Using Relational Databases for Improved Sequence Similarity Searching and Large-Scale Genomic Analyses

As protein and DNA sequence databases have grown, characterizing evolutionarily related sequences (homologs) through sequence similarity has paradoxically become a more challenging endeavor. In the early 1990s, a similarity search might identify a dozen homologs only once in three searches; many searches would reveal only one or two homologs, if any. With today's comprehensive sequence libraries, most similarity searches will identify several dozen homologous sequences, and many searches will yield hundreds of homologs from dozens of species. As scientifically interesting as these results may be, they are often impractical to organize and analyze manually. Moreover, modern genome-scale studies do 1,000 to 10,000 searches in a single analysis, producing millions of lines of comparison results. Fortunately, relational databases (*UNITS 9.1 & 9.2*) can manage large sets of search results, greatly simplifying genome-scale analyses—for example identifying the most conserved sequences shared by two organisms, or the proteins that are found in plants but not animals. Relational databases are designed to integrate diverse types of information: e.g., sequence, taxonomy, similarity to other proteins, and gene location. Relational databases can also make searches more efficient by focusing on subsets of the protein databases—proteins found in similar organisms or with similar functions. Thus, relational databases are not only essential for the management and analysis of large-scale sequence analyses, but can also be used to improve the statistical significance of similarity searches by focusing the search on subsets of sequence libraries most likely to contain homologs, based, e.g., on taxonomy, structure, or function.

The protocols in this unit use relational databases to improve the efficiency of sequence similarity searching and to demonstrate various large-scale genomic analyses of homology-related data. Basic Protocol 1 illustrates the installation and use of a simple protein sequence database, `seqdb_demo`, which will be used as a basis for all the other protocols. Basic Protocol 2 then demonstrates basic use of the `seqdb_demo` database to generate a novel sequence library subset. Basic Protocol 3 shows how to extend and use `seqdb_demo` for the storage of sequence similarity search results. Basic Protocols 4 to 6 make use of various kinds of stored search results to address three different aspects of comparative genomic analysis. All of the SQL statements used in these protocols are available in the `seqdb_demo` package, described in Basic Protocol 1. While many of the SQL statements are briefly explained in each protocol, the concepts in Basic Protocols 2 to 4 will easier to understand if the reader is familiar with basic SQL (*UNIT 9.2*).

INSTALLING AND POPULATING THE `seqdb_demo` RELATIONAL DATABASE

BASIC PROTOCOL 1

In this protocol, a very simple protein sequence database, `seqdb_demo` (Fig. 9.4.1) will be installed and then populated with data obtained from a "flat-file" sequence library. The database includes: (1) a table for the raw sequence data; (2) a table to hold information about the sequence, including its description and various public database accession numbers; and (3) tables to store taxonomic information about the organism from which the sequence was obtained, and how those organisms are themselves related to each other.

Sequence and annotation information are loaded from a sequence library "flat file" into an empty `seqdb_demo` database using the Perl program `load_seqdemo.pl`, found in

Contributed by Aaron J. Mackey and William R. Pearson
Current Protocols in Bioinformatics (2004) 9.4.1-9.4.25

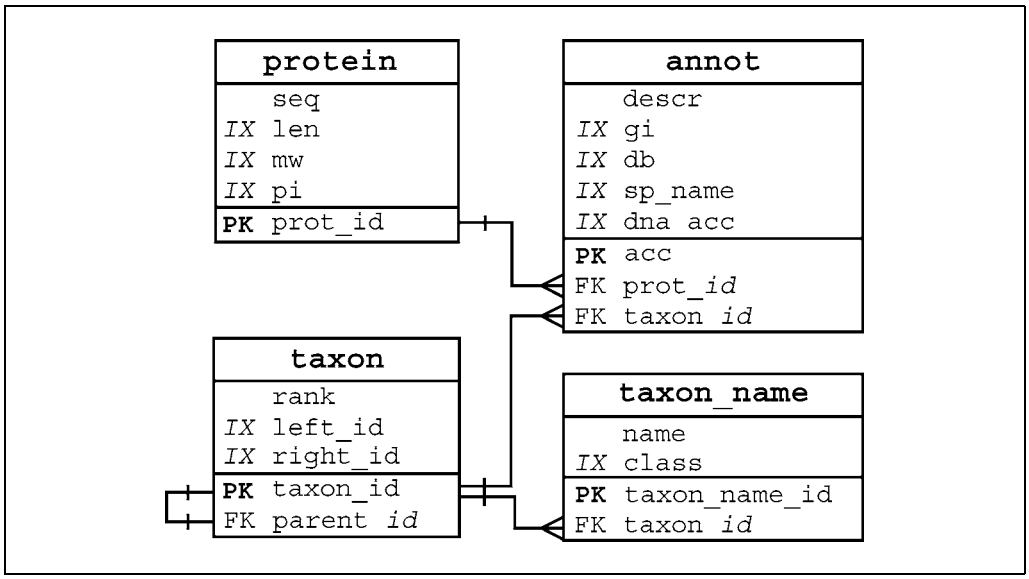

Figure 9.4.1 A schema for protein sequence data. Each of the boxes represents one of the tables in the `seqdb_demo` database. Sequences are stored in the `protein` table, their descriptions and accession information are stored in the `annot` table, and taxonomic information is stored in the `taxon` and `taxon_name` tables. The links between the tables are shown with dashed lines. The symbols at the ends of the line indicate the type of relationship; e.g., the `protein:annot` relationship is a *one-to-many* relationship; each protein sequence can have *many* descriptions or annotations but an annotation refers to only *one* protein sequence. The abbreviations to the left of the table entry names indicate whether the entry is a primary key (PK) or foreign key (FK, a foreign key in one table is a primary key in another, and allows the information in the two tables to be "joined"), or if the entry is indexed (IX) for rapid lookup.

the `seqdb_demo.tar.gz` package. Although the comprehensive `nr` protein sequence library from the NCBI will be used, any FASTA-formatted database (*APPENDIX 1B*) can be used, provided that descriptions follow the NCBI nonredundant DefLine format, e.g.:

```
>gi|15241446|ref|NP_196966.1| (NM_121466) putative protein
[Arabidopsis thaliana]^Agi|11281152|pir||T48635
hypothetical protein T15N1.110 — Arabidopsis
thaliana^Agi|7573311|emb|CAB87629.1| (AL163792) putative
protein [Arabidopsis thaliana]
```

See *ftp://ftp.ncbi.nih.gov/blast/db/blastdb.txt* for further description of this specialized FASTA sequence format.

The protocol steps below demonstrate how to extract subsets of sequences from specific taxonomic groupings.

Necessary Resources

Hardware

Computer with at least 2 Gb of disk space available for the raw data flat-files and the MySQL sequence database files

Software

Windows- or Unix-based operating system (including Linux or Mac OS X)
Working version of MySQL, with functional database permissions. MySQL can be downloaded from *http://www.mysql.com* and installed as described in *UNIT 9.2*. All interactions with MySQL databases in these protocols will be via the `mysql` command-line utility.

A terminal application connected to a Unix environment in which one can execute Unix-like commands. For Windows-based operating systems, this entails installing the Cygwin Unix emulation (*http://www.cygwin.com*).

The Perl scripting language interpreter (any version since 5.005_03) and the `DBI`, and `DBD::mysql` modules. With Unix-like systems, the `DBI` and `DBD::mysql` modules can be installed from the CPAN Perl software repository with the following commands (typed input indicated in bold):

```
% perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

In some cases, it may be necessary to type **`force install DBD::mysql`** (at the `cpan` prompt) if errors are encountered (generally, these errors can safely be ignored). Under Windows-based operating systems, the `ppm` package management utility should be used instead to install both the `DBI` and `DBD::mysql` packages.

Files

The `seqdb_demo` package of SQL and Perl scripts for creating and maintaining a relational database of protein sequences, downloaded from *ftp://ftp.virginia.edu/pub/fasta/CPB/seqdb_demo.tar.gz*. This package includes all of the utilities to create, load, and maintain the simple protein sequence database described in these protocols.

A FASTA-format (*APPENDIX 1B*) "flat-file" protein sequence library, such as SwissProt or `nr`. These sequence libraries can be downloaded from *ftp://ftp.ncbi.nih.gov/blast/db/FASTA/swissprot.gz*, or *nr.gz*. The `nr` library is more comprehensive, but the SwissProt library is a smaller, more manageable dataset. In these protocols, the `nr` sequence library will be exclusively used.

Creating the seqdb_demo database

1. In a Unix terminal window, traverse into the directory in which the `seqdb_demo.tar.gz` package file was downloaded and execute the commands listed below (type the text in **`bold`** to issue the command; the computer response is in `lightface` and comments about the commands are written in *italics*).

   ```
   % tar -xzvf seqdb_demo.tar.gz
   ```

   Uncompresses and unpacks `seqdb_demo`.

   ```
   % cd seqdb_demo
   ```

   Changes directory into `seqdb_demo`.

   ```
   % mysql < mysql/seqdb_demo.sql
   ```

   Creates the database and its tables.

   Before executing the third command, one may wish to edit the top few lines of `mysql/seqdb_demo.sql` to change the user name, and password from the defaults (`seqdb_user and seqdb_pass`, respectively).

2. To confirm that the database has been created correctly (and to become familiar with the database's schema), type the following:

   ```
   % mysql -u seqdb_user -pseqdb_pass seqdb_demo
   mysql> SHOW TABLES;
   ```

   Provides a listing of the tables found in this database (Fig. 9.4.2A).

A

```
+---------------------+
|Tables_in_seqdb_demo |
+---------------------+
| annot               |
| protein             |
| taxon               |
| taxon_name          |
+---------------------+
```

B

```
+----------+---------------------------------+------+-----+-------+-----+
| Field    | Type                            | Null | Key |Default|Extra|
+----------+---------------------------------+------+-----+-------+-----+
| gi       | int(10) unsigned                |      | PRI |  0    |     |
| acc      | varchar(20)                     |      | MUL |       |     |
| ver      | tinyint(3) unsigned             |  YES | MUL |       |     |
| db       | enum('emb','ref','pdb','pir',..|      | MUL |  gb   |     |
| prot_id  | int(10) unsigned                |      | MUL |  0    |     |
| taxon_id | int(10) unsigned                |  YES | MUL |  NULL |     |
| sp_name  | varchar(20)                     |  YES |     |       |     |
| dna_acc  | varchar(20)                     |  YES |     |       |     |
| descr    | text                            |      |     |       |     |
| pref     | tinyint(3) unsigned             |      | MUL |  0    |     |
+----------+---------------------------------+------+-----+-------+-----+
```

Figure 9.4.2 (**A**) List of tables in the database created in Basic Protocol 1, step 1, retrieved via the **`SHOW TABLES`** command. (**B**) Description of columns in the database, retrieved via the **`DESCRIBE annot`** command.

`mysql>` **`DESCRIBE annot;`**

Gets a description of the columns in the `annot` *table (Fig. 9.4.2B).*

These commands confirm that one has successfully created the `seqdb_demo` *database with four tables, as described in Fig. 9.4.1. Briefly, the* `protein` *table will store raw protein sequences and the* `annot` *table (short for "annotation") will contain the description of the protein and any links to external public databases (SwissProt, Genpept, PIR, TrEMBL, etc.), while the other two tables (*`taxon` *and* `taxon_name`*) will provide taxonomic species information.*

Populating the **`seqdb_demo`** *database*

3. To load the sequences from the `nr` FASTA-format sequence library, type the following:

`%` **`gunzip /seqdata/nr.gz`**

Uncompresses the file.

`%` **`load_seqdb.pl /seqdata/nr`**

Loads the data into the database.

In these commands, `/seqdata` should be changed to the directory of the compressed `nr.gz` file previously downloaded from *ftp://ftp.ncbi.nih.gov/blast/db/FASTA/nr.gz* (see Necessary Resources, above). The `load_seqdb.pl` script reads every sequence entry from the specified sequence library, storing the sequence data in the `protein` table and the header information in the `annot` table. For a large protein database like `nr` (which in March of 2004 contained nearly 2 million entries), this initial loading may take 6 to 12 hr.

4. To confirm that the database has successfully loaded the protein sequences and their annotations, type (from a `MySQL` prompt):

A

```
+----------+
| count(*) |
+----------+
|  2715099 |
+----------+
```

B

```
+----------+
| count(*) |
+----------+
|  4239984 |
+----------+
```

C

```
+----------+--------------------------------------+-----+-----+---------+
| prot_id  | seq                                  | len | pi  | mw      |
+----------+--------------------------------------+-----+-----+---------+
|      100 | MKIETSRFGlldidddkiitlpdAMPGFAETR...  | 153 | 4.6 | 16980.6 |
+----------+--------------------------------------+-----+-----+---------+
```

D

```
+----------+------+-----------+-----------------------------------------------+
| gi       | db   | acc       | descr                                         |
+----------+------+-----------+-----------------------------------------------+
| 13812230 | ref  | NP_113361 | putative ribosome bi[...]n [Guillardia theta] |
| 25396809 | pir  | F90098    | putative ribosome bi[...]ia theta nucleomorph |
| 13794542 | gb   | AAK39917  | putative ribosome bi[...]n [Guillardia theta] |
+----------+------+---------=-+-----------------------------------------------+
```

Figure 9.4.3 (**A**) Number of protein sequences loaded into database from the `nr` sequence library, retrieved via the **`SELECT COUNT(*) FROM protein`** command. (**B**) Number of different descriptions loaded into the database from `nr`, retrieved via the **`SELECT COUNT (*) FROM annot`** command. (**C**) Information on a single protein, retrieved via the **`SELECT * FROM protein WHERE prot_id = 100`** command. (**D**) All annotations of a protein, retrieved via the **`SELECT gi, db, acc, descr`** command.

```
mysql> SELECT COUNT(*) FROM protein;
```

Reports the number of protein sequences (Fig. 9.4.3A).

```
mysql> SELECT COUNT (*) FROM annot;
```

Reports the number of different descriptions (Fig. 9.4.3B).

```
mysql> SELECT * FROM protein WHERE prot_id = 100;
```

Get a single protein (Fig. 9.4.3C).

```
mysql> SELECT gi, db, acc, descr
   +> FROM annot WHERE prot_id = 100;
```

Get all annotations of a protein (Fig. 9.4.3D).

Because the `nr` *database is constantly growing, results may not exactly match those above.*

5. To add species taxonomic information to all of the protein sequence entries in the database, it is necessary to download information from the NCBI Taxonomy database. The `updatetax.pl` script automatically downloads this information and uses it to load the taxonomy-related tables in the `seqdb_demo` database. Type the following:

```
% mkdir /seqdata/taxdata
```

Makes a new directory for NCBI Taxonomy download.

A
```
+----------+
| count(*) |
+----------+
|   212242 |
+----------+
```

B
```
+--------------+---------------------+
| name         | class               |
+--------------+---------------------+
| Homo sapiens | scientific name     |
| human        | genbank common name |
| man          | common name         |
+--------------+---------------------+
```

Figure 9.4.4 (**A**) Total number of taxa loaded from the NCBI Taxonomy database, retrieved via the **`SELECT COUNT(*) FROM taxon`** command. (**B**) NCBI's ID for human, retrieved via the **`FROM taxon_name WHERE taxon_id = 9606`** command.

```
% updatetax.pl /seqdata/taxdata
```

Downloads and imports the NCBI Taxonomy database.

6. To confirm that the NCBI Taxonomy database was successfully loaded into the database, type the following commands:

```
mysql> SELECT COUNT(*) FROM taxon
```

Gets total number of taxa (Fig. 9.4.4A).

```
mysql> SELECT name, class
    +> FROM taxon_name WHERE taxon_id = 9606
```

Gets NCBI's ID for human (Fig. 9.4.4B).

Again, one may expect to see slightly different values, as the NCBI Taxonomy database continues to grow.

BASIC PROTOCOL 2

EXTRACTING SEQUENCES FROM `seqdb_demo` FOR SIMILARITY SEARCHING TO IMPROVE HOMOLOG SEARCHING

The inference of sequence homology is based on the identification of statistically significant sequence similarity. If an alignment between two sequences is statistically significant, one can reliably infer that the sequences are homologous. However, if the score is *not* significant, one cannot be certain the sequences are not homologous; in fact, many truly homologous proteins (where homology is inferred by significant structural similarity) do not share significant sequence similarity. The significance of an alignment is measured by the expectation value *E*, which describes the number of alignments of similar or better similarity that could be expected to occur by chance alone. The *E* value is calculated as $E = P \times D$, where *P* is the probability of seeing an alignment this good between any given pair of sequences and *D* is the total number of pairwise comparisons performed during the search. Therefore, one of the easiest ways to improve the sensitivity of a similarity search is to search a subset of sequence libraries, reducing *D* and improving the significance of all *E* values (nonhomologous alignments will continue to have *E* values $\approx$1.0 or greater). This strategy is particularly effective now that many complete prokaryotic and eukaryotic genomes and proteomes are available. For example, searching only against the proteins predicted from a complete genome instead of the entire `nr` sequence library,

can improve the statistical significance of homologous alignments by 100 to 1000-fold, greatly enhancing the efficiency of the search for homologs in the given organism.

In addition, by searching against specific taxonomic subsets of a sequence library, one can tailor various scoring parameters to the evolutionary distance being considered. For example, modern mammals shared a common ancestor only about 100 million years ago, and so most mammalian orthologs share modestly high protein sequence identity (70% to 85%, on average). The BLOSUM50 scoring matrix (the default for FASTA), or BLOSUM62 scoring matrix (the default for BLAST), is "tuned" to be able to identify distant homologs that share less than 30% identity over long regions, but in return may not be able to reliably identify shorter homologies that have high identity. Conversely, the PAM40 matrix is targeted to sequences that share approximately 70% identity, and thus should be more effective at identifying and accurately aligning mammalian orthologs, particularly those that are too short to identify using the default matrices. Gap penalties can be similarly adjusted to be more or less forgiving, based on the approximate evolutionary distance between library and query sequences.

There are many other motivations for wanting to search against smaller subsets of available sequences. The most general strategy for searching against a taxonomic (or other) subset of a larger sequence database is to use the fully populated `seqdb_demo` database to generate customized, FASTA-formatted sequence libraries. This protocol will demonstrate how to generate both species-specific and clade-specific sequence database flat files from the `seqdb_demo` relational database.

Necessary Resources

Hardware

Computer with at least 2 Gb of disk space available

Software

Windows- or Unix-based operating system (including Linux or Mac OS X)
Working version of MySQL, with functional database permissions. MySQL can be downloaded from *http://www.mysql.com* and installed as described in *UNIT 9.2*. All interactions with MySQL databases in these protocols will be via the `mysql` command-line utility.
A terminal application connected to a Unix environment in which one can execute Unix-like commands. For Windows-based operating systems, this entails installing the Cygwin Unix emulation (*http://www.cygwin.com*).
The Perl scripting language interpreter (any version since 5.005_03) and the `DBI`, and `DBD::mysql` modules

Files

Generated as in Basic Protocol 1

1. Complete Basic Protocol 1.

To generate a species-specific sequence library

2a. To generate a library of human sequences (or sequences from any other species for which the preferred scientific name is known), create a text file (e.g., `human.sql`, found in the `seqdb_demo` distribution) with SQL code (see *UNIT 9.2*) that generates the desired sequences. In this case the SQL code would be that shown in Figure 9.4.5.

3a. Once this file has been created and saved, use it as input to the `mysql` client with the following command:

```
SELECT CONCAT(">gi|", gi, "|", db, "|", acc, "| ", descr, "\n", seq)
FROM   protein
       INNER JOIN annot USING (prot_id)
       INNER JOIN taxon_name USING (taxon_id)
WHERE  annot.pref = 1
  AND  taxon_name.class = "scientific name"
  AND  taxon_name.name = "Homo sapiens";
```

Figure 9.4.5 SQL code used to generate a library of human sequences (note the space following the fourth "|" symbol).

```
>gi|8924190|ref|NP_061004| hypothetical protein PRO2714
MQILRFSVYRSQPREHLFFTFLLKIHWGGIKFTPKKFAWAKSLQVPSENKILMIFFFFWL...
>gi|18585169|ref|XP_102310| hypothetical protein XP_102310
MGWGSNqqvvlcfllcqlYSSRLFFYggkklrvvkgvgDLQERQSGSCPEGGQGITNCDK...
```

Figure 9.4.6 FASTA-formatted human sequences, printed to `human.lib`.

```
% mysql -rN seqdb_demo < human.sql > human.lib
```

The –r flag tells mysql that the output should be left "raw," so that the embedded newline characters, \n, will be correctly interpreted; the -N flag prevents mysql from printing any column names. Together, this command selects all human sequences (and their preferred annotations) from the `seqdb_demo` database and prints them to `human.lib`, already converted into FASTA format, e.g., Figure 9.4.6.

4a. The SQL command script in Figure 9.4.5 generates valid FASTA-formatted files, but the sequence is all on one line. This can be problematic for sequence analysis tools that read sequences line-by-line into small buffers. To reformat the database so that sequences are on multiple lines with a maximum length of 60, the `reformat.pl` Perl script is included in the `seqdb_demo` distribution.

```
% reformat.pl human.lib
```

To generate taxonomic subsets

The `updatetax.pl` script described in Basic Protocol 1 calculates additional information (the `left_id` and `right_id` values) that can be used to select entire taxonomic subgroupings of species, e.g., all mammals or all vertebrate species. These two `left_id`/`right_id` numbers have the useful property that any descendents of a taxonomic node will have `left_id`'s (and `right_id`'s) that are between the `left_id` and `right_id` range of all their parent node; this is referred to as a "nested-set" representation of the hierarchy, and can be used to select entire hierarchical subsets without recursion (Celko, 1999).

2b. Generate a library of mammalian sequences from `seqdb_demo`; to do so, create a file (e.g., `mammalia.sql`, found in the `seqdb_demo` distribution) with the SQL code shown in Figure 9.4.7.

3b. As in Step 3a, use this `SQL` script to generate the sequence library with the following command:

```
% mysql -rN seqdb_demo < mammalia.sql > mammalia.lib
```

4b. Reformat the library as in step 4a with the following command.

```
% reformat.pl mammalia.lib
```

```
SELECT  CONCAT(">gi|", gi, "|", db, "|", acc, "| ", descr, "\n",
seq)
FROM    protein
        INNER JOIN annot USING (prot_id)
        INNER JOIN taxon AS all_mammalia USING (taxon_id)
        INNER JOIN taxon AS mammalia
            ON (all_mammalia.left_id BETWEEN mammalia.left_id
                                         AND mammalia.right_id)
        INNER JOIN taxon_name USING (taxon_id)
 WHERE annot.pref = 1
   AND taxon_name.name = 'Mammalia'
   AND taxon_name.class = 'scientific name';
```

Figure 9.4.7 SQL code used to generate a library of mammalian sequences from `seqdb_demo`.

To generate a BLAST-searchable taxonomic subset

The BLAST algorithms (*UNITS 3.3 & 3.4*) require sequence libraries to be specially formatted and indexed to accelerate searches. The NCBI-BLAST and WU-BLAST versions use the `formatdb` and `xdformat` utilities, respectively, to perform this reformatting. However, the NCBI-BLAST versions provide a mechanism to specify a subset of a sequence library (by GI numbers) without the generation of custom sequence libraries and reformatting.

5. Using the `formatdb` utility, reformat the `nr` database for use with NCBI-BLAST programs:

   % formatdb -p T -i /seqdata/nr

6. Alter the `SELECT` line from the SQL script (in step 3a or step 3b) to select only `gi` numbers:

   ```
   SELECT gi
   FROM [...]
   WHERE [...]
   ```

7. Execute the revised SQL:

   % mysql -rN seqdb_demo < mammalia_gi.sql > mammalia.gi

8. Use this `GI` list file (specified with `-l`) for any BLAST search against the `nr` sequence library:

   % blastall -p blastp -i query.fa -l mammalia.gi -d /seqdata/nr

   See UNITS 3.3 & 3.4 for further discussion of many of the commands and arguments used in the steps above.

STORING SIMILARITY SEARCH RESULTS IN `seqdb_demo`

BASIC PROTOCOL 3

Most sequence-similarity search programs produce human-readable, textual output. While this text has important information embedded within it—sequence descriptions, scores, alignment boundaries, etc.—it is not practical for an investigator to look at all the results when hundreds of homologies are detected, or when thousands of independent searches are run. To manage and make efficient use of large sets of search results, the data must be organized and indexed for easy querying and retrieval. Furthermore, the ratio of actual similarity and alignment data to white space and formatting text in the output is often fairly low, making the files easy to read, but much larger than necessary. Finally,

keeping the search results in separate results files makes it more difficult to integrate search results with other information. This protocol addresses many of these problems by storing results from sequence similarity searches in the `seqdb_demo` relational database.

Every similarity-searching program—e.g., BLAST (*UNITS 3.3 & 3.4*), FASTA (*UNIT 3.9*), SSEARCH (*UNIT 3.10*), or HMMER—produces somewhat different similarity and alignment results. Some programs produce alignments with both gaps and frameshifts, while other programs may provide many separate alignment blocks (e.g., BLAST HSP's). To create a generic table structure able to store results from most similarity-search programs, the focus of this protocol will be on the common types of data produced by these programs; any data specific only to one algorithm will be ignored, and the BioPerl software will be used to extract these common data. In general, the programs perform many pairwise comparisons between one (or more) query sequence(s) and many entries in a sequence library, reporting only the most similar (or most significant) sequence comparisons. Each pairwise comparison produces an alignment with an associated raw score and statistical score (usually expressed as *bits*), as well as an overall estimate of the alignment's statistical significance (*E* value). Additionally, some alignment information, including the boundaries in the query and library sequences, the number and position of gaps, etc., is usually available. Finally, summary information such as percent identity and lengths of the two sequences may be provided.

Necessary Resources

Hardware

Computer with at least 2 Gb of disk space available

Software

Windows- or Unix-based operating system (including Linux or Mac OS X)

Working version of MySQL, with functional database permissions. MySQL can be downloaded from *http://www.mysql.com* and installed as described in *UNIT 9.2*. All interactions with MySQL databases in these protocols will be via the `mysql` command-line utility.

A terminal application connected to a Unix environment in which one can execute Unix-like commands. For Windows-based operating systems, this entails installing the Cygwin Unix emulation (*http://www.cygwin.com*).

The Perl scripting language interpreter (any version since 5.005_03) and the `DBI`, and `DBD::mysql` modules.

The BioPerl toolkit (*http://www.bioperl.org*; available via CPAN, see Basic Protocol 1) should be installed.

Files

The `seqdb_demo` package of SQL and Perl scripts for creating and maintaining a relational database of protein sequences, downloaded from *ftp://ftp.virginia.edu/pub/fasta/CPB/seqdb_demo.tar.gz*. This package includes all of the utilities to create, load, and maintain the simple protein sequence database described in these protocols.

Similarity search results from FASTA (*UNIT 3.8*), BLAST (*UNITS 3.3 & 3.4*), SSEARCH (*UNIT 3.10*), or HMMER

A sample set of similarity results is available from *ftp://ftp.virginia.edu/pub/fasta/CPB/ec_human.results.gz* to produce the file `ec_human.results`.

1. Complete Basic Protocol 1.

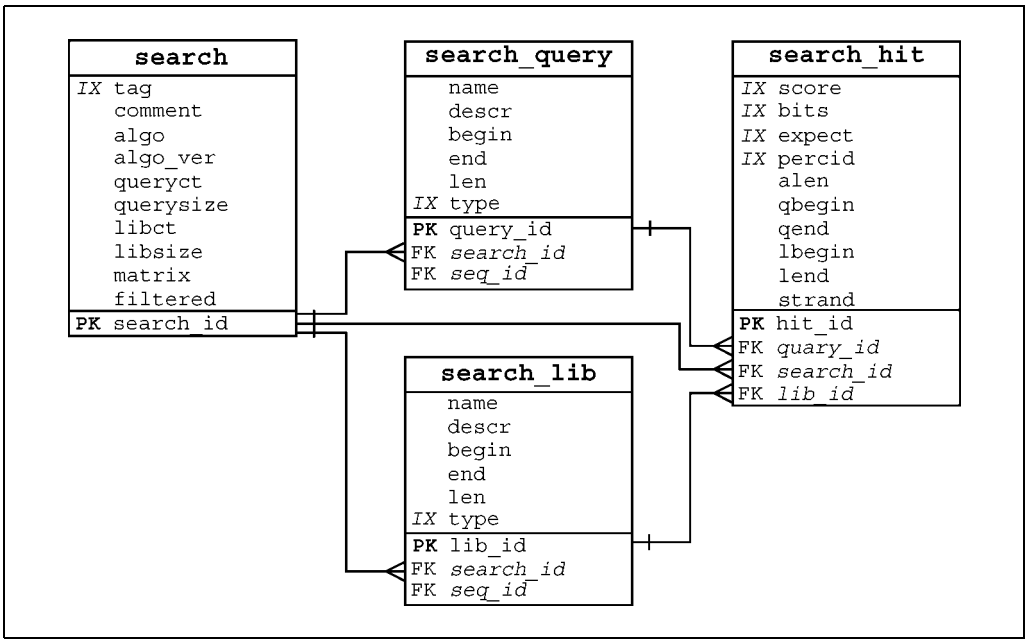

Figure 9.4.8 A schema for similarity search results. Each of the boxes represents one of the tables used to collect alignment data in the `seqdb_demo` database. The `search` table records the parameters of the search; `search_query` and `search_lib` record information about the query and library sequences used for the search, and the `search_hit` table records the scores and boundaries of alignments between query and library sequences. The links between tables, primary keys (PK), and foreign keys (FK) are indicated as in Figure 9.4.1.

Extending **seqdb_demo** *to include similarity search results*

2. An SQL script, `search.sql`, is included in the `seqdb_demo` distribution to add the tables related to sequence similarity search results:

```
% mysql seqdb_demo < mysql/search.sql
```

3. As in Basic Protocol 1, step 2, again execute `SHOW TABLES` and `DESCRIBE <table>` statements for each of the `search`, `search_query`, `search_lib`, and `search_hit` tables to confirm their existence in the database, and to become familiar with them (also see Fig. 9.4.8). Briefly, for any one set of similarity results, a single row will be stored in the `search` table, summarizing the search (algorithm used, parameters, etc.). Each query used for the search will be stored in the `search_query` table, while any library sequence reported in the search will be stored once in the `search_lib` table. Information about the alignments between any query and library sequences is stored in the `search_hit` table.

Importing similarity search results

4. Run the `loadsearch.pl` script, provided with the `seqdb.demo` distribution to parse and load the sequence similarity search data (e.g., `ec_human.results`) into the database:

```
% loadsearch.pl --format fasta --tag ecoli_vs_human \
--comment 'E. coli vs human proteome' < fasta.results
```

Similarity search results are imported into the database by parsing the raw text output and entering the sequence names, scores, and boundaries into the various search-related tables. The BioPerl toolkit provides functions for parsing BLAST, FASTA, and HMMER text results, among others, which are easily combined with Perl DBI database modules to store search results. The provided `loadsearch.pl` *script from*

the seqdb_demo *distribution makes use of the BioPerl-based result parsers, so it theoretically should be able to accommodate any result formats that BioPerl can parse. Furthermore,* loadsearch.pl *assumes that all query and library sequences either (a) have the NCBI-like "DefLine" header ID found in the* nr *and similar flat files (e.g.,* gi|123456|gb|CAA1128383.1*), or (b) have a customized ID of the form* table.field|key *(e.g.,* contig.contig_id|9876 *or* annot.acc|X12983*) that references a sequence obtainable via the provided table and key field. The key will be used in the* seq_id *field of the* search_query *and* search_lib *tables, and either* GI *or* annot.acc*, etc., will be used as the* type*.*

Additionally, the FASTA*-specific* @C:1001 *syntax for defining the coordinate offset of the sequence (which, for this parser to work, must follow the ID) may also be included. An example entry might look like:*

```
mysql> SELECT COUNT(*)
    +> FROM search_query
    +> INNER JOIN search USING (search_id)
    +> WHERE search.tag = "ecoli_vs_human";
+----------+
| COUNT(*) |
+----------+
|     4289 |
+----------+
1 row in set (0.00 sec)

mysql> SELECT query_id, lib_id, score, expect, percid
    +> FROM search_hit
    +> INNER JOIN search USING (search_id)
    +> WHERE search.tag = "ecoli_vs_human"
    +> LIMIT 5;
+----------+--------+-------+--------+--------+
| query_id | lib_id | score | expect | percid |
+----------+--------+-------+--------+--------+
|        1 |      1 |    23 |    9.4 |      1 |
|        2 |      2 |    32 |   0.81 |  0.253 |
|        2 |      3 |    33 |    2.3 |  0.299 |
|        2 |      4 |    30 |    2.4 |  0.291 |
|        2 |      5 |    32 |    2.4 |  0.252 |
+----------+--------+-------+--------+--------+
5 rows in set (0.00 sec)

mysql> SELECT COUNT(*)
    +> FROM search_hit
    +> INNER JOIN search USING (search_id)
    +> WHERE search.tag = "ecoli_vs_human"
    +> AND search_hit.expect <= 1e-3;
+----------+
| COUNT(*) |
+----------+
|     8588 |
+----------+
1 row in set (1 min 1.18 sec)
```

Figure 9.4.9 SQL statements to confirm successful importing of results. Bold text represents input; lightface text represents output.

```
>contig.contig_id|9876 @C:1001 Fragment of assembled
contig
ACTAGCTACGACTACGATCAGCGACTACGAGCGCGCATCGAC ...
```

Finally, `loadsearch.pl` *also assumes that if the report contains multiple results from multiple queries, then the same library database and parameters were used in all searches (i.e., the* `search` *table data remains constant, and the entire result set is considered as one search execution, with multiple independent queries). The script expects to receive the report via STDIN, and to obtain the name "tag" and any descriptive commentary via command-line arguments.*

Confirm successful result importing

5. Execute a few basic SQL statements to check that the data has been successfully imported into the database (Fig. 9.4.9).

The result shown in Figure 9.4.9 further exemplifies the need to store similarity results in a relational database: manually examining and evaluating over 8500 statistically significant alignments is simply not feasible.

ANALYZING SIMILARITY SEARCH RESULTS: IDENTIFYING ANCIENT PROTEINS

BASIC PROTOCOL 4

Once the data from sequence similarity searches are stored in a relational database, it becomes possible to build "genome-scale" summaries that incorporate data about thousands of sequences almost as easily as reporting results from one or two searches. Once one has saved all the results of a large-scale sequence comparison (e.g., all *E. coli* protein sequences used as queries in searches against a database of human protein sequences), comprehensive summaries of the similarities between the proteins in two genomes can be generated with a few SQL statements. To illustrate, the authors of this unit searched all 4,289 *E. coli* K12 predicted proteins against approximately 40,000 human sequences from the `nr` database that are also found in the curated human IPI database, and saved the results in a `seqdb_demo` database as `ecoli-vs-human`. It is then possible to identify ancient genes—genes shared by human and *E. coli*, presumed to be present in the last common ancestor of bacteria and man.

Necessary Resources

Hardware

Computer with at least 2 Gb of disk space available

Software

Windows- or Unix-based operating system (including Linux or Mac OS X)
Working version of MySQL, with functional database permissions. MySQL can be downloaded from *http://www.mysql.com* and installed as described in *UNIT 9.2*. All interactions with MySQL databases in these protocols will be via the `mysql` command-line utility.
A terminal application connected to a Unix environment in which one can execute Unix-like commands. For Windows-based operating systems, this entails installing the Cygwin Unix emulation (*http://www.cygwin.com*).
The Perl scripting language interpreter (any version since 5.005_03) and the `DBI`, and `DBD::mysql` modules.

Files

Generated as in Basic Protocols 1 and 3

1. Complete Basic Protocols 1 and 3.

```
+----------+-----------------------------------------+-----------+
| query_id | descr                                   |MIN(expect)|
+----------+-----------------------------------------+-----------+
|      935 | orf, hypothetical protein [Escherichi   |         0 |
|     4239 | host restriction; endonuclease R [Esc   |         0 |
|     2839 | glycine decarboxylase, P protein of g   |  2.7e-206 |
|     2852 | methylmalonyl-CoA mutase (MCM) [Esche   |  1.2e-176 |
|     3351 | glycogen phosphorylase [Escherichia c   |  3.8e-176 |
|     3913 | B12-dependent homocysteine-N5-methylt   |  9.9e-173 |
|       33 | carbamoyl-phosphate synthase large su   |  1.8e-165 |
|     3919 | glucosephosphate isomerase [Escherich   |  5.6e-159 |
|      542 | IS5 transposase [Escherichia coli]      |  2.7e-153 |
|      251 | IS5 transposase [Escherichia coli]      |  2.7e-153 |
+----------+-----------------------------------------+-----------+
```

Figure 9.4.10 List of highest-scoring *E. coli* homologs to human sequences, obtained via the commands shown in step 3 of Basic Protocol 4.

2. Once the search results are loaded (using `loadsearch.pl`, as described in Basic Protocol 3), a simple summary of the number of *E. coli* sequences that share significant similarity to human sequences can be produced:

```
mysql > SELECT COUNT(DISTINCT search_hit.query_id) AS
shared
-> FROM search_hit
-> INNER JOIN search USING (search_id)
-> WHERE search.tag = ''ecoli-vs-human''
-> AND expect < 1e-6;
```

This query returns a count of 926 *E. coli* sequences.

One could also ask the opposite question, how many human proteins have a significant match with E. coli, simply by changing the `DISTINCT query_id` *clause to* `DISTINCT lib_id`.

3. In addition to knowing the numbers of matches that obtain an *E* value less than `1e—6`, one might also like to identify the highest-scoring homologs. It is relatively easy to identify the *E. coli* sequences involved in the ten most significant (i.e., lowest *E* value) alignments between *E. coli* and human sequences:

```
mysql> SELECT search_hit.query_id, search_query.descr,
MIN(expect)
    -> FROM search
    -> INNER JOIN search_hit USING (search_id)
    -> INNER JOIN search_query USING (query_id)
    -> WHERE search.tag = ''ecoli-vs-human''
    -> GROUP BY query_id
    -> ORDER BY expect
    -> LIMIT 10;
```

To get the listing (Fig. 9.4.10) of *E. coli* sequences (rather than just the count), the `COUNT (DISTINCT search_hit.query_id)` clause from step 2 was replaced with a `GROUP BY query_id`; both statements ensure that *E. coli* proteins that match several human proteins will be counted only once.

```
SELECT  search_hit.query_id,  search_hit.expect,
        search_query.descr  AS  query_descr,
        search_lib.descr  AS  lib_descr
 FROM   search
        INNER  JOIN  search_hit  USING  (search_id)
        INNER  JOIN  search_query  USING  (query_id)
        INNER  JOIN  search_lib
            ON  (search_hit.lib_id  =  search_lib.lib_id)
 WHERE  expect  IN  (
        SELECT  MIN(besthit.expect)
        FROM    search_hit  AS  besthit
          WHERE   besthit.query_id  =  hit.query_id
    )
  AND  search.tag  =  "ecoli-vs-human"
ORDER BY expect ASC
LIMIT  10;
```

Figure 9.4.11 SQL statement to identify human sequences involved in alignments from from step 3 of Basic Protocol 4, for a database system that allows subselects (see step 4a of Basic Protocol 4).

4a. *For database systems that allow "subselects":* It is more difficult to identify the human sequences involved in each of these alignments because the `GROUP BY` clause used in step 3 means that all the rows from `search_hit` that share the same `query_id` have been collapsed; if one were also to request `search_hit.lib_id`, from which of the collapsed rows will the `lib_id` come? One might guess that the selected `lib_id` would be from the same row where the value of `expect` is equal to `MIN(expect)`, but, with SQL, *there is nothing that guarantees this to be true*. In a database system that allows "subselects" (SQL clauses that are themselves complete `SELECT` statements), one could instead do something like what is illustrated in Figure 9.4.11. Note that in this solution, multiple rows may be obtained for a given query, if the best hits happen to share the same expectation value (e.g., an `expect` of `0`).

4b. *For database systems that do not allow "subselects":* Versions of MySQL prior to 4.1 lacked "subselect" capability; getting the related hit information without subselects is a bit more complicated, but demonstrates a useful approach. A temporary intermediate table is first created to store the `hit_id` and `query_id` values for the rows of interest (i.e., the `hit_id` corresponding to the row or rows having `MIN(expect)` for each `query_id`). Because the aggregate functions `MIN` and `MAX` only operate on the first numeric value found in an entry, the trick to getting valid `hit_id`'s is to embed each `hit_id` in a string that also contains the numeric log-transformed *E* value, separated by white space. One can then extract the `hit_id` that corresponds to `MIN(expect)` [or `MAX(-LOG(expect)`, as the case may be] from the aggregate function's result (see `ancient.sql`, found in the `seqdb_demo` distribution), using the statement shown in Figure 9.4.12. The intermediate `besthits` table (Fig. 9.4.13) can now be used to retrieve only the rows of interest. For instance, the script shown in Figure 9.4.14 produces a list of the ten best matches between *E. coli* and human proteins, excluding any obvious transposase insertion sequences.

These SQL queries show that there are many very highly conserved proteins shared by both E. coli and humans; because these genes have shared ancestry, they must have been present in the last common ancestor of bacteria and humans.

```
DROP TABLE IF EXISTS besthits;
CREATE TEMPORARY TABLE besthits
SELECT query_id,
       SUBSTRING(
         MAX(CONCAT(RPAD(LPAD(
           FLOOR(IF(expect = 0,
                    10000,       -- special case for E value of 0
                    -LOG(expect) -- turn E value into whole number
                   ) * 10000
                 ), 9, " "), 30, " "), hit_id)),31
       ) AS hit_id
FROM   search
       INNER JOIN search_hit USING (search_id)
WHERE  search.tag = "ecoli-vs-human"
GROUP BY query_id;
```

Figure 9.4.12 SQL statement to identify human sequences involved in alignments from step 3 of Basic Protocol 4, for versions of MySQL that do not allow subselects (see step 4b of Basic Protocol 4).

E()	%ID	alen	query_descr	lib_descr
6.3e-207	53.3	953	glycine decarboxylase, P protein o	Glycine dehydroge
1.3e-177	59.3	708	methylmalonyl-CoA mutase (MCM)	Methylmalonyl-CoA
4.8e-177	50	812	glycogen phosphorylase	Glycogen phosphor
5.2e-173	54.2	1252	B12-dependent	5-methyltetrahydr
8e-166	41.2	1046	carbamoyl-phosphate synthase large	Carbamoyl-phospha
1.4e-160	64.8	549	glucosephosphate isomerase	Glucose-6-phospha
7e-144	52.9	867	aconitate hydrase 1	Iron-responsive e
5.2e-141	60.5	626	chaperone Hsp70; DNA biosynthesis;	Stress-70 protein
1.6e-135	71.7	466	membrane-bound ATP synthase, F1 se	ATP synthase beta
2.7e-121	55.4	554	succinate dehydrogenase, flavoprot	Succinate dehydro

Figure 9.4.13 Intermediate `besthits` table produced by SQL from Figure 9.4.14.

```
SELECT  expect  AS 'E()',
        percid*100 AS '%ID', alen,
        search_query.descr AS query_descr,
        search_lib.descr AS lib_descr
FROM  search
      INNER JOIN search_query USING (search_id)
      INNER JOIN besthits USING (query_id)
      INNER JOIN search_hit USING (hit_id)
      INNER JOIN search_lib USING (lib_id)
WHERE  search_query.descr NOT LIKE  "%IS5 transposase%"
AND  search_query.descr NOT LIKE "%IS2  hypo%"
AND  search.tag = "ecoli-vs-human"
ORDER  BY expect ASC
LIMIT 10;
```

Figure 9.4.14 Script used to produce a list of the ten best matches between *E. coli* and human proteins from the intermediate `besthits` table shown in Figure 9.4.13.

ANALYZING SIMILARITY SEARCH RESULTS: TAXONOMIC SUBSETS

One can generalize the genome-genome comparison from Basic Protocol 4 to determine a taxonomic distribution (i.e., the presence or absence in a given species or taxonomic clade) for any gene of interest. In this protocol, sequence similarity searches will be used against a database such as that described in Basic Protocol 1, where species information is available for each sequence. For any library sequence identified, it is possible to use the `seq_id` field from the `search_lib` table to look up `taxon_id` values from the `annot` table. The goal is to generate a summary table of gene counts that reflect various taxonomic subsets, i.e., the number of genes that have homology with proteins in *Bacteria*, *Archaea*, and *Eukaryota*, or only with proteins found in *Bacteria* (but not *Archaea* or *Eukaryota*), or only with proteins found in *Archaea*, or with proteins found in both *Bacteria* and *Archaea* but not *Eukaryota*, etc. Although the relational database concepts required to generate the summary table are a bit more complex than in the examples given elsewhere in this unit, which involve "joining" only a handful tables, the SQL shown in this protocol demonstrates how relational databases can provide summaries of datasets where the data must satisfy many conditions.

The data for this example come from a sequence similarity search of all 4289 *E. coli* K12 proteins against the entire NCBI `nr` database. The goal is to generate the necessary data to create a summary table, shown in Table 9.4.1. Note that this protocol is not intended to obtain knowledge about matches that occurred to other *E. coli* proteins already in the database, only to homologs in bacterial species other than *E. coli*. Thus, the last line in the table demonstrates that 355 *E. coli* proteins have no known homologs in any other species.

Necessary Resources

Hardware

Computer with at least 2 Gb of disk space available

Software

Windows- or Unix-based operating system (including Linux or Mac OS X)
Working version of MySQL, with functional database permissions. MySQL can be downloaded from *http://www.mysql.com* and installed as described in *UNIT 9.2*. All interactions with MySQL databases in these protocols will be via the `mysql` command-line utility.

Table 9.4.1 Taxonomic Distribution of *E. coli* Homologs

Eukaryota	Archaea	Bacteria	Totals
+	+	+	893
+	−	+	**661**
−	+	+	394
+	+	−	0
−	−	+	1986
−	+	−	0
+	−	−	0
−	−	−	355
1560	**1289**	3934	4289

```
CREATE TEMPORARY TABLE temp_results (
   query_id  INT UNSIGNED NOT NULL DEFAULT 0,
   taxon_id INT UNSIGNED NOT NULL DEFAULT 0,
   PRIMARY KEY (query_id, taxon_id)
) TYPE = HEAP;

INSERT INTO temp_results ( query_id, taxon_id )
SELECT search_query.seq_id, kingdom.taxon_id
FROM   search
        INNER JOIN search_query USING (search_id)
        INNER JOIN search_hit USING (query_id)
        INNER JOIN search_lib USING (lib_id)
        INNER JOIN annot ON (search_lib.seq_id = annot.gi)
        INNER JOIN taxon USING (taxon_id)

        INNER JOIN taxon AS kingdom
             ON (taxon.left_id BETWEEN kingdom.left_id
                                   AND kingdom.right_id)
        INNER JOIN taxon_name
             ON (kingdom.taxon_id = taxon_name.taxon_id)

        INNER JOIN taxon AS ecoli
             ON (taxon.left_id NOT BETWEEN ecoli.left_id
                                       AND ecoli.right_id)
        INNER JOIN taxon_name AS ecoli_name
             ON (ecoli.taxon_id = ecoli_name.taxon_id)

WHERE   taxon_name.name IN ('Bacteria', 'Eukaryota', 'Archaea')
  AND   taxon_name.class = 'scientific name'
  AND   ecoli_name.name = 'Escherichia coli'
  AND   ecoli_name.class = 'scientific name'
  AND   search.tag = 'eco-vs-nr'
;

Query OK, 7125 rows affected (2 min 18.64 sec)
Records: 565464  Duplicates: 558339  Warnings: 0
```

Figure 9.4.15 SQL statement used to create a temporary intermediate results table to store the `taxon_id` of all species in which a homolog to each query was found (see step 2 of Basic Protocol 5). Bold text represents input; lightface text represents output.

A terminal application connected to a Unix environment in which one can execute Unix-like commands. For Windows-based operating systems, this entails installing the Cygwin Unix emulation (*http://www.cygwin.com*).

The Perl scripting language interpreter (any version since 5.005_03) and the `DBI`, and `DBD::mysql` modules

Files

Generated as in Basic Protocols 1 and 3

1. Complete Basic Protocols 1 and 3.
2. Create a temporary intermediate results table to store the `taxon_id` of all species in which a homolog to each query was found, using the SQL statement shown in Figure 9.4.15 (see `taxcat.sql`, found in the `seqdb_demo` distribution). For efficiency, specify that the table should exist only in memory (remove the `TYPE=HEAP` clause if the results do not fit into available memory). Having built this `temp_result` table, it can now be used for every combination of desired taxonomic subsets.
3. To generate the counts for genes found in *Bacteria* and *Eukaryota*, but not *Archaea*, generate a second temporary table, `excludes`, which contains the `query_id`'s of homologs in the *undesired* taxonomic subsets, using the SQL statement shown in Figure 9.4.16.

```
DROP TABLE IF EXISTS excludes;
CREATE TEMPORARY TABLE excludes TYPE = HEAP
SELECT DISTINCT(query_id) AS query_id
FROM   temp_results
       INNER JOIN taxon USING (taxon_id)
       INNER JOIN taxon_name
           ON (taxon.taxon_id = taxon_name.taxon_id)
WHERE  taxon_name.name NOT IN ('Bacteria', 'Eukaryota')
  AND  taxon_name.class = 'scientific name';

Query OK, 1289 rows affected (0.42 sec)
Records: 1289  Duplicates: 0  Warnings: 0
```

Figure 9.4.16 SQL statement used to generate the temporary `excludes` table (see step 3 of Basic Protocol 5). Bold text represents input; lightface text represents output.

```
SELECT  COUNT(temp_results.query_id) AS total, annot.descr
FROM    temp_results
        INNER JOIN taxon USING (taxon_id)
        INNER JOIN taxon_name
            ON (taxon.taxon_id = taxon_name.taxon_id
        INNER JOIN annot
            ON (temp_results.query_id = annot.gi)
        LEFT JOIN exclude
            ON (temp_results.query_id = exclude.query_id)
WHERE   exclude.query_id IS NULL
  AND   taxon_name.name IN ('Bacteria', 'Eukaryota')
  AND   taxon_name.class = 'scientific name')
GROUP BY temp_results.query_id
HAVING  total = 2;
```

Figure 9.4.17 SQL statement used to select the count of rows in `temp_results` where the `query_id` appears, given the desired taxonomic subsets.

The `WHERE` *constraint in this query is equivalent to* `taxon_name.name = 'Archaea'`*; therefore the number of records inserted (*`1289`*) is the total number of E. coli proteins that have homologs in Archaea (regardless of what other homologies there may be). These are the source of the column totals found at the bottom of the summary table.*

4. For each `query_id` not in `excludes`, select the *count* of rows in `temp_results` where the `query_id` appears, given the desired taxonomic subsets, using the SQL statement shown in Figure 9.4.17. If that count equals the number of taxonomic subsets, then that `query_id` satisfies the condition (note the `HAVING` clause that enforces this behavior).

   The number of rows that this query returns (`661`*; Fig. 9.4.18) is the number of genes that have hits against proteins in both Bacteria and Eukaryota species, but have no significant hits against proteins from Archaea species (the +/−/+ row in Table 9.4.1). Also, by joining the results back to the* `annot` *table, it is possible to see which genes have this taxonomic distribution.*

5. Repeat steps 3 and 4 for each taxonomic combination of interest (changing only the names of the taxa to include, and the `HAVING` clause to reflect the number of taxa) to generate the summary table. Note that the last combination (—/—/—) denotes *E. coli* proteins that did not align against any other protein sequence; the value for that row (`355`) is the difference between the total number of *E. coli* proteins used in the search (`4289`) and the sum of all the other totals.

```
+-------+-------------------------------------------------------------------------------+
| total | descr                                                                         |
+-------+-------------------------------------------------------------------------------+
|     2 | (3R)-hydroxymyristol acyl carrier protein dehydratase                         |
|     2 | (p)ppGpp synthetase I (GTP pyrophosphokinase) reg of RNA synth; stringent     |
|     2 | (p)ppGpp synthetase II; G-3',5'-bis pyrophosphate 3'-pyrophosphohydrolase     |
|     2 | 1-acyl-sn-glycerol-3-phosphate acyltransferase                                |
|     2 | 10-formyltetrahydrofolate:L-methionyl-tRNA(fMet) N-formyltransferase          |

                                    ...

|     2 | uroporphyrinogen III synthase                                                 |
|     2 | vitamin B12 transport                                                         |
|     2 | [2FE-2S] ferredoxin, electron carrier protein                                 |
+-------+-------------------------------------------------------------------------------+
661 rows in set (5.74 sec)
```

Figure 9.4.18 Table returned by the query in Figure 9.4.17. The number of rows that this query returns (`661`) is the number of genes that have hits against proteins in both *Bacteria* and *Eukaryota* species, but have no significant hits against proteins from Archaea species.

BASIC PROTOCOL 6

ANALYZING SIMILARITY SEARCH RESULTS: INTERGENIC REGIONS

While ab initio gene prediction is difficult in eukaryotes (and can be difficult for prokaryotes with sequencing errors), many genes are easily identifiable by homology to known protein sequences. However, comparing complete genomic DNA sequences against the entire `nr` protein database is time consuming. Gene finding by homology can be much more efficient if one only searches against protein sequences from closely related organisms. Having identified the "low-hanging fruit," remaining stretches of intergenic sequence can be searched against a larger database. This approach is both more sensitive and faster, because a smaller database is used in the initial search, and fewer comparisons are made overall. Here, a two-step search strategy will be described, which could also be extended over multiple iterations using subsequent nested taxonomic subsets.

First, a taxonomic subset of proteins are selected that share homology with most of the genes in the target organism. For example, to identify genes in *E. coli*, one might search against the approximately 45,000 proteins from the parental family *Enterobacteriaceae*. The choice depends on the evolutionary distance to organisms with comprehensive protein data: for the puffer fish (*Fugu rubripes*), the parent order *Tetraodontiformes* includes only about 700 protein sequences; the parent class *Actinopterygii* (ray-finned fishes) includes approximately 16,000 protein sequences, while the parent superclass *Gnathostomata* (jawed vertebrates) 330,000 proteins; however, species from across the superclass have diverged over 500 million years of evolution, and these may be difficult to identify. Next, the genomic DNA would be compared to the chosen taxonomic subset of protein sequences (using a DNA-translating search algorithm—e.g., BLASTX (*UNIT 3.3*) or FASTX (*UNIT 3.9*)—and the search results would be stored in `seqdb_demo`. Then, the next step in this process would be to identify the unmatched regions of "intergenic" DNA sequence—i.e., subregions of `search_query` entries that did not produce a significant alignment, and use only these regions to search a more complete protein set. This protocol demonstrates how to produce intergenic regions from prior search results, using *S. typhimurium* (STM) sequences searched against *E. coli* (ECO) proteins.

While the process of searching a new sequence library with unmatched DNA sequences is easy to conceptualize, identifying those sequences requires several steps. Importantly, the approach illustrated here assumes a bacterial or archaeal genome without introns—i.e., any sequence-similarity hit can be considered a gene and any unmatched DNA as

intergenic (and not intronic). However, the same technique could be used in eukaryotes, but only after exon-based alignments have been assembled into complete gene models and the ranges of those gene models saved as search hits in the database.

Necessary Resources

Hardware

Computer with at least 2 Gb of disk space available

Software

Windows- or Unix-based operating system (including Linux or Mac OS X)

Working version of MySQL, with functional database permissions. MySQL can be downloaded from *http://www.mysql.com* and installed as described in *UNIT 9.2*. All interactions with MySQL databases in these protocols will be via the `mysql` command-line utility.

A terminal application connected to a Unix environment in which one can execute Unix-like commands. For Windows-based operating systems, this entails installing the Cygwin Unix emulation (*http://www.cygwin.com*).

The Perl scripting language interpreter (any version since 5.005_03) and the `DBI`, and `DBD::mysql` modules.

Files

Generated as in Basic Protocols 1 and 3

1. Complete Basic Protocols 1 and 3.

   A sample set of similarity results is available from ftp://ftp.virginia.edu/pub/fasta/CPB/stm_eco.results.gz. This file must be uncompressed with the command `gunzip stm_eco.results.gz` *to produce the file* `stm_eco.results`*, which can then be loaded into the database with the* `loadsearch.pl` *command.*

2. Build a temporary table that contains the ranges of the successful hits using the SQL statement shown in Figure 9.4.19 (see `ranges.sql`, found in the `seqdb_demo` distribution). Note that it is not possible to declare this table as `TEMPORARY` because it is later going to be joined against itself).

3. For each set of hits *A* that have the same beginning on the same DNA sequence, pair them with all hits *B* on the same DNA sequence that begin after any of the *A* hits end. Take the max of the endings of *A* as the beginning of an intergenic range; from all the *B*'s, choose the smallest begin as the end of the intergenic range. Use the SQL statement shown in Figure 9.4.20.

```
CREATE  TABLE  hitranges  TYPE=HEAP
SELECT  search_query.seq_id AS query_id,
        IF(strand = 'f', qbeg, qend) AS begin,
        IF(strand = 'f', qend, qbeg) AS end
FROM    search
        INNER JOIN search_hit USING (search_id)
        INNER JOIN search_query USING (query_id)
WHERE   search.tag = 'stm-vs-eco'
  AND   search_hit.expect <= 1e-10;
```

Figure 9.4.19 SQL statement used to build a temporary table that contains the ranges of the successful hits, used in step 2 of Basic Protocol 6.

```
CREATE  TEMPORARY TABLE igranges TYPE=HEAP
SELECT  A.query_id AS query_id,
        MAX(A.end + 1) AS begin,
        MIN(B.begin - 1) AS end
FROM    hitranges AS A
        INNER JOIN hitranges AS B
          ON (A.query_id = B.query_id AND B.begin > A.end)
GROUP BY A.query_id, A.begin;
```

Figure 9.4.20 SQL statement used in step 3 of Basic Protocol 6, which contains an initial set of intergenic ranges for each `query_id`.

```
INSERT INTO igranges (query_id, begin, end)
SELECT query_id AS query_id,
       1 AS begin,
       MIN(begin - 1) AS end
FROM   hitranges
GROUP BY query_id
HAVING end >= begin;

INSERT INTO igranges (query_id, begin, end)
SELECT hitranges.query_id AS query_id,
       MAX(hitranges.end + 1) AS begin,
       search_query.len AS end
FROM   hitranges
       INNER JOIN search_query
           ON (hitranges.query_id = search_query.seq_id)
GROUP BY hitranges.query_id
HAVING end >= begin;
```

Figure 9.4.21 Two SQL statements used for adding the missed classes of beginning and ending "intergenic" DNA sequence to the `igranges` table (see step 4 of Basic Protocol 6).

4. The `SELECT` statement used in step 2 (Fig. 9.4.20) missed two important classes of "intergenic" DNA sequence: the range from the beginning of the DNA sequence to the first hit, and the range from the last hit to the end of the DNA sequence. The two SQL statements in Figure 9.4.21 add those ranges to the `igranges` table.

5. Finally, it is desirable to add any DNA sequence queries that did not match against anything (and thus have no rows in the `hitranges` table), using the SQL statement in Figure 9.4.22. This must be done in two steps because it is not possible to simultaneously `SELECT` from a table into which one is also `INSERT`-ing.

6. What remains is to clean the `igranges` table of a few sources of artifactually overlapping ranges. The first is caused when a collection of hits look like the diagram shown in Figure 9.4.23A, leading to two `igrange`'s as shown in Figure 9.4.23B. Only the lowermost `igrange`, marked by the arrow, is desired. The unwanted longer range is removed by grouping the `igrange`'s on `end` and selecting `MAX(begin)` as the new boundary:

```
CREATE TEMPORARY TABLE clean_igranges TYPE=HEAP
SELECT query_id, MAX(begin) AS begin, end
FROM igranges
GROUP BY query_id, end;
```

```
CREATE  TEMPORARY  TABLE missing  TYPE=HEAP
SELECT  search_query.seq_id  AS  query_id,
        1 AS begin,
        search_query.len AS end
FROM    search_query
        LEFT JOIN hitranges
            ON (search_query.seq_id = hitranges.query_id)
WHERE   hitranges.query_id IS NULL;

INSERT  INTO igranges (query_id, begin, end)
SELECT  query_id, begin, end
FROM    missing;
```

Figure 9.4.22 SQL statement to add any DNA sequence queries that did not match against anything and that have no rows in the `hitranges` table (see step 5 of Basic Protocol 6).

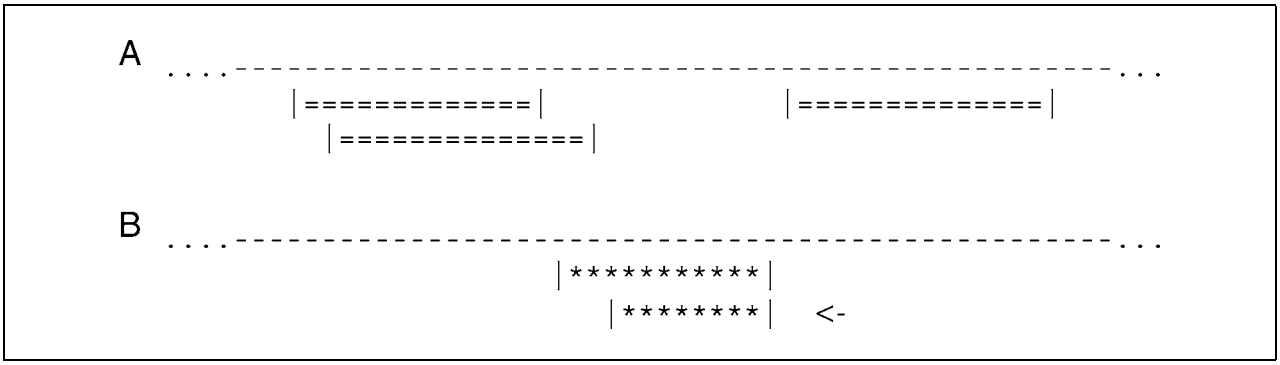

Figure 9.4.23 Schematic illustration of one possible source of artifactually overlapping ranges; the collection of hits in (**A**) lead to two `igrange`'s as shown in (**B**). Only the lowermost `igrange`, marked by the caret, is desired. See step 6 of Basic Protocol 6.

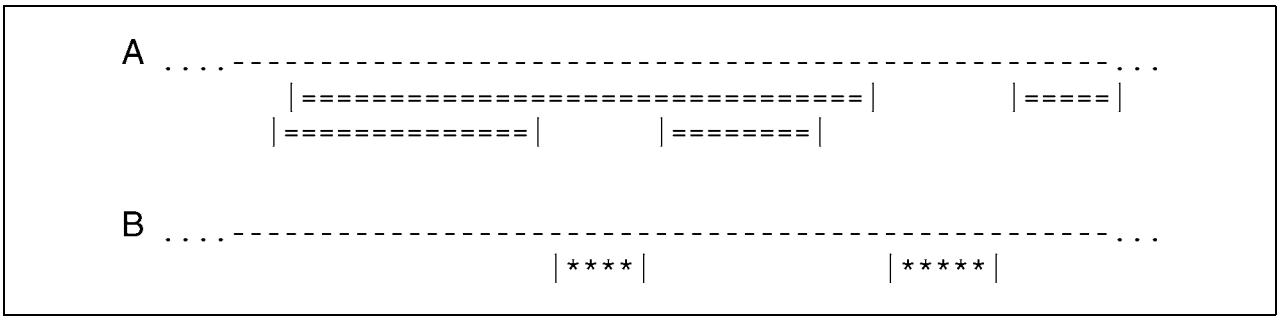

Figure 9.4.24 Schematic illustration of a second possible source of artifactually overlapping ranges: (**A**) the begin and end of two small hits are spanned by a third, larger hit, leading to the ranges shown in (**B**).

7. The second set of artifactual overlap ranges stems from hits where the begin and end of two small hits are spanned by a third, larger hit as shown in Figure 9.4.24A, leading to the ranges shown in Figure 9.4.24B. The unwanted ranges are eliminated by checking to see if any of the ranges overlap within the original set of hits using the SQL statement in Figure 9.4.25; any that do are not selected into the final set of intergenic ranges.

   The `final_igranges` *table now contains the intergenic regions. These regions could be used as the basis for queries in a subsequent search of a larger taxonomic subset of protein sequences; the above process can then be repeated for each new subset of intergenic regions.*

```
CREATE TEMPORARY TABLE overlaps TYPE=HEAP
SELECT DISTINCT clean_igranges.query_id,
                clean_igranges.begin, clean_igranges.end
FROM   clean_igranges
       INNER JOIN hitranges USING (query_id)
WHERE  clean_igranges.end <= hitranges.end
  AND  clean_igranges.begin >= hitranges.begin;

CREATE TEMPORARY TABLE final_igranges (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
  query_id INT UNSIGNED NOT NULL DEFAULT 0,
  begin INT UNSIGNED NOT NULL DEFAULT 0,
  end INT UNSIGNED NOT NULL DEFAULT 0
);

INSERT INTO final_igranges (query_id, begin, end)
SELECT clean_igranges.query_id, clean_igranges.begin,
clean_igranges.end
FROM   clean_igranges
       LEFT JOIN overlaps USING (query_id, begin, end)
WHERE  overlaps.query_id IS NULL;
```

Figure 9.4.25 SQL statement for eliminating unwanted ranges from the final set of intergenic ranges.

COMMENTARY

Background Information

Relational databases provide a powerful and flexible foundation for large-scale sequence comparison, and make it much easier to implement the "management controls" necessary to keep track of sequences, alignment positions, and scores. The `seqdb_demo` database and the accompanying Basic Protocols in this unit are meant to serve as examples of the many ways that relational databases can simplify genome-scale analyses in an investigator's research.

These protocols use relational databases and SQL to provide comprehensive summaries of large-scale sequence comparisons. To provide relatively compact examples, the authors have focused on evolutionary analyses, e.g., the number of homologs that are shared between different taxonomic classes. The power of relational approaches greatly increases as additional data are added to the database. In addition to sequence and taxonomic data, relational databases can store information about protein families and domains (e.g., PFAM) or protein functional assignments (the Gene Ontology or GO classification). Relational databases are particularly powerful when they are used to associate different kinds of data; for example, one might ask how often homologous proteins (proteins that share statistically significant similarity) are distant in the GO hierarchy and thus are likely to have different functions. As biological databases become more diverse, including not only sequence data but also genome locations, biological function, interaction results, and biological pathways, SQL databases provide powerful tools for exploring relationships between very different sets of biological data on a genome scale.

Literature Cited

Celko, J. 1999. Joe Celko's SQL for Smarties. Morgan Kaufmann, San Francisco.

Internet Resources

ftp://ftp.ncbi.nih.gov/pub/blast/db/FASTA/nr.gz

Comprehensive `nr` database (flat file protein sequence database).

ftp://ftp.ncbi.nih.gov/pub/blast/db/FASTA/swissprot.gz

SwissProt protein database (flat file protein sequence database).

ftp://ftp.pir.georgetown.edu.pir_databases/psd/mysql/

The Protein Identification Resource (PIR) at Georgetown University, which distributes the PIR protein database in relational format for the MySQL database program.

Contributed by Aaron J. Mackey and William R. Pearson
University of Virginia
Charlottesville, Virginia