MSc Bioinformatics with Systems Biology
Birkbeck College, University of London

Biocomputing II Coursework:

Genbank group project

Reflective essay

Georgina R. Toye

15/05/2019

1. Approach to the project

The team to which I was appointed consisted of Sam D., Diego and myself.

Initially, the team members had a quick chat concerning which roles would be taken on by each person. I asked if anyone had a particular preference in mind, and Sam made it known that he was very keen to take on the BL. Diego seemed noncommittal at first, but immediately agreed that Sam should take the BL role. I then explained that I felt I had learned a lot from the Database module, but had had very little exposure to webpage design and, consequently, that I felt it would benefit me greatly if I took on a project in that area. Diego and Sam then made clear that I would be doing the DB.

Overall project requirements & interactions with the team
The biggest stumbling block for the group, in my opinion, was in deciding the exact nature of the data which should be provided and received by the DB-API and, in particular, how this data and the requests for it should be modelled in real time. During the group presentation it had been made clear that this was the key issue which should be sorted out early and unambiguously.

It took me a while to study what data might reasonably be required by the BL and the FE, and how best that could be provided by the DB, given that the data first had to be extracted from the Genbank records, and that some processing of data was required. I studied the project documents and the Genbank record format in detail and thought this over, trying to avoid redundancy of data transfer.

At the second team meeting, and based on the above, I described the model of user interaction I perceived, where the user would likely require for there to be at least two distinct cycles of information exchange between the FE and the DB via the BL. This would be regardless of whether or not the user was entering a search term. However, this concept was entirely rejected by the others: indeed, the possibility of any kind of modelling regarding user-database interactions was fully shut down. As strongly recommended, I also requested for us to clearly define the parameters passed back and forth between the three layers by the APIs under different conditions. The response was vague, but broadly, “Everything referred to in the specification”, and I was even sent a photo of the specification with all of these highlighted. In order to clarify this, I subsequently sent a word document attached by email to the team members describing in detail my take on exactly which parameters might be made available by the DB layer to the BL. I included the following*:

Accession number
Chromosomal location
Gene identifier
Protein product name
Locus DNA sequence
Gene span (yields complete DNA sequence of the genetic unit via reference to locus DNA sequence)
CDS span (together with gene span, this might be useful for the restriction enzyme task)
Component CDS element spans (together yield CDS, and thus amino acid sequence, via reference to locus DNA sequence)
*Start codon was a later addition to this list when I realized it was necessary

I would have expected clarification on which of these data types should be made available under which particular circumstances e.g. during an initial string search; when requesting detailed information on one record etc. However, this email was ignored by both, as were all subsequent requests. The subject was never clarified beyond “everything in all cases”. It was not possible to glean the information indirectly from the BL, since the first direct reference to the database API in the BL layer coding was the day before the deadline: this was despite the API having been made available on Github several weeks earlier, together with detailed documentation.

The other team members were of the opinion that if multiple and detailed results pages came up following an initial single search term then that would be fine, and a more targeted follow-up search would be surplus to requirements. In fact, the other team members did not see any point in withholding data at all, and specifically wanted **all** the above data types on **each** record returned from an initial non-unique search to be made available immediately and simultaneously via a **single** request to the database, even if there were several records returned and several screenfuls of data to scroll through. I began to wonder if I was the only member of the team to appreciate the value of a RDMS in the first place. I cannot say for sure whether my viewpoint was not understood or not entertained. In any case it was rejected vehemently without explanation and, in order to avoid discord, I had no alternative but to acquiesce to the others' wishes.

I had already written some code based on a two- or three-step approach, whereby a non-unique search string e.g. a chromosome locus might be entered by the user initially, and then, after brief details of candidate records had been displayed (i.e. accession number, chromosomal location, gene ID and protein product), the user would select the best fit, and subsequently receive a full details page back for that one gene. A similar principle could work without a string being entered, with one extra step. For example, all chromosomal locations could first be called, then one selected after brief details of candidate records had been displayed (i.e. all chromosomal locations or, possibly, a table containing all permutations of accession number/chromosomal location/gene ID/protein product), the user would again select the best fit, and subsequently receive a full details page back for that one gene.

However, the other team members wanted all information returned to the BL following the very first query, which my initial two-step approach could not handle, so I had to redesign the API as a single function encompassing all such possibilities.

Requirements for my contribution
I initially arranged the concepts on a UML logical diagram, as shown in Figure 1. The diagram considers the Genbank database itself, rather than the requirements of the project e.g. there is a many to many relationship between Genbank sequence record and Protein coding gene: this is since some entries have several different genes along their locus length. This also means that an Accession number is not necessarily unique to a single gene and, further, that a composite key was required for most tables in the logical diagram as indicated.

After deducing the exact nature of the project requirements, it seemed that a much more straightforward physical model was in order (Figure 2). For example, only one isoform of each gene product was to be included. All relationships became essentially one to one after such considerations. Looking at the nature of the data, however, the extremely large size of the DNA sequences meant that I was inclined to store them in a separate table (as MEDIUMTEXT): this was not based on anything concrete, it just seemed appropriate.

A further point of importance for the physical model tables was the issue of the CDS element spans. Grouped together as a single string, these were also quite long, but noticeably absent from most records. Therefore, I decided to put them in a separate table to avoid numerous NULL cells. I did consider dividing the CDS element span groups into individual span columns, but I couldn't really see how this would differ from dividing sets of poems into columns with each column being one line: better to store the whole set together as there would never be any mixing and matching. I'm still not sure if this was the correct decision. (The DNA sequence and CDS element spans do look very untidy when selected and displayed in a MySQL, table, but I'm sure there must be some way round this).

By contrast, I decided to populate each and every cell of the main table by using 'NF' (not found) if necessary for missing fields. I designed all my parsing code to provide a definitive return statement for each and every record, even if only 'NF'. I considered this to be an important aspect of quality control, particularly if dealing with a larger chromosome, so as to avoid getting data arising from different genes mixed up. All fields therefore yielded 111 record outputs (one entry was later discarded, as described below). As a general rule, in the DB-API, it was intended to supply all lists of results with an accompanying list of accession numbers in order to provide a unique label for data arising from each record.

Logical schema

Figure 1. Logical schema describing relationships between selected Genbank data fields describing protein coding genes.
AA = amino acid; CDS = coding DNA sequence; PK = Primary Key; FK = Foreign Key

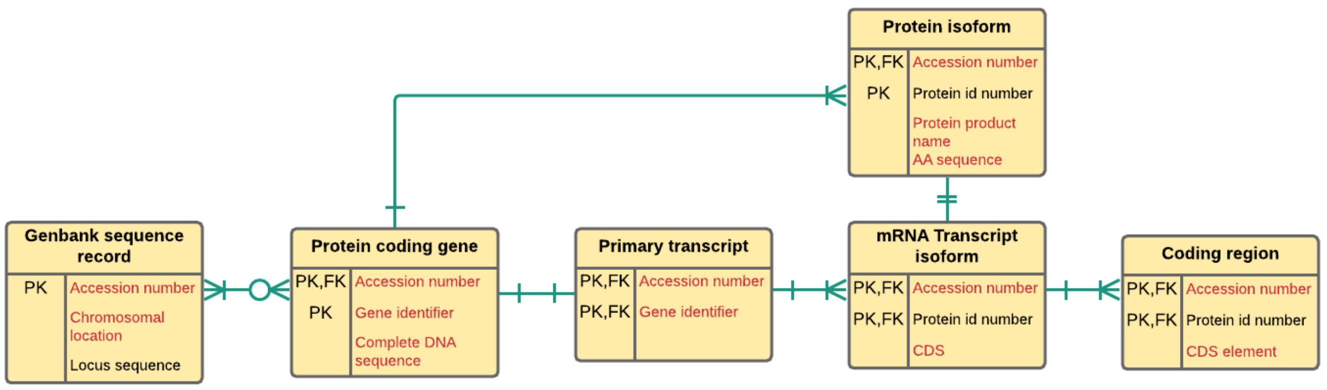

Figure 1. Logical schema.

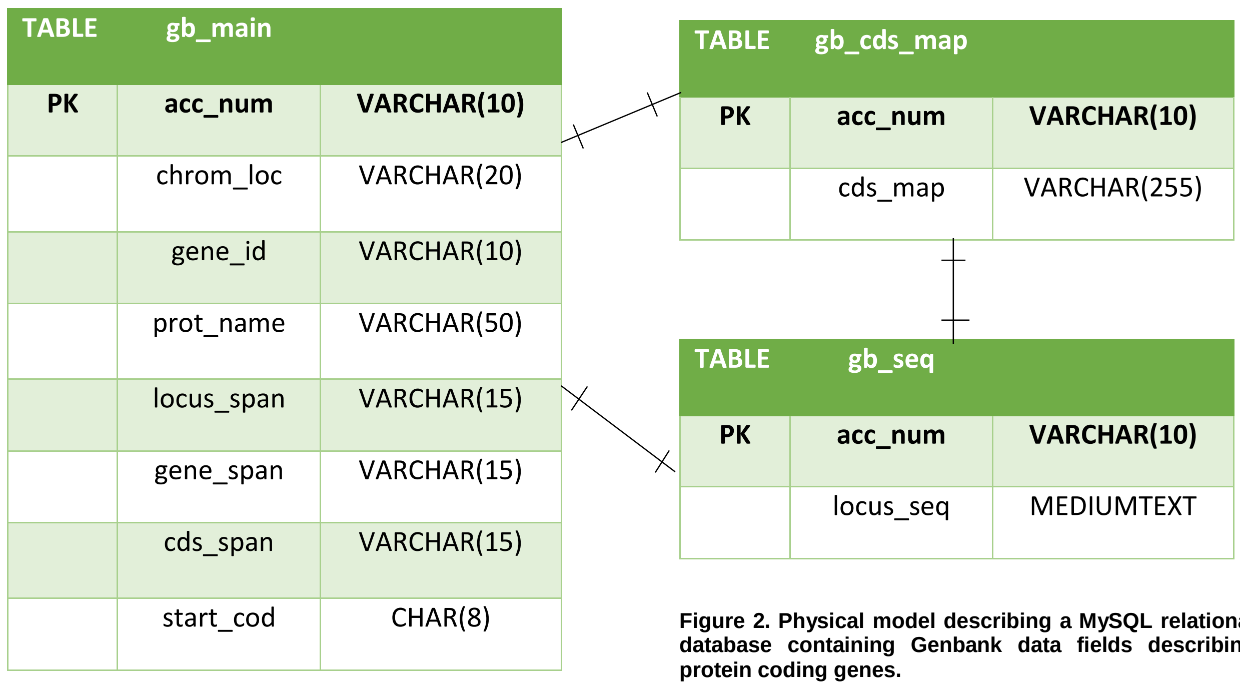

TABLE	gb_main	
PK	**acc_num**	**VARCHAR(10)**
	chrom_loc	VARCHAR(20)
	gene_id	VARCHAR(10)
	prot_name	VARCHAR(50)
	locus_span	VARCHAR(15)
	gene_span	VARCHAR(15)
	cds_span	VARCHAR(15)
	start_cod	CHAR(8)

TABLE	gb_cds_map	
PK	**acc_num**	**VARCHAR(10)**
	cds_map	VARCHAR(255)

TABLE	gb_seq	
PK	**acc_num**	**VARCHAR(10)**
	locus_seq	MEDIUMTEXT

Figure 2. Physical model describing a MySQL relational database containing Genbank data fields describing protein coding genes.

CDS = coding DNA sequence; acc_no = accession number; chrom_loc = chromosomal location; gene_id = name of gene feature; locus_seq = Genbank record DNA sequence; gene_span = gene sequence span; cds_span = CDS span; cds_map = list of CDS elements comprising CDS span; prot_name; PK = Primary Key; FK = Foreign Key

Sequence spans are expressed as a range of base numbers.

Figure 2. Physical schema.

2. Performance of the development cycle

There was little communication between the group members after the end of term, although I made sure to notify the others whenever I changed something important. I regularly updated my coding documents to enable the others to see exactly what was being provided by the DB-API and in what form. Sam, in particular, seemed reluctant to use Github, which, again, made it difficult to know what was expected from the DB layer.

We had a face-to-face meeting at Birkbeck the day before the project was due in, which was very useful as we organized the files on Github together, and Diego explained to me about configuration files, which I hadn't known anything about before. Unfortunately, however, I didn't get round to incorporating these into my own programs.

3. The development process

This was simply a case of myself taking the project a step at a time and reviewing progress as I went along, frequently checking my code for appropriate functionality. I spent a lot of time working out how the dummy API (and, consequently, the actual API) could work in terms of providing information to the BL in the way in which the other team members required, whilst also allowing the program to be used in a two- or three-step fashion as I had originally envisaged. I was keen to ensure that the dummy version would serve to be fairly realistic for both the user and the BL, so I wouldn't affect the progress of the other team members. However, the dummy DB-API was not accessed until the day before the deadline. I planned to make the actual DB-API deliver exactly the same form of data as the dummy version, so I didn't see why Sam had been reluctant to use it.

Once the dummy API was tested and ready, I turned to the parsing task. This was because it had been highlighted as being a very long and arduous process. I did indeed spend a long time on this and found it to be very subtle, complicated and unforgiving. I carried out a lot of ad hoc testing with the parsing stage, to ensure resilient retrieval of appropriate data. I carried this out using a short Genbank file containing twelve records. I made sure these records were eclectic e.g. I included a record that had three different genes in it and made sure only the first gene and the first gene's details were extracted. Next, I designed the DB itself, in order to populate it with the parsed data.

As my final programming task, I tackled the critical issue of using PyMySQL to provide wrappers for the MySQL queries to access the database: this sounded extremely complicated to me, having missed that lecture. However, using the relevant bioinf website pages, the PyMySQL turned out to be relatively simple and straightforward as a method of accessing the content stored in the database.

Since the requirements of the BL were not available the complication arose with the sheer number of combinations possible with the DB-API. This meant that I became bogged down and weary of translating repetitive sections of coding from the Dummy DB-API into appropriate MySQL request statements without knowledge of which would actually be required. The final list of MySQL queries requiring wrappers is quite long, although fewer would suffice, probably 12. Unfortunately, the deadline was not met by my DB layer.

4. Code testing

I was very committed to testing my programs, once I had got them more or less working. For example, I tried a multiplicity of possible inputs for the dummy DB-API and scrutinized the results. Importantly, with the parsing code, I counted the number of results automatically, so that I could be aware when I had exactly 111 outputs of a particular data type. I made my regexes more, and then less exacting, to ensure they were capturing only the correct examples that I had based my design upon. Once the MySQL tables were populated, it was easy to analyze the data for errors e.g. I could select and examine the unique instances of variables in each column of a table.

I would have liked to have tested the code on the longest Chromosome, but didn't get round to this, unfortunately.

5. Known issues

All known issues are noted at the bottom of each code. This helped to remind me of what I could do next to improve my code, and I did fix a lot of bugs, often with the help of StackOverflow, which proved invaluable. There are still plenty of issues which need sorting out, such as making the code pick out more of the component CDS element spans. They are particularly difficult to parse since different scientists upload these details in different ways. I think I might have misinterpreted the meaning of "complete DNA sequence": I initially took it to mean complete locus, but later could see it meant complete gene. In retrospect, it might have made more sense to have extracted only the actual gene code before passing the sequence to the BL.

The main known issue, of course, is that the DB-API function was not ready by the deadline. I feel very bad about this both personally and on behalf of my team. I did work extremely hard on this project and did do my best. On the other hand, I suspect using one function rather than two made my task take a little longer than it otherwise would have. In addition, I was not informed as to which MySQL statements would be necessary, so had to second guess the BL.

6. What worked and what didn't - problems and solutions

What definitely worked was leaving a problem alone as the answer would naturally come when returning to a problem with fresh eyes. Becoming obsessive about overcoming a particular problem is something I should avoid in future: as stated above, solutions can come more easily if you put some space between you and your code, and it makes more sense to get on with another aspect of the task. For example, I spent fully two days trying to load the MySQL tables automatically using "load data [local] infile…" etc., but couldn't get it to work. I should have given up and loaded it manually sooner - after all we did have the shortest chromosome, and I found you can load multiple rows into MySQL, which is very efficient.

There was one DNA sequence entry that the MySQL database seemed to refuse to accept. It relatedly caused the database to crash back to hope whenever I tried to enter it. I didn't resolve this issue and had to omit it entirely. It was very large, but I had assumed MEDIUMTEXT could have coped with it.

7. Alternative strategies

I also learnt that sometimes problems in programming are best solved just by finding an alternative route to the same outcome rather than pushing against a wall that won't give, as you can waste a lot of time that way. It really doesn't matter why the original method didn't work once the new one does.

I definitely need to work on effective, assertive communication. However, I'm not sure the other team members were ever ready to engage with me and I remain uncertain as to why that might be.

.

8. Personal insights

I am very much enjoying the course and the challenges it has involved. I have learned an enormous amount from this particular project, all of it extremely valuable, and it has greatly promoted my programming development.

Georgina R Toye   *15/05/2019*