COSMIC

Protokoll zum Projekt im Modul „Molekularbiologische Datenbanken“

Thomas Dencker[*] Robert Kratel[†] Pascal Schmitt[‡]

1. Mai 2013

Inhaltsverzeichnis

[*] mailto:thomas.dencker@stud.uni-goettingen.de

[†] mailto:robert.kratel@stud.uni-goettingen.de

[‡] mailto:pascal.schmitt1@stud.uni-goettingen.de

1 Aufgabenstellung

Die Aufgabenstellung des Datenbankprojektes war es, eine auf SQLite[1] basierende Datenbank aufzubauen. Den Ausgangspunkt dieser Datenbank stellt eine Liste von „UniProt accession numbers“ dar. Die Datenbank sollte Informationen aus den entsprechenden Einträgen der UniProt[2]-Datenbank enthalten, wie den Protein- und Gennamen. Den Gennamen sollten außerdem alle Synonyme zugeordnet werden, welche in der GeneNames[3]-Datenbank gespeichert sind. Neben diesen allgemeinen Aufgaben sollten kodierende und nichtkodierende Mutationen in Krebserkranken in den Genen der Datenbank identifiziert werden. Diese Informationen sollten der COSMIC[4]-Datenbank entnommen werden. Darüber hinaus sollte das relationale Datenbank-Schema auf Grundlage eines Entity-Relationship-Modells entworfen werden.

Das Projekt kombiniert Informationen aus drei Datenbanken, deren Inhalt im Folgenden kurz beschrieben wird.

UniProt Die UniProt-Datenbank ist eine freizugängliche Datenbank, die Sequenzen und funktionelle Informationen zu Proteinen enthält. Diese werden von der Swiss-Prot- und der TrEMBL-Datenbank bereitgestellt. Die Einträge der Datenbank können u.a. durch die UniProtJAPI abgefragt und gespeichert werden.

GeneNames genenames.org ist eine Datenbank, welche vom HGNC(HUGO Gene Nomenclature Committee) akzeptierte Gennomenklatur enthält. Unter anderem sind das anerkannte Symbol, die vorherigen Symbole und Synonyme zu rund 19000 proteinkodierenden Genen downloadbar.

COSMIC Die COSMIC-Datenbank wird vom Sanger-Institut bereitgestellt und enthält Informationen zu somatischen kodierenden und nicht-kodierenden Mutationen in Krebserkrankungen.

[1] `http://sqlite.org/`
[2] `http://uniprot.org/`
[3] `http://genenames.org/`
[4] `http://cancer.sanger.ac.uk/cancergenome/projects/studies/`

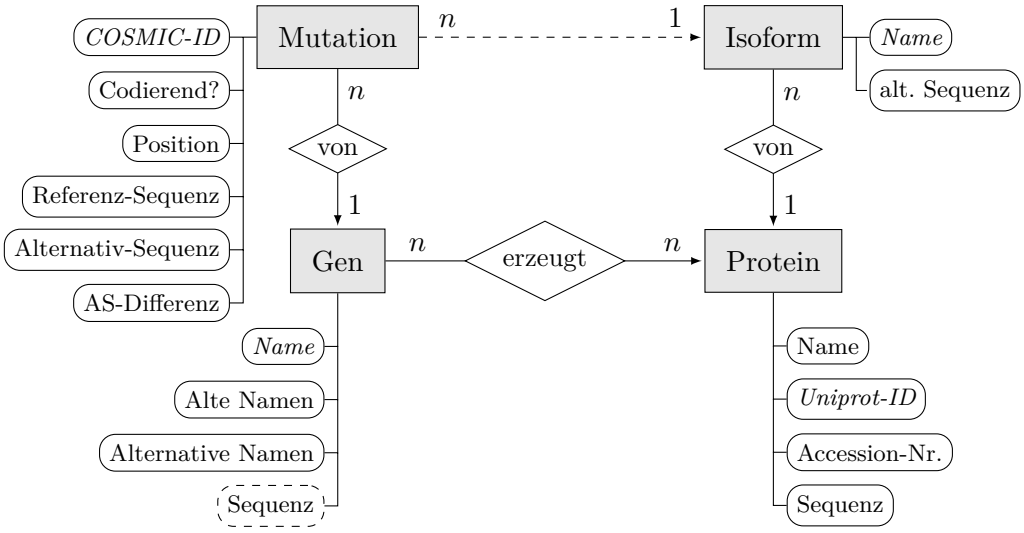

Abbildung 1: Entity-Relationship-Modell

2 Entity-Relationship-Modell

Ausgangspunkt der Datenbank sind die durch die „UniProt accession numbers“ referenzierten Proteine. Als Attribute wurden der in der Aufgabenstellung geforderte Proteinname, die UniProtID und die Accession Number gewählt, über die weitere Informationen abgerufen werden könnten. Außerdem gibt es die Aminosäuresequenz als weiteres Attribut, um diese ggf. mit den Sequenzen der Isoformen vergleichen zu können.

Proteine sind Produkte eines oder mehrerer Gene ($n : n$), da hier keine Gendatenbank verwendet wird, speichern wir die Basensequenz der Gene und ihre Position im Genom nicht. Die Genenames-Datenbank enthält Informationen über den offiziellen, alternative und alte Namen für Gene, wobei COSMIC und UniProt nicht durchgehend den aktuellen offiziellen Namen verwenden. Gene können mehrere Mutationen besitzen ($1 : n$), wobei die COSMIC-Datenbank speichert welche Basenpaare der Referenz-Sequenz eines Gens durch alternative Basenpaare ersetzt werden und ob dies Auswirkungen auf das erzeugte Protein hat. Falls ja wird diese Mutation „kodierend“ genannt und COSMIC speichert zusätzlich die resultierende Änderung an der Aminosäuresequenz des Proteins, was eine Isoform des Proteins erzeugt ($1 : n$). UniProt und COSMIC speichern allerdings nicht, welche Mutation zu welcher Isoform gehört ($n : 1$), daher ist dies nicht Teil unseres Modells.

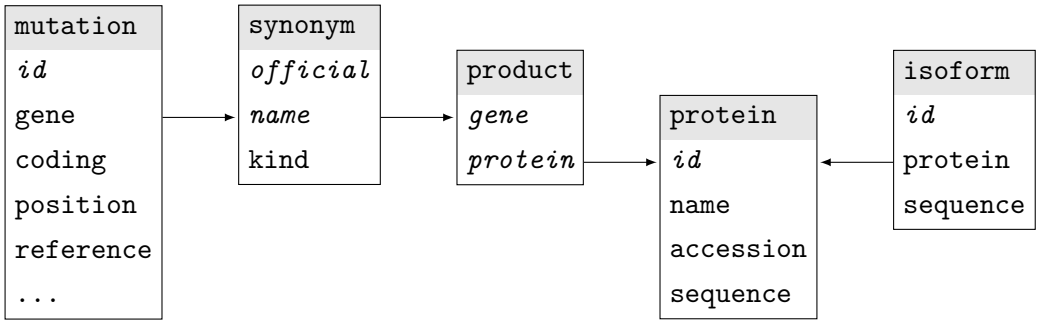

Abbildung 2: Datenbanktabellen und Referenzen

3 Implementierung

Das im Rahmen dieses Projekts erstellte Programm liest Daten aus den Quelldatenbanken ein, speichert sie in einer eigenen SQLite-basierten Datenbank und führt darauf Abfragen aus.

Zum Einlesen und Speichern der Daten wurde als Programmiersprache Java gewählt, da es für die UniProt-Datenbank eine Java-Schnittstelle gibt und die Daten einfach über JDBC in die SQLlite-Datenbank eingespeichert werden können. Zum Ausführen der Abfragen wurde eine einfache GUI erstellt, die eine SQL-Anfrage entgegennimmt und nach Ausführen das Ergebnis bzw. die Fehlermeldung darstellt.

3.1 Relationales Datenbank-Schema

Das Entity-Relationship-Modell aus Abb. 1 wurde in ein relationales Datenbankschema (siehe Abb. 2) überführt, dabei ist zu beachten dass hier keine explizite `gene`-Tabelle existiert sondern die beiden $n:n$ und $n:n$-Relationen `synonym` und `product` direkt gejoined werden. Dabei ist zu beachten dass `mutation.gene` und `product.gene` keine Foreign Keys auf `synonym.name` sind, da beide verschiedene Alternativen zum offiziellen Namen verwenden könnten.

```
CREATE TABLE synonym (official, name, kind,
                      PRIMARY KEY (official, name));
CREATE TABLE mutation (id PRIMARY KEY, gene, coding, position,
                       reference, alternative, strand, aa, count);
```

Die UniProt-Referenzen auf Protein-IDs sind allerdings eindeutig und wurden als Foreign Keys definiert. Im Gegensatz zu anderen Datenbanken unterstützt SQLite keine Datentyp-Annotationen für Spalten und speichert alles als Zeichenketten.

```
CREATE TABLE protein (id PRIMARY KEY, name, accession, sequence);
CREATE TABLE product (gene, protein REFERENCES protein(id),
                      PRIMARY KEY(gene, protein));
CREATE TABLE isoform (id PRIMARY KEY, protein REFERENCES protein(id),
                      sequence);
```

Die Synonymtabelle speichert den offiziellen und alternative Namen. Allerdings verwenden weder UniProt noch COSMIC für sämtliche Gene die offiziellen Namen, Joins müssen daher symmetrisch erfolgen. Der Einfachheit halber wird diese symmetrische Relation als View definiert.

```
CREATE VIEW genename AS
SELECT DISTINCT s1.name AS a, s2.name AS b, s2.official
FROM synonym s1, synonym s2 WHERE s1.official = s2.official;
```

Um Abfragen weiter zuvereinfachen, wird der Join von Mutationen über Synonyme und Genprodukten zu Proteinen in eine weitere View zusammengefasst. Um in Abfragen weitere Joins zu vermeiden und die Abfragen zu beschleunigen wird eine temporäre Tabelle als Zwischenspeicher verwendet.

```
CREATE TEMP TABLE proteinmut AS
SELECT m.id AS mid, m.*, p.id AS pid, p.*, n.official AS gene
FROM mutation m, genename n, product g, protein p
WHERE m.gene = n.a AND n.b = g.gene AND g.protein = p.id;
```

3.2 Einlesen der Daten

UniProt Bevor die UniProt-Einträge abgefragt werden konnten, musste zunächst die `uniprot.acc`-Datei geparst werden. Dabei musste berücksichtigt werden, dass die Liste Duplikate enthielt. Diese wurden aussortiert, um unnötige Anfragen an die UniProt-Datenbank zu sparen.

Darüber hinaus musste die Liste geteilt werden, da der durch die UniProtJAPI vorgestellte `UniProtQueryBuilder` nur Listen mit maximal 1024 Einträgen unterstützt. Sind die entsprechenden Daten nicht bereits in der Datenbank enthalten, werden die UniProt-Einträge von der Datenbank abgefragt und der empfohlene Name in der Proteinbeschreibung, die UniProtID, die primäre „accession number“, die Gennamen und die Isoformen (beschrieben durch „Alternative Products“ in den Kommentaren) durch entsprechende Funktionen der UniProtJAPI ausgelesen. Einige Eintäge, z.B. `Q8WUN3`, `Q6N045` und `A6NF79` existieren nicht mehr und wurden daher ignoriert.

GeneNames Auf der GeneNames-Website kann der Datenbestand in diversen Formaten abgefragt werden, unser Programm lädt die Daten in einem Tab-separierten Format per HTTP herunter, parst sie zeilenweise und schreibt sie in die `synonym`-Tabelle. Die Spalte „Approved Symbol“ enthält dabei den offiziellen Namen, in „Previous Symbols“ und „Synonyms“ stehen, durch Komma getrennt, alternative Namen. Für jede Kombination dieser Namen mit dem offiziellen Namen wird eine Zeile eingefügt und in der `kind`-Spalte entsprechend `previous` oder `alias` vermerkt. Der offizielle Name verweist auch auf sich selbst, dies wird mit `refl` markiert.

Dieser Ansatz ist allerdings mehrdeutig: so führt GeneNames z.B. `ARX` als neuen offiziellen Namen für `PRTS` auf, und auch `ARX` als Synonym zu `UBA2`, aber nicht

umgekehrt. Auf solche Inkonsistenzen wird keine Rücksicht genommen und die Daten so verwendet, als gäbe es einen offiziellen Namen für jedes Gen, in dessen Zeile im GeneNames-Datensatz alle Synonyme aufgeführt werden.

COSMIC Die Listen der kodierenden und nichtkodierenden Genmutationen werden, sofern sie veraltet sind, vom FTP-Server des Sanger-Instituts geladen. Sie liegen mittels GZip gepackt im „Variant Call Format“ (VCF) vor, ein auf dem Tab-separierten Format basierendes Textformat das für die Speicherung von SNPs entworfen wurde. Für jede Mutation ist das Chromosom, die Position, Ursprungs- und Alternativsequenz angegeben. Falls die Mutation in einem Gen liegt, wird in der Info-Spalte auch der Genname genannt, dies ist bei allen kodierenden Mutationen der Fall. Einige nichtkodierende Mutationen liegen ebenfalls innerhalb eines Gens, allerdings z.B. in Introns, wirken sich also nicht auf das Endprodukt aus. Alles wird in die `mutation`-Tabelle geschrieben.

4 Abfragen

Mit der einfachen GUI können SQL-Anfragen ausgeführt werden um komplizierte Fragen zu beantworten wie in Tabellen 1, 2, 4, 3 und 5 aufgeführt.

```
SELECT p.name,
       GROUP_CONCAT(g.gene)
FROM protein p, product g
WHERE g.protein = p.id
GROUP BY p.id
HAVING COUNT(*) > 1
```

Protein	Gene
Basic helix-loop-helix T.F.	DMRTC1,DMRTC1B
Doublesex-& mab-3-rel. T.F.	HSDX1,HSFX2
Heat shock T.F., X-linked	HSFY1,HSFY2
Heat shock T.F., Y-linked	SCXA,SCXB

Tabelle 1: Welche Proteine werden von mehr als einem Gen kodiert?

```
SELECT gene, mid, aa
FROM proteinmut
WHERE pid = "DMRTC_HUMAN"
```

Gen	COSMIC-ID	Prot.-Mutation
DMRTC1	COSM458031	p.R171G
DMRTC1	COSM195610	p.G159V

Tabelle 2: Welche Mutationen beeinflussen das DMRTC-Protein?

```
SELECT DISTINCT gene, accession
FROM proteinmutt
WHERE CAST(position AS INTEGER) > 1700000
AND   CAST(position AS INTEGER) < 2000000
ORDER BY position
```

Gen	Prot.-AccNr.
CRAMP1L	Q96RY5
ONECUT3	O60422
MYT1L	Q9UL68
KLF16	Q9BXK1
IRX4	P78413
WHSC1	O96028
HIC1	Q14526

Tabelle 3: Welche Proteine stammen aus einem best. Abschnitt des Genoms?

```
SELECT m.name,
       i.iso AS isoforms,
       COUNT(*) AS mutations,
       SUM(m.count) AS cases
FROM proteinmut m,
     (SELECT protein,
             COUNT(*) AS iso
      FROM isoform
      GROUP BY protein
      HAVING iso >= 10
      ORDER BY iso DESC) i
WHERE i.protein = m.pid
GROUP BY m.pid
ORDER BY cases DESC
```

Name	#Iso	Stellen	Fälle
Runt-related TF 1	11	223	303
TF 4	11	196	152
TF 7-like 2	12	122	103
Homeobox prot...	11	112	95
Tumor protein 63	12	81	74
Zinc finger prot...	15	58	51
Glucocorticoid receptor	10	43	41
NF of act. T-cells...	24	38	40
Microphthalmia-ass. TF	11	39	36
Tumor prot. p73	10	23	24
Nuclear receptor...	11	19	16
cAMP-resp. el...	23	21	15
TF 7	16	16	12

Tabelle 4: Wie viele Genmutationen wirken auf die Proteine mit den meisten Isoformen?

```
SELECT DISTINCT
     m.gene,     n.official, g.gene
FROM mutation m, genename n, product g
WHERE m.gene = n.a AND n.b = g.gene
AND m.gene != g.gene
LIMIT 4
```

COSMIC	GeneNames	UniProt
MGC33414	ZNF683	ZNF683
CREB3L4	CREB3L4	CREB3
TCF7L2	TCF7L2	TCF4
RNF141	RNF141	ZNF230

Tabelle 5: Beide Datenbanken folgen nicht immer der offiziellen Benennung