Kapitel 17: Föderierte Datenbanken, Workflow-Management und Data-Warehouses

In großen Unternehmen existiert oft ein breites Spektrum, weitgehend unabhängig voneinander entwickelter und auf unterschiedlichen Softwarekomponenten basierender, datenbankgestützter Anwendungssysteme, die mehr oder weniger „Insellösungen“ darstellen. Mehr und mehr entsteht jedoch der Bedarf zur Integration solcher Insellösungen, um betriebliche Abläufe globaler optimieren zu können und Marketing- wie auch unternehmensstrategische Entscheidungen besser vorbereiten zu können. Für die sich hiermit stellende Problemstellung der Integration heterogener, strukturierter, durch Schemata beschriebener Datenquellen wird generell eine sogenannte *Wrapper-Mediator-Architektur* verfolgt, die aufbaut auf

- je einem *Wrapper* pro Datenquelle, der die eigentliche Datenquelle (z.B. eine relationale Datenbank) kapselt und deren Daten oder Operationen in aufbereiteter, zur Integration vorbereiteter Form bereitstellt, und
- mindestens einem *Mediator* (Broker, Föderations-Integrator), der die eigentliche Integration der von den Wrappern bereitgestellten Daten oder Operationen leistet und den Clients integrierte Daten und/oder Operationen anbietet.

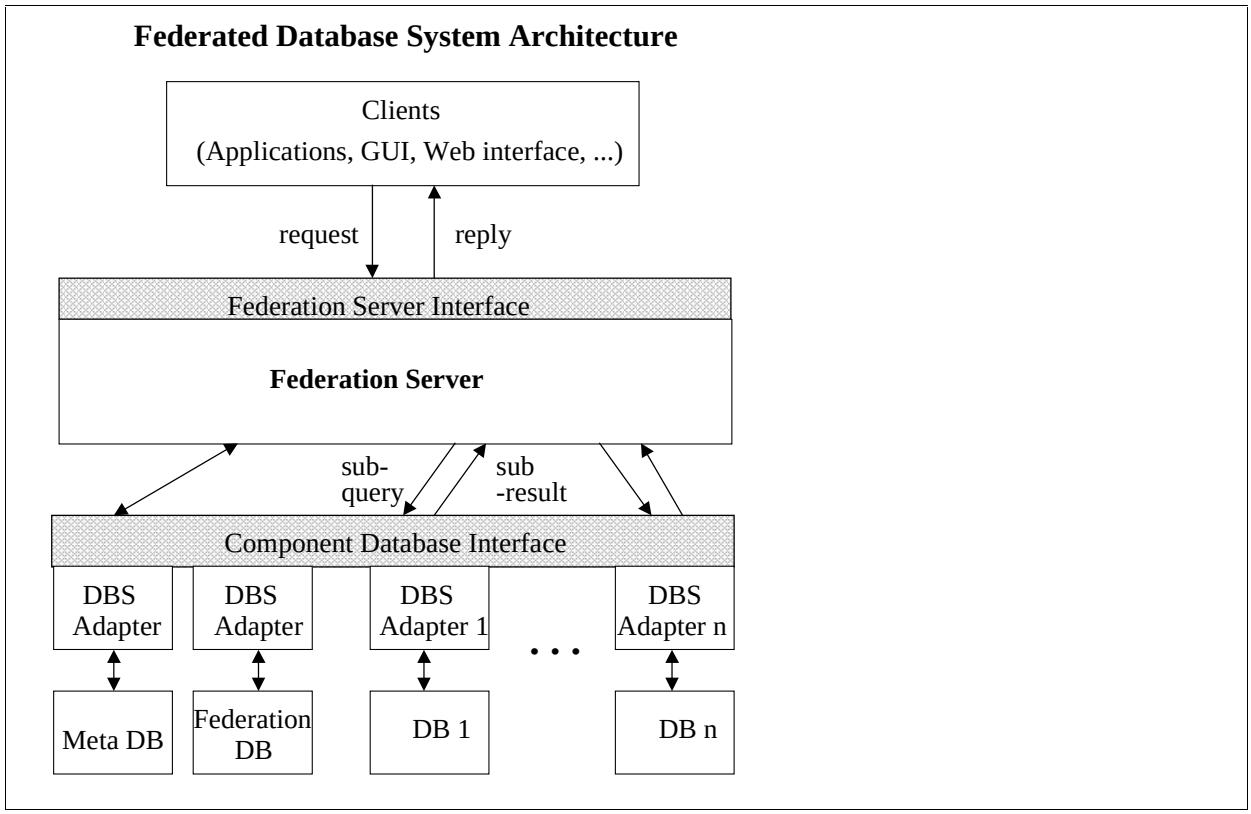
Je nach „Tiefe“ der Integration und Aufgabenteilung zwischen Wrappern, Mediator und ggf. den Clients ergeben sich verschiedene Familien von Integrationsarchitekturen für strukturierte Daten, die im folgenden skizziert werden.

17.1 Föderierte Datenbanken

Bei der Architektur einer aus mehreren *Komponenten-Datenbanken bestehenden Datenbank-Föderation* stellen Wrapper – hier auch *Datenbank-Adapter* genannt – ein aufbereitetes, zur Integration vorbereitetes Schema der jeweiligen Komponenten-Datenbanken zur Verfügung, lassen aber alle Daten unter der Kontrolle des jeweiligen Komponentensystems.

Häufig wird nur ein Teil der jeweiligen zur Integration benötigt bzw. freigegeben; daher wird das vom Wrapper bereitgestellte Schema häufig auch *Export-Schema* genannt. Dieses enstpricht quasi einer View auf die eigentlichen Daten des Komponenten-Systems. Der Wrapper muß daher zusätzlich eine Abbildung dieser View auf das Original-Schema der Komponenten-Datenbank, im wesentlichen also eine enstprechende Familie vordefinierter Queries, beinhalten.

Der Mediator – hier auch *Föderations-Server* genannt – integriert die exportierten „lokalen“ Schemata der Komponenten in ein globales Schema, das in einer Meta-Datenbank abgelegt wird, und unterstützt Queries und Update-Operationen auf den gemäß dieses globalen Schemas virtuell integrierten Daten. Dazu sind globale Queries in Subqueries zu zerlegen, die jeweils auf den exportierten Views ausgeführt werden, indem die jeweiligen Datenbank-Adapter die Viewdefinitionen in den Subqueries substitutieren und die resultierenden Queries ausführen. Der Föderations-Server übernimmt auch die Integration der von den Komponenten-Systemen bzw. Datenbank-Adaptern gelieferten Teilresultate zu einem globalen Resultat. Um temporäre Zwischenresultate weiterverarbeiten (z.B. aggregieren) zu können, kann dem Föderations-Server ggf. eine spezielle Föderations-Datenbank zur Verfügung stehen.



Die größten Schwierigkeiten im Aufbau einer Datenbankföderation liegen in

1) der *Heterogenität der Komponentendatenbanken* bzgl. ihrer Struktur und vor allem ihrer Schemasemantik und
2) der *Autonomie der Komponentensysteme*, die neben globalen Queries und Transaktionen weiterhin rein lokale Anwendungen bedienen und Beeinträchtigungen der lokalen Last hinsichtlich Leistung und Verfügbarkeit so weit wie möglich ausschließen möchten (ohne jedoch globale Inkonsistenzen der Daten zu riskieren).

Ersteres führt auf das Problem der *Schemaintegration*, letzteres auf Probleme mit der Behandlung *globaler Transaktionen*.

17.1.1 Schemaintegration

Bei der Integration unabhängig voneinander entworfener Komponenten-Schemata treten verschiedene Arten von strukturellen und vor allem semantischen Konflikten auf, die auf Föderations-Ebene zu bereinigen sind:

- *Strukturkonflikte* entstehen, wenn dieselbe Information in verschiedenen Komponenten mit unterschiedlichen Datenmodellkonstrukten repräsentiert wird.
  Beispiel:
  DB1: VerwAngestellte (PersNr, Name, AbtNr, AbtBez)
  WissAngestellte (PersNr, Name, AbtNr, AbtBez)
  DB2: Angestellte (PersNr, Name, PersKategorie, AbtNr)
  Abteilungen (AbtNr, AbtBez)
  In DB1 werden Abteilungen als Attribute von Angestellten repräsentiert, in DB2 in Form einer eigenen Relation. In DB1 sind Verwaltungsangestellte und wissenschaftliche Angestellte in zwei Relationen partititioniert, in DB2 sind alle Angestellten in einer Relation und die Spezialisierung ist durch ein zusätzliches Attribut repräsentiert.

In ausdrucksstärkeren Datenmodellen, insbesondere objektorientierten Datenmodellen oder semantischen Datenmodellen à la ERM, sind derartige Strukturkonflikte tendenziell noch augenfälliger, wenn z.B. dieselbe Information einmal als Attribut und einmal als Objektklasse bzw. Entitymenge repräsentiert ist.

- *Namenskonflikte* entstehen, wenn dieselbe Informationseinheit in verschiedenen Komponenten unterschiedlich benannt ist (*Synonyme*) oder gleich benannte Informationseinheiten unterschiedliche Bedeutung haben (*Homonyme*).
  Beispiel:
  DB1: Teile (TeileNr, TeileBez, ..., Einkaufspreis)
  DB2: Teile (TeileNr, TeileBez, ..., Preis)
  DB3: Teile (TeileNr, TeileBez, ..., Preis)
  In DB1 und DB2 (z.B. separaten Datenbanken für Lagerverwaltung und Buchführung) könnten Einkaufspreis und Preis Synonyme sein, wohingegen die Preise von DB2 und DB3 (letzteres z.B. eine Datenbank für den Vertrieb) Homonyme wären, wenn der Preis in DB3 ein Verkaufspreis ist.

- *Datentypkonflikte* liegen vor, wenn dieselbe Informationseinheit mit unterschiedlichen Datentypen repräsentiert wird.
  Beispiel:
  DB1: Teile (..., Preis, ...) als Float oder Number
  DB2: Teile (..., Preis, ...) als Varchar

- *Codierungskonflikte* liegen vor, wenn dieselbe Informationseinheit mit demselben Datentyp auf unterschiedliche Weise codiert wird, z.B. aufgrund landes- oder sprachspezifischer Konventionen.
  Beispiel:
  DB1: Bestellungen (BestNr, ..., BestDatum, ...)
  mit Datum als Varchar in der Form "5.2.1999"
  DB2: Order (OrderNo, ..., OrderDate, ...)
  mit OrderDate als Varchar in der Form "2/5/99"

- *Dimensionskonflikte* liegen vor, wenn dieselbe Informationseinheit in unterschiedlichen Maßeinheiten angegeben ist.
  Beispiel:
  DB1: Angestellte (PersNr, ..., Gehalt, ...)
  mit Gehalt in DM pro Monat als Bruttogehalt
  DB2: Employees (EmpNo, ... Salary, ...)
  mit Salary in $ pro Jahr nach Abzug der Steuern

Bei der Schemaintegration sind daher - sehr grob gesprochen - folgende Schritte zur Bereinigung dieser Konfliktarten notwendig:
- Strukturvereinheitlichungen durch Umstrukturierung der exportierten Daten
- Namensanpassungen durch Umbenennung von Homonymen und Synonymen
- Festlegung von Konvertierungsprozeduren für Datenwerte zur Bereinigung von Datentyp-, Codierungs- und Dimensionskonflikten

Bei der abschließenden Zusammenfassung der bereinigten Komponentenschemata sind dann ggf. Generalisierungen zu bilden, wenn spezialisierte Daten desselben virtuellen "Supertyps" aus verschiedenen Datenbanken vereinigt werden sollen.

Beispiel für die Vorgehensweise bei der Schemaintegration (zur Illustration der Problematik):

DB1 (Bibliotheksdatenbank):
- Buch (ISBN, Name, Verlag, BibliotheksName)
- Autor (Name, Adresse)
- Bibliothek (BibName, Adresse)
- Deskriptoren (ISBN, Bezeichnung)
- Autorenschaft (ISBN, AutorenName)

DB2 (Literaturdatenbank):
- Publikationen (Name, Titel, Verlag)
- Stichwörter (Code, Fachgebiet)
- Katalog (Titel, Verlag, Code)

1) Korrespondierende Informationseinheiten bestimmen:
   DB1.Autor.Name entspricht DB2.Publikationen.Name
   DB1.Buch.Name entspricht DB2.Publikationen.Titel bei Büchern
   DB1.Deskriptoren.Bezeichnung entspricht DB2.Stichwörter.Fachgebiet
   usw.
2) Umbenennungen:
   DB1.Buch.Name in DB1.Buch.Titel umbenennen
   DB1.Autor.Name und DB2.Publikationen.Name in AutorenName umbenennen
   DB2.Stichwörter.Fachgebiet in DB2.Stichwörter.Bezeichnung umbenennen
   usw.
3) Generalisierungen:
   DB1.Buch als Spezialisierung von DB2.Publikationen interpretieren
   oder eine gemeinsame Generalisierung bilden

Ein mögliches, aus den beiden Komponenten DB1 und DB2 erstelltes föderatives Schema (das nur einen Teil der Information übernimmt und integriert) wäre z.B.:

FDB:
- Publikationen (Titel, Verlag)
- Bücher (Titel, Verlag, ISBN)
- Autoren (Titel, Verlag, Autorenname)
- Stichwörter (Titel, Verlag, Bezeichnung)

Um auf die integrierten Daten über das FDB-Schema zugreifen zu können, benötigt der Föderations-Server noch die entsprechenden Transformationen, die quasi die Relationen der FDB als Views auf die Relationen von DB1 und DB2 definieren:

FDB.Publikationen (Titel, Verlag) :=
π[Name, Verlag] (DB1.Buch) $\cup$ π[Titel, Verlag] (DB2.Publikationen)

FDB.Bücher (Titel, Verlag, ISBN) :=
π[Name, Verlag, ISBN] (DB1.Buch)

FDB.Autoren (Titel, Verlag, Name) :=
π[Name, Verlag, AutorenName] (DB1.Autorenschaft |X| DB1.Buch) $\cup$
π[Titel, Verlag, Name] (DB2.Publikationen)

FDB.Stichwörter (Titel, Verlag, Bezeichnung) :=
π[Name, Verlag, Bezeichnung] (DB1.Buch |X| DB1.Deskriptoren) $\cup$
π[Titel,Verlag,Fachgebiet] (DB2.Publikationen |X| DB2.Katalog |X| DB2.Stichwörter)

17.1.2 Globale Transaktionen

Für die Atomarität globaler Transaktionen muß ein verteiltes Commit-Protokoll durchgeführt werden. Da praktisch alle kommerziell relevanten Komponenten-DBS das *standardisierte 2PC* (nach X/Open XA bzw. CORBA OTS) unterstützen, ist dies trotz der Heterogenität der beteiligten System relativ einfach zu realisieren. Potentiell problematisch kann dabei aber sein, dass das 2PC u.U. blockierend ist und sich generell lange Wartezeiten in einem Komponentensystem auf die Leistung in einem anderen Komponentensystem negativ auswirken können. Dies kann unter dem Gesichtspunkt der Autonomie für ein Komponentensystem unakzeptabel sein. Die einzige Alternative besteht darin, verteilte Transaktionen zu vermeiden und stattdessen mehrere Transaktionen auf einzelnen Komponentensystemen in einen übergreifenden Workflow einzubetten. Beim Abbruch eines Workflows sind dann u.U. *Kompensationstransaktionen* für die bereits mit Commit beendeten Transaktionen notwendig (z.B. die Stornierung einer Buchung u.ä.). Dies setzt jedoch eine explizite Definition und Programmierung der Kompensationstransaktionen voraus.

Die Verwendung des standardisierten 2PC garantiert nur die Atomarität globaler Transaktionen, nicht jedoch deren globale Serialisierbarkeit. A priori folgt aus der lokalen Serialisierbarkeit in jedem einzelnen Komponentensystem eben keineswegs direkt die Serialisierbarkeit der globalen Transaktionen.
Beispiel ($R_i^{(k)}(x)$ bezeichne eine (Lese-) Aktion der Transaktion Ti auf dem Objekt x, das zum Komponentensystem DBk gehört):

$$R_1^{(1)}(x)\ W_1^{(1)}(x)\ R_2^{(1)}(x)\ W_2^{(1)}(x)\ R_2^{(2)}(y)\ W_2^{(2)}(y)\ R_1^{(2)}(y)\ W_1^{(2)}(y)$$

Die Projektion dieses Schedules auf DB1 ist:

$$R_1^{(1)}(x)\ W_1^{(1)}(x)\ R_2^{(1)}(x)\ W_2^{(1)}(x)$$

Die Projektion des Schedules auf DB2 ist:

$$R_2^{(2)}(y)\ W_2^{(2)}(y)\ R_1^{(2)}(y)\ W_1^{(2)}(y)$$

Offensichtlich sind beide Projektionen (lokal) serialisierbar; der globale Schedule ist aber nicht serialisierbar (da sich die beiden lokalen Serialisierungsreihenfolgen widersprechen).

Man benötigt also entweder zusätzliche Concurrency-Control-Maßnahmen auf der föderativen Ebene oder muß zusätzliche Einschränkungen der lokalen Schedules (bzw. Projektionen globaler Schedules) erzwingen. Falls alle beteiligten Komponentensysteme nur lokale Schedules erzeugen, die konflikt-serialisierbar sind und deren lokale Serialisierungsreihenfolge mit der Reihenfolge der (globalen) Commits übereinstimmt (*Commit-Ordnungs-Serialisierbarkeit*), sind auch alle daraus resultierenden globalen Schedules konfliktserialisierbar. Für Datenbanksysteme, die konsequent striktes 2PL verwenden (für Schreib- und Leseoperationen), ist diese Eigenschaft erfüllt. Leider verwenden kommerzielle Datenbanksysteme, darunter auch Oracle, pragmatisch abgewandelte Concurrency-Control-Protokolle, die die Eigenschaft der Commit-Ordnungs-Serialisierbarkeit nicht erfüllen.

Man beachte, dass die Eigenschaft der lokalen Konflikt-Serialisierbarkeit alleine nicht genügt, um globale Serialisierbarkeit sicherzustellen, selbst wenn globale Transaktionen nur seriell laufen. Schuld daran ist der potentielle Einfluß rein lokaler Transaktionen, die ohne den "Umweg" über den Föderations-Server direkt auf einem der Komponentensysteme laufen.
Beispiel (mit globalen Transaktionen T1, T2 und einer lokalen Transaktion T3):

$$W_3^{(1)}(x)\ R_1^{(1)}(x)\ R_1^{(2)}(z)\ W_1^{(2)}(z)\ R_2^{(1)}(y)\ R_2^{(2)}(z)\ W_2^{(2)}(z)\ W_3^{(1)}(y)$$

Die Projektion auf DB1,

$$W_3^{(1)}(x)\ R_1^{(1)}(x) \qquad R_2^{(1)}(y) \qquad W_3^{(1)}(y)\ ,$$

ist konflikt-serialisierbar mit der Serialisierungsreihenfolge T2 < T3 < T1.
Die Projektion auf DB2,

$$R_1^{(2)}(z)\ W_1^{(2)}(z) \qquad R_2^{(2)}(z)\ W_2^{(2)}(z)\ ,$$

ist konflikt-serialisierbar mit der Serialisierungsreihenfolge T1 < T2, so daß der globale Schedule nicht (konflikt-) serialisierbar ist.

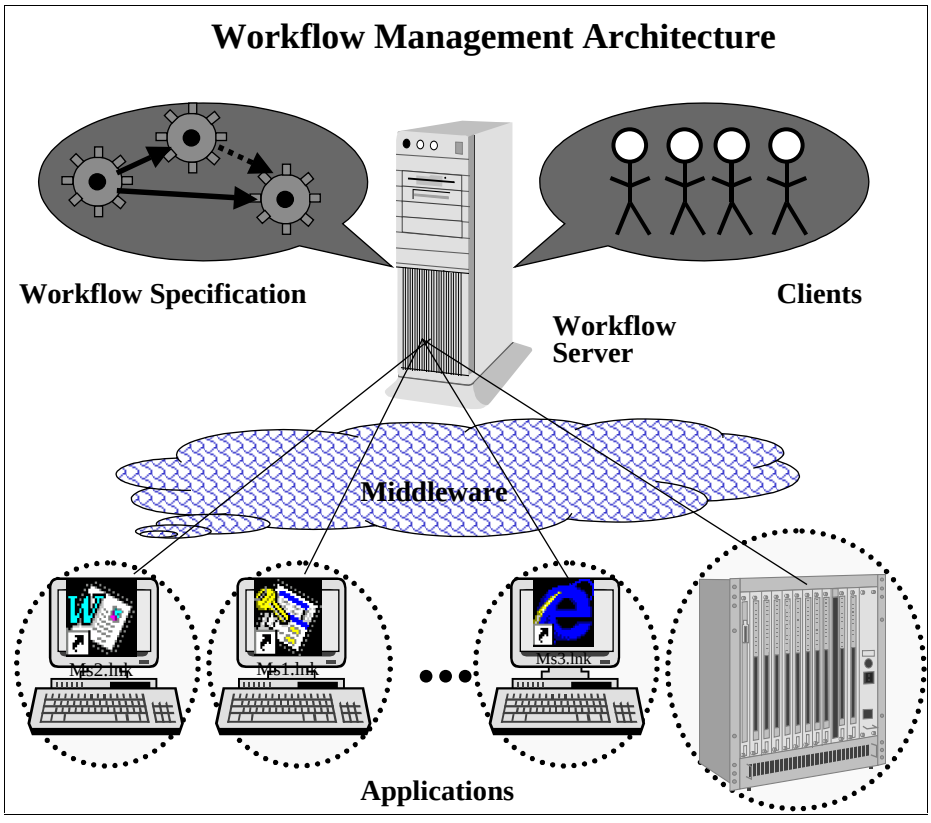
17.2 Workflow-Management und Middleware

Bei der Integration über ein Workflow-Management-System bleiben die Daten der Komponentensysteme gekapselte Objekte in dem Sinne, dass auf sie nur über die lokal existierenden Applikationen der Komponentensysteme zugegriffen werden kann. Das Workflow-Management-System steuert den Kontroll- und Datenfluß globaler Abläufe durch Interpretation einer Workflow-Spezifikation. In diese Workflows sind Applikationsaufrufe eingebettet.

Damit auf diese Weise ein möglichst breites Spektrum verschiedenartiger Applikationen auf unterschiedlichen Plattformen integriert werden kann, müssen Applikation durch entsprechende Wrapper erst workflow- bzw. integrationsfähig gemacht werden und werden dann über entsprechend standardisierte "Middleware" aufgerufen. Die in der Praxis etabliertesten Middleware-Architekturen sind

- CORBA (Common Object Request Broker Architecture), spezifiziert vom Industriekonsortium und Standardisierungsgremium OMG (Object Management Group) und
- COM bzw. DCOM (Distributed Component Object Model) von Microsoft.

Beide erwarten für jede aufrufbare Applikation eine IDL-Schnittstelle (Interface Definition Language), die ggf. vom entsprechenden Wrapper zu realisieren ist und realisieren auf dieser Basis - ggf. über einen sogenannten Object Request Broker (ORB) - ortstransparente Methodenaufrufe auf beliebig verteilten, heterogen (z.B. in verschiedenen Programmiersprachen) implementierten Objekten. Aus den existierenden Applikationen werden somit "Business Objects", die in standardisierter Form aufrufbar und damit auch leichter wiederverwendbar sind.



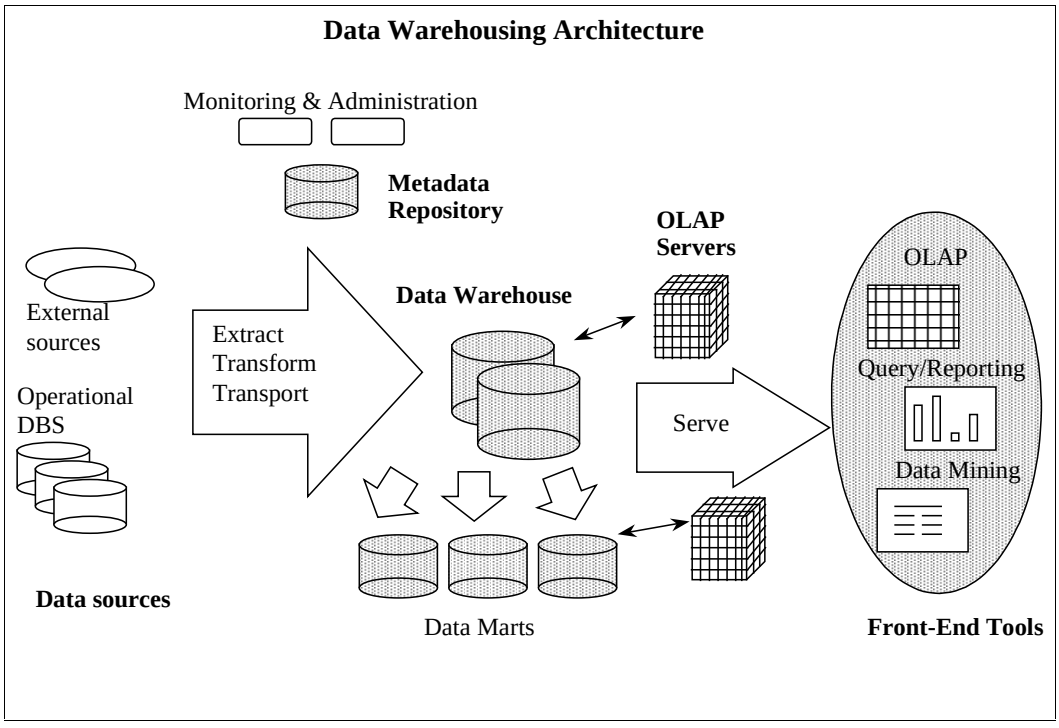
17.3 Data-Warehouses

Wenn die Integration verschiedener Datenbanken ausschließlich entscheidungsunterstützenden Queries und Datenanalysen mit rein lesendem Zugriff dient, bietet sich der Aufbau eines sogenannten Data-Warehouses an, in das periodisch oder kontinuierlich Daten aus verschiedenen Datenquellen kopiert und dabei bedarfsgerecht umstrukturiert werden. Im Gegensatz zur Datenbank-Föderation werden also wirkliche Daten und nicht nur Schemata integriert. Dennoch ist dieser Ansatz einfacher realisierbar als ein vollständiger Föderations-Server, da

- die Daten im Data-Warehouse nur für Lesezugriffe zur Verfügung stehen müssen,
- die Autonomie der zugrundeliegenden Datenquellen hinsichtlich lokaler Performance und Verfügbarkeit weitgehend unangetastet bleibt und
- das Schema des Data-Warehouses auf spezifische Datenanalysen - *sogenannte OLAP-Anwendungen (Online Analytical Processing)* zugeschnitten werden kann.

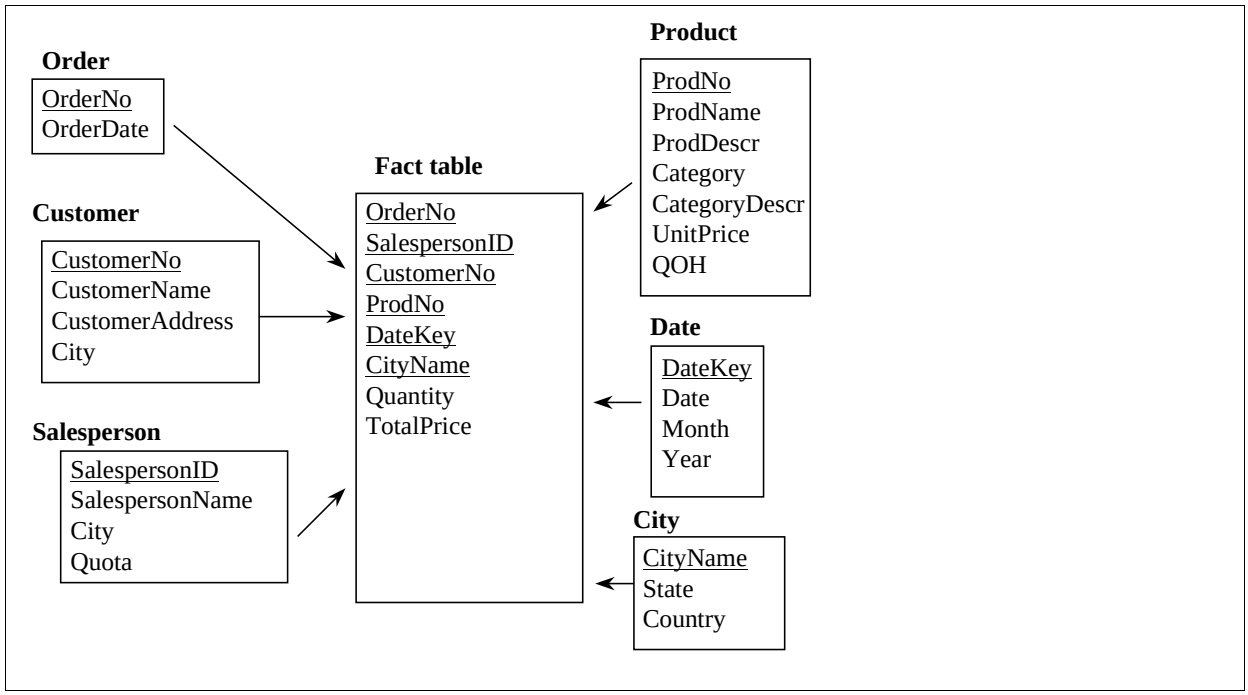
Architektur, Aufbau und Pflege eines Data-Warehouses umfassen also die folgenden Komponenten bzw. Schritte:

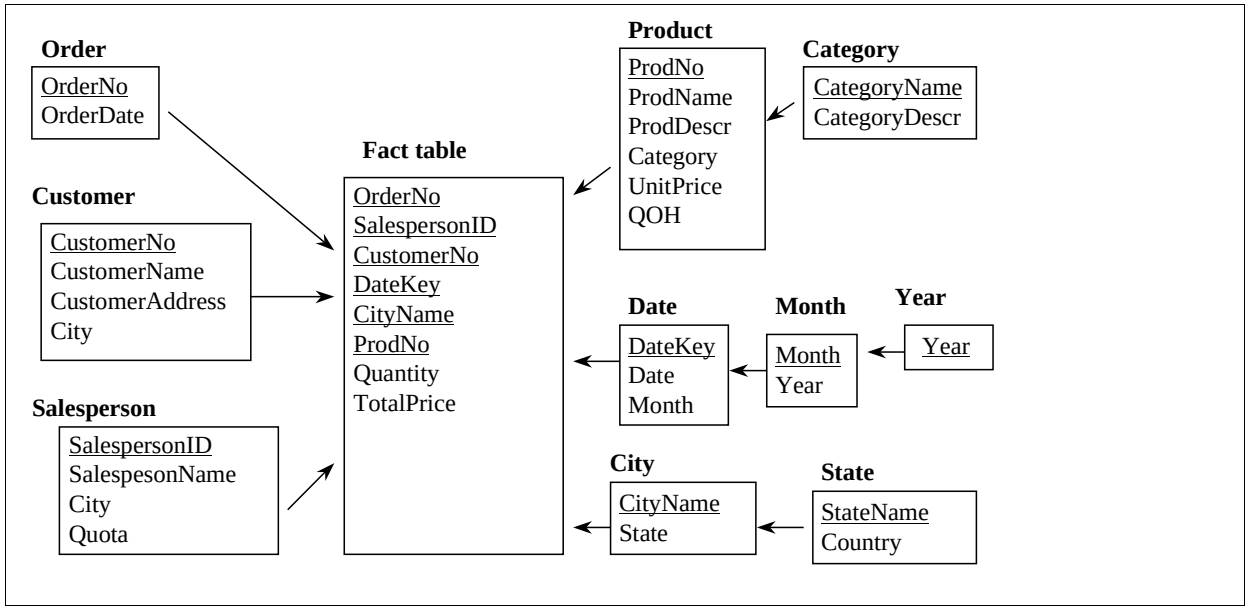
- Entwurf eines Schemas für das Data-Warehouse
- Vorbereitung der Datenextraktion aus den Datenquellen
- Bereinigung, Transformation und Kopieren der Daten in das Data-Warehouse
- Betrieb eines OLAP-Servers auf den Daten des Data-Warehouse
  (ggf. Extraktion von Teilmengen in spezielle OLAP-Tools der Clients)
- Belieferung spezialisierter, kleinerer Data-Marts
- Physischer Datenbankentwurf und Tuning des Data-Warehouses
  (Indexauswahl, Einsatz von Bitmap-Indizes, Join-Indizes, View-Materialisation, usw.)



Als Richtlinie für den konzeptionellen Datenbankentwurf des Data-Warehouses werden häufig sogenannte Stern- oder Schneeflocken-Schemata propagiert. Einem Stern-Schema liegt eine - in der Regel sehr große - sogenannte *Fakten*-Relation als "Zentrum" zugrunde, die zu einer Anzahl kleinerer, sogenannter *Dimensions*-Relationen jeweils eine Fremdschlüsselbeziehung hat; die Dimensions-Relationen haben untereinander keine Fremdschlüsselbeziehungen. Ein Schneeflocken-Schema verallgemeinert diesen Ansatz dahingehend, daß die Dimensions-Relationen Fremdschlüsselbeziehungen zu hierarchisch übergeordneten Dimensions-Relationen haben können.

Beispiele für ein Stern-Schema und ein Schneeflocken-Schema:





Typische Queries auf einem Data-Warehouse sind von der Art:

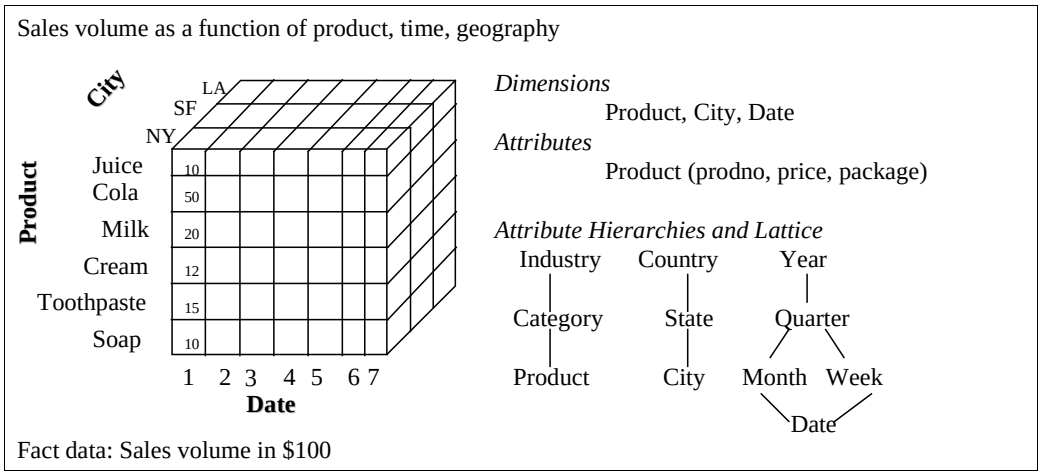
- Wie gut verkaufen sich Badeartikel im Saarland während der Wintermonate?
- Welche Entwicklung haben die Umsätze mit PC-Zubehör in den letzten drei Jahren genommen – bundesweit und aufgeschlüsselt nach Regionen? Wie sehen diese Trends bei bestimmten Arten von Zubehör aus (Peripherie, Software, Spiele, usw.)?
- Welches sind unsere umsatzstärksten Kunden? Mit welchen Produkten verdienen wir das meiste Geld? (bundesweit, in spezifischen Regionen, usw.)
- Um wieviel (und warum) sind die Verkaufszahlen für Badeartikel im Saarland schlechter als in Schleswig-Holstein?

Strukturell ähnliche Queries treten auch in nichtbetrieblichen Anwendungen auf, z.B. medizinischen Anwendungen:

- Wie hängt die Dauer der stationären Klinikaufenthalte von den verschriebenen Medikamentgruppen ab, aufgeschlüsselt nach Geschlecht, Altersgruppe und Berufsgruppe der Patienten?

Derartige Queries und Analysen auf den Daten eines Data-Warehouse erfolgen vielfach mit speziellen OLAP-Tools, die aus einer Client-Komponente zur Datenvisualisierung und einem OLAP-Query-Server auf der Seite des Data-Warehouse bestehen. Sehr häufig werden die Daten dabei als mehrdimensionaler Würfel (engl.: Data Cube) dargestellt, dessen Zellen verschiedene Werte der Fakten-Relation für die entsprechenden Werte der Dimensions-Relationen enthalten. Auf solchen Würfeln ist ein Spektrum von intuitiv einfach handhabbaren Basisoperationen ausführbar, die vom OLAP-Server in entsprechende SQL-Queries umgesetzt werden müssen:

- *Roll-up*: Vergröbernde Aggregationen durch Gruppierung in einer oder mehreren Dimensionen mit Berechnung von Aggregationsfunktionen wie Sum, Avg, usw.
- *Drill-down*: Verfeinern der Werte(gruppen) einer oder mehrerer Dimensionen mit entsprechender Aufschlüsselung der aggregierten Faktenwerte
- *"Pivot", "Slice&Dice"*: Projektionen des Würfels auf bestimmte Dimensionen mit entsprechender Werteaggregation ("Drehen und Schneiden des Würfels")



Solche an der Würfel-Metapher orientierte Operationen können häufig mit OLAP-spezifischen Erweiterungen von SQL kombiniert werden. Derartige Erweiterungen sind u.U. auch OLAP-Server-intern zur effizienteren Auswertung der OLAP-Queries sinnvoll.

Beispiele:

- Zusätzliche Aggregationsfunktionen wie
  - *Median* (Fakten.Attribut),
  - *Mode* (Fakten.Attribut),
  - *MovingAverage* (Fakten.Attribut, 3 Rows) und *MovingSum* (...) bzw. *MovingAverage* (Fakten.Attribut, 3 Months)

  usw.
- Ranking bezüglich aggregierter Werte, z.B.
  - *Rank* (...)
    zur Bestimmung des Rangs eines Kunden, z.B. der 10 umsatzstärksten Kunden
  - *Tile* (...)
    z.B. zur Bestimmung der meistverkauften Produkte, die zusammen einen bestimmten Prozentsatz des Gesamtumsatzes ausmachen
- Mehrfachgruppierung und entsprechende Aggregationen:
  Group By A1, A2, ..., Am Cube entspricht mehreren Gruppierungen
  Group By A1; *Group By A1, A2*; ...; *Group By A1, A2, ..., Am*
  in einer einzigen Query

Ergänzende Literatur zu Kapitel 17

S. Conrad, Föderierte Datenbanksysteme, Springer-Verlag, 1997

T. Özsu, P. Valduriez, Principles of Database Systems, 2nd Edition, Prentice-Hall, 1999

A.K. Elmagarmid, M. Rusinkiewicz, A. Sheth (Editors), Management of Heterogeneous and Autonomous Database Systems, Morgan Kaufmann Publishers, 1998

C. Batini, M. Lenzerini, S.B. Navathe, A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys Vol.18 No.4, 1986

Y. Breitbart, H. Garcia-Molina, A. Silberschatz, Overview of Multidatabase Transaction Management, VLDB Journal Vol.1 No.2, 1992

A. Dogac, L. Kalinichenko, T. Özsu, A. Sheth (Editors), Advances in Workflow Management and Interoperability, NATO Advanced Study Institute, Springer-Verlag, 1998

S. Jablonski, C. Bussler, Workflow Management – Modeling Concepts, Architecture, and Implementation, International Thomson Computer Press, 1996

C. Mohan, Workflow Management in the Internet Age, Tutorial, EDBT Summer School, La Baule, France, 1999, http://rodin.inria.fr/~mohan/workflow.ps

IEEE Data Engineering Bulletin Vol.21 No.3, September 1998, Special Issue on Interoperability

S. Chaudhuri, U. Dayal, An Overview of Data Warehousing and OLAP Technology, ACM SIGMOD Record Vol.26 No.1, 1997

W.H. Inmon, Building the Data Warehouse, 2nd Edition, John Wiley & Sons, 1996

R. Kimball, The Data Warehouse Toolkit, John Wiley & Sons, 1996

J. Gray, A. Bosworth, A. Layman, H. Pirahesh, Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals, IEEE International Conference on Data Engineering, New Orleans, 1996

IEEE Data Engineering Bulletin Vol.20 No.1, March 1997, Special Issue on Supporting On-Line Analytical Processing