# 

**Prof. Jun Wang**

University of central missouri

Denver crime report

*Improve query performance with indexing and partitioning*

**Fast and furious**

Deepa Ravichandran (700687747)

Krishna Poojitha vaka (700691512)

Mannepalli Rohini (700658261)

Nangunoori MedhaSree (700688970)

# 

# 

5/1/2019

**Table of Contents:**

[**1.Objective: 2**](#_Toc7637542)

[**2. Indexing: 3**](#_Toc7637543)

[**2.1 Implementation procedure: 3**](#_Toc7637544)

[**2.2 Files to be used in the dataset: 3**](#_Toc7637545)

[**2.3. Data Import: 4**](#_Toc7637546)

[**2.4. Indexing implementation: 4**](#_Toc7637547)

[**3. Partitioning: 7**](#_Toc7637548)

[**3.1 Horizontal Partitioning: 8**](#_Toc7637549)

[**3.2 Filegroups: 8**](#_Toc7637550)

[**4. Software: 9**](#_Toc7637551)

[**5. Conclusion: 9**](#_Toc7637552)

[**6. References: 10**](#_Toc7637553)

# 

# **1.Objective:**

Denver Data Crime includes criminal offenses in the City and County of Denver for the previous five calendar years plus the current year to date. The data is based on the National Incident Based Reporting System (NIBRS) which includes all victims of person crimes and all crimes within an incident. The data is dynamic, which allows for additions, deletions and/or modifications at any time, resulting in more accurate information in the database. Due to continuous data entry, the number of records in subsequent extractions are subject to change.

For such applications there are millions of transactions and immense number of records in each table. In order to search for any crime, the processing time would be more and is tedious.

To overcome this, we need to Import the dataset into SQL server and check the query performance using some scenarios before implementing indexing and partitioning. Implement the indexing and partitioning on the tables and then check the query performance and achieve the required improvement of the performance.

# **2.Indexing:**

Index is a structure created on a table to improve the speed of data access. It is best suited when you require accessing a small set of randomly distributed rows from the table. It can reduce disk I/O (input/output) by using a rapid path access method to locate data quickly. An index helps to speed up select queries and where clauses, but it slows down data input, with the update and the insert statements.

For example, if you want to reference all pages in a book that discusses a certain topic, you first refer to the index, which lists all the topics alphabetically and are then referred to one or more specific page numbers.

SQL Server supports table and index partitioning. The data of partitioned tables and indexes is divided into units that can be spread across more than one file group in a database. The data is partitioned horizontally, so that groups of rows are mapped into individual partitions. All partitions of a single index or table must reside in the same database. The table or index is treated as a single logical entity when queries or updates are performed on the data.

## 2.1Implementation procedure:

Denver Crime dataset encompasses of information pertaining to the below areas:

* How to list distinct Incidents, Offense type and Crime year in the neighborhood
* How toinclude Data of all victims of persons crimes and all crimes within an incident?
* How to List Incident Id and category names where offense type is 'criminal-trespassing' or 'drug-methamphetamine-possess' or 'disturbing-the-peace'?
* How to List all incidents in any year related to Drug-Alcohol category?

## 2.2 Files to be used in the dataset:

**2.2.1.Crimes.csv:**

Each record reveals the information about the reported incident, place of the incident and the time when incident occurred. It also portraits the details whether the incident occurred can be considered as a crime. In this entity INCIDENT\_ID and OFFENSE\_ID together can be considered as a primary key.

**2.2.2.OffenseCode.csv:**

The offense code entity reveals the details about the offense. It exposes the name of the offense and the category the offense falls in to, whether the offense comes under crime or traffic.To implement Indexing (Sparse and dense Index), partitioning on the tables, above open source dataset will be used. Focus of the project will be to analyze the real-world scenarios using tables and compare the

results with and without partitioning and indexing the tables.

## 2.3. Data Import:

We are importing the data into the SQL server management studio. Select the CSV file, which needs to be imported and give the destination path correctly with the required parameters.Once thedata is loaded successfully, data is depicted,and the queries can be performed on the database.

Data is imported from above mentioned .csv files into SQL server using SSIS tool. Data Flow Task is used for loading data from above mentioned csv files (Source) to SQL Server databases (target). Preliminary analysis on the input fields that were causing issue for data extraction i.e., (fields having blank data and fields having data like paragraphs) was completed. The properties of the output columns of the source files were modified such that data can be loaded into tables from source files. In Data flow.

task, “flat file source” is used as source component and “OLE DB Destination” as destination component. Connection managers utilized for the data flow task:

* Flat file Connection manager
* OLE DB connection using SQL Server Native Client 11.0(Provider)

Select the CSV file, which needs to be imported and give the destination path correctly with the required parameters.

**SSMS TOOL:** SQL Server Management Studio (SSMS) is used to query, design, and manage your databases and data warehouses. SSMS tool can also be used to load data to SQL Server installed on machine. Software used for the project:

## 2.4. Indexing implementation:

Clustered index and non-clustered index can be applied to the Denver crime dataset. We are considering few scenarios that analyses the

performance applying clustered and non-clustered index and the performance without clustered and non-clustered index.

**2.4.1. Scenario1:**

List Incident Id and category names where offense type is 'criminal-trespassing' or 'drug- methamphetamine possess' or 'disturbing-the-peace'

**SELECTc.[\_INCIDENT\_ID\_],o.[\_OFFENSE\_CATEGORY\_NAME\_ ]**

**FROM[denver\_crime].[dbo].[crime]c, [denver\_crime].[dbo].[offense\_codes2] o**

**WHEREo.[\_OFFENSE\_TYPE\_ID\_]=c.[\_OFFENSE\_TYPE\_ID\_]ANDc.[\_OFFENSE\_TYPE\_ID\_] IN('criminal-trespassing','drug-methampetamine-possess','disturbing-the-peace')**

The performance of the query without introducing any indexing is: **1.08s**.Clustered index is created on\_OFFENSE\_CATEGORY\_NAME\_ column in the entity offense\_codes.Once we have performed the clustered index, the performance of the query improved to **83ms.**We have created the non-clustered index on the same entity but different column \_OFFENSE\_TYPE\_ID\_ and recorded the performance of **27ms.**We must drop the clustered index and let the non-clustered index remain the same. Allowing the non-clustered index on the entity, the performance of the query was recorded as **52ms.**

Hence based on the performance recorded, the query shows better performance with clustered and non-clustered index

**Comparison of Query performance:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Query1 | Without  indexing | Clustered Index | Clustered and non-clustered | Only non-clustered |
| CPU parse time | 28ms | 0 | 0 | 0 |
| Compile time | 28ms | 2ms | 2ms | 4ms |
| CPU time in execution | 0 | 0 | 0 | 0 |
| SQL server execution time | 1.08s | 83ms | 27ms | 52ms |

**2.4.2 Scenario2:**

list Incident ID where crime is reported in a different year than the crime has occurred.

**SELECT o.\_OFFENSE\_CATEGORY\_NAME\_, c.\_REPORTED\_DATE\_, c.\_FIRST\_OCCURRENCE\_DATE\_**

**FROM [denver\_crime].[dbo].[crime] c**

**LEFT JOIN [denver\_crime].[dbo]. [offense\_codes]**

**on o.[\_OFFENSE\_TYPE\_ID\_]= c.[\_OFFENSE\_TYPE\_ID\_]**

**WHERE c.[\_REPORTED\_DATE\_])> c.[\_FIRST\_OCCURRENCE\_DATE\_])**

The performance of the query without introducing any indexing is**729ms.**

Non-Clustered index is created on columns named \_REPORTED\_DATE\_ and \_FIRST\_OCCURRENCE\_DATE\_ in the entity crime.

Once we have performed the query using non-clustered index, the performance of the query degraded to **625ms.**

We have created the clustered index on the column \_INCIDENT\_ID\_of the entity CRIME and recorded the performance as **55ms.**

We have created the clustered index on \_OFFENSE\_CATEGORY\_NAME\_ of entity OFFENSE\_CODESand recorded the performance as**122ms.**

Based on the performance analysis, the performance has improved from non-clustered index to clustered index on the entity offense\_codes.

**2.4.3 Scenario3:**

List the count of offense type ID from the entity Crime.

**SELECT \_OFFENSE\_TYPE\_ID\_, count(\_OFFENSE\_TYPE\_ID\_) as offensetypecount**

**FROM [denver\_crime].[dbo].[crime]**

**GROUP BY \_OFFENSE\_TYPE\_ID\_**

**ORDER BY \_OFFENSE\_TYPE\_ID\_ DESC**

The performance of the query without introducing any indexing is: **293ms**.We have created Clustered indexon the column named\_OFFENSE\_TYPE\_ID\_ of the entity Offense\_codes.Once we have recorded the performance as **36ms**. We have created the clustered index on \_OFFENSE\_CATEGORY\_NAME\_ of entity OFFENSE\_CODE and recorded the performance as **67ms.**

Based on the performance analysis, the performance has improved from non-clustered index to clustered index on the entity offense\_codes.

|  |  |  |  |
| --- | --- | --- | --- |
| Query3 | Without  indexing | Clustered Index | Drop Clustered and create non-clustered |
| CPU parse time | 0 | 0 | 0 |
| Compile time | 4ms | 1ms | 1ms |
| CPU time in execution | 16ms | 0 | 0 |
| SQL server execution time | 293ms | 36ms | 67ms |

**2.3.4 Scenario 4:**

List the incident id’s, category name of offense and if the incident is crime.

**SELECT C.[\_INCIDENT\_ID\_], c.[\_IS\_CRIME\_], o.[\_OFFENSE\_CATEGORY\_NAME\_]**

**FROM [denver\_crime].[dbo].[crime] c, [denver\_crime].[dbo].[offense\_codes] o**

**Comparison of Query performance:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Query2 | Without  indexing | Non-Clustered Index | Clustered | Clustered on Offense\_Category\_Name |
| CPU parse time | 7ms | 0 | 0 | 0 |
| Compile time | 11ms | 7ms | 5ms | 1ms |
| CPU time in execution | 31ms | 63ms | 16ms | 31ms |
| SQL server execution time | 729ms | 625ms | 55ms | 122ms |

**WHERE o.[\_OFFENSE\_TYPE\_ID\_]=**

**c.[\_OFFENSE\_TYPE\_ID\_]**

**AND C.[\_OFFENSE\_CATEGORY\_ID\_]='drug-alcohol'**

The performance of the query without creating any clustered and non-clustered index is **1.39s.**

We have created the non-clustered index on the columns \_IS\_CRIME\_ and \_OFFENSE\_CATEGORY\_NAME\_ of entity offense\_codes. With the non-clustered index, the performance of the query is improved to **80ms**.

|  |  |  |
| --- | --- | --- |
| Query4 | Without indexing | Non-Clustered Index |
| CPU parse time | 13ms | 0 |
| Compile time | 13ms | 2ms |
| CPU time in execution | 0 | 0 |
| SQL server execution time | 1.39s | 80ms |

**2.3.5 Scenario 5:**

List of all incidents where the offense type is traffic-accident

**SELECTc.[\_INCIDENT\_ID\_],c.[\_IS\_CRIME\_],O.[\_OFFENSE\_CATEGORY\_NAME\_]**

**FROM [denver\_crime].[dbo].[crime] c, [denver\_crime].[dbo].[offense\_codes2] o**

**WHERE o.[\_OFFENSE\_TYPE\_ID\_]= c.[\_OFFENSE\_TYPE\_ID\_]**

**AND c.[\_OFFENSE\_TYPE\_ID\_]='traffic-accident'**

The performance of the query without creating any

Indexing is **1.42s.**

We created the clustered index on column \_OFFENSE\_TYPE\_ID\_ on entity offense\_codes and the performance improved to **77ms**.

|  |  |  |
| --- | --- | --- |
| Query5 | Without indexing | Clustered Index |
| CPU parse time | 0 | 0 |
| Compile time | 14ms | 8ms |
| CPU time in execution | 31ms | 0 |
| SQL server execution time | 1.42s | 77ms |

**2.3.6 Scenario 6:**

List the incident id’s and offense type id’s for neighborhood id either overland or montclair.

**SELECT DISTINCT c.[INCIDENT\_ID],o.[OFFENSE\_TYPE\_ID]**

**FROM [denver\_crime].[dbo].[crime] c**

**JOIN [denver\_crime].[dbo].[offense\_codes] o ON o.[OFFENSE\_TYPE\_ID]= c.[OFFENSE\_TYPE\_ID]**

**AND c.[NEIGHBORHOOD\_ID]='overland'**

**UNION**

**SELECT DISTINCT c.[INCIDENT\_ID],o.[OFFENSE\_TYPE\_ID]**

**FROM [denver\_crime].[dbo].[crime2] c**

**JOIN [denver\_crime].[dbo].[offense\_codes2] o ON**

**o.[\_OFFENSE\_TYPE\_ID]= c.[OFFENSE\_TYPE\_ID]**

**AND c.[NEIGHBORHOOD\_ID]='montclair'**

|  |  |  |
| --- | --- | --- |
| Query6 | Without indexing | Clustered Index |
| CPU parse time | 0 | 0 |
| Compile time | 21ms | 10ms |
| CPU time in execution | 15ms | 4ms |
| SQL server execution time | 590ms | 6ms |

The performance of the query without introducing

indexing is recorded as **590ms.** We created the clustered index on column **\_OFFENSE\_TYPE\_ID** of entity Offense-codes and we have seen the improvement in performance of **6ms.**

Partitioning large tables or indexes can have the following manageability and performance benefits.

# **3. Partitioning:**

* Data can be transferred quickly and efficiently, while maintaining the integrity of a data collection.
* Maintenance of partitioned tables is easy,

and operations are very fast, instead of accessing data in the entire table. For example, there will be an option to compress data in one or more partitions or rebuild one or more partitions of an index.

* Query performance can be improved based on the types of queries frequently run and on hardware configuration. For example, the query optimizer can process equijoin queries between two or more partitioned tables faster when the partitioning columns in the tables are the same because the partitions themselves can be joined.

In addition, performance can be improved by enabling lock escalation at the partition level instead of a whole table. This can reduce lock contention on the table.

## 3.1Horizontal Partitioning:

Horizontal partitioning involves creating two or more tables with exactly same structure and splitting rows between those tables. For instance, Antique Optical might use this technique to solve the problem with the order and order items tables becomingincreasinglylarge.

Optimized query performance is shown by implementing index and to enhance it further, ‘partition by’ function will be used.

**Approach to Partitioning:**

* Data in a partitioned table is partitioned based on a single column, the partition column, which is called as the partition key.
* It is important to select a partition column that is almost always used as a filter in queries. When the partition column is used as a filter in queries, SQL Server can access only the relevant partitions. This is called partition elimination and can greatly improve performance when querying large tables.
* If the partition column value is NULL, the rows are placed in the first partition.
* Partition Function: The partition function defines how to partition databased on the partition column. The partition function does not explicitly define the partitions and which rows are placed in each partition. Instead, the partition function specifies boundary values, the points between partitions. The total number of partitions is always the total number of boundary values + 1.
* Partition Scheme: The partition scheme maps the logical partitions to physical filegroups. It is possible to map each partition to its own filegroup or all partitions to one filegroup.

## 3.2 Filegroups:

First step in partitioning is to create the Filegroups. We use these to store the table partitions. For this demonstration, we will create the required number of files stored in equal number of filegroups spanned across four disk subsystems. The number of files and filegroups you create for your databases are bound by your available disk resources.

Assign each file to the respective filegroup.

Create partition function and schema based on partition key. In this part we define the ranges and map the ranges to each file group.

**3.2.1 Scenario 1:**

List distinct Incidents, Offence type and Crime year in the neighborhood 'overland' and 'montclair'

**SELECT DISTICT c.\_ INCIDENT\_ID, o.[OFFENSE\_TYPE\_ID], c.crimeyear**

**FROM [FastAnd Furious].[dbo].[crimes] c**

**,[FastAnd Furious].[dbo].[offense\_codes] o**

**WHERE o.[OFFENSE\_TYPE\_ID]= c.OFFENSE\_TYPE\_ID**

**and c.NEIGHBORHOOD\_ID in ('overland','montclair')**

In this scenario, we have evaluated the performance of the query without partitioning, with partitioning, with indexing and partitioning together.

**3.2.2. Scenario2**

List offense which are done on roads but not in the year 2015

**SELECT o. [OFFENSE\_CODE] ,o.[OFFENSE\_CATEGORY\_ID],o.[OFFENSE\_CATEGORY\_NAME],**

**o.[IS\_CRIME],o.[IS\_TRAFFIC]**

**FROM [FastAnd Furious].[dbo].[offense\_codes] o**

**INNER JOIN [FastAnd Furious].[dbo].[crimes]c ON o.OFFENSE\_TYPE\_ID=c.OFFENSE\_TYPE\_ID**

**WHERE o.OFFENSE\_TYPE\_NAME like '%vehicl%'and c.crimeyear!='2015'**

**3.2.3 Scenario3:**

List all incidents in year 2016 not related to Drug-Alcohol category or murder

**SELECT C.INCIDENT\_ID,C.IS\_CRIME,O.OFFENSE\_CATEGORY\_NAME FROM [FastAnd Furious].[dbo].[crimes] c**

**JOIN [FastAnd Furious].[dbo].[offense\_codes] o**

**ON o.[OFFENSE\_TYPE\_ID]= c.OFFENSE\_TYPE\_ID**

**AND C.OFFENSE\_CATEGORY\_ID='drug-alcohol' or C.OFFENSE\_CATEGORY\_ID like '%murd%'AND C.CRIMEYEAR='2016'**

**3.2.4 Scenario4:**

List offence codes which are even numbers and give the details of ID and name of the offense.

**SELECT o.[OFFENSE\_TYPE\_ID], o.OFFENSE\_TYPE\_NAME, o.OFFENSE\_CODE**

**FROM [dbo].[offense\_codes2] o**

**JOIN [dbo].[crime2] c ON c.OFFENSE\_CATEGORY\_ID = o.OFFENSE\_CATEGORY\_ID**

**WHERE o.OFFENSE\_CODE IN (SELECT o.OFFENSE\_CODE**

**FROM [dbo].[offense\_codes2]**

**WHERE o.\_OFFENSE\_CODE%2 = 0)**

**Comparison for Partitioning scenarios:**

|  |  |  |
| --- | --- | --- |
| Time comparison | Without  Partitioning | With partitioning |
| Query1 | 576ms | 44ms |
| Query2 | 1.08s | 32ms |
| Query3 | 1.22s | 29ms |
| Query4 | 1.5s | 65ms |

# **4. Software:**

* SSMS -SQL Server Management Studio 17.5 – (Database Client to access the server)
* SQL Server 2017 Express Edition - (Database server)
* SSIS – To Load the data into Database3.

# **5. Conclusion:**

The Denver crime data uses files that contain data about the crimes and offenses occurred in Denver, CO. Due to the huge and multi variate data, we have implemented indexing and partitioning on the two entities. We have implemented both clustered and non-clustered index on the entities and we have analyzed the changes in the performance of scenarios. We have also implemented partitioning on the two entities, we have analyzed the performance of few other scenarios and we have concluded that implementing indexing and partitioning will show immense improvement in data retrieval. Using both portioning and indexing is advisable for large and complex database.

# **6. References:**

[1] DBA from the cold, “Indexing and partitioning,” Ramblings on working as a SQL Server DBA, 2018. Available: <https://dbafromthecold.com/2018/02/21/indexing-and-partitioning/>. [Accessed Feb.19, 2019].

[2] T. P. S. d. group, "PostGre SQL 9.1.24 documentation". Documentation->PostGre SQL. [Online]. Available: <https://www.postgresql.org/docs/9.1/sql-explain.html>. [Accessed Feb. 27, 2019].

[3] Hans-Jurgen, Schonig, "PostGre SQL performance analysis," 3 ways to detect slow queries inPostGreSQL,2018. Available: <https://www.cybertec-postgresql.com/en/3-ways-to-detect-slow-queries-in-postgresql/> . [Accessed Feb. 27, 2019].

[4] Pavel, Tiunov, "Simple tips for PostGre SQL query optimization”.

Available:<https://statsbot.co/blog/postgresql-query-optimization/> . [Accessed Feb. 27, 2019].