfe eines gerichteten Graphen. Aus diesem Grund kann eine Datensatzinstanz mehrere Elternteile aufweisen und unterscheidet sich daher grundlegend vom Ansatz des hierarchischen Datenbankmodells, in dem ausschließlich ein Elternteil vorliegen kann.
Im Detailbeispiel sind die drei Datensatztypen Supplier, Part und Supply in einem gerichteten Graphen angeordnet. Die beschrifteten Pfeile Supplies und Supplied_by stellen gerichtete Kanten dar und werden in CODASYL als Set-Typ bezeichnet. Eine gerichtete Kante startet stets vom Member-Typ, (Supplier, Part) und endet beim Owner-Typ (Supply).

Eine typische Operation unter Verwendung der CODASYL Datenmanipulationssprache besteht in der Navigation der im Graphen-Netzwerk verbundenen Entitäten. Zuerst wird eine Datensatzinstanz als Einstiegspunkt gewählt und anschließend über die Kanten der gerichteten Graphen zu den gesuchten Daten navigiert.

Im Detailbeispiel CODASYL können alle roten Ersatzteile des Lieferanten #16 mittels folgender Abfrage aufgefunden werden:

```
Find Supplier (sno = 16)
Until no-more {
      Find next Supply record in Supplies
      Find owner Part record in Supplied_by
      Get current record
      check for red
}
```

Der erste Befehl setzt den Lieferanten #16 als Einstiegspunkt, danach werden in der Schleife alle verbundenen Datensätze in der Entität Supply und die wiederum verknüpften Datensätze in der Entität Parts zu dem Lieferanten #16 gesucht und jeder gefundene Datensatz auf das Attribut Rot verglichen. Falls der letzte Datensatz in der Entität Supply erreicht ist, wird die Schleife beendet.

Das hierarchische und das netzwerkorientierte Datenbankmodell sind in der heutigen Zeit nur mehr selten implementiert und werden nicht mehr für die Entwicklung neuer Datenbankanwendungen herangezogen. Bei großen Banken und Versicherungsunternehmen sind die veralteten

Datenbankmodelle jedoch nach wie vor anzutreffen, da viele Altsysteme mit den rechenintensiven und umfangreichen Transaktion noch nicht auf die flexibleren relationalen Datenbanksysteme umgestellt werden konnten.