Northeastern University
CS5200 – DBMS
Spring 2025
Professor Derbinsky
Erdun E
Mar 03, 2025

Homework 5 Analysis

The goal of this assignment is for you to empirically discover and justify the guidelines regarding usage of indexes within a large database application. Towards this end, you will be using the MariaDB relational database management system as an experimental platform: for a large data set provided to you, you will be measuring the amount of time and disk space various database operations require given different combinations of indexes. All work in this assignment should be done using the `MariaDB Console` application.

For this assignment, you are to submit a **single PDF file** with all of your data as well as written analysis in response to questions posed in the assignment. You will be evaluated on the completeness of the data, the correctness/insightfulness of your analysis (with respect to your data), as well as the professionalism of your writeup (including aesthetics, spelling/grammar, and organization). You must typeset all responses – hand-written/drawn work will receive **0% credit**.

START THIS ASSIGNMENT EARLY. Much of the time involved goes to collecting data in various indexing situations, and to collect clean data your computer should not be working on other tasks (e.g. web browsing). Give yourself plenty of time to gather this data.

1 Preliminaries

For this assignment you will only need to run the MariaDB Server, not Apache.

2 Using the MariaDB Console

The console application is named `mysql` (or `mysql.exe` on Windows). You should start the terminal program on your platform (command prompt, or `cmd` on Windows), and navigate to the `bin` directory under your XAMPP installation[1]. Once there, execute the following command[2]: `mysql -u root` and you should be brought to a MariaDB prompt.

Unfortunately the console application does not have autocompletion for file/directory names, and so you should place all files required for this assignment in an easy-to-type location, such as `/tmp` (or `c:\temp` on Windows).

When you are done, you can type `quit` to exit the console.

[1] Defaults: Windows (`c:\xampp\mysql\bin`), Mac (`/Applications/XAMPP/bin`)
[2] For Mac: `./mysql -u root`

3 Data Set

Provided with this assignment is a base schema (`schema.sql`), reproduced below:

```
CREATE TABLE data (
    a0 INT,
    a1 INT PRIMARY KEY,
    a2 INT,
    a5 INT,
    a10 INT,
    a100 INT,
    a1000 INT
);
```

The data that will populate this schema has been specially constructed for this assignment:

- The value of `a0` is always 0.
- The value of `a1` is unique (first row = 1, second row = 2, ...).
- For all remaining fields `a`n, the values repeatedly count from 1 ... n.

3.1 Loading the Dataset

At the beginning of this assignment, you should create a new database (e.g. `hw5`) and then switch to using it:

```
CREATE DATABASE hw5;
USE hw5;

MariaDB [(none)]> CREATE DATABASE hw5;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> use hw5
Database changed
MariaDB [hw5]>
```

You should then load the base schema to create the `data` table (use forward slashes in the path, even on Windows):

```
SOURCE path/to/schema.sql;

MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/schema.sql;
Query OK, 0 rows affected (0.015 sec)
```

Finally, load the data into the table. The data we will use will include 10 million rows and is thus relatively large. You will need to uncompress the supplied data file, and then source just as with the schema:

```
SOURCE path/to/data_10m_10k.sql;
```

```
MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/data_10m_10k.sql;
Query OK, 0 rows affected (0.001 sec)

. . . (Last 5 Queries)

Query OK, 10000 rows affected (0.023 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.023 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.022 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.034 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.021 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.001 sec)

MariaDB [hw5]>
```

The commands in this file are a set of `INSERT` statements, each adding 10,000 rows (i.e., 0.1%) to the `data` table. After each `INSERT` executes, you should see something like the following:

```
Query OK, 10000 rows affected (0.034 sec)
Records: 10000  Duplicates: 0  Warnings: 0
```

If your insert time is consistently greater than 1 sec, stop the process (`CTRL-C`). You likely have another disk-intensive process running and/or an anti-virus. Identify and stop the offending program and restart.

Once the inserts complete, confirm that all the data was inserted successfully via the following queries (note that each will take approximately 1–2 seconds or more, basically just to perform a table scan):

```
SELECT COUNT(*) FROM data;     -- 10,000,000
```

```
MariaDB [hw5]> SELECT COUNT(*) FROM data;
+----------+
| COUNT(*) |
+----------+
| 10000000 |
+----------+
1 row in set (1.143 sec)

SELECT SUM(a0) FROM data;      -- 0
MariaDB [hw5]> SELECT SUM( a0 ) FROM data ;
+------------+
| SUM( a0 ) |
+------------+
|          0 |
+------------+
1 row in set (1.436 sec)

SELECT SUM(a2) FROM data;      -- 15,000,000
MariaDB [hw5]> SELECT SUM( a2 ) FROM data ;
+------------+
| SUM( a2 ) |
+------------+
|  15000000 |
+------------+
1 row in set (1.507 sec)

SELECT SUM(a10) FROM data;     -- 55,000,000
MariaDB [hw5]> SELECT SUM( a10 ) FROM data ;
+-------------+
| SUM( a10 ) |
+-------------+
|   55000000 |
+-------------+
1 row in set (1.534 sec)

SELECT SUM(a100) FROM data;    -- 505,000,000
MariaDB [hw5]> SELECT SUM( a100 ) FROM data ;
+--------------+
| SUM( a100 ) |
+--------------+
|   505000000 |
+--------------+
1 row in set (1.553 sec)

SELECT SUM(a1000) FROM data;   -- 5,005,000,000
```

```
MariaDB [hw5]> SELECT SUM( a1000 ) FROM data ;
+---------------+
| SUM( a1000 ) |
+---------------+
|   5005000000 |
+---------------+
1 row in set (1.520 sec)
```

Right now this table has no indexes (other than the automatic primary key on `a1`), so we will use these insert times as a baseline for later comparison. Take the average of the reported time (in the example above, 0.034 sec) for the last 5 queries and enter that number below:

$$\frac{0.023 + 0.023 + 0.022 + 0.034 + 0.021}{5} = 0.0246$$

DP_INSERT_TIME_0 0.0246 **seconds/10,000 rows**

We also need to record the approximate space being used by the table – again, having minimal indexes, this measurement will serve as a baseline for future comparison. We will query the MariaDB catalog for this information using the following query:

```
SELECT ((data_length + index_length) / POWER(1024, 2)) AS
   tablesize_mb
FROM information_schema.tables
WHERE table_schema='hw5' AND table_name='data';

MariaDB [hw5]> SELECT ((data_length + index_length) / POWER(1024, 2)
   ) AS tablesize_mb
    -> FROM information_schema.tables
    -> WHERE table_schema='hw5' AND table_name='data';
+---------------+
| tablesize_mb |
+---------------+
|    488.96875 |
+---------------+
1 row in set (0.014 sec)
```

The query adds memory used for the table itself, as well as any indexes, and divides by an appropriate constant to convert to MB from bytes. Record the result of your query:

DP_SPACE_0 488.96875 **MB**

3.2 Selectivity Analysis and `EXPLAIN`

For this data set you know that, by construction, each field has a fixed cardinality (i.e. number of distinct values). In general, however, you will not know this fact ahead of time. In the space below,

write the SQL query that gets the number of distinct values for the `a2` column:

```
SELECT COUNT(*) FROM data WHERE a2 = 1;
```

Now run this query three times and record the average time below:

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (1.381 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (1.402 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (1.361 sec)

MariaDB [hw5]>
```

$$\frac{1.381 + 1.402 + 1.361}{3} = 1.381$$

DP_DISTINCT_TIME_0(a2) 1.381 **seconds**

Now modify this query to get the number of distinct values for the `a1000` column, run three times, and compare to **DP_DISTINCT_TIME_0** – these should not be radically different.

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
```

```
+----------+
1 row in set (1.319 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (1.323 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (1.326 sec)

MariaDB [hw5]>
```

$$\frac{1.319 + 1.323 + 1.326}{3} = 1.323$$

DP_DISTINCT_TIME_0(a1000) _____ 1.323 **seconds**

We can confirm this last hypothesis by looking at how MariaDB executed these queries via the `EXPLAIN` command (simply type `EXPLAIN` before your query and execute). The output should look something like the following (the value for `rows` may be slightly different – it's just a heuristic estimate by the DBMS):

```
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+
|  1 | SIMPLE      | data  | ALL  | NULL          | NULL | NULL    | NULL | 9719061 |       |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------+


MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1;
+------+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id   | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra
|
+------+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|    1 | SIMPLE      | data  | ALL  | NULL          | NULL | NULL    | NULL | 9719061 | Using where |
+------+-------------+-------+------+---------------+------+---------+------+---------+-------------+
1 row in set (0.002 sec)

MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a1000 = 1;
+------+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id   | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra
|
+------+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|    1 | SIMPLE      | data  | ALL  | NULL          | NULL | NULL    | NULL | 9719061 | Using where |
```

```
+------+-------------+-------+------+---------------+------+---------+------+---------+-------------+
1 row in set (0.001 sec)
```

The takeaways from this output are that the query is scanning the `data` table and not making use of any index. By contrast, if you execute the same query for the `a1` column, your timing data should be different and the output of the `EXPLAIN` command will look something like the following:

```
+----+-------------+------+------+---------------+--------+---------+------+----------+-------------+
| id | select_type | table| type | possible_keys| key    | key_len | ref  | rows     | Extra       |
+----+-------------+------+------+---------------+--------+---------+------+----------+-------------+
|  1 | SIMPLE      | data | index| NULL          | PRIMARY| 4       | NULL | 9719061  | Using index|
+----+-------------+------+------+---------------+--------+---------+------+----------+-------------+

MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a1 = 1;
+----+-------------+------+------+---------------+--------+---------+-------+------+-------------+
| id | select_type | table| type | possible_keys| key    | key_len | ref   | rows | Extra       |
+----+-------------+------+------+---------------+--------+---------+-------+----------+--------+
|  1 | SIMPLE      | data | index| NULL          | PRIMARY| 4       | const | 1    | Using index|
+----+-------------+------+------+---------------+--------+---------+-------+------+-------------+
1 row in set (0.002 sec)
```

MariaDB is reporting that for the non-indexed attributes (at this point, all of them aside from `a1`), the query was executed using a *table scan*. For the primary key, however, an index was used (though every row still had to be touched to generate the result set).

3.3 Baseline Query Performance

Now we are going to obtain query performance for a set of simple queries with single-attribute and multi-attribute `WHERE` clauses. Later we will compare these results with various indexing schemes.

All of our test queries will be aggregations, in order to control for large/different result set sizes. The basic format will be...

```
SELECT AVG(a1), COUNT(*) FROM data WHERE ...
```

For each set of attribute=value pairs below, perform the corresponding query three times and report the average query time. In each case, execute the query above, but with the `WHERE` condition indicated, noting that {x=a, y=b} should be interpreted as **x=a AND y=b**.

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.003 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a1 = 1;
+---------+----------+
```

```
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.001 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.001 sec)
```

$$\frac{0.003 + 0.001 + 0.001}{3} = 0.00167$$

a1 = 1 $\underline{\qquad\qquad 0.00167}$ **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 5000000.0000 |  5000000 |
+--------------+----------+
1 row in set (1.607 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 5000000.0000 |  5000000 |
+--------------+----------+
1 row in set (1.600 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 5000000.0000 |  5000000 |
+--------------+----------+
1 row in set (1.605 sec)
```

$$\frac{1.607 + 1.600 + 1.605}{3} = 1.604$$

a2 = 1 $\underline{\qquad\qquad 1.604}$ **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999996.0000 |  1000000 |
+--------------+----------+
1 row in set (1.455 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999996.0000 |  1000000 |
+--------------+----------+
1 row in set (1.435 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999996.0000 |  1000000 |
+--------------+----------+
1 row in set (1.440 sec)
```

$$\frac{1.455 + 1.435 + 1.440}{3} = 1.443$$

a10 = 1 $\underline{\qquad\qquad 1.443}$ **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999951.0000 |   100000 |
+--------------+----------+
1 row in set (1.384 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999951.0000 |   100000 |
+--------------+----------+
1 row in set (1.378 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a100 = 1;
```

```
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999951.0000 |   100000 |
+--------------+----------+
1 row in set (1.354 sec)
```

$$\frac{1.384 + 1.378 + 1.354}{3} = 1.372$$

a100 = 1 $\underline{\qquad\qquad 1.372}$ **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a1000 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.378 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a1000 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.391 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a1000 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.410 sec)
```

$$\frac{1.378 + 1.391 + 1.410}{3} = 1.393$$

a1000 = 1 $\underline{\qquad\qquad 1.393}$ **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
```

```
+---------+----------+
1 row in set (0.004 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.002 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.001 sec)
```

$$\frac{0.004 + 0.002 + 0.001}{3} = 0.0023$$

a2 = 1, a1 = 1 $\underline{\qquad\qquad 0.0023}$ **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999996.0000 |  1000000 |
+--------------+----------+
1 row in set (1.745 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999996.0000 |  1000000 |
+--------------+----------+
1 row in set (1.740 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a10 = 1;
```

```
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999996.0000 |  1000000 |
+--------------+----------+
1 row in set (1.775 sec)
```

$$\frac{1.745 + 1.740 + 1.775}{3} = 1.753$$

a2 = 1, a10 = 1 1.753 **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999951.0000 |   100000 |
+--------------+----------+
1 row in set (1.852 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999951.0000 |   100000 |
+--------------+----------+
1 row in set (1.687 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999951.0000 |   100000 |
+--------------+----------+
1 row in set (1.731 sec)
```

$$\frac{1.852 + 1.687 + 1.731}{3} = 1.757$$

a2 = 1, a100 = 1 1.757 **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1;
+--------------+----------+
```

```
| AVG(a1)       | COUNT(*) |
+---------------+----------+
| 4999501.0000 |    10000 |
+---------------+----------+
1 row in set (1.747 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1;
+---------------+----------+
| AVG(a1)       | COUNT(*) |
+---------------+----------+
| 4999501.0000 |    10000 |
+---------------+----------+
1 row in set (1.692 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1;
+---------------+----------+
| AVG(a1)       | COUNT(*) |
+---------------+----------+
| 4999501.0000 |    10000 |
+---------------+----------+
1 row in set (1.721 sec)
```

$$\frac{1.747 + 1.692 + 1.721}{3} = 1.720$$

a2 = 1, a1000 = 1 $\underline{\qquad\qquad 1.720}$ **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.002 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.001 sec)
```

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a1 = 1;
+---------+----------+
| AVG(a1) | COUNT(*) |
+---------+----------+
|  1.0000 |        1 |
+---------+----------+
1 row in set (0.001 sec)
```

$$\frac{0.002 + 0.001 + 0.001}{3} = 0.0013$$

a2 = 1, a1000 = 1, a1 = 1 0.0013 **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.761 sec)
```

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.789 sec)
```

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a10 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.781 sec)
```

$$\frac{1.761 + 1.789 + 1.781}{3} = 1.777$$

a2 = 1, a1000 = 1, a10 = 1 1.777 **seconds**

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.756 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.771 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND
   a1000 = 1 AND a100 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 4999501.0000 |    10000 |
+--------------+----------+
1 row in set (1.807 sec)
```

$$\frac{1.756 + 1.771 + 1.807}{3} = 1.778$$

a2 = 1, a1000 = 1, a100 = 1	1.778 **seconds**

You shouldn't encounter wildly different times between runs for the same query. If you do, stop any other processes on your machine that may be diverting CPU usage and retry. If you are surprised by a result, run the `EXPLAIN` command to understand the query plan.

3.3.1 Analysis

Based upon these results, what are your preliminary conclusions regarding query performance on this data set? You should comment on characteristics including prime/non-prime, selectivity, query plan (i.e. the result of using the `EXPLAIN` statement), and query length (i.e. number of attributes in the query). You are welcome to report additional results/analysis to bolster your conclusions.

Prime vs. Non-Prime Attributes
The primary key column `a1` is indexed by default, while other attributes `a2`, `a10`, `a100`, `a1000` are not. This distinction significantly affects query performance. Queries involving `a1` can be executed

efficiently with index-based lookups, while queries involving non-indexed columns require a full table scan.
For example, executing:

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a1 = 1;
```

produced an execution time of:

$$\frac{0.003 + 0.001 + 0.001}{3} = 0.00167 \text{ seconds}$$

And, querying a non-indexed attribute, such as:

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
```

resulted in a significantly higher execution time:

$$\frac{1.607 + 1.600 + 1.605}{3} = 1.604 \text{ seconds}$$

```
MariaDB [hw5]> Explain SELECT AVG(a1), COUNT(*) FROM data WHERE a1 = 1;
+---+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+
| id| select_type | table | type  | possible_keys | key     | key_len | ref   | rows | Extra       |
+---+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+
| 1 | SIMPLE      | data  | const | PRIMARY       | PRIMARY | 4       | const | 1    | Using index |
+---+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+
1 row in set (0.007 sec)

MariaDB [hw5]> Explain SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
+---+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id| select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra       |
+---+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| 1 | SIMPLE      | data  | ALL  | NULL          | NULL | NULL    | NULL | 9719061 | Using where |
+---+-------------+-------+------+---------------+------+---------+------+---------+-------------+
1 row in set (0.001 sec)
```

This indicate the importance of indexing that the query on the primary key column `a1` was approximately is significantly faster than the same query on the non-indexed column `a2`. The `EXPLAIN` statement further confirms that queries on `a1` utilize an indexed search, whereas queries on `a2` trigger a full table scan.

Selectivity and Query Performance
Selectivity refers to the number of unique values in a column. A lower selectivity means fewer distinct values, resulting in more rows matching a given condition, leading to higher execution times. To analyze this, we computed the number of matching rows for `a2` and `a1000`:

```
SELECT COUNT(*) FROM data WHERE a2 = 1;
```

Returned:

```
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
```

$$\frac{1.381 + 1.402 + 1.361}{3} = 1.381 \text{ seconds}$$

indicating that `a2` has very low selectivity, meaning a query filtering on `a2` will return approximately half of the dataset. And, `a1000` has higher selectivity:

```
SELECT COUNT(*) FROM data WHERE a1000 = 1;
```

Returned:

```
+-----------+
| COUNT(*)  |
+-----------+
|     10000 |
+-----------+
```

$$\frac{1.319 + 1.323 + 1.326}{3} = 1.323 \text{ seconds}$$

Comparing query execution times:

$$\textbf{a2} = \textbf{1: 1.381} \text{ seconds}, \quad \textbf{a1000} = \textbf{1: 1.323} \text{ seconds}$$

We observe that the high-selectivity column `a1000` has a lower execution time due to fewer matching rows.

Query Execution Plan

To confirm this, we analyzed how MariaDB executed these queries using the `EXPLAIN` statement. For `a2 = 1` and `a1000 = 1`, the results were:

```
MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1;
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|  1 | SIMPLE      | data  | ALL  | NULL          | NULL | NULL    | NULL | 9719061 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
1 row in set (0.002 sec)

MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows    | Extra       |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
|  1 | SIMPLE      | data  | ALL  | NULL          | NULL | NULL    | NULL | 9719061 | Using where |
+----+-------------+-------+------+---------------+------+---------+------+---------+-------------+
1 row in set (0.001 sec)
```

This indicates that both queries used a full table scan rather than an index.
By contrast, executing the same query on the primary key a1 shows a different execution plan:

```
MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a1 = 1;
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref   | rows | Extra       |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+
|  1 | SIMPLE      | data  | const | PRIMARY       | PRIMARY | 4       | const | 1    | Using index |
+----+-------------+-------+-------+---------------+---------+---------+-------+------+-------------+
1 row in set (0.002 sec)
```

This confirms that:

- Queries on a2 and a1000 both used a full table scan, resulting in longer execution times.
- The primary key a1 utilized an index, making the query significantly faster.

Query Length

Query length refers to the number of conditions present in the `WHERE` clause. In general, increasing the number of conditions can impact performance due to increased row filtering complexity and additional comparisons required during query execution. We conducted experiments with single-attribute and multi-attribute queries to analyze their performance.

Single-Attribute Query Performance:

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
```

Returned:

```
+---------------+-----------+
| AVG(a1)       | COUNT(*)  |
+---------------+-----------+
| 5000000.0000  |   5000000 |
+---------------+-----------+
```

$$\frac{1.607 + 1.600 + 1.605}{3} = 1.604 \text{ seconds}$$

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a1000 = 1;
```

Returned:

```
+---------------+-----------+
| AVG(a1)       | COUNT(*)  |
+---------------+-----------+
| 4999501.0000  |     10000 |
+---------------+-----------+
```

$$\frac{1.378 + 1.391 + 1.410}{3} = 1.393 \text{ seconds}$$

Comparing execution times:

$$\textbf{a2 = 1: 1.604 seconds,} \quad \textbf{a1000 = 1: 1.393 seconds}$$

The higher-selectivity column `a1000` resulted in a lower execution time due to fewer matching rows.

Multi-Attribute Query Performance

To analyze the effect of increasing query length, we tested multi-attribute queries:

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
```

Returned:

```
+---------------+-----------+
| AVG(a1)       | COUNT(*)  |
+---------------+-----------+
| 4999996.0000  |   1000000 |
+---------------+-----------+
```

$$\frac{1.745 + 1.740 + 1.775}{3} = 1.753 \text{ seconds}$$

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND a100 = 1;
```

Returned:

```
+---------------+-----------+
| AVG(a1)       | COUNT(*)  |
+---------------+-----------+
| 4999951.0000  |    100000 |
+---------------+-----------+
```

$$\frac{1.852 + 1.687 + 1.731}{3} = 1.757 \text{ seconds}$$

Comparing execution times:

a2 = 1 AND a10 = 1: 1.753 seconds, a2 = 1 AND a100 = 1: 1.757 seconds

Queries involving multiple conditions took longer to execute than single-attribute queries. This suggests that adding more conditions in a table scan increases execution time, as more comparisons must be evaluated for each row.

Effect of Indexing on Query Length

To further investigate the impact of indexing, we tested queries including the primary key:

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1 AND a1 = 1;
```

Returned:

```
+---------+-----------+
| AVG(a1) | COUNT(*)  |
+---------+-----------+
|  1.0000 |         1 |
+---------+-----------+
```

$$\frac{0.004 + 0.002 + 0.001}{3} = 0.0023 \text{ seconds}$$

Comparing execution times:

a2 = 1 AND a1 = 1: 0.0023 seconds

This result demonstrates that:

- Queries that include the primary key (`a1`) execute significantly faster because the database utilizes an index.
- A longer query does not necessarily mean worse performance if the indexed column is involved.

After compared and analyzed we have the conclusions:

- Queries with more attributes in the `WHERE` clause generally take longer, as they require filtering a larger dataset.
- If the query includes an indexed attribute, execution is significantly faster, regardless of length.
- Without indexing, query length and execution time have a near-linear relationship due to table scans.

4 Adding Indexes

We begin with single-attribute indexes, and then move into multiple indexes and multi-column indexes.

4.1 Individual Single-Attribute Indexes

We are now ready to add an index and assess its impact. The goal will be to populate the following table with experimental results:

	Insert Time (sec/10k rows)	**Space on Disk** (MB)	**Query Time** (seconds)
a2	0.0296	629.671875	0.655
a10	0.0296	580.671875	0.183
a100	0.1362	614.703125	0.040
a1000	0.3684	636.703125	0.011

To obtain these results, we are going to take a sequence of steps. This text will detail the steps for `a2` (the first row), then you will repeat for the remaining attributes (in each subsequent row).

4.1.1 Clear the Table

To see what impact the index will have on time to insert the rows, first drop the table (`DROP TABLE data;`) – this may take a minute or two. Now re-create the table using the schema file as we did at the beginning of the assignment.[3]

```
MariaDB [hw5]> DROP TABLE IF EXISTS data;
Query OK, 0 rows affected (0.006 sec)

MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/schema.sql;
Query OK, 0 rows affected (0.014 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.001 sec)
```

4.1.2 Create the Index

Add a new index on the column of interest (`CREATE INDEX foo ON data(a2);`).

```
MariaDB [hw5]> CREATE INDEX idx_a2 ON data(a2);
Query OK, 0 rows affected (0.022 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [hw5]> SHOW INDEX FROM data;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
data	0	PRIMARY	1	a1	A	0	NULL	NULL		BTREE		
data	1	idx_a2	1	a2	A	0	NULL	NULL	YES	BTREE		

```
2 rows in set (0.003 sec)
```

4.1.3 Input the Data

Execute the `SOURCE` command above for the 10 million rows of data. Remember to (1) take the average of the last 5 batches of 10k rows, and make this the result in the second column of the table above and (2) perform the catalog-space query from above to fill in the third column in the

[3] You could also just delete the data in the table via the `DELETE` command, but often this takes more time than just dropping the table as a whole.

table. It would aid your analysis to note the percentage increase in time between these values and **DP_INSERT_TIME_0**/**DP_SPACE_0** above.

```
Query OK, 10000 rows affected (0.027 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.028 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.027 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.038 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.028 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.001 sec)

MariaDB [hw5]>
```

$$\frac{0.027 + 0.028 + 0.027 + 0.038 + 0.028}{5} = 0.0296$$

DP_INSERT_TIME_a2 $\underline{\qquad 0.0296 \qquad}$ **seconds/10,000 rows**

```
MariaDB [hw5]> SELECT ((data_length + index_length) / POWER(1024, 2)
   ) AS tablesize_mb
    -> FROM information_schema.tables
    -> WHERE table_schema='hw5' AND table_name='data';
+---------------+
| tablesize_mb |
+---------------+
|   629.671875 |
+---------------+
1 row in set (0.008 sec)
```

DP_SPACE_2a $\underline{\qquad 629.671875 \qquad}$ **MB**

Insertion Time Analysis

To measure the impact of indexing on insertion time, we compare the average insertion time per 10,000 rows before and after adding the index.

Baseline insertion time (without index):

$$\textbf{DP\_INSERT\_TIME\_0} = \textbf{0.0246} \text{ seconds}$$

Insertion time after adding index on a2:

$$\textbf{DP\_INSERT\_TIME\_a2} = \textbf{0.0296} \text{ seconds}$$

Percentage increase in insertion time:

$$\frac{\textbf{DP\_INSERT\_TIME\_a2 - DP\_INSERT\_TIME\_0}}{\textbf{DP\_INSERT\_TIME\_0}} \times 100\% = \frac{\textbf{0.0296 - 0.0246}}{\textbf{0.0246}} \times 100\% = \textbf{20.33}\%$$

This indicates that inserting data into the indexed column `a2` is 20.33% slower compared to when there was no index.

Disk Space Analysis
Next, we analyze how adding the index affects the storage space used by the table.
Baseline table size (without index):

$$\textbf{DP\_SPACE\_0} = \textbf{488.96875} \text{ MB}$$

Table size after adding index on a2:

$$\textbf{DP\_SPACE\_a2} = \textbf{629.671875} \text{ MB}$$

Percentage increase in table size:

$$\frac{\textbf{DP\_SPACE\_a2 - DP\_SPACE\_0}}{\textbf{DP\_SPACE\_0}} \times 100\% = \frac{\textbf{629.671875 - 488.96875}}{\textbf{488.96875}} \times 100\% = \textbf{28.75}\%$$

Adding an index on `a2` increased the table size by 28.75%, which is expected since indexes require additional storage to maintain sorted structures for efficient lookups.

4.1.4 Perform Baseline Query

Check to see how the index changes the time to run your cardinality query (look back and compare to **DP_DISTINCT_TIME_0**) and single-attribute query (e.g. `WHERE a2=1`) – remember to average over three runs (the first may be large, due to loading from memory). You may also retry other queries that contain the attribute and see how they change as a result of the query – in cases that interact with the primary key, look at the output of `EXPLAIN` to see which index the query optimizer chose to utilize (see the key column).

```
SELECT COUNT(*) FROM data WHERE a2 = 1;
```

Before adding the index:

$$\textbf{DP\_DISTINCT\_TIME\_0} \quad \underline{\qquad\qquad 1.381} \textbf{ seconds}$$

After adding the index:

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.756 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.600 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.609 sec)
```

$$\frac{0.756 + 0.600 + 0.609}{3} = 0.655 \text{ seconds}$$

DP_DISTINCT_TIME_a2 0.655 **seconds**

Comparing query execution times:

$$\text{Improvement } \% = \left(\frac{\text{DP\_DISTINCT\_TIME\_0} - \text{DP\_DISTINCT\_TIME\_a2}}{\text{DP\_DISTINCT\_TIME\_0}} \right) \times 100$$

$$= \left(\frac{1.381 - 0.655}{1.381} \right) \times 100 = 52.6\%$$

Performance Improvement 52.6%

To verify that the optimizer utilized the index, we ran the `EXPLAIN` statement:

```
EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1;
```

```
+----+-------------+-------+------+---------------+--------+---------+-----+---------+------------+
| id | select_type | table | type | possible_keys | key    | key_len |ref  | rows    |Extra       |
+----+-------------+-------+------+---------------+--------+---------+-----+---------+------------+
|  1 | SIMPLE      | data  | ref  | idx_a2        | idx_a2 | 5       |const| 4859530 |Using index |
+----+-------------+-------+------+---------------+--------+---------+-----+---------+------------+
```

This confirms that:

- The optimizer used the index `idx_a2`.
- The query type changed from a full table scan (`ALL`) to an indexed lookup (`ref`).
- The `Extra` column shows `Using index`, meaning MariaDB optimized the lookup.

We also tested how the index affects the execution time of a query filtering on `a2`:

```
SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
```

Before adding the index:

DP_QUERY_TIME_0(a2) _____ 1.604 **seconds**

After adding the index:

```
MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 5000000.0000 |  5000000 |
+--------------+----------+
1 row in set (0.862 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 5000000.0000 |  5000000 |
+--------------+----------+
1 row in set (0.790 sec)

MariaDB [hw5]> SELECT AVG(a1), COUNT(*) FROM data WHERE a2 = 1;
+--------------+----------+
| AVG(a1)      | COUNT(*) |
+--------------+----------+
| 5000000.0000 |  5000000 |
+--------------+----------+
1 row in set (0.783 sec)
```

$$\frac{0.862 + 0.790 + 0.783}{3} = 0.812 \text{ seconds}$$

DP_QUERY_TIME_a2 _____ 0.812 **seconds**

Comparing query execution times:

$$\text{Improvement } \% = \left(\frac{\text{DP\_QUERY\_TIME\_0(a2)} - \text{DP\_QUERY\_TIME\_a2}}{\text{DP\_QUERY\_TIME\_0(a2)}} \right) \times 100$$

$$= \left(\frac{1.604 - 0.812}{1.604}\right) \times 100 = 49.4\%$$

Performance Improvement 49.4%

4.1.5 Analysis

Repeat these steps for the remaining attributes (remembering to change `a2` in each situation for the attribute in row of the table) and then analyze the results. Your analysis should focus on the relative costs of adding a single index as compared to query benefits. Did there seem to be a trend as to which indexes cost/improved performance more? And for which type of query?
We have repeated the steps for `a10`, `a100`, and `a1000`, completing the table with our experimental results. The focus of our analysis is on the relative costs of adding a single index compared to query benefits.

Index Storage
We measured the disk space used by the table before and after adding each index:

- **Without Index**: **DP_SPACE_0** = 488.97 MB
- **After Adding Index on a2**: 629.67 MB (+28.7% increase)
- **After Adding Index on a10**: 580.67 MB (+18.8% increase)
- **After Adding Index on a100**: 614.70 MB (+25.7% increase)
- **After Adding Index on a1000**: 636.70 MB (+30.2% increase)

Observation:

- Indexing increases disk space consumption, but the relationship between index size and distinct values is not strictly linear.
- The `a2` index has the highest storage increase, even though it has only 2 distinct values. This suggests that factors such as page allocation and B+ tree structure impact index storage.

Insertion Time
Adding an index impacts insertion time due to the need to update the index structure for each inserted row. Below is the comparison:

$$\text{Insertion Time Increase (\%)} = \frac{\text{DP\_INSERT\_TIME\_aX} - \text{DP\_INSERT\_TIME\_0}}{\text{DP\_INSERT\_TIME\_0}} \times 100\%$$

- **a2**:

$$\left(\frac{0.0296 - 0.0246}{0.0246}\right) \times 100 = 20.3\% \text{ increase}$$

- **a10**:
$$\left(\frac{0.0296 - 0.0246}{0.0246}\right) \times 100 = 20.3\% \text{ increase}$$
- **a100**:
$$\left(\frac{0.1362 - 0.0246}{0.0246}\right) \times 100 = 453.7\% \text{ increase}$$
- **a1000**:
$$\left(\frac{0.3684 - 0.0246}{0.0246}\right) \times 100 = 1397.6\% \text{ increase}$$

As the selectivity of the indexed attribute increases because of fewer matching rows, the cost of inserting data grows significantly. The indexing overhead is minimal for low-selectivity attributes like `a2` and `a10`, but for higher selectivity attributes like `a100` and `a1000`, the cost of maintaining the index during insertion becomes much more pronounced.

Query Performance Improvements
After indexing, the query execution times improved significantly. The percentage reduction in execution time compared to the baseline **DP_DISTINCT_TIME_0** is computed as follows:

$$\text{Speedup} = \left(1 - \frac{\text{DP\_DISTINCT\_TIME\_x}}{\text{DP\_DISTINCT\_TIME\_0}}\right) \times 100$$

- **a2**: $\left(1 - \frac{0.655}{1.381}\right) \times 100 = 52.6\%$ faster
- **a10**: $\left(1 - \frac{0.183}{1.381}\right) \times 100 = 86.7\%$ faster
- **a100**: $\left(1 - \frac{0.040}{1.381}\right) \times 100 = 97.1\%$ faster
- **a1000**: $\left(1 - \frac{0.011}{1.381}\right) \times 100 = 99.2\%$ faster

These results show that indexing significantly improves query performance, especially for attributes with higher selectivity. The impact of indexing is more pronounced as the number of unique values increases, reducing the number of rows scanned during query execution.

Conclusion
The experiments confirm that indexing dramatically improves query speed while increasing both insertion time and storage requirements. The trade-offs observed are:

- Insertion time increases significantly with higher cardinality indexes.
- Storage usage moderate growth in disk space but more pronounced for high-cardinality attributes.
- Query performance that shows substantial improvements, especially for attributes with more unique values.

Overall, indexes are most beneficial when query performance is a higher priority than insertion speed and storage constraints.

4.2 Multiple Single-Attribute Indexes

We now examine the impact of naïvely adding multiple single-attribute indexes to a table. For this section, drop the table and then add individual indexes for ALL of the columns we have been examining (a2, a10, a100, and a1000) and report on insert time and table space.

To confirm all the indexes have been added, execute: `SHOW INDEX FROM data;`

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
data	0	PRIMARY	1	a1	A	0	NULL	NULL		BTREE		
data	1	foo2	1	a2	A	0	NULL	NULL	YES	BTREE		
data	1	foo10	1	a10	A	0	NULL	NULL	YES	BTREE		
data	1	foo100	1	a100	A	0	NULL	NULL	YES	BTREE		
data	1	foo1000	1	a1000	A	0	NULL	NULL	YES	BTREE		

```
MariaDB [hw5]> DROP TABLE IF EXISTS data;
Query OK, 0 rows affected (0.048 sec)

MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/schema.sql;
Query OK, 0 rows affected (0.015 sec)

MariaDB [hw5]> CREATE INDEX idx_a2 ON data(a2);
Query OK, 0 rows affected (0.014 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [hw5]> CREATE INDEX idx_a10 ON data(a10);
Query OK, 0 rows affected (0.014 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [hw5]> CREATE INDEX idx_a100 ON data(a100);
Query OK, 0 rows affected (0.014 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [hw5]> CREATE INDEX idx_a1000 ON data(a1000);
Query OK, 0 rows affected (0.015 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [hw5]> SHOW INDEX FROM data;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
data	0	PRIMARY	1	a1	A	0	NULL	NULL		BTREE		
data	1	idx_a2	1	a2	A	0	NULL	NULL	YES	BTREE		
data	1	idx_a10	1	a10	A	0	NULL	NULL	YES	BTREE		
data	1	idx_a100	1	a100	A	0	NULL	NULL	YES	BTREE		
data	1	idx_a1000	1	a1000	A	0	NULL	NULL	YES	BTREE		

5 rows in set (0.001 sec)

As with the prior section, now add the data to collect insert time/space data...

```
Query OK, 10000 rows affected (0.698 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.502 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.409 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.693 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.776 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.002 sec)

MariaDB [hw5]>
```

$$\frac{0.698 + 0.502 + 0.409 + 0.693 + 0.776}{5} = 0.6156$$

DP_INSERT_TIME_ALL $\underline{\qquad 0.6156 \qquad}$ **seconds/10,000 rows**

```
SELECT ((data_length + index_length) / POWER(1024, 2)) AS
   tablesize_mb
FROM information_schema.tables
WHERE table_schema='hw5' AND table_name='data';

MariaDB [hw5]> SELECT ((data_length + index_length) / POWER(1024, 2)
   ) AS tablesize_mb
    -> FROM information_schema.tables
    -> WHERE table_schema='hw5' AND table_name='data';
+---------------+
| tablesize_mb |
+---------------+
|  1062.921875 |
+---------------+
1 row in set (0.013 sec)
```

DP_SPACE_ALL $\underline{\qquad 1062.921875 \qquad}$ **MB**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
```

```
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.599 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.603 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.602 sec)
```

$$\frac{0.599 + 0.603 + 0.602}{3} = 0.601$$

DP_DISTINCT_TIME_ALL $\underline{\qquad\qquad 0.601}$ **seconds**

and collect data query time with multiple attributes...

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1 = 1;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.007 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1 = 1;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.001 sec)
```

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1 = 1;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.001 sec)
```

$$\frac{0.007 + 0.001 + 0.001}{3} = 0.003$$

a2 = 1, a1 = 1 $\underline{\qquad\qquad 0.003}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.076 sec)
```

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.025 sec)
```

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.026 sec)
```

$$\frac{1.076 + 1.025 + 1.026}{3} = 1.042$$

a2 = 1, a10 = 1 $\underline{\qquad\qquad 1.042}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
```

```
+----------+
1 row in set (0.823 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (0.802 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (0.798 sec)
```

$$\frac{0.823 + 0.802 + 0.798}{3} = 0.808$$

a2 = 1, a100 = 1 $\underline{\qquad\qquad 0.808}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.780 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.750 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
```

```
1 row in set (0.781 sec)
```

$$\frac{0.780 + 0.750 + 0.781}{3} = 0.770$$

a2 = 1, a1000 = 1 0.770 **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a1 = 1;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.001 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a1 = 1;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.001 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a1 = 1;
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
1 row in set (0.001 sec)
```

$$\frac{0.001 + 0.001 + 0.001}{3} = 0.001$$

a2 = 1, a1000 = 1, a1 = 1 0.001 **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.264 sec)
```

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.241 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.269 sec)
```

$$\frac{0.264 + 0.241 + 0.269}{3} = 0.258$$

a2 = 1, a1000 = 1, a10 = 1	$\underline{0.258}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.121 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.118 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1
   AND a100 = 1;
+----------+
```

```
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.122 sec)
```

$$\frac{0.121 + 0.118 + 0.122}{3} = 0.120$$

a2 = 1, a1000 = 1, a100 = 1 0.120 **seconds**

4.2.1 Analysis

Now discuss your results, covering at least a response to all the following questions. What was the overall impact of adding these indexes in terms of insert time/space (in terms of percentage increase)? To what extent did query time decrease in the test queries? For the multi-attribute queries, was there a pattern for when MariaDB chose which index(es)? If you were dealing with a database that had a high write-to-read ratio, would you advise this strategy? What about the reverse? Are there particular query loads that would make this indexing strategy more appropriate?

What was the overall impact of adding these indexes in terms of insert time/space (in terms of percentage increase)?

DP_INSERT_TIME_0 00246 **seconds/10,000 rows**

DP_INSERT_TIME_ALL 0.6156 **seconds/10,000 rows**

$$\frac{0.6156 - 0.0246}{0.0246} \times 100\% = 2402.44\%$$

Insert Time Increase 2402.44% **increase**

DP_SPACE_0 488.96875 **MB**

DP_SPACE_ALL 1062.921875 **MB**

$$\frac{1062.921875 - 488.96875}{488.96875} \times 100\% = 117.4\%$$

Storage Space Increase 117.4% **increase**

To what extent did query time decrease in the test queries?

DP_DISTINCT_TIME_0 1.381 **seconds**

DP_DISTINCT_TIME_ALL 0.601 **seconds**

$$\frac{1.381 - 0.601}{1.381} \times 100\% = 56.5\%$$

Speedup Percentage 56.5% **faster**

For the multi-attribute queries, was there a pattern for when MariaDB chose which index(es)

1. When the Query Includes `PRIMARY KEY (a1)`, MariaDB Selects the `PRIMARY` Index

```
EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1 = 1;
```

```
+------+-------------+-------+-------+----------------+---------+---------+-------+------+-------+
| id   | select_type | table | type  | possible_keys  | key     | key_len | ref   | rows | Extra |
+------+-------------+-------+-------+----------------+---------+---------+-------+------+-------+
|    1 | SIMPLE      | data  | const | PRIMARY,idx_a2 | PRIMARY | 4       | const | 1    |       |
+------+-------------+-------+-------+----------------+---------+---------+-------+------+-------+
```

MariaDB selects the `PRIMARY` index:

- Since `a1` is the primary key, querying `a1 = 1` directly identifies a single row, reducing the search space to just one row.
- Other indexes such as `idx_a2` are ignored because the `PRIMARY` index provides the most efficient lookup.

2. For Non-Primary Key Queries, MariaDB Uses the `index_merge` Strategy

```
EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
```

```
+------+-------------+-------+-------------+---------------+---------------+---------+------+---------+-----------------------------------------------------------------+
| id   | select_type | table | type        | possible_keys | key           | key_len | ref  | rows    | Extra                                                           |
+------+-------------+-------+-------------+---------------+---------------+---------+------+---------+-----------------------------------------------------------------+
|    1 | SIMPLE      | data  | index_merge | idx_a2,idx_a10 | idx_a10,idx_a2 | 5,5     | NULL | 1063442 | Using intersect(idx_a10,idx_a2); Using where; Using index |
+------+-------------+-------+-------------+---------------+---------------+---------+------+---------+-----------------------------------------------------------------+
```

MariaDB optimizes the query using `index_merge`:

- `index_merge` allows MariaDB to utilize multiple indexes simultaneously and intersect them to improve query performance.
- In the case of `a2 = 1 AND a10 = 1`, MariaDB selects both `idx_a10` and `idx_a2`, using an index merge to filter results.

3. Index Selection is Influenced by Column Cardinality

```
EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
```

```
+----+-------------+-------+-------------+---------------+---------------+---------+------+------+-----------------------------------------------------------------+
| id | select_type | table | type        | possible_keys | key           | key_len | ref  | rows | Extra                                                           |
+----+-------------+-------+-------------+---------------+---------------+---------+------+------+-----------------------------------------------------------------+
|  1 | SIMPLE      | data  | index_merge | idx_a2,idx_a1000 | idx_a1000,idx_a2 | 5,5     | NULL | 5000 | Using intersect(idx_a1000,idx_a2); Using where; Using index |
+----+-------------+-------+-------------+---------------+---------------+---------+------+------+-----------------------------------------------------------------+
```

- MariaDB selects both `idx_a1000` and `idx_a2` and uses an index merge to optimize the query.
- Since `a1000` has a higher cardinality that more unique values than `a2`, MariaDB likely prioritizes filtering on `idx_a1000` first to minimize the number of scanned rows.
- The query result is reduced from 5 million rows (a2) to 5000 rows (a1000), showing that MariaDB effectively used index-based filtering.

4. Multi-Index Queries

```
EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1 AND a100 =
1;
```

```
+----+-------------+-------+-------------+---------------+---------------+---------+------+------+-----------------------------------------------------------------+
| id | select_type | table | type        | possible_keys | key           | key_len | ref  | rows | Extra                                                           |
+----+-------------+-------+-------------+---------------+---------------+---------+------+------+-----------------------------------------------------------------+
|  1 | SIMPLE      | data  | index_merge | idx_a2,idx_a100,idx_a1000 | idx_a1000,idx_a100 | 5,5     | NULL | 209  | Using intersect(idx_a1000,idx_a100); Using where |
+----+-------------+-------+-------------+---------------+---------------+---------+------+------+-----------------------------------------------------------------+
```

- MariaDB continues to use `index_merge` and selects `idx_a1000` and `idx_a100` to compute an intersection.
- Due to index optimization, the query result size is reduced from 5000 rows (only a1000) down to 209 rows, demonstrating that MariaDB effectively combines multiple indexes to refine search results.

If you were dealing with a database that had a high write-to-read ratio, would you advise this strategy? What about the reverse?

If the database has a high write-to-read ratio:

- Adding multiple indexes is not recommended.
- Each insert, update, or delete operation requires maintaining the indexes, significantly increasing write overhead.
- With a single index on `a2`, **DP_INSERT_TIME_a2** increased by **20.3%**, whereas with multiple indexes (**DP_INSERT_TIME_ALL**), insert time increased by more than **2400%**. This demonstrates that indexes can severely impact insertion performance.

If the database has a high read-to-write ratio:

- Adding indexes is recommended to optimize query performance.
- The query `a2 = 1 AND a1000 = 1` took **1.381s** without indexing, but with indexing, it decreased to **0.750s**, achieving a **45.7%** speedup.
- Indexes significantly improve query speed by reducing the number of scanned rows, making them beneficial for read-heavy workloads.

Are there particular query loads that would make this indexing strategy more appropriate?

Indexing multiple single attributes is most appropriate for query loads that frequently involve filtering, searching, or joining on these indexed columns. Based on our analysis, the following query patterns benefit the most:

- High Selectivity Queries:
  Queries that filter on attributes with high selectivity benefit significantly from indexing.
  Example:

  ```
  SELECT COUNT(*) FROM data WHERE a1000 = 1;
  ```

  Execution time decreased by **99.7%** after indexing.
- Multi-Attribute Queries with Index Merging:
  Queries combining multiple indexed attributes using `AND` conditions benefit from index intersection strategies.
  Example:

  ```
  SELECT COUNT(*) FROM data WHERE a2 = 1 AND a100 = 1;
  ```

  Reduced execution time from **1.381s** to **0.802s**, a **41.9%** speedup.
- Read-Heavy Analytical Workloads:
  Indexing is highly beneficial for online multiple analysis that perform frequent aggregations and reporting.
- Joins on Indexed Attributes:
  When indexed attributes are used as join keys, query performance improves significantly by reducing the number of scanned rows.

Indexing multiple attributes is beneficial for read-heavy workloads, analytical queries, and queries with high selectivity conditions. However, in write-heavy environments, excessive indexing should be avoided to reduce insert/update overhead.

4.3 Multi-Attribute Indexes

We now focus on indexes that use multiple attributes, and in particular the degree to which order affects applicability of the index, as well as speed. First, drop the table and re-create the schema; next, focus on the attribute set {a2, a1000}: collect all the data in the table's first column below for a single index declared as `CREATE INDEX first ON data(a2, a1000);` Second, drop the table and re-create the schema; now `CREATE INDEX second ON data(a1000, a2);` and collect the second column's data. Note that the order does matter, as you will see – both in terms of query-time benefits, as well as when the index can be used.

	(a2, a1000)	(a1000, a2)
Insert Time	0.3202	0.319
Space on Disk	688.859375	688.9375
{ a2 = 1 }	0.718	1.472
{ a10 = 1 }	1.299	1.354
{ a100 = 1 }	1.297	1.295
{ a1000 = 1 }	1.347	0.005
{ a2 = 1, a10 = 1 }	29.456	1.619
{ a10 = 1, a100 = 1 }	1.483	1.505
{ a2 = 1, a1000 = 1 }	0.007	0.006

(a2, a1000)

```
MariaDB [hw5]> DROP TABLE IF EXISTS data;
Query OK, 0 rows affected (0.033 sec)
```

```
MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/schema.sql;
Query OK, 0 rows affected (0.130 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data;
+-----------+
| COUNT(*) |
+-----------+
|         0 |
+-----------+
1 row in set (0.002 sec)

MariaDB [hw5]> CREATE INDEX first ON data(a2, a1000);
Query OK, 0 rows affected (0.019 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [hw5]> SHOW INDEX FROM data;
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
data	0	PRIMARY	1	a1	A	0	NULL	NULL		BTREE		
data	1	first	1	a2	A	0	NULL	NULL	YES	BTREE		
data	1	first	2	a1000	A	0	NULL	NULL	YES	BTREE		

```
3 rows in set (0.001 sec)

MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/data_10m_10k.sql;
Query OK, 0 rows affected (0.001 sec)

Query OK, 10000 rows affected (0.341 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.299 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.334 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.336 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.291 sec)
Records: 10000  Duplicates: 0  Warnings: 0
```

```
Query OK, 0 rows affected (0.002 sec)
```

$$\frac{0.341 + 0.299 + 0.334 + 0.336 + 0.291}{5} = 0.3202$$

DP_INSERT_TIME_first $\underline{\qquad 0.3202 \qquad}$ **seconds/10,000 rows**

```
MariaDB [hw5]> SELECT ((data_length + index_length) / POWER(1024, 2)
   ) AS tablesize_mb
    -> FROM information_schema.tables
    -> WHERE table_schema='hw5' AND table_name='data';
+---------------+
| tablesize_mb |
+---------------+
|   688.859375 |
+---------------+
1 row in set (0.009 sec)
```

DP_SPACE_first $\underline{\qquad 688.859375 \qquad}$**MB**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.743 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.703 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (0.707 sec)
```

$$\frac{0.743 + 0.703 + 0.707}{3} = 0.718$$

Query Time $a2 = 1$ $\underline{\qquad 0.718 \qquad}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.285 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.302 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.309 sec)
```

$$\frac{1.285 + 1.302 + 1.309}{3} = 1.299$$

Query Time $a10 = 1$ $\underline{\qquad 1.299 \qquad}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.272 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a100 = 1;
+----------+
| COUNT(*) |
+----------+
```

```
|   100000 |
+----------+
1 row in set (1.287 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.332 sec)
```

$$\frac{1.272 + 1.287 + 1.332}{3} = 1.297$$

Query Time $a100 = 1$ $\underline{\qquad 1.297}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (1.345 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (1.339 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (1.357 sec)
```

$$\frac{1.345 + 1.339 + 1.357}{3} = 1.347$$

Query Time $a1000 = 1$ $\underline{\qquad 1.347}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
```

```
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (29.555 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (29.391 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (29.422 sec)
```

$$\frac{29.555 + 29.391 + 29.422}{3} = 29.456$$

Query Time $a2 = 1, a10 = 1$ $\underline{\qquad\qquad 29.456}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.495 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.461 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
```

```
+-----------+
|    100000 |
+-----------+
1 row in set (1.492 sec)
```

$$\frac{1.495 + 1.461 + 1.492}{3} = 1.483$$

Query Time $a10 = 1, a100 = 1$ $\underline{\qquad 1.483 \qquad}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+-----------+
| COUNT(*) |
+-----------+
|     10000 |
+-----------+
1 row in set (0.009 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+-----------+
| COUNT(*) |
+-----------+
|     10000 |
+-----------+
1 row in set (0.004 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+-----------+
| COUNT(*) |
+-----------+
|     10000 |
+-----------+
1 row in set (0.007 sec)
```

$$\frac{0.009 + 0.004 + 0.007}{3} = 0.007$$

Query Time $a2 = 1, a1000 = 1$ $\underline{\qquad 0.007 \qquad}$ **seconds**

```
MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----+-------------+-----+----+-------------+-----+-------+-----+---------+-------------+
| id |select_type|table|type|possible_keys|key  |key_len|ref  | rows    | Extra       |
+----+-------------+-----+----+-------------+-----+-------+-----+---------+-------------+
|  1 |SIMPLE     | data|ref |first        |first| 5     |const| 4867349 | Using where |
+----+-------------+-----+----+-------------+-----+-------+-----+---------+-------------+
1 row in set (0.001 sec)

MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----+-------------+-----+----+-------------+------+---------+------+---------+-------------+
| id |select_type|table|type|possible_keys| key  | key_len | ref  | rows    | Extra       |
+----+-------------+-----+----+-------------+------+---------+------+---------+-------------+
|  1 |SIMPLE     |data |ALL |NULL         | NULL | NULL    | NULL | 9734699 | Using where |
+----+-------------+-----+----+-------------+------+---------+------+---------+-------------+
```

```
+-------+------------+---------+------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name | Seq_in_index | Column_name | Collation | Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+---------+------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+
| data  |          0 | PRIMARY  |            1 | a1          | A         |           0 |     NULL | NULL   |      | BTREE      |         |               |
| data  |          1 | second   |            1 | a1000       | A         |           0 |     NULL | NULL   | YES  | BTREE      |         |               |
| data  |          1 | second   |            2 | a2          | A         |           0 |     NULL | NULL   | YES  | BTREE      |         |               |
+-------+------------+---------+------------+-------------+-----------+-------------+----------+--------+------+------------+---------+---------------+

1 row in set (0.001 sec)

MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+----+-------------+-----+----+---------------+-------+---------+-------------+-------+-------------+
| id |select_type|table|type| possible_keys | key   | key_len | ref         | rows  | Extra       |
+----+-------------+-----+----+---------------+-------+---------+-------------+-------+-------------+
|  1 |SIMPLE     |data |ref | first         | first | 10      | const,const | 18244 | Using index |
+----+-------------+-----+----+---------------+-------+---------+-------------+-------+-------------+
1 row in set (0.001 sec)
```

EXPLAIN Analysis:

- $a2 = 1, a10 = 1$: Uses ‘first‘ index but only partially (‘Using where‘), causing slow query time (29.456s).
- $a10 = 1, a100 = 1$: No index used (‘NULL‘), leading to poor performance.
- $a2 = 1, a1000 = 1$: Fully utilizes ‘first‘ index (‘Using index‘), resulting in a fast query (0.007s).

(a1000, a2)

```
MariaDB [hw5]> DROP TABLE IF EXISTS data;
Query OK, 0 rows affected (0.031 sec)

MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/schema.sql;
Query OK, 0 rows affected (0.019 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
1 row in set (0.002 sec)

MariaDB [hw5]> CREATE INDEX second ON data(a1000, a2);
Query OK, 0 rows affected (0.029 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [hw5]> SHOW INDEX FROM data;

3 rows in set (0.002 sec)
```

```
MariaDB [hw5]> SOURCE /Users/erdune/Desktop/NortheasternMiami/
   CS5200DatabaseManagementSystems/Homework5/data_10m_10k.sql;
Query OK, 0 rows affected (0.001 sec)

Query OK, 10000 rows affected (0.275 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.352 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.306 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.292 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 10000 rows affected (0.370 sec)
Records: 10000  Duplicates: 0  Warnings: 0

Query OK, 0 rows affected (0.002 sec)
```

$$\frac{0.275 + 0.352 + 0.306 + 0.292 + 0.370}{5} = 0.319$$

DP_INSERT_TIME_second $\underline{\qquad 0.319 \qquad}$ **seconds/10,000 rows**

```
MariaDB [hw5]> SELECT ((data_length + index_length) / POWER(1024, 2)
   ) AS tablesize_mb
    -> FROM information_schema.tables
    -> WHERE table_schema='hw5' AND table_name='data';
+---------------+
| tablesize_mb |
+---------------+
|     688.9375 |
+---------------+
1 row in set (0.008 sec)
```

DP_SPACE_second $\underline{\qquad 688.9375 \qquad}$ **MB**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
```

```
+----------+
1 row in set (1.390 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (1.382 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1;
+----------+
| COUNT(*) |
+----------+
|  5000000 |
+----------+
1 row in set (1.645 sec)
```

$$\frac{1.390 + 1.382 + 1.645}{3} = 1.472$$

Query Time $a2 = 1$ $\underline{\qquad\qquad 1.472}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.316 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.400 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
```

```
+----------+
1 row in set (1.345 sec)
```

$$\frac{1.316 + 1.400 + 1.345}{3} = 1.354$$

Query Time $a10 = 1$ $\underline{\qquad 1.354}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.289 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.298 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.299 sec)
```

$$\frac{1.289 + 1.298 + 1.299}{3} = 1.295$$

Query Time $a100 = 1$ $\underline{\qquad 1.295}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.004 sec)
```

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.005 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.007 sec)
```

$$\frac{0.004 + 0.005 + 0.007}{3} = 0.005$$

Query Time $a1000 = 1$ 0.005 **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.618 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.613 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----------+
| COUNT(*) |
+----------+
|  1000000 |
+----------+
1 row in set (1.628 sec)
```

$$\frac{1.618 + 1.613 + 1.628}{3} = 1.619$$

Query Time $a2 = 1, a10 = 1$ $\underline{\qquad 1.619}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.527 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.465 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----------+
| COUNT(*) |
+----------+
|   100000 |
+----------+
1 row in set (1.522 sec)
```

$$\frac{1.527 + 1.465 + 1.522}{3} = 1.505$$

Query Time $a10 = 1, a100 = 1$ $\underline{\qquad 1.505}$ **seconds**

```
MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+----------+
| COUNT(*) |
+----------+
|    10000 |
+----------+
1 row in set (0.006 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+----------+
| COUNT(*) |
+----------+
```

```
|     10000 |
+-----------+
1 row in set (0.006 sec)

MariaDB [hw5]> SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+-----------+
| COUNT(*) |
+-----------+
|     10000 |
+-----------+
1 row in set (0.008 sec)
```

$$\frac{0.006 + 0.006 + 0.008}{3} = 0.006$$

Query Time $a2 = 1, a1000 = 1$ $\underline{\qquad\qquad 0.006}$ **seconds**

```
MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a10 = 1;
+----+-------------+-----+----+-------------+-----+-------+-----+---------+-------------+
| id |select_type|table|type|possible_keys|key  |key_len|ref  | rows    | Extra       |
+----+-------------+-----+----+-------------+-----+-------+-----+---------+-------------+
|  1 |SIMPLE     | data|ALL |NULL         | NULL| NULL  |NULL | 9734960 | Using where |
+----+-------------+-----+----+-------------+-----+-------+-----+---------+-------------+
1 row in set (0.003 sec)

MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a10 = 1 AND a100 = 1;
+----+-------------+-----+----+-------------+------+---------+------+---------+-------------+
| id |select_type|table|type|possible_keys| key  | key_len | ref  | rows    | Extra       |
+----+-------------+-----+----+-------------+------+---------+------+---------+-------------+
|  1 |SIMPLE     |data |ALL |NULL         | NULL | NULL    | NULL | 9734960 | Using where |
+----+-------------+-----+----+-------------+------+---------+------+---------+-------------+
1 row in set (0.001 sec)

MariaDB [hw5]> EXPLAIN SELECT COUNT(*) FROM data WHERE a2 = 1 AND a1000 = 1;
+----+-------------+-----+----+---------------+-------+---------+-------------+-------+-------------+
| id |select_type|table|type| possible_keys | key   | key_len | ref         | rows  | Extra       |
+----+-------------+-----+----+---------------+-------+---------+-------------+-------+-------------+
|  1 |SIMPLE     |data |ref | second        |second | 10      | const,const | 18138 | Using index |
+----+-------------+-----+----+---------------+-------+---------+-------------+-------+-------------+
1 row in set (0.003 sec)
```

EXPLAIN Analysis:

- $a2 = 1, a10 = 1$: No index used `NULL`, leading to poor performance.
- $a10 = 1, a100 = 1$: No index used `NULL`, resulting in slow query execution.
- $a2 = 1, a1000 = 1$: Fully utilizes `second` index `Using index`, significantly improving query performance.

4.3.1 Analysis

Now discuss your results, covering at least a response to all the following questions. How did the multi-attribute index compare to multiple single-attribute indexes? What were the trends you discovered with respect to when a multi-attribute index could apply for a query (in full or part)?

Under what situations would you use a multi-attribute query (with respect to insert/query ratio, query load)? What is a good rule of thumb for ordering attributes? Would there be any advantage to having a single-attribute index on (a2) if there were a multi-attribute index on (a2, a1000) vs. (a1000, a2)?

How did the multi-attribute index compare to multiple single-attribute indexes?

Multi-attribute indexes can significantly improve query performance when the query conditions match the leading column of the index. In such cases, the index can be fully leveraged for efficient lookups. However, compared to multiple single-attribute indexes, multi-attribute indexes require more time for insertions and occupy more disk space. This trade-off should be considered when designing an indexing strategy.

When could a multi-attribute index apply to a query (fully or partially)?

- If the query condition matches the leading column of the index, the multi-attribute index provides the best performance since it can be used directly for lookups.
- If the query condition does not follow the order of attributes in the index, the optimizer may not fully utilize the index, leading to slower performance or even a full table scan as seen in EXPLAIN results showing `Using where`.
- For example, with an index on `(a2, a1000)`, a query like `WHERE a2 = 1 AND a1000 = 1` can efficiently use the index, whereas `WHERE a1000 = 1` alone may not fully benefit from it.

When should a multi-attribute index be used (in terms of insert/query ratio, query load)?

- If the database has read-heavy workloads, using a multi-attribute index is beneficial as it significantly improves query speed.
- However, if the database experiences high write-to-read ratios, the overhead of maintaining a multi-attribute index may outweigh its benefits, leading to slower insert/update operations.

What is a good rule of thumb for ordering attributes in a multi-attribute index?

- Place high-selectivity columns first to maximize index efficiency and minimize the number of scanned rows.
- Prioritize frequently queried columns as leading attributes, ensuring that the index is leveraged effectively for common query patterns.
- Group attributes that are often queried together in a single index. For example, `(a2, a1000)` is more effective than separate indexes if queries frequently include `WHERE a2 = 1 AND a1000 = 1`.

Would there be any advantage to having a single-attribute index on `(a2)` if there were a multi-attribute index on `(a2, a1000)` vs. `(a1000, a2)`?

- If queries frequently involve filtering only by `a2`, then having a separate single-attribute index on `a2` is beneficial, as it allows the optimizer to directly use it without being constrained by the order of a composite index.
- However, if `a2` is primarily used in combination with `a1000`, then the multi-attribute index `(a2, a1000)` is sufficient, and adding a separate index on `a2` may not provide a noticeable performance boost.