Datenbanken in der Bioinformatik

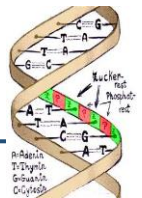
Kapitel 4
Modellierung von Bio-Datenbanken

Wintersemester 2014/15
Dr. Anika Groß
Universität Leipzig, Institut für Informatik, Abteilung Datenbanken
http://dbs.uni-leipzig.de

UNIVERSITÄT LEIPZIG

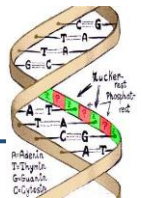
Gliederung

1. Motivation und Grundlagen
2. Bio-Datenbanken
3. Datenmodelle und Anfragesprachen
4. Modellierung von Bio-Datenbanken
5. Sequenzierung und Alignments
6. Genexpressionsanalyse
7. Annotationen
8. Matching
9. Datenintegration: Ansätze und Systeme
10. Versionierung von Datenbeständen
11. Neue Ansätze



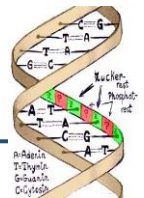
Lernziele

- Kennen und Anwenden
  - Modellierung verschiedener biologischer Konzepte
  - Modellierungsalternativen und ihre Anwendungsszenarien



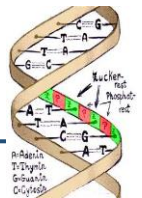
Gliederung

- Motivation
- Anforderungen
- Konzeptionelle Modellierung, Beispiel
- Applikationsspezifische Schemata
- Modelle zur generischen Speicherung
- Multidimensionale Speicherung
- Open vs. Closed World Assumption
- Ausblick



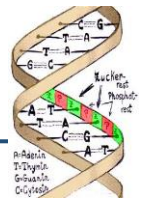
Motivation

- Unterschiedliche Szenarien der Datenverwaltung mit verschiedenen Anforderungen
  - Klinische Studien
  - Genetische Analysen, z.B. Genexpression, Sequenzierung
  - Beschreibungen von biol. Objekten unter Nutzung von Ontologien



Anforderungen

- Korrekte Erfassung und Modellierung von Entitätstypen, deren Attribute und Beziehungen
- Evolution: Hinzufügen/Entfernen von Daten
  - Anpassung/Erweiterbarkeit des Schemas (Schemaevolution)
  - Flexibilität bzgl. Aufnahme neuer Entitäten (bislang fremde Datenarten) und Attribute
  - Auswirkung auf abhängige Programme, Auswertungen
- Schnelle Anfrageverarbeitung
  - Analysis auf großen Datenmengen
- Bezugnahme auf Daten anderer Quellen (Referenzierung)



Wdh. Konzeptionelle Modellierung

- Anforderungsanalyse
  - Identifikation der Anforderungen der Anwendung und Informationsstruktur
  - Was sind relevante Objekte, Attribute, Beziehungen, typische Operationen, …?
- Konzeptionelles Design
  - Transformation der Anforderungsanalyse in einen konzeptionellen Entwurf
  - Erstellung eines Modells unabhängig vom konkreten DBMS (z.B. ER, UML)
  - Eventuelle Inkonsistenzen, notwendige Änderungen →iterative Überarbeitung in dieser Phase sinnvoll, um spätere, aufwendigere Änderungen zu vermeiden
- Logisches Design
  - Abbildung des konzeptionellen Modells auf konkretes Datenbanksystem
  - Definition der Datenstruktur, z.B. Erstellung der Relationen

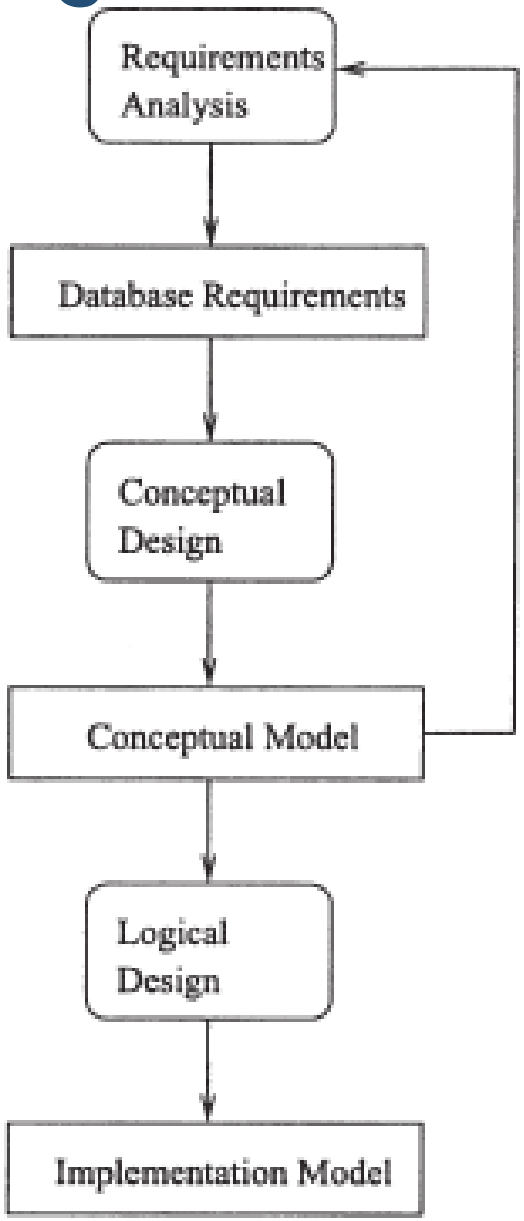
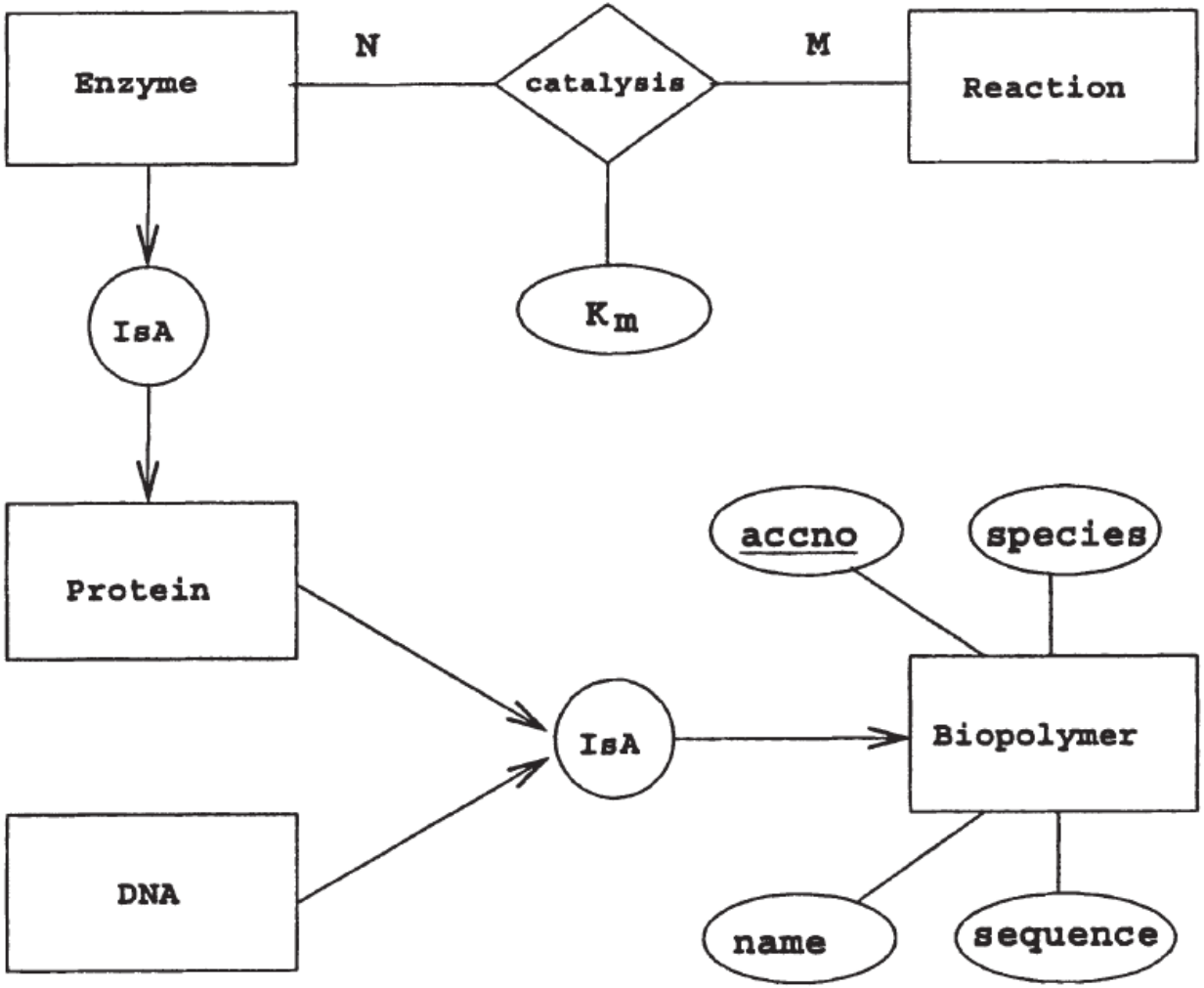


Abb.: Bornberg-Bauer, and Paton: Conceptual data modelling for bioinformatics. Briefings in Bioinformatics, 2002

Modellierung biologischer Konzepte

- Reaktion
- Enzym
- Biopolymer
- Protein
- DNA

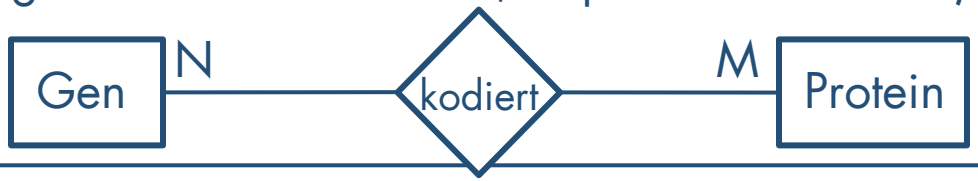


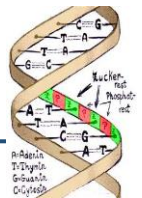
Bornberg-Bauer, and Paton: Conceptual data modelling for bioinformatics. Briefings in bioinformatics, 2002

„Ein-Gen-ein-Enzym-Hypothese“

- 1940er Jahre, George Beadle, Edward Tatum, Nobelpreis
- Probleme durch neuere Erkenntnisse:
  - DNA-Abschnitt kodiert für Proteine allgemein (auch Strukturproteine)
  - Proteine können aus mehreren Polypeptiden bestehen; jeder DNA-Abschnitt, der für ein Polypeptid codiert, ist ein "Gen"

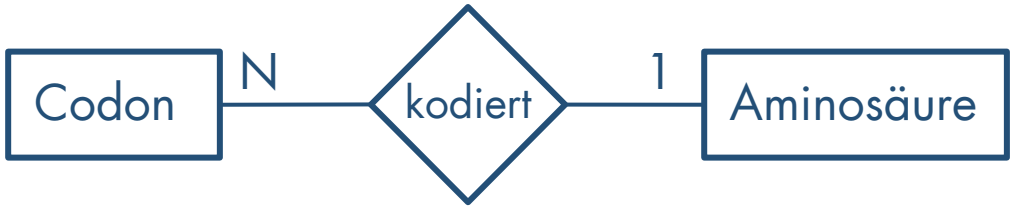
  → Modifikation: „Ein-Gen-ein-Polypeptid “
  - Alternatives Spleißen (splicing): derselbe DNA-Abschnitt kann zu unterschiedlichen mRNA-Molekülen und damit zu unterschiedlichen Polypeptiden führen
  - "Gene" codieren auch funktionelle RNAs (rRNA, tRNA, ..)
  - Eine Möglichkeit: ein Gen kodiert eine biologisch aktive RNA, die in ein Polypeptid translatiert werden kann.
  - Ein Protein/Polypeptid kann aus mehreren Genen entstehen (aufgrund genetischer Redundanz, Kopienzahlvarianten)



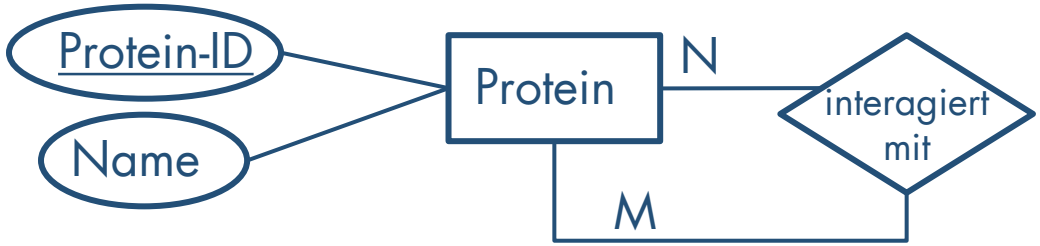


Beispiele

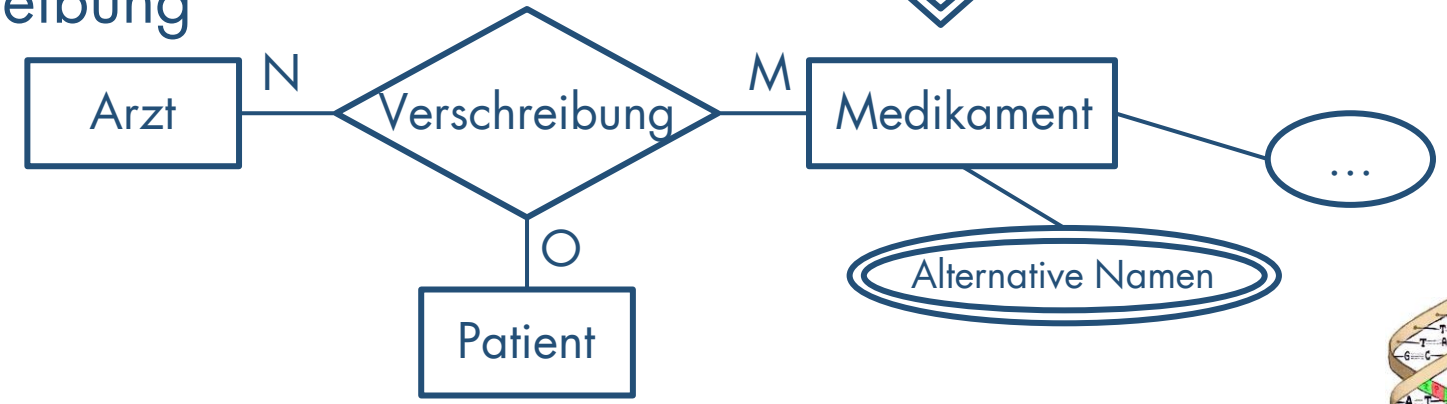
- Codons, Aminosäuren



- Protein-Interaktion



- Gen, Promotor
- Verschreibung

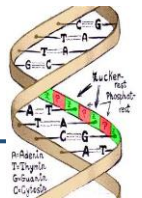


BioDB Design

Artikel: Nelson et al.: Designing databases to store biological information. *Biosilico*, 2003.

1) Lernen aus bestehender Literatur und Quellen
   - DB-Buch
   - Wiederverwenden von bestehenden Datenmodellen (z.B. Literaturverwaltung, Kontaktinformationen)
   - Bessere Interoperabilität durch Verwenden strukturierter Vokabulare (Gene Ontology, Enzyme Codes, …)
   - Veröffentlichen des eigenen Schemas

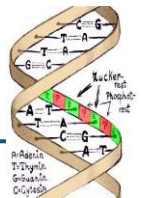
2) Zeit für DB-Design nehmen
   - Keine ungeplante DB-Erstellung



BioDB Design

3) Berücksichtigen der individuellen Bedürfnisse der Biologen

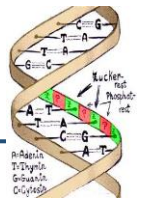
- Unterschied zu anderen Domänen:
  Nicht möglich alle „Business rules“ zu kennen, um constraints zu erstellen (Bsp. ‘Bestellung darf nicht ohne Lieferadresse abgespeichert werden.’)
- Regeln oft nicht bekannt, verschiedene Experten interpretieren Fakten unterschiedlich
  → BioDB muss „Unsicherheiten“ erlauben
  - Unstrukturierter Text in „Comments“-Feld
    (unterstützt Experte, aber keine queries)
  - Kompromiss: Strukturierte Kategorisierung erzwingen und zusätzlich Kommentarfeld für Erklärungen
- Daten unterschiedlich glaubwürdig:
  „quality scores“ zur Bewertung der Daten einplanen



BioDB Design

4) Nutzen der DB um Datenintegrität zu erzwingen

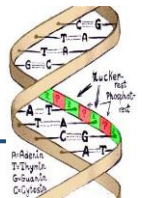
- Normalisierung: Vermeiden von Inkonsistenzen durch Redundanzen etc.
- Denormalisierung: Trigger oder Applikation muss Datenintegrität sicherstellen!
- Häufiges „redesign“ vermeiden→ Software muss auch angepasst werden; möglichst die tatsächliche Struktur der Daten reflektieren, um auch zukünftige Anforderungen zu erfüllen
- Normalisierung → Performance Probleme
  Bio-Daten werden häufiger gelesen als geschrieben
  → Nutzen von Indices, materialisierte Sichten …



BioDB Design

5) DB Umfang sollte handhabbar sein

- DB-Design kann Informationen ignorieren oder vereinfachen (sollte aber vollständig in der betrachteten biologischen Untermenge der DB sein)
- Iteratives Design, bei großen Projekten, erst Teilprojekte lösen, Feedback von Nutzern
- Umgang mit Flatfiles bei „redesign“
  - Parsen und in DB integrieren, falls Daten inkonsistent; Flatfile von User erwünscht → Export aus DB
  - (Teile in) Flatfile erhalten; falls Flatfiles konsistent (Überführung in DB eventuell zu hoher Aufwand)
- Bilder in DB
  - Schnell sehr große Datenmenge
  - Oft nur Referenz auf extern gespeichertes Bild



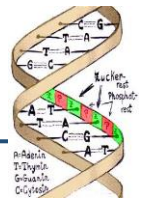
BioDB Design

6) Kommunikation mit Nutzern & *UserInterface* Programmierer

- DB nur erfolgreich, wenn sie genutzt wird
- Gutes Interface ist sehr wichtig;
  Kollaboration zwischen DB Designer, Programmierer & Nutzer
- *"User feedback should not be dismissed because 'they do not understand databases'."*
  Endnutzer sind die einzigen, die die Daten und wie sie genutzt werden, wirklich verstehen!

7) Testen des DB Designs mit Real-World-Daten

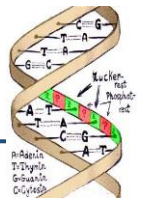
- Verwendung repräsentativer Testdaten
- Einige Beziehungen in Daten sind eventuell (testen!) nicht wichtig für das DB Ziel
  - Beispiel: in Proteinfunktions-DB ist ein Link vom Protein auf ein Gen ausreichend, in DB für Splicing Varianten detaillierteres Schema notwendig



BioDB Design

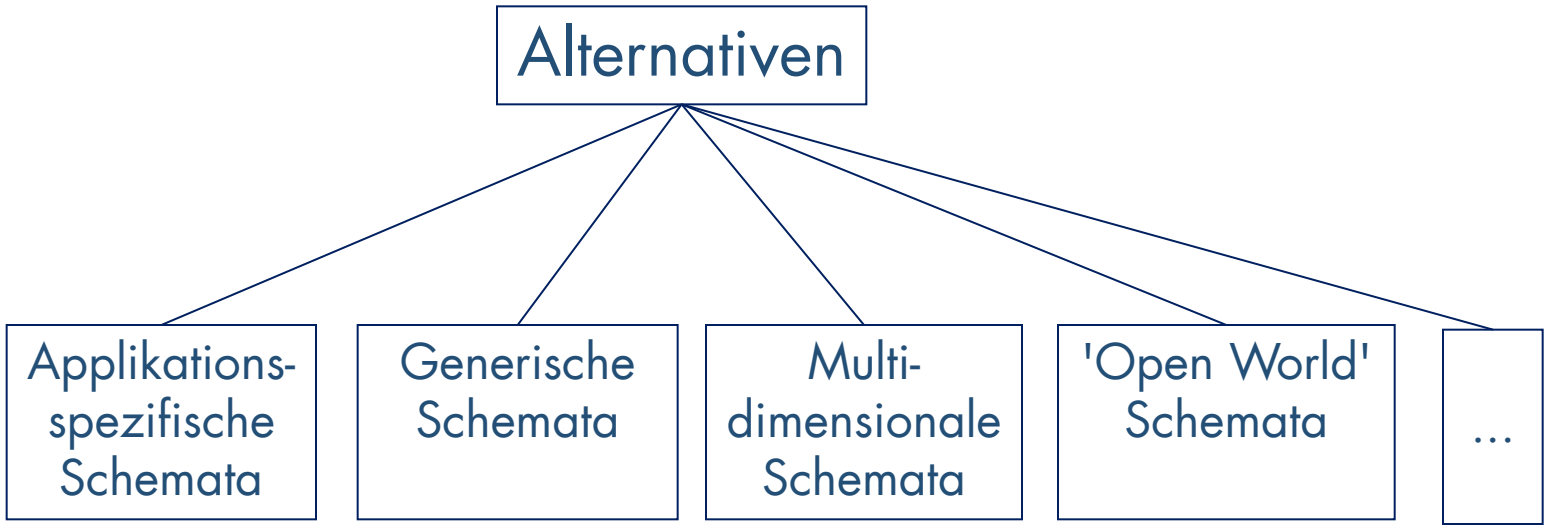
8) Verständliche, leicht zu verwaltende DB Struktur

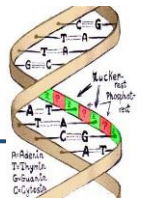
- Dokumentation ist wichtig: Übliche Fehler sollten in neuen DB-Projekten nicht wiederholt werden
- Doppeldeutigkeiten erklären (RELATED_PROTEIN: Sequenzähnlichkeit, Strukturähnlichkeit, Funktionale Ähnlichkeit, interagierendes Protein…?)
- Verwenden von „Naming standards“, kontrollierten Vokabularen



Schema-Alternativen

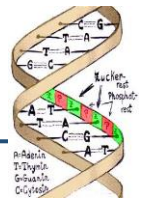
- Schema wird im Prozess des Datenbankentwurfs erstellt
- Kombination verschiedener Alternativen möglich





Applikationsspezifische Schemata

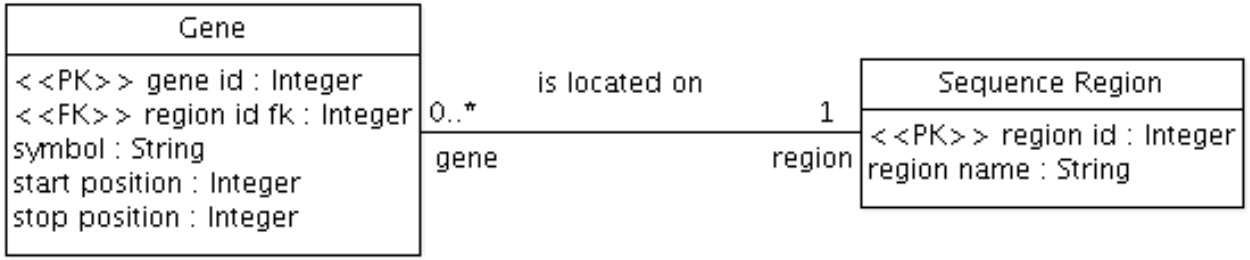
- Erstellung des Schemas ausgerichtet auf eine Applikation
- Abbildung von Realwelt-Sachverhalten in
  - Entitäten, die mit einer Menge von Attributen beschrieben werden
  - Beziehungen zwischen Entitäten
- Relationale Schemata: 2./3. Normalform
- Restriktionen durch DB-Constraints (not null, …)
- Vorteile
  - Genau, zugeschnittene Modellierung; auf Applikation optimiert (bspw. individuelle Datentypen, normalisiertes Schema)
  - Verständlich für Endnutzer, klare Bezeichnungen für Attribute, Entitäten etc.
- Nachteile
  - Ressourcen-aufwendiger Modellierungsprozess
  - Änderungen nicht einfach: wenig Flexibilität, Schemaevolution notwendig



Beispiel

- Abbildung einer Menge von Genen mit ihrer Lokalisation

Schema



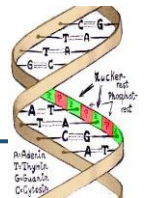
Schema mit Instanzdaten

Gene				
gene id	region id fk	symbol	start position	stop position
1	346	TAPBP	33.375.449	33.390.114
2	341	SFRS11	70.433.953	70.502.757
…	…	…	…	…

Sequence Region	
region id	region name
341	Chrom. 1
346	Chrom. 6
…	…

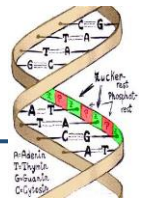
Beispiel-anfrage

$$\pi_{\text{Gene.symbol, Gene.start position, Gene.stop position, Sequence Region.region name}} (\sigma_{\text{Sequence Region.region name = 'Chrom. 1'}} (\text{Gene} \bowtie_{\text{Gene.region id fk = Sequence Region.region id}} \text{SequenceRegion}))$$



Beispiel *„protein motif fingerprints“*

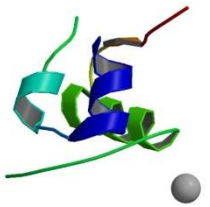
- Motiv
  - Kurzes konserviertes Sequenzmuster, das zu bestimmten Proteinfunktionen assoziiert ist
  - Motive überlappen typischerweise nicht
- *Fingerprint (auch Signatur)*
  - Gruppe konservierter Motive zur Charakterisierung einer Proteinfamilie (nicht zwingend angrenzende Motive)
- Datenbank für Fingerprints PRINTS
  - Typisches Bsp. einer BioDB:
    zunächst Nutzung von ASCII Dateien, dann Migration auf relationale Plattform (aufgrund der Performanz und Funktionalität)
  - > 2100 fingerprints, >12000 single motifs



Insulin signature : PR00277

PR00277 Insulin signature
PR00276 Insulin family signature
PR02003 Bombyxin signature
PR02004 Relaxin signature
PR02033 Placentin signature
PR02002 Insulin-like arowth factor sian.
... ...

- 3-elementiger Fingerprint (Signatur für Insuline)
- Bestimmung anhand eines Alignments (zwischen 28 Sequenzen)
- Bestimmung der Motive in konservierten Regionen (signifikante Abweichung von anderen engen Verwandten der Insulinfamilie)



INSULIN: **Insulin signature**

Seed alignment containing 28 sequences:

Fingerprint ‚INSULIN signature' (PR00277)

Motif accession: INSULIN1

- Length of motif = 12
- Motif number = 1

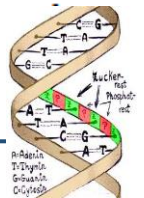
```
FINAL MOTIF SETS

  INSULIN1          Length of motif = 12   Motif number = 1
  Insulin motif I - 3
                                  PCODE            ST    INT
  FVNQHLCGSHLV                  A0ELZ0_MERUN      25    25
  FVNQHLCGSHLV                  Q7M0G1_CRISP       1     1
  FVNQHLCGSHLV                     INS_SPETR      25    25
  FVNQHLCGSHLV                     INS_SHEEP      25    25
  FVNQHLCGSHLV                     INS_RABIT      25    25
  FVNQHLCGSHLV                     INS_PSAOB      25    25
  FVNQHLCGSHLV                     INS_PONPY      25    25
  FVNQHLCGSHLV                       INS_PIG      25    25
  FVNQHLCGSHLV                     INS_PHYCA       1     1
  FVNQHLCGSHLV                     INS_PANTR      25    25
  FVNQHLCGSHLV                     INS_MACFA      25    25
  FVNQHLCGSHLV                     INS_HUMAN      25    25
  FVNQHLCGSHLV                     INS_HORSE       1     1
```

Motif accession: INSULIN2

- Length of motif = 7
- Motif number = 2

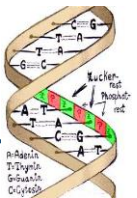
```
  INSULIN2          Length of motif = 7    Motif number = 2
  Insulin motif II - 3
                                  PCODE            ST     INT
  EALYLVC                       A0ELZ0_MERUN      37     0
  EALYLVC                       Q7M0G1_CRISP      13     0
  EALYLVC                          INS_SPETR      37     0
  EALYLVC                          INS_SHEEP      37     0
  EALYLVC                          INS_RABIT      37     0
  EALYLVC                          INS_PSAOB      37     0
  EALYLVC                          INS_PONPY      37     0
  EALYLVC                            INS_PIG      37     0
  EALYLVC                          INS_PHYCA      13     0
  EALYLVC                          INS_PANTR      37     0
  EALYLVC                          INS_MACFA      37     0
  EALYLVC                          INS_HUMAN      37     0
```



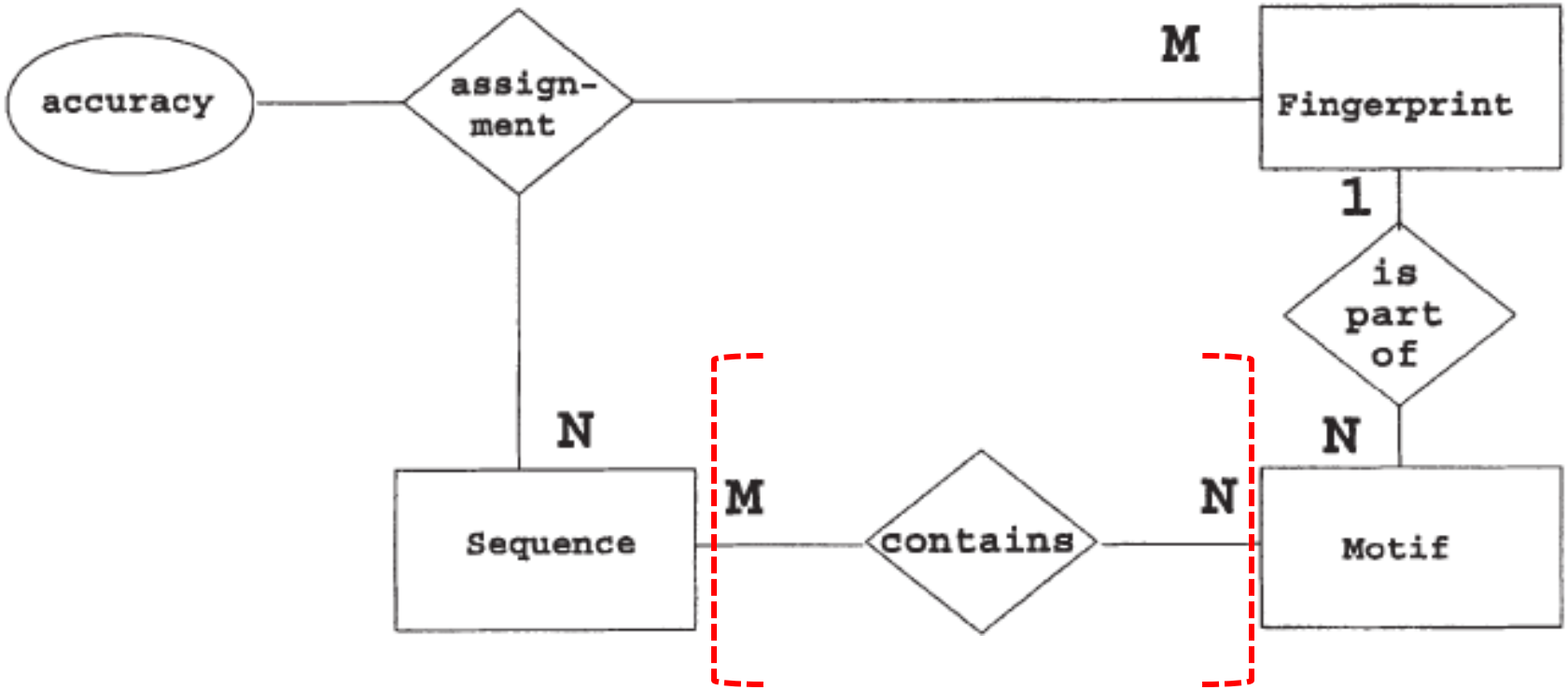
Modellierung in PRINTS

- Jede **Sequenz** enthält mehrere **Motive**, von denen jedes in einer oder mehreren Sequenzen auftaucht
- Jeder **Fingerprint** ist eine Sammlung von Motiven, aber jedes Motiv erscheint in genau einem Fingerprint
- Beziehung zwischen Fingerprint und bestimmter Sequenz repräsentiert die funktionale Charakterisierung eines Proteins
  - Genauigkeit: Abhängig von der Anzahl der Motive in einem Fingerprint, die zu einer bestimmten Sequenz matchen, ist diese Zuordnung mehr oder weniger glaubwürdig (z.B. true positive, true partial, false partial)
- Entitäten haben jeweils einige Attribute

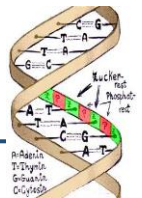
Bornberg-Bauer, and Paton: Conceptual data modelling for bioinformatics. Briefings in bioinformatics, 2002



Modellierung PRINTS



Bornberg-Bauer, and Paton: Conceptual data modelling for bioinformatics. Briefings in bioinformatics, 2002



Relationales Modell – Beispiel PRINTS

Motif

accession	fingerprint accession (FK)	motif number	motif sequence	…
INSULIN1	PR00277	1	FVNQHLCGSHLV	
INSULIN2	PR00277	2	EALYLVC	
INSULINFAMLY1	PR00276	1	HLCGSHLVEALYLVCGE	

Fingerprint

accession	title	creation date	…
PR00277	Insulin signature	14-APR-1994	…
PR00276	Insulin family signature	14-APR-1994	

Sequence

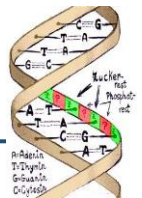
accession	identifier
P01308	INS_HUMAN
P01325	INS1_MOUSE

Fingerprint_Sequence_Assignment

sequence accession (FK)	fingerprint accession (FK)	accuracy
P01308	PR00277	TP
P01325	PR00277	TP
P01308	PR00276	TP

Generische Schemata

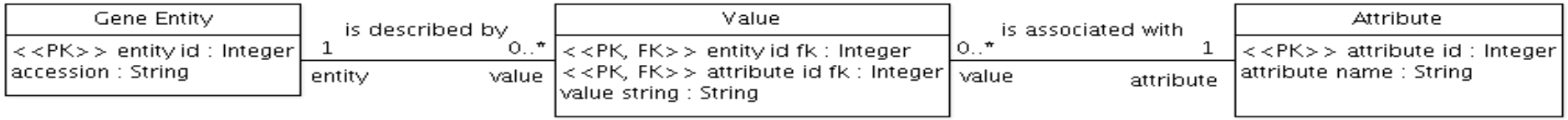
- **Generisch**: Schema ist unabhängig von der Applikation und damit für viele Anwendungen (wieder-) verwendbar
- Ausgewählte Ansätze
  - Entity Attribute Value (EAV) – relationale Schemata (Varianten)
  - Name Type Value (NTV) – vorwiegend im Bereich XML
  - Resource Description Framework (RDF)
- Vorteile: Einfachheit, Flexibilität, robust gegenüber Änderungen
- Nachteile: Semantik wird in Instanzdaten kodiert, aufwändige Anfragegenerierung und –verarbeitung, steigende Anzahl von Tupeln ..



Beispiel EAV

- Abbildung einer Menge von Genen und ihrer Lokalisation unter Nutzung des EAV-Ansatzes

Schema



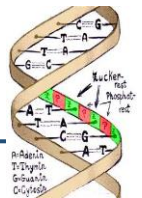
Schema mit Instanzdaten

Gene Entity	
entity id	accession
1	ENS0815
…	…

Value		
entity id fk	attribute id fk	value
1	102	TAPBP
1	103	33.375.449
…	…	…

Attribute	
attribute id	attribute name
102	symbol
103	start position
…	…

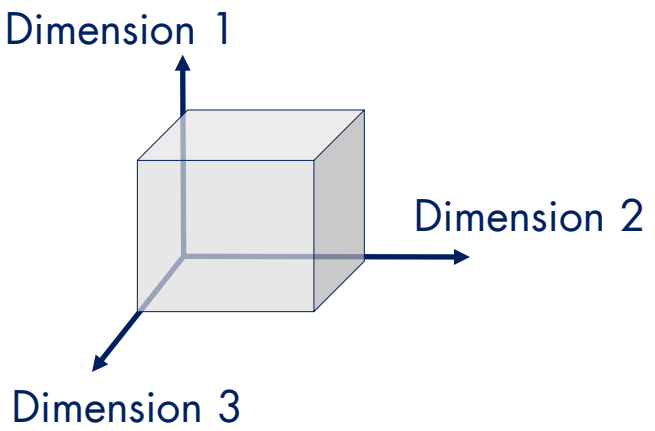
- Vorteil: Flexibilität, falls neue Attribute hinzugefügt oder bestehende Attribute entfernt werden sollen
- Nachteil: Datentyp des Attributwerts ist festgelegt (z.B. String)

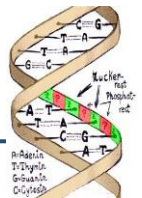


Multidimensionale Schemata

- Modellierungsparadigma für Data Warehouses und Data Marts
- **Dimensionen**: Beschreibungen von Einflussgrößen
- **Fakten**: numerische Daten, die von Dimensionen beeinflusst werden

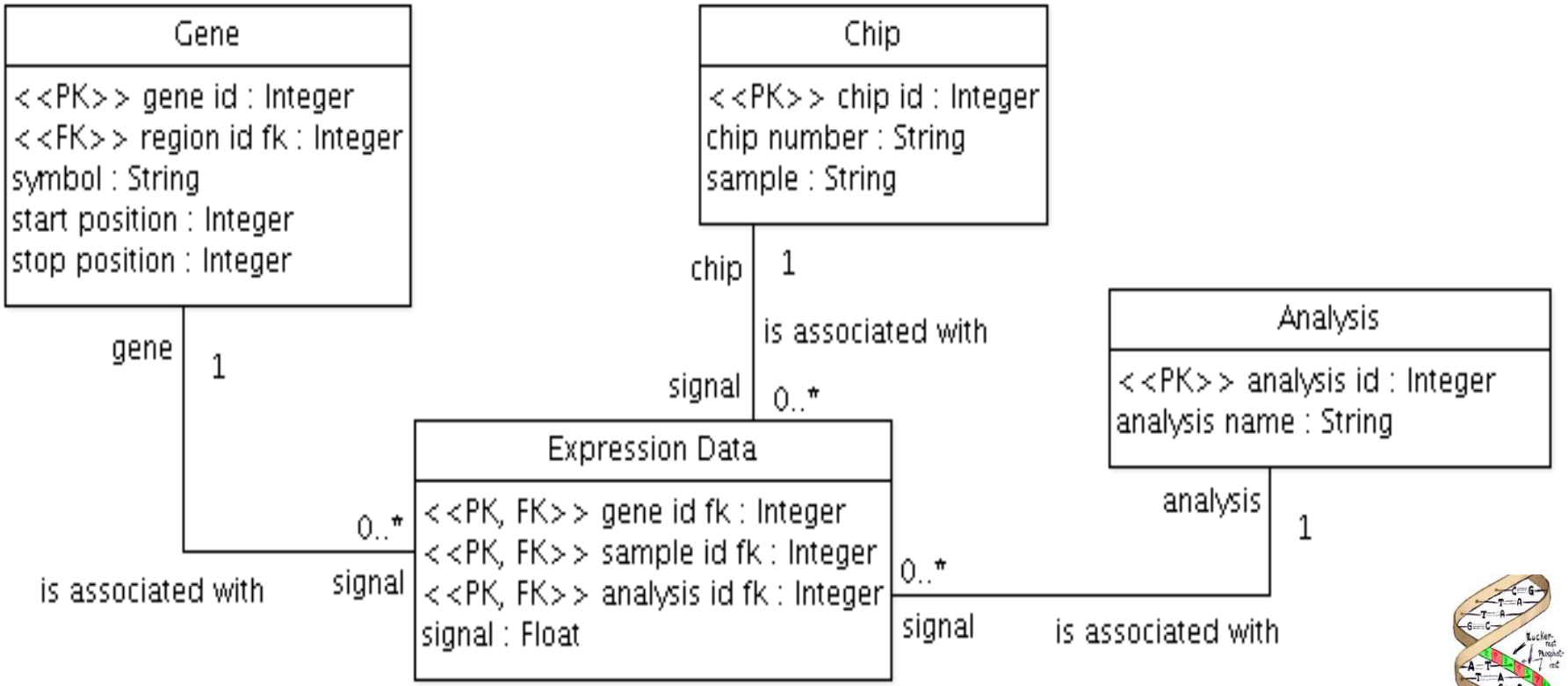
Beispiel: Data Cube mit drei Dimensionen





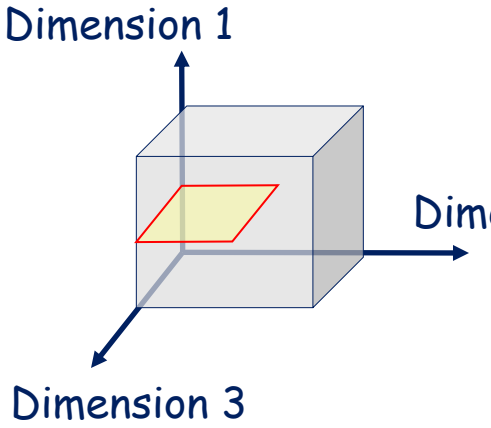
Beispiel

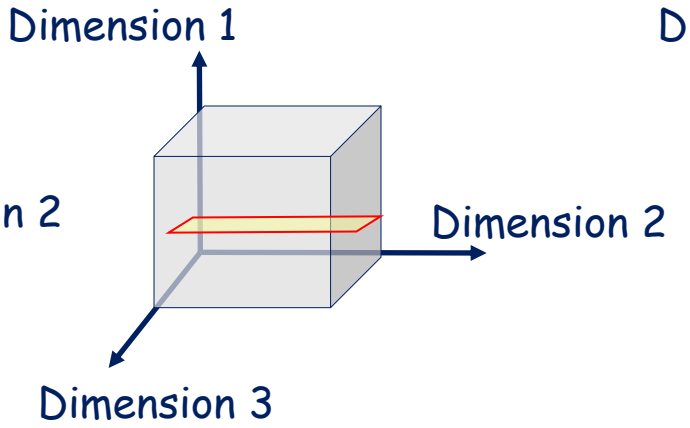
- Expressionssignale pro Gen und Chip, die in einem Expressionsexperiment gemessen wurden
- Dimensionen: Gene, Chip, Analysis
- Fakten: Signal (Messwert)

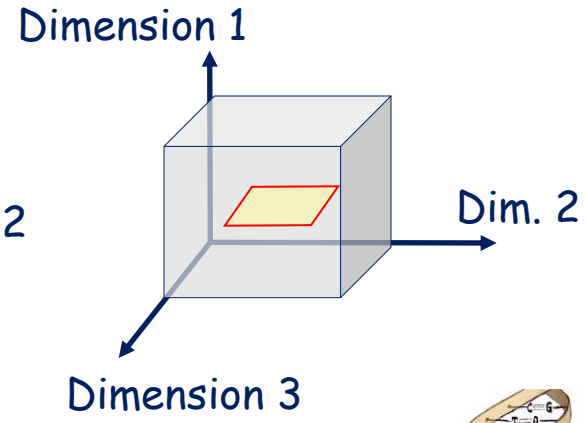


Multidimensionale Schemata

- Erweiterbarkeit und Skalierbarkeit
  - Hinzufügen neuer Instanzdaten (Dimensionsdaten + Fakten) ohne Änderungen am zugrundeliegenden Datenmodell
- Schnelle Anfrageverarbeitung: Verzicht auf Normalisierung der Dimensionstabellen spart Joins, …
- Multidimensionale Selektion und Analyse
  - Einfache Selektion, Aggregation und Vergleich der Fakten

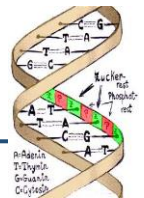






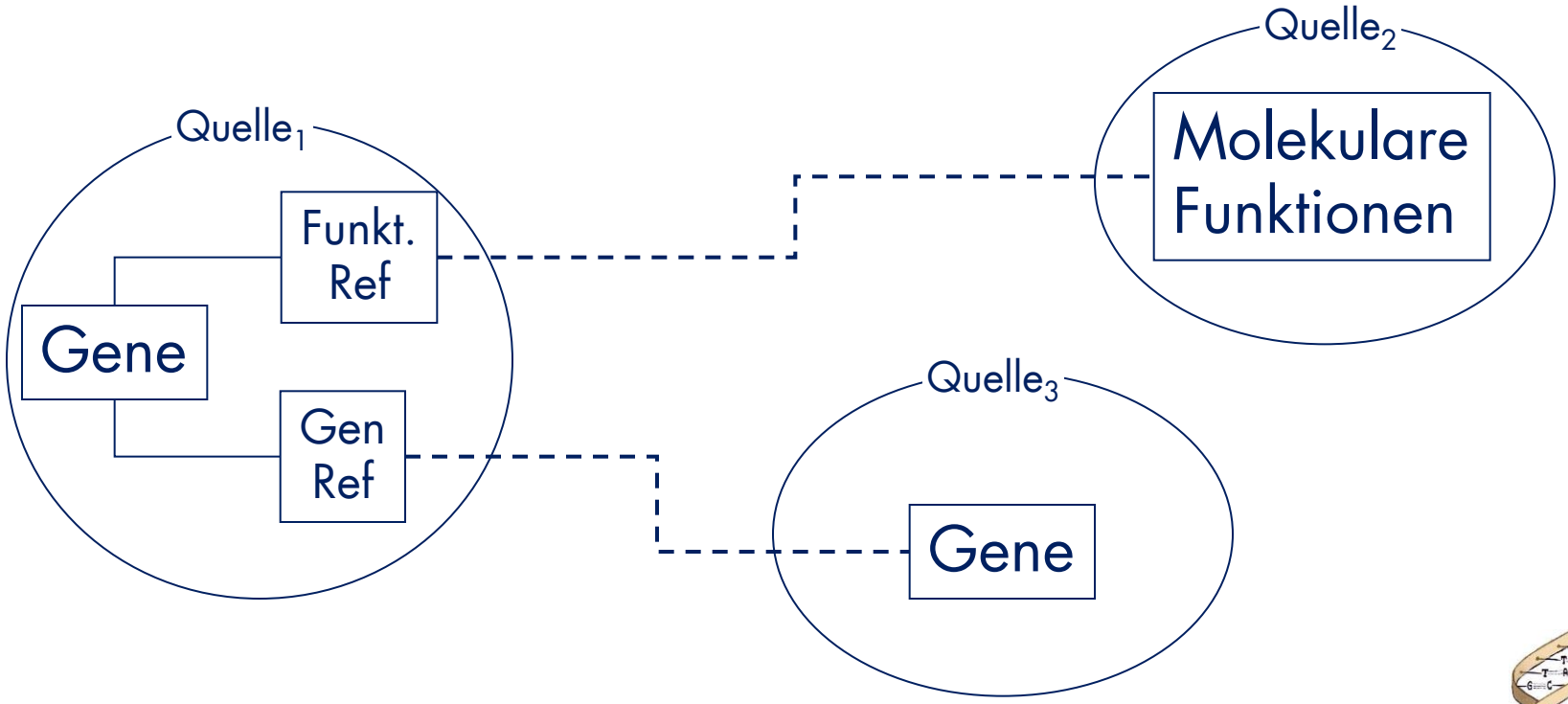
'Open World' Schemata

- '**Closed World Assumption**': Alle Datenobjekte sind innerhalb des Schemas bekannt und beschrieben
- '**Open World Assumption**': schemaübergreifende Beschreibung von Objekten unter Nutzung von spezifischen Objekt-Referenzen (external references, cross references, XREF …)
- **Referenz**: Verbindung zwischen zwei Objekten aus zwei getrennten Datenquellen
- Vorteile: Rückgriff auf bestehende Objektbeschreibungen in anderen Datenquellen möglich
- Nachteil: Datenintegration notwendig, wenn Verknüpfung der Daten benötigt wird



Referenzen

- Beispiel: Referenzen zwischen zwei Mengen von Genen zweier Datenquellen und der ontologischen Beschreibung ihrer molekularen Funktion
- Zuordnung über Mapping-Tabellen

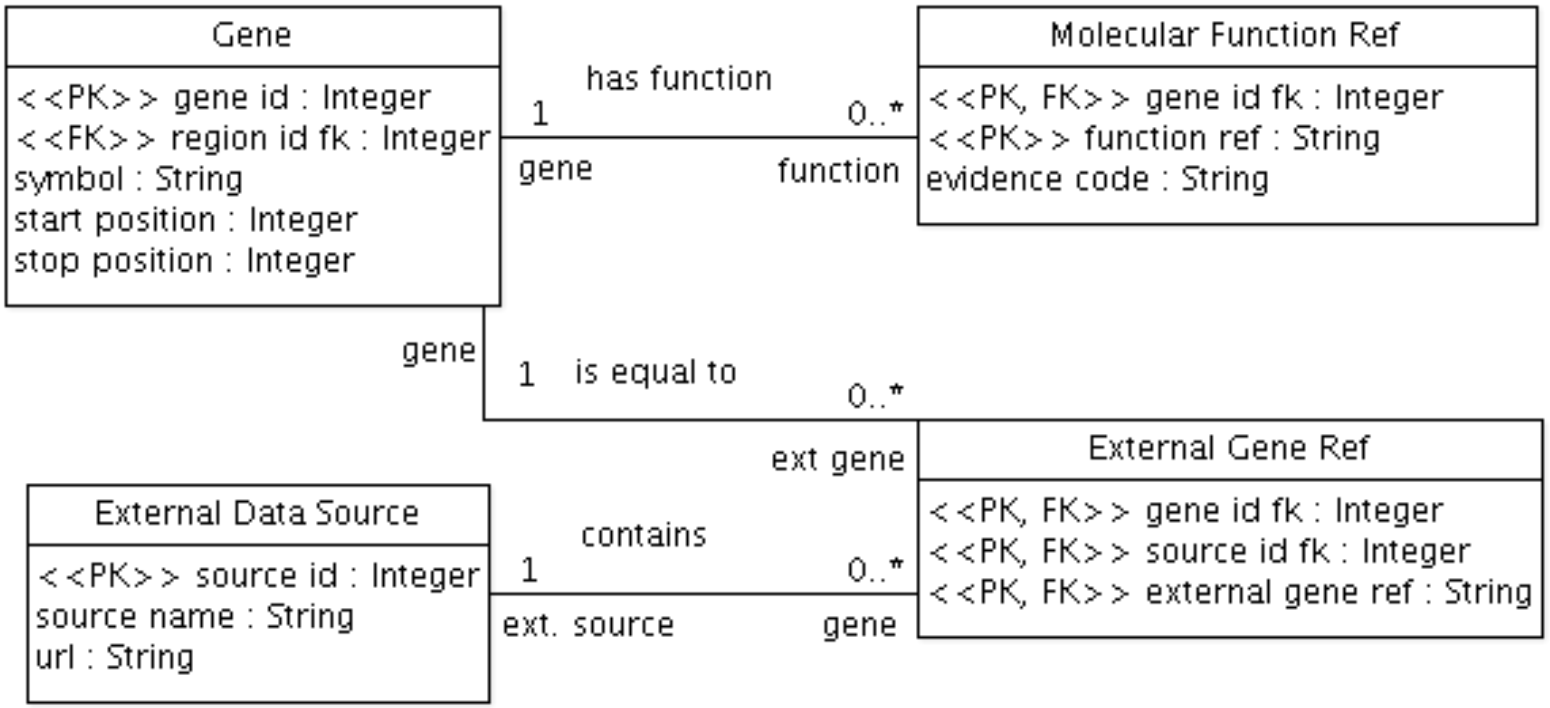


Beispiel

- Abbildung einer Menge von Genen mit ihrer Lokalisation und Beschreibung molekularer Funktionen

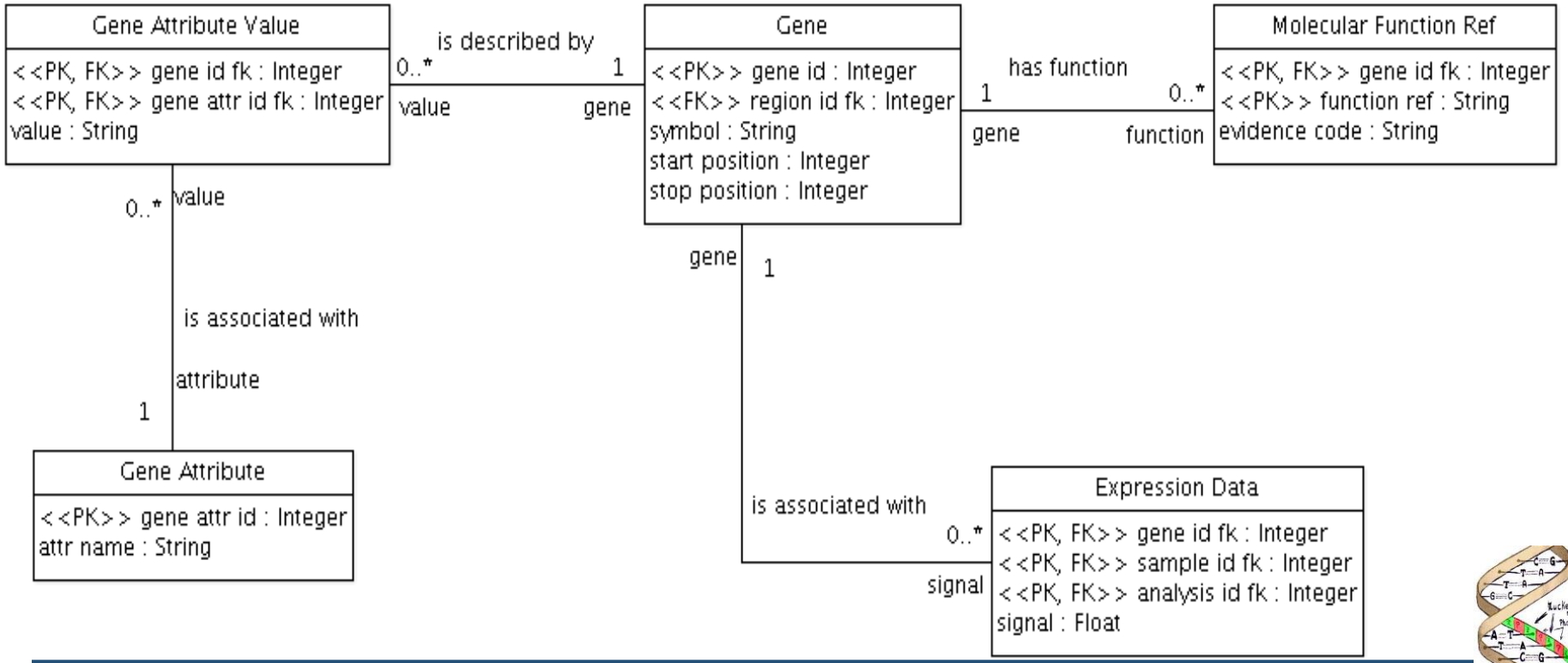
Schema (Auszug für Quelle$_1$)

Objekt-Referenz;
Objekt ist in einer anderen Datenquelle beschrieben



Kombination verschiedener Alternativen

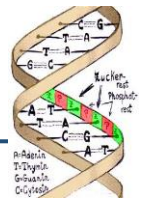
- Hybrid (Genattribute teilweise appl.-spez. + generisch)
- Referenzen zu anderen Datenquellen (Mapping-Tabelle)
- Performanzsteigerung durch mgl. materialisierte Sichten über Genattribute



Datenmanagement in der Cloud

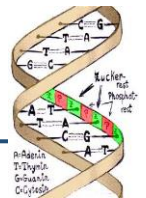
- Cloud: Nutzung verfügbarer Ressourcen von (externen) Anbietern: Compute-, Storage-Ressourcen
  - Übergang von monolithischen zu dynamischen Informationssystemstrukturen
  - Nutzung "on-demand"
  - Abrechnung nach Ver-/Gebrauch
- Zugriff über geeignete Schnittstellen: REST, WS-API
- Beispiele für Cloud-Lösungen
  - Amazon Simple DB, Elastic Compute Cloud (EC2)
  - Google App Engine incl. BigTable
  - Yahoo Cloud
  - ...

→ Nutzung von NoSQL Datenbanken



NoSQL-Datenbanken

- Not only SQL: schemafrei / schwache Restriktionen
- Key-Value Stores: z.B. Amazon Dynamo, …
  - Speicherung eines Werts (z.B. BLOB) pro nutzer-definiertem Schlüssel
  - Zugriff über Schlüssel, d.h. put (key, value) und get (key) –Methode
- Document Stores: z.B. SimpleDB, MongoDB, …
  - Speicherung semistrukturierter Daten als Dokument (z.B. JSON)
  - Zugriff über Schlüssel oder einfache Anfragesprache
- Wide-column stores: z.B. BigTable / Hbase, ..
  - Tabellen-basierte Speicherung mit flexibler Erweiterung um neue Attribute
  - Zugriff über Schlüssel oder SQL-ähnliche Anfragesprache
- Graphdatenbanken: z.B. Neo4J
- …



Beispiel - simpleDB

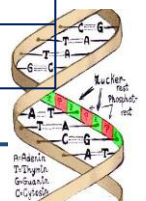
- Abbildung einer Menge von Genen und ihre
- Begrenzte Menge von Operatoren (Amazon SimpleDB*)
  - CreateDomain, ListDomains, DeleteDomain
  - PutAttributes, GetAttributes, DeleteAttributes
  - Select Query

PUT(domain,MeineGene)(item,Ens0815),(region,chrom. 1),(function,transport)
PUT(domain,MeineGene)(item,Ens0816),(region,chrom. 1),(start,200),(stop,250)

Domäne: MeineGene

item	region	start	stop	function
Ens0815	chrom. 1			transport
Ens0816	chrom. 1	200	250	
…	…	…	…	…

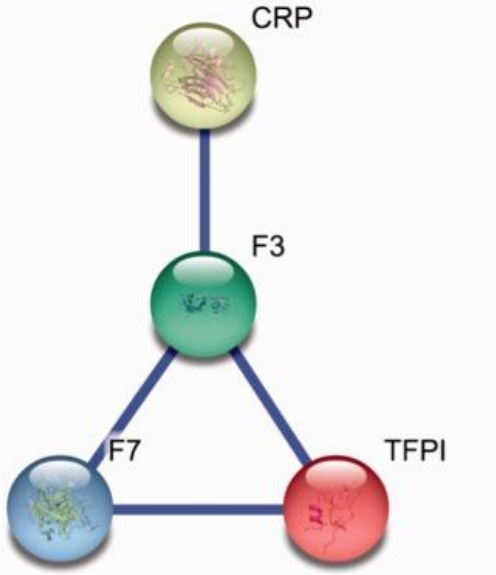
* http://aws.amazon.com/simpledb



Beispiel – Neo4J

interactions:

p1	p2	confidence
CRP	F3	0.947
F3	F7	0.999
F3	TFPI	0.993
F7	TFPI	0.977

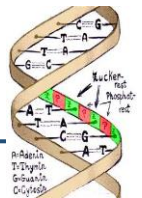


SQL:
```
SELECT i2.p2 FROM interactions AS i1
INNER JOIN interactions AS i2 ON i1.p2=i2.p1
WHERE i1.p1='CRP'
```

Cypher:
```
START p1=node:names(name: 'CRP')
MATCH p1-->()-->p2
RETURN p2.name
```

Relational versus graph database representation of a small protein interaction network. In the relational database, the network is stored as an interactions table (left). By contrast a graph database directly stores interactions as pointers between protein nodes (right). Below, we show the queries to identify second-order interaction partners in SQL and Cypher, respectively

Quelle: Christian Theil Have, Lars Juhl Jensen: Are graph databases ready for bioinformatics? *Bioinformatics* 29(24): 3107–3108, 2013.



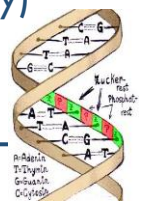
Query Benchmark: Relationale vs. Graph-Datenbank

	Neighbor network	Best-scoring path	Shortest path
			**
PostgreSQL	206.31 s	1147.74 s	976.22 s
Neo4j	5.68 s[a] *	1.17 s	0.40 s
Speedup	36×	981×	2441×

- Gut für spezifische (einfache) Graphtraversierungsprobleme
- Aber wichtige Algorithmen wie Dijkstra (shortest path mit Kantengewichten = best scoring path) fehlen noch
- Intuitive Formulierung von Queries
- Mengenoperationen → Relationale Datenbanken

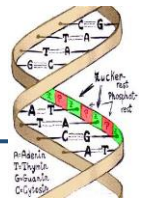
* Speedup 49 x durch Query Optimierung (mehrere einfache statt einer komplexen Query)

** Pfadlänge beschränkt



Zusammenfassung

- Verschiedene Möglichkeiten der Datenverwaltung
- Unterschiedliche Modellierungsformen: Schema-Alternativen
  - Applikationsspezifische Schemata
  - Generische Schemata
  - Multidimensionale Schemata
  - 'Open World' Schemata
- Optimale Lösung ist abhängig von gegebenen Anforderungen und Rahmenbedingungen
- Oftmals Kombination der verschiedenen Modellierungsparadigmen



Fragen ?