CS3860 — Lab #6 — Indexing

Objectives

- Understand the effects of proper indexing on SQL query results

- Explore three major storage engines for SQL.

- Explore the connection between storage engine and index structure.

Deliverables

Complete a lab report containing the following:

- What you've learned from the lab

- How it could be improved

- A table showing the *final* time for each of five questions using each of the three storage engines.

- Observations and requested values from each question, *for each storage engine*

- Overall observations comparing the results from the table.

Storage Engines

The *storage engine* is a MySQL component that handles the physical schema. That is, the storage engine determines how the tables are stored, and how they are indexed. In this lab, we are comparing three different storage engines:

InnoDB This is the default storage engine and is intended for general-purpose use. It supports clustered indexes using a B-tree.

MyISAM was the original storage engine for MySQL. It stores each table in two files: one for the data and one for the indexes (if any).

MEMORY As the name suggests, this stores the tables in RAM, rather than on disk. Memory tables support hash indexes.

These are not the only storage engines available for MySQL.
For each engine we will look at both general speed of retrieval and at speed of access for indexes.

The Data

For this lab we will be using data first seen in Lab 1:

gene_info Entréz gene data, and

gene2pubmed Connecting genes to PubMed articles.

As noted above, the MEMORY storage engine stores its tables entirely in RAM, so we will be using

gene_info50000.csv the first 50,000 rows of gene_info for the lab.

After finishing the lab, you can try the other two storage engines with the full data set. This will really show the impact of the indexes. However, do all work for the lab itself using the 50,000-entry version so the data is consistent for comparison purposes.

What you are doing in this lab

In this lab, we will compare the performance of three storage engines in several different situations. This means that while there is a small amount of set-up we will need to do once only, most steps will be repeated three times; once for each engine.

   All of these steps are in the `genomics-db-indexing.sql` script. We can open the script in the Workbench and run each step one at a time. However, recall that we need to be running the command-line client (with the `--local-infile` flag) in order to read in `csv` files.

Only once

1. Create/re-create the genomics schema, using the Can+ or the `create database genomics;` command. You can reuse the schema if its still there, but you will need to drop the tables.

2. Read the MySQL documentation page on indexing (`http://dev.mysql.com/doc/refman/8.0/en/mysql-index.html`)

Only once (per connection)

1. *Use* the `genomics` schema.

2. Because the MEMORY engine stores all tables in memory, we need to increase our allotment. This only needs to be done once for a given connection.

```
# record variables
SHOW VARIABLES LIKE 'max_heap_table_size';
# 16777216
SHOW VARIABLES LIKE 'tmp_table_size';
# 87031808

select @@max_heap_table_size;
# 16777216
select @@tmp_table_size;
# 87031808

# set global heap size to 2G
set @@max_heap_table_size=1024 * 1024 * 1024 * 2;
# 2147483648
set @@tmp_table_size=1024 * 1024 * 1024 * 2;

#verify
select @@max_heap_table_size;
# 2147483648
select @@tmp_table_size;
# 2147483648
```

Once for each storage engine

1. **Drop the existing tables!** Drop both the `gene_info`/`gene_info2` and the `gene2pubmed` tables each time. The storage engine is set in the `CREATE TABLE` statement; if we re-use the existing table, we will reuse the existing storage engine.

2. Step through the `genomics-db-indexing.sql` script to create `gene_info` and `gene2pubmed` tables using the appropriate engine at the end of each `create table` statement.

   For example, the first time through, we will use the InnoDB Engine, so we would un-comment the `ENGINE= INNODB` line and comment out the other two engines.

```
# create gene_info
drop table if exists gene_info;
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(48),
Symbol_from_nomenclature_authority varchar(64),
Full_name_from_nomenclature_authority varchar(256),
Nomenclature_status varchar(24),
Other_designations varchar(4000),
Modification_date varchar(24)) ENGINE=INNODB;
#Modification_date varchar(24)) ENGINE=MYISAM;
#Modification_date varchar(24)) ENGINE=MEMORY;
```

For the second pass, we will comment out `ENGINE=INNODB` and un-comment `ENGINE=MYISAM`. For the third pass, we will use `ENGINE=MEMORY`

Note the drop table statement at the top—don't forget to drop the table before creating it!

3. Import the data into the table. Because we are reading a local input file, we must run this from the command line using `--local-infile`. Use the import statement from the script, modified to refer to the location where your `csv` file is located.

4. Step through the queries making up the five questions. Pay attention to the comments; record any data you are asked to "Note," answer any questions that are asked, and make observations as requested.

   For example, Q1 asks you to compare times for the same query with no index, and with two different indexes. Record all three times, and comment about why you think each of the times were faster than/slower than/the same as the others.

   Similarly, some steps ask you to analyze a query (plan) using explain (see Appendix B). Look at the explanation, describe (in your own words) what the query is doing, and explain why it would be doing that, given what you know about the query and any indexing.

   Toward the end of each question, you will be asked to record a time in the table. Put those times in your table (and no others).

   N.B. For many of the observations of query time, the query is being profiled (see Appendix A). Use the time reported in the profile, not the time reported by the client.

Once you finish these steps for one storage engine; go back and repeat them for the next.

The Table

In this lab, you will make a table of execution times, with columns for each storage engine, and rows for each of the five questions.

	InnoDB	MyISAM	MEMORY
Q1 - join			
Q2 - restrict			
Q3 - range query			
Q4 - insert			
Q5 - update			

For each entry, record the time that is specified as going in the table. Make sure you use the time reported in the profile.

After completing the table, you may wish to make observations about the relative strengths of the different engines based on its contents.

Paragraph each, topic will be on final.

Appendices

A Profiling

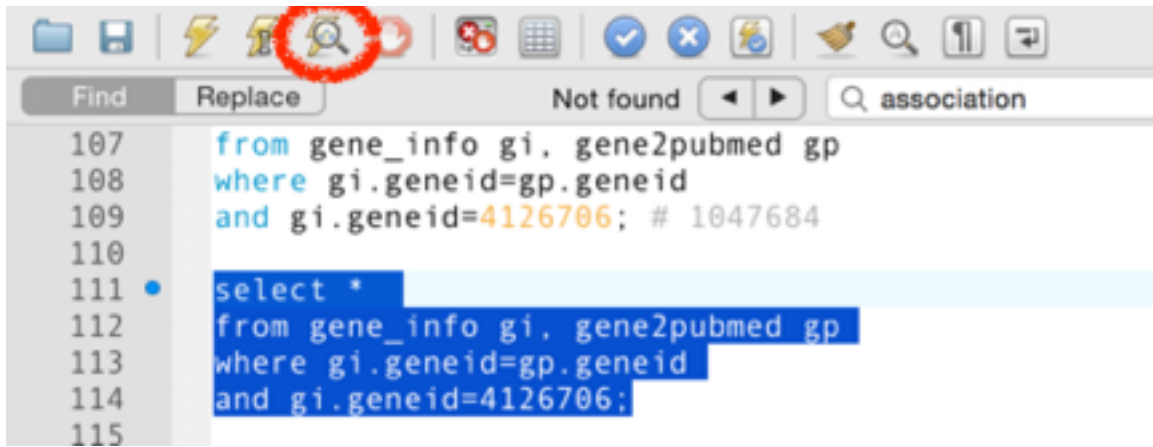Some queries contain a series of commands, such as:

```
# first, set profiling on
set profiling=0; # off
set profiling=1; # on
select *
from gene_info gi, gene2pubmed gp
where gi.geneid=gp.geneid
and gi.geneid=4126706;
# list each query with duration
show profiles;
```

Turning on profiling causes MySQL to record detailed execution/timing information about all queries. When we ask it to `show profiles` it will report that information for all queries that occurred while profiling was on.

For lab purposes, this allows us to record more precise execution times that we could get from the workbench or the command-line client directly. Please use the more precise times in your report.

While profiles are deprecated, we will continue to use them for this lab.

B Explaining Queries

When MySQL executes a query, the query evaluator creates and runs a *query plan*, describing which tables it is using where, and how it is looking thing up in them. Conventionally, the `explain` command will cause MySQL to present its query plan.

So running:

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

will produce the following results:

Result Grid		Filter Rows:	Q Search		Export:						
id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
▶ 1	SIMPLE	gi	NULL	ALL	NULL	NULL	NULL	NULL	50268	10.00	Using where
1	SIMPLE	gp	NULL	ALL	NULL	NULL	NULL	NULL	12589307	10.00	Using where; Using join buffer (Block Nested Loop)

Result 1   ⓘ Read Only

The query went through 50000 rows of the `gi` (`gene_info`) table and 1.29 million rows of `gp` (`gene2pubmed`),[1] using a nested loop to join the two tables.

We can get the same effect by highlighting the query (without the `explain`) and selecting the "lightning-bolt-with-magnifying-glass" icon in the Workbench.

---

[1]That's a lot of rows!

```
107     from gene_info gi, gene2pubmed gp
108     where gi.geneid=gp.geneid
109     and gi.geneid=4126706; # 1047684
110
111  •  select *
112     from gene_info gi, gene2pubmed gp
113     where gi.geneid=gp.geneid
114     and gi.geneid=4126706;
115
```

When doing so, we can get the explain data as before:

But we can use the tabs to alternately see a graphical representation of the query plan:

or a detailed description:

Of course, the information shown by all three is the same.