Paige Rosynek

CS 3860 041

Professor Magana

10.27.2022

Lab 06 – Indexing

InnoDB

gene_info number of records loaded: 50000

```
mysql> Load Data local Infile
    -> 'C:/Users/rosynekp/OneDrive - Milwaukee School of Engineering/Desktop/cs 3860/labs/lab06/gene_info50000.csv' into
 table
    -> genomics.gene_info fields terminated by ',' lines terminated by '\n' ignore 1 lines;
Query OK, 50000 rows affected (0.59 sec)
Records: 50000  Deleted: 0  Skipped: 0  Warnings: 0
```

Row count:

```
mysql> select count(*) from gene_info;
+----------+
| count(*) |
+----------+
|    50000 |
+----------+
1 row in set (0.04 sec)
```

gene2pubmed number of records loaded: 12917351

```
mysql> Load Data local Infile
    -> 'C:/Users/rosynekp/OneDrive - Milwaukee School of Engineering/Desktop/cs 3860/labs/lab06/gene2pubmed' into table
    -> genomics.gene2pubmed fields terminated by '\t' lines terminated by '\n' IGNORE 1 LINES;
Query OK, 12917351 rows affected (1 min 35.35 sec)
Records: 12917351  Deleted: 0  Skipped: 0  Warnings: 0
```

Row count:

```
mysql> select count(*) from gene2pubmed;
+----------+
| count(*) |
+----------+
| 12917351 |
+----------+
1 row in set (1.22 sec)
```

InnoDB: Q1 – Join

Query:

```
select *
from gene_info gi, gene2pubmed gp
where gi.geneid=gp.geneid
and gi.geneid=4126706;
```

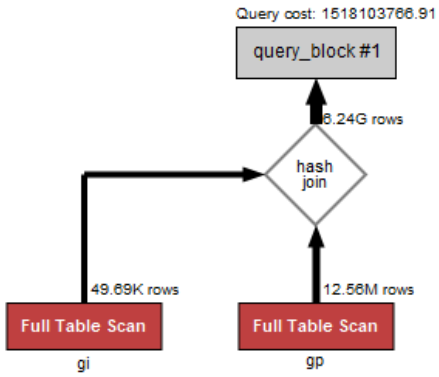
Query execution time:

Time: 12.48106875 s

11	12.48106875	select * from gene_info gi, gene2pubmed gp where gi.geneid=gp.geneid and gi.geneid=4126706 LIMIT 0, 1000

Analyze query plan:

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ALL	NULL	NULL	NULL	NULL	49686	10.00	Using where
1	SIMPLE	gp	NULL	ALL	NULL	NULL	NULL	NULL	12563732	10.00	Using where; Using join buffer (hash join)



It can be seen in the table above, the join query accessed 49686 rows from gene_info and 12563732 from gene2pubmed. In addition, no keys were used for the query because there are no constraints on either table. The time for this query was much longer than the final Q1 time because there are no constraints on either table to improve the efficiency of the query.

Adding geneid as PK for gene_info table: 1.016 sec

33	15:20:36	alter table gene_info add constraint primary key (geneid)	1.016 sec	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Adding (pmid, geneid) as PK for gene2pubmed table: 44.765 sec

34	15:23:54	alter table gene2pubmed add constraint primary key (pmid, geneid)	44.765 sec	0 row(s) affected Records: 0 Duplicates: 0 Warnings: 0

Re-execute Q1 (join) with primary key constraints on tables

Time : 3.77908575 s

2	3.77908575	select * from gene_info gi, gene2pubmed gp where gi.geneid=gp.geneid and gi.geneid=4126706 LIMIT 0, 1000

Analyze query plan after adding PK

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	gp	NULL	ALL	NULL	NULL	NULL	NULL	12891572	10.00	Using where

As shown above, the query accessed 1 row from the gene_info table and 12891572 rows from the gene2pubmed table. The keys used for the query is the primary key of the gene_info table which is geneid. As shown in the screenshot above, the query did not use the primary keys of the gene2pubmed table. It can be observed that the query used the geneid from the gene_info table and then searched the rows of the gene2pubmed table to find the matching geneid within the gene2pubmed table to then join the tables. The primary key of the gene2pubmed was not used because the query is joining the two tables were joined on the gene_info table and the order of the composite primary key constraint lists the pmid before the geneid. The time of this query was longer than the final time but faster than the no-key query run for Q1 because the keys help improve the efficiency of the lookup operation for accessing data.

Re-adding PK for gene2pubmed table:

Re-execute join query:

Time: 0.00999025 s

8	0.00999025	select * from gene_info gi, gene2pubmed gp where gi.geneid=gp.geneid and gi.geneid=4126706 LIMIT 0, 1000

Analyze query plan after changing PK for gene2pubmed

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	gp	NULL	ref	PRIMARY	PRIMARY	4	const	6	100.00	NULL

After changing the order of the gene2pubmed primary key to (geneid, pmid), the query accessed 1 row from the gene_info table and 6 rows from the gene2pubmed table. The query also used the primary keys of both tables. It can be observed that switching the order of the primary key of gene2pubmed table such that the primary keys, geneid and pmid, such that the shared key between the tables (geneid) is first improved the efficiency of the query.

InnoDB: Q2 – restriction

Query:

```
select *
from gene_info gi
where gi.locustag='p49879_1p15';
```

Time: 0.10408200 s

```
11    0.10408200    select * from gene_info gi where gi.locustag='p49879_1p15' LIMIT 0, 1000
```

Analyze Query Plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ALL	NULL	NULL	NULL	NULL	50024	10.00	Using where

As shown in the screenshot above, the query accessed 50024 rows of the gene_info table, however this value does not make sense as there are only 50000 rows in the gene_info table itself. Therefore, the row result shown may not be representative of the true number of rows accessed during the query but the results I got are most likely due to a setting in MySQL Workbench. In addition, the query did not use any primary keys because the value being searched for was not a primary key value (geneid); it was locustag. The time this query took was longer than the final time for Q2 because the table is searching for data not by the primary key.

Create gene_info index on locustag

Time: 0.281 s

```
56  16:26:33  create index gene_info_locustag on gene_info( locustag )  0.281 sec  0 row(s) affected Records: 0  Duplicates: 0  Warnings: 0
```

Re-execute query after adding index

Time: 0.00033275 s

```
20    0.00033275    select * from gene_info gi where gi.locustag='p49879_1p15' LIMIT 0, 1000
```

Analyze the query plan from re-executed query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ref	gene_info_locustag	gene_info_locustag	195	const	1	100.00	NULL

After adding the index to the gene_info table on the locus_tag field, the query accessed only 1 row of the table. The query used the index, gene_info_locustag, as the key to access data in the table. The use of the index made the query more efficient than running the query without.

InnoDB: Q3 – Range Query

Query:

```
select *
from gene_info
where geneid between '5961931' and '5999886';
```

Execute query

Time: 0.00174350 s

23	0.00174350	select * from gene_info where geneid between '5961931' and '5999886' LIMIT 0, 1000

Analyze query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gene_info	NULL	range	PRIMARY	PRIMARY	4	NULL	526	100.00	Using where

As shown in the table above, the query accessed 526 rows of the gene_info table using the primary key, geneid.

InnoDB: Q4 – insert

Query:

```
insert into gene2pubmed values (9606, 5555, 6666);
```

Execute query

Time: 0.01201800 s

26	0.01201800	insert into gene2pubmed values (9606, 5555, 6666)

Analyze query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	INSERT	gene2pubmed	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

It can be observed from the screenshot above, the insert query did not access any rows of the gene2pubmed table because it was simply adding another row to the table. Additionally, the insert query didn't use any primary keys because it doesn't access any of the rows.

InnoDB: Q5 – update

Query:

```
update gene_info set locustag='No Locus Tag' where locustag='-';
```

Execute query

Time: 0.02179875 s

29	0.02179875	update gene_info set locustag='No Locus Tag' where locustag='-'

Analyze query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	UPDATE	gene_info	NULL	range	gene_info_locustag	gene_info_locustag	195	const	1	100.00	Using where; Using temporary

As shown in the screen shot above, the update query accessed 1 row of the gene_info table. In addition, the query used the gene_info_locustag index of gene_info as the primary key.

MyISAM

Row count: gene_info

```
mysql> select count(*) from gene_info;
+----------+
| count(*) |
+----------+
|    50000 |
+----------+
1 row in set (0.00 sec)
```

Row Count: gene2pubmed

```
mysql> select count(*) from gene2pubmed;
+----------+
| count(*) |
+----------+
| 12917351 |
+----------+
1 row in set (0.00 sec)
```

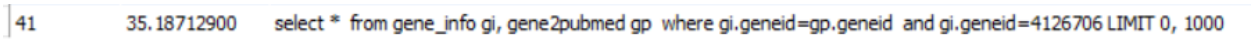
MyISAM: Q1 – Join

Query:

```
select *
from gene_info gi, gene2pubmed gp
where gi.geneid=gp.geneid
and gi.geneid=4126706;
```

Execute join query

Time: 35.18712900 s

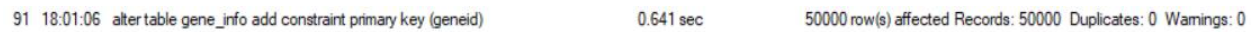


Analyze join query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ALL	NULL	NULL	NULL	NULL	50000	10.00	Using where
1	SIMPLE	gp	NULL	ALL	NULL	NULL	NULL	NULL	12917351	10.00	Using where; Using join buffer (hash join)

It is shown in the screenshot above, that the join query accessed all 50000 rows of the gene_info table and all 12917351 rows of the gene2pubmed table. It can also be observed that the query didn't use any keys to access the data which is because we have not added key constraints to the tables.

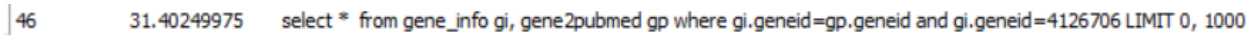
Create PK for gene_info where PK = 'geneid'



Create PK for gene2pubmed where PK = ('pmid', 'geneid')

Re-execute join query

Time: 31.40249975 s

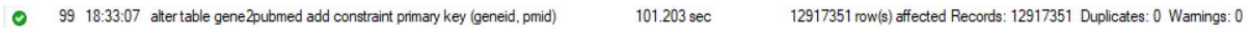


Analyze re-executed join query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	gp	NULL	ALL	NULL	NULL	NULL	NULL	12917351	10.00	Using where

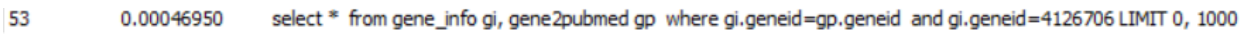
It can be observed from the table above that the join query, after adding primary keys to the tables, accessed only 1 row from gene_info and all 12917351 rows from gene2pubmed. The new join query used only the primary key from the gene_info and not the primary keys from gene2pubmed. Only the gene_info primary key needs to be used because the tables are related.

Change PK for gene2pubmed where PK = ('geneid', 'pmid')



Re-execute join query

Time: 0.00046950 s



Analyze re-executed query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	gp	NULL	ref	PRIMARY	PRIMARY	4	const	5	100.00	NULL

It can be observed from the table above, that the join query executed after changing the order of the primary key of gene2pubmed only accessed 1 row of gene_info and 5 rows of gene2pubmed. In addition, it can be observed that the primary key from each table was used to join the tables on the specified data.

<u>MyISAM: Q2 – restriction</u>

Query:

```
select *
from gene_info gi
where gi.locustag='p49879_1p15';
```

Execute restriction query

Time: 0.22882075 s

```
56          0.22882075      select *  from gene_info gi  where gi.locustag='p49879_1p15' LIMIT 0, 1000
```

Analyze restriction query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ALL	NULL	NULL	NULL	NULL	50000	10.00	Using where

It can be observed from the screenshot above, that the restriction query accessed all 50000 rows of the gene_info table because it must search for all (if existing or if more than one) occurrences of the specified locustag. The query used no keys to access the data.

Create gene_info index on locustag

Time:

```
111  18:52:52  create index gene_info_locustag on gene_info( locustag )    0.765 sec    50000 row(s) affected Records: 50000  Duplicates: 0  Warnings: 0
```

Re-execute query after adding index

Time: 0.00048800 s

```
60          0.00048800      select *  from gene_info gi  where gi.locustag='p49879_1p15' LIMIT 0, 1000
```

Analyze the query plan from re-executed query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ref	gene_info_locustag	gene_info_locustag	195	const	1	100.00	NULL

It can be observed from the table above, that the restriction query accessed only 1 row of the gene_info table after creating an index on the locustag field of gene_info. The query used the mentioned index as the key to access the specified row.

<u>MyISAM: Q3 – range query</u>

Query:

```
select *
from gene_info
where geneid between '5961931' and '5999886';
```

Execute restriction query

Time: 0.00321550 s

63	0.00321550	select * from gene_info where geneid between '5961931' and '5999886' LIMIT 0, 1000

Analyze restriction query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gene_info	NULL	range	PRIMARY	PRIMARY	4	NULL	695	100.00	Using index condition

From the table above, it can be observed that the range query accessed 695 rows of the gene_info table and used the primary key, geneid, to access rows of the data.

MyISAM: Q4 – insert

Query:

```
insert into gene2pubmed values (9606, 5555, 6666);
```

Execute insert query

Time: 0.01100975 s

66	0.01100975	insert into gene2pubmed values (9606, 5555, 6666)

Analyze insert query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	INSERT	gene2pubmed	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

It can be observed from the screenshot above, the insert query did not access any rows of the gene2pubmed table because it was simply adding another row to the table. Additionally, the insert query didn't use any primary keys because it doesn't access any of the rows.

MyISAM: Q5 – update

Query:

```
update gene_info set locustag='No Locus Tag' where locustag='-';
```

Execute update query

Time: 0.02847500 s

69	0.02847500	update gene_info set locustag='No Locus Tag' where locustag='-'

Analyze update query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	UPDATE	gene_info	NULL	range	gene_info_locustag	gene_info_locustag	195	const	1	100.00	Using where; Using temporary

As shown in the screen shot above, the update query accessed 1 row of the gene_info table. In addition, the query used the gene_info_locustag index of gene_info as the primary key.

MEMORY

Row count: gene_info

```
mysql> select count(*) from gene_info;
+----------+
| count(*) |
+----------+
|    50000 |
+----------+
1 row in set (0.00 sec)
```

Row Count: gene2pubmed

```
mysql> select count(*) from gene2pubmed;
+----------+
| count(*) |
+----------+
| 12917351 |
+----------+
1 row in set (0.00 sec)
```

MEMORY: Q1 – Join

Query:

```
select *
from gene_info gi, gene2pubmed gp
where gi.geneid=gp.geneid
and gi.geneid=4126706;
```

Execute join query

Time: 1.09355100 s

89	1.09355100	select * from gene_info gi, gene2pubmed gp where gi.geneid=gp.geneid and gi.geneid=4126706 LIMIT 0, 1000

Analyze join query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ALL	NULL	NULL	NULL	NULL	50000	10.00	Using where
1	SIMPLE	gp	NULL	ALL	NULL	NULL	NULL	NULL	12917351	10.00	Using where; Using join buffer (hash join)

It is shown in the screenshot above, that the join query accessed all 50000 rows of the gene_info table and all 12917351 rows of the gene2pubmed table. It can also be observed that the query didn't use any keys to access the data which is because we have not added key constraints to the tables.

Create PK for gene_info where PK = 'geneid'

153	20:21:25	alter table gene_info add constraint primary key (geneid)	1.360 sec	50000 row(s) affected Records: 50000 Duplicates: 0 Warnings: 0

Create PK for gene2pubmed where PK = ('pmid', 'geneid')

154	20:21:36	alter table gene2pubmed add constraint primary key (pmid, geneid)	9.906 sec	12917351 row(s) affected Records: 12917351 Duplicates: 0 Warnings: 0

Re-execute join query

Time: 0.40412800 s

94	0.40412800	select * from gene_info gi, gene2pubmed gp where gi.geneid=gp.geneid and gi.geneid=4126706 LIMIT 0, 1000

Analyze re-executed join query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	gp	NULL	ALL	NULL	NULL	NULL	NULL	12917351	10.00	Using where

It can be observed from the table above that the join query, after adding primary keys to the tables, accessed only 1 row from gene_info and all 12917351 rows from gene2pubmed. The new join query used only the primary key from the gene_info and not the primary keys from gene2pubmed. Only the gene_info primary key needs to be used because the tables are related.

Change PK for gene2pubmed where PK = ('geneid', 'pmid')

161	20:24:46	alter table gene2pubmed add constraint primary key (geneid, pmid)	10.375 sec	12917351 row(s) affected Records: 12917351	Duplicates: 0 Warnings: 0

Re-execute join query

Time: 0.41811575 s

100	0.41811575	select * from gene_info gi, gene2pubmed gp where gi.geneid=gp.geneid and gi.geneid=4126706 LIMIT 0, 1000

Analyze re-executed query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	const	PRIMARY	PRIMARY	4	const	1	100.00	NULL
1	SIMPLE	gp	NULL	ALL	PRIMARY	NULL	NULL	NULL	12917351	10.00	Using where

It can be observed from the table above, that the join query executed after changing the order of the primary key of gene2pubmed only accessed 1 row of gene_info and all 12917351 rows of gene2pubmed. In addition, it can be observed that the primary key from each table was used to join the tables on the specified data.

<u>MEMORY: Q2 – restriction</u>

Query:

```
select *
from gene_info gi
where gi.locustag='p49879_1p15';
```

Execute restriction query

Time: 0.32222900 s

103	0.32222900	select * from gene_info gi where gi.locustag='p49879_1p15' LIMIT 0, 1000

Analyze restriction query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ALL	NULL	NULL	NULL	NULL	50000	10.00	Using where

It can be observed from the screenshot above, that the restriction query accessed all 50000 rows of the gene_info table because it must search for all (if existing or if more than one) occurrences of the specified locustag. The query used no keys to access the data.

Create gene_info index on locustag

Time: 1.360 s

```
171  20:35:51  create index gene_info_locustag on gene_info( locustag )   1.360 sec   50000 row(s) affected Records: 50000  Duplicates: 0  Warnings: 0
```

Re-execute query after adding index

Time: 0.00035700 s

```
107        0.00035700      select *  from gene_info gi  where gi.locustag='p49879_1p15' LIMIT 0, 1000
```

Analyze the query plan from re-executed query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gi	NULL	ref	gene_info_locustag	gene_info_locustag	195	const	2	100.00	NULL

It can be observed from the table above, that the restriction query accessed 2 rows of the gene_info table after creating an index on the locustag field of gene_info. The query used the mentioned index as the key to access the specified row.

MEMORY: Q3 – range query

Query:

```
select *
from gene_info
where geneid between '5961931' and '5999886';
```

Execute restriction query

Time: 0.37032600 s

```
111        0.37032600      select *  from gene_info where geneid between '5961931' and '5999886' LIMIT 0, 1000
```

Analyze restriction query

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	gene_info	NULL	ALL	PRIMARY	NULL	NULL	NULL	50000	11.11	Using where

From the table above, it can be observed that the range query accessed 50000 rows of the gene_info table and used the primary key, geneid, to access rows of the data.

MEMORY: Q4 – insert

Query:

```
insert into gene2pubmed values (9606, 5555, 6666);
```

Execute insert query

Time: 0.00198175 s

115	0.00198175	insert into gene2pubmed values (9606, 5555, 6666)

Analyze insert query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	INSERT	gene2pubmed	NULL	ALL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

It can be observed from the screenshot above, the insert query did not access any rows of the gene2pubmed table because it was simply adding another row to the table. Additionally, the insert query didn't use any primary keys because it doesn't access any of the rows.

MEMORY: Q5 – update

Query:

```
update gene_info set locustag='No Locus Tag' where locustag='-';
```

Execute update query

Time: 0.01769000 s

119	0.01769000	update gene_info set locustag='No Locus Tag' where locustag='-'

Analyze update query plan

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	UPDATE	gene_info	NULL	range	gene_info_locustag	gene_info_locustag	195	const	2	100.00	Using where; Using temporary

As shown in the screen shot above, the update query accessed 2 rows of the gene_info table. In addition, the query used the gene_info_locustag index of gene_info as the primary key.

Final Time Results

	InnoDB	MyISAM	MEMORY
Q1 – join	0.00999025 s	0.00046950 s	0.41811575 s
Q2 – restrict	0.00033275 s	0.00048800 s	0.00035700 s
Q3 – range query	0.00174350 s	0.00321550 s	0.37032600 s
Q4 – insert	0.01201800 s	0.01100975 s	0.00198175 s
Q5 - update	0.02179875 s	0.02847500 s	0.01769000 s

Final Observations

It can be observed from the final table above, the join query ran the fastest using the MyISAM engine because it used the primary key for each table to join the tables on the primary key, geneid. The restrict query ran the fastest on the InnoDB engine, however all of the engines produced fairly similar results. The range query ran the fastest on the InnoDB engine because it utilizes B+ trees which are highly efficient for range queries. Memory was the slowest for the range query because data is not stored sequentially in memory, it uses a hash table to lookup data locations. The insert query ran the fastest in memory because it utilizes a hash table, so inserting data is O(1) and requires just adding another item to the hash table. The update query ran the fastest in memory because memory uses a hash table which has fast lookup that is O(1).

What I Learned & What Could be Improved

In this lab I learned the advantages and disadvantages of various NoSQL databases. I learned that memory is fast and efficient at queries that involve finding a single record because it uses a hash table to lookup locations in memory. However, memory is slow for range queries because data is not store sequentially. Additionally, I learned that InnoDB is fast and efficient for range queries because it utilizes B+ trees which are efficient for ranges. Lastly, I learned MyISAM is efficient for join queries because of its static tree implementation. One thing that I think could be improved about the lab is its repetitive nature. The steps of this lab were very repetitive and quite monotonous. I think maybe not requiring screen shots for some things might speed up the overall time spent on this lab.