Extensible Markup Language (XML)

XML: Definition von Datentypen

- XML erlaubt die Definition von Dokumenttypen
  - XML Schema Definition (XSD)
  - Document Type Definition (DTD)
- dadurch wird die Syntax einer speziellen Beschreibungssprache (für eine Klasse von Dokumenten) festgelegt
- Festlegung der Semantik eines Dokuments nur eingeschränkt machbar
  - Datentypen in XML Schema
  - Abbildung von XML-Dokumenten auf andere Dokumente

XML Schema

- XML Schema ist wie DTD eine Sprache zur Definition von Dokumenttypen
- XML Schema ist gleichzeitig eine XML-Anwendung
  - ist Instanz (notiert als XML-Dokument)
  - beschreibt Typ (Definition von XML-Dokumentstrukturen)
- legt fest, welche Elemente, Attribute und Verschachtelungsmöglichkeiten es gibt
- Schema sagt nichts über die Bedeutung des Dokuments
  - Beschreibung des Schemas per Text („sprachliche Semantik“)
  - Implementierung einer Software, die Dokumente liest/schreibt („operative Semantik“)
- Deklaration

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema
   xmlns:xs=http://www.w3.org/2001/XMLSchema>
   …
</xs:schema>
```

  - xs: Bezeichner für den Namensraum
    - kennzeichnet vorgegebene Sprachelemente von XSD
    - vs. selbst deklarierte Elemente (ohne Präfix)
  - xmlns: XML namespace
    - URI dient nicht als Verweis auf eine Internetadresse, sondern definiert lediglich den Namensraum für die verwendeten Element- und Attribut-Bezeichner
- Anwendungsbeispiel ohne Zielnamespace
  - XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="adresse">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="ort" type="xs:string"/>
            <xs:element name="plz" type="xs:string"/>
            <xs:element name="strasse" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

- XML-Dokument

```
<adresse
   xmlns:xsi=„http://www.w3.org/2001/XMLSchema-instance“
   xsi:noNamespaceSchemaLocation="adresse.xsd">
   <ort>Essen</ort>
   <plz>45141</plz>
   <strasse>Universitaetsstrasse 9</strasse>
</adresse>
```

- Anwendungsbeispiel mit Zielnamespace
  - XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
   xmlns:adr="http://www.example.org/adresse"
   targetNamespace="http://www.example.org/adresse">
   <xs:element name="adresse">
      …
   </xs:element>
</xs:schema>
```

  - XML-Dokument

```
<adr:adresse
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:adr="http://www.example.org/adresse"
   xsi:SchemaLocation="http://www.example.org/adresse adresse.xsd">
   …
</adr:adresse>
```

- Namespaces
  - ohne Namespaces kann es zu Mehrdeutigkeiten kommen (z.B. verschiedene Adressarten)
  - Anwendungsbeispiel für Namespaces:

```
<dokument
   xmlns:adr="http://www.bis1pa.org/adresse"
   xmlns:comp="http://www.somwhereelse.org/computer">
   …
   <adr:adresse>
      <strasse>Hauptstr. 7</strasse>
      <plz>45219</ort>
      <ort>Essen</ort>
   </adr:adresse>
      …
   <comp:adresse>
      <ram>7e25a4be</ram>
      <reg1>45</reg1>
      <reg2>c2</reg2>
   </comp:adresse>
   …
</dokument>
```

- Datentypen
  - es wird zwischen einfachen (atomic) und komplexen (derived) Datentypen unterschieden
    - einfacher Datentyp

```
<xs:element name=„Wohnort“ type=„ort“/>
```

      - Bsp: string, time, date, boolean, double, float
      - einfache Datentypen enthalten keine Attribut- oder Elementdeklarationen

- komplexer Datentyp

```
<xs:element name="adresse">
   <xs:complexType name="lettertype" mixed="false">
      <xs:sequence>
         <xs:element name="strasse" type="xs:string"/>
         <xs:element name="nr" type="xs:positiveInteger"/>
         <xs:element name="plz" type="xs:positiveInteger"/>
       </xs:sequence>
   </xs:complexType>
</xs:element>
```

  - xs:sequence: jedes Element kann keinmal, einmal oder mehrmals auftauchen (occurs-Defaultwert 1), Reihenfolge ist wichtig
  - xs:choice: eine Alternative zur Auswahl
  - xs:all: Reihenfolge beliebig, jedes Element darf maximal einmal auftreten (min- und maxOccurs dürfen nur den Wert 0 oder 1 annehmen)

- auf dieser Basis können weitere benutzerdefinierte Datentypen abgeleitet werden
  - Restriction: Facets schränken gültige Werte ein

```
<xs:simpleType name=„Name“>
   <xs:restriction base=„xs:string“>
      <xs:maxLength value=”50”/>
   </xs:restriction>
</xs:simpleType>
```

  - Extension: Attribute und Elemente werden hinzugefügt

```
<xs:element name="betrag>
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="waehrung" type="xs:string"
               use="required"/>
         </xs:extension>
       </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

  - List: Sequenzbildung aus einfachen Datentypen

```
<xs:simpleType name="Telefonliste">
   <xs:list itemType="Telefonummer"/>
</xs:simpleType>
```

  - Union: Vereinigung der Wertbereiche mehrerer einfacher Datentypen

```
<xs:simpleType name="Kontakt">
   <xs:union memberTypes="Telefonummer email"/>
<xs:simpleType>
```

  - Basistyp: anyType (zu verwenden, wenn kein anderer vordefinierter Typ passt)

- Referenzierung vordefinierter Typen
  - Referenzierung vordefinierter Typen innerhalb eines Dokumentes in Form von
    - Datentypen

```
<xsd:element name="empfaenger" type="Name"/>
```

    - Elementen

```
<xsd:element ref="absender"/>
```

- Referenzierung externer Schemata
  - include: für Schemata des selben Namensraums

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:pcTeile="http://www.example.com/pcTeile"
        targetNamespace="http://www.example.com/pcTeile">
  ...
  <include schemaLocation="http://www.example.com/schemata/ram.xsd"/>
  ...
</schema>
```

    - targetNamespace wichtig, wenn weitere NS inkludiert werden
  - import: für Schemata eines anderen Namensraums

```
<xs:schema
  xmlns:cus="http://www.example.org/customer">
  ...
<import namespace=http://www.example.org/customer
   schemaLocation="customer.xsd"/>
```

DTD

- Anwendungsbeispiel
  - XML-Dokument

```
<adresse>
   <ort>Duisburg</ort>
   <plz>47057</plz>
   <strasse>Forsthausweg</strasse>
   <nummer>2</nummer>
</adresse>
```

  - XML-DTD

```
<?xml version=„1.0“ encoding=„UTF-8“?>
<!ELEMENT adresse (ort, plz, strasse, nummer?)>
<!ELEMENT ort (#PCDATA)>
<!ELEMENT plz (#PCDATA)>
<!ELEMENT strasse (#PCDATA)>
<!ELEMENT nummer (#PCDATA)>
```

- Grundlegende Konzepte
  - Sequenz: (A, B)
    - A und B treten in dieser Reihenfolge auf
  - Alternative: (A | B)
    - entweder A oder B tritt im Dokument auf
  - Wiederholung (Kardinalität)
    - A: (1..1)
    - A?: Option (0..1)
    - A+: Iteration (1..N)
    - A*: Iteration optional (0..N)
  - Konstrukte können beliebig geschachtelt werden
- Attribute
  - Attribute werden einem Element des XML-Dokuments zugeordnet
    - XML-Dokument

```
<dozent tutorial=„BIS“ geschlecht=„m“>Jens Gulden</dozent>
```

    - XML-DTD

```
<!ELEMENT dozent (#PCDATA)>
<!ATTLIST dozent tutorial CDATA #REQUIRED geschlecht (m | f) „f”>
```

  - Elementtypen
    - #PCDATA: Parsed Character Data – Text ohne XML-Markup
    - #EMPTY: leeres Element – hat keinen Inhalt, kann aber Attribute besitzen
    - #ANY: Kombination beliebiger Inhaltstypen, verwendete Markups müssen in der DTD definiert sein
  - Attributtypen
    - CDATA: Character Data – Zeichenkette beliebigen Inhalts
    - ID: eindeutige Identifikation – Deklaration eindeutiger Werte im Dokument
    - IDREF: Deklaration von Referenzen auf ID
    - NMTOKEN: eindeutige Token
    - (Wert 1 | Wert 2 | …): Aufzählungstypen
  - Attributoptionen
    - #REQUIRED: Attribut muss angegeben wernde
    - #IMPLIED: Attribut ist optional
    - Default-Wert: Ausprägung des Attributs hat standardmäßig diesen Wert
    - #FIXED: Wert ist immer ein Default-Wert, welcher in der DTD spezifiziert sein muss
- genügt ein XML den aufgestellten Regeln des DTD, ist es gültig (valid); genügt es den allgemeinen Regeln der XML 1.0 Spezifikation, wird es wohlgeformt (well-formed) genannt
- Nachteile
  - andere Syntax als XML-Dokumente, daher müssen Werkzeuge zwei Sprachen beherrschen
  - stark eingeschränkte Auswahl an Datentypen (Zeichenketten, Identifikatoren)
  - keine Namensräume, die den Kontext für die Gültigkeit von Bezeichnern definieren
  - keine weitergehenden Kardinalitätsrestriktionen

XML Struktur Definition

- Spezifikation durch
  - XML Schema Definition Language (XSD)
  - Document Type Definition Language (DTD)
- Vorteile
  - Explizierung der Struktur
  - Validierung der Struktur
  - Vorbelegung von Attributen und Entities
- Voraussetzung für
  - strukturierte Speicherung in Datenbanken
  - Transformation in andere Formate

XML Information Set

- allgemeines Datenmodell für XML
- besteht aus sog. Information-Items, die jeweils abstrakte Repräsentationen von Teilen des XML-Dokuments darstellen
- es gibt insgesamt 15 verschieden Typen, die vier wichtigsten sind Document, Character, Element und Attribute
- weitere Datenmodelle: DOM (quasi als API für das XML Infoset) und XPath

Sprachen zur Navigation in Dokumenten/Ressourcen

- XML Path Language (XPath), XML Pointer Language (XPointer), XML Linking Language (XLink)

XPath

- Hauptaufgabe ist das Adressieren von Teilbereichen eines XML-Dokuments
- nicht alleine anwendbar, aber Grundlage für XML-Abfragesprachen (XQuery, XSLT und XPointer)
- XPath interpretiert XML-Dokumente als Bäume, Attribute sind eigene Knoten
- Grundformen
  - Pfadausdrücke
    - bestehen aus Bausteinen (Steps): album/song/title
  - logische mathematische Verknüpfungen
    - boolesche Ausdrücke: //album/song[position()=2]
    - numerische Ausdrücke: //album/song[3]/title
  - Funktionsaufrufe
    - Knoten: position() → number
    - Zeichenketten: contains(string, string) → boolean
    - numerische Werte: number([object]) → number
    - boolesche Werte: not(boolean) → boolean
- Pfadangaben
  - Beispiel:

```
<AAA>          ——————————  parent
   <BBB/>      ————————    child of AAA, preceding of CCC
   <CCC>       ————————    following of BBB
      <BBB/>
   </CCC>
</AAA>
```

  - absolute Pfadangaben:

    `/AAA`

    `/AAA/CCC/BBB` → ausführlich: `/child::AAA/child::CCC/child::BBB`

  - relative Pfadangaben:

    `//CCC/BBB`

  - weitere Beispiele:

    `/AAA/BBB[last()]`
    `//@id`
    `//BBB[@id]`
    `//BBB[not(@*)]` → alle BBB-Elemente ohne Attribute
    `//book[year>=2005]` → die kompletten Bücher als Ergebnis
    `//book[year>=2005]/year` → liefert nur die entsprechenden Jahre als Ergebnis

  - Überblick

Kürzel	Langform	Bedeutung
	child::	alle Kinder des Kontextknotens
/		Wurzelknoten
//	/descendant-or-self::node()	Nachkommen des Kontextknotens
.	self::node()	Kontextknoten
..	parent::node()	Vaterknoten des Kontextknotens
@	attribute::	Auswahl von Attributen
@*		alle Attribute des Kontextknotens
[expr]		boolescher Ausdruck zur Auswahl eines Teilpfades
[n]		das n-te Element aus der Liste von Knoten

- Navigationsachsen
  - child: die Kinder des Kontextknotens
  - descendant: alle Nachkommen (inkl. Kindeskinder etc.)
  - parent: der Elternknoten (leer wenn Wurzel)
  - ancestor: alle Vorgänger bis zur Wurzel
  - following-sibling: alle rechten Geschwisterknoten
  - preceding-sibling: alle linken Geschwisterknoten
  - following: alle nachfolgenden Knoten (ohne die eigenen Nachfolger)
  - preceding: alle vorherigen Knoten (ohne die eigenen Eltern)
- Knotentypen
  - Wurzel-, Element-, Attribut-, Text-, Namensraum- und Kommentarknoten sowie Verarbeitsungshinweise

Extensible Stylesheet Language for Transformations (XSLT)

- als W3C Standard eine Sprache zur Beschreibung von Transformationen von XML-Dokumenten in beliebige Ausgabeformate (z.B. HMTL, PDF, …)
- XSLT-Transformation selber aufrufbar durch Tools, in Eclipse z.B. durch ANT-Skript
- XPath dient zur Navigation innerhalb eines XML-Dokuments
- Baumdarstellung gemäß DOM wird (i.d.R.) von XSLT-Parsern als Basis genutzt
- Anwendungsbeispiel:

```
<xsl:stylesheet version="1.0" xmlns:xsl="…">
   <xsl:output method="html"/>
   <xsl:template match="adr:address">
   <prs:person>
      <firstname><xsl:value-of select="name/surname"/></firstname>
      <lastname><xsl:value-of select="name/familyname"/></lastname>
      <street><xsl:apply-templates select="residence/city"/></street>
   </prs:person>
   </xsl:template>
   <xsl:template match="residence/city">
   …
   </xsl:template>
</xsl:stylesheet>
```

  - mit dem Template-Element können Templates für bestimmte Elemente erstellt werden
  - value-of select extrahiert den Inhalt des aktuellen Elements
  - apply-templates transformiert das aktuelle Element mittels sämtlicher dafür anwendbarer Regeln (ohne select werden alle Elemente des aktuellen Knotens genommen)
  - weitere Elemente: for-each, if, choose, sort
  - Attribute zuweisen

```
<residence country=”{country}”>
```

- XSLT zur Typkonvertierung
  - wichtig zur Nutzung von Integrationspotentialen
  - Konvertierung verschiedener Dokumenttypen ineinander
  - möglich nur, wenn semantisch kompatible Daten
  - aber: Syntax kann konvertiert werden

XLink

- Sprache zum Platzieren von Links zu Ressourcen in XML-Dokumenten (z.B. Hyperlinks)

```
<Homepage xlink:type=„simple“ xlink:href:http://www.example.org/>
```

- Attribute
  - type: Typ des Verweises (simple, extended)
  - href: Ziel des Verweises (URI)
  - role/arcrole: Semantik des Verweises
  - title: Name des Verweises
  - actuate: wann wird der Verweis ausgeführt (onRequest, onLoad)
  - show: Verhalten bei Ausführung des Verweises (new, replace, embed)
- um auch auf Teile von Dokumenten zu verweisen, nutzt XLink XPointer

XML Pointer Language (XPointer)

- Sprache zum Verweisen auf Teildokumente
- XPointer erlaubt die Verwendung von XPath-Ausdrücken innerhalb von URI (XLink)
- im Gegensatz zu XPath kann auch auf Stellen innerhalb von Knoten verwiesen werden
- Verwendung von XPointer/XPath und XLink

  ```
  xlink:href=„students.xml#element(/1/2)
  ```

  - XLink-Verweise können durch XPointer/XPath-Ausdrücke ergänzt werden
- Verwendung von start-point und end-point (Location Types)
  - start-point(//book) zeigt auf den Anfang des ersten Knotens, end-point(//book) entsprechend auf das Ende
- XPointer Beispiele
  - Vorwahl zweiter Kunde

    ```
    //reklamationsdokument[2]/kunde/kontakt/telefonnummer/vorwahl
    ```

  - letzte Ziffern der Jahreszahlen der Rechnungsdaten

    ```
    stringe-range(//rechnungsdatum, range(endpoint(./text()), -1, 1)
    ```
- XLink und XPointer sind orthogonal zueinander und ergänzen sich gegenseitig
  - XLink definiert eine Syntax zur Formulierung von Hyperlinks in XML-Dokumenten
  - die Definition der Ziele dieser Hyperlinks geschieht durch XPointer-Ausdrücke

Parsen von Dokumenten – DOM

- Vorgehen: Dokumente werden vollständig in eine Baumstruktur umgewandelt, welche dann traversiert werden kann
- Vorteile
  - W3C-Standard; Plattformunabhängigkeit
  - komfortabler Zugriff; Lesen und Schreiben von XML-Dokumenten
- Nachteile
  - nur eine Spezifikation, keine standardisierte Implementierung
  - Dokumententransformation aufwändig
- Anwendung
  - komplexe Dokumente (von idealerweise geringem Umfang)
  - komplexe Anfragen/Transformationen
  - wiederholter Zugriff auf Dokument

Parsen von Dokumenten – SAX

- Programmierschnittstelle, die ein Parsen von XML-Dokumenten erlaubt
- dabei werden vom Parser Ereignisse (start document, start element, end element etc.) generiert
- es wird üblicherweise kein (kompletter Baum) im Speicher aufgebaut, stattdessen durchläuft der Parser die XML-Datei und meldet, welchen Daten er dabei begegnet

- aktive API: API ruft Funktionen des Programms auf („Callback“)
- Vorteile
  - prinzipiell weniger ressourcenintensiv als DOM
  - Implementierung für Java und andere Sprachen verfügbar
- Nachteile
  - kein Standard
  - zustandslos, d.h. Parsen muss oftmals von vorne beginnen
  - XML-Dokumente können nicht manipuliert werden
- Anwendung
  - einfach und gleichartig strukturierte Dokumente (Sequenzen)
  - sehr große Dokumente
  - einmalige Zugriffe auf Dokumente (Stream)

XML – Entwurfsziele

- XML soll soll ohne Umwege über das Internet nutzbar sein
- XML soll eine Vielzahl an Applikationen unterstützen
- XML soll kompatibel mit SGML sein
- Programme sollen XML einfach verarbeiten können
- auf optionale Funktionen soll möglichst komplett verzichtet werden
- Dokumente sollen für Menschen lesbar und klar verständlich sein
- das Design von XML sollte formal und knapp sein
- XML-Dokumente sollen leicht zu erstellen sein

XML – Bewertung

- Vorteile
  - Plattform- und Layoutunabhängigkeit
  - einfache Erweiterbarkeit der Dokumente
  - Lesbarkeit
  - große Anzahl an (freien) Werkzeugen: Editor, Parser etc.
- Nachteile
  - XML-Dokumente können sehr groß werden
  - keine integrierten Sicherheitskonzepte (bisher)
  - Umwandlung existierender Dokumente in XML sehr aufwändig
  - oft unklar, ob Informationen als Element oder Attribut dargestellt werden sollen

XML-Standards

Standard	Funktion
XML	Metasprache
XSD/DTD	Strukturierung von Dokumenten
XSL(T)	Präsentation/Transformation
XPath/XPointer	Adressierung von Teildokumenten
XLink	Definition von Hyperlinks
DOM/SAX	API zum Parsen von Dokumenten
XQuery	Anfragesprache

XML-Datenbanken

XML-Datenbanken

- dienen der Verwaltung persistenter XML-Dokumente
- Schemadefinitionen mittels XML-DTD oder XML Schema
- zunehmende Bedeutung durch wachsende Verbreitung von XML-Dokumenten
- unterschiedliche Speicherungstechniken
  - Speicherung als Ganzes
  - Speicherung der Dokumentenstruktur
  - Abbilden auf Datenbankstruktur

Ansätze zur Verwaltung von XML-Dokumenten in DB

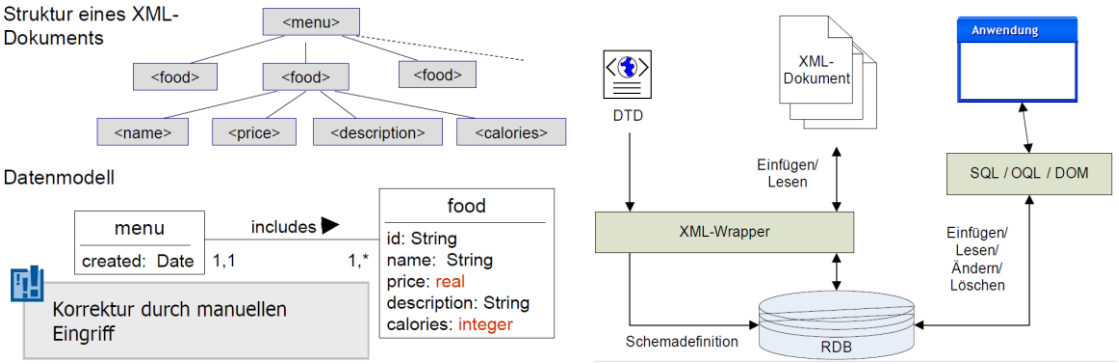
- spezieller Typ für die Speicherung kompletter XML-Dokumente in RDBMS
  - Datentyp „XML“ in SQL (standardisiert in SQL2003)
  - beim Einfügen neuer Instanzen kann Wohlgeformtheit durch RDBMS geprüft werden
  - z.B. Unterstützung der Recherche durch Volltextsuche in XML-Dokument
  - einige Produkte bieten dedizierte Methoden zum Zugriff auf XML-Inhalt (z.B. integrierter XSLT-Prozessor)
  - Beurteilung
    - relativ leicht als Erweiterung von RDBMS zu implementieren
    - Wiederverwendung existierender Verfahren zur Volltextsuche
    - nicht immer Überprüfung auf Konformität (bzgl. DTD oder XML Schema)
    - Redundanzen sind konzeptionell inhärent
- Generierung von XML-Dokumenten aus RDBMS
  - XML wird nicht direkt in DBMS persistent gemacht (also keine XML-DB)
  - es existiert eine umkehrbare Abbildung von Daten im RDBMS auf XML-Elemente
  - XML-Dokumente werden bei Bedarf aus den Inhalten eines RDBMS erzeugt
  - Beurteilung
    - gute Voraussetzung für Verwaltung von Daten (relativ hohe Integrität, differenzierte Suche möglich)
    - existierende DBMS müssen nicht modifiziert werden
    - Nachteil: Abbildungsaufwand u.U. erheblich
    - Erzeugung von XML-Dokumenten ggf. mit Verlust von Semantik verbunden
- Erweiterungen von RDBMBS („XML-fähige DBMS“)
  - i.d.R. relationale Datenbanken mit ergänzenden Werkzeugen
  - Alternative 1: Speichern der Graphstruktur von Dokumenten
    - Speichern der Graphstruktur von Dokumenten
    - Abbildung des XML-Metamodells auf relationales Schema

DocID	Elementname	ID	Vorgänger	Kind-Nr	Wert
M001	menu	01		1	
M001	food	02	01	1	
M001	name	03	02	1	Belgian Waffles
M001	Price	04	02	2	$5.95

    - Speichern jedweder XML-Dokumente unabhängig von ggfs. vorhandener Schemabeschreibung
    - Verwenden von Relationen zur Speicherung von Elementen und Attributen

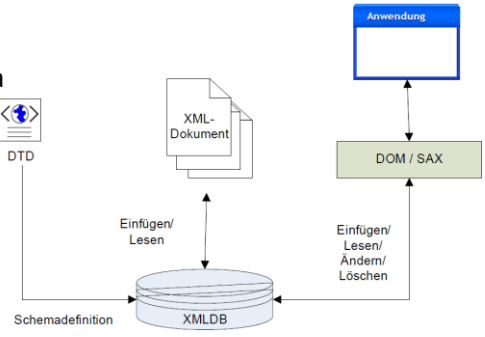
- Bewertung
  - generischer Ansatz
    - keine Modifikation existierender (R)DBMS
    - Transformation kann transparent durch eine Zugriffsschicht („Wrapper“) durchgeführt werden
    - Schemabeschreibung nicht erfoderlich
  - aber:
    - XML-Dokumente werden unabhängig vom Typ in gleicher Struktur abgelegt
    - deswegen: keine (inhärente) Gewährleistung der Dateintegrität
    - wenig intuitive Strukturierung der Daten
    - aufwändige Rekonstruktion der Dokumente

- Alternative 2: Transformation XML ↔ DB-Schema



  - Bewertung
    - keine Modifikation existierender (R)DBMS
    - Transformation kann durch einen Wrapper durchgeführt werden (schließt Validierung des Dokuments ein, Transformationsvorschrift erfodert i.d.R. manuelle Festlegungen)
    - referentielle Integrität wird durch RDBMS gesichert
    - Abfragen: SQL, OQL – ggfs. auch XPath, XQuery
    - Ändern: SQL, OQL – ggs. auch DOM, SAQ, XUpdate
    - nichtsdestotrotz: Bruch zwischen Paradigmen (Performancenachteile, u.U. ungünstige Speicherökonomie)

- „native“ XML-DB
  - von Grund auf für XML entworfen
  - Schemadefinitionen mittels XML-DTD oder XML Schema
  - Datenmanagement: Rechte, Trigger, Indizes
  - internes Speicherformat sollte logischem Modell folgen
    - Elemente, Attribute, PCData
    - Dokumentenreihenfolge
    - bspw. DOM → Tabellen „Elemente”, „Attribtue”, „Text”



  - Datenzugriff erfolgt ausschließlich über XML-Technologien
    - lesender Zugriff über XPath und XQuery
    - schreibender Zugriff über Schnittstellen zur Programmiersprache (DOM) oder XUpdate
  - Bewertung
    - erfüllt spezielle Anforderungen an XML-DB
    - Zugriff auf Daten ausschließlich über XML-Technologien

- hohe Abfrageleistung
- nichtsdestotroz: Nischendasein nativer XML-DBMS

Konzepte im Vergleich

XML-DB	Zentr. Konzept	Minimum	Zugriff (lesend)	Zugriff (schreibend)	Ref. Integrität
XML-Typ in SQL	Relation	SQL-2003	SQL	SQL	--
XML-fähig (Schema)	Relation, Objekt	SQL-1999 (Wrapper)	SQL, OQL (XML-basiert)	Methoden (Progr.-Sprachen)	(O)RDBMS
XML-fähig (Graph)	Relation	SQL-92 (Wrapper)	SQL	SQL	--
XML-basiert	XML-Dokument	XML-DBMS	XQuery, XPath, proprietär	DOM, SAX, proprietär	XML-DBMS

Data Warehouse

Data Warehouse

- der Umfang eines DWH wird in Literatur und Praxis nicht eindeutig abgegrenzt
- wesentliche Merkmale
  - Unterstützung der Integration von Daten, die in heterogenen Quellen vorliegen (vertikale Integration!)
  - Auswertung bzw. Aufbereitung komplexer Datenmengen zur wirksamen Entscheidungsunterstützung → OLAP
  - entsteht durch einen (permanenten) Prozess der Datenextraktion und -aufbereitung
  - i.d.R. nur lesender Zugriff durch den Benutzer

OLTP vs. OLAP

Charakterstik	OLTP	OLAP
Benutzertyp	z.B. Sachbearbeiter (operativ)	z.B. Management
Benutzerzahl	eher viele	tendenziell wenige
Antwortzeit	Sekunden	Sekunden bis Minuten, sogar Stunden bei kompl. Anfragen
Anwendung	operatives Geschäft	Entscheidungsunterstützung
Anfragetyp	Tabellen-orientiert, vordefiniert	multidimensional, aggregiert, ad-hoc
Zugriffstyp	kurze Lese-/Schreibtransaktionen	lange Leseoperationen
Dateneigenschaften	zeitaktuell, tendenziell hohe Änderungsrate	konsolidiert, historisiert, integriert

DWH Auswertungsoperationen

- Slice: Reduktion des Gesamtwürfels durch Einschränkung der Dimension (Ausschnittsbildung)
- Dice: Reduktion des Gesamtwürfels auf eine Dimension
- Roll-up: Aggregation (Summierung) der Daten über eine weitere Dimension
- Drill-down: Disaggregation aggregierter Daten zur Unterstützung einer differenzierten Betrachtung (Ursachenanalyse)
- Drill Thru: Wechsel der Datenquelle

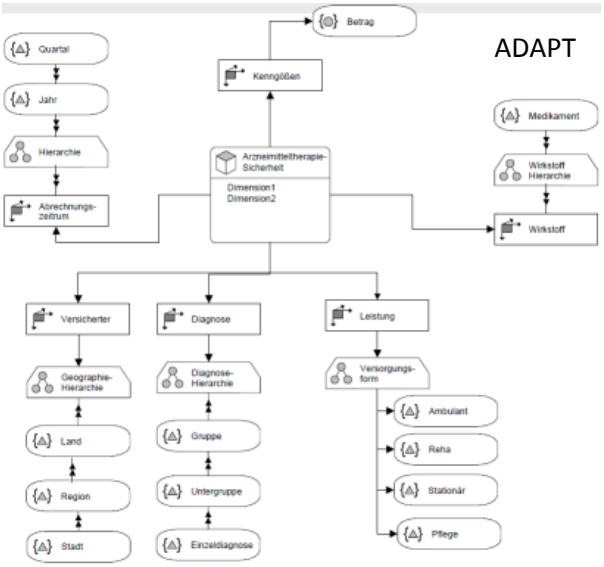
Exemplarisches Vorgehensmodell zur Erstellung eines DWH

1. Identifikation der Quelldaten
   - Beispiele für Kriterien bei der Auswahl von Quelldaten
     - Zweck des DWH
     - Qualität der Daten
     - Verfügbarkeit der Daten (rechtlich, organisatorisch, technisch etc.)
     - Preis für den Erwerb der Daten
   - Monitoring wichtig für die Identifikation von neuen und veränderten Daten in ausgewählten Quellen
     - zeitstempel- oder protokollbasierte Entdeckung, Dateivergleich
2. Extraktion in Basistabellen
   - Quelldaten werden in Basistabellen extrahiert, um dort transformiert und später in die konsolidierte DB geladen zu werden
   - Basistabellen häufig nur temporär als Arbeitsbereich für die Datenbeschaffung und Datenaufbereitung
   - wichtig: Zeitpunkt der Extraktion
     - periodisch, anfrage- oder ereignisgesteuert
3. Reinigung der Daten
   - Erkennen und Bereinigen von Datenkonflikten und -inkonsistenzen
     - Fehler im Datenmaterial, die im OLTP-Betrieb mitunter nicht zum Tragen kommen
     - Datenkonflikte zwischen Quellsystemen, die redundante Informationen enthalten
   - Erkennen von Anomalien, fehlerhaften Daten und Ausreißern
   - semantische Probleme (Benennung, Maßeinheit, Format etc.)
   - Beispiele für Datenbereinigungsmaßnahmen
     - Anpassung von Datentypen, Konvertierung von Kodierungen, Vereinheitlichung von Datumsangaben, Umrechnung von Maßeinheiten bzw. Skalierung
     - Surrogate (Zuordnungstabellen)

Quelle	Relation	Attribut	lok. Schlüssel	glob. Surrogat
System1	kunde	kunden_nr	12345	66
System1	kunde	kunden_nr	44444	67
System2	customer	customer_id	A134	67

     - ggfs. auch Aggregierung (nicht auf bestimmte Analyse gerichtet) und Berechnung von abgeleiteten Attributen
4. Konsolidierung der Daten
   - Konsolidierte DB durch Integration der Quelldaten gekennzeichnet
   - weitgehend unabhängig von spezifischen Analyseanfragen
   - Problem beim Laden: Zugriff auf große Datenmengen in einem z.T. nur kurzen Zeitfenster
   - operative Systeme und die konsolidierte DB sollten daher während des Ladevorgangs aus Konsistenz- und Perfomance-Gründen offline sein
   - konsolidierte DB sollte gegen Veränderungen in den Analyseanforderungen möglichst resistent sein
5. Erstellung dispositiver Datenbank
   - dispositive DB wird durch konsolidierte DB versorgt
   - dispositive DB orientiert sich an den Analysebedürfnissen der Anwender
   - Daten müssen eine geeignete Granularität für die späteren Analysen besitzen

- Erfassung der Analysebedürfnisse auch mit konzeptuellen Datenmodellen (semiformal)
  - optimiert für Analyseszenarien
  - Voranalyse/Vorverdichtung von Daten, bspw. Vorberechnung von Summen
- Vorgehen
  1) Anforderungsanalyse: Fokus auf Geschäftsprozesse
  2) konzeptionelles Schema, Ansätze: mE/R, mUML, ADAPT etc.



  3) Verwaltung der Daten
     - ROLAP: Star/Snowflake-Schema als relationale (logische) Schemata
     - MOLAP: multi-dimensional memory organisation, proprietäre Implementierung

Multidimensional Expressions (MDX)

- Entwurf von Datenwürfeln
- Abfragen von Daten einer multidimensionalen DB
- Formatierung von Abfrageergebnissen
- Definition abgeleiteter Elemente (Summen etc.)
- weitere Vertreter multidimensionaler Anfragesprachen
  - Multidimensional SQL (SQLM), Multidimensional Query Language (MDSQL), Red Brick Intelligent SQL (RISQL)
- Beispiel

```
SELECT {[Dimension].[Element]} ON COLUMNS,
{[Dimension].[Element]} ON ROWS
FROM [cube] WHERE [Dimension].[Element]
```

Diskussion aktueller Trends

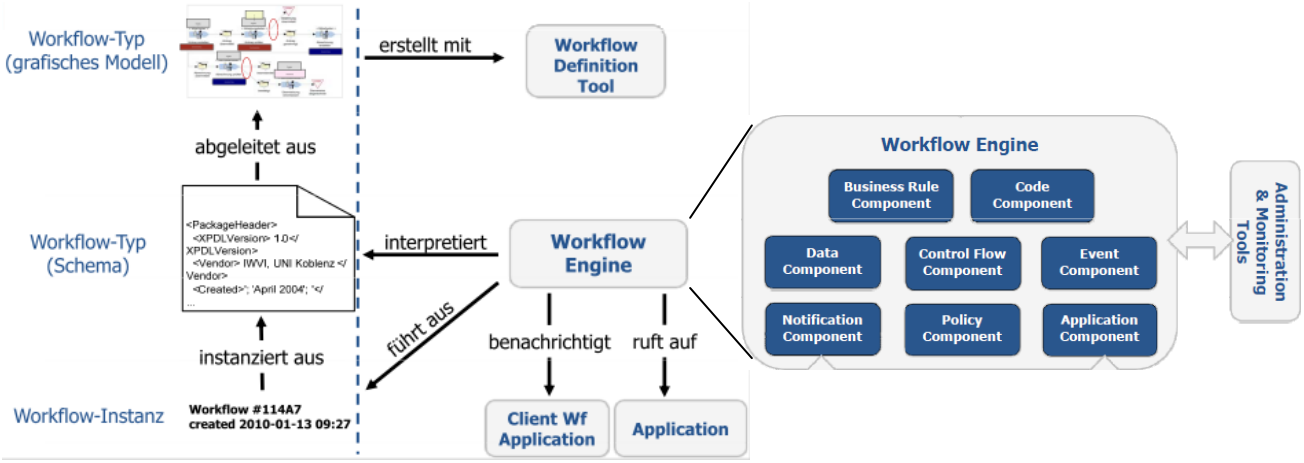
- Process Mining und Process Warehouse
- (Neal) Real-time / Right-time Data Warehouse System
- Markt-Konsolidierung (SAP + Business Objects, IBM + Cognos, Oracle + Hyperion)
- Open Source Lösungen (z.B. Pentaho BI Suite)

Workflow Management

Charakterisierung grundlegender Begriffe

- Geschäftsprozess
  - wiederkehrende Abfolge von Aktivitäten
  - verwendet knappe Ressourcen
  - steht in einem direkten Zusammenhang mit der marktgerichteten Leistungserstellung eines Unternehmens
- Workflow
  - Abstraktion eines Geschäftsprozesses
  - ist gerichtet auf den Fluss digitaler Dokumente bzw. Objekte
  - ergänzt Geschäftsprozesse z.B. um Dokumente, Anwendungen und Entscheidungsregeln
- WFMS
  - System zur Beschreibung, Ausführung und Kontrolle eines Workflows
  - verwendet vorhandene Software und ggfs. eigene Anwendungsteile
  - Ziele: Steuerung, Integrität und Transparenz von Workflows

Funktionsweise und Architektur von WFMS



Vorgehensmodell im Workflow Management

1. Modellierung der Geschäftsprozesse
   - häufig motiviert aus nicht-technischen Gründen, z.B. Dokumentation
   - unterstützt betriebswirtschaftliche Analysen, z.B. Reorganisation, Organisationsbrüche, Redundanzen
   - Abstraktionsniveau tendenziell hoch, selten einzelne Tätigkeiten („Tasks“)
   - zur Ausführung notwendige Details selten vollständig spezifiziert
2. Verfeinerung zum Workflow
   - Anreicherung der Geschäftsprozessmodelle um diejenigen Details, die zur Interpretation und Ausführung der Prozesse durch Workflow Engines notwendig sind, u.a.
     - Anwendungen, Dokumente (Variablen), Entscheidungsregeln, auszuführende Skripte, Berechtigungen, relevante Ereignisse
   - erweiterte Konzepte der BPMN: Manual/User/Service Task; Intermediate Events (interrupting, non interrupting): Message, Timer, Error

3. Ableitung des Workflow-Schemas
   - Workflow-Schemas umfassen alle Informationen, die für die Ausführung eines Workflows erforderlich sind, d.h. Ausführungsregeln (Prozessfluss), Ereignisse, Daten, Applikationen und Rollen
   - XML-basierte Workflow Schemas: WfMC XPDL, OASIS WS-BPEL, JBoss jPDL
   - Verheißungen der Standardisierung: Investitionsschutz, Wiederverwendbarkeit, Interoperabilität
4. Instanziierung des Schemas
   - Deployment des Workflow-Schemas auf einem Workflow-Server (Workflow-Engine)
   - Anpassung: ggf. weitere Details notwendig, bspw. matching von Usern auf Rollen oder ToolAgents auf Applikationen
   - Instanziierung des Workflow-Schemas
5. Kontrolle & Anpassung des Workflow (Management Systems)
   - Auswertung der Instanzdaten (Workflows) im WfMS
   - zugleich: Auswertung nicht-digitalisierter Informationen (z.B. Ausnahmen, Abweichungen)
   - Überarbeitung des Workflowtyps durch Anpassung der Geschäftsprozessmodelle (beginnend bei Schritt 1) oder auch der Workflowmodelle (beginnend bei Schritt 2)

BPMN

- von der OMG getriebene Modellierungsnotation für Prozesse
- soll „Standard“ für Prozess-Modellierungssprachen werden/sein
- angelehnt an bekannte Modellierungssprachen (z.B. EPK)
- Notationselemente
  - Basiselemente: Activity, Event (Start, Ende), Flow, Gateway
  - weitere Elemente: Sub-Process, Gateway Exclusive/Parallel/Inclusive
  - Pool: ein Prozess (Abteilung, Unternehmen)
  - Lane: ein Verantwortlicher für einzelne Prozessschritte
- Ziele und Verheißungen
  - soll intuitiv anschaulich und verständlich sein
  - soll Brücke zwischen technischen und betriebswirtschaftlich orientierten Akteuren schließen (auf Ebene der Kommunikation sowie auf Ebene der Formate)
  - soll inter-organisationalen Austausch von Modellen ermöglichen (BPMN als Standard für Prozessmodellierung)
- Beurteilung
  - große Beliebtheit im Praxis, vermutlich wegen (vordergründiger) Einfachheit und Nähe zu anderen Modellierungssprachen (z.B. UML Aktivitätsdiagramme)
  - besitzt Konzepte für verschiedene Anspruchsgruppen auf betriebswirtschaftlicher wie technischer Ebene
  - bietet zudem Unternehmen viele Freiheitsgrade zur individuellen Anpassung und/oder Interpretation (Notation, keine Sprache!)
  - großer Markt an entsprechender Software
  - zur vollständigen Modellierung ausführbarer Workflows noch nicht geeignet (in Version 2 deutliche Besserung)
  - aufgrund fehlender Konzepte weiterhin Bruch zur Geschäftsprozessmodellierung (Unternehmensmodellierung) und dortige Anwendungsgebiete

- unklare Abgrenzung zu anderen Modellierungssprachen, insbesondere „innerhalb desselben Hauses“ (OMG, UML-Aktivitätsdiagramme)
- fehlende „formale“ Spezifikation (z.B. durch Metamodell) und kaum definierte Semantik
- Folge: kaum einheitliche Nutzung im Detail (Standard?), Modellaustausch eher unwahrscheinlich
- insgesamt: wenig Innovation im Vergleich zu anderen Ansätzen

Schlussbetrachtung Workflow Management Systeme

- versprechen hohe Unterstützung von Abläufen in Unternehmen
- vielfältiges Angebot verfügbar an
  - methodischer Unterstützung (Modellierung, Standards)
  - sowie korrespondierender Software
- jedoch: keine echten Standards oder Referenzarchitekturen
  - hohe Varianz an Ausprägungen (Modellierungssprachen, Schemata, unterstützter Komponenten)
  - somit: Austauschmöglichkeit zwischen Werkzeugen gering