Lab 1 – Database

Introduction:

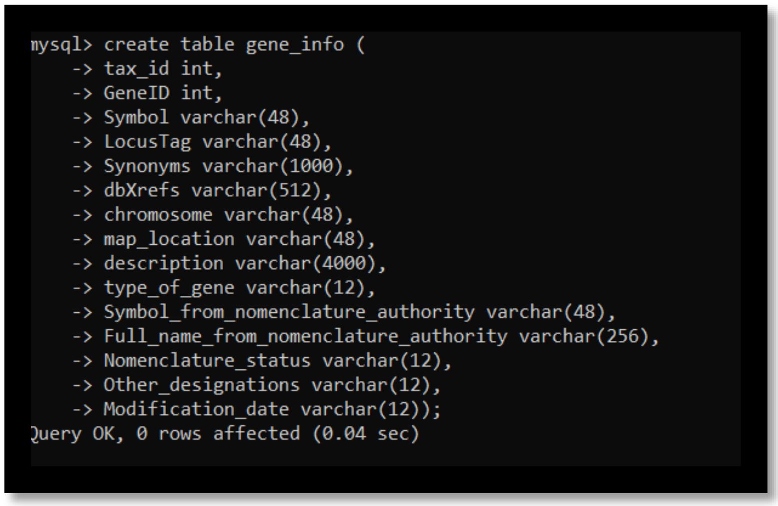
This lab's purpose is to gain an understanding of database management system installation and gain a general overview on the process of database creation, defining table schemas, the pragmatics of creating tables, importing data, and defining primary keys and indexes.

Observations and results:

- **Initial Observations**
    - Before the creation of the database, inspection of the gene_info100 file, a subset of the data that will be used in the database, resulted in several discoveries about the data. First, while it is noted later in the lab instructions, the headers for each column were missing, which makes interpretation of the data very difficult. Secondly, it is interesting to see that there appears to be 6 attributes that no entry has a value for. Lastly, it was observed that all but 3 columns seem to act as alphanumeric identifiers, which adds to the difficulty of data interpretation.

Importing data into a database

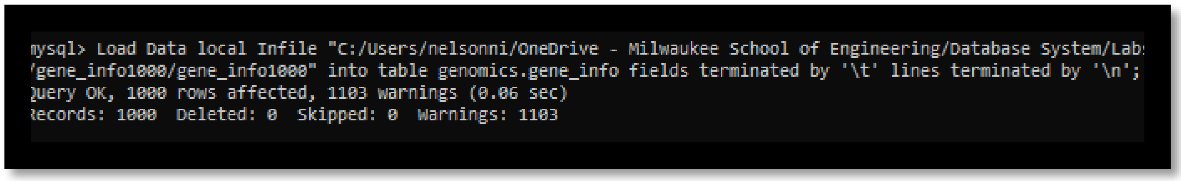
- **Creating initial gene_info table**
    - Query:



```
nysql> create table gene_info (
    -> tax_id int,
    -> GeneID int,
    -> Symbol varchar(48),
    -> LocusTag varchar(48),
    -> Synonyms varchar(1000),
    -> dbXrefs varchar(512),
    -> chromosome varchar(48),
    -> map_location varchar(48),
    -> description varchar(4000),
    -> type_of_gene varchar(12),
    -> Symbol_from_nomenclature_authority varchar(48),
    -> Full_name_from_nomenclature_authority varchar(256),
    -> Nomenclature_status varchar(12),
    -> Other_designations varchar(12),
    -> Modification_date varchar(12));
Query OK, 0 rows affected (0.04 sec)
```

- Observation:
  - This SQL query's purpose is to create a table that will allow the loading of data that fits this table's defined structure, such as the gene_info1000 file. It appears that each argument specifies an attribute, followed by a data type, and if the type is in a varchar format, an argument needs to be specified for the number of characters allowed to be stored in that attribute.

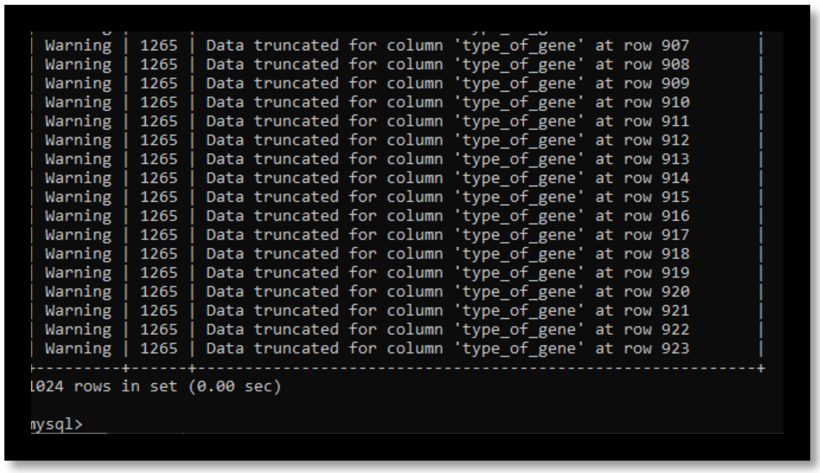
- **Loading *gene_info1000***
  - Query:



```
nysql> Load Data local Infile "C:/Users/nelsonni/OneDrive - Milwaukee School of Engineering/Database System/Lab:
'gene_info1000/gene_info1000" into table genomics.gene_info fields terminated by '\t' lines terminated by '\n';
Query OK, 1000 rows affected, 1103 warnings (0.06 sec)
Records: 1000  Deleted: 0  Skipped: 0  Warnings: 1103
```

  - Observation:
    - The syntax for loading data from a file into a table via a SQL query seems very straight forward after executing the above command. It appears as simple as specifying the location of the data file, then specifying how fields are terminated and how the lines are terminated, and finishing by specifying what table the data should be placed into. One curious observation from this query is that there is a large amount of warning, it is surprising that with this large number of warnings that the Query is still deemed "OK".

- **Showing warning for *gene_info1000***
  - Query:

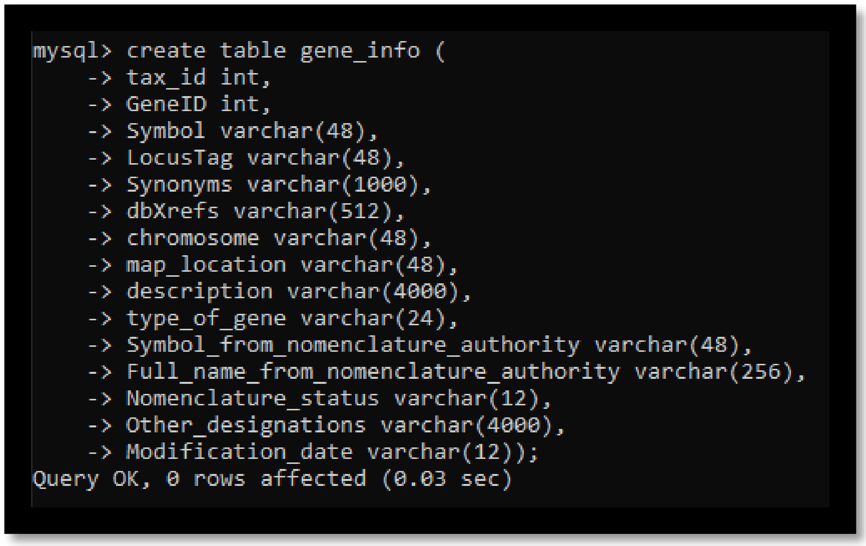


```
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 907      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 908      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 909      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 910      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 911      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 912      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 913      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 914      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 915      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 916      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 917      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 918      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 919      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 920      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 921      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 922      |
| Warning | 1265 | Data truncated for column 'type_of_gene' at row 923      |
+---------+------+----------------------------------------------------------+
1024 rows in set (0.00 sec)

mysql>
```

- Observation:
  - It appears that all of the warnings for the previous load query are a result of truncated data. This likely means that for certain fields not enough space was specified in the creation of the table.

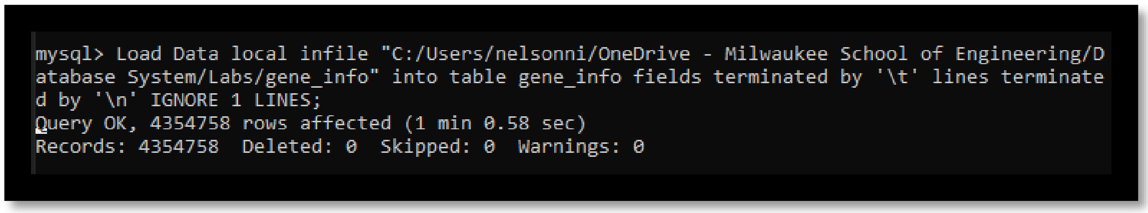
- **Creating *gene_info* with larger field for *type_of_gene* and *Other_designation***
  - Query:



```
mysql> create table gene_info (
    -> tax_id int,
    -> GeneID int,
    -> Symbol varchar(48),
    -> LocusTag varchar(48),
    -> Synonyms varchar(1000),
    -> dbXrefs varchar(512),
    -> chromosome varchar(48),
    -> map_location varchar(48),
    -> description varchar(4000),
    -> type_of_gene varchar(24),
    -> Symbol_from_nomenclature_authority varchar(48),
    -> Full_name_from_nomenclature_authority varchar(256),
    -> Nomenclature_status varchar(12),
    -> Other_designations varchar(4000),
    -> Modification_date varchar(12));
Query OK, 0 rows affected (0.03 sec)
```

  - Observation:
    - As discussed in the previous query is does appear that the correct way to remove the truncated data warnings is to ensure that enough memory is allocated for the entries of an attribute in a table.

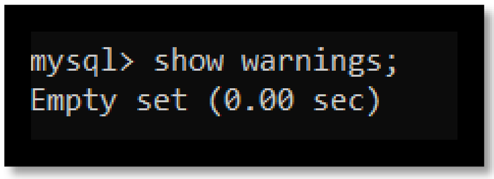
- **Loading *gene_info***
  - Query:



```
mysql> Load Data local infile "C:/Users/nelsonni/OneDrive - Milwaukee School of Engineering/D
atabase System/Labs/gene_info" into table gene_info fields terminated by '\t' lines terminate
d by '\n' IGNORE 1 LINES;
Query OK, 4354758 rows affected (1 min 0.58 sec)
Records: 4354758  Deleted: 0  Skipped: 0  Warnings: 0
```

- Observation:
  - Note the major time difference between this load query for *gene_info* compared to gene_info1000, 1 minutes vs. 6 seconds, respectively. This shows how time consuming certain data loads can be.

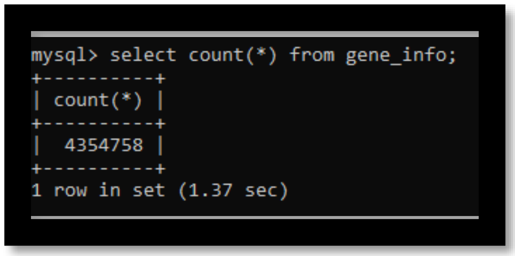
- **Checking for warning from loading *gene_info***
  - Query:



```
mysql> show warnings;
Empty set (0.00 sec)
```

  - Observation:
    - It appears that increasing the amount of memory for certain fields fixed the data truncated warning messages that were appearing earlier. This is an important lesson in ensuring that fields are sized appropriately for the entries that will be loaded into them.

- **Verifying length of *gene_info* table**
  - Query:

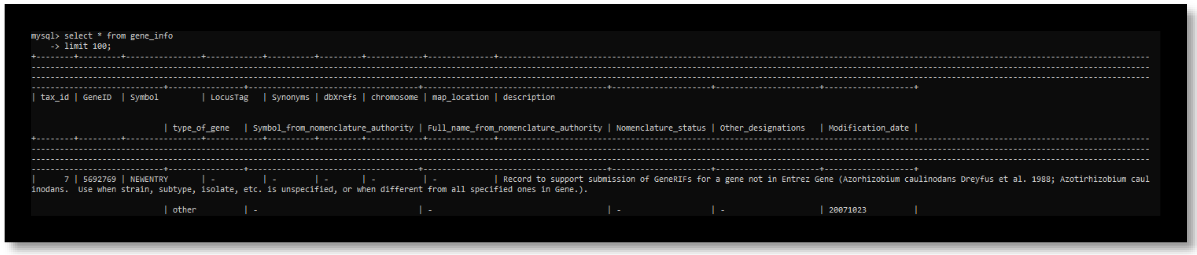


```
mysql> select count(*) from gene_info;
+----------+
| count(*) |
+----------+
|  4354758 |
+----------+
1 row in set (1.37 sec)
```

  - Observation:
    - From this query it is clear to see that there are 4,354,758 entries in the *gene_info* data set. This makes it much more clear why this data file took much longer to load than the *gene_info1000* file which only had 1000 entries.

Creating Working Table

- **Inspecting the *gene_info* table created**
  - Query:



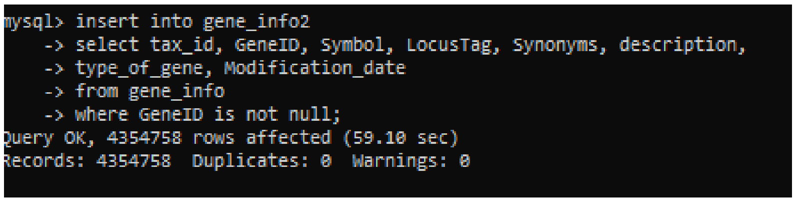
```
mysql> select * from gene_info
    -> limit 100;
+--------+---------+---------------+------------+----------+---------+-----------+--------------+-------------------------------------------------------------------------------------------------------------
| tax_id | GeneID  | Symbol        | LocusTag   | Synonyms | dbXrefs | chromosome | map_location | description


                         | type_of_gene   | Symbol_from_nomenclature_authority | Full_name_from_nomenclature_authority | Nomenclature_status | Other_designations   | Modification_date |
+--------+---------+---------------+------------+----------+---------+-----------+--------------+-------------------------------------------------------------------------------------------------------------
|      7 | 5692769 | NEWENTRY      | -          | -        | -       | -          | -            | Record to support submission of GeneRIFs for a gene not in Entrez Gene (Azorhizobium caulinodans Dreyfus et al. 1988; Azotirhizobium caul
inodans.  Use when strain, subtype, isolate, etc. is unspecified, or when different from all specified ones in Gene.).

                         | other          | -                                  | -                                     | -                   | -                    | 20071023          |
```

  - Observation:
    - This query's purpose was to inspect the first 100 entries of the *gene_info* table. However, after seeing the result I'm not sure how practical the command window is for inspecting tables, as the formatting is not very readable.

- **Inserting *gene_info* data into *gene_info2* table**
  - Query:

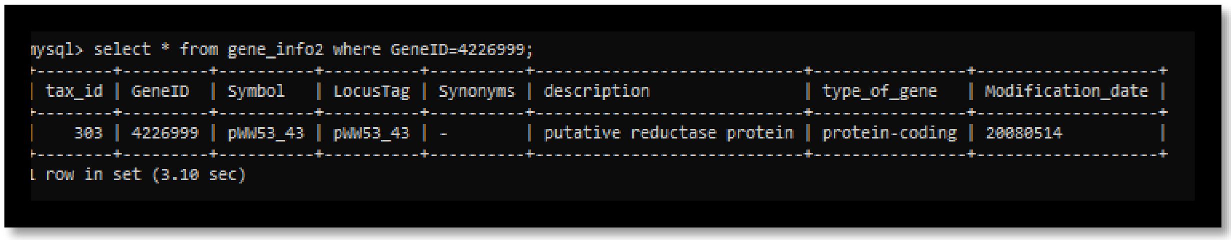


```
mysql> insert into gene_info2
    -> select tax_id, GeneID, Symbol, LocusTag, Synonyms, description,
    -> type_of_gene, Modification_date
    -> from gene_info
    -> where GeneID is not null;
Query OK, 4354758 rows affected (59.10 sec)
Records: 4354758  Duplicates: 0  Warnings: 0
```

  - Observation:
    - This was an interesting query as I was previously unaware that you could select attributes form one table and insert them into another. This command seems very useful for what was demonstrated in this command: distilling down data to only the desired attributes.

Primary Keys and Indexes

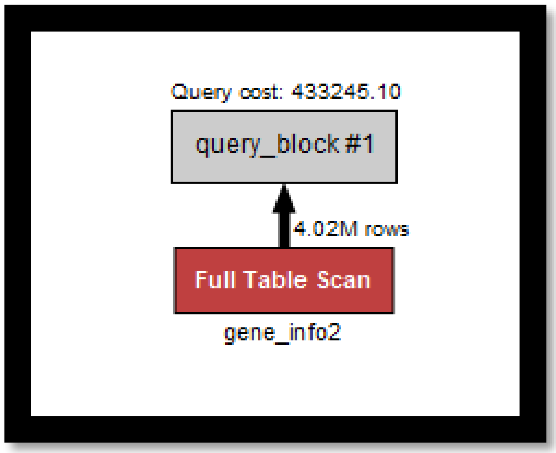
- **Retrieving record for specific gene (command line)**
  - Query:



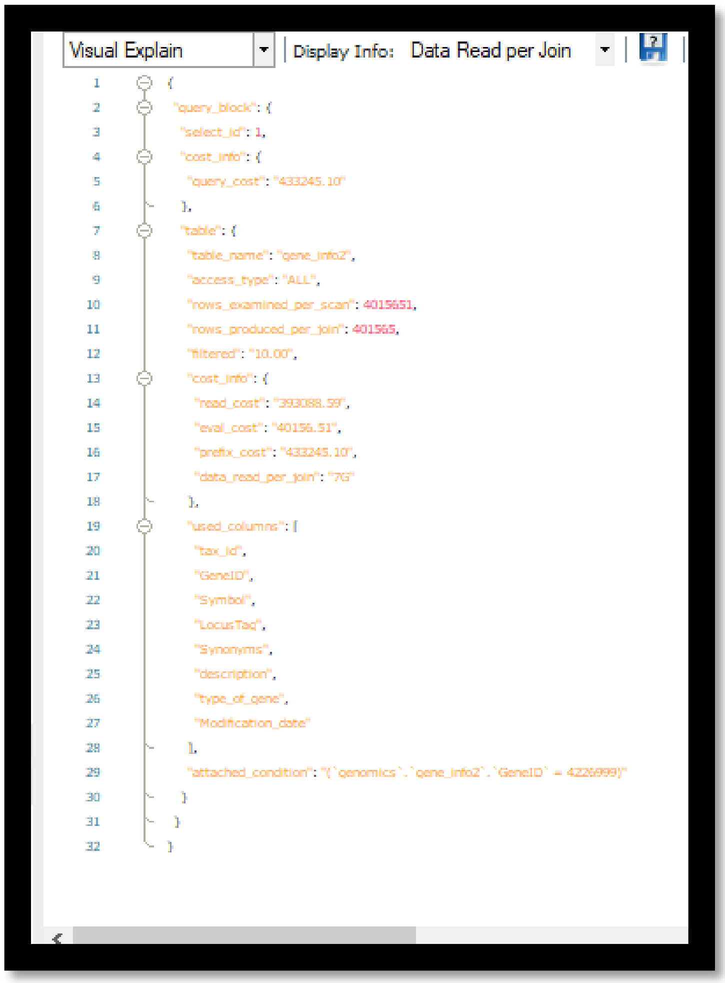
```
ysql> select * from gene_info2 where GeneID=4226999;
+--------+---------+----------+----------+----------+---------------------------+---------------+-------------------+
| tax_id | GeneID  | Symbol   | LocusTag | Synonyms | description               | type_of_gene  | Modification_date |
+--------+---------+----------+----------+----------+---------------------------+---------------+-------------------+
|    303 | 4226999 | pWW53_43 | pWW53_43 | -        | putative reductase protein | protein-coding | 20080514         |
+--------+---------+----------+----------+----------+---------------------------+---------------+-------------------+
 row in set (3.10 sec)
```

- Observation:
  - The time to execute a select query seems dangerously long. While this table does have millions of entries, I can imagine that this table can be seen as a comparatively small table, and if countless selects are being made in a significantly larger table, a massive time for execution may result.
- Time to retrieve *without* primary key: **3.10 seconds**

- **Retrieving record for specific gene (GUI)**
  - Figure:



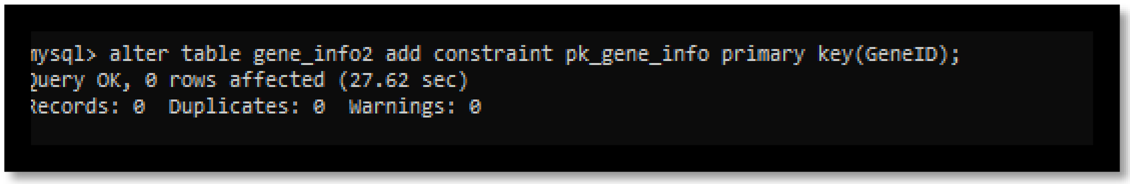
  - Text:



```
{
 "query_block": {
  "select_id": 1,
  "cost_info": {
   "query_cost": "433245.10"
  },
  "table": {
   "table_name": "gene_info2",
   "access_type": "ALL",
   "rows_examined_per_scan": 4015651,
   "rows_produced_per_join": 401565,
   "filtered": "10.00",
   "cost_info": {
    "read_cost": "393088.59",
    "eval_cost": "40156.51",
    "prefix_cost": "433245.10",
    "data_read_per_join": "7G"
   },
   "used_columns": [
    "tax_id",
    "GeneID",
    "Symbol",
    "LocusTag",
    "Synonyms",
    "description",
    "type_of_gene",
    "Modification_date"
   ],
   "attached_condition": "(`genomics`.`gene_info2`.`GeneID` = 4226999)"
  }
 }
}
```

- Observation:
  - Using the GUI to analyze the select statement seems very powerful for diagnosing a database that is performing slowing. The GUI does an excellent job of explaining if a full table scan was done, and at what cost that select was executed at.

- **Defining Primary key**
  - Query:

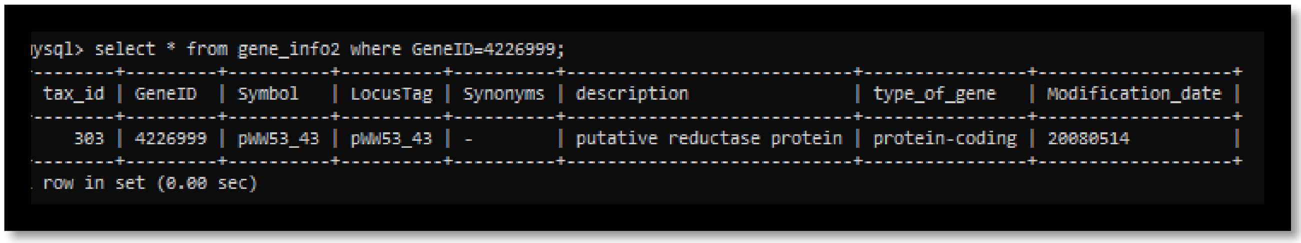


```
mysql> alter table gene_info2 add constraint pk_gene_info primary key(GeneID);
Query OK, 0 rows affected (27.62 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

- Observation:
  - The defining of a primary key took much longer than I was expecting, I initially envisioned this operation as an O(1) operation in big-O notation.

However, after more thought I realize that this is likely an O(n) operation, where n is the number of entries in the database.

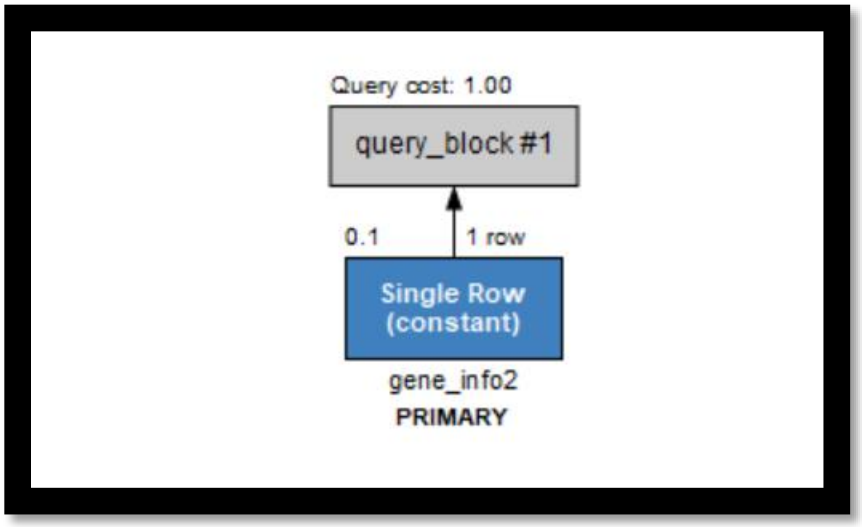
- **Retrieving record for specific gene with Primary key defined (command line)**
    - Query:



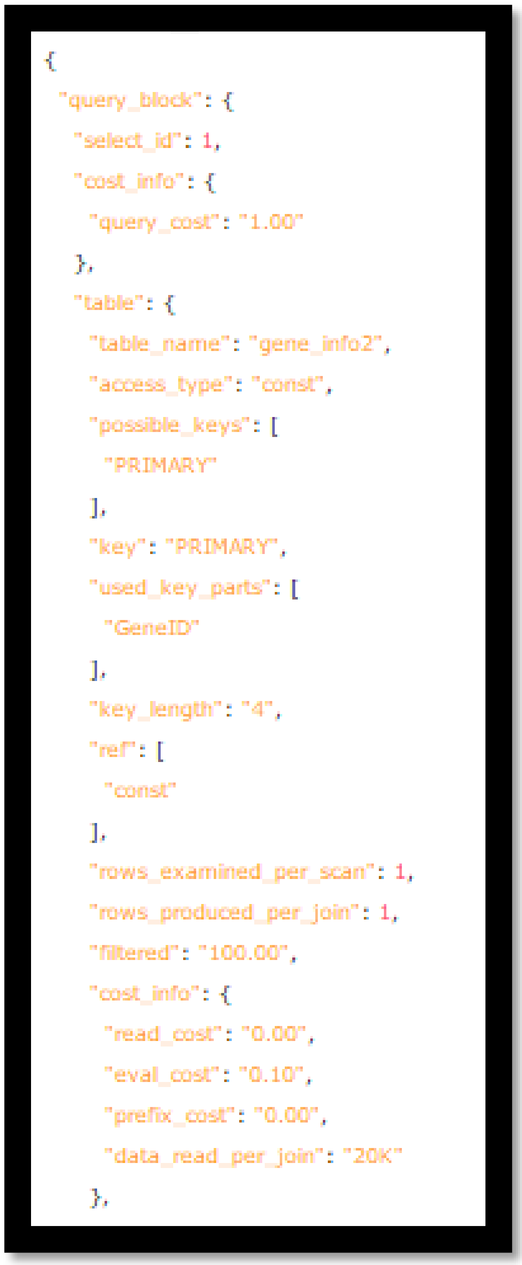
```
ysql> select * from gene_info2 where GeneID=4226999;
--------+---------+----------+----------+----------+---------------------------+---------------+-------------------+
 tax_id | GeneID  | Symbol   | LocusTag | Synonyms | description               | type_of_gene  | Modification_date |
--------+---------+----------+----------+----------+---------------------------+---------------+-------------------+
    303 | 4226999 | pWW53_43 | pWW53_43 | -        | putative reductase protein | protein-coding | 20080514         |
--------+---------+----------+----------+----------+---------------------------+---------------+-------------------+
 row in set (0.00 sec)
```

- Observation:
    - It's clear from this execution of the select statement, which has a primary key defined this time, that primary keys have a great impact on the performance of queries on tables, and exponentially decrease execution time.
- Time to retrieve *with* primary key: **0.0 seconds**

- **Retrieving record for specific gene with Primary key defined (GUI)**
    - Figure:



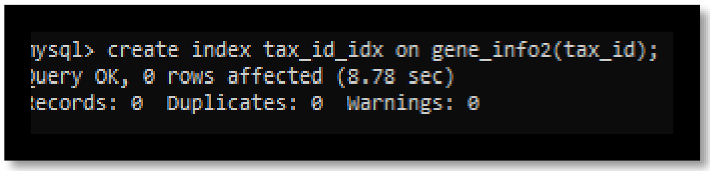
- Text:



```
{
  "query_block": {
    "select_id": 1,
    "cost_info": {
      "query_cost": "1.00"
    },
    "table": {
      "table_name": "gene_info2",
      "access_type": "const",
      "possible_keys": [
        "PRIMARY"
      ],
      "key": "PRIMARY",
      "used_key_parts": [
        "GeneID"
      ],
      "key_length": "4",
      "ref": [
        "const"
      ],
      "rows_examined_per_scan": 1,
      "rows_produced_per_join": 1,
      "filtered": "100.00",
      "cost_info": {
        "read_cost": "0.00",
        "eval_cost": "0.10",
        "prefix_cost": "0.00",
        "data_read_per_join": "20K"
      },
```

- Observation:
  - Using the GUI to analyze the select statement after defining a primary key makes the performance advantage offered by this difference very clear. Instead of scanning the whole table like was previously done, the GUI now explains that this select statement is a constant time operation.
-

- **Creating index on *tax_id***
  - Query:

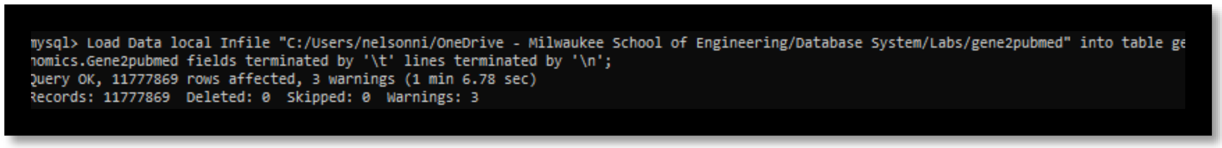


```
ysql> create index tax_id_idx on gene_info2(tax_id);
uery OK, 0 rows affected (8.78 sec)
ecords: 0  Duplicates: 0  Warnings: 0
```

- Observation:
  - Similar to adding a primary key to a table, adding an index to a table likely has similar performance gains. The rationale for this is due to the fact that this may allow for a binary search tree approach to be used, instead of needing to iterate through each entry.

- Loading *gene2pubmed* data

- Query:

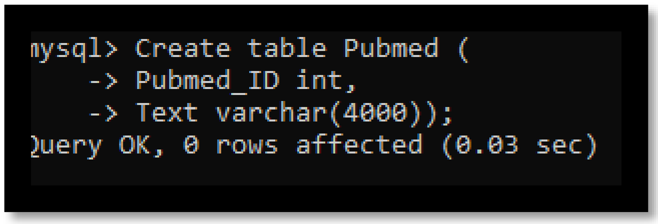


```
mysql> Load Data local Infile "C:/Users/nelsonni/OneDrive - Milwaukee School of Engineering/Database System/Labs/gene2pubmed" into table ge
omics.Gene2pubmed fields terminated by '\t' lines terminated by '\n';
Query OK, 11777869 rows affected, 3 warnings (1 min 6.78 sec)
Records: 11777869  Deleted: 0  Skipped: 0  Warnings: 3
```

- Observation:
  - This load query was performed almost identical to the other queries. However, it is interesting to see that this data file has 11,777,869 entries, which is the largest data file to be loaded yet. For having over double the entries of *gene_info*, it has a similar load time, this must mean that *gene2pubmed* has less attributes than *gene_info*.

- Creating *pubmed* table

- Query:

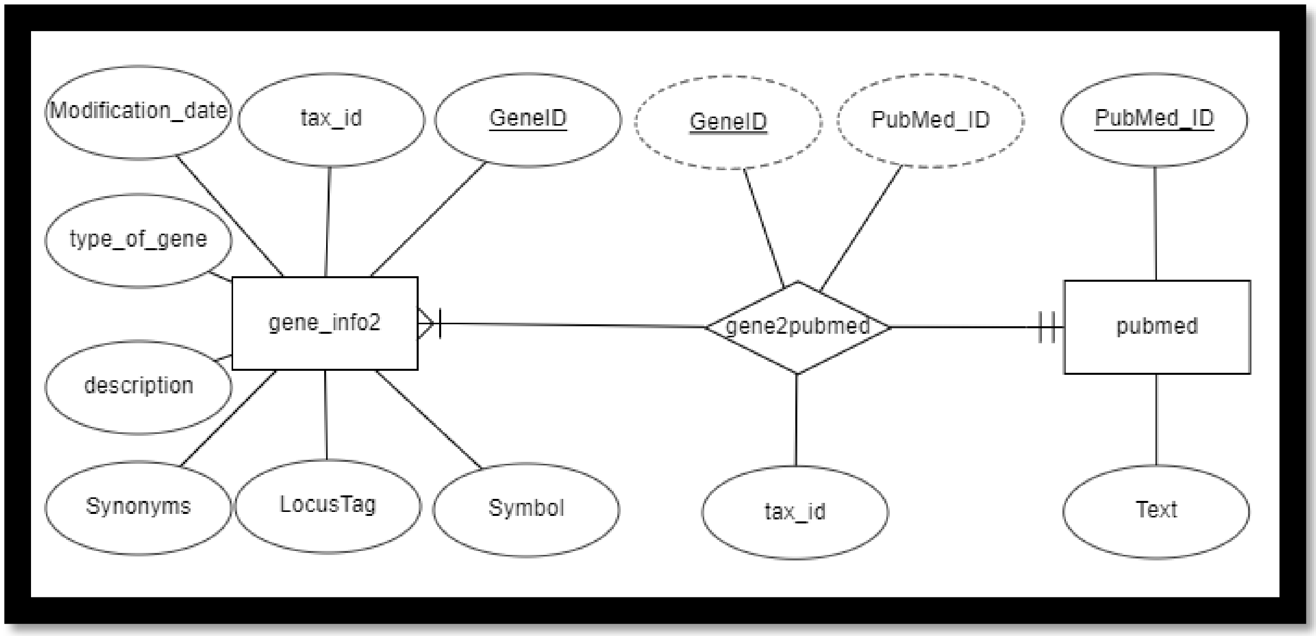


```
ysql> Create table Pubmed (
    -> Pubmed_ID int,
    -> Text varchar(4000));
uery OK, 0 rows affected (0.03 sec)
```

- Observation:
  - This table created seems to be the simplest structure defined yet. With only two attributes defined, it appears that this table will contain a unique attribute, *Pubmed_ID*, and whose *Text* attribute will be used to describe *gene_info* data.

ERD Diagram

- The ERD diagram below is a depiction of the interactions between the gene_info2 table, the gene2pubmed table, and the pubmed table.



Rationale:

- From the tables created in the aforementioned steps, it was concluded that gene_info2 would be an entity, as it stored a wide array of attributes not seen else where in the database, whose structure appeared to be important, and lastly because there were multiple gene_info2 entries per pubmed entity. The Pubmed table was created as an entity because its structure appeared important and a single pubmed entity can have multiple gene_info2 entities. The gene2pubmed table was determined to be a relationship. The reason for this is that its structure seemed to be a link between gene_info2 and pubmed. This is further proven by the fact that it has two derived attributes, GeneID which is unique for each gene2pubmed entry but is derived from gene_info2's GeneID attribute, and PubMed_ID, which is derived from pubmed's Pubmed_ID attribute. Lastly, it was determined that a geneinfo2 entity had a mandatory relationship with a pubmed entity, who also had a mandatory relationship in the reverse direction. The reason for this is that without a single gene_info2 entry, there is no pubmed entries, and without a single pubmed entry, that means there are no gene_info2 entries.

Analysis

What Was Learned:

- Many things were learned in this lab as this was my first real experience creating a database, and also my first time utilizing MySQL. The first thing that I learned was how to make a database connection to my local machine using MySQL. Following this I learned how to create a new schema/database utilizing MySQL. In addition, I learned how to properly import a file of data that is not in the .xlsx format into excel. Furthermore, I learned how to use the MySQL query window to create a table which allocates space for data that needs to be imported, and also how to use the MySQL command line connection to load desired data into created tables. Through this I also discovered how to verify that data had been loaded and interpreted correctly by verifying the length of the database table and checking for warnings. I then learned how to select data from an existing table and insert it into a newly created table. One of the biggest things that I learned was the performance increase that can be gained by defining a primary key and/or creating an index for a large table. Lastly, hands on experience creating ERD diagrams for databases was gained and a basic understanding of the online application ERDPlus was achieved.

How This Lab Could be Improved:

- One of the details that I encountered in this lab where I thought there could be room for improvement was in the formatting of gene_info1000. The reason for this is that for someone new to databases and MySQL, this was one of the most readable instances of the data for me in this lab, but without the headers included it was difficult to understand what data I was using in this lab and what it meant. This didn't hinder me from completing the lab, but it would have been nice if the header was included so that I could have had more context when working through the lab. Another thing that would have been nice for this first lab would be to have examples of lab reports from the past so that students can understand the expectation for the format of reports in this course. Lastly, it would have been nice to receive a few hints about the relationships between the entities for the ERD diagram that is part of this lab report. The reason for this is that with minimal context on what the data inside our database is and their use, it is difficult to understand the connection between the 3 tables and make coherent connections between all of them.

Conclusion:

This lab's purpose was to gain an understanding of database management system installation, gain a general overview on the process of database creation, defining table schemas, the pragmatics of creating tables, importing data, and defining primary keys and indexes. These goals were accomplished and thanks to this lab my understanding of databases and specifically the use of them inside MySQL has grown significantly. While I believe there were minor areas for improvement, overall, I really enjoyed this lab and I feel much more confident using and understanding databases.
