SQL QUERY OPTIMIZATION FOR HIGHLY NORMALIZED BIG DATA

Nikolay I. GOLOV
Lecturer, Department of Business Analytics, School of Business Informatics,
Faculty of Business and Management, National Research University Higher School of Economics
Address: 20, Myasnitskaya Street, Moscow, 101000, Russian Federation
E-mail: ngolov@hse.ru

Lars RONNBACK
Lecturer, Department of Computer Science, Stocholm University
Address: SE-106 91 Stockholm, Sweden
E-mail: lars.ronnback@anchormodeling.com

This paper describes an approach for fast ad-hoc analysis of Big Data inside a relational data model. The approach strives to achieve maximal utilization of highly normalized temporary tables through the merge join algorithm. It is designed for the Anchor modeling technique, which requires a very high level of table normalization. Anchor modeling is a novel data warehouse modeling technique, designed for classical databases and adapted by the authors of the article for Big Data environment and a massively parallel processing (MPP) database. Anchor modeling provides flexibility and high speed of data loading, where the presented approach adds support for fast ad-hoc analysis of Big Data sets (tens of terabytes).

Different approaches to query plan optimization are described and estimated, for row-based and column-based databases. Theoretical estimations and results of real data experiments carried out in a column-based MPP environment (HP Vertica) are presented and compared. The results show that the approach is particularly favorable when the available RAM resources are scarce, so that a switch is made from pure in-memory processing to spilling over from hard disk, while executing ad-hoc queries. Scaling is also investigated by running the same analysis on different numbers of nodes in the MPP cluster. Configurations of five, ten and twelve nodes were tested, using click stream data of Avito, the biggest classified site in Russia.

Key words: Big Data, massively parallel processing (MPP), database, normalization, analytics, ad-hoc, querying, modeling, performance.

Citation: Golov N.I., Ronnback L. (2015) SQL query optimization for highly normalized Big Data. *Business Informatics*, no. 3 (33), pp. 7–14.

Introduction

Big Data analysis is one of the most popular IT tasks today. Banks, telecommunication companies, and big web companies, such as Google, Facebook, and Twitter produce tremendous amounts of data. Moreover, nowadays business users know how to monetize such data [2]. Various artificial intelligence marketing techniques can transform big customer behavior data into millions and billions of dollars. However, implementations and platforms fast enough to execute various analytical queries over all available data remain the main issue. Until now, Hadoop has been considered the universal solution. But Hadoop has its drawbacks, especially in speed and in its ability to process difficult queries, such as analyzing and combining heterogeneous data [6].

This paper introduces a new data processing approach, which can be implemented inside a relational DBMS. The approach significantly increases the volume of data that can be analyzed within a given time frame. It has been implemented for fast ad-hoc query processing inside the column oriented DBMS Vertica [7]. With Vertica, this approach allows data scientists to perform fast ad-hoc queries, processing terabytes of raw data in minutes, dozens of times faster than this database can normally operate. Theoretically, it can increase the performance of ad-hoc queries inside other types of DBMS, too (experiments are planned within the framework of further research).

The approach is based on the new database modeling technique called Anchor modeling. Anchor modeling was first implemented to support a very high speed of loading new data into a data warehouse and to support fast changes in the logical model of the domain area, such as addition of new entities, new relationships between them, and new attributes of the entities. Later, it turned out to be extremely convenient for fast ad-hoc queries, processing high volumes of data, from hundreds of gigabytes up to tens of terabytes.

The paper is structured as follows. Since not everyone may be familiar with Anchor modeling, Section 1 explains its main principles, Section 2 discusses the aspects of data distribution in an massively parallel processing (MPP) environment, Section 3 defines the scope of analytical queries, Section 4 introduces the main principles of query optimization approach for analytical queries. Section 5 discusses the business impact this approach has on Avito, and the paper is concluded in the final section.

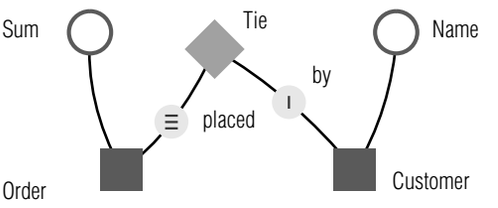
1. Anchor Modeling

Anchor modeling is a database modeling technique, based on the usage of the 6th normal form (6NF) [9]. Modeling in 6NF yields the maximal level of table decomposition, so that a table in 6NF has no non-trivial join dependencies. That is why tables in 6NF usually contain as few columns as possible. The following constructs are used in Anchor modeling:

1. *Anchor*, table of keys. Each logical entity of domain area must have a corresponding Anchor table. This table contains unique and immutable identifiers of objects of a given entity (surrogate keys). Customer and Order are two example anchors. An anchor table may also contain technical columns, such as metadata.

2. *Attribute*, table of attribute values. This table stores the values of a logical entities attributes that cannot be described as entities of their own. The Name of a Customer and the Sum of an Order are two example attributes. An attribute table must contain at least two columns, one for the entity key and one for the attribute value. If an entity has three attributes, three separate attribute tables have to be created.

3. *Tie*, table of entity relationships. For example, which Customer placed a certain Order is typical for a tie. Tie tables must contain at least two columns, one for the first entity key (that of the Customer) and one for the second (that of the related Order).



Anchor		Attribute		Attribute		Tie	
Ord ID	Cust ID	Ord ID	Sum	Cust ID	Name	Ord ID	Cust ID
12	4	12	150	4	Dell	12	4
26	8	26	210	8	HP	26	8
35		35	98			35	

Fig. 1. Anchor modeling sample data model

Ties and attributes can store historical values using principles of slow changing dimensions type 2 (SCD 2) [4]. Distinctive characteristic of Anchor modeling is that historicity is provided by a single date-time field (so-called FROM DATE), not a two fields (FROM DATE – TO DATE), as for Data Vault modeling methodology. If historicity is provided by a single field (FROM DATE), then second border of interval (TO DATE) can be estimated as FROM DATE of next sequential value, or NULL, if the next value is absent.

2. Data distribution

Massive parallel processing databases generally have shared-nothing scale-out architectures, so that each node holds some subset of the database and enjoy a high degree of autonomy with respect to executing parallelized parts of queries. For each workload posed to a cluster, the most efficient utilization occurs when the workload can be split equally among the nodes and

each node can perform as much of its assigned work as possible without the involvement of other nodes. In short, data transfer between nodes should be kept to a minimum. Given arbitrarily modeled data, achieving maximum efficiency for every query is unfeasible due to the amount of duplication that would be needed on the nodes, while also defeating the purpose of a cluster. However, with Anchor modeling, a balance is found where most data is distributed and the right amount of data is duplicated in order to achieve high performance for most queries. The following three distribution schemes are commonly present in MPP databases and they suitably coincide with Anchor modeling constructs.

Broadcast the data to all nodes. Broadcasted data is available locally, duplicated, on every node in the cluster. Knots are broadcasted, since they normally take up little space and may be joined from both attributes and ties. In order to assemble an instance with knotted attributes or relationships represented through knotted ties without involving other nodes, the knot values must be available locally. Thanks to the uniqueness constraint on knot values, query conditions involving them can be resolved very quickly and locally, to reduce the row count of intermediate result sets in the execution of the query.

Segment the data according to some operation that splits it across the nodes. For example, a modulo operation on a hash of the surrogate identifier column in the anchor and the attribute tables could be used to determine on which node data should end up. Using the same operation on anchors and attributes keeps an instance of an ensemble and its history of changes together on the same node. Assembling an instance in part or in its entirety can therefore be done without the involvement of other nodes. Since anchors and attributes retain the same sort order for the surrogate identifier, no sort operations are needed when they are joined.

Project the data with a specified sort order and specified segmentation. If table has multiple columns, if it is segmented by one column, and is joined by another column, than this join operation will require redistribution of data across all nodes. MPP databases support such joins with an ability to create *projection* – full copy of a table, segmented and sorted by another column. There can be multiple projections of a single table, according to business needs. Ties have one projection for each role in the tie, where each segmentation is done over and ordered by the surrogate identifier representing the corresponding role. This ensures that all relationships that an instance takes part in can be resolved without the involvement of other nodes, and that joining a tie will not require any sort operations. Therefore, tie will require no more than one additional projection.

Usage of single date-time field for historicity (SCD2) gives significant benefit – it does not require updates. New values can be added exclusively by inserts. This feature is very important for column-based databases, that are designed for fast select and insert operations, but are extremely slow for delete and update operations. Single date-time field historicity also requires efficient analytical queries to estimate closing date for each value. Distribution strategy, described above, guarantees, that all values for single anchor are stored on a single node and properly sorted, so closing date estimation can be extremely efficient.

3. Analytical queries

Databases can be utilized to perform two types of tasks: OLTP tasks or OLAP tasks. OLTP means online transaction processing: huge number of small insert, update, delete, select operations, where each operation processes small chunk of data. OLTP tasks are typical for operational databases. OLAP means online analytical processing: relatively small number of complex analytical select queries, where each query processes significant part of data (or whole data). Given article is focused on OLAP queries to Big Data.

Analytical queries may include three types of subtasks:

1. Combine dataset according to query. Big Data is stored as tables, which have to be combined (joined) according to given conditions.

2. Filter dataset according to query. Query can contain conditions on the data: single column conditions, column comparisons and sampling conditions (for example «top 10 percent of user accounts according to total number of payments of each user account»).

3. Calculate aggregates over filtered datasets.

4. Efficient plans for analytical queries

SQL can describe almost all types of analytical queries. It is one of the main advantages of SQL and relational databases: if analytical query can be formulated using SQL, than relational database can process it and return correct result. Main problem lies in the optimization of the query execution time.

In this paper, queries are analyzed from a Big Data analyst's point of view. Analysts want to get their answers as fast as possible. Query with suboptimal execution plan can be processed tens of times slower than optimal one. If query processes Big Data, inefficient plan can work hours and days, while efficient one can return result in minutes.

This section contains main principles of optimization analytical queries to Big Data in MPP environment, obtained by Avito. Growth of Avito data warehouse enabled its analysts to study various approaches to query optimization and chose most efficient ones, able to speed queries to Big Data in an MPP environment from hours to minutes. These researches were launched at the initial phase of DWH development, because there were concerns about speed of OLAP queries to a Anchor Model. On the further phases, those results were applied to analysis of both, normalized data (Anchor modeling) and denormalized data (data marts) with equal effectiveness.

Here is a list of main risks of query execution plan in a MPP environment:

✦ **Join spill.** According to Section 3, analytical tasks can require joins. Joins can be performed via Hash Join, Nested Loop Join and Sort-Merge join algorithms. Nested loop join is inefficient for OLAP queries (it's best for OLTP) [3]. Sort phase of sort-merge algorithm join can be inefficient for MPP environment, so if source date is not sorted, query optimizer prefers hash join algorithm. Hash join is based on storing left part of the join in RAM. If RAM is not enough, query will face a join spill. Join spill (term can differ in various DBMS-s) mean that some part of query execution plan requires too much RAM, and source data have to be separated into N chunks (small enough to fit into available RAM) to be processed consequently and at the end - to be combined together. Join spill reduces maximum RAM utilization, but increases disk I/O, a slower resource. Following diagram illustrates, why a condition reducing a table on one side of the join may not reduce the number of disk operations for the table on the other side of the join. Therefore, disk I/O operations can be estimated according to *optimistic* (concentrated keys) or *pessimistic* (spread out keys) scenarios. In the optimistic one, the number of operations is the same as in a RAM join, whereas in the pessimistic one the whole table may need to be scanned for each chunk, so disk I/O can increase N times. *According to modern servers(>100Gb of RAM for a node), this risk is actual when left part of the join contains over one billion of rows.* Hash join for tables of millions of rows is safe.

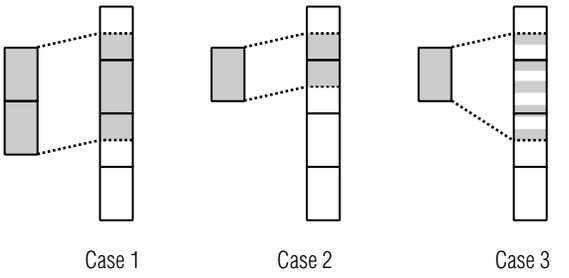


Fig. 2. Three cases of inner join

Figure above demonstrates three cases of inner join. First case: smaller table contain two disk pages of keys, correspondent keys in bigger table occupy three disk pages. Second case: smaller table reduced twice, to one disk page, correspondent keys in bigger table are concentrated, so to read them one need to scan only two disk pages instead of three. Third case: smaller table reduced twice, but correspondent keys are not concentrated but distributed (see gaps), so one need to read same number of disk pages, as before (three).

✦ Group by spill. Group by phase of the query can also be executed by Hash Group algorithm, and it also can be spilled on disk, as a join phase. *This risk is actual if there are more than hundreds of millions of groups in a group by phase.*

✦ Resegmentation/broadcast. If two tables are joined together, and they are joined and segmented over nodes of MPP cluster using different keys, than joining may be performed either by resegmentation of one table across all nodes, or by broadcasting all data of one table across all nodes (see Section 2). *This risk is actual if tables in question contains billions of rows.*

✦ **Wrong materialization**. Materialization strategy is extremely important for column-based databases [8]. Materialization strategy defines process of combining column data together (in a column based databases they are stored separately). It is extremely important for join queries. There can be early materialization (read columns, combine columns, than apply filter), or late materialization (read filtered column, apply filter, read other columns according to filtered rows, combine). According to [1], it is more efficient to select late materialization strategy. But if a query is complex, query optimizer can make a mistake. If query requires join of tables A, B and C, and has a filtering condition to tables A and B, late materialization may lead to loading of filtered values from A, B in parallel, all values from C, and join them together. This strategy is efficient if condition on table A and B reduce data a lot, while join with tables B and C don't reduce data. Otherwise, if each condition takes ¼ of A and B, and conditions are uncorrelated, than join-

ing filtered tables A and B will result in 1/16 part of data, so early materialization can significantly reduce disk I/O of tables B and C.

Risks, listed above, can be avoided by using modified query execution plans, based on following principles:

✧ **Maximum merge join utilization.** Sort-Merge join algorithm is a most effective for non-selective analytical queries among such algorithms as Hash Join, Nested Loop Join and Sort-Merge join, especially if analyzed data are already sorted on join key, so only merge join operation is required, without sort operations [3].

✧ **Single time disk page access.** Query plan with a join spill has awful characteristics in terms of pessimistic number of disk page access operations because many disk pages are accessed multiple times during plan execution. To create plan with optimal characteristics each required disk page must be accessed no more than once.

✧ **Massive temporary table utilization.** Two principles, mentioned above, cannot be guaranteed for all ad-hoc queries to real-world Big Data databases, created according to 1/2/3 normal form. Table of dozen columns cannot be sorted on each column (without duplicating this table dozen times). Also it cannot be guaranteed that all keys in joined tables are concentrated, to equalize pessimistic and optimistic disk page operation count (see *fig. 2*). But this principles can be guaranteed in a Anchor Modeling data model, if each data subset, retrieved after each step of filtering, is stored as a temporary table. Temporary tables can be sorted on specific column, to optimize further merge join. Also those temporary tables contain concentrated keys, only keys, required for given ad-hoc query. This feature helps to avoid double disk page access. Concerning speed of creating temporary table for large data set: modern HDDs provide equal speed for sequential read and sequential write. **Important.** Given algorithms don't require creation of indexes of any kind. Only storing sorted rows as a temporary table, so this operation can be as fast, as saving a data file on HDD. Sorting is enough for efficient Merge Join.

Efficient plans for analytical queries, utilizing principles, listed above, can be created using following algorithm:

0. Assume that query is based on joining tables $T_1,...T_N$, filtered according to complex conditions $C_1^1,...C_N^1,...C_1^N,...C_N^N$, there C_j^i is a condition on table j, which can be estimated only when i tables are already loaded. So, C_j^i is single table filtering condition. C_j^i is a combination of all conditions on table j.

1. Estimate data reduction rate for each table

$$\frac{R(C_j^1(T_j))}{R(T_j)},$$

where $R(T_j)$ means number of rows. Chose the tables with best reduction rate to be the first one (assume that it is table T_1).

2. Take k_1 tables, identically segmented with table T_1. Assume that those tables are $T_1,...T_{k_1}$. Join all k_1 tables, using all applicable filtering conditions, using merge join algorithm, consequently, starting from table T_1. So table T_1 has to be scanned and filtered first, than only corresponding rows from T_2 has to be scanned and filtered according to conditions C_2^1 and C_2^2, than the same for tables $T_3,...T_{k_1}$.

3. Estimate data reduction rate for remaining tables

$$\frac{R\left(C_j^{k1+1}\left(T_j\right)\right)}{R\left(T_j\right)},$$

using data tables, joined and filtered on step 2. Chose the tables with best reduction rate to be the next one (assume that it is table T_{k1+1}).

4. Save results of joining and filtering tables from step 2 inside a temporary table T_1^{new}, identically segmented with table T_{k1+1}.

5. Replace tables $T_1,...T_{k_1}$ of initial list with a table T_1^{new}.

6. Return to step 2 with reduced list of tables.

7. It list of tables contain single table, perform final aggregations, save results.

Given algorithm has important drawbacks:

1. It requires a lot of statistics about source tables, about data reduction rate of conditions, for step 1. In some cases first estimation of statistics can require more efforts, than analytical query itself.

2. Statistics estimation, as well as creation of temporary tables with dedicated segmentation, has to be implemented manually, because current version of query optimizers and query processing engines can not do it. Avito used Python implementation.

Drawback 1 can be avoided using estimates from previously calculated queries, continuously. First processing of brand new query can be time-consuming.

Drawback 2 can be avoided for identically segmented tables, because algorithm ends on step 2. That is why Anchor Modeling data model for MPP environment is especially favorable for approach. Step 2 does not require implementation of additional logic, it can be performed using SQL, database query execution engine and some hints.

5. Business applications

The approach described in Section 4 has been implemented for ad-hoc querying in a Big Data (data warehouse) solution at Avito. Based in Moscow, Avito is Russias fastest growing e-commerce site and portal, «Russia's Craiglist». The Avito data warehouse is implemented according to the Anchor modeling methodology. It contains ≈ 200 Anchors, ≈ 600 Attributes, and ≈ 300 Ties. Using this model it loads, maintains, and presents many types of data. The greatest volume can be found in the part loaded from the click stream data sources. Click stream data are loaded every 15 minutes, with each 15-minute batch containing 5 mln. rows at the beginning of 2014 and 15 mln. at the beginning of 2015.

Each loaded row contained approximately 30 columns at the beginning of 2014 and approximately 70 columns at the beginning of 2015. Each column is a source for a separate Anchor, Attribute or Tie table. The current size of the Avito data warehouse has been limited to 51Tb for licensing reasons. It contains years of consistent historical data from various sources (back office, Google DFP/AdSense, MDM, accounting software, various aggregates), and a rolling half year of detailed data from click stream.

The presented approach for ad-hoc queries was successfully implemented by the Avito BI team in less than a year. It provides analysts at Avito with a tool for fast (5-10-2060 minutes) analysis of near-real time Big Data, according to various types of tasks, such as direct marketing, pricing, CTR prediction, illicit content detection, and customer segmentation.

Conclusions

A high degree of normalization has the advantage of flexibility. The resulting data models can be extended easily and in a lossless manner. Anchor modeling also enables parallel loading of all entities, their attributes, and links with or without historization of each link and attribute. The supposed drawback of a highly normalized data model is slow ad-hoc reporting, which is why Inmon [4] recommends combining normalized models for centralized repositories with denormalized data marts. The paper, however, demonstrates that while Anchor modeling may require some sort of denormalization for single-entity analysis, it can yield very high performance for queries that involve multiple linked entities, and particularly so when there is a risk of join spill (lack of RAM).

The experiments carried out showed benefits of the given approach for the simplified case of two linked entities. Realworld business cases sometimes require three, four or even a dozen linked entities in the same ad-hoc query. Such cases multiply the risk of join spill occurring in some step, and amplify its negative effect. If number of joins, tables and filtering conditions increases, query RAM requirements estimation accuracy decreases. Query optimizer can significantly overestimate RAM requirements and cause an unnecessary join spill with all associated drawbacks. The approach from Section 4 has been in use at Avito for over a year, for ad-hoc and regular reporting, and even for some near-real time KPIs. It has demonstrated stability in terms of execution time and resource consumption, while ordinary queries often degraded after months of usage because of data growth.

One can wonder if the approach from Section 4 can be applied for implementations based on other techniques than Anchor modeling. The answer is yes, it can be applied, but with additional work. The strictly determined structure of Anchor modeling enables Avito to have a fully automated and metadata-driven implementation of the given approach. To conclude, the approach, while not being uniquely tied to Anchor modeling, suits such implementations very well and compensates for the commonly believed drawback of normalization in the perspective of ad-hoc querying. ■

References

1. Abadi D.J., Myers D.S., DeWitt D.J., Madden S.R. (2007) Materialization strategies in a Column-Oriented DBMS. Proceedings of the *23rd IEEE International Conference on Data Engineering (ICDE 2007), April 15-20, 2007, Istanbul, Turkey*, pp. 466–475.
2. Chen M., Mao S., Liu Y. (2014) Big Data: A survey. *Mobile Networks and Applications*, no. 19, pp. 171–209.
3. Graefe G. (1999) The value of merge-join and hash-join in SQL Server. Proceedings of the *25th International Conference on Very Large Data Bases (VLDB 1999), September 7-10, 1999, Edinburgh, Scotland*, pp. 250–253.
4. Inmon W.H. (1992) *Building the Data Warehouse*, Wellesley, MA: QED Information Sciences.
5. Kalavri V., Vlassov V. (2013) MapReduce: Limitations, optimizations and open issues. Proceedings of the *12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2013)*, July 16-18, 2013, Melbourne, Australia, pp. 1031–1038.

6. Knuth D. (1998) *The art of computer programming. Volume 3: Sorting and searching*. 2nd edition, Boston, MA: Addison–Wesley.
7. Lamb A., Fuller M., Varadarajan R., Tran N., Vandiver B., Doshi L., Bear C. (2012) The Vertica analytic database: C-store 7 years later. Proceedings of *the VLDB Endowment*, vol. 5, no. 12, pp. 1790–1801.
8. Shrinivas L., Bodagala S., Varadarajan R., Cary A., Bharathan V., Bear C. (2013) Materialization strategies in the Vertica analytic database: Lessons learned. Proceedings of the *29th IEEE International Conference on Data Engineering (ICDE 2013), April 8-11, 2013, Brisbane, Australia*, pp. 1196–1207.
9. Ronnback L., Regardt O., Bergholtz M., Johannesson P., Wohed P. (2010) Editorial: Anchor Modeling – agile information modeling in evolving data environments, *Data and Knowledge Engineering*, vol. 69, no. 12, pp. 1229–1253.

ОПТИМИЗАЦИЯ SQL-ЗАПРОСОВ ДЛЯ ВЫСОКОНОРМАЛИЗОВАННЫХ БОЛЬШИХ ДАННЫХ

Н.И. ГОЛОВ
преподаватель кафедры бизнес-аналитики,
школа бизнес-информатики, факультет бизнеса и менеджмента,
Национальный исследовательский университет «Высшая школа экономики»
Адрес: 101000, г. Москва, ул. Мясницкая, д. 20
E-mail: ngolov@hse.ru

Л. РОНБАК
преподаватель факультета компьютерных наук, Университет Стокгольма
Адрес: SE-106 91 Stockholm, Sweden
E-mail: lars.ronnback@anchormodeling.com

В данной статье описывается подход для быстрого анализа больших данных в реляционной модели данных. Целью данного подхода является достижение максимального использования высоконормализанных временных таблиц, объединяемых посредством алгоритма соединения слиянием (merge join algorithm). Подход был разработан для методологии Anchor Modeling, предполагающей крайне высокий уровень нормализации таблиц. Anchor Modeling – это новейшая методология построения хранилищ данных, разработанная для классических баз данных и адаптированная для задач больших данных и MPP (массивно-параллельных) баз данных авторами статьи. Anchor Modeling обеспечивает гибкость расширения и высокую скорость загрузки данных, в то время как представленный подход к оптимизации запросов дополняет методологию возможностью «на лету» проводить быстрый анализ больших выборок данных (десятки Тб).

В статье описаны и оценены различные подходы к оптимизации планов выполнения запросов для колоночных и обычных (строчных) баз данных. Представлены и сопоставлены результаты теоретических оценок и практических экспериментов на реальных данных, проведенных на платформе колоночной массивно-параллельной (MPP) базы данных HP Vertica. Результаты сравнения демонстрируют, что подход особенно эффективен для случаев нехватки доступной оперативной памяти, в результате чего оптимизатору запросов базы данных при обработке аналитических запросов приходится переходить от наиболее оптимального режима обработки в оперативной памяти (in-memory) к режиму подкачки с жесткого диска. Также изучен вопрос масштабирования нагрузки. Для этого один и тот же анализ производился на кластерах массивно-параллельной СУБД Вертика, состоящих из разного количества серверов. Были испытаны конфигурации из пяти, десяти и двенадцати серверов. Для анализа применялись данные типа «поток кликов» – обезличенные данные о кликах пользователей Авито, крупнейшего российского сайта объявлений.

Ключевые слова: большие данные, массивно-параллельная обработка (MPP), база данных, нормализация, аналитика, аналитика «на лету», запросы, моделирование, производительность.

Цитирование: Golov N.I., Ronnback L. SQL query optimization for highly normalized Big Data // Business Informatics. 2015. No. 3 (33). P. 7–14.

Литература

1. Abadi D.J., Myers D.S., DeWitt D.J., Madden S.R. Materialization strategies in a Column-Oriented DBMS // Proceedings of the 23rd IEEE International Conference on Data Engineering (ICDE 2007), April 15-20, 2007, Istanbul, Turkey. IEEE, 2007. P. 466–475.
2. Chen M., Mao S., Liu Y. Big Data: A survey // Mobile Networks and Applications. 2014. No. 19. P. 171–209.
3. Graefe G. The value of merge-join and hash-join in SQL Server // Proceedings of the 25th International Conference on Very Large Data Bases (VLDB 1999), September 7-10, 1999, Edinburgh, Scotland. 1999. P. 250–253.
4. Inmon W.H. Building the Data Warehouse. Wellesley, MA: QED Information Sciences, 1992. 272 p.
5. Kalavri V., Vlassov V. MapReduce: Limitations, optimizations and open issues // Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom 2013), July 16-18, 2013, Melbourne, Australia. IEEE, 2013. P. 1031–1038.
6. Knuth D. The art of computer programming. Volume 3: Sorting and searching. 2nd edition. Boston, MA: Addison–Wesley, 1998. 782 p.
7. The Vertica analytic database: C-store 7 years later / A. Lamb [et al] // Proceedings of the VLDB Endowment. 2012. Vol. 5, No. 12. P. 1790–1801.
8. Shrinivas L., Bodagala S., Varadarajan R., Cary A., Bharathan V., Bear C. Materialization strategies in the Vertica analytic database: Lessons learned // Proceedings of the 29th IEEE International Conference on Data Engineering (ICDE 2013), April 8-11, 2013, Brisbane, Australia. IEEE, 2013. P. 1196–1207.
9. Editorial: Anchor Modeling – agile information modeling in evolving data environments / L. Ronnback [et al] // Data and Knowledge Engineering. 2010. Vol. 69, No. 12. P. 1229–1253.