Kapitel 11
Implementierung und Algorithmen

- XML als Datenstruktur:
  - *abstrakter Datentyp* mit API: DOM
  - (zumeist Hauptspeicher-) Implementierungen
  - Low-Level API mit variablenbasierten Einzelzugriffen
- Datenbanken?
  - High-Level API: XPath, XQuery
  - Abbildung auf relationales Modell (Oracle, IBM DB2)
  - "Native" Speicherung: Software AG-Tamino
  - Datenbankfunktionalität: Mehrbenutzer, Transaktionen, Recovery
- Stream Processing:
  - XML-Datenübertragung als Sequenz von Ereignissen
  - SAX (Simple Application Interface for XML)

11.1 DOM

- DOM (Document Object Model) definiert eine plattform- und sprachunabhängige *Schnittstelle* bzw. *abstrakten Datentyp* um XML-Daten zu erstellen, verarbeiten und manipulieren.
- sprachliche Basis der DOM-Spezifikation: OMG-IDL
- Implementierungen u.a. in Java und C++; in der Regel hauptspeicherbasiert
- DOM ist *kein* Datenmodell und *keine* Programmiersprache!
- DOM *implementiert* ein *logisches* Datenmodell (siehe "XML als Datenmodell"), in dem XML-Daten als Bäume repräsentiert werden:
  - Document: repräsentiert komplettes Dokument,
    - Anfrage-Methoden, z.B. NodeList getElementsByTagName(string)
  - allg. Klasse Node: getNodeType(), getChildren(), getFirstChild(), getNextSibling(), getParentNode(), ...
  - Klasse Element: getTagName(), getAttributes(), ...
  - Klasse Attribut: getName(), getValue(), ...
  - entsprechende Methoden, um Knoten zu erzeugen und zu verändern.

DOM: BESTANDSAUFNAHME

- Hauptspeicherbasiert: nur für kleinere Anwendungsprogramme (die meisten "Lightweight"-Tools (siehe ap34) basieren intern auf DOM)
- Schritt-für-Schritt-Zugriffe auf die Daten: basierend auf Variablenzuweisungen in der umgebenden imperativen/objektorientierten Programmiersprache und Iteratoren (vgl. Vorgehensweise beim Netzwerkmodell)
- kein deklarativer Zugriff
- darauf aufbauend kann man XPath und XQuery implementieren (vgl. Apache Xerces (XML/DOM)/Xalan (XSLT)/Xindice (DB)
- oft ineffizient (keine Indexe, Anfrageoptimierung)

⇒ Datenbanken müssen anders vorgehen: Welches *physische Datenmodell* ist am geeignetsten für XML?

11.2 XML-Datenspeicherung

11.2.1 "Native"

- XML-Dokumente als ganzes gespeichert
- Datenmodell kennt die Begriffe "Baum", "Element", "Attribut", "Dokumentordnung"
- Ansätze mit textbasierter Speicherung (CLOBs, Files) und Text-Indexen (Volltextindexe, statistische Verfahren, ...)
  Anwendung und Weiterentwicklung klassischer Algorithmen
- Ansätze basierend auf internen Objektmodellen (oft auf bestehenden objektorientierten DB-Systemen mit Anwendung von Konzepten aus hierarchischen und Netzwerk-DBs)
  - Speicherung basierend auf der Graphenstruktur der XML-Instanzen (kann man im Prinzip auch in geeigneten Tabellen speichern ...)
  - Navigation
- Produkte: Tamino (Software AG), Excelon

11.2.2 Abbildung XML → Relationales Modell

Zwei grundlegende Umsetzungsalternativen:

- Eine oder mehrere "massgeschneiderte" Tabellen für jeden Elementtyp (→ entsprechend relationaler Normalisierungstheorie)
  - viele Nullwerte
  - effizienter Zugriff auf zusammengehörende Daten
- eine große Tabelle: (Element-id, Name der Eigenschaft, Wert/id des Ergebnisses)
  - keine Nullwerte
  - trotzdem hoher Speicheraufwand (Namen, die nach (1) nur einmal als Attributnamen gespeichert sind, sind jetzt bei jedem Auftreten abgelegt)
  - zusammengehörende Daten auf mehrere Tupel verstreut

⇒ in jedem Fall kann die Theorie und Effizienz relationaler Datenbanken voll genutzt werden.

- Literatur: Klettke/Meyer "XML & Datenbanken" (dpunkt-Verlag)
  Chaudhri/Rashid/Zicari: XML Data Management

11.3 SAX

SAX (The Simple API for XML) ist ein anderes, *event-basiertes Interface/Modell*:

- stellt XML-Dokument als Folge von Ereignissen dar (Tiefensuche), z.B.
  - startDocument(), endDocument()
  - startElement(Name, AttributeList) – Attribute nicht aufgelöst
  - endElement(Name)
  - characters(string,int,int)
- Wenn man SAX-basiert programmiert, schreibt man sich einen Event-Handler, der auf die gemeldeten Ereignisse reagiert und etwas tut:
  - z.B. DOM-Baumstruktur erstellen
  - linear etwas suchen
    z.B. XPath-Anfragen wie //country[@car_code='D']/population können sehr (zeit- und speicher-)effizient beantwortet werden
  - geht nur für Anfragen, die nur vorwärts suchen
  - welche XPath-Ausdrücke sind äquivalent (und mit welchem Aufwand) in vorwärts-Ausdrücke umwandelbar? [Bry et al, XPath: Looking forward; 2002; LMU München]

11.4 Vergleich

- DOM's Baum-Struktur stellt hohe Anforderungen an Hauptspeicher
  (wenn sie explizit gespeichert ist – man kann natürlich auch ein DOM-Interface auf einer effizienten Datenbankspeicherung implementieren)
- für Parsen/Lesen genügt SAX
- für komplexere Anfragen ist SAX ungeeignet
- am effizientesten für große Mengen sind spezielle Datenbankansätze (wiederum unterschiedlich für dokumentorientiertes XML und datenbankorientiertes XML).

Kapitel 12
Sonstiges

In der Vorlesung nicht behandelt wurden u.a.:

- XML: Namensräume
- XML: Processing Instructions
- Theorie: XML Data Model, XPath/XQuery Formal Semantics
  (werden derzeit mit XML Schema auf eine Linie gebracht)
- der SQLX-Standard: generische Abbildung zwischen XML/SQL, womit man an dieselben Daten Anfragen in SQL und XPath/XQuery stellen kann.

Anwendungsbereiche (u.a.)

- Web Services
- Multimedia-Anwendungen mit XML
- Semantic Web

NACHWORT

Alles was es für relationale Datenbanken gab, gibt es auch hier wieder:

- physikalische Speicherung
- Optimierung, Auswertungsalgorithmen
- Mehrbenutzerbetrieb, Transaktionen (ACID), Sicherheit, Zugriffsrechte
- ECA-Regeln, Trigger

Die Algorithmen und theoretischen Grundlagen sind im großen und ganzen dieselben.

Was sollte die Vorlesung vermitteln?

- Wissen für den praktischen Umgang mit XML
- XML ist mehr als nur spitze Klammern
- Die XML-Welt bietet Beispiele für viele grundlegende Konzepte der Informatik und ihr Zusammenspiel
- Illustration, wie sich Konzepte in der Informatik entwickeln.