Context-Aware Query Performance Optimization for Big Data Analytics in Healthcare

Manoj Muniswamaiah, Tilak Agerwala, Charles C. Tappert
Seidenberg School of CSIS, Pace University, White Plains, New York
{mm42526w, tagerwala, ctappert}@pace.edu

Abstract **Big data analytics is playing a critical role in the healthcare industry in providing better healthcare delivery to patients and in disease exploration research. New big data tools have been developed which help in integrating and analyzing structured and unstructured data produced by different healthcare systems. Different databases have been used to store and process these healthcare-related data. In this paper, we propose and evaluate a cost-based, context-aware query optimizer which executes queries quickly and efficiently, while improving its performance.**

Index Terms **Big data, analytics, database, cost-based query optimizer.**

I. INTRODUCTION

Big data is characterized by its volume, velocity, variety, and complexity, which has led to the adoption of new hardware and software for successful processing of such data. The healthcare industry produces data which is spread among different healthcare systems, research institutes, and health insurance agencies. These systems are siloed, which makes integration of data and providing a global integrated view a challenging task. Determining the veracity of this collected healthcare data is critical for researchers to obtain meaningful insight. Advancements in technology have made it possible to capture data over a long period of time. The structured and unstructured data produced from clinical systems are aggregated to understand and predict the diseases. The characteristics and taxonomy of the data help in gaining meaningful insights for further analysis. Storing medical images requires large storage capacity and algorithms to efficiently process them. Signal processing systems also produce data which is high in volume and velocity from the monitoring systems connected to the patients. This type of output requires analysis and correlation of multi-modal time series data generated from monitoring devices. The adoption of big data analytics has the potential to save lives and to improve patient care delivery and healthcare practices.

Healthcare analytics helps to treat patients in cost effective ways and also to build predictive models that are utilized in the production of drugs. Furthermore, these analytics can determine the disease patterns from outbreaks, improve vaccine development, perform gene sequencing, and monitor real-time data produced from monitoring devices that are time series based [1].

Recent technology advancement in virtualization and scalable cloud computing resources has been beneficial for the storage and management of data. Volume, veracity, velocity, and variety of big data pose new challenges for analysis and decision making. Monitoring and analyzing real-time data from the medical devices helps healthcare workers in taking appropriate actions to address trauma and infections. Relational databases are used to store structured data which can be queried easily. Unstructured data, such as medical device readings, medical images, genetics, and waveform data, are stored in NoSQL data stores [1].

Integrating, storing, and analyzing heterogeneous data from data sources is a tedious and challenging task. Healthcare systems need tools which are effective in collecting and transforming unstructured data into structured data for analysis by researchers. Structured data which is well defined, including patients' bio data, can be stored in relational databases. More recent data stores such as NoSQL and NewSQL are used to store time series and waveform data for scalability and retrieval.

The benefit of big data analytics in healthcare lies in integrating and analyzing data which help in practical research and discovery of patterns of diseases and outbreaks. Extensive analysis of data from different data stores is required of pharmaceutical researchers in order to facilitate integration of genomic and clinical data and help produce drugs quickly [2].

Extracting insight from big data is one of the quickest and most effective ways to improve people's healthcare. Descriptive analytics answer queries related to what happened. Diagnostic analytics provide reasoning about why it happened. Predictive analytics provide information about what is going to

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

happen, and prescriptive analytics provide details about how we can make it happen [2].

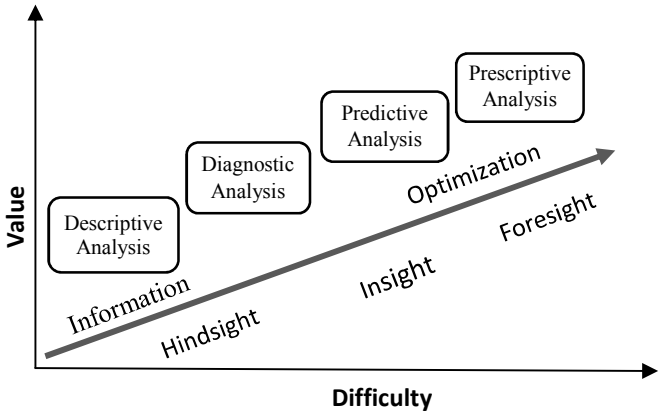


Figure 1: Various types of analytics

Healthcare big data analytics is executed on massive parallel and distributed servers. Data scientists need to perform statistics and execute algorithms and visualizations on the databases efficiently. Traditionally, extract-transform-load (ETL) and, more recently, data virtualization processes have been adopted to collect, transform, and store data in databases. Improving healthcare analytics depends on effective retrieval and analysis of data from different systems [2].

In order to have an integrated view of the data and perform analyses, ETL or data virtualization processes need to be followed. This helps in bringing together data from different data sources. The integrated data undergoes various cleaning and transformation steps before been presented to the researchers.

In an organization database administrators have to define materialized views, partition tables, and create indexed columns on databases; then they need to communicate and document these changes to the users. In this paper, we improve a cost-based query optimizer which makes use of the external optimized copy registered in a columnar database and improves the performance of the analytical query execution time. This improvement is achieved by rewriting the query plan to make use of the externally registered optimized copy of the data in the relational expression tree during the query optimization step of the analytical query execution. The optimized copy created by the database administrator from all the different databases is stored separately in a columnar database for faster reads. The cost-based query optimizer determines whether the cost of execution of the query plan can be reduced by substituting the optimized copy with the incoming query and later executing it against the optimized copy stored in the columnar database instead of the actual base table. The results show that this improvement to the query optimizer reduces the query execution time and provides better performance.

II. BACKGROUND

The success of healthcare organizations depends on the decision making processes and the quality of data used to aid this process. Data-driven decision making requires integration and analysis of data from heterogeneous systems. Data integration is an essential step used both in the ETL process and data virtualization to obtain a single holistic view of different data stores which have varied data models.

Electronic health records (EHR) are primary data sources used by researchers to access and analyze healthcare data. Data from multiple EHR systems need to be cleaned and transformed into a common integrated data model. Data synchronization involves significant time, effort, and computing resources. A common challenge with ETL and data virtualization processes is compatibility between source and required target data for analysis. Local healthcare systems often have different data formats which need to be cleaned, transformed, and stored in databases for analysis and scalability. ETL and data virtualization follow series of operations that allows source and target to be syntactically and semantically synchronized [3].

ETL and data virtualization techniques can be supported using data integration tools and technologies such as Denodo, NoSQL data stores, Pentaho, and Map/Reduce. Querying different data stores is challenging because of the data models being used to store the data and varied syntax of the query language. Database administrators create and store optimized copies and indexes for faster query execution and better performance, which is a manual and tedious process. In a big data environment with multiple databases, the above tasks become challenging to implement and maintain. Hence, a cost-based query optimizer which detects these optimized copies and substitutes them in the query plan during execution is required. Such an optimizer also reduces execution time and increases the performance of the query [4].

Data quality is important in order to have reliable results in research where data integration is expensive. Data provenance provides understanding of how data was collected, cleaned, and transformed. This record is important not only for reproducibility but also for the outcome of the clinical research. Data provenance is a key aspect in understanding the origin of data, which, in turn, influences the analysis significantly. The combination of multi-modal data results in challenges related to interoperability. Interoperability issues commonly result from differing national and international health data standards.

Medical research is a data-driven science; however, most of the multi-modal data being generated is unstructured. The goal should be to obtain information from such data sources and make information available to clinicians. Medical data comes with complex metadata which needs to be considered in order to have appropriate hypotheses and for clinical decisions. Time-critical healthcare applications require actions to be taken instantaneously based on the insight that has been derived from heterogeneous data streams. Data streaming refers to the ability to analyze and process the data quickly, which requires high-end, in-memory databases for processing rather than storing and retrieving the data at later point. Complex events detection refers to discovery of patterns in multiple streams of data

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

sources which are semi-structured in nature and need to be appropriately stored in NoSQL for faster retrieval and manipulation.

Earlier traditional relational databases were used to perform all the tasks due to their querying and storing capabilities. However, today these databases face challenges in meeting the performance requirements of big data. NoSQL data stores are scalable and process significant amounts of unstructured data without performance degradation. NoSQL database stores were built to deal with complex unstructured data, ensure high availability, are horizontally scalable, and support distributed parallel processing. Each NoSQL stores a different type of data. Key-value stores consist of keys and their corresponding values, and they support schema-less data storage. This helps in quick search operation over millions of rows. Document stores consist of documents with data in XML or JSON format. These stores also allow for storage of semi-structured data. Graph stores consist of nodes linked by edges. NoSQL data stores operate without schema, which allows for the addition of records without having to make changes to the structure. Healthcare organizations use different data management systems to store different data formats. Hybrid transactional and analytical processing systems (HTAP) are databases that exhibit the characteristics of operational databases as well as analytical systems. HTAP can be used for healthcare applications that are intelligent – that is, they require analysis of real-time data and provide useful insights. HTAP enables new technologies such as machine learning, real-time reporting, and reactive systems. Reactive systems, stream or batch process the requests which need to be analyzed later. Healthcare applications such as electronic medical records alert clinicians about critical conditions of patients. By integrating machine learning models with HTAP, applications can become more adaptive and help clinicians more effectively. Columnar databases are designed for read heavy analytical queries and are often used to store optimized copies created by database administrators in big data environments for faster reads.

III. METHOD

Apache Calcite framework provides query processing and support to many data-processing systems. This framework has many built-in optimization rules and extensible adaptor architecture to provide support for various data stores. Organizations have built data-processing systems to suit their requirements. Apache Calcite was developed to support efficient queries across different data systems [7]. Apache Calcite query optimizer is based on the Volcano and Cascades frameworks, which are extended in our research work to provide better performance and reduce query execution time by making use of optimized copies created by database administrators. Spark SQL is a module that has been integrated with Apache Spark for relational query processing. Spark SQL provides DataFrame API that supports relational query capabilities and it also includes the query optimizer Catalyst, which adds data sources and rules for query processing. Spark SQL also executes queries over heterogeneous data stores, but the Catalyst optimizer, which is cost-based, lacks the support of dynamic programming optimization as in Apache Calcite [5]. Another federated data-query framework is the BIGDAWG. This framework supports location independence, which assists the query in figuring out the intended target data store and ensures semantic completeness; consequently, the framework can make use of features being supported by native data stores. The island component provides location independence, whereas the shim component allows cross-island query capabilities [6]. The Apache Calcite framework is built on relational algebra for query processing and optimization. The framework supports complex data manipulation operations – including window operator, which is a common operation in analytical and streaming applications to execute aggregate-related queries. Apache Calcite does not have different objects to represent logical and physical operations. Instead, it uses traits to describe the physical properties of the operators, which also helps the optimizer to evaluate the different costs of the query plans. Converter interfaces help in converting traits of the relational expression from one form to another and help in execution against the backend databases. Executing calling conventions as traits allows for the transparent performance of optimizations where the queries might reach across heterogeneous data stores. Traits represent expressions which are executed against the data engines. Scans of tables residing in different data stores are done in their respective conventions. One of the primary tasks of the optimizer is to find plans that are expensive in terms of the resource usage and prevent the bottlenecks they create in order for the subsequent transactions to be performed effectively. The addition of different traits helps the query optimizer with effective query execution against different databases which have their own in-built features to handle data processing. Apache Calcite integrates different databases by extending the implementation of adaptors, and it defines models which are integral components of adaptors that include the physical properties of the database being queried. Schemas are a part of the adaptor model and consist of data formats and layouts. The adaptors define a set of transformation rules which are applied to the logical expression to get an optimal relational expression using convention calls. A schema factory generates the related schema by making use of the metadata provided to it [7].

When a query is issued, it is validated for the query language syntax and generates an abstract syntax tree which is later converted into a logical expression to be passed to the query optimizer for generation of an optimal plan. During this stage, operators are generated which are used by the optimizer to scan tables and perform operations such as filtering and sorting to reduce the cost of the plan. Apache Calcite supports enumerable calling conventions which act as an iterator interface over databases with millions of rows. An optimizer effectively handles analytical queries which require small subsets of rows to be scanned while iterating over an entire table. The optimizer pushes down all possible logic to the backend data stores and

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

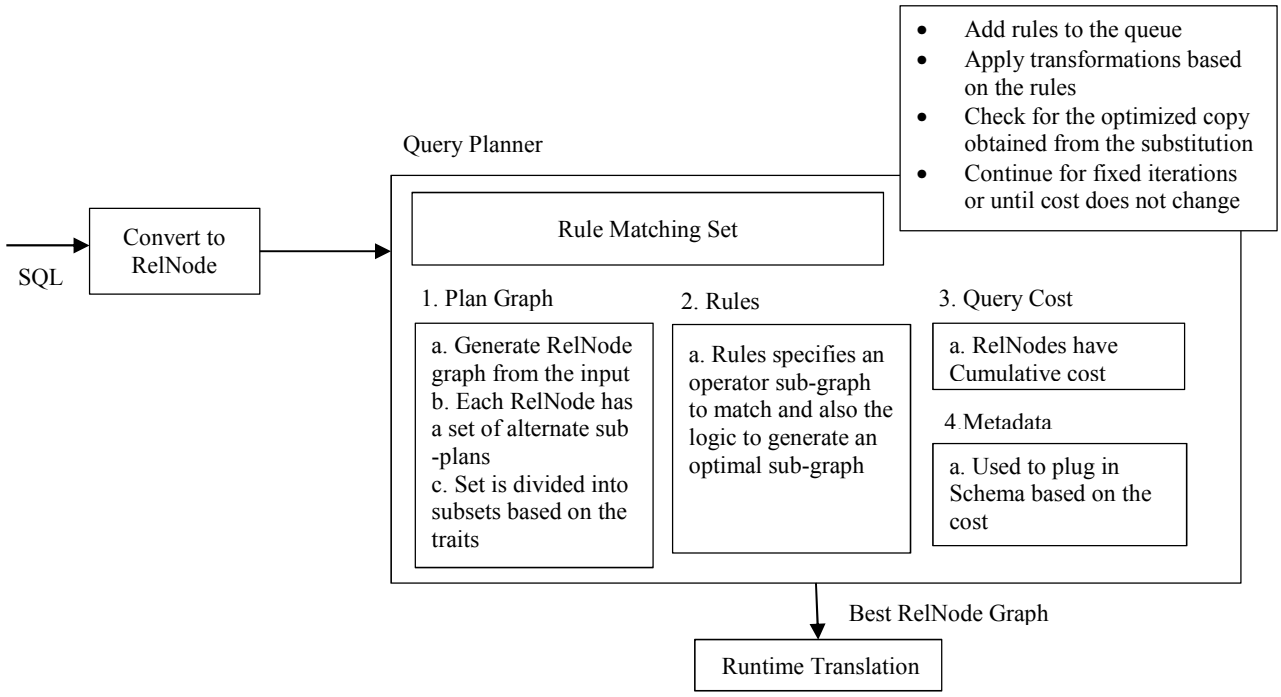


Figure 2: Query Optimizer Components [7]

then performs the required sorting and aggregations on the resulting data being returned [7].

IV. QUERY OPTIMIZER

The query optimizer is a volcano cost-based optimizer implemented using dynamic programming which repeatedly applies the planner rules to perform transformations on the relational node which is generated from the abstract syntax tree from the input query. The cost module of the optimizer generates different alternative relational expressions that produce the same semantics as the original relational expression but with reduced resource cost. The optimizer contains a set of rules and, when a rule matches a pattern, the optimizer performs the transformation which results in the substitution of the sub-tree of the relational node and still preserves the semantics of the relational expression. Most of these rules for performing transformations and executing the query are specific to individual databases [7].

The optimizer consists of a cost module which provides details on the metadata being used to reduce the cost of executing the query plan and also assists the planner rules to perform the related transformations. Metadata functionalities determine the overall cost of execution of the relational expression, taking into consideration the amount of data scanned in the tables, the degree of concurrency and parallelism that can be achieved through a single node or cluster of nodes, and also the sorting and filter conditions to be performed on the queries [7].

Apache Calcite uses a volcano cost-based planner engine which triggers various rules in order to reduce the cost of the query execution. A dynamic programming algorithm, which creates different plans by transforming the relational expression, is used to implement the volcano optimizer. The relational expressions are initially registered with the planner. When a planner rule is triggered on relational expression Expr1, it produces a new transformed relational expression, Expr2. The planner later adds Expr2 to the set of equivalence expressions which have the same semantics (Set1) present in Expr1. The planner then compares the new relational expression, Expr2, with the previously registered relational expressions. If there are any duplicates in relational expression Expr3 which are in the same equivalence set (Set3), then the planner will integrate both Set1 and Set3 into a new set of equivalence relations. This iteration of the query optimizer continues until the cost plan is not improved significantly compared to the threshold value. The cost module of the optimizer takes into consideration the IO resource access cost, memory, and the CPU cycles to come up with an optimal plan [7].

One of the methodologies which have been successfully adopted in analytical processing of queries is to use optimized copies which are created by database administrators using big data analytics. The query optimizer has the ability to make use of these optimized copies to rewrite the incoming queries and use these copies, which improves performance and reduces query execution time. The cost-based optimizer which is extended successfully substitutes part of the relational expression tree with optimized copies. Later, the substituted copies are registered with the planner and transformation rules are fired to validate if the cost can be reduced further [7].

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

Figure 2 shows the modules of the extended query optimizer where the incoming query is being parsed and validated for syntax errors and converted into an abstract syntax tree which forms the relational node expression. Each relational expression has a trait associated with it which is specific to the native database. Planner rules are added to the queue and get triggered when a pattern is matched, resulting in a partial or whole transformation of the sub-graph of the relational expression without changing the semantic value. The planner continues to apply these rules until the cost of the relational expression remains constant or does not improve much in relation to the threshold value. The cost model of the optimizer provides the cumulative cost of the relational expression and the metadata provider gives information for the generation of the schema.

Algorithm overview

Query optimization for relational expression R:

1. Register relational expression R with query optimizer.
a. Check for the existence of an appropriate optimized copy that can be used to substitute in the relational expression R.
b. If such a copy exists, then register the new relational expression R1, which has been substituted with the optimized copy to reduce the logical plan space and cost.
c. The query planner triggers the transformation rules on relational expression R1.
d. The query optimizer obtains the best possible relational expression when the cost minimizes and remains constant.
e. The optimizer uses the optimal relational expression and converts it into an appropriate physical plan to be executed against the native database.
2. If the query optimizer cannot substitute the relational expression R with an optimized copy, then:
a. The optimizer applies the transformation rules on relational expression R.
b. The optimizer finds the best relational node when the cost factor does not improve further and executes it against the native database.

The optimizer triggers transformation rules on operands of the parsed query to optimize and reduce the cost of the query plan. Transformation rules such as the Push Filter project rule and Combine Project rule are performed on the operands of MIMIC-III data tables [8].

```
Project (patient_id)                                [Exp1]
  Filter (gender='F')                               [Exp2]
    Project (patient_id, gender,dob)                [Exp3]
      Project (patient_id, gender,dob,expireflag)   [Exp4]
        TableScan (patient)                         [Exp0]
```

Apply PushFilterThroughProjectRule to [Exp2, Exp3]:

```
Project (patient_id)                                [Exp1]
  Project (patient_id, gender,dob)                  [Exp5]
    Filter (gender='F')                             [Exp6]
      Project (patient_id, gender,dob,expireflag)   [Exp4]
        TableScan (patient)                         [Exp0]
```

When Exp5 is registered it triggers CombineProjectsRule[Exp1,Exp5] resulting in

```
Project (patient_id)                                [Exp7]
  Filter (gender='F')                               [Exp6]
    Project (patient_id, gender,dob,expireflag)     [Exp4]
      TableScan (patient)                           [Exp0]
```

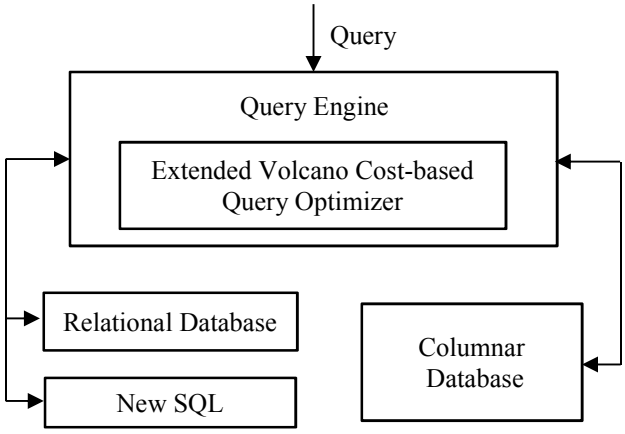


Figure 3: SQL query framework

Figure 3 shows the architecture of the SQL query framework with the extension to the cost-based volcano query optimizer which parses the incoming query, performs the required validation, and generates the abstract syntax tree which is converted into the relational expression. During the run time, the optimizer substitutes the query plan with optimized copy, if applicable, and executes the query to reduce the latency and improves the performance. If an optimized copy is found, it is executed against the columnar database where the optimized copies are stored by database administrators, or it is executed against the native databases.

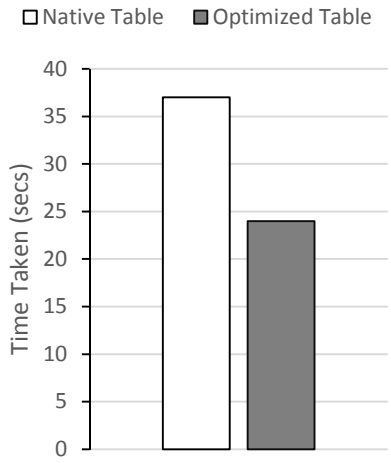
V. EVALUATION

The experimental setup outline below was designed to evaluate the performance of the extended cost-based query optimizer which substitutes the optimized copy to the issued query, if applicable, in order to reduce the query time and improve the performance. Columnar databases are used for storing optimized copies. A healthcare MIMIC-III dataset was used to evaluate the performance of the optimizer [8]. Data were stored in Oracle [9], MySQL [10] and the optimized copies were stored in a MonetDB database [11].

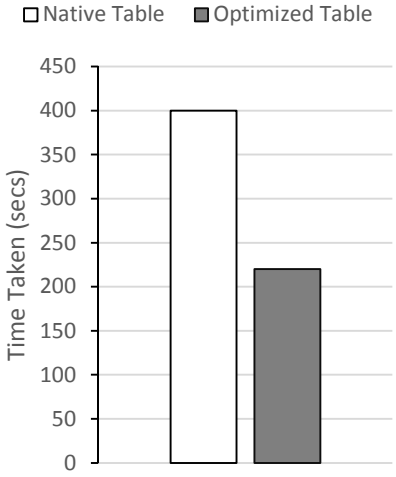
978-1-7281-5020-8/19/$31.00 ©2019 IEEE

The query optimizer performance evaluation was obtained based on the following setup:

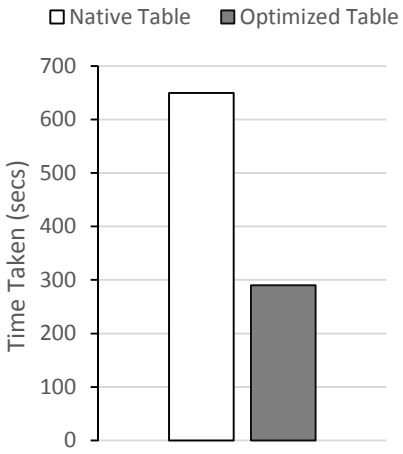
a.) Tables had rows ranging from ~4,000 to ~8,000,000.
b.) Oracle, MySQL, and MonetDB databases were running on single node instance with Intel Quad-Core 3.1 GHz, 24GB RAM and 1TB HDD.
c.) Base table data were stored in respective databases.
d.) Optimized copies were precomputed and stored in MonetDB.
e.) Extended volcano cost-based SQL query optimizer was implemented to substitute and execute optimized copy in the query plan, when found.



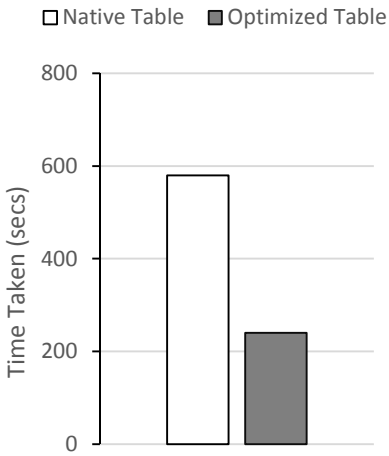
Graph 1: select gender, count (*) from patients group by gender [8]



Graph 2: select expire_flag, count (*) from patients group by expire_flag [8]



Graph 3: select subject_id, hadm_id, marital_status, admission_type group by admission_type from demographic_detail [8]



Graph 4: select p.subject_id, p.dob, a.hadm_id, a.admittime, p.expire_flag from admissions a inner join patients p on p.subject_id = a.subject_id [8]

As shown in the above graphs, different analytical queries were executed against the base table compared with optimized copies stored in columnar databases with greater performance improvement. The extended query optimizer automatically substituted the given query relational expression with the optimized copy relational expression, when applicable, and executed the query.

VI. CONCLUSION

In this research, we were able to achieve improved query performance, using the optimized copies created by database administrators, by extending the query optimizer to substitute it automatically at runtime. Further extension to this research could include evaluating the performance of the optimizer against cloud databases.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE

REFERENCES

[1] Murdoch, Travis B., and Allan S. Detsky. "The inevitable application of big data to health care." *Jama* 309.13 (2013): 1351-1352.

[2] Wang, Yichuan, LeeAnn Kung, and Terry Anthony Byrd. "Big data analytics: Understanding its capabilities and potential benefits for healthcare organizations." *Technological Forecasting and Social Change* 126 (2018): 3-13.

[3] Kruse, Clemens Scott, et al. "Challenges and opportunities of big data in health care: a systematic review." *JMIR medical informatics* 4.4 (2016): e38.

[4] Ong, T.C., Kahn, M.G., Kwan, B.M., Yamashita, T., Brandt, E., Hosokawa, P., Uhrich, C. and Schilling, L.M., 2017. Dynamic-ETL: a hybrid approach for health data extraction, transformation and loading. *BMC medical informatics and decision making*, *17*(1), p.134.

[5] Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., Meng, X., Rosen, J., Venkataraman, S., Franklin, M.J. and Ghodsi, A., 2016. Apache spark: a unified engine for big data processing. *Communications of the ACM*, *59*(11), pp.56-65.

[6] Duggan, J., Elmore, A.J., Stonebraker, M., Balazinska, M., Howe, B., Kepner, J., Madden, S., Maier, D., Mattson, T. and Zdonik, S., 2015. The bigdawg polystore system. *ACM Sigmod Record*, *44*(2), pp.11-16.

[7] Begoli, Edmon, et al. "Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources." *Proceedings of the 2018 International Conference on Management of Data*. ACM, 2018.

[8] Johnson, Alistair EW, Tom J. Pollard, Lu Shen, H. Lehman Li-wei, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G. Mark. "MIMIC-III, a freely accessible critical care database." *Scientific data* 3 (2016): 160035.

[9] https://www.oracle.com/database/12c-database/

[10] https://www.mysql.com/

[11] Liarou, E., Idreos, S., Manegold, S., & Kersten, M. (2012). MonetDB/DataCell: online analytics in a streaming column-store. *Proceedings of the VLDB Endowment*, *5*(12), 1910-1913.

978-1-7281-5020-8/19/$31.00 ©2019 IEEE