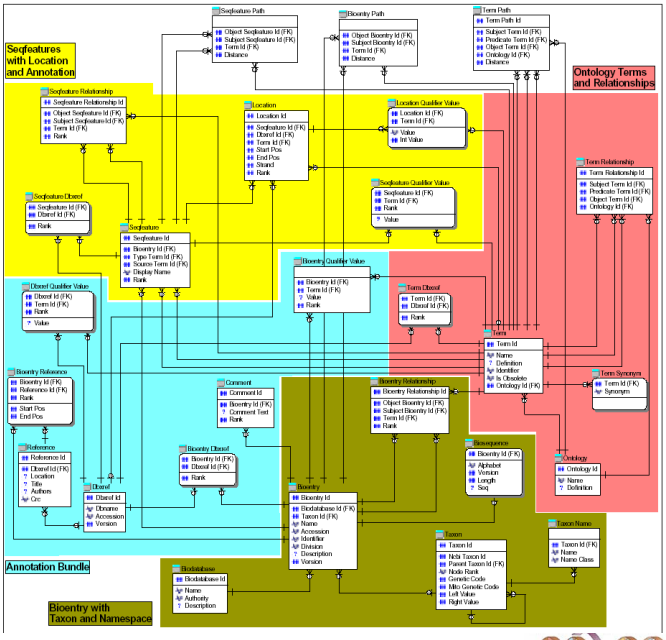
Kapitel 7 (Forts.): Deklarative Anfragesprachen in Bio-Datenbanken

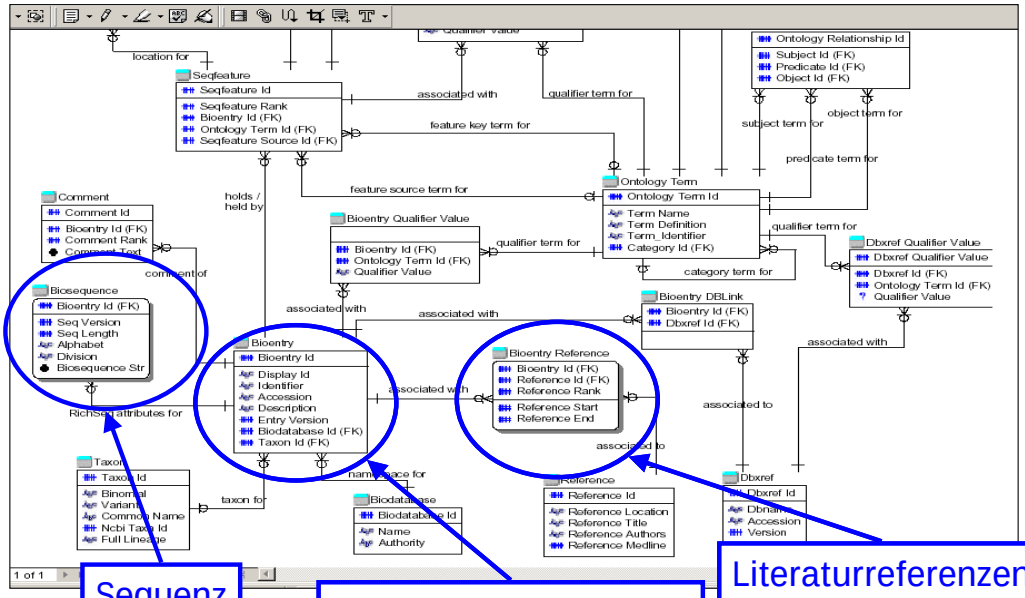
- Unterstützen (semi-)strukturierte Anfragen (i. allg. via Mengenoperationen)
- Vertreter
  - SQL92 (relational), SQL:2003 (objekt-relational)
  - OQL (objektorientiert)
  - XPath / XQuery (XML-basiert)
- Verwendbarkeit für Bio-Datenbanken?
  - Bio-Daten: Oft nur semi-strukturiert
  - Ggf. hoher Multimedia-Anteil: Text, Images (z.B. bei Genexpression), Verlaufskurven (z.B. bei medizinischen Daten)
- Kaum Verwendung als "öffentliches" Bio-Interface, da zu komplex
- Bio-Spezialsprachen: BioSQL, PQL, ...

Beispiel: BioSQL

- Auf Basis von SQL92
- Keine SQL-Spracherweiterung, sondern spezifische DDL-Ausprägung
- OpenSource Projekt (http://cvs.bioperl.org/)
- Relationale Schemata (Oracle, PostGres, mySQL)
- EMBL & SwissProt
  - Ca. 20 SQL-Tabellen

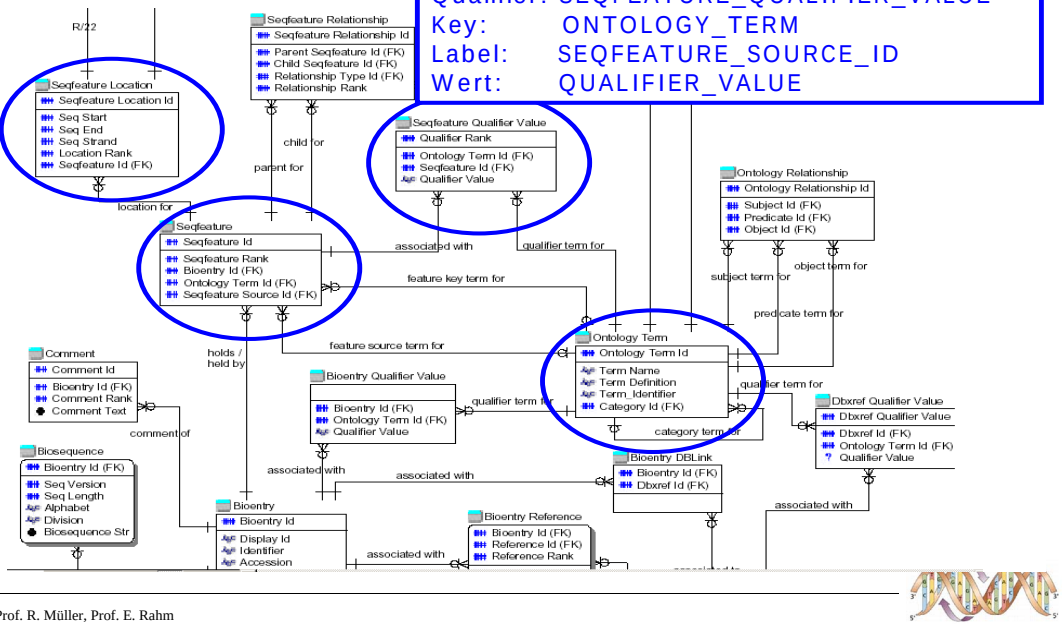


BioSQL Core



BioSQL Feature

Feature von BIOENTRY_ID
Location: SEQFEATURE_LOCATION
Qualifier: SEQFEATURE_QUALIFIER_VALUE
Key: ONTOLOGY_TERM
Label: SEQFEATURE_SOURCE_ID
Wert: QUALIFIER_VALUE



BioSQL und BioPython

- BioSQL liefert API für mehrere Programmiersprachen
  Python, Perl, Ruby, Java, ...
- Zugriff auf BioSQL DB aus Python

```
>>> from BioSQL import BioSeqDatabase
>>> server = BioSeqDatabase.open_database(driver = "MySQLdb", user = "chapmanb",
...             passwd = "biopython", host = "localhost", db = "bioseqdb")
>>> db = server.new_database("cold")
>>> from Bio import GenBank
>>> parser = GenBank.FeatureParser()
>>> iterator = GenBank.Iterator(open("cor6_6.gb"), parser)
>>> db.load(iterator)
6
>>> record = db.lookup(accession = "X62281")
>>> record.id
'X62281'
>>> record.name
'ATKIN2'
>>> record.description
'A.thaliana kin2 gene.'
```

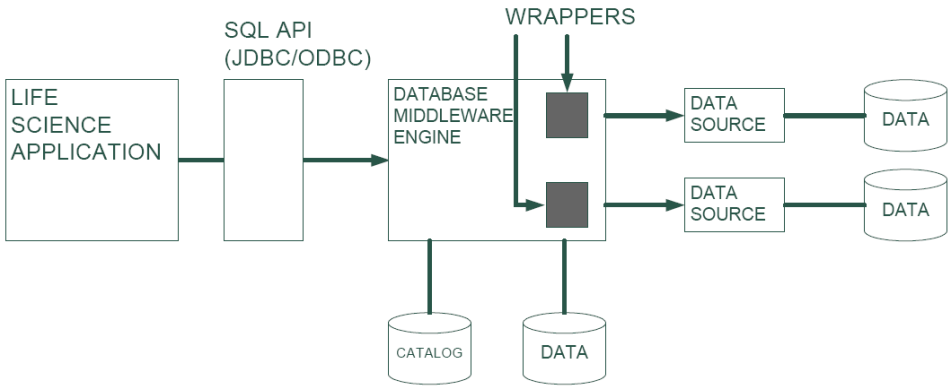
DiscoveryLink

[L.M. Haas, P.M. Schwarz, P. Kodali, E. Kotlar, J. E. Rice undW. C. Swope. DiscoveryLink: A system for integrated access to life sciences data sources. IBM Systems Journal, 40(2):489–511, 2001.]

- kommerzielles System von IBM
- vollkommene Transparenz für Anwender
- Anwender denkt: Ich arbeite mit einer Datenquelle!
- Vorgänger war GARLIC-Projekt:
  Integration von heterogenen Quellen mit Spezifikation eines integrierten Schemas und unter Nutzung von Wrappern
- Ankopplung von Analysewerkzeugen ist primäres Ziel von Discovery Link
- FDBMS-Ansatz

DiscoveryLink

Architektur:



DiscoveryLink

- Wrapper sind in C++ implementiert und werden dynamisch geladen

```
CREATE WRAPPER ChemWrapper LIBRARY
    'libchemdb.a'
```

- Definition eines Servers auf den mittels Wrapper zugegriffen wird

```
CREATE SERVER Chem-HTS WRAPPER ChemWrapper
        OPTIONS(NODE 'hts1.bigpharma.com',
                PORT '2003', VERSION '3.2b')
```

- Schemadefinition eines Wrappers

Protein_Sequence Wrapper Schema

```
CREATE NICKNAME PROTEINS {
    PROTEIN_ID VARCHAR(30)  NOT NULL,
    NAME VARCHAR(60),
    FAMILY VARCHAR(256),
    DISEASES VARCHAR(256)
} SERVER PROTEINDB
```

Assay Wrapper Schema

```
CREATE NICKNAME ASSAYS {
    COMPOUND_ID VARCHAR(10)  NOT NULL,
    SCREEN_NAME VARCHAR(30)  NOT NULL,
    IC50 DECIMAL(12,10)
} SERVER ORACLE12
```

Molecule Wrapper Schema

```
CREATE NICKNAME COMPOUNDS {
    COMPOUND_ID VARCHAR(10)  NOT NULL,
    STRUCTURE LONG VARCHAR,
    MOL_WT DECIMAL(10,6),
    LOGP DECIMAL(10,2)
} SERVER CHEM-HTS
CREATE FUNCTION MAPPING FOR
    SIMILARITY(LONG VARCHAR, LONG VARCHAR)
    RETURNS FLOAT
    SERVER CHEM-HTS
```

DiscoveryLink

- Anfragen werden mittels SQL formuliert

```
SELECT a.compound_id, a.IC50, p.name,
   c.structure
FROM Assays a, Proteins p, Compounds c
WHERE a.screen_name = p.protein_id
   AND a.compound_id = c.compound_id
   AND p.family LIKE '%serotonin%'
   AND a.IC50 < 1E-8
```

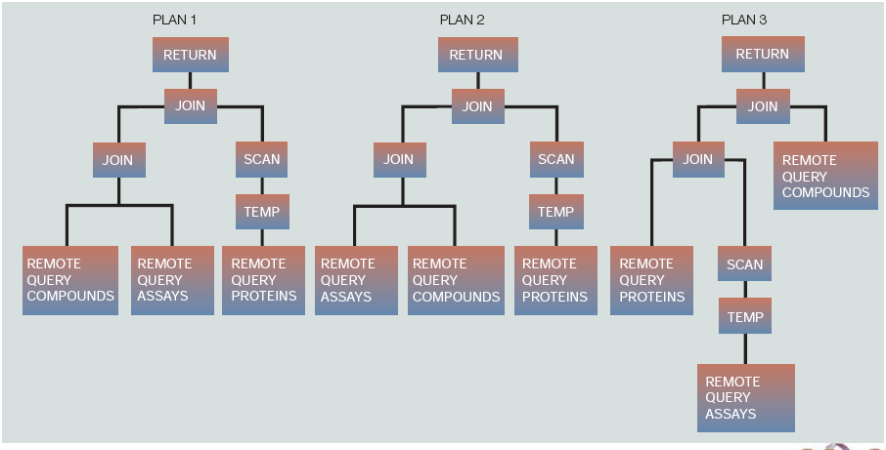
DiscoveryLink

- Anfragen werden optimiert
- Erste Phase: Zugriff auf einzelne Tabellen optimieren
- Predikate an Datenquellen delegieren

RemoteQuery	RemoteQuery	RemoteQuery
Tables: Assays	Tables: Compounds	Tables: Proteins
Columns: compound_id IC50 screen_name	Columns: compound_id structure	Columns: protein_id name
Predicates: IC50 < 1E-8	Predicates: (None)	Predicates: family LIKE '%serotonin%'

DiscoveryLink

- Zweite Phase: Joins zwischen Tabellen optimieren (links -> rechts)



DiscoveryLink

- Komplexere Anfragen

```
SELECT a.compound_id., a.IC50, p.name,
       c.mol_wt, c.logP, c.structure,
       similarity(c.structure,
                  :KETANSERIN_MOL) AS rank
FROM Assays a, Proteins p, Compounds c
WHERE a.screen_name = p.protein_id
   AND a.compound_id = c.compound_id
   AND p.family LIKE '%serotonin%'
   AND a.IC50 < 1E-8
   AND c.mol_wt BETWEEN 375 AND 425
   AND c.logP BETWEEN 4 AND 5
ORDER BY rank
```

DiscoveryLink

- FDBMS -> integriertes Schema
- Zugriff auf Originaldaten durch Views
- kommerzielles Produkt
  Kostet sehr viel Geld!!! ☹
- Plattformunabhängigkeit
- Internetfähigkeit
- lesendes SQL
- JDBC-, ODBC-Treiber
- verschiedenste Ausgabeformate
- keine Einbindung eigener Datenquellen, immer Wrapper-Implementation notwendig
- Informationsfusion wird unterstützt

SQL92: Bewertung für Bio-Datenbanken

- Vorteile
  - Breite Verwendung in kommerziellen Datenbank-Systemen
  - Gute Einbindung in Programmierspachen
  - Mächtig, effizient, stabil
- Nachteile
  - Für Anfrage muss zugrundeliegenes Datenschema im Detail bekannt sein
  - Eingeschränkte Unterstützung von semi-strukturierten Daten und Multimedia-Objekten
- Ausweg: SQL:2003 ??

Typsystem von SQL:2003

- Erweiterbares Typkonzept
  - vordefinierte Datentypen
  - konstruierte Typen (Konstruktoren): REF, Tupel-Typen (ROW-Typ), Kollektionstyp ARRAY
  - benutzerdefinierte Datentypen (User-Defined Types, UDT): Distinct Types und Structured Types
- Definition von UDTs unter Verwendung von vordefinierten Typen, konstruierten Typen und vorher definierten UDTs
- UDTs unterstützen u.a.
  - Kapselung
  - Vererbung (Subtypen)
- alle Daten werden weiterhin innerhalb von Tabellen gehalten
  - Definition von Tabellen auf Basis von strukturierten UDTs möglich
  - Bildung von Subtabellen (analog zu UDT-Subtypen)
- neue vordefinierte Datentypen
  - Boolean (Werte: TRUE, FALSE, UNKNOWN)
  - Large Objects (Binärdarstellung bzw. Texte), Multimedia-Objekte

Large Objects

- 3 neue Datentypen:
  - BLOB (Binary Large Object)
  - CLOB (Character Large Object): Texte mit 1-Byte Character-Daten
  - NCLOB (National Character Large Objects): ermöglicht 2-Byte Character-Daten für nationale Sonderzeichen (z.B. Unicode)
- Verwaltung großer Objekte im DBS (nicht in separaten Dateien)
  - umgeht große Datentransfers und Pufferung durch Anwendung; Zugriff auf Teilbereiche
- Beispiel Genexpression (stark vereinfacht):

```
CREATE TABLE GenExpression-Data
        (Exp-ID         INTEGER,
        Lab             VARCHAR (40),
        Annotation      CLOB (75K),
        Exp-Image       BLOB (12M) )
```

- indirekte Verarbeitung großer Objekte über Locator-Konzept (ohne Datentransfer zur Anwendung) (SQL/MED; IBM DB2 DataLink)

Large Objects (2)

- unterstützte Operationen
  - Suchen und Ersetzen von Werten (bzw. partiellen Werten)
  - LIKE-Prädikate, CONTAINS, POSITION, SIMILAR
  - Konkatenation ||, SUBSTRING, LENGTH, IS [NOT] NULL ...
- einige Operationen sind auf LOBs nicht möglich
  - Schlüsselbedingung
  - Kleiner/Größer-Vergleiche
  - Sortierung (ORDER BY, GROUP BY)
- Beispielanfrage Genexpression

```
CREATE TABLE GenExpression-Data
    (Exp-ID         INTEGER,
     Lab            VARCHAR (40),
     Annotation     CLOB (75K),
     Exp-Image      BLOB (12M) )
```

```
SELECT    Exp-Id, Exp-Image
FROM      GenExpression-Data
WHERE     Annotation.Contains("E. Coli")
```

SQL/MM (Multimedia and Applications Packages)*

- Multimedia/Anwendungs-Erweiterungen auf Basis von SQL:2003
  - Full Text (Datentypen und Funktionen zur Volltextsuche), z.B. für Bio-Annotationen
  - Spatial (räumliche Daten), z.B. für Tertiär- und Quartärstruktur von Proteinen
  - Still Image (unbewegte Bilder), z.B. für Genexpression-Images
  - Data Mining
- SQL/MM Full Text
  - Deutlich leistungsfähiger als CLOBs
  - Neue Datentypen `FullText` und `FT_Pattern` (Suchmuster)
  - Suchausdruck: `Contains`
  - Ähnlichkeitssuche: `Rank` (Grad der Übereinstimmung)
  - Kontextsuche: `in same sentence as, in same paragraph as`
  - linguistische Suche: `stemmed form of`

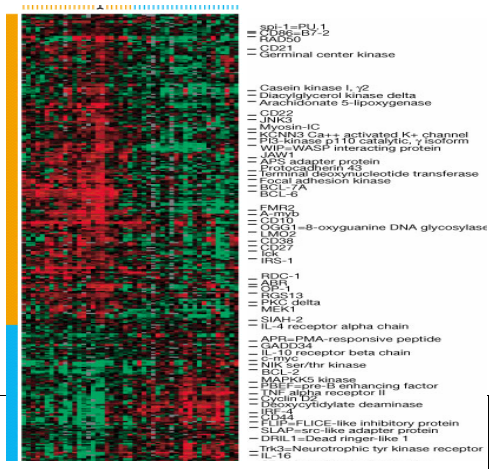
* J. Melton, A. Eisenberg: *SQL Multimedia and Application Packages (SQL/MM).* ACM Sigmod Record, SIGMOD Record 30(4): 97-102 (2001)

SQL/MM (2)

- SQL/MM Still Image
  - Datentyp `SI_StillImage` für digitale Rasterbilder (interne Speicherung als BLOB)
  - Methoden für Formatkonversionen, Skalierung, Rotation, Ausschnittbildung, Farbbestimmung ...
- Unterstützung der Suche durch Feature-Datentypen:
  - `SI_AverageColor` (durchschnittlicher Farbwert): Aufsummierung der einzelnen Farbwerte jedes Pixels (z.B.rot, grün, blau); Division durch Anzahl der Pixel
  - `SI_ColorHistogram` (Farbverteilung),
  - `SI_PositionalColor` (Farbverteilung bezüglich Bildlage): Bild wird in n mal m Rechtecke eingeteilt; Berechnung des Farbdurchschnittswertes eines Bereich analog zu SI_AverageColor
  - `SI_Texture` (Angaben zu Auflösung, Kontrast etc.)
  - Suchmethode `SI_Score` (liefert Maß der Übereinstimmung)
- SQL/MM Spatial
  - zunächst nur 0-, 1- und 2-dimensionale Objekte (Punkte, Linien, Flächenformen)
  - Bereitstellung standardisierter Typhierarchien wie `ST_Geometry` (Subtypen `ST_Point`, `ST_Curve`, `ST_MultiPolygon`) und `ST_SpatialRefSys` (räumliche Referenzsysteme)
  - Methoden zur Erzeugung von Geo-Objekten und Berechnungen (intersect, overlap, area ...)

SQL/MM (3)

- Beispiel Genexpression (stark vereinfacht)




```
CREATE TABLE GenExpression-Data
(Exp-ID       INTEGER,
Lab           VARCHAR (40),
Annotation    FULLTEXT,
Exp-Image     SI_STILLIMAGE)
```

```
SELECT Exp-Id, Exp-Image
FROM  GenExpression-Data
WHERE Annotation.Contains('"T-Lymphocyte"in same paragraph as  "Liver"')
        AND P.Annotation.Rank ("Hon-Hodgkin-Lymphoma") > 0.25
        AND WHERE 1.2 < SI_InitFeatureList
        (SI_findTexture(Sample-Exp-Image), 0.8)
        .SI_Append
        (SI_findPositionalColor(Sample-Exp-Image), 0.2)
        .SI_Score(Exp-Image)
```

SQL:2003: Bewertung

- SQL:2003-Standardisierung
  - Kompatibilität mit existierenden SQL-Systemen
  - Objektorientierung
  - Unterstützung von semi-strukturierten Daten und Multimedia
  - SQL/MED: DBS-Einbindung von externen Daten / Dateien
- Daher: Als Anfragesprache für Biodaten prinzipiell geeignet
- Aber: Bzgl. Datenschema muss a priori immer noch bekannt sein, welche Teile wie strukturiert und welche unstruktuiert sind (im Ggs. zu XML)

XML-basierte Anfragesprachen

- XQuery: W3C-Standardisierungsprozeß für einheitliche XML-Anfragesprache
- abgeleitet von vorangegangenen proprietären XML-Anfragesprachen (Quilt, XPath, XQL, XML-QL, ...) sowie SQL und OQL
- funktionale Anfragesprache
- komplexe Pfadausdrücke (basierend auf XPath 2.0), Funktionen, konditionale und quantifizierte Ausdrücke, Ausdrücke zum Testen/Modifizieren von Datentypen, Elementkonstruktoren
- FLWOR-Syntax (For ... Let ... Where ... Order By ... Return)

XQuery: FLWOR

- Generelle Vorgehensweise
  - Dokumentzugriff mit doc(url)
  - Knotennavigation mit XPath 2.0 (z.B. //book oder /book/author[Name = "Rahm"]
  - Variablenbindung im Kontext existierender Bindungen
  - Operationen auf den Knotenmengen
  - Erzeugung neuer Knoten
- FLWOR-Ausdrücke

FLWOR-expr	::=	(FOR-expr | LET-expr)+ WHERE-expr? ORDERBY-expr? RETURN-expr
FOR-expr	::=	**for** $var **in** expr (, $var **in** expr)*
LET-expr	::=	**let** $var := expr (, $var := expr)*
WHERE-expr	::=	**where** expr
ORDERBY-expr	::=	(**order by** | **stable order by**) OrderSpecList
OrderSpecList	::=	OrderSpec (, OrderSpec)*
OrderSpec	::=	expr OrderModifier
OrderModifier	::=	(**ascending** | **descending**)? (**empty greatest** | **empty least**)? (**collation** StringLiteral)?
RETURN-expr	::=	**return** expr

XQuery: FLWOR (2)

- LET-Ausdruck
  - bindet Menge von Werten an Variable
  - Menge wird geschlossen an Variable gebunden

```
let $n := //Book/Author/Name
return $n
```

- FOR-Ausdruck
  - für jedes Element der Ergebnismenge erfolgt Bindung an Variable (Iteration über die Menge)

```
for $n in //Book/Author/Name
return $n
```

  - Beispiel liefert gleiches Ergebnis wie bei LET, jedoch wird es anders abgearbeitet:
    - $n wird jeweils an Elemente der Sequenz von Autorennamen gebunden (für jeden Namen genau einmal)
    - RETURN wird für jede Bindung einmal ausgeführt
    - Zwischenergebnisse werden zu Gesamtergebnis zusammengefaßt

XQuery: Bio-Beispiel

- Ausgangs-schema*

DTD

```
<!ELEMENT seq (dbref, family?, full_seq, annotation*) >
  <!ATTLIST seq  id       ID       #REQUIRED
                 name     CDATA    #IMPLIED
                 length   CDATA    #IMPLIED >
<!ELEMENT dbref (database, unique_id, version?)>
<!ELEMENT database  (#PCDATA)>
<!ELEMENT unique_id (#PCDATA)>
<!ELEMENT version   (#PCDATA)>
<!ELEMENT family    (#PCDATA)>
<!ELEMENT full_seq  (#PCDATA)>
  <!ATTLIST full_seq      type  (dna | rna | aa)  #REQUIRED>
<!ELEMENT annotation (#PCDATA)>
```

Daten-satz

```
<seq id="my_seq" name="NUCLEAR RIBONUCLEOPROTEIN">
  <dbref>
    <database>SWISS-PROT</database>
    <unique_id>P09651</unique_id>
  </dbref>
  <family>RIBONUCLEASEN TYPE 15></family>
  <full_seq type="aa">
SKSESPKEPEQLRKLFIGGLSFETTDESLRSHFEQWGTLTDCVVMRDPNTKRSRGFGFVTYATVEEVDAAMNARPH
KVDGRVVEPKRAVSREDSQRPGAHLTVKKIFVGGIKEDTEEHHLRDYFEQYGKIEVIEIMTDRGSGKKRGFAFVTF
DDHDSVDKIVIQKYHTVNGHNCEVRKALSKQEMASASSSQRGRSGSGNFGGGRGGGFGGNDNFGRGGNFSGRGGFG
GSRGGGGYGGSGDGYNGFGNDGGYGGGGPGYSGGSRGYGSGGQGYGNQGSGYGGSGSYDSYNNGGGRGFGGGSGSN
FGGGGSYNDFGNYNNQSSNFGPMKGGNFGGRSSGPYGGGGQYFAKPRNQGGYGGSSSSSSYGSGRRF
  </full_seq >
  <annotation> This protein binds to RNA and … </annotation>
</seq>
```

---

* industry.ebi.ac.uk/~alan/XMLWorkshop/ Presentations/XML-Bio.ppt

XQuery: Bio-Beispiel (2)

```
<!ELEMENT seq (dbref, family?, full_seq, annotation*) >
  <!ATTLIST seq  id       ID       #REQUIRED
                 name     CDATA    #IMPLIED
                 length   CDATA    #IMPLIED >
<!ELEMENT dbref (database, unique_id, version?)>
<!ELEMENT database  (#PCDATA)>
<!ELEMENT unique_id (#PCDATA)>
<!ELEMENT version   (#PCDATA)>
<!ELEMENT family    (#PCDATA)>
<!ELEMENT full_seq  (#PCDATA)>
  <!ATTLIST full_seq      type  (dna | rna | aa)  #REQUIRED>
<!ELEMENT annotation (#PCDATA)>
```

– Liste alle Elemente, die etwas zu "RIBONUCLE..." enthalten

```
doc("http://data.bio-server.bio-university.edu/biodata.xml")//*[contains(text(), 'RIBONUCLE')]
```

– Liste alle Aminosäuresequenzen mit Namen auf, welche die Teilsequenz "SFETTDESLRSH" enthalten, in alphabetischer Reihenfolge

```
for $s in doc("http://.../biodata.xml")//seq
where       $s/full_seq/@type = aa and
            $s/full_seq[contains(text(), 'SFETTDESLRSH')]

order by $s/@name
return $s/@name
```

XQuery: Bio-Beispiel (3)

```
<!ELEMENT seq (dbref, family?, full_seq, annotation*) >
  <!ATTLIST seq  id       ID       #REQUIRED
                 name     CDATA    #IMPLIED
                 length   CDATA    #IMPLIED >
<!ELEMENT dbref (database, unique_id, version?)>
<!ELEMENT database  (#PCDATA)>
<!ELEMENT unique_id (#PCDATA)>
<!ELEMENT version   (#PCDATA)>
<!ELEMENT family    (#PCDATA)>
<!ELEMENT full_seq  (#PCDATA)>
  <!ATTLIST full_seq      type  (dna | rna | aa)  #REQUIRED>
<!ELEMENT annotation (#PCDATA)>
```

– Geben Sie zu jeder Proteinfamilie die durchschnittliche Länge der Proteinsequenzen aus

```
for $f in distinct-values(doc("http://.../biodata.xml")//family)
  where $f/../full-seq/@type = aa
  let $a := avg(doc("http://.../biodata.xml")//seq[family = $f]/full-seq/string-length( text() ) )
  return
       <family>
              <name> {$f/text()} </name>
              <avglength> {$a} </avglength>
       </family>
```

XML/XQuery: Bewertung

- Standardisiert, objektorientierte Mächtigkeit
- Als Anfragesprache für Bio-Daten prinzipiell geeignet
  - Gute Unterstützung von semi-struktrierten Daten
  - Anfragen auch mit sehr wenig Metawissen über Struktur der Bio-Daten möglich
- Nachteile
  - Noch unklare Situation bzgl. Produkten (Native XML-DBMS vs OR-DBMS mit XML-Modulen)
  - Keine Untersützung von Multimedia-Objekten