Kapitel 16

Data Warehouse

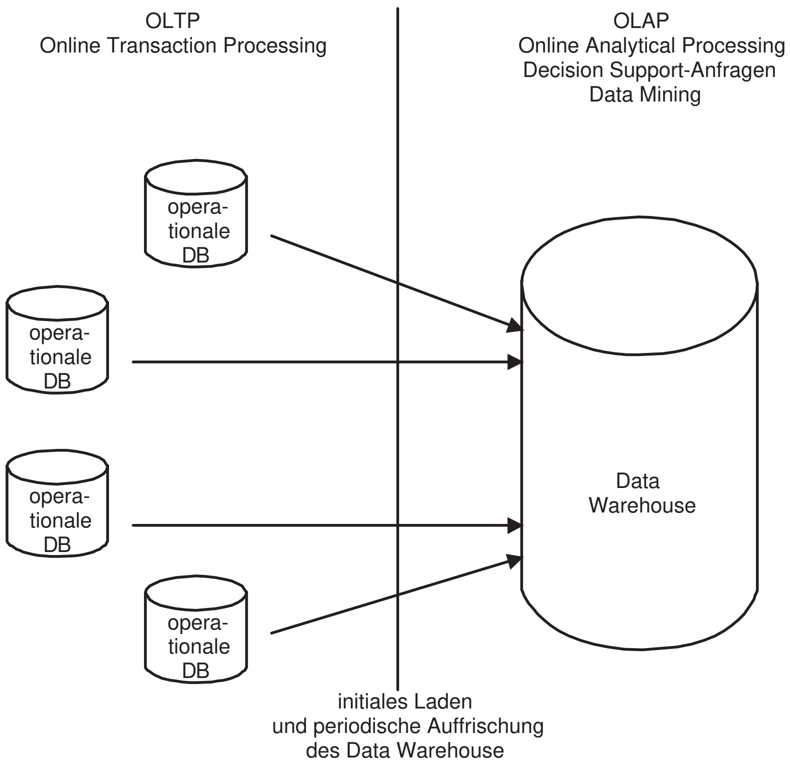

Abbildung 16.1: Zusammenspiel zwischen OLTP und OLAP

Man unterscheidet zwei Arten von Datenbankanwendungen:

- **OLTP** (*Online Transaction Processing*):
  Hierunter fallen Anwendungen wie zum Beispiel das Buchen eines Flugs in einem Flugreservierungssystem oder die Verarbeitung einer Bestellung in einem Handelsunternehmen. OLTP-Anwendungen verarbeiten nur eine begrenzte Datenmenge und operieren auf dem jüngsten, aktuell gültigen Zustand der Datenbasis.

- **OLAP** (*Online Analytical Processing*):
  Eine typische OLAP-Query fragt nach der Auslastung der Transatlantikflüge der letzten zwei Jahre oder nach der Auswirkung gewisser Marketingstrategien. OLAP-Anwendungen verarbeiten sehr große Datenmengen und greifen auf historische Daten zurück. Sie bilden die Grundlage für *Decision-Support-Systeme*.

OLTP- und OLAP-Anwendungen sollten nicht auf demselben Datenbestand arbeiten aus folgenden Gründen:

- OLTP-Datenbanken sind auf Änderungstransaktionen mit begrenzten Datenmengen hin optimiert.

- OLAP-Auswertungen benötigen Daten aus verschiedenen Datenbanken in konsolidierter, integrierter Form.

Daher bietet sich der Aufbau eines *Data Warehouse* an, in dem die für Decision-Support-Anwendungen notwendigen Daten in konsolidierter Form gesammelt werden. Abbildung 17.1 zeigt das Zusammenspiel zwischen operationalen Datenbanken und dem Data Warehouse. Typischerweise wird beim Transferieren der Daten aus den operationalen Datenbanken eine Verdichtung durchgeführt, da nun nicht mehr einzelne Transaktionen im Vordergrund stehen, sondern ihre Aggregation.

16.1 Datenbankentwurf für Data Warehouse

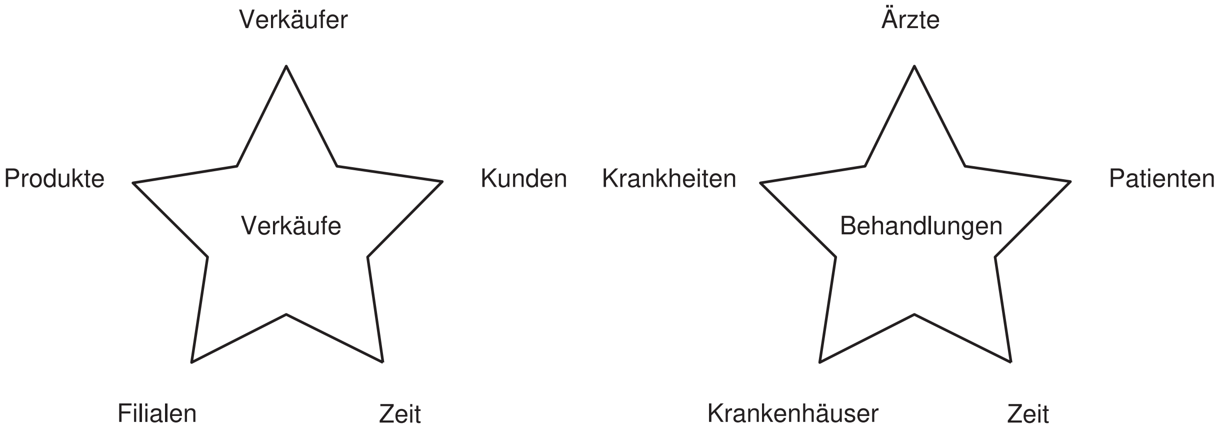

Abbildung 16.2: Sternschemata für Handelsunternehmen und Gesundheitswesen

Als Datenbankschema für Data Warehouse-Anwendungen hat sich das sogenannte *Sternschema* (engl.: *star scheme*) durchgesetzt. Dieses Schema besteht aus einer *Faktentabelle* und mehreren *Dimensionstabellen*. Abbildung 17.2 zeigt die Sternschemata für zwei Beispielanwendungen in einem Handelsunternehmen und im Gesundheitswesen.

Bei dem Handelsunternehmen können in der Faktentabelle *Verkäufe* mehrere Millionen Tupel sein, während die Dimensionstabelle *Produkte* vielleicht 10.000 Einträge und die Dimensionstabelle *Zeit*

Verkäufe					
VerkDatum	Filiale	Produkt	Anzahl	Kunde	Verkäufer
30-Jul-96	Passau	1347	1	4711	825
...	...	...	...	...	...

Filialen			
Filialenkennung	Land	Bezirk	...
Passau	D	Bayern	...
...	...	...	...

Kunden			
KundenNr	Name	wiealt	...
4711	Kemper	38	...
...	...	...	...

Verkäufer					
VerkäuferNr	Name	Fachgebiet	Manager	wiealt	...
825	Handyman	Elektronik	119	23	...
...	...	...	...	...	...

Zeit								
Datum	Tag	Monat	Jahr	Quartal	KW	Wochentag	Saison	...
...	...	...	...	...	...	...	...	...
30-Jul-96	30	Juli	1996	3	31	Dienstag	Hochsommer	...
...	...	...	...	...	...	...	...	...
23-Dec-97	27	Dezember	1997	4	52	Dienstag	Weihnachten	...
...	...	...	...	...	...	...	...	...

Produkte					
ProduktNr	Produkttyp	Produktgruppe	Produkthauptgruppe	Hersteller	...
1347	Handy	Mobiltelekom	Telekom	Siemens	...
...	...	...	...	...	...

Abbildung 16.3: Ausprägung des Sternschemas in einem Handelsunternehmen

vielleicht 1.000 Einträge (für die letzen drei Jahre) aufweist. Abbildung 17.3 zeigt eine mögliche Ausprägung.

Die Dimensionstabellen sind in der Regel nicht normalisiert. Zum Beispiel gelten in der Tabelle *Produkte* folgende funktionale Abhängigkeiten: $ProduktNr \rightarrow Produkttyp$, $Produkttyp \rightarrow Produktgruppe$ und $Produktgruppe \rightarrow Produkthauptgruppe$. In der *Zeit*-Dimension lassen sich alle Attribute aus dem Schlüsselattribut *Datum* ableiten. Trotzdem ist die explizite Speicherung dieser Dimension sinnvoll, da Abfragen nach Verkäufen in bestimmten Quartalen oder an bestimmten Wochentagen dadurch effizienter durchgeführt werden können.

Die Verletzung der Normalformen in den Dimensionstabellen ist bei Decision-Support-Systemen nicht so gravierend, da die Daten nur selten verändert werden und da der durch die Redundanz verursachte erhöhte Speicherbedarf bei den relativ kleinen Dimensionstabellen im Vergleich zu der großen (normalisierten) Faktentabelle nicht so sehr ins Gewicht fällt.

16.2 Star Join

Das Sternschema führt bei typischen Abfragen zu sogenannten *Star Joins*:

Welche Handys (d.h. von welchen Herstellern) haben junge Kunden in den bayrischen Filialen zu Weihnachten 1996 gekauft ?

```
select sum(v.Anzahl), p.Hersteller
from  Verkäufe v, Filialen f, Produkte p, Zeit z, Kunden k
where z.Saison = 'Weihnachten' and z.Jahr = 1996 and k.wiealt < 30
and   p.Produkttyp = 'Handy' and f.Bezirk = 'Bayern'
and   v.VerkDatum = z.Datum and v.Produkt = p.ProduktNr
and   v.Filiale = f.Filialenkennung and v.Kunde = k.KundenNr
group by Hersteller;
```

16.3 Roll-Up/Drill-Down-Anfragen

Der Verdichtungsgrad bei einer SQL-Anfrage wird durch die **group by**-Klausel gesteuert. Werden mehr Attribute in die **group by**-Klausel aufgenommen, spricht man von einem *drill down*. Werden weniger Attribute in die **group by**-Klausel aufgenommen, spricht man von einem *roll up*.

Wieviel Handys wurden von welchem Hersteller in welchem Jahr verkauft ?

```
select Hersteller, Jahr, sum(Anzahl)
from   Verkäufe v, Produkte p, Zeit z
where  v.Produkt = p.ProduktNr
and    v.VerkDatum = z.Datum
and    p.Produkttyp = 'Handy'
group  by p.Hersteller, z.Jahr;
```

Das Ergebnis wird in der linken Tabelle von Abbildung 17.4 gezeigt. In der Tabelle rechts oben bzw. rechts unten finden sich zwei Verdichtungen.

Durch das Weglassen der Herstellerangabe aus der **group by**-Klausel (und der **select**-Klausel) entsteht ein *roll up* entlang der Dimension *p.Hersteller*:

Wieviel Handys wurden in welchem Jahr verkauft ?

```
select Jahr, sum(Anzahl)
from   Verkäufe v, Produkte p, Zeit z
where  v.Produkt = p.ProduktNr
and    v.VerkDatum = z.Datum
and    p.Produkttyp = 'Handy'
group  by z.Jahr;
```

Durch das Weglassen der Zeitangabe aus der **group by**-Klausel (und der **select**-Klausel) entsteht ein *roll up* entlang der Dimension *z.Jahr*:

Handyverkäufe nach Hersteller und Jahr		
Hersteller	Jahr	Anzahl
Siemens	1994	2.000
Siemens	1995	3.000
Siemens	1996	3.500
Motorola	1994	1.000
Motorola	1995	1.000
Motorola	1996	1.500
Bosch	1994	500
Bosch	1995	1.000
Bosch	1996	1.500
Nokai	1995	1.000
Nokai	1996	1.500
Nokai	1996	2.000

Handyverkäufe nach Jahr	
Jahr	Anzahl
1994	4.500
1995	6.500
1996	8.500

Handyverkäufe nach Hersteller	
Hersteller	Anzahl
Siemens	8.500
Motorola	3.500
Bosch	3.000
Nokai	4.500

Abbildung 16.4: Analyse der Handy-Verkäufe nach unterschiedlichen Dimensionen

Wieviel Handys wurden von welchem Hersteller verkauft ?

```
select Hersteller, sum(Anzahl)
from   Verkäufe v, Produkte p
where  v.Produkt = p.ProduktNr and v.VerkDatum = z.Datum
and    p.Produkttyp = 'Handy'
group  by p.Hersteller;
```

Die ultimative Verdichtung besteht im kompletten Weglassen der **group-by**-Klausel. Das Ergebnis besteht aus einem Wert, nämlich 19.500:

Wieviel Handys wurden verkauft ?

```
select sum(Anzahl)
from   Verkäufe v, Produkte p
where  v.Produkt = p.ProduktNr
and    p.Produkttyp = 'Handy';
```

Durch eine sogenannte *cross tabulation* können die Ergebnisse in einem n-dimensionalen Spreadsheet zusammengefaßt werden. Abbildung 17.5 zeigt die Ergebisse aller drei Abfragen zu Abbildung 17.4 in einem 2-dimensionalen Datenwürfel *data cube*.

16.4 Materialisierung von Aggregaten

Da es sehr zeitaufwendig ist, die Aggregation jedesmal neu zu berechnen, empfiehlt es sich, sie zu materialisieren, d.h. die vorberechneten Aggregate verschiedener Detaillierungsgrade in einer Relation

Hersteller \ Jahr	1994	1995	1996	Σ
Siemens	2.000	3.000	3.500	8.500
Motorola	1.000	1.000	1.500	3.500
Bosch	500	1.000	1.500	3.000
Nokai	1.000	1.500	2.000	4.500
Σ	4.500	6.500	8.500	19.500

Abbildung 16.5: Handy-Verkäufe nach Jahr und Hersteller

abzulegen. Es folgen einige SQL-Statements, welche die linke Tabelle von Abbildung 17.6 erzeugen. Mit dem **null**-Wert wird markiert, daß entlang dieser Dimension die Werte aggregiert wurden.

```
create table Handy2DCube (Hersteller varchar(20),Jahr integer,Anzahl intege
insert into  Handy2DCube
(select p.Hersteller, z.Jahr, sum(v.Anzahl)
from Verkäufe v, Produkte p, Zeit z
where v.Produkt = p.ProduktNr and p.Produkttyp = 'Handy'
and    v.VerkDatum = z.Datum
group  by z.Jahr, p.Hersteller)
union
(select p.Hersteller, to_number(null), sum(v.Anzahl)
from  Verkäufe v, Produkte p
where v.Produkt = p.ProduktNr and p.Produkttyp = 'Handy'
group by p.Hersteller)
union
(select null, z.Jahr, sum(v.Anzahl)
from Verkäufe v, Produkte p, Zeit z
where v.Produkt = p.ProduktNr and p.Produkttyp = 'Handy'
and   v.VerkDatum = z.Datum
group by z.Jahr)
union
(select null, to_number(null), sum(v.Anzahl)
from  Verkäufe v, Produkte p
where v.Produkt = p.ProduktNr and p.Produkttyp = 'Handy');
```

Offenbar ist es recht mühsam, diese Art von Anfragen zu formulieren, da bei n Dimensionen insgesamt 2^n Unteranfragen formuliert und mit **union** verbunden werden müssen. Außerdem sind solche Anfragen extrem zeitaufwendig auszuwerten, da jede Aggregation individuell berechnet wird, obwohl man viele Aggregate aus anderen (noch nicht so stark verdichteten) Aggregaten berechnen könnte.

16.5 Der Cube-Operator

Um der mühsamen Anfrageformulierung und der ineffizienten Auswertung zu begegnen, wurde vor kurzem ein neuer SQL-Operator namens **cube** vorgeschlagen. Zur Erläuterung wollen wir ein 3-

Handy2DCube		
Hersteller	Jahr	Anzahl
Siemens	1994	2.000
Siemens	1995	3.000
Siemens	1996	3.500
Motorola	1994	1.000
Motorola	1995	1.000
Motorola	1996	1.500
Bosch	1994	500
Bosch	1995	1.000
Bosch	1996	1.500
Nokai	1995	1.000
Nokai	1996	1.500
Nokai	1996	2.000
null	1994	4.500
null	1995	6.500
null	1996	8.500
Siemens	**null**	8.500
Motorola	**null**	3.500
Bosch	**null**	3.000
Nokai	**null**	4.500
null	**null**	19.500

Handy3DCube			
Hersteller	Jahr	Land	Anzahl
Siemens	1994	D	800
Siemens	1994	A	600
Siemens	1994	CH	600
Siemens	1995	D	1.200
Siemens	1995	A	800
Siemens	1995	CH	1.000
Siemens	1996	D	1.400
...	...	...	...
Motorola	1994	D	400
Motorola	1994	A	300
Motorola	1994	CH	300
...	...	...	...
Bosch	...	...	...
...	...	...	...
null	1994	D	...
null	1995	D	...
...	...	...	...
Siemens	**null**	**null**	8.500
...	...	...	...
null	**null**	**null**	19.500

Abbildung 16.6: Materialisierung von Aggregaten in einer Relation

dimensionales Beispiel konstruieren, indem wir auch entlang der zusätzlichen Dimension *Filiale.Land* ein *drill down* vorsehen:

```
select p.Hersteller, z.Jahr, f.Land, sum(Anzahl)
from   Verkäufe v, Produkte p, Zeit z, Filialen f
where  v.Produkt    = p.ProduktNr
and    p.Produkttpy = 'Handy'
and    v.VerkDatum  = z.Datum
and    v.Filiale    = f.Filialenkennung
group  by z.Jahr, p.Hersteller, f.Land with cube;
```

Die Auswertung dieser Query führt zu dem in Abbildung 17.7 gezeigten 3D-Quader; die relationale Repräsentation ist in der rechten Tabelle von Abbildung 17.6 zu sehen. Neben der einfacheren Formulierung erlaubt der Cube-Operator dem DBMS einen Ansatz zur Optimierung, indem stärker verdichtete Aggregate auf weniger starken aufbauen und indem die (sehr große) *Verkäufe*-Relation nur einmal eingelesen werden muß.

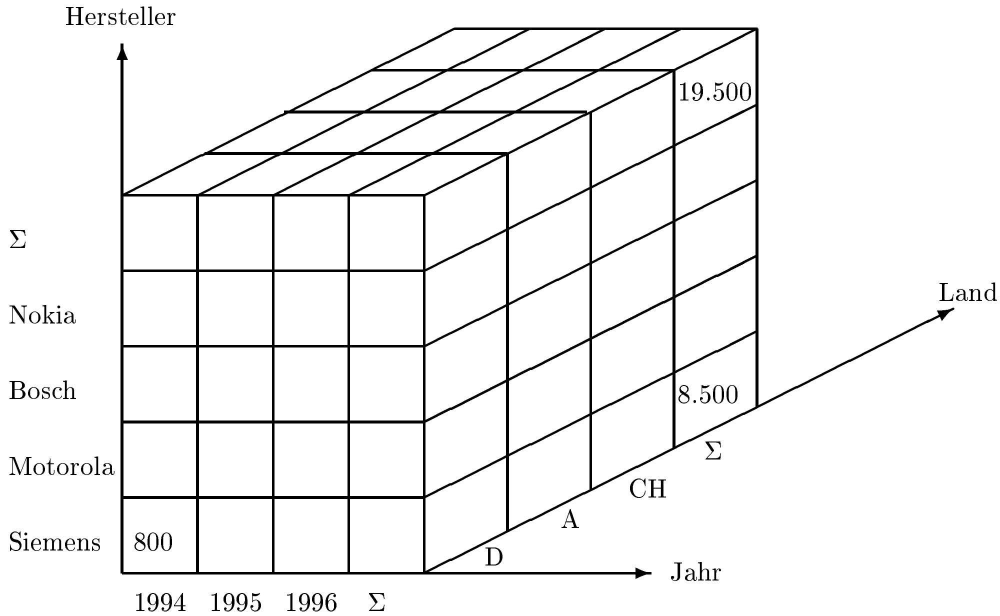

Abbildung 16.7: Würfeldarstellung der Handyverkaufszahlen nach Jahr, Hersteller und Land

16.6 Data Warehouse-Architekturen

Es gibt zwei konkurrierende Architekturen für Data Warehouse Systeme:

- **ROLAP**: Das Data Warehouse wird auf der Basis eines relationalen Datenmodells realisiert (wie in diesem Kapitel geschehen).
- **MOLAP**: Das Data Warehouse wird auf der Basis maßgeschneiderter Datenstrukturen realisiert. Das heißt, die Daten werden nicht als Tupel in Tabellen gehalten, sondern als Einträge in mehrdimensionalen Arrays. Probleme bereiten dabei dünn besetzte Dimensionen.

16.7 Data Mining

Beim *Data Mining* geht es darum, große Datenmengen nach (bisher unbekannten) Zusammenhängen zu durchsuchen. Man unterscheidet zwei Zielsetzungen bei der Auswertung der Suche:

- Klassifikation von Objekten,
- Finden von Assoziativregeln.

Bei der Klassifikation von Objekten (z. B: Menschen, Aktienkursen, etc.) geht es darum, Vorhersagen über das zukünftige Verhalten auf Basis bekannter Attributwerte zu machen. Abbildung 17.8 zeigt ein Beispiel aus der Versicherungswirtschaft. Für die Risikoabschätzung könnte man beispielsweise vermuten, daß Männer zwischen 35 und 50 Jahren, die ein Coupé fahren, in eine hohe Risikogruppe gehören. Diese Klassifikation wird dann anhand einer repräsentativen Datenmenge verifiziert.

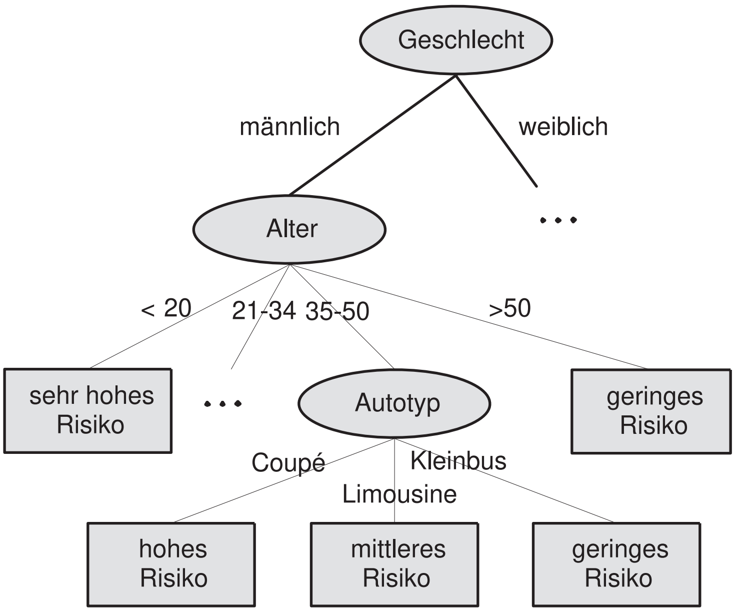

Abbildung 16.8: Klassifikation für Haftpflicht-Risikoabschätzung

Die Wahl der Attribute für die Klassifikation erfolgt benutzergesteuert oder auch automatisch durch “Ausprobieren“.

Bei der Suche nach Assoziativregeln geht es darum, Zusammenhänge bestimmter Objekte durch Implikationsregeln auszudrücken, die vom Benutzer vorgeschlagen oder vom System generiert werden. Zum Beispiel könnte eine Regel beim Kaufverhalten von Kunden folgende (informelle) Struktur haben:

> **Wenn** jemand einen PC kauft
> **dann** kauft er auch einen Drucker.

Bei der Verifizierung solcher Regeln wird keine 100 %-ige Einhaltung erwartet. Stattdessen geht es um zwei Kenngrößen:

- **Confidence:** Dieser Wert legt fest, bei welchem Prozentsatz der Datenmenge, bei der die Voraussetzung (linke Seite) erfüllt ist, die Regel (rechte Seite) auch erfüllt ist. Eine *Confidence* von 80% sagt aus, daß vier Fünftel der Leute, die einen PC gekauft haben, auch einen Drucker dazu genommen haben.
- **Support:** Dieser Wert legt fest, wieviel Datensätze überhaupt gefunden wurden, um die Gültigkeit der Regel zu verifizieren. Bei einem Support von 1% wäre also jeder Hundertste Verkauf ein PC zusammen mit einem Drucker.