

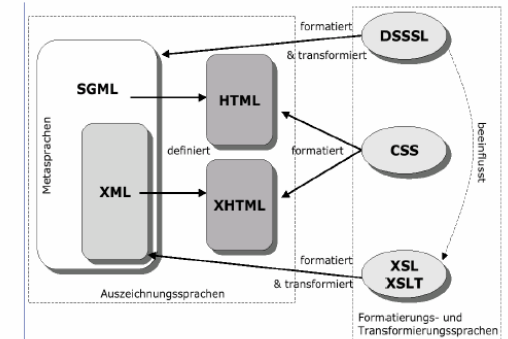
Vorlesungsplan

1. Übersicht
2. Datenmodelle
 - Textdateien, Entry-Modell
 - Relationale DB
 - XML
3. Genom-DB
4. Genexpressions-DB
5. Protein-DB
6. Pathway-DB
7. Publikations-DB
8. Zugriff und Anfragesysteme
9. Datenintegration von Bio-DB
10. Webservices

XML basierte Modelle, Spezielle Modellierungsaspekte

n XML: Extensible Markup Language

- Version 1: 1998
- Einschränkung von SGML, Erweiterung von HTML
- W3C-Standard
- Kern einer Sprachgruppe: XSL, Xpath, XQuery, XLink, ...



n Standard zur Definition von Austauschformaten

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<default>
  Mode W.S., DefHippis V.R.
</default>
<default>
  The window of taxonomic resolution ...
</default>
```

XML, DTD und XML Schema

n Document Type Definition (DTD)

- Definition erlaubter Elemente und ihrer Attribute
- Assoziationen durch ID, IDREF
- Unzureichende Datentypisierung

n XML-Schema: Erweiterung

- Constraints und Kardinalitäten
- Vordefinierte und benutzerdefinierte Datentypen
- Einfache und komplexe Datentypen

n XML-Dokument ist

- Wohlgeformt: Entspricht XML Syntax
- Gültig/valid bzgl. einer DTD: Entspricht einer gegebenen DTD

n Speicherung

- Flatfile, XML-Datenbank (Tamino, ...), Relationales Mapping

XML in der Bioinformatik

- BIOML (BIOpolymer Markup Language)
 - Information über Experimente mit Proteinen, Genen, ...
 - Verbindet physikalisches Objekt mit Experimentaldaten
 - <http://www.bioml.com/BIOML/>
- BSML (Bioinformatic Sequence Markup Language)
 - Daten-Modell für Bezüge zwischen Sequenzen und Phänomenen auf verschiedenen Ebenen (molekular bis Genomebene)
 - <http://www.labbook.com>
- PSDML (Protein Sequence Database Markup Language)
 - Austausch für PIR, <http://pir.georgetown.edu/>
- GAME (Genome Annotation Markup Elements)
 - Informationen über Sequenzabschnitte, (automatisch oder manuell)
 - <http://www.bioxml.org/Projects/game/game.dtd.html>

XML in der Bioinformatik

- SBML (Systems Biology Markup Language)
 - Beschreibung für Modelle (Pathways), <http://www.sbml.org>
- CellML
 - Computerbasierte biologische Modelle, <http://www.cellml.org>
- MAGE-ML (Microarray Gene Expression)
 - Informationen über Genexpressions-Experimente
 - Design,
 - Hersteller Infos,
 - Experiment Setup und Ausführung
 - Genexpressionsdaten
 - Datenanalyse
 - <http://www.mged.org/Workgroups/MAGE/>

Elemente und Attribute

- Zwei Arten um Daten zu kapseln
- Attribute
 - Meist 1:1 Beziehung zum Datenelement
 - Metainformationen, z.B. Maßeinheiten
- Daten Elemente
 - Normale Syntax mit Start und Ende, <gene>CD4</gene>
 - Leere Syntax, keine gekapselten Daten außer Attribute
 - <db_xref db="EC" dbkey="2.7.4.0"/>
 - Effiziente Art Attribute zu speichern
 - Strukturelemente ohne Daten -> hierarchische Informationen
 - Leere Elemente als Begrenzer für Listen

Ein einfaches XML Fragment

```
<colorimeter_reading>
  <device> X-Rite Digital Swatchbook </device>
  <patch> cyan </patch>
  <RGB resolution=8>
    <red> 0 </red>
    <green> 255 </green>
    <blue> 255 </blue>
  </RGB>
</colorimeter_reading>
```

Analyse: wurden die XML-Elemente gut genutzt

- Datenelemente
 - Sind dies Daten oder sind hier eigentlich Metadaten gespeichert (Informationen über andere Datenelemente)?
- Attribute
 - Beschreibt das Attribut wie Datenelemente interpretiert, verwendet oder präsentiert werden?
 - Beschreibt das Attribut wirklich Metadaten, und nicht eigentlich ein Datenelement?
 - Bezieht es sich tatsächlich auf alle Datenelemente in seinem Bereich?
- Tags (XML Label)
 - Hilft ein Tag alle Datenelemente in seinem Bereich zu beschreiben?
- Gruppierungen (Hierarchie-Ebenen)
 - Stehen alle Elemente einer Gruppe in der Beziehung, welche die Elternknoten beschreiben?
 - Sind die Beziehungen der Elemente einer Hierarchieebene eindeutig?

Interpretation des XML Fragments

• Fragment

```
<colorimeter_reading>
  <device> X-Rite Digital Swatchbook </device>
  <patch> cyan </patch>
  <RGB resolution=8>
    <red> 0 </red>
    <green> 255 </green>
    <blue> 255 </blue>
  </RGB>
</colorimeter_reading>
```

• Kontext

```
<colorimeter_reading><device> X-Rite Digital Swatchbook
<colorimeter_reading><patch> cyan
<colorimeter_reading><RGB resolution=8><red> 0
<colorimeter_reading><RGB resolution=8><green> 255
<colorimeter_reading><RGB resolution=8><blue> 255
```

Gruppierungen und Bereiche

• Gruppierungen

- "-colorimeter_reading-" enthält
 - "-device-", "-patch-" und "-RGB-" (plus dessen Kinder)
 - "-colorimeter_reading-" ist die Wurzel, d.h. alles steht dazu in Beziehung
- "-RGB-" enthält
 - "-red-", "-green-", and "-blue-"
 - diese Elemente haben alle die gleiche Beziehung zu einander

• Bereiche der Attribute

- "resolution=8" hat die Elemente "-red-", "-green-" und "-blue-" in seinem Bereich.
- "resolution=8" kann sinnvoll auf alle diese Element angewendet werden

Häufige Fehler beim Modellieren mit XML

- Unvollständiger Kontext, der beschreiben soll was ein Datenelement ist (Unvollständige Nutzung von Tags)
- Unvollständige Instruktionen wie Datenelementen interpretiert werden sollen (Unvollständige Nutzung von Attributen)
- Nutzung von Attributen als Datenelementen
- Nutzung von Datenelementen als Metadaten (z.B. Name/Wert Paaren)
- Unnötige, nicht in Beziehung stehende oder redundante Tags (schlechte Hierarchiekonstruktion)
- Attribute, die nichts mit der Interpretation von Datenelementen zu tun haben (schlechte Hierarchiekonstruktion oder falsche Anwendung von Attributen)
- Syntaktische Probleme
 - Special Character in Daten
 - Datentyp ist abhängig von anderem Attributewert (Metadaten)
 - Objektreferenzen müssen ersetzt werden, wenn sie XML Markup enthalten
 - Datenfeld kann Listen enthalten
- Strukturprobleme
 - sehr viele Strukturelemente
 - stark verschachtelte Struktur
 - Datenobjekt ist auf mehrere Dateien verteilt
 - Redundante Einträge

Attribute als Datenelemente

• XML ohne Datenelemente

```
<colorimeter_reading>
  <device> X-Rite Digital Swatchbook </device>
  <patch> cyan </patch>
  <RGB resolution=8 red=0 green=255 blue=255 />
</colorimeter_reading>
```

• resolution=8

- beschreibt echte Meta-Daten

• red=0 green=255 blue=255

- beschreiben eigentlich keine Daten, sondern sind selbst Daten
- zur Interpretation ist resolution=8 erforderlich

Datenelemente als Metadaten

- XML-Datenelemente als Metadaten

```
<colorimeter_reading>
  <device> X-Rite Digital Swatchbook </device>
  <patch> cyan </patch>
  <RGB>
    <item>
      <band> red </band>
      <value> 0 </value>
    </item> <item>
      <band> green </band>
      <value> 255 </value>
    </item> <item>
      <band> blue </band>
      <value> 255 </value>
    </item>
  </RGB>
</colorimeter_reading>
```

- Informationskontext
 - Zeilen 3,5,7 enthalten keine Daten, sondern Metadaten

```
<colorimeter_reading><device> X-Rite Digital Swatchbook
<colorimeter_reading><patch> cyan
<colorimeter_reading><RGB><item><band> red
<colorimeter_reading><RGB><item><value> 0
<colorimeter_reading><RGB><item><band> green
<colorimeter_reading><RGB><item><value> 255
<colorimeter_reading><RGB><item><band> blue
<colorimeter_reading><RGB><item><value> 255
```

Ungünstige Nutzung von Tags

- Nutzung von Name/Wert Paaren

```
<colorimeter_reading>
  <device> X-Rite Digital Swatchbook </device>
  <patch> cyan </patch>
  <mode> RGB </mode>
  <band> red </band>
  <value> 0 </value>
  <band> green </band>
  <value> 255 </value>
  <band> blue </band>
  <value> 255 </value>
</colorimeter_reading>
```

- Informationskontext
 - Zeile 3 sind zwar auch Daten, bildet aber den Kontext für die folg.Zeilen
 - Kontext-Info fehlt

```
<colorimeter_reading><device> X-Rite Digital Swatchbook
<colorimeter_reading><patch> cyan
<colorimeter_reading><mode> RGB
<colorimeter_reading><band> red
<colorimeter_reading><value> 0
<colorimeter_reading><band> green
<colorimeter_reading><value> 255
<colorimeter_reading><band> blue
<colorimeter_reading><value> 255
```

GAME

- n DTD + Tools für Austausch von Genom-Annotationen

- n Ergänzt GFF*-Format

- GFF: standardisiert Darstellung der Genstrukturen
- GAME erweitert GFF mit Metainformationen (Annotationen)

- n Überschneidung mit OMG-LSR[†]
BSA[‡]

GAME Semantics

- Annotation
 - "A collection of features found on an associated set of sequences"
- Features
 - "Conclusions describing intervals on different sequences. Supported by analytical evidence"
- Analyses
 - "Computer or biological experiments on a sequence. Results apply to sequence interval"
- Sequences
 - "Biological sequences in which we're interested"

* GeneFinding File Format: Format zur Abspeicherung von Genstrukturen

† Life Science Research Domain Task Force der OMG

‡ Biomolecular Sequence Analysis

GAME: DTD Ausschnitt

```
<!ELEMENT game ANY>
...
<!ELEMENT offset (#PCDATA)>
<!ELEMENT length (#PCDATA)>

<!-- ENTITY % site_operator
" site_operator (less_than | greater_than)" -->

<!-- ELEMENT fuzzy_start (span) -->
<!-- ENTITY fuzzy_start
%site_operator; #IMPLIED -->

<!-- ELEMENT fuzzy_end (span) -->
<!-- ENTITY fuzzy_end
%site_operator; #IMPLIED -->

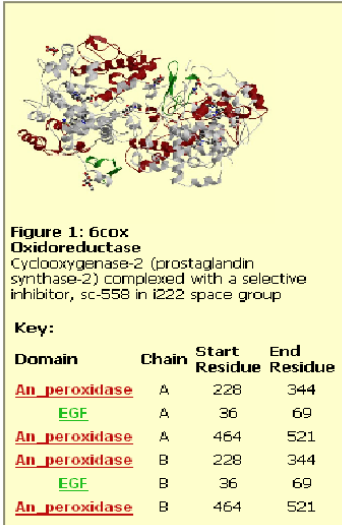
<!-- ELEMENT fuzzy_span (fuzzy_start, fuzzy_end) -->

<!-- ELEMENT span (offset, length) -->
<!-- ENTITY span
between (TRUE) #IMPLIED
either_dir (TRUE) #IMPLIED -->

Location: 340..565
<span>
  <offset>339</offset>
  <length>225</length>
</span>

Location: <345..500
<fuzzy_span>
  <fuzzy_start site_operator="less_than">
    <span>
      <offset>344</offset>
      <length>1</length>
    </span>
  </fuzzy_start>
  <fuzzy_end>
    <span>
      <offset>499</offset>
      <length>1</length>
    </span>
  </fuzzy_end>
</fuzzy_span>
```

GAME: Pfam Beispiel



Beispiel für Protein-Domain: EGF-like domain (Pfam-ID PF00008)

A sequence of about thirty to forty amino-acid residues long found in the sequence of epidermal growth factor (EGF) has been shown PUB00001077, PUB00001077, [MEDLINE:84117505], [MEDLINE:91145344], [MEDLINE:85063790], PUB00004964 to be present, in a more or less conserved form, in a large number of other, mostly animal proteins.

The list of proteins currently known to contain one or more copies of an EGF-like pattern is large and varied. The functional significance of EGF domains in what appear to be unrelated proteins is not yet clear. However, a common feature is that these repeats are found in the extracellular domain of membrane-bound proteins or in proteins known to be secreted (exception: prostaglandin G/H synthase). The EGF domain includes six cysteine residues which have been shown (in EGF) to be involved in disulphide bonds. The main structure is a two-stranded β -sheet followed by a loop to a C-terminal short two-stranded sheet. Subdomains between the conserved cysteines vary in length.

GAME: XML Darstellung von PF00008

```
<computational analysis seq="dmNotch">
<date>08/26/1999</date> <program>hmmpfam</program> <version>2.1.1</version>
<database>
  <name>Pfam</name> // Bezug auf "Pfam" (Protein families database of
    // alignments and HMMs
  <date>god (and Sean Eddy) knows when it was created</date>
  <version>4.1</version>
</database>
<result_set>
  <dbxref>
    <database> <name>Pfam</name> </database>
    <unique_id>PF00008</unique_id> // Pfam accession number
  </dbxref>
  <output> <type>Description</type><value>EGF-like domain</value> </output>
  ...
  <result_span>
    <score> 22.6 </score>
    <type>Motif</type>
    <subtype>EGF</subtype> // EGF: epidermal growth factor (family)
    <seq_relationship seq="dmNotch" type="query">
      <span>
        <offset>62</offset>
        <length>32</length>
      </span>
      <alignment>CTSV-GCQNGGTCVTQLN-----GKTYCACDSH-----YVGDYC</alignment>
    </seq_relationship>
    <seq_relationship seq="EGF" type="subject">
      <span>
        <offset>0</offset>
        <length>44</length>
      </span>
      <alignment>CapnnpCsgGtCvntpggssdnfggytCeCpPgdylytGkrC</alignment>
    </seq_relationship>
  </result_span>
</result_set>
</computational analysis>
```

XML in der Bioinformatik: Bewertung

n Gut geeignet für semi-strukturierte Biodaten

n Vorteile (insb. gegenüber Entry-basiertem Modell)

- Industriestandard, viele Tools (Editoren)
- DTD generierbar aus UML-, Java-Spezifikationen, ...
- Effiziente Parser
- Unterstützung durch relationale DB-Hersteller (IBM, Oracle, ...)
- Zunehmend XML-Datenbanken verfügbar (Tamino etc.)
- Strukturierte Anfragen (XQuery) und Textsuche möglich

n Nachteile

- Dokumente sehr lang, daher oft nicht sehr gut lesbar
- Ohne DTD: Keine Dokumentvalidität, keine Semantik für Datenaustausch
- Mit DTD: Geringere Flexibilität, Dokumente evtl. ungültig bei Änderungen

Datenmodelle: Zusammenfassung

n "Austauschformate"

- Entry-based
- ASN.1
- XML

n Speichern und Anfragen

- Relationales Modell
- Objektorientiertes/Objektrelationales Modell

n Vorteile der Flatfiles nicht unterschätzen

- Viele Bio-Einrichtungen ohne RDBMS / Informatiker

n I.d.R. mehrere Formate/Datenmodelle in *einem* Bio-Projekt

Spezielle Modellierungsaspekte

- n Objektidentifikation
- n Versionierung
- n Widersprüchliche Daten

Objektidentifikation

- Trivial?
- „Namen“ einer Sequenz in Genbank
 - Locus name
 - Accession number
 - GI number
 - NID number
 - Accession.Version
 - Weitere Ids in EMBL / DDBJ
- Existierende Crossreferenzen wechseln dadurch nicht
 - Falsche Verweise
 - Tote Verweise (Dangling Pointer)

Identifikation

- Aspekte
 - Definition von Objekten zur Untersuchung
 - Identifikation von Objekten der realen Welt
 - Identifikation von Datenbankobjekten
- Alle drei in der Bioinformatik nicht trivial
 - „Objekte“ verändern sich: EST-Cluster, Proteindomäne, etc.
 - Identifikation identischer Objekte i.d.R. schwierig
 - unzureichenden Daten (Clonennamen)
 - Fehlenden Standards (Protein / Gennamen)
 - Evolution: Splitting, Merging, Deleting, Versioning, ...

Modelle von Identifikatoren

- Semantikfrei
 - Object-Ids
 - „Surrogate“ Keys
- Semantikbehaftet
 - „Sprechende Schlüssel“
 - Beispiel: „CYC_BOVIN“ = „Protein_Species“ (SP)
 - Problem: Können sich ändern
 - Neue Erkenntnisse über Proteinfunktion
 - Neue Erkenntnisse über Vorkommen in Species
 - Heirat
 - ...

Schlüssel in MDB

- Wissen über Objekte wächst ständig
 - Korrekturen – andere Annotation, Sequenz, ...
 - Objektverschmelzung
 - Zwei Loci sind nur einer
 - Zwei Proteinsequenzen sind redundant
 - Objektteilung
 - Ein Locus sind eigentlich zwei
 - Ein Clone sind zwei (Änderung durch Vermehrung)
- Crossreferenzen sehr weit verbreitet
 - In anderen Datenbanken und in Publikationen
- Links sollen erhalten bleiben
- Versionierung notwendig

Beispiel GenBank

- Versuch 1: „Locus name“
 - Sprechend: „HUMHBB“ – „Human Betaglobin region“
 - Definitionen verändern sich - als Ids abgelöst
 - ID und Daten können sich ändern
- Versuch 2: Accession-Number
 - Eindeutige, globale ID für jede Submission
 - Keine Versionierung
 - ID bleibt, aber Daten können sich ändern
- Versuch 3: GID (Genbank ID)
 - Eindeutige, interne ID für jede Version einer Submission
 - Versionen von Entries – unterschiedliche gid
 - ID und Daten immer gleich
 - Aber: Zugriff auf „aktuellsten“ Entry nicht möglich (Nur über Comment line)

Beispiel GenBank 2

- Versuch 4: NID (Nucleotide ID)
 - Eingeführt als übergreifende GID für EMBL/Genbank/DDBJ
 - Versionen von Entries – unterschiedliche NID
 - ID und Daten immer gleich
 - Aber: Zugriff auf „aktuellsten“ Entry nicht möglich (Nur über Comment line)
 - Obsolet sein Versuch 5
- Versuch 5: Accession-No.Version (Seit 1999)
 - Accession-No ist eindeutige, globale ID für jede eingesandte Sequenz
 - Bei Updates eines Eintrags – neue Version, aber keine neue Accession-no
 - Referenzierung der aktuellsten Version: Accession_no
 - Gleiche Accession-No -> gleiches Objekt
 - Gleiche Accession-No.Version -> gleiche Daten

Objektnamen bleiben schwierig

- Semantische Keys (Namen) haben viele Vorteile
 - Menschen verstehen sie
- Standards für Namen setzen sich bislang nicht durch
- Clone – kein Standard
- Loci – kein Standard
- Proteinnamen – kein Standard
- Gennamen – halbherziger Standard (HUGO)
- Enzymnamen – schlechter Standard (Reaktion statt Objekt)
- ...

Objekt Identifikation, Beispiel (Stand 2001)

- TP53 (neu), P53 (alt)
- SIRT1 (neu), SIR2L1 (alt)

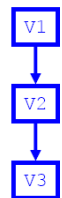
HUGO name	GDB	GenAtlas	OMIM	GeneCards	LocusLink
TP53	1	33	52	22	13
P53	1	17	188	69	63
SIRT1	1	0	5	1	2
SIR2L1	0	0	1	1	2

Versionierung

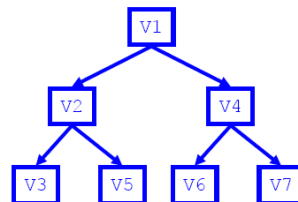
- Das molekularbiologische Wissen wächst und verändert sich ständig
 - Ensembl: ca. 40% Änderungen pro Release
 - Swiss-Prot: Ca. 30% der Einträge ändern sich pro Release
- Analysen benutzen bestimmte Version (aktuellste)
- **Nachvollziehbarkeit von Analysen nicht gewährleistet**
 - wenn Daten eines bestimmten Release nicht widerherstellbar sind
- **Versionierung von Daten essentiell**
 - Version der Datenbank (Release)
 - Versionen von Objekten / Attributen
- Verschiedene Aspekte
 - Speichern von verschiedenen Versionen
 - Zugriff auf (alte) Versionen
 - Referenzieren auf Versionen
 - Vergleich von Versionen (Deltas erstellen)

Versionierungsmodelle

Linear



Hierarchisch



- Zeitliche Reihenfolge fest
- Standardverfahren: Release / Version

- Zeitliche Reihenfolge halbgeordnet
- Verfahren von CVS
- Deutlich komplexer

Weitere Kriterien

- Granularität
 - Datenbank
 - Tabelle
 - Objekt (Tuple)
 - Attribut
- Repräsentation von Änderungen
 - Komplette Versionen: Speicherverbrauch
 - Deltas: Berechnung von alten Versionen teuer
- Repräsentation
 - Versionsnummern (Vorgänger / Nachfolger)
 - Timestamps (Nur linear)

Speicherung von Deltas

- Urzustand: A_0 , aktuelle Version: A_n
- Schreibweise: $\text{Delta}(a_x \rightarrow a_y) \equiv (a_y, a_x)$
- Variante 1: $(a_n, a_{n-1}), (a_{n-1}, a_{n-2}), \dots, a_0$
 - Aktuelle Version in n Schritten berechenbar
 - Kompakte Deltas
- Variante 2: $(a_n, a_0), (a_{n-1}, a_0), \dots, a_0$
 - Aktuelle Version in 2 Schritten berechenbar
 - Zunehmend große Deltas
 - a_0 muss zugreifbar bleiben (Archivierung)
- Variante 3: $a_n, (a_{n-1}, a_n), (a_{n-2}, a_{n-1}), \dots, (a_0, a_1)$
 - Aktuelle Version immer vorhanden
 - Alte Versionen schwer zugreifbar
 - Kompakte Deltas

Speicherung von Deltas 2

- Variante 4: $a_n, (a_{n-1}, a_n), (a_{n-2}, a_n), \dots, (a_0, a_n)$
 - Aktuelle Version sofort zugreifbar
 - Erzeugen einer neuen Version: Neuberechnung aller Deltas
 - Zunehmend große Deltas
- Variante 5: $a_n, a_{n-1}, a_{n-2}, \dots, a_1, a_0$
 - Keine Speicherung von Deltas
 - Höchster Speicherplatzverbrauch
 - Alle Versionen im direkten Zugriff

Existierende Versionierung in MDB

- Linear
- Granularität
 - Datenbank Releases
 - Versionierung von Einträgen (Entry-Based Modell)
- Repräsentation
 - Komplette Versionen, keine Deltas (Variante 5)
 - Versionsnummern (Genbank)
 - Änderungsdatum (Swiss-Prot)
- Anmerkung: Wenige MDB „löschen“ Objekte
 - Externe Referenzen vorhanden
 - Stattdessen (Ids bleiben gültig)
 - Merging von Objekten
 - Inaktivierung

Senario

- Einspielen von Datenbankreleases in ein **versioniertes Schema**
- Applikation vergleicht aktuelles Release R_x mit Vorgängerrelease R_{x-1}
 - Objekt K in R_x und nicht in R_{x-1} : `INSERT k`
 - Objekt K in R_x und in R_{x-1}
 - Unterschied: `UPDATE K`
 - Kein Unterschied: Nichts tun
 - Objekt K nicht in R_x aber in R_{x-1} : `DELETE k`
- Annahme: **Keine Schlüsselwiederverwertung**
 - Schlüssel K wird niemals für verschiedene Objekten benutzt
 - Kann z.B. durch Sequenz sichergestellt werden
 - Objekte können nicht wieder belebt werden

Versionen im relationalen Modell

- Aufgabe
 - RDBMS soll Versionen verwalten
 - Lineare, tuple-basierte Versionierung
- Anforderungen: Zugriff muss möglich sein auf
 - Aktuelle Version
 - Zustand der Datenbank zu beliebigem Zeitpunkt d_0
- 2 Varianten
 - Single Table
 - Schattentabellen

Variante 1: Single-table

- Erweiterung jeder Tabelle T um Attribute
 - Versionsnummer V
 - ALIVE Flag A
 - VALIDFROM D
 - Schlüsselveränderung $K \rightarrow (K+V)$

Variante 1: INSERT

- **INSERT** Objekt K in T
 - Gibt es K schon in T?
 - Nein
 - INSERT K INTO T (V=0, A=true, D=SYSDATE)
 - Ja
 - Letzte Version von K in T finden (V_x)
 - INSERT K INTO T (V= V_x+1 , A=true, D=SYSDATE)

Variante 1: DELETE

- **DELETE** Objekt K aus T
 - Gibt es K schon in T?
 - Nein
 - Nichts tun
 - Ja
 - Letzte Version von K in T finden (K_{alt} mit V_x)
 - $K_{alt}.A=T$?
 - Ja
 - » INSERT K INTO T (V= V_x+1 , A=false, D=SYSDATE)
 - Nein
 - » Nichts tun
- Werte von K beim INSERT sind beliebig; es zählt V, D und A

Variante 1: UPDATE

- **UPDATE** Objekt K in T
 - K in T vorhanden?
 - Nein
 - Nichts tun
 - Ja
 - Letzte Version von K in T finden (K_{alt} mit V_x)
 - $K_{alt}.A=T$?
 - Ja
 - » INSERT K INTO T ($V=V_x+1$, $A=true$, $D=SYSDATE$)
 - Nein
 - » Nichts tun – Fehler?

Variante 1: SELECT

- **SELECT** aktuelle Version von k_0
- **SELECT** alle Tupel zum Zeitpunkt d_0

```
SELECT *  
FROM t  
WHERE a=T' AND  
       k=k0 AND  
       d = (SELECT MAX(d)  
            FROM t t2  
            WHERE t2.K=k0)
```

```
SELECT *  
FROM t t1  
WHERE a=T' AND  
       d<=d0 AND  
       d = (SELECT MAX(d)  
            FROM t t2  
            WHERE t2.K=t1.K AND  
                  t2.d<=d0)
```

Was könnte man
sparen ?

Variante 1: Bewertung

- Varianten
 - Versionsnummer weglassen (Datum reicht)
 - Markierung der aktuellen Version hilfreich
 - Nur in aktuellster Version gilt $A=T$
 - INSERT/UPDATE erfordert 1INSERT + 1UPDATE
 - Zugriff auf aktuelle Version schneller
- Bewertung
 - INSERT / DELETE / UPDATE erfordern Trigger
 - Verlangsamung bei SELECTs
 - Verlangsamung durch wachsende Tabelle
 - Syntaxkomplexität durch Views abfangen

Variante 2: Schattentabellen

- Pro Tabelle T anlegen einer Tabelle T^s
 - Zusätzliche Attribute
 - Versionsnummer V
 - VALIDUNTIL D
 - Schlüssel in T^s : $K \rightarrow (K+V)$
- T bleibt unverändert
- T^s speichert alte Versionen
- T speichert nur aktuellste Version

Variante 2: INSERT

- **INSERT** Objekt K in T
 - K in T vorhanden?
 - Nein
 - INSERT K in T
 - Ja: Sei dies das Tupel K_{alt}
 - Letzte Version von K in T^S finden (V_x)
 - V_x existiert: INSERT K_{alt} INTO T^S ($V=V_x+1$, $D=SYSDATE$)
 - V_x existiert nicht: INSERT K_{alt} INTO T^S ($V=0$, $D=SYSDATE$)
 - DELETE K_{alt} FROM T
 - INSERT K INTO T
- Tupel wird von T nach T^S verschoben

Variante 2: DELETE

- **DELETE** Objekt K aus T
 - K in T vorhanden?
 - Nein
 - Nichts tun
 - Ja: Sei dies das Tupel K_{alt}
 - Letzte Version von K in T^S finden (V_x)
 - V_x existiert: INSERT K_{alt} INTO T^S ($V=V_x+1$, $D=SYSDATE$)
 - V_x existiert nicht: INSERT K_{alt} INTO T^S ($V=0$, $D=SYSDATE$)
 - DELETE K_{alt} FROM T
- Objekt in T vorhanden - Objekt ist gültig
- Objekt nicht in T vorhanden, aber in T^S - Objekt war gültig bis D

Variante 2: UPDATE

- **UPDATE** Objekt K in T
 - K in T vorhanden?
 - Nein
 - Nichts tun
 - Ja: Sei dies das Tupel K_{alt}
 - Letzte Version von K in T^S finden (V_x)
 - V_x existiert: INSERT K_{alt} INTO T^S ($V=V_x+1$, $D=SYSDATE$)
 - V_x existiert nicht: INSERT K_{alt} INTO T^S ($V=0$, $D=SYSDATE$)
 - DELETE K_{alt} FROM T
 - INSERT K INTO T

Variante 2: SELECT

- **SELECT** aktuelle Version von k_0
 - ```
SELECT *
FROM t
WHERE k=k0
```
- **SELECT** alle Tupel k zum Zeitpunkt  $d_0$ 
  - Kompliziert ....
  - Erfordert Zugriff auf T und  $T^S$
  - Vorsicht: Datum in  $T^S$  gibt **Ende der Gültigkeit** an

## Fallunterscheidung

|                            | Gelöscht vor $d_0$                                            | Gelöscht nach $d_0$                                                                                    | Bisher nicht gelöscht                                                                                                 |
|----------------------------|---------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| Letzte Änderung vor $d_0$  | K nicht in T;<br>Kein Eintrag in T <sup>s</sup> mit $d > d_0$ | K nicht in T;<br>Kleinsten Eintrag in T <sup>s</sup> mit $d > d_0$ nehmen (es gibt nur einen) <b>1</b> | K in T nehmen;<br>Bedingung: kein Eintrag in T <sup>s</sup> mit $d > d_0$ <b>2</b>                                    |
| Letzte Änderung nach $d_0$ | Fall unmöglich                                                | K nicht in T;<br>Kleinsten Eintrag in T <sup>s</sup> mit $d > d_0$ nehmen (es gibt mehrere) <b>3</b>   | K in T vorhanden (ignorieren);<br>Kleinsten Eintrag in T <sup>s</sup> mit $d > d_0$ nehmen (es gibt mehrere) <b>4</b> |

## In SQL

```

SELECT *
FROM t
WHERE NOT EXISTS
 (SELECT *
 FROM t_s
 WHERE t.k = t_s.k AND
 t_s.d > d_0)

UNION

SELECT *
FROM t_s
WHERE t_s.d > d_0 AND
 t_s.d =
 (SELECT MIN(d)
 FROM t_s t2
 WHERE t_s.k=t2.k AND
 t_s.d > d_0)

```

Fall 2

Fälle 1,3,4

## Variante 2: Bewertung

- Bewertung
  - INSERT / DELETE / UPDATE erfordern Trigger
  - Sehr schneller Zugriff auf aktuellste Version (kein Unterschied zu nicht-versioniert)
  - Komplexer Zugriff auf Zustand zu Zeitpunkt d
  - Komplizierteres INSERT /DELETE / UPDATE
  - Gut geeignet zur Archivierung (T bleibt unangetastet)

**Keine Objekte (Schlüssel) wiederbeleben !**

## Vergleich

- Häufige Änderungen, eher wenig Lesezugriffe - Variante 1
- Seltenere Änderungen, vor allem Zugriff auf aktuellste Version – Variante 2
- **Variante 2 eher für MDB geeignet**
- Außerdem zu klären
  - Referenzen / Fremdschlüssel auf versionierte Objekte
    - Entry-based Daten haben keine Referenzen
  - Identifikation von Änderungen (Delta Berechnung)

# Widersprüchliche Daten

- n Experimentergebnisse können (und werden) sich "widersprechen"
  - "Widersprüche" sind eine Frage des zugrundeliegenden, angenommenen kausalen Modells
  - Auflösung der Widersprüche i.d.R. nicht möglich; zusätzliches Experiment liefert oft nur weiteres Ergebnis
  - "Widersprüche" lösen sich bei Modellüberarbeitung oft auf
- n Rohdaten versus Ergebnisdaten
- n Metadaten wichtig
  - Datenquelle
  - Motivation
  - Zeitpunkt
  - Abschätzung der Datenqualität
  - Vergleichbarkeit der Ergebnisse
- n Bedeutung der Versionierung

# Zusammenfassung

- n Datenmodelle
  - Flat-file basierte Datenmodelle traditionell weit verbreitet
  - XML-basierte Modelle von zunehmender Bedeutung
  - langfristige Rolle relationaler/objektrelationaler Modelle noch unklar
- n Für Bio-DB wichtige Modellierungsaspekte
  - Objektidentifikation
  - Versionierung
  - widersprüchliche Daten