MSc Bioinformatics with Systems Biology
Birkbeck College, University of London

Biocomputing II Coursework:

Genbank group project

Reflective essay

Georgina R. Toye

10/05/2019

Approach to the project

The team to which I was appointed consisted of Sam D., Diego and myself. We had not had the opportunity to interact much before that point.

Initially, the three team members had a quick chat concerning which roles would be taken on by each team member. 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. I was glad to hear this, as I had little idea about what the BL would do. Diego was less committal initially as to what role he wanted, but he agreed that Sam should take the BL role. So I explained that I felt I had learned a lot on the Database module, but had had very little exposure to Webpage design and, consequently, that I felt that it would benefit me greatly if I took on a project in that area. That was my preference at the time: learning something new. Diego and Sam then made it clear that I would be doing the DB. This was actually fine with me as I thought I would probably find it a lot easier than learning about webpages at that point in time, after all, there are always other opportunities.

Interaction with the team

Interaction between the team members was fairly minimal. We initially had a couple of brief meetings, during which it became clear that my DB end and Diego's FE were at cross-purposes. Diego had already parsed the Genbank data, which did confuse me a little. At this same meeting I raised for the second time that the model of user interaction which I perceived was one in which the user would likely require for there to be two exchanges of information between the FE and the DB via the BL. This concept was entirely rejected by both of the others, yet it still makes sense to me.

I sent several emails early on to which I attached my most recent database API coding, together with important questions requesting opinions on the exact interpretation of the task. The DB API requirements, in terms of exactly what information could be provided by the API, and what was actually required by the BL and FE, were still very wooly, and I was still hoping that a two-stage model of interaction would be agreed upon or, at least, tolerated.

Once I had got the hang of GitHub, and after we had in fact settled on a one-step process, I regularly updated my coding documents to enable the others to see exactly what was being provided by the database API and in what form. The others seemed reluctant to use Github or emails but both did upload some programs when I encouraged them to do so. The preferred method of communication appeared to be Whatsapp, which is fine, but it can lack detail and, to me, is unsuited to formal discussions of logic. I do, however, understand that there is a generational difference in perception and familiarity at play here.

Thus, 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 Lecture session, it had become plain and clear that this was the key issue which should be sorted out early and in an unambiguous manner. Whilst I grasped that important concept after the lecture, it did still take me a while to study what data was 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, making the observation that the DB would have to be called twice in order to guarantee that a unique record could be displayed as a "details" page.

As already mentioned, when I discussed this with the other group members, they were consistently clear and unequivocal that this was not actually the case: 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 wanted **all** data on **each** record to be returned by an initial search. I certainly did not agree with them. I had already written some code based on a two-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 (this would contain, for example, information derived by the BL from the locus sequence.

The other team members remained adamant that all data types including DNA sequence should immediately be made available via a single request to the database, even if there were several records returned and several screenfuls of data to scroll through. I could see that I would have to go along with this to avoid discord, and thus was born my rather strange and convoluted database API function.

Overall project requirements

The remit was explained in clear detail in the project documents, though there is always some ambiguity to be considered and discussed where biological data is concerned.

Requirements for my contribution

The other members mentioned that I should not consider storing or sending over any amino acid data, as that could be gleaned from the DNA code of the CDS region. This made sense and gave me a little less to worry about. I decided upon which pieces of data were required by the BL in order to meet the project requirements, and sent a detailed memo to the others for their opinion and/or approval. I included the following:

Accession number
Chromosomal location
Gene identifier
Protein product name
Locus DNA sequence (may be very long)
Gene span (-> complete gene DNA sequence)
CDS span (together with gene span, this would be useful for the restriction enzyme task)
Component CDS element spans (-> CDS DNA sequence and amino acid sequence)

I then 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 shown.

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 all this was considered. Looking at the nature of the data, however, the extremely large size of the DNA sequence data meant that I was inclined to store it in a separate table as (MEDTEXT): 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 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 vary untidy when selected from in MySQL, but I'm sure there must be some way round this.)

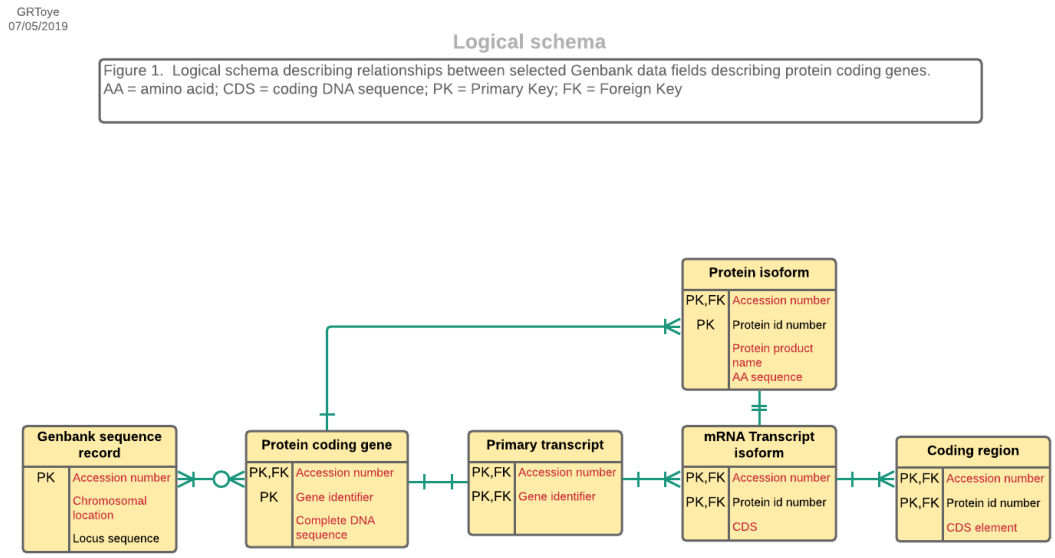

Figure 1 . Logical schema.

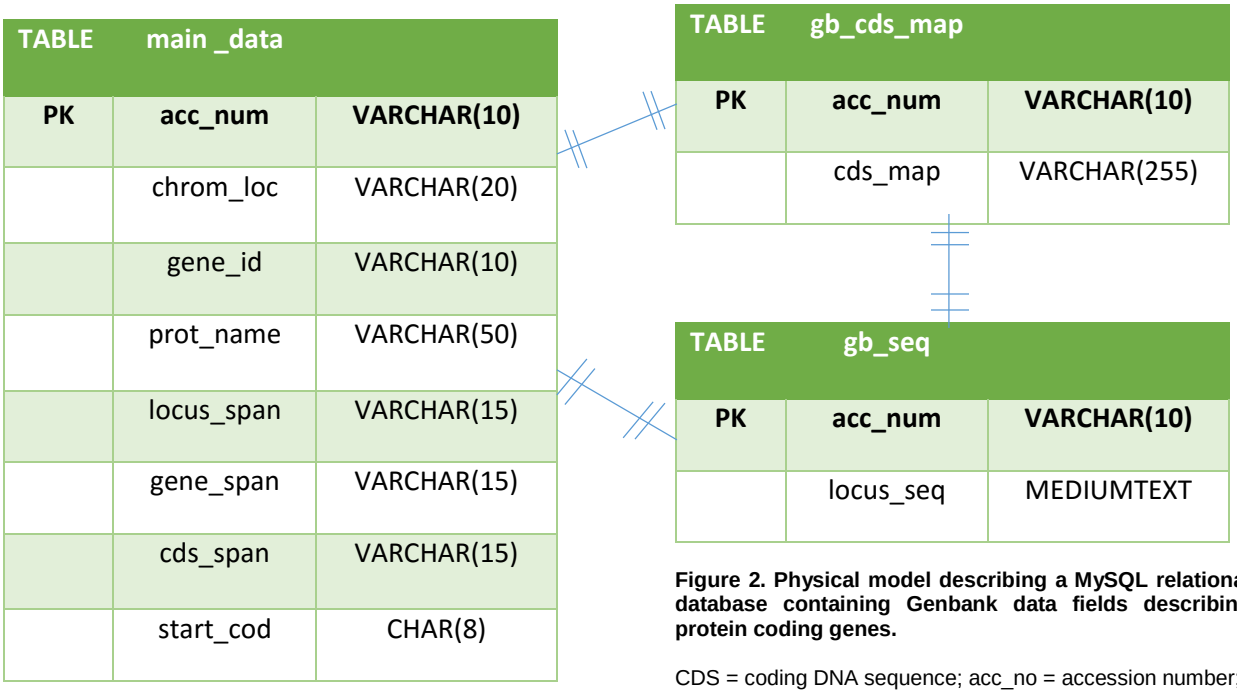

Figure 2. Physical model describing a MySQL relational database containing Genbank data fields describing protein coding genes.

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.

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 initially.

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 by updating on Github, often on a daily basis. I had little idea what they were doing until quite late on. 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.

The development process

This was simply a case of myself taking the project a step at a time. 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-step fashion as I envisaged. I was keen to ensure that the dummy version would serve to be as realistic as possible for both the user and the BL, so I wouldn't affect the progress of the other team members.

Once the dummy API was 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. Like all computer tasks, however, there is great satisfaction to be had with the rare hard-earned successes. 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 eg 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. After this had been established, the complications arose with the sheer number of combinations possible with my 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. The final list of MySQL queries requiring wrappers is quite long, and I'm sure there must be a better way to have done it. However, at this late stage I was perhaps becoming less imaginative in my approach. I was also concerned that I wouldn't meet the deadline.

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 examine the unique instances of variables in each column of a table.

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, if only to demonstrate that you are not alone when you spend hours analyzing your code, and finally see that it's just missing a parenthesis! 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.

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. However, I do not have the experience to know for sure either way. I would definitely listen to my instinct next time though.

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 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 whole 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 pleasing.

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 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.

Personal insights

As mentioned above, this project has greatly promoted my programming development. Coding is now beginning to come more naturally to me. I no longer need to think about how I will do the code for very long, it just happens more subconsciously. This is something I've noticed before about learning something new and complicated. If I didn't already know this to be the case, I would never have embarked on this course, as it was quite formidable to suddenly go into computing at this stage of my scientific career. I feel much more confident now and can't understand why I was so afraid of the command line!