Improving Speed Performance of Select Random Query in SQL Database

Muhammad Nur Yasir Utomo[1,a], Alvian Bastian[2,b] and Anggun Winursito[3,c]
[1,2] Department of Electrical Engineering, Politeknik Negeri Ujung Pandang, Makassar
[3] Department of Electrical Engineering and Information Technology, Universitas Gadjah Mada, Yogyakarta
[a] yasirutomo@poliupg.ac.id
[b] alvianbastian@poliupg.ac.id
[c] anggun.winursito@mail.ugm.ac.id

***Abstract*— Select random is a query in a SQL database that can retrieve data randomly from a table. Select random is often used to present data in various applications such as websites, data mining and others. Unfortunately, ordinary select random query is inefficient in terms of processing time if used in large table. This paper, tries to solve this problem by proposing two optimized methods of select random query, namely the Small Percentage Order by Rand (SPO-Rand) and the Filtered Column Order by Rand (FCO-Rand). The two proposed methods are then compared in terms of processing speed with a standard Select Random query or Normal Order by Rand (NO-Rand). The scenario of the experiment is to collect five random data from several data sets, ranging from 10.000 to 200.000 data. Based on the results of experiments that have been conducted, the proposed FCO-Rand method obtained the best process speed with 0.074 seconds at 200.000 data, followed by SPO-Rand with 0.265 seconds. These result are much faster than the standard random select method (NO-Rand) which takes up to 7,035 seconds for the same task.**

Keywords*— *SQL database; select random; SQL query speed; SQL optimization; relational database; data manipulation language

I. Introduction

Database is an important component in information technology to store various types of data with various purposes such as supporting web applications, mobile and data mining [1]. One of database type that is widely used currently is Structured Query Language (SQL) database [2], [3]. This database has a component called Data Manipulation Language (DML) that can be used to retrieve data easily [4].

One of the queries in DML SQL that can be used to retrieve data is a Select Random query. This is the query used to retrieve data randomly from a table in an SQL database. Based on its function, Select Random query is widely used in various applications such as article recommendations in website [5], random sample data selection in data mining [6], big data [7] and machine learning. However, the standard Select Random query in SQL has a disadvantage in terms of speed and time processing when used in tables with large data rows [8] or also called big tables. Research to speed up the process of Select Random is strongly needed.

Several researches have been conducted to solve the problem of slow processing time in SQL. Research [8] proposes query optimization rules to speed up SQL process. This research concludes that query speed does not only depend on infrastructure but also query optimization. The same concept is carried out by research [4] which also tries to optimize the query by using index and query reducing method. Research [4] concluded that the use of indexes makes queries work faster. The disadvantage of this research is that it really depends on index which is not always available in every table, in several cases it needs to be created first.

Other SQL performance optimization approach is proposed by research [9]. This research proposes query normalization method by replacing (*), all column statement, in every query to targeted unique column. Research [9] claimed that the proposed method can work faster than a normal query. The query normalization method is also conducted by research [1] and [10] to fasten the process of a query by utilizes the ordering / sequence concept. The results of these studies are very good to retrieve general data collection, but further

DOI : http://dx.doi.org/10.31963/intek.v7i1.1536

research is still needed for implementation in the case of random data collection.

Researches that have been described show that SQL query optimization can be done in several approaches such as query normalization and column specifications. This research, utilizing those approaches to propose two methods of query optimization for Select Random. The first method works with a schema to reduce the data that has to be scrambled and the second method works with randomization based on unique column filtering.

This paper is organized as follows. Section 1 explains the research objectives and studies related to SQL database queries. Section 2 explains the proposed methods of optimizing Select Random query in SQL database including the steps, comparison scenarios of each method and how the methods are evaluated. Section 3 explains the results and analysis of research experiments. Finally, section 4 concludes the result of this research.

II. Research Methodology

There are three main stages in this study, namely the collection of experimental data, experiments of proposed methods and evaluation of experimental results. These stages are described as follows:

A. Experimental Data and Setup

Experimental data used in this research came from address information portal called IDalamat.com. This portal has address data for various places in Indonesia. The total address data that can be used as experimental data is 200.000 address data. These data are stored in a table that has 23 fields / columns such as id, location name, category, phone number, description, coordinates etc.

All data that becomes experimental data is stored in a MySQL database. This MySQL database is installed in a Personal Computer (PC) with an Intel Core i5 processor, 8GB of RAM and uses Windows 10 operating system. Examples of experimental data are shown in Figure 1 below:

id_alamat	nama_lokasi	nama_lokasi_seo	deskripsi_lokasi	alamat_lokasi
1	Politeknik Negeri Ujung Pandang (PNUP)	politeknik-negeri-ujung-pandang-pnup	<p> Politeknik merupakan lembaga pendidikan tingg...	Jalan Perintis Kemerdekaan KM.10 Tamalanrea, Makas...
2	Universitas Hasanuddin Makassar (UNHAS)	universitas-hasanuddin-makassar-unhas	Universitas Hasanuddin (UNHAS) merupakan salah sat...	Jl. Perintis Kemerdekaan Km.10 Makassar, 90245, Ko...
3	Universitas Muslim Indonesia (UMI) Makassar	universitas-muslim-indonesia-umi-makassar	<p>Universitas Muslim Indonesia (UMI) adalah unive...	Menara UMI Lt. 09, Urip Sumohardjo KM.05 Makassar,...
4	Universitas Negeri Makassar (UNM)	universitas-negeri-makassar-unm	Universitas Negeri Makassar, disingkat UNM, adalah...	Kampus Gunungsari Baru Jl. A.P. Pettarani Makassar...
5	Universitas 45 Bosowa Makassar	universitas-45-bosowa-makassar	Universitas 45 Makassar adalah perguruan tinggi sw...	JL. Urip Sumoharjo Km. 4 Makassar, Kota Makassar, ...

Figure 1. Examples of Experimental Data

B. Select Random Methods

Experiments in this study were conducted using three different Select Random query methods. Each method used can be explained as follows:

1. *Normal Order by Rand* (NO-Rand)

Normal Order by Rand (NO-Rand) method is a standard form of random data query in SQL that use `ORDER BY RAND` command in SQL without any modification and additions. Because its simplicity, this query is also the most commonly used method for random data retrieval. Example of Normal Order by Rand query is shown as follows:

```
SELECT * FROM table_name ORDER BY RAND()
LIMIT 5;
```

The query above works by creating a random fragment value for each row of data in a table, it then sort the data based on random fragment values that have been created. Finally, the query display five data in accordance with the defined "`LIMIT 5`".

Normal Order by Rand method does not have to have an `auto_increment` type of column in the table so it is flexible. It also easy to remember. However, this query also has disadvantage, this method is very dependent on the speed of making random fragment value for each row. The greater the data and the more fields it has will make NO-Rand method query become slower.

DOI : http://dx.doi.org/10.31963/intek.v7i1.1536

2. *Small Percentage Order by Rand* (SPO-Rand)

Small Percentage Order by Rand method is a query that performs random data retrieval by reducing the percentage of data that must be scrambled from all existing data rows. This method eliminates a large amount of data and leaves a small percentage of it to be scramble. The elimination process conducted randomly and the size of eliminated data is depend on the target number of data that user want to retrieve. For example, the following is a random query using SPO-Rand:

```
SELECT * FROM table_name WHERE
RAND()<(SELECT ((5/COUNT(*))*10) FROM
table_name) ORDER BY RAND() LIMIT 5;
```

Above query retrieves five random data from the reduced data set using the `WHERE RAND ()` query. The percentage of data reduced is based on the random fragment value of each data row, if the data fragment value is greater than the defined limit then the data will be eliminated. The limit of the fragment value itself is made by the following equation:

$$Fragmen\ Limit = \frac{Data\ to\ Retrive}{Total\ Data} x100 \qquad (1)$$

For example the data to be taken randomly is 5 data from 200,000 (200K) data, the limit value of the fragment is (5/200k)*10=0.00025. This value becomes the basis to eliminate data with greater fragment value. In the end, only a small amount of data is randomized, so the query can work faster.

3. *Filtered Column Order by Rand* (FCO-Rand)

FCO-Rand method is a method that works with the same concept as select random in the first method, the difference is the random process is limited to only one column. This column is selected based on its type. It has to be `primary key` and `auto_increment` type of column.

The random process that only applies to specific fields makes the query does not need to calculate and consider fragment value for other column, thus making random query can work faster. An example of FCO-Rand query is shown as follows:

```
SELECT * FROM table_name WHERE id IN
(SELECT id FROM (SELECT id FROM
table_name ORDER BY RAND() LIMIT 5) t);
```

Random query above works in three stages. First stage is the random process in the "id" column (`SELECT id`) with five limit data to retrieve (`LIMIT 5`). The results of the first stage are five random "id" data. Second stage then stored these "id" data in the alias table "t". In the third stage, five "id" data in the alias table "t" are then used to retrieve complete data (all columns) (`SELECT *`) related to each "id" (`WHERE id`).

C. Performance Evaluation

Performance evaluation is done by creating data sets ranging from 10.000, 50.000, 100.000, 150.000, to 200.000 data. Each query method will be assigned the same task which is to collect five data randomly from each data set. For each data set, Select Random methods will be tested five times. The speed of the query process for each experiment will be recorded.

Query speed value for each experiment is obtained based on the calculation of phpMyAdmin console. For each query executed through this console, query processing time will appear as shown in the following figure:

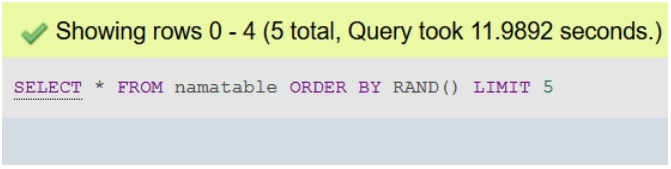

Figure 2. Data Acquisition of SQL Query Processing Time

After conducting five experiments for each method in each dataset, an evaluation of query speed performance can be performed. For each method in each dataset, the speed value will be totaled, then the average speed query can be calculated using the following equation:

$$\bar{x} = \frac{Sum\ of\ Speed\ Value}{Number\ of\ Experiment} = \frac{x_1 + x_2 + \cdots + x_n}{n} \qquad (2)$$

In the end, speed comparison results of each method can be obtained. This result then used to conclude the best method for Select Random query in SQL database.

DOI : http://dx.doi.org/10.31963/intek.v7i1.1536

III. Results and Discussion

The experimental results in this research are divided into two which are speed performance results of each method for each dataset and performance comparison of Select Random methods. Each of these results can be explained as follows:

A. Query Speed Results of the Select Random Method

1. Select Random Results on 10.000 Data Rows

Speed performance results for 10.000 data rows are shown in the following Table 1:

Table 1. Experimental Results of the Methods on 10.000 Data

10.000 data	Processing Time (Seconds)		
Num. of Trials	NO-Rand	SPO-Rand	FCO-Rand
1	0.2725	0.189	0.0057
2	0.2635	0.1848	0.0068
3	0.2566	0.1976	0.0067
4	0.2838	0.1904	0.0058
5	0.2883	0.1849	0.0061
Average	0.27294	0.18934	**0.00622**

Table 1 above shows the speed of each Select Random methods used to retrieve five data from a table with 10.000 rows of data. The results shows that all methods are fast enough where the speed still less than 1 second. Method with the highest speed is the Filtered Column Order by Rand or FCO-Rand method.

2. Select Random Results on 50.000 Data Rows

Speed performance results for 50.000 data rows are shown in the following Table 2:

Table 2. Experimental Results of the Methods on 50.000 Data

50.000 data	Processing Time (Seconds)		
Num. of Trials	NO-Rand	SPO-Rand	FCO-Rand
1	1.3693	0.233	0.0604
2	1.1651	0.1826	0.0198
3	0.9505	0.243	0.0166
4	0.9347	0.1984	0.0188
5	0.9783	0.2779	0.0243
Average	1.07958	0.22698	**0.02798**

Table 2 shows that the usual random select method (NO-Rand) starts to show poor performance when used in table with 50.000 data, NO-Rand method has exceeded one second. In contrast, two proposed methods namely SPO-Rand and FCO-Rand still show good performance, which is under 0.5 seconds.

3. Select Random Results on 100.000 Data Rows

Results of Random Select methods query in a table with 100.000 data are shown in the following table:

Table 3. Experimental Results of the Methods on 100.000 Data

100.000 data	Processing Time (Seconds)		
Num. of Trials	NO-Rand	SPO-Rand	FCO-Rand
1	3.0271	0.2102	0.0325
2	3.1651	0.2324	0.035
3	2.9405	0.2176	0.0323
4	3.1316	0.2803	0.0394
5	3.0133	0.2347	0.0367
Average	3.05552	0.23504	**0.03518**

Table 3 shows that NO-Rand method has been seen working with slow speed. NO-Rand method takes 3.055 seconds to be processed while the FCO-Rand gets 0.035 seconds for the same task, this makes FCO-RAND become the fastest method to retrieve five data randomly from 100.000 data rows.

4. Select Random Results on 150.000 Data Rows

Performance of Select Random methods on 150.000 rows of data is shown in following table:

Table 4. Experimental Results of the Methods on 150.000 Data

150.000 data	Processing Time (Seconds)		
Num. of Trials	NO-Rand	SPO-Rand	FCO-Rand
1	4.3569	0.2309	0.062
2	4.6136	0.246	0.0469
3	4.8985	0.2445	0.0805
4	4.4997	0.2513	0.0677
5	4.1664	0.2632	0.0731
Average	4.50702	0.24718	**0.06604**

It can be seen that the FCO-Rand method can still work quickly, its only need 0.066 seconds to complete the process. In contrasts, NO-Rand method become the slowest method with 4.5 seconds for the same task.

DOI : http://dx.doi.org/10.31963/intek.v7i1.1536

5. Select Random Results on 200.000 Data Rows

Results of Select Random methods on 200.000 rows of data are shown in the following table:

Table 5. Experimental Results of the Methods on 200.000 Data

200.000 data	Processing Time (Seconds)		
Num. of Trials	NO-Rand	SPO-Rand	FCO-Rand
1	7.6417	0.2561	0.0796
2	7.1837	0.2733	0.0782
3	6.3504	0.2686	0.0687
4	7.2713	0.2675	0.073
5	6.7289	0.2603	0.0712
Average	7.0352	0.26516	**0.07414**

In 200.000 rows of data experiment, the best Select Random method is FCO-Rand that complete its process only in 0.074 seconds, followed by SPO-Rand with 0.265 seconds and NO-Rand with 7.0352 seconds.

B. Speed Performance Comparison

Based on experiment conducted in the previous section, the comparison of each method for each data set can be described in Table 6 and Figure 3 as follows:

Table 6. Speed Comparison of Select Random Methods

Num. of Data Rows	Average Processing Time (Seconds)		
	NO-Rand	SPO-Rand	FCO-Rand
10.000	0.273	0.189	**0.006**
50.000	1.080	0.227	**0.028**
100.000	3.056	0.235	**0.035**
150.000	4.507	0.247	**0.066**
200.000	7.035	0.265	**0.074**

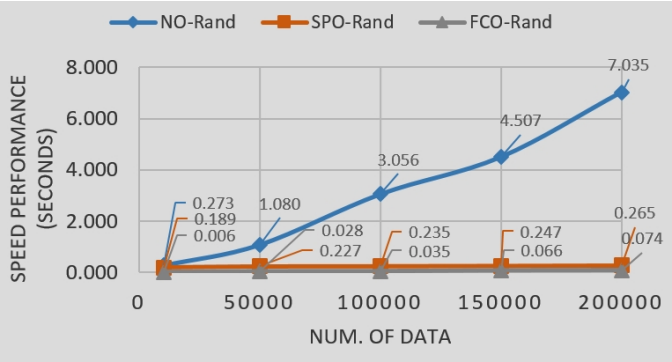

Figure 3. Performance Comparison of Select Random Methods

Table 6 and Figure 3 above show that the usual random method, the NO-Rand method, can only work well on table with small number of data or table with no more than 10.000 data. If used in a larger table (50.000 or more data rows), the NO-Rand method can no longer be recommended. In contrast, the proposed FCO-Rand method can work consistently fast on all datasets. Even for large table with 200.000 rows of data, the FCO-Rand speed is consistently below 0.074 seconds as shown in the following figure:

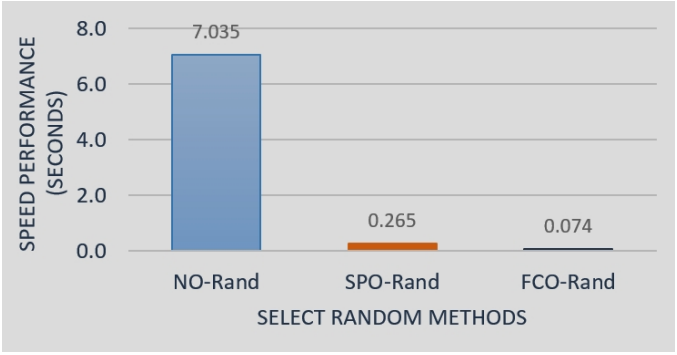

Figure 4. Method Speed Comparison of 200.000 Data

Figure 4 shows the significant speed difference between the Normal Order by Rand (NO-Rand) method and the proposed methods, namely Small Percentage Order by Rand (SPO-Rand) and Filtered Column Order by Rand (FCO-Rand). SPO-Rand is 6.77 seconds faster than usual Select Random query / NO-Rand and FCO-Rand method is 6.96 seconds faster than NO-Rand method.

Based on these results, this research recommends FCO-Rand Select Random method as the best method for the task of retrieving random data on tables with large amounts of data.

IV. Conclusion

This research proposes two optimization methods for Select Random query in SQL database to solve speed performance problem of usual Select Random query. The first method is Small Percentage Order by Rand (SPO-Rand) which works by eliminating a certain amount of data randomly before taking the actual random data. The second method is Filtered Column Order by Rand (FCO-Rand) that use column specifications approach. Based on the experiment that

DOI : http://dx.doi.org/10.31963/intek.v7i1.1536

have been conducted, the two proposed methods have fast and consistent performance. For the task of retrieving five random data from 200.000 rows of data, the usual Select Random method (NO-Rand) takes 7.035 seconds, SPO-Rand takes 0.265 seconds and FCO-Rand only takes 0.074 seconds for the same task. This research finally concludes and recommends the FCO-Rand method for Select Random query in SQL database tables with large number of data (big table). As suggestion, further research can be done by increasing the amount of experimental data to more than 200.000 data to test the consistency of proposed select random methods.

Acknowledgement

This work is partially supported by IDalamat.com which has provided dataset used for experiment and analysis.

References

[1] N. Sangeeth and R. Rejimoan, "An Intelligent System for Information Extraction from Relational Database using HMM," International Conference on Soft Computing Techniques and Implementations, ICSCTI 2015, pp. 14–17, 2015.

[2] V. K. Myalapalli and P. R. Savarapu, "High Performance SQL Finesse for Lucrative Programming," Annual IEEE India Conference (INDICON), 2014.

[3] J. L. Viescas, D. S. Steele, and B. G. Clothier, Effective SQL: 61 Specific Ways to Write Better SQL. Addison-Wesley, 2017.

[4] D. Saisanguansat and P. Jeatrakul, "Improving Optimization Performance on PL/SQL," International Conference on ICT and Knowledge Engineering, pp. 1–6, 2017.

[5] H. Halimi and I. Jound, "Comparison of Performance between Raw SQL and Eloquent ORM in Laravel," Blekinge Institute of Technology, 2016.

[6] M. N. Y. Utomo, A. E. Permanasari, E. Tungadi, and I. Syamsuddin, "Determining Single Tuition Fee of Higher Education in Indonesia: A Comparative Analysis of Data Mining Classification Algorithms," in Proceedings of 2017 4th International Conference on New Media Studies, CONMEDIA 2017, pp. 113–117, 2017.

[7] S. Minukhin, V. Fedko, and D. Sitnikov, "SQL-On-Hadoop Systems: Evaluting Performance of Polybase for Big Data Processing," International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T), no. 1, pp. 591–594, 2018.

[8] J. Habimana, "Query Optimization Techniques - Tips For Writing Efficient And Faster SQL Queries," International Journal of Scientific & Technology Research, vol. 4, no. 10, pp. 22–26, 2015.

[9] F. Mithani, S. Machchhar, and F. Jasdanwala, "A Novel Approach for SQL Query Optimization," International Conference on Computational Intelligence and Computing Research (ICCIC), 2016.

[10] V. Leis, K. Kundhikanjana, A. Kemper, and T. Neumann, "Efficient Processing of Window Functions in Analytical SQL Queries," Proceedings of the VLDB Endowment, vol. 8, no. 10, pp. 1058–1069, 2015.

DOI : http://dx.doi.org/10.31963/intek.v7i1.1536