Comp 4380 Project: Phase 3

Muhammad Faheem Bunally
Computer Science student
University of Manitoba
Winnipeg, Canada
bunallmf@myumanitoba.ca

Victor Ifeakachi Ezendu
Computer Science student
University of Manitoba
Winnipeg, Canada
ezenduvi@myumanitoba.ca

***Abstract*—This document is a report for phase 3 of the project for COMP 4380, regarding the optimization of SQL queries.**

Keywords—experiment, analysis

I. Introduction

The data set is a record of all parking tickets that have been issued in Chicago from January 1996 to May 2018. It contains different types of violations, and the locations where violations occurred. The data set is large enough to act as a population sample for different statistical computations, such as analysing trends.

The queries are:

1. Finding how the number of parking tickets produced changes over each year.
2. Finding which make of vehicles has been receiving the most tickets over the years.
3. Finding how the number of parking tickets given in different community areas changes over the years.
4. How many tickets were given each year for prohibited parking due to blocking access, alleys, driveways, or fire lanes?
5. In which years, after 1999, was the most amount of money spent on paying parking tickets?
6. What is the average amount of level 1 fines issued per month?
7. How many tickets were issued for parking violations in each community area?
8. What is the most common type of violation and in which areas is it most frequently issued?
9. Which vehicle received the least number of tickets?
10. What is the average number of tickets per parking enforcement officer?
11. How many tickets were issued between 1998-2000 and what was the total amount of level 2 fines collected?
12. What month has the highest number of tickets given?
13. What officer has given the most tickets?

From phase 2, query 3 was changed. It is now ordered by "community_area_name", then by year.

II. Data

- The database management system is SQLite DB browser.
- The machine has a RAM capacity of 16.00 GB.
- The database consists of a single table.
- It contains 36 columns and 54430546 rows of data.
- The database has a total size of 19.7 GB on disk.
- The database was initially a ".csv" file and was converted to ".db" using the DBMS.

III. Experiments

1. Finding how the number of parking tickets produced changes over each year.

```
"SELECT year, count(year) AS
num_tickets
FROM parking_tickets
GROUP BY year
ORDER BY year ASC;"
```

Attempted to create an index on "year".
Attempted to rewrite query to be more efficient.

2. Finding which make of vehicles has been receiving the most tickets over the years.

```
"SELECT vehicle_make,
count(vehicle_make) AS
num_tickets
FROM parking_tickets
GROUP BY vehicle_make
ORDER BY vehicle_make ASC;"
```

Attempted to create an index on "vehicle_make".

Attempted to rewrite query to be more efficient.

3. Finding how the number of parking tickets given in different community areas changes over the years.

```
“SELECT community_area_name,
year,
count(community_area_name) AS
num_tickets
FROM parking_tickets
GROUP BY
community_area_name, year
ORDER BY
community_area_name, year
ASC;”
```

Attempted to create an index on “community_area_name AND year”.
Attempted to rewrite query to be more efficient.

4. How many tickets were given each year for prohibited parking due to blocking access, alleys, driveways, or fire lanes.

```
“SELECT year,
count(violation_code) AS
num_tickets
FROM parking_tickets
WHERE violation_code =
‘0964100C’
GROUP  BY year
ORDER BY year ASC;”
```

Attempted to create an index on “violation_code AND year”.
Attempted to rewrite query to be more efficient.

5. In which years, after 1999, was the most amount of money spent on paying parking tickets?

```
“SELECT year,
sum(total_payments) AS
yearly_sum
FROM parking_tickets
WHERE year > 1999
GROUP BY year
ORDER BY yearly_sum DESC;”
```

Attempted to create an index on “year AND total_payments”.
Attempted to rewrite query to be more efficient.

6. What is the average amount of level 1 fines issued per month?

```
“SELECT month,
AVG(fine_level1_amount) AS
avg_level_1_fine
FROM parking_tickets
GROUP BY month;”
```

Attempted to create an index on “month AND fine_level1_amount”.
Attempted to rewrite query to be more efficient.

7. How many tickets were issued for parking violations in each community area?

```
“SELECT community_area_name,
COUNT(*) AS num_tickets
FROM parking_tickets
GROUP BY
community_area_name;”
```

Attempted to create an index on “community_area_name”.
Attempted to create an index on “year” and “num_tickets”.
Attempted to rewrite query to be more efficient.

8. What is the most common type of violation and in which areas is it most frequently issued?

```
“SELECT violation_description,
community_area_name,
COUNT(*) AS num_tickets
FROM parking_tickets
GROUP BY violation_description,
community_area_name
ORDER BY num_tickets DESC
LIMIT 1;”
```

Attempted to create an index on “violation_description AND community_area_name”.
Attempted to create an index on year “num_tickets”.
Attempted to rewrite query to be more efficient.

9. Which vehicle received the least number of tickets?

```
“SELECT vehicle_make,
COUNT(*) AS num_tickets
FROM parking_tickets
GROUP BY vehicle_make
ORDER BY num_tickets ASC
LIMIT 1;”
```

Attempted to create an index on “vehicle_make”
Attempted to create an index on year “num_tickets”.
Attempted to rewrite query to be more efficient

10. What is the average number of tickets per parking enforcement officer?

```
“SELECT AVG(num_tickets) AS
avg_num_tickets_per_officer
FROM (SELECT officer,
COUNT(*) AS num_tickets
    FROM parking_tickets
    GROUP BY officer) AS t;”
```

Attempted to create an index on “officer”.
Attempted to create an index on year “num_tickets”.
Attempted to rewrite query to be more efficient.

11. How many tickets were issued between 1998-2000 and what was the total amount of level 2 fines collected?

```
“SELECT COUNT(*) AS
num_tickets,
SUM(fine_level2_amount) AS
total_fines
FROM parking_tickets
WHERE year BETWEEN '1998'
AND '2000';”
```

Attempted to create and index on “fine_level2_amount AND year”.
Attempted to create an index on “year AND fine_level2_amount”.
Attempted to create an index on year “num_tickets”.
Attempted to rewrite query to be more efficient.

12. What month has the highest number of tickets given?

```
“SELECT month, COUNT(*) AS
num_tickets
FROM parking_tickets
GROUP BY month
ORDER BY num_tickets DESC
LIMIT 1;”
```

Attempted to create an index on “month”.
Attempted to create an index on year “num_tickets”.
Attempted to rewrite query to be more efficient.

13. What officer has given the most tickets?

```
“SELECT officer, COUNT(*) AS
num_tickets
FROM parking_tickets
GROUP BY officer
ORDER BY num_tickets DESC
LIMIT 1;”
```

Attempted to create an index on “officer”.
Attempted to create an index on year “num_tickets”.
Attempted to rewrite query to be more efficient.

IV. Analysis

From the above experiments, a general trend was that rewriting the queries in a more efficient way was unsuccessful because all our queries were already as efficient as they could be.

For each query, the respective experiment resulted in the following:

1. After adding an index on “year”, the runtime was improved from 237601 ms to 11632 ms.

2. Adding an index on “vehicle_make”, caused the runtime to improve from 138878 ms to 13107 ms.

3. Adding an index on "community_area_name AND year", decreased the runtime from 587078 ms to 16460 ms.

4. After adding an index on "violation_code AND year", the runtime was improved from 94538 ms to 33 ms.

5. Adding an index on "year AND total_payments", the runtime was improved from 185007 ms to 13966 ms.

6. After adding an index on "month AND fine_level1_amount", the runtime was improved from 149669 ms to 8077 ms.

7. Creating an index on the num_tickets attribute would also be expensive, as it would require us to create a new table with a num_tickets column. However, after adding an index on "community_area_name", the runtime was improved from 386510 ms to 7237 ms.

8. Creating an index on the num_tickets attribute would also be expensive, as it would require us to create a new table with a num_tickets column. Adding an index on "violation_description AND community_area_name", caused the runtime decrease from 1005306 ms to 29153 ms.

9. Creating an index on the num_tickets attribute would also be expensive, as it would require us to create a new table with a num_tickets column.After adding an index on "vehicle_make", the runtime was improved from 432868 ms to 17881 ms.

10. Creating an index on the num_tickets attribute would also be expensive, as it would require us to create a new table with a num_tickets column.After adding an index on "officer", the runtime was improved from 526285 ms to 16148 ms.

11. Creating an index on the num_tickets attribute would also be expensive, as it would require us to create a new table with a num_tickets column. Adding an index on "fine_level2_amount AND year" causes the runtime to be reduced from 248967 ms to only 28963 ms. But adding an index on "year AND fine_level2_amount", reduces the runtime to 3938 ms.

12. Creating an index on the num_tickets attribute would also be expensive, as it would require us to create a new table with a num_tickets column.After adding an index on "month", the runtime was improved from 401162 ms to 15746 ms.

13. Creating an index on the num_tickets attribute would also be expensive, as it would require us to create a new table with a num_tickets column.After adding an index on "officer", the runtime was improved from 543243 ms to 16025 ms

V. Discussions

Breaking down that one big table into multiple smaller ones, each with attributes with different focus might have provided more statements with different types and a lot more options when it comes to the experiment and analysis parts.
For example, we could have one table "Violation" with the attributes "violation_code", and "violation_description".

References

[1] https://www.propublica.org/datastore/dataset/chicago-parking-ticket-data

[2] SQLite DB browser.

[3] IEEE templates