Genome-Browser/database/src ----- Documentation

Instructions for setting up the database:

1. Run records_processor.py
   - input is ‘chrom_CDS_3’ file, output is ‘processed_records.txt’ and ‘processing_report.txt’
   - purpose is to remove unwanted records such as pseudogenes and records with joins to remote records; processing_report.txt lists the accession number of records that have been removed, just in case the user wants too look through them
2. Run parser.py
   - input is ‘processed_records.txt’, output is ‘summary_table.dat’, ‘coding_seq_table.dat’ and ‘full_seq_table.dat’
3. Log on to your mysql account and run create_and_populate_tables.sql
   - tables are created and data from the above ‘.dat’ files are bulk loaded

---

The database schema:

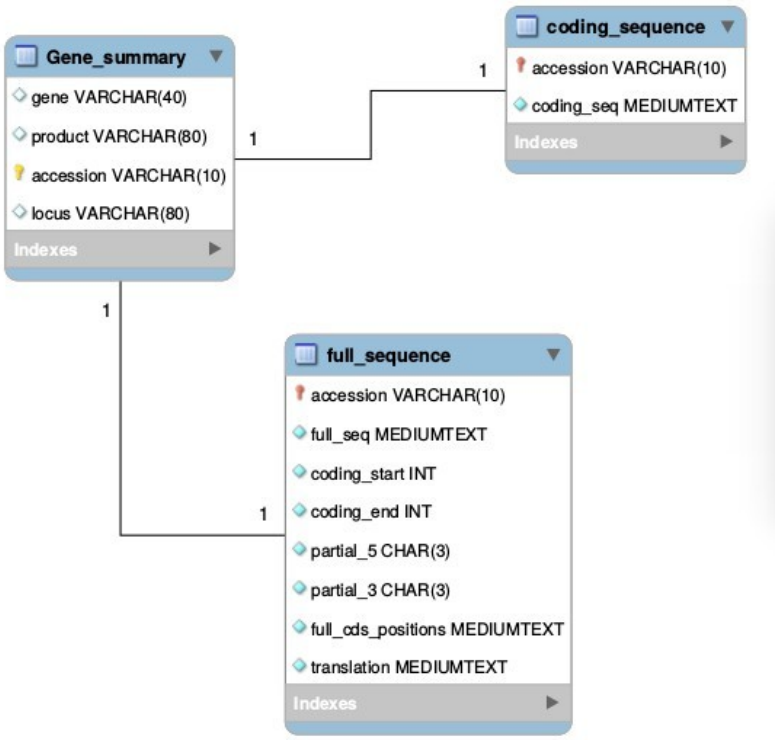

Table definitions can be found in create_and_populate_tables.sql

---

parser_module.py

Used to store all functions that are used in the parser.py script

parse_accession(gene)
input: (str) one record in genbank format
output: (str) accession number

parse_cds_feature(gene)
input: (str) one record in genbank format
output: (str) the whole ‘CDS’ section of the record, up to and including the translation
note: in parser.py, the outupt of this function is used as the input for functions that only need the cds part of the record. This ensures the same cds is always being dealt with (many records have multiple CDS sections, and features could be mixed and matched during parsing).

parse_coding_seq(gene)
input: (str) one record in genbank format
output: (str) dna sequence of the coding region, starting from the first in-frame codon; length will always be a multiple of 3; a warning is printed if dna length != 3*translation length

parse_coding_boundaries(gene)
input: (str) one record in genbank format
output: (int) (start of coding region, end of coding region)

parse_complement_or_not(cds)
input: (str) one ‘CDS’ section of a genbank record
output: (bool) True if complement, else False

parse_full_cds_coordinates(gene)
input: (str) one record in genbank format
output: (str) coding region positions, in the format ‘1..2,3..4,5..6’ (3 fragments in example, but can have 1 or more)

parse_full_dna(gene)
input: (str) one record in genbank format
output: (str) full dna sequence from the ORIGIN section of the record (without whitespace or newlines)

parse_full_reverse_complement(gene)
input: (str) one record in genbank format
output: (str) same as parse_full_dna(), but creates the reverse complement of the dna
note: this is separate from parse_full_dna() because in parse_coding_seq(), it is more efficient to use parse_full_dna() for complements as well, and only create the complement for the coding region (usually much shorter than the full dna)

parse_gene(cds)
input: (str) one ‘CDS’ section of a genbank record
output: (str) gene name, or ‘NULL’ if none found

parse_locus(gene)
input: (str) one record in genbank format
output: (str) chromosomal location, or ‘NULL’ if none found

parse_partial(cds)
input: (str) one ‘CDS’ section of a genbank record
output: ((str) partial_5, (str) partial_3) where each can be ‘yes’ or ‘no’

parse_product(cds)
input: (str) one ‘CDS’ section of a genbank record

output: (str) name of protein product, or ‘NULL’ if none found

parse_translation(cds)
input: (str) one ‘CDS’ section of a genbank record
output: (str) one-letter amino acid sequence

---

data_access.py

API that is imported for middle layer functions

get_all_coding_seqs()
input: no arguments taken
output: (str) concatenated string of all coding dna sequences from the database
notes: used for whole chromosome codon usage precalculation

get_coding_seq(accession)
input: (str) a valid accession number that exists in the database
output: (str) coding dna sequence for that record
notes: output will always be in-frame and the length will be a multiple of 3

get_full_seq_and_positions(accession)
input: (str) a valid accession number that exists in the database
output: ((str) full dna sequence, (int) coding start, (int) coding end, (str) partial 5, (str) partial 3, (str) full coordinates)
notes: partial 5 and partial 3 is one of two options: ‘yes’ or ‘no’
notes: full coordinates is in the format ‘1..2[,3..4]’ with 0 or more repetitions of what’s inside ‘[]’; the intended use is to supply the information for the highlighting of coding regions of the full dna sequence displayed on the page for each gene

get_summary_table_as_list()
input: no arguments taken
output: [(str) gene, (str) product, (str) accession, (str) locus], [(str) gene, (str) product, (str) accession, (str) locus], etc for all rows of the table
notes: intended purpose is to generate the summary table on the home page of the website

search_by_accession(accession)
input: (str) a valid accession number that exists in the database
output: [(str) gene, (str) product, (str) accession, (str) locus] for the searched record