**Increasing Memory Allotment:**

Initial Memory Values:

* Max\_heap\_table\_size value:
  + Graphical user interface, text, application

    Description automatically generated
* TMP\_table\_size value:
  + Graphical user interface

    Description automatically generated with medium confidence

Memory Values after queries to increase storage:

* Max\_heap\_table\_size value:
  + Graphical user interface, text, application

    Description automatically generated
* TMP\_table\_size value:
  + Text

    Description automatically generated

Creating tables using InnoBD engine:

* Loading gene\_info5000 data:
  + Text

    Description automatically generated
  + Records loaded: 50,000
* Verifying gene\_info5000 data loaded properly
  + Table

    Description automatically generated
  + Graphical user interface, application

    Description automatically generated
* Loading gene2pubmed data:
  + Text

    Description automatically generated
  + Rows loaded: 12,917,351
* Verifying gene\_info5000 data loaded properly
  + Table

    Description automatically generated
  + Table

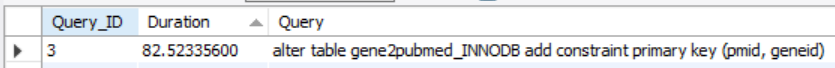
    Description automatically generated
* Noting table row counts:
  + Gene\_info\_InnoDB:
  + Graphical user interface, application, website

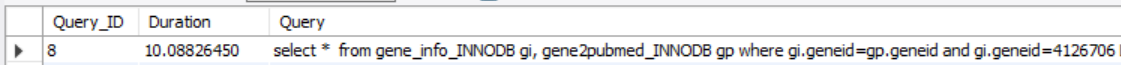
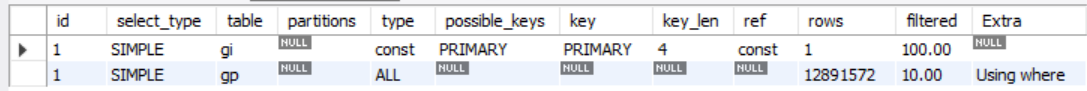
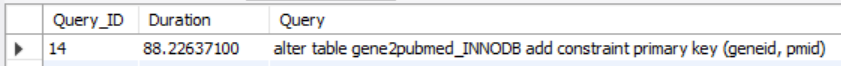
    Description automatically generated
  + Gene2pubmed\_InnoDB:
  + Graphical user interface, application, website

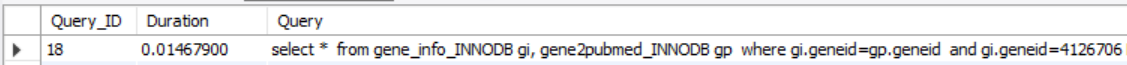
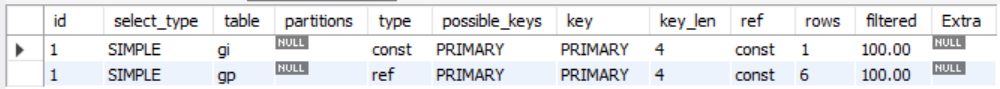
    Description automatically generated

Question 1 - Join

* Joining Tables:
  + Table

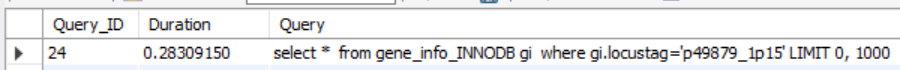
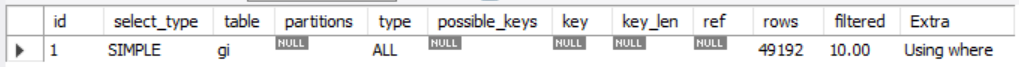
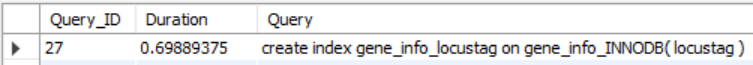
    Description automatically generated­­­
  + Execution Time:
    - 
  + Analyzing Query Plan:
    - 
    - Explanation:
      * This explain statement is showing that for the above join statement, two tables are being joined, where the two are being joined via a hash join. In this join, just under 50k entries from gene\_info\_InnoDB were iterated through, and just over 1.25 million entries from gene2pubmed\_InnoDB. Without any primary keys, SQL has no option but to iterate through just about every element.
* Adding Primary Keys:
  + - 
    - 
* Re-executing join query:
  + Table

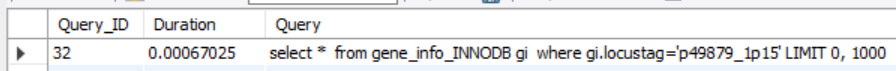
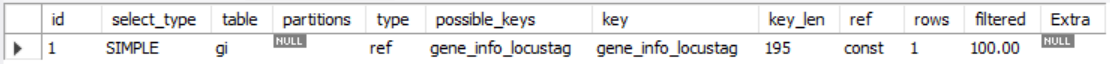
    Description automatically generated
  + Execution Time:
    - 
  + Analyzing Query Plan
  + 
  + Explanation:
    - The execution of this join is only marginally faster than the previous execution, gaining about a 2 second advantage. The execution plan shows that gene\_info\_INNODB is able to use its primary key. This allows the query to only attempt to join on a single row for gene\_info\_INNODB, which is where its geneid attribute equals 4126706. However, gene2pubmed\_INNODB is not able to use a primary key on this join because gene2pubmed\_INNODB’s primary key is a combination key in the order pmid, geneid and the join occurs exclusively on geneid.
* Redefining gene2pubmed\_INNODB primary key
  + 
* Re-executing join query:
  + Table

    Description automatically generated
  + Execution Time:
    - 
  + Analyzing Query Plan
  + 
  + Explanation:
    - The execution of this join query is almost 100 times faster than the original query, which originally took about 12 seconds. The execution plan shows that gene\_info\_INNODB and gene2pubmed\_INNODB are able to use their primary key. This allows the query to only attempt to join on a single row for gene\_info\_INNODB, which is where its geneid attribute equals 4126706. Also, gene2pubmed\_INNODB is able to use a primary key on this join and therefore only searches the 6 rows of entries that were referenced by the geneid 4126706. The ability to utilize gene2pubmed\_INNODB’s combination key is likely due to the fact that geneid is now the first portion of the combination key.

Question 2 - Restrict

* Executing select query:
* Graphical user interface, text, application, Word

  Description automatically generated
  + Execution Time:
    - 
  + Execution Plan:
    - 
    - Explanation:
      * This query filtered through 49,192 rows of entries, no key was used as locustag is not defined as a primary key or index, as such it must iterate through almost all of the entries.
* Creating index for locustag
  + 
  + Execution Time:
    - 
  + Re-executing select query:
    - Graphical user interface, text, application, Word

      Description automatically generated
    - Execution time:
      * 
    - Execution Plan:
      * 
      * Explanation:
        + With an index defined, this restrict query is executed in almost a 100th of the original time that it took for the first query. Only a single row needed to be searched, the reason for this that now with an index defined for locustag, sql identifies gene\_info\_locustag as a key and was able to immediately find the exact entry for the select query.

Question 3 – Range Query

* Executing select query
  + Graphical user interface, text, application

    Description automatically generated
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + 526 rows needed to be searched for this query. This query was able to utilize the primary key, and as such was able to search through less rows of entries.

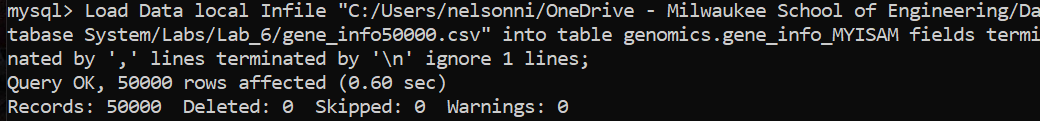
Question 4 - Insert

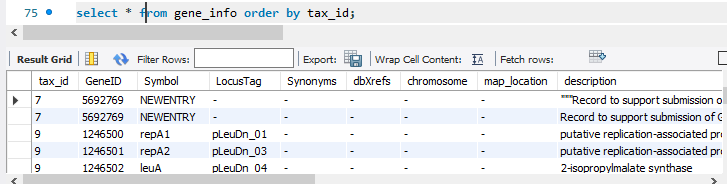
* Executing insert statement
  + 
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + This query is interesting in that it appears that the InnoDB engine was able to insert without utilizing any key, and also able to do it without iterating through any disclosed rows. This is the likely reason for the quick execution time.

Question 5 – Update

* Executing update statement
  + 
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + Only a single row was searched through for this query, likely due to that fact that because locustag is an index, SQL was able to identify it as a key, resulting in a relatively quick execution time.

Creating tables using MyISAM engine:

* Loading gene\_info5000 data:
  + 
  + Records loaded: 50,000
* Verifying gene\_info5000 data loaded properly
  + Table

    Description automatically generated
  + 
* Loading gene2pubmed data:
  + A screenshot of a computer

    Description automatically generated with medium confidence
  + Rows loaded: 12,917,351
* Verifying gene\_info5000 data loaded properly
  + Graphical user interface, table

    Description automatically generated
  + Graphical user interface, application, table

    Description automatically generated
* Noting table row counts:
  + Gene\_info\_MYISAM:
  + Graphical user interface

    Description automatically generated with medium confidence
  + Gene2pubmed\_ MYISAM:
  + Graphical user interface

    Description automatically generated

Question 1 - Join

* Joining Tables:
  + Graphical user interface, table

    Description automatically generated ­­­
  + Execution Time:
    - 
  + Analyzing Query Plan:
    - 
    - Explanation:
      * This explain statement is showing that for the above join statement, two tables are being joined, where the two are being joined via a hash join. In this join, 50k entries from gene\_info\_InnoDB were iterated through, and just over 1.29 million entries from gene2pubmed\_InnoDB. Without any primary keys, SQL has no option but to iterate through just about every element. However, this execution time of 25 seconds is much slower than InnoDB’s original execution. This is likely due to the fact that without any indexes or keys defined, MyISAM is only able to use the file that contains the data, and has no way to utilize its index file.
* Adding Primary Keys:
  + - 
    - 
* Re-executing join query:
  + Graphical user interface, table

    Description automatically generated
  + Execution Time:
    - 
  + Analyzing Query Plan
  + Graphical user interface, application, table

    Description automatically generated
  + Explanation:
    - This execution plan shows that gene\_info\_MyISAM is able to use its primary key. This allows the query to only attempt to join on a single row for gene\_info\_INNODB, which is where its geneid attribute equals 4126706. However, gene2pubmed\_INNODB is not able to use a primary key on this join because gene2pubmed\_INNODB’s primary key is a compound key in the order pmid, geneid and the join occurs exclusively on geneid. This results in 1.29 million rows needed to be iterated through, which is the likely reason for the high execution time of 32 seconds.
* Redefining gene2pubmed\_INNODB primary key
  + 
* Re-executing join query:
  + Graphical user interface, table

    Description automatically generated
  + Execution Time:
    - 
  + Analyzing Query Plan
  + Graphical user interface, text, application

    Description automatically generated
  + Explanation:
    - This execution plan shows that gene\_info\_MySAM and gene2pubmed\_MySAM are able to use their primary key. This allows the query to only attempt to join on a single row for gene\_info\_INNODB, which is where its geneid attribute equals 4126706. Also, gene2pubmed\_INNODB is able to use a primary key on this join and therefore only searches the 5 rows of entries that were referenced by the geneid 4126706. This is a major improvement in the original execution time which was 22 seconds, resulting in above a 4,000x speed up.

Question 2 – Restrict:

* Executing select query:
* Graphical user interface, text, application, Word, email

  Description automatically generated
  + Execution Time:
    - 
  + Execution Plan:
    - 
    - Explanation:
      * This query filtered through 50k rows of entries, no key was used as locustag is not defined as a primary key or index, as such it must iterate through almost all of the entries. This time is comparable to InnoDB’s initial time for execution as well.
* Creating index for locustag
  + 
  + Execution Time:
    - 
  + Re-executing select query:
    - Graphical user interface, text, application, Word, email

      Description automatically generated
    - Execution time:
      * 
    - Execution Plan:
      * 
      * Explanation:
        + Only a single row needed to be searched, the reason for this that with an index defined, the key gene\_info\_locustag was used to immediately find the exact entry for the select query.

Question 3 - Range

* Executing select query
  + Graphical user interface, text, application

    Description automatically generated
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + 695 rows needed to be searched for this query, with the utilization of the of the primary key assisting in this smaller search size. However, for the same query in InnoDB, MyISAM must search through over 100 more entries, this is likely due to the fact that InnoBD utilizes a B+ tree while MyISAM uses a data page and an index page.

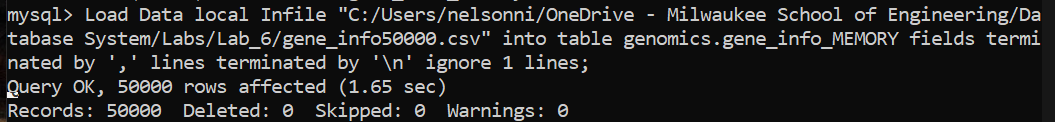
Question 4 - Insert

* Executing insert statement
  + 
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + This query is interesting in that it appears that the MyISAM engine was able to insert without utilizing any key, and also able to do it without iterating through any disclosed rows.

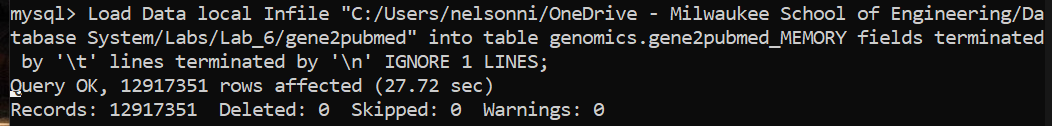
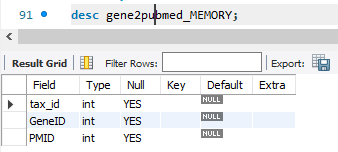
Question 5 – Update

* Executing update statement
  + 
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + Only a single row was searched through for this query, likely due to that fact that because locustag is an index, SQL was able to identify it as a key, resulting in a relatively quick execution time.

Creating tables using Memory engine:

* Loading gene\_info5000 data:
  + 
  + Records loaded: 50,000
* Verifying gene\_info5000 data loaded properly
  + Table

    Description automatically generated
  + Table

    Description automatically generated
* Loading gene2pubmed data:
  + 
  + Rows loaded: 12,917,351
* Verifying gene\_info5000 data loaded properly
  + 
  + Graphical user interface, text, table

    Description automatically generated
* Noting table row counts:
  + Gene\_info\_MYISAM:
  + Graphical user interface, application, website

    Description automatically generated
  + Gene2pubmed\_ MYISAM:
  + Graphical user interface

    Description automatically generated with medium confidence

Question 1 - Join

* Joining Tables:
  + Table

    Description automatically generated ­­­
  + Execution Time:
    - 
  + Analyzing Query Plan:
    - 
    - Explanation:
      * This explain statement is showing that for the above join statement, two tables are being joined, where the two are being joined via a hash join. In this join, 50k entries from gene\_info\_InnoDB were iterated through, and just over 1.29 million entries from gene2pubmed\_InnoDB. Without any primary keys, SQL has no option but to iterate through just about every element. However, this execution time of 0.6553 seconds is much faster than either of the other two engines. This is likely due to the fact that the data is stored on RAM rather than on the disk, which allows much faster lookup times.
* Adding Primary Keys:
  + - 
    - 
* Re-executing join query:
  + Table

    Description automatically generated
  + Execution Time:
    - 
  + Analyzing Query Plan
  + Table, Teams

    Description automatically generated with medium confidence
  + Explanation:
    - This execution plan shows that gene\_info\_Memory is able to use its primary key. This allows the query to only attempt to join on a single row for gene\_info\_Memory, which is where its geneid attribute equals 4126706. However, gene2pubmed\_Memory is not able to use a primary key on this join because gene2pubmed\_Memory’s primary key is a compound key in the order pmid, geneid and the join occurs exclusively on geneid. This results in 1.29 million rows needed to be iterated through, which is the likely reason for the similar execution time as the initial execution.
* Redefining gene2pubmed\_INNODB primary key
  + 
* Re-executing join query:
  + Table

    Description automatically generated
  + Execution Time:
    - 
  + Analyzing Query Plan
  + Graphical user interface, application, Teams

    Description automatically generated
  + Explanation:
    - This execution plan shows that gene\_info\_Memory is able to use the primary key to only iterate over 1 entry. However, gene2pubmed\_Memory is still unable to utilize the primary key, and still must iterate over 1.29 million entries, resulting in a comparatively slower execution time to the other storage engines. This is likely an artifact of the RAM storage utilized by the Memory engine.

Question 2 – Restrict

* Executing select query:
* Graphical user interface, application

  Description automatically generated
  + Execution Time:
    - 
  + Execution Plan:
    - 
    - Explanation:
      * This query filtered through 50k rows of entries, no key was used as locustag is not defined as a primary key or index, as such it must iterate through almost all of the entries. This time is comparable to the other storage engine’s initial time for execution as well.
* Creating index for locustag
  + 
  + Execution Time:
    - 
  + Re-executing select query:
    - Graphical user interface, application

      Description automatically generated
    - Execution time:
      * 
    - Execution Plan:
      * 
      * Explanation:
        + Only 2 rows were needed to be searched, the reason for this that the key gene\_info\_locustag was able to be used as a primary key. However, with the same index, the other engines were able to only check a single entry, which shows a downside for the Memory engine in terms of indexing. Regardless of this, Memory executed this statement faster than the other engines, again likely due to its RAM storage.

Question 3 - Range

* Executing select query
  + Table

    Description automatically generated
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + 50k rows needed to be searched for this query due to the fact that the primary key was unable to be used for this query. This results in the Memory engine taking close to 100 times longer to execute this query when compared to the other two engines.

Question 4 - Insert

* Executing insert statement
  + 
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + This query is interesting in that it appears that the Memory engine was able to insert without utilizing any key, and also able to do it without iterating through any disclosed rows.

Question 5 – Update

* Executing update statement
  + 
  + Execution Time:
    - 
  + Execution Plan:
    - * 
      * Explanation:
        + 2 rows needed to be iterated through in this query, which again is in contrast to the other two engines that were able to only search through a single row for their identical queries. Interestingly though, the Memory engine has the fastest execution time of the 3 engines, likely due to its utilization of RAM.

Execution Timetable:

|  |  |  |  |
| --- | --- | --- | --- |
|  | InnoDB | MyISAM | MEMORY |
| Q1 – join | 0.0147 s | 0.0005 s | 0.4081 s |
| Q2 - restrict | 0.0007 s | 0.0006 s | 0.0004 s |
| Q3 – range query | 0.0036 s | 0.0056 s | 0.2504 s |
| Q4 - insert | 0.0056 s | 0.0048 s | 0.0039 s |
| Q5 – update | 0.0490 s | 0.0356 s | 0.0171 s |

Observations:

* Overall, it appears that in this suite of experiments, that the MyISAM engine performed the best overall. When joining tables, all but the Memory engine, which stores tables in RAM, seemed to take advantage of the primary keys. As for the restrict query, all engines gain approximately the same efficiency once an index is defined for the restricted attribute. The range query once again was a disadvantage for the Memory engine, showing that it struggled to utilize the primary key to the utmost of its potential. The insert statement also saw similar performance across all engines, likely due to the fact that one of the inserted attributes was a primary key for the tables. Lastly, updating attributes had similar performances across the engines, which was likely due to the fact that this inserted attribute was assigned an index previously. However, a general observation that was noted in these experiments is that if no keys or indexes are defined, the Memory storage engine is likely the best choice. The reason for this is that it was noted that the Memory engine consistently had the fastest execution time when no keys or indexes were used for all storage engines. The likely reason for this is that it stores tables in RAM, instead of on the disk, which allows much faster read and write times. However, if primary keys and indexes are to be used in a database, it is likely not to be the best choice as there were several instances where the Memory engine was unable to take advantage of these indexes to the same degree that the other engines were able to. This contrasts with the MySAM storage engine that was extremely inefficient without any indexes or keys defined, which is likely due to the fact that it stores data in one file and indexes in another. However, with keys and indexes defined, it was the absolute fastest engine to join tables, and consistently the 2nd fastest for all other queries. To summarize, InnoDB appeared to handle all situations to a moderate degree of efficiency, where it did not suffer too bad when keys and indexes were not defined, and gained substantial speedup with keys and indexes. The MyISAM storage engine exceeds when it is supplied keys and indexes to work with, but suffers greatly without the uses of keys and indexes. Lastly, the Memory Engine did extremely well in niche queries where single values were being inserted or selected for, but suffers in comparison to the other engines when keys are defined and multiple values are being queried.

What I learned from the lab:

* The first lesson that I learned from this lab is the ability to change the storage engine used by SQL. Previously, I was unaware that this was a feature that could be changed. Coinciding with this is I learned the importance of specifying the proper storage engine for the problem space. I learned that the InnoDB engine does fairly well in all situations. I also realized that the MyISAM storage engine exceeds with keys and indexes defined, but suffers to a much higher degree than other engines when they are not defined. Lastly, I learned that the Memory engine is the fastest engine to use if keys and indexes are not going to be defined, but if keys and indexes are used, the Memory engine doesn’t benefit to the same degree as other engines.