Protein Query Language: A Novel Approach

Sherif Elfayoumy and Paul Bathen
School of Computing, University of North Florida, Jacksonville, FL, USA

Abstract - This paper introduces a Protein Query Language (PQL) for querying protein structures in an expressive yet concise manner, utilizing the work of Patel [1] and introducing constructs in principal similar to those in Roldan-Garcia [2]. One of the objectives of the paper is to demonstrate how such a language would be beneficial to protein researchers to obtain in-depth protein data from a relational database without extensive SQL knowledge. The language features options such as limiting query results by key protein characteristics such as methyl donated hydrogen bond interactions, minimum and maximum phi and psi angles, repulsive forces, CH/Pi calculations, and other pertinent factors. In addition, front end applications can be developed to support retrieving, transforming, and preprocessing of information from the Research Collaboratory for Structural Bioinformatics (RCSB) [3] into the backend data repository.

Keywords: Proteomics; Bioinformatics; Query Languages; Protein Query Language.

1 Background

One of the major challenges facing biology and biochemistry researchers is the ability to view relationships among protein data, structures, functions, and pathways in a single query or at least in a concise and expressive manner [4]. For example, biochemists are performing cutting edge research into carbon-donated hydrogen bonds and their effect on protein structures [5]. To do so, they require data at the atomic level of the protein to perform calculations such as determining methyl-donated hydrogen bonds, repulsive forces, and CH/Pi interactions. Yet no online database is known to exist which supplies experimental data in an easy-to-use format at the atomic level without parsing the data manually, nor do tools exist to facilitate the calculations once data is parsed. To support their research, chemists have been downloading files from the RCSB in Protein Data Bank (.pdb) format, parsing data manually, and loading data into spreadsheets to perform calculations. This approach is tedious and potentially error prone, and spreadsheet limitations as well as other limiting factors obviate the need for a more efficient solution. For example, it is complicated in spreadsheets to answer the question “find all ‘acceptor’ atoms (i.e., oxygen, nitrogen, sulfur, or carbon) in a given model and chain of a protein within +/- 5 angstroms of a hydrogen atom which is considered potential methyl-donated hydrogen, and calculate the distance between the two atoms.” In addition, the number of atoms alone in large proteins may not fit within older spreadsheet program row limitations.

1.1 Dataset Repositories

Research chemists around the world do have access to various public protein data sources, but the access is not designed to support processing and retrieval at the atomic level. Online ‘databases’ supporting biochemistry research include Genbank, EMBL Data Library in the UK, the DNA Data Bank of Japan (DDBJ), and COLUMBA [6]. In essence, the only known public access to these databases is via a supplied front-end, and the returned data is formatted for user reading rather than for storing the data into a database for further processing and analysis.

Genbank provides meta level information about research performed on a given protein. Researchers can view what chains have been investigated, and download the meta data in a variety of formats including variations of XML. Researchers may view sequence alignments, and view pictures of what the protein chain looks like. However, this meta level is of insufficient detail for performing the atomic level research required.

Protein research data can be obtained via sites like the EMBL Data Library in the UK and the RCSB. The protein data is submitted to the RCSB by research investigators performing atom-level X-Ray crystallography, Nuclear Magnetic Resonance (NMR), and other types of studies. From these sites, researchers can download text files in a variety of formats (.pdb, .mmCIF, .xml and others) containing the detailed information required for research, yet the researcher has to parse the files to obtain the required detail data. Doing so for multiple files is a laborious process. In addition, error checking within the file and across multiple files for conditions which would preclude the researcher from using the protein is again a laborious process. These error conditions include but are not limited to: 1) atoms too close together in a given residue based on the van der Waals radius and/or in comparison to the average distance from one atom to another over multiple proteins; 2) atoms too close together across residues within a given protein under similar considerations as #1. In this research relational queries against a preliminary data model have already been written to find or confirm several active protein files on the RCSB where data is in an obvious error state.
Trissl presents a high level data model [6] as seen in Figure 1. This includes data from PDB, KEGG, SWISSPROT, CATH, SCOP, and others.

Figure 1: COLUMBA high-level schema

BioMolQuest [7] and iProt [4] both discuss importing protein data at the atomic level into a relational database. Also, Pryor and Fetrow discuss a relational database named PDB-SQL built in MySQL to handle protein-specific data at the atomic level [8]. The high level data model of PDB-SQL can be seen in Figures 2 and 3. In addition, this application was built for a specific purpose and had limited support for atom level data. Pryor proposed an extension to this model with fifty one (51) atom-level tables, one for each type of atom that may be included in a protein file.

Figure 2: Base schema. The residue table would store the only atom-level detail, and only on carbon atoms.

Eltabakh et al discussed an extensible database engine for biological databases [9][10]. The proposed engine “extends the functionalities of current DBMSs with (1) annotation and provenance management including storage, indexing, manipulation, and querying of annotation and provenance as first class objects in bdbms, (2) local dependency tracking to track the dependencies and derivations among data items, (3) update authorization to support data curation via content-based authorization, and (4) new access methods and their supporting operators that support pattern matching on various types of compressed biological data types.” While interesting, the focus of these researches is on annotation and provenance tracking, and is not the primary focus of this paper. However the concepts of annotation and provenance should be kept in mind during database design and maintenance. As demonstrated, there have been multiple attempts to import detailed protein data into relational databases. Yet the resulting database and data is either hidden by a front-end interface or is not available to the general public.

Figure 3: Extended schema: ‘alpha_carbon’ and ‘beta-carbon’ are representatives of the 51 atom-level tables.

1.2 Protein Data Parsing

As mentioned above, protein data obtained experimentally does exist and can be downloaded in .pdb, .mmCIF, and .xml formats. The .pdb format is designed for easy human reading. The .mmCIF and .XML formats are more structured in design, and use a data dictionary infrastructure [11]. These files may be downloaded from the RCSB via a web interface or via FTP. Software tools to parse the files include, but are not limited to, BioJava, BioPython, and CIFPARSE-OBJ. Software tools to facilitate loading the data into relational databases include but are not limited BioJava, BioSql, and “Db Loader”. A list of tools is available at the RCSB site [12] and the Bioinformatics Link Directory (Biolinks). BioJava was chosen to parse .pdb files. More detail on this approach, including strengths and weaknesses, is expanded upon later.

1.3 Query Tools

Online tools exist to find primary and secondary protein sequences, and to compare related primary sequences across
proteins [13][14]. However, there are no known tools which support querying the secondary structure of proteins, one of the goals of this research. And, again, there are no known tools for querying data at the atomic level.

The current dearth of tools for querying protein data can be likened to the banking industry some thirty years ago, where procedural rather than declarative code was written to access data. This made for fragile code from an architectural standpoint. This research provides a declarative query language against standard relational databases and investigates the strengths and impediments of this approach in consideration of the vast data sets involved and the various query types that may be necessary.

Patel describes a declarative protein secondary structure query language as well as an efficient implementation using histograms and optimization [1]. Since secondary structures at their simplest form can be described using an alphabet of three character (h=helices, e=beta-sheets, and l=turns or loops), a secondary structure for a given protein might be “eeeeeeeelllllh.” Patel suggests expressing the secondary structure sequences as a series of segments. The above example secondary structure would be represented as nine (9) e’s, five (5) l’s, and two (2) h’s. Patel then suggests a query language based on a triplet predicate of the form <type, min length, max length> where type refers to whether the segment is a helix, a beta-sheet, or a loop/turn, and the min length and max length refer to the length of the segment being queried. In addition, the type can be a wildcard. So, for example, a query such as <e 8 10><h 2 2> would match the example, since the first segment is of type ‘e’ and has length 9 (between 8 and 10), the second segment type matches the wildcard and is of length 5 (between 3 and 5 units long, inclusive), and the last segment matches the type ‘h’ and is 2 units long. The query would then be translated into SQL and executed against the database. Assuming the full sequence data is stored in the table protTbl and the segment data is stored in the table segTbl, the equivalent SQL code would be:

```sql
SELECT *
FROM protTbl p, segTbl s1, segTbl s2
WHERE s1.type = 'e'
AND s1.length BETWEEN 8 AND 10
AND s1.id = s2.id
AND s1.id = p.id
AND s2.start pos-(s1.start pos+s1.length) <= 5
AND s2.start pos-(s1.start pos+s1.length) >= 1
```

Note how the second segment in the query needs to be written as a relation between the first and third segment. For anyone unfamiliar with SQL, writing such a query might be a daunting task, and subject to error. As such, the proposed query language is an elegant way for researchers to query secondary structures of proteins. The language as developed, however, does not extend into the atomic level. In addition, this language does not operate against multiple columns in the same table or across multiple tables.

Although the subject area is the Semantic Web instead of a relational database, Roldan-Garcia proposed an interesting logic-based language named Extended Conjunctive Queries (ECQ) [2]. An example stated in the paper is:

```sql
SELECT distinct u1.url, u2.url
FROM uri index u1, uri index u2, worksfor 1 p w1, teacherof 1 p t1, course 1 c1
WHERE u2.id=t1.object and url1.id=w1.subject
AND w1.subject=t1.subject AND (w1.subject in (SELECT url FROM fullprofessor 1 c) OR w1.subject IN (SELECT url FROM assistantprofessor 1 c))
AND t1.object-cl1.id w1.object in (SELECT url FROM fullprofessor 1 c)
AND t1.subject-cl1.id w1.object in (SELECT url FROM fullprofessor 1 c)
AND w1.object LIKE '%university%' AND u1.url LIKE '%university%'
AND t1a.subject FROM teacherof 1 p t1a GROUP BY subject
HAVING COUNT(DISTINCT object) >=3 ) AND
t1a.object-cl1.id t1a.subject IN (SELECT tla.subject FROM teacherof 1 p tla GROUP BY tla.subject
HAVING COUNT(DISTINCT object) = (SELECT count (DISTINCT object) FROM teacherof 1 p WHERE subject=t1a.subject AND object IN (SELECT url FROM course 1 c) GROUP BY subject)) ORDER BY u1.url
```

which after processing would translate to the SQL query:

```sql
SELECT * FROM protTbl p, segTbl s1, segTbl s2
WHERE s1.type = 'e'
AND s1.length BETWEEN 8 AND 10
AND s1.id = s2.id
AND s1.id = p.id
AND s2.start pos-(s1.start pos+s1.length) <= 5
AND s2.start pos-(s1.start pos+s1.length) >= 1
```

An ECQ expression has the form:

```sql
ans(V1,V2,V3,...,Vn) \< Q1 AND Q2 AND … Qn
```

where each Qi can take the form:

1. C(x)
2. P(x,y)
3. C(x) OR D(x)
4. ALL C(x)
5. <=n P(x,y)
6. >=n P(x,y)
7. =n P(x,y)

where C and D are class names, P is a property name, x and y are instance names or variables, and n is a natural number. The simplicity of ECQ’s approach may be useful in the target query language.

Another language with interesting features is TQL, proposed by Conforti et al [15]. Again, TQL is a language for semi-structured data that can be used to query XML, but is built on set comprehension in the tradition of SQL and other languages. An example stated in the paper of a query in TQL would be:

```sql
FROM $Bib |= .bib[.book[.year[1991] And .title[$t]]]
SELECT title[$t]
```

where should be read: “there is a path .bib[.book[.year[1991] And .title] that reaches a place that matches .year[1991] And .title, i.e. a
place where you find both a path .year[] leading to 1991 and a path .title[ ] leading to something, that you will call $t$.

2 **PQL**

The Protein Query Language (PQL) is declarative in nature. Users of the language have access to the following features:

1. Users may utilize familiar terms when referring to proteins, models, chains, residues, atoms, and other chemistry terms. The underlying relational model is abstracted from the user.
2. The ability to use mathematical, boolean, and string functions as part of the language. However, constructs such as conditionals and looping are supported at this time.
3. The user shall be able to save PQL constructs for later utilization.

2.1 **Grammar**

The grammar for the PQL was developed in Backus-Naur Form (BNF) using the grammar constructs provided within a software package named Gold-Parser Builder. The SQL statement was used to calculate the potential methyl-donated hydrogen bonds for a given protein, and therefore represents a practical example in biochemistry research. It can be easily seen the PQL representation of this calculation may be much easier for a non-SQL expert to develop. Further explanation of the grammar follows below.

The grammar is divided into five (5) distinct areas, two (2) of which are required and three (3) of which are optional as described here:

1. EQUIVALENCE (optional): An example best illustrates the use of an equivalence statement. Say, for example, in some portions of their query a user would like to reference a hydrogen atom as ‘h’ for brevity, whereas in other sections it might be more instructive to reference that same hydrogen atom as ‘hydrogenAtom’. A user may add the equivalence statement ‘h hydrogenAtom’ with the semantics ‘h is the same object as hydrogenAtom’.

2. INSTANCE: At least one statement required. An instance statement allows a user to tie an instance variable to a ‘table’, or in user-terms a group of chemically related items. For example, the statement:

\[
\text{Protein}(p).\text{Model}(?).\text{Chain}(?).\text{Residue}(r).\text{Atom}(a)
\]

allows the user to tie the instance variable ‘p’ to a protein structure, ‘r’ to a residue structure, and ‘a’ to an atom structure. In the ASSIGNMENT and CONSTRAINT sections, the user can place stipulations on how these instance variables are bound. The ‘?’ variables are used as wildcards. In addition, multiple statements using the same instance variable tie statements together. For example:

\[
\text{Protein}(p).\text{Model}(m).\text{Chain}(c).\text{Residue}(r).\text{Atom}(c)
\]

means in essence that atom instance variables ‘c’ and ‘h’ share the same protein, model, chain, and residue. Again, in the ASSIGNMENT and CONSTRAINT sections the user might further restrict ‘c’ to be a carbon atom, and ‘h’ to be a hydrogen atom. ‘Fields’ within the instance variable can then be accessed in the ASSIGNMENT, CONSTRAINT, and RESULTS sections. For example, an atom has an atom name and potentially X, Y, and Z coordinates (if it has 3-D data associated with it). In the above example, these fields within the hydrogen structure would be accessed as h.atomName, h.xcoor, h.ycoor, h.zcoor. The user would have a list of the accessible fields per structure.

3. ASSIGNMENT (optional): An assignment statement takes the form:

\[
\begin{align*}
\text{a.} & \quad \text{thetaAngle}=\text{ThetaAngle}(\text{chDist},\text{cxDist},\text{hxDist}) \\
\text{b.} & \quad \text{tempName} = \text{StringAdd}('C', \\
& \quad \text{Substring(h.atomName,2,Len(h.atomName)-2),}'%) \\
\text{c.} & \quad s = A \text{ AND } B \text{ OR } C \\
\text{d.} & \quad h.\text{atomName} = "H11"
\end{align*}
\]

Assignment can be made to a temporary variable (e.g., tempName) or to an instance variable’s field (e.g., h.atomName). Assignments can include combinations of boolean, string, and mathematical expressions.

4. CONSTRAINT (optional): The user can constrain certain conditions on the resultant returned data. Examples of constraint statements include:

\[
\begin{align*}
\text{a.} & \quad \text{cxDist} \geq 4.2 \\
\text{b.} & \quad \text{thetaAngle} \text{ BETWEEN 150.0 AND 210.0F} \\
\text{c.} & \quad \text{carbonAtom.}\text{atomName LIKE ('C' +} \\
& \quad \text{Substring(h.atomName,2,Len(h.atomName)-2) +} \\
& \quad \text{')%')} \\
\text{d.} & \quad \text{carbonAtom.}\text{atomName LIKE "CH11"}
\end{align*}
\]

Constraints can include combinations of boolean, string, and mathematical expressions.

5. RESULTS: At least one statement is required. Result statements are where users specify what the returned dataset looks like and how it is sorted. Typical statements look like:

\[
\begin{align*}
\text{a.} & \quad \text{h.}\text{atomName ASC 1} \\
\text{b.} & \quad \text{tempString as carbonAtomName DESC 2} \\
\text{c.} & \quad \text{tempString2 DESC 3 NO OUTPUT}
\end{align*}
\]

User can specify an instance variable field (e.g., h.atomName) or a temporary variable (e.g., tempString) as well as an optional output name for that variable (e.g., ‘carbonAtomName’ above) and an ascending or descending order (e.g., the ‘ASC x’ and ‘DESC y’ portions of the statements above). The result set is returned in the column order specified line-by-line, and sorted in the order
specified by the ASC (or ASCENDING) and DESC (or DESCENDING) sub-statements. Lastly, user can specify ‘NO OUTPUT’ to restrict a given Result statement from the output.

Users also have access to useful ‘method’ calls in the ASSIGNMENT and CONSTRAINT sections including methods such as:

1. Distance(atom1,atom2)
2. Distance(xcoor1,xcoor2,ycoor1,ycoor2,zcoor,zcoor2)
3. ThetaAngle(distancelto3, distancelto2, distanc e2to3)

3 Conclusion

As detailed above, the vast preponderance of computational tools available to protein researchers seem to concentrate on predictions at the amino acid residue level, including prediction of the secondary state. Important biochemistry research is being done at the atom level, yet little or no computing tools are publicly available to biochemists to support this work. The PQL language is an attempt to provide an intuitive declarative language within query application to researchers who are unfamiliar with SQL coding. The PQL query system allows users to interrogate a relational database containing protein data downloaded from the RCSB Protein Data Bank. Users can create queries to identify important research interactions between methyl-donated hydrogen bonds, amine repulsions, and CH/Pi interactions. We expect users of the new system to gain significant insight into research areas such as the tertiary structure of proteins.

4 References


