Global Journal of Computer Science and Technology: C
Software & Data Engineering
Volume 22 Issue 1 Version 1.0 Year 2022
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

Optimising Sargable Conjunctive Predicate Queries in the Context of Big Data

By Veronica V.N. Akwukwuma & Patrick O. Obilikwu
Benue State University

Abstract- With the continued increase in the volume of data, the volume dimension of big data has become a significant factor in estimating query time. When all other factors are held constant, query time increases as the volume of data increases and vice versa. To enhance query time, several techniques have come out of research efforts in this direction. One of such techniques is factorisation of query predicates. Factorisation has been used as a query optimization technique for the general class of predicates but has been found inapplicable to the subclass of sargable conjunctive equality predicates. Experiments performed exposed a peculiar nature of sargable conjunctive equality predicates based on which insight, the concatenated predicate model was formulated as capable of optimising sargable conjunctive equality predicates. Equations from research results were combined in a way that theorems describing the application and optimality of the concatenated predicate model were derived and proved.

Keywords: concatenated predicate, conjunctive equality predicate, sargable predicate, query, factorisation, database, software applications.

GJCST-C Classification: I.2.4

Strictly as per the compliance and regulations of:

Optimising Sargable Conjunctive Predicate Queries in the Context of Big Data

Veronica V.N. Akwukwuma [α] & Patrick O. Obilikwu [σ]

Abstract- With the continued increase in the volume of data, the volume dimension of big data has become a significant factor in estimating query time. When all other factors are held constant, query time increases as the volume of data increases and vice versa. To enhance query time, several techniques have come out of research efforts in this direction. One of such techniques is factorisation of query predicates. Factorisation has been used as a query optimization technique for the general class of predicates but has been found inapplicable to the subclass of sargable conjunctive equality predicates. Experiments performed exposed a peculiar nature of sargable conjunctive equality predicates based on which insight, the concatenated predicate model was formulated as capable of optimising sargable conjunctive equality predicates. Equations from research results were combined in a way that theorems describing the application and optimality of the concatenated predicate model were derived and proved. The theorems proved that the novel concatenated predicate model transforms a sargable conjunctive equality predicate such that the resultant concatenated predicate is an optimal equivalent of the sargable conjunctive equality predicate from which it is derived. The model enhances conjunctive sargable equality queries making our results capable of application in software applications, majority of whose queries are of the conjunctive query type. The results are equally useful in optimising query time within the context of Big Data where the continuous increase in the volume dimension of data calls for query structures that enhance query time.

Keywords: concatenated predicate, conjunctive equality predicate, sargable predicate, query, factorisation, database, software applications.

I. Background to Study

The fundamental Vs of Big Data are volume, velocity and variety [1]. Volume refers to the size of data being created, Velocity is the speed at which data is created, captured, extracted, processed, and stored while variety connotes different data types and sources ranging from structured, semi-structured to unstructured data. Of the three Vs, volume is most directly associated with big data and to put its importance in a perspective that emphasizes its relevance to query optimisation, volume may be redefined as voluminosity, vacuum, and vitality – three additional V-dimensions of data as exposed by [2]. Voluminosity states that there is already a very large set of data collected and even much more is available that can be harvested. Voluminosity speaks of a significant gap that can be filled by data yet to be collected. From the perspective of voluminosity, volume refers to the size of data being created from all sources in an organization including text, audio, video, social networks, research studies, medical data, space images, crime reports, weather forecasting and natural disaster [3].

The vacuum dimension of volume states that there is a strong requirement for storage to store large volumes of data. Due to the fact that the data is acquired incrementally, empty spaces will always be needed for use in the creation of room to store, process and manage tremendous data set as they are harvested from different sources. This dimension of volume pops up the research question about how much storage space is available for incoming data rather than how much data has already been stored. The process of creating storage space for incoming data is equally as challenging as it is with managing vast sets of already stored data. Empty spaces that serve this purpose are created by either augmenting storage devices or techniques used to compress the size of data [4].

Vitality may be defined as the survival of data in the storage environment and thus its reliability and usefulness. Data in the storage environment falls into the two categories, namely active served and unserved. In a large data bank, some data are actively used while some are not [4]. Vitality redefines volume as meaning that data and its subsets are used actively at different times. While a portion of data may be actively used data at a time or within a specific transaction, the rest are stored for future uses. There is the risk that data stored for future may take so long for it to be used which may lead to such sub-datasets to be abandoned or not properly maintained. As the risk of being abandoned gets higher, anything can happen to those datasets not currently in use. In other words, with less investment and attention to the unserved data, they are exposed to incidences of fire, earthquake, flood, war, and terrorist which are the prominent causes of data loss. Thus, vitality is a critical component of volume. The lack of vitality, in any case, is symptomatic of the absence of disaster management systems which decimates data reliability or can lead to complete data loss. Apart from reliability, vitality also describes flexibility, dependability,

Author α: Ph.D, Department of Computer Science, University of Benin, Benin City, Nigeria.
Author σ: Ph.D, Department of Computer Science, Benue State University, Makurdi, Nigeria. e-mail: poblikwu@gmail.com

and security which are all integral components of volume,

As data gets larger in the dimensions of big data, partitioning strategies have been used to reduce the data to smaller subsets over which queries become faster compared to the original dataset [5]. Popular among these partitioning strategies is the horizontal scaling (scaling out), Horizontal scaling refers to resource increment by the addition of complete and independent units that work in unison with an existing system. The additional units may be of smaller capacity, making it cheaper compared to the replacement of an existing single unit with one of larger capacity. The scale out effect of the horizontal partitioning strategy creates a hardware infrastructure platform on which partitioned data is then distributed across multiple units or servers, hence, reducing the excess load of the entire data set on a single machine [6,7]. This platform comes with the added advantage of keeping the entire system up even if some of the units go down, thus, avoiding the "single point of failure" problem associated with vertical scaling. The vertical scaling (scaling up) strategy refers to increasing the ability of a single hardware unit such as a server to handle the ever-increasing workload as a way of achieving resource increment. From the perspective of hardware, this includes adding memory and processing power to the single unit.

The horizontal scaling strategy is at the heart of the implementation of big data stores namely p-stores, c-stores and NoSql among others that have pioneered the paradigm shift of "No One Size Fits-All" proposed by Stonebraker and Çetintemel [8]. The horizontal scaling strategy partitions data such that queries can be fired selectively on the partitions with the aim of retrieving the desired data in optimal query time. As is applicable to all datasets, the desired data in a partition is indicated in a query using a boolean expression of conditions called predicates. Predicates are used in joins as well in search arguments of queries. A join predicate is a predicate that relates columns of two tables to be joined and the columns referenced in a join predicate are called join columns. When used in Search ARGuments (SARGs), predicates are referred to as sargable predicates [9]. A sargable predicate is one of the form (or which can be put into the form) "column comparison-operator value". Matalqa and Mustafa [5] experimentally demonstrated that restructuring big data into partitions produces query enhancement results. Using the theorem and axiom, Obilikwu, Kwaghtyo and Ogbuju [10] theoretically proved the result of [5] as follows:

Theorem: Given P_1, P_2 … P_n as the partitions of a relation R, then R = {P_1, P_2, …, P_n} where n = the number of distinct values in the value set associated with the partition key that generated P_1, P_2 … P_n

Axiom: The following axioms are applicable:

1. A partition key has a value set, V whose element cannot be null
2. The number of distinct values of V is n= number of partitions produced

Proof: Let $\boldsymbol{\sigma}$ be the partition predicate associated with a distinct value of V, then Arity($\boldsymbol{\sigma}$) is the arity of the tuples filtered by $\boldsymbol{\sigma}$.

Given any value of n, there exists σ_1, σ_2, ..., σ_n, where

σ_1 filters all tuples in P_1 from relation R,

σ_2 filters all tuples in P_2 from relation R, and

σ_n filters all tuples in P_n from relation R,

Since the elements of V cannot be null, then Arity(V) = Arity (R)

Since σ_1, σ_2, ..., σ_n filter the tuples of R according to the distinct values of V, it follows that

$\text{Arity(V)}=\text{Arity}(\sigma_1) + \text{Arity}(\sigma_2) + ... + \text{Arity}(\sigma_n)=\sum_i^n Arity(\sigma_i)$

This implies that $\sum_i^n Arity(\sigma_i)$= Arity (R) since n is the number of distinct values of V defined in R

This shows that R = {P_1, P_2, …, P_n} since σ_1, σ_2, ..., σ_n filter the tuples of R. QED.

The use of partitioning strategies makes queries faster [5]. This is because retrieving a record or a set of records from a relation is done relative to the number of the total number of records in the relation (R). Based on this relationship, query time can be computed as a ratio using equation 1.

$$q_t = \frac{t_R}{T_R} \quad \ldots \qquad (1)$$

where q_t is query time, t_R is the number of tuples retrieved from a relation R using a predicate σ and T_R is number of tuples in R. Equation 1 assumes an asymptotic value of t_R as well as the fact that other factors that affect query time are held constant. Among others, these other factors are processor speed, RAM and ROM size, communication traffic and code efficiency.

The implication of equation 1 is that an increase in volume implies an increase in query time. The query works with the DBMS as part of the algorithms that ensure data is retrieved seamlessly. While the DBMS suggests how the data can be located and retrieved, the query syntax tells what data is to be retrieved. These make up the two components of a database management system as depicted in Figure 1.

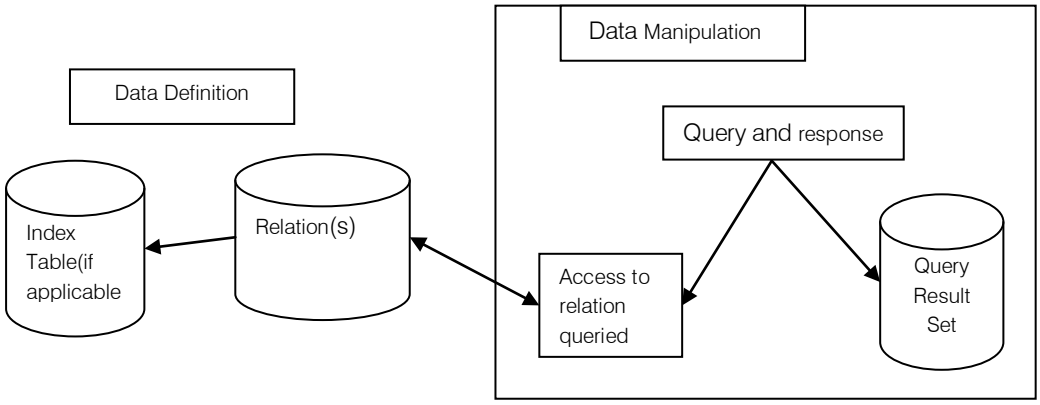

Figure 2: Architecture of Database Management

This paper is motivated by the critical need to optimise queries in the context of big data, big data being a development that has led to the ubiquitous incidence of big databases. The objectives of the paper are therefore as follows: (i) show that query time increases as the arity of database storage structures increase; (ii) show that optimising query time can be approached by organizing the storage structure using techniques like indexing and storage partitioning. It is also shown that queries can be modified or transformed to an equivalent form such that query time is reduced; (iii) use a combination of mathematical techniques to develop the concatenated predicate model thus enhancing the query time of sargable conjunctive equality predicates (iv) prove using mathematical induction and other applicable techniques that the concatenated predicate model optimises the sargable conjunctive equality predicate.

The rest of this paper is organized as follows: Section 2 reviews literature on the general concept of query optimisation and subsequently narrows the discussion down to the specific class of conjunctive predicates and how optimisation of predicates enhances query time. In Section 3, the product function is presented as a mathematical model to describe the product of atomic predicates, an operation also referred to as concatenation. Concatenation achieves literal minimisation as an alternative to factorization where there are no common atomic predicates. Concatenation in this paper to propose the concatenated predicate model. In Section 4, the results of this study are demonstrated using mathematical induction and other proofs. The proofs are discussed relative to the expected behavior of the concatenated predicate model.Finally, Section 5 concludes the paper and makes suggestions for future work.

II. Related Work

A predicate is that part of the query that filters records based on certain conditions. The properties of a predicate are multifarious and their study has exposed opportunities for optimising them, given that optimising them ultimately optimises database query time. Techniques for optimising queries are dependent on the query type.

a) Conjunctive Queries

Conjunctive queries represent one of the query languages used to retrieve data from relational databases [11,12,13] among other database models. Conjunctive queries correspond to the non-recursive Datalog rules [14]. In recursive datalog rules, conjunctive queries are of the form,

$$R_0(u_0) \leftarrow Ri_1(u_1) \wedge Ri_2(u_2) \wedge \ldots Ri_m(u_m)$$

where Ri_j is the relation name of the underlying database. R_0 is the output relation, and where each argument u_j is a list of $|u_j|$ variables, where $|u_j|$ is the arity of the corresponding relation Ri_j.

Conjunctive queries consist strictly of conjunctive predicates and they are the most widely used database queries in practice. It is against the background that optimising them makes a whole lot of sense [15,16,17,18,19]. The wide use of conjunctive queries are observable in not only their ubiquitous use in decision support systems based on relational databases but in other areas such as Description Language queries used to query knowledge representation (KR) systems , ontology-based queries and query answering frameworks in general [20,21]. Optimising a conjunctive query simply means optimising the conjunctive predicate component.

Heimel et al. [22] defined conjunctive predicates mathematically as

$$\theta = \Lambda_{i=1}^{m} \theta_i$$

where θ_i are atomic predicates joined by the AND relational operators and $i = 1,2, .., m$ are predicate terms (predicate literals, Boolean variables or atomic predicates) making up the conjunctive predicate. Sargable conjunctive predicates were defined by Yu X et al.[23] as conjunctive predicates of the form,

$$Q = P_1 \wedge P_2 \wedge \ldots \wedge P_m$$

where each component P_i, $i>0$ is an atomic predicate of the attribute value pair (*attribute* op *value*) with *op* being one of the comparison operators $<, \leq, =, \neq, \geq$ *or* $>$.

Practically speaking, conjunctive predicates are identified in the filter component of the project-select-join queries in the relational algebra, and in the where-clause of SQL queries having the general form SELECT . . . FROM . . . WHERE . . . where the where-clause is a predicate clause. Predicates are the conditions based on which database queries filter tuples in a relation or group of related relations. In its basic form, a query predicate is an atomic conditional expression also referred to as an atomic predicate. Several atomic predicates can be combined using logical operators to make up complex predicates [24] and the number of atomic predicates in a complex predicate is the boolean factor [9]. An atomic predicate has a boolean factor of 1. Boolean factors are notable because every tuple returned by a query must satisfy every boolean factor. A complex predicate made up of atomic predicates joined strictly using the AND logical operator is referred to as a conjunctive predicate. If all the atomic predicates in a complex predicate consist strictly of the equality operator, the complex predicate is referred to as a conjunctive equality predicate. Assuming the logical operator in the complex predicate is the OR logical operator then the resulting predicate will be a disjunctive predicate [25]. If the relational operator is the equality operator, then the complex predicate is a conjunctive equality predicate. If the conjunctive equality predicate is sargable, then it referred to as a sargable conjunctive equality predicate This paper is a study on how predicates of the class of sargable conjunctive equality predicates can be optimised.

b) Query Optimisation

Big data is resource-intensive and hence requires that both storage and query time are optimised for effective resource utilization. Resource optimisation, be it hardware or otherwise has been discussed within the larger context of solutions that we can never have enough of [26]. As a matter of fact the optimisation problem domain is one we are not yet done with [27]. Optimizing a number of running processes is considered an optimisation strategy though via software. Optimising query time by software (algorithms) is traditionally a function of the query optimizer, which is internal to the DBMS [9]. The algorithms associated with the query optimiser manipulate a query plan in its internal structure to choose an optimal plan for implementing a query. Query optimization gained research attention when the advantages of the relational data model in terms of user productivity and data independence became widely recognized in response to Codd's original ideas about the concept of relational databases [28]. Following this development, researchers began to ask questions about whether or not an automatic system can choose as efficient an algorithm for processing a complex query as a trained programmer would. System R, an experimental system was then constructed at the San Jose IBM Research Laboratory to demonstrate that a relational database system can incorporate the high performance and complete function, including automatic query optimisation required for everyday production use [9,29].

Query optimization has also been associated with modifying the structure of relations. In this regard, indexing can be said to be a pioneering effort at optimising query time from the dimension of database structure [30,31]. In processing a query that has a predicate, the attributes in the predicate are examined to find out if an index has been defined for any of the attributes, a concept referred to as index availability. The availability of an index makes searching relations faster compared to a full scan which is the search option used in the absence of an index. On the other hand, an index scan is used for the search if an index is available. The implementation of a full scan uses sequential search while an index scan is implemented using binary search. It is established in algorithmic theory that sequential search is of $O(n)$ and binary search is $O(\log n)$ making it obvious that an index scan is faster thereby enhancing query time.

Queries are also faster when relations are normalized. Partitioning relations also achieve good results. Optimising query operators, especially SELECTION and JOIN operators equally enhance query time. Incidentally, research into the optimisation of query operators has focused on joins and their ordering to the near neglect of research into the optimization of selection predicates [24]. Query optimisation is an open ended research question and hence it has been the object of research efforts over the years [26,9,15,32,33,34,35,36].

c) Predicate Optimisation

Query optimisation research efforts over the years in the specific area of predicate optimization have resulted in several optimization techniques notable among which are Predicate Pushdown [37], LDL approach [38,39], Predicate Move-around [12], Predicate Migration [40], By-Pass Predicate Processing [25], Optimising User-defined functions using Pruning Strategies [41,42]. Prominent among this technique is factorisation, a technique used to mininise the number of atomic predicates or terms in a complex predicate. Kemper et al. [43] and Chaudhuri et al.[24] used factorization to minimise atomic predicates in queries. The objective of factorization is to represent a Boolean function in a logically equivalent factored form having a

minimum number of literals [44]. The concept of minimizing atomic predicates (predicate literals) in a Boolean expression means that such expressions can be made simpler by reducing the terms in them. Predicate literals are found in the design of VSLI [45], compilers [46], and database query predicates [43,24] and minimization techniques of various types have been applied in each of these application areas thereby optimising the expressions involved. Factorisation is however only possible where the predicate expression has common atomic predicates.

Muralikrishna and DeWitt [47] established that the number of times a relation in a query is scanned is equal to the number of terms in which attributes of the relation are involved. This means that minimising the number of terms equally minimises the number of scans for each relation. Scanning constitutes a fundamental operation in query processing and thus a reduction in the number of scans done by a query equally reduces query time. Chaudhuri et al. [24] showed that factorization can be used to minimize predicate terms in scenarios where there are common atomic predicate factors. In this work, the sargable conjunctive equality predicates have been exposed as incidences of predicates where there are no common atomic predicate factors implying that factorization is inapplicable as a predicate minimisation technique. Sargable conjunctive equality predicates do not have common Boolean factors because an atomic predicate appearing more than once in a sargable conjunctive equality predicate duplicates such an atomic predicate. The duplicate atomic predicate is redundant and the result of such is unsatisfiable and evaluating them would lead to incorrect results [22]. This motivates the study of the nature of optimisation problems inherent in sargable conjunctive equality predicates. The experiments performed exposed interesting insights as to why existing predicate optimistion techniques, particularly factorization are inapplicable.

d) Nature of Optimisation Problem Posed by Sargable conjunctive equality predicates

To optimise sargable conjunctive equality predicates, there is need to understand the nature of optimisation problem posed by them. Series of experiments were conducted using a simulated data of students scores in an examination to expose what happens in terms of query time when the number of atomic predicates in a sargable conjunctive equality predicate is varied in a query. The experiments performed assumed that a number of students took an examination in the Department of Physics of a hypothetical University. The examination results are captured in a database relation, named studentscores. A schema is defined for the relation as studentscores(sno, studentID, level, courseCode, semesterID, sessionID, status, score) where the attributes are described as follows: sno (serial number); studentID (unique identifier for student); courseCode (semester course code); semesterID (identifier for semester); sessionID (identifier for session); status (semester course status) and score (an attribute for students score in the examination). Five instances of this schema are shown in Table 1.

Table 1: Instances of Examination Results Schema (Query Table)

Sno.	StudentID	Level	CourseCode	SemesterID	SessionID	Status	Score
1	SCN890178254	400	PHY412	1ST	2016/2017	C	94
2	SCN907524101	400	PHY412	1ST	2016/2017	C	65
3	SCN901782548	400	PHY412	1ST	2016/2017	C	76
4	SCN898888254	400	PHY412	1ST	2016/2017	C	35
5	SCN895428266	400	PHY412	1ST	2016/2017	C	58

The instances of the relational schema generated in Table 1 are five but the assumption is that as many students as there wrote the examination in the physics course (PHY412). The level is 400 (a course taken at the fourth year of study except when taken as a carry over). SessionID and semesterID are 2016/2017 and 1ST respectively. The semester course is a core course hence it has the code "C" for the status. A core course in this context is a course that is compulsory for all the students doing the same course of study or programme. Elective courses on the other hand are not compulsory. They are offered by students as a matter of choice.

The experiments conducted involved sargable conjunctive equality predicates and it involved varying the number of atomic predicates from two to five. Five atomic predicates are realistic enough to test the behavior of a complex predicate [41]. For each sargable conjunctive equality predicate, the number of schema instances was varied from 600,000 to 1,000,000. A data set of 1,000,000 records was assumed to be asymptotic (big data) and sufficient based on the use of the same number of records in a similar database experiment [41]. For a sargable conjunctive equality predicate to select an instance, the atomic conditions in the conjunct must all be true for the instance. For this reason, it is common in experiments testing conjunctive equality predicates to have record instances with repeated values [48]. The predicate attributes in the experimental data are grouped in terms of the number of predicates in the sargable conjunctive equality predicate and presented in Table 2.

Table 2: Predicates attributes used in the Sargable conjunctive equality predicates

Number of predicate attributes	Predicate attributes	Sargable conjunctive equality predicates
2	courseCode, semesterID	courseCode="PHY412" and semesterID='1st'
3	level, coursecode, semesterID	Level ="400" and courseCode="PHY412" and semesterID='1st'
4	courseCode,semesterID, sessionID, status	courseCode="PHY" and semesterID='1st' and sessionID = "2015/2016" and status = "C"
5	level,courseCode, semesterID, sessionID, status	Level ="400" and courseCode="PHY" and semesterID='1st' and sessionID = "2015/2016" and status = "C"

The predicate attributes listed in Table 2 are classified according to the number of their atomic predicate attributes beginning from 2 to 5 with their corresponding sargable conjunctive equality predicates. The combination of the predicate attributes of each sargable conjunctive equality predicate in any order or pattern is commutative and equivalent. They retrieve the same number of records and hence the order does not matter. The query times obtained from the experiment performed are shown in Table 3.

Table 3: Query times for sargable conjunctive equality predicates

Number of records (n)	Query time in microseconds according to number of predicates			
	2 predicates	3 predicates	4 predicates	5 predicates
600,000	7.653684139	8.102646112	8.527706862	8.91692996
800,000	9.84117198	10.45632792	10.88422108	11.36055684
1,000,000	12.14951396	12.91807389	13.49754906	14.54573202

The data obtained from the experiment exposed a pattern whereby the query times associated with sargable conjunctive equality predicates increase as the number of atomic predicates are varied from two through five which implies that when the atomic predicates are reduced, the query time is equally reduced. It is obvious from this observation that minimizing the number of atomic predicates of the sargable conjunctive equality predicate enhances query time. Figure 2 shows the query times of the sargable conjunctive equality predicates as a graph.

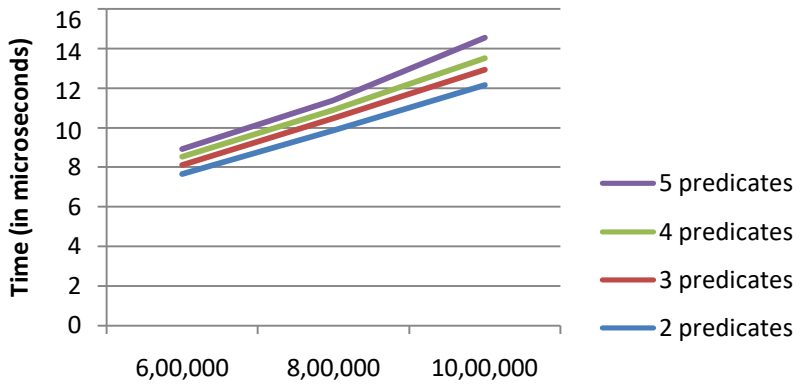

Figure 2: Query times of sargable conjunctive equality predicates

In the DBMS architecture shown in Figure 1, a query is a component of the DBMS that works in conjunction with the query optimiser to ensure queries run optimally. The query times obtained in Figure 2 includes every internal optimisation done by optimiser as well as any restructuring that can be done to the database such as indexing, partitioning, normalization and the introduction of primary keys. This scenario was earlier modeled in equation 1 as:

$$q_t = \frac{T_6}{T_R}$$

The point in this paper is that the query can be optimised even before it is submitted to the optimiser. A

typical case in point is the use of subqueries (nested queries also referred to as queries in predicates) in place of joins in scenarios where subqueries and joins are equivalent queries. An example is where the join retrieves a single tuple, then it is less costly to use subqueries than joins. The equivalent query that optimises query time introduces an optimisation factor, q_{opt} to equation 1 to produce equation 2.

$$q_t = \frac{T_6}{T_R} . q_{opt} \quad (2)$$

Where q_{opt} lies in the range $0 < q_{opt} < 1$. $q_{opt} = 1$ means there was no optimisation by the optimisation technique applied. The concept of optimizing sargable conjunctive predicates is rooted in theory of equivalent queries. For any sargable conjunctive predicates, there exists a corresponding sargable concatenated predicate.

On the basis of this insight from the experimental results in Figure 2, the concatenated Predicate model is formulated as consisting of a concatenated predicate and a corresponding surrogate index that is exploited by the concatenated predicate to enhance the query time of an equivalent sargable conjunctive equality predicate. The experiments are restricted to single table access and by implication, sargable conjunctive equality predicates [17]. Based on the experimental results, the methodology of this study consists of equations describing the product of terms (atomic predicates) which were subsequently used to formulate the concatenated predicate model. Theorems describing the application and optimality of the model as capable of optimising sargable conjunctive equality predicates are derived and also proved. It is hoped that the clarity of the concepts using the single table access will help in extrapolating the model to the other types of table access.

III. Methodology

The basic materials for this research were published literatures. Chaudhuri et al. [24] used the factorization technique to optimize a class of predicates that have common Boolean factors. The class of sargable conjunctive equality predicates on the other hand do not have common Boolean factors which makes factorization inapplicable to them. Motivated by this insight, this study unravelled some properties of the sargable conjunctive equality predicate which gave an insight on how this class of predicates can be optimised.

a) Mathematical Model

In describing the proposed model, mathematical models have been used extensively. Muralikrishna and DeWitt [47] referred to the product of the atomic predicates, P_i, i =1,2, .., m in a join or selection clause as,

$$\prod_{i=1}^{m} P_i, \quad m > 0$$

Each of the atomic predicates is referred to as a term. Assuming each atomic predicate, P_i to be of the form, $a_i = v_i$ and a_i denotes an attribute name of relation R and v_i is a value, then the predicate defined is an equality predicate. The product of terms operation is also referred to as the concatenation of the terms [49]. Since P_i in $\prod_{i=1}^{m} P_i$ is of the form, $a_i = v_i$, then $\prod_{i=1}^{m} P_i$ can be decomposed to become,

$$\prod_{i=1}^{m} a_i = \prod_{i=1}^{m} v_i$$

$\prod_{i=1}^{m} a_i$ is the product of attribute names which for ease of reference can be assigned a variable name, say C to get,

$$C = \prod_{i=1}^{m} v_i \quad \ldots \quad (3)$$

where $\prod_{i=1}^{m} v_i = v_1 . v_2, \ldots, v_m = v_1 v_2 \ldots v_m$ and equation (3), defining an atomic predicate can be referred to as the concatenated predicate.

b) The Concatenated Predicate Model

Concatenation amounts to finding the product of terms, the result of which is a single term. Given the equality predicates of a sargable conjunctive equality predicate as terms, concatenation can be used to find the product of the equality predicates which results in a single atomic predicate. Put differently, concatenation reduces (minimises) the number of terms (atomic predicates) in a sargable conjunctive equality predicate to one irrespective of the number of terms [49,11]. Concatenation in mathematics is the joining of two numbers by their numerals in contrast to arithmetic operations on numbers. Arithmetic operations such as addition, multiplication and all the others are based not only on the numerals but also on the magnitude of the numerals involved. Generalising, concatenation is an operation on the literals of an expression. If the term is a number, the literals are the numerals; the literals are alphabets or alphanumeric if the term is alphabetic or alphanumeric respectively.

Deen [50] exposed concatenation to be a very useful operation in computer programming and used it to generate surrogate keys as the product of an internal relation number (*irn*) and an effective key value (*ekey* value). The surrogate key generated is given by *surrogate* ::= <*irn*> <*ekey* value>. In Oracle noSQL, the concatenation of a *Major Key Path* and a *Minor Key Path* was used to generate record keys [51]. All records sharing a Major Key Path are co-located to achieve data locality. Within a co-located collection of Major Key Paths, the full key, comprising of both the Major and

Minor Key Paths, provides fast indexed lookups. Concatenation has also been applied in the theory of languages [52].

To use concatenation as a product of atomic predicates in a sargable predicate, the following conditions must be met:

1. The values of the atomic predicate attributes of the predicate must be exact and this can only be guaranteed by the equality relational operator
2. The predicate terms must not be less than two and each of them must be an atomic predicate in the predicate to be concatenated. This condition can only be guaranteed when the AND logical operator is used to join the atomic predicates

The second condition is a necessary condition because different values defined for the same atomic predicate attribute in a conjunctive equality predicate is unsatisfiable and evaluating them would lead to incorrect results [22]. In practical terms we cannot have A=12 and A=10 as a valid atomic predicates in a conjunctive equality predicate. The predicate attribute, A in a conjunctive equality predicate cannot have different values at the same time. Sargable conjunctive equality predicates meet the two conditions specified above hence concatenation is applicable to them as an optimisation technique. For every sargable conjunctive equality predicate, an equivalent concatenated predicate is derivable by concatenating the atomic predicates of the sargable conjunctive equality predicate.

The transformation of the sargable conjunctive equality predicate to the concatenated predicate can be shown diagrammatically using a logical plan tree, the height of which depends on the number of atomic predicate operations involved in the predicate. Considering a sargable conjunctive equality predicate having three atomic predicates, σ_1, σ_2 and σ_3 defined on relation, *R* for example, the plan tree will have three predicate operations as shown in Figure 3a. The equivalent concatenated predicate, say *C* is a product of the atomic predicates and hence it has a single predicate operation, σ defined on relation *R* as shown in Figure 3b.

Sargable conjunctive equality predicate Concatenated predicate

Figure 3: Equivalent Predicate Logical Plan Trees

In general, each atomic predicate in a sargable conjunctive equality predicate corresponds to a predicate operator (σ), on the logical plan tree. Each additional operator increases the height of 3a by 1 meanwhile the height of 3b remains constant. It is obvious from the logical plan trees that irrespective of the number of atomic predicates, the concatenated predicate has one atomic predicate and is assumed to be the transformation of an equivalent sargable conjunctive equality predicate.

The concatenated predicate is derived from the atomic predicate attributes of the sargable conjunctive equality predicate meaning that the atomic predicate attributes must be natural attributes of the relation queried by the sargable conjunctive equality predicate. The atomic predicate attributes are said to be sargable because they are used to search the relation. In a similar fashion, the concatenated predicate, being a product has a single attribute which it equally uses to search the relation. This also means that the concatenated predicate is also a sargable predicate. Sargable predicates search relations based on the value set of the attribute involved in the predicate. Incidentally, the attribute involved in the concatenated predicate is not a natural attribute in the relation and it has to be constructed as an artificial or surrogate attribute, call it *S*. The value sets of *S* are arrived at by concatenating the value sets of each of the natural attributes in the sargable conjunctive equality predicate as follows:

$$v(S) = \prod_{i=1}^{m} v_i{}^{pa} = v_1^{pa}.v_2^{pa},\ldots,v_m^{pa} = v_1^{pa}v_2^{pa}\ldots v_m^{pa} \quad \ldots \quad (4)$$

where *pa* is a predicate attribute of a sargable conjunctive equality predicate and v_i, i>0 is the value set with the natural fields involved in the sargable conjunctive equality predicate. It follows from this definition that, *S* is the artificial attribute whose value sets is the concatenation of all v_i for each value set of *m* atomic predicates in the sargable conjunctive equality predicate. This makes *S* one of the attributes defined for the query relation, *t* relative to which *t*(*S*) can be defines at the tuples of *S* in relation, *t* shown in equation (5).

$$t(S) = \prod_{i=1}^{m} v_i{}^{pa} \quad \ldots \quad (5)$$

making q_{opt} <1 in $q_t = \frac{T_6}{T_R}.q_{opt}$

The artificial attribute and its tuples referred to in equation 5 is a surrogate attribute and works very much like a user-defined index or surrogate value [30,53]. Surrogate indexes are very useful in database query optimisation [49,54,55]. The original concept of a surrogate value was to provide a unique identifier for each tuple (a kind of system primary key) that does not change irrespective of what the user chooses to do with the primary key value or the value of any of the other fields in terms of modifying them. These were called permanent surrogates. Deen [50] implemented the inpure type of surrogates in which the surrogate key changes if any of the values concatenated to generate the surrogate changes. The inpure surrogates were generated from the primary key using a hashing and a key compression algorithm, supported by an overflow mechanism. To effectively achieve this, surrogates are maintained using the following operations. When a tuple is inserted, a surrogate must be generated and the surrogate directory updated. This operation is referred to as surrogate generation. When a tuple is deleted, the surrogate directory must be updated, releasing the surrogate for possible re-use. This operation is referred to as surrogate release. Given the value of the attributes that make up the surrogate key value, the system should be able to find the surrogate. This operation is referred to as surrogate access. Given a surrogate, it should be possible to find the stored tuple. This operation is referred to as storage access and in this role, the surrogate serves the purpose of a data structure that can be exploited by predicates to locate records.

Diagrammatically, when the surrogate index is exploited by a concatenated predicate, tuples of the associated relation that match the predicate condition are fetched. The tuples defined in Equation (5) that are fetched by the concatenated predicate are depicted in Figure 4.

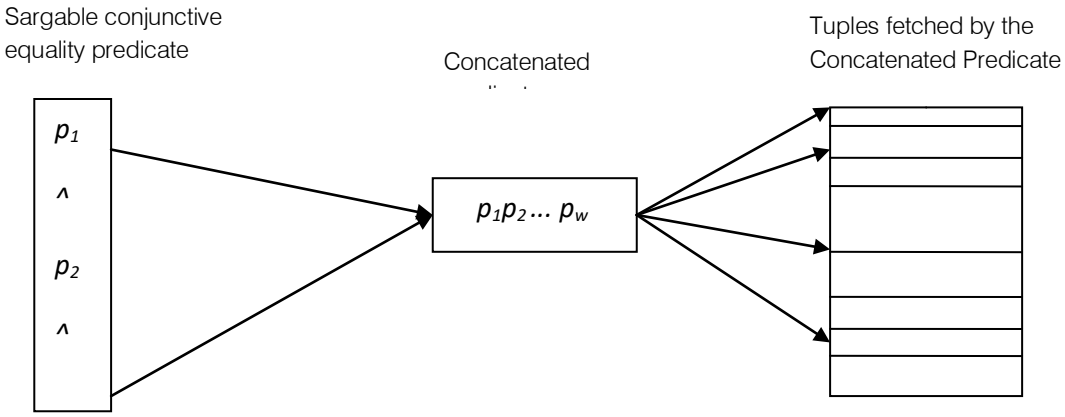

Figure 4: Tuples Returned by the Concatenated Predicate

Figure 4 is a diagrammatic representation of how the concatenated predicate is evaluated. Let θ be the concatenated predicate on the query table, *t,* then equation (6) models the evaluation of θ. A tuple, is returned from the query table by θ when the concatenated predicate is evaluated and the result is true.

$$\theta_i(t) = \begin{cases} true, & if\ t(S_i) = C \\ false, & otherwise \end{cases} \quad \ldots \quad (6)$$

IV. Results and Discussion

Resulting from the experiments performed in order to gain an understanding of the nature of optimisation problem posed by sargable conjunctive equality predicates, the Concatenated Predicate model was formulated as consisting of a concatenated predicate and a corresponding surrogate index that is exploited by the concatenated predicate to enhance the query time of an equivalent sargable conjunctive equality predicate. This result is proved using formal

methods for their correctness. The correctness of this result is discussed as a theoretical proof of the concatenated predicate model.

a) Proof of Existence of the Concatenated Predicate

Lemma 1: The equality condition in a sargable conjunctive equality predicate guarantees uniqueness of its atomic predicates because a conjunct of two different filters on the same attribute is unsatisfiable [22].

Theorem 1: For every sargable conjunctive equality predicate, there exists a product of its atomic conditions called the concatenated predicate

Proof: Theorem 1 follows from the work of [47] and [22]. Muralikrishna and DeWitt [47] and Heimel et al. [22] defined a conjunctive predicate as, P_1 AND P_2 AND... AND P_m and as being equivalent to a product of its atomic predicate terms expressed as $\prod_{i=1}^{m} P_i$, where each P_i in both the product term and the conjunctive term is strictly a Boolean expression and each P_i in the sargable conjunctive equality predicate is an equality predicate.

b) Proof of Equivalence

Lemma 2: Two query predicates (conditions) are equivalent if they return *the same records from a compatible database* [11].

Theorem 2: The product of atomic predicates (concatenation operation) is a bijective function on the set of concatenated predicates from the set of sargable conjunctive equality predicates thereby defining their equivalence

Proof: Let $a \in A$, where A is the set of sargable conjunctive equality predicates and $b \in B$, where B is the set of the concatenated predicates and *D* is a compatible database containing the surrogate field derived from the concatenation of the natural fields of D in the sargable conjunctive equality predicates. *Let f* represent the operation that concatenates the atomic conditions in *a* to get *b*. Then $a \equiv b$ if and only if, *f* is a bijective function. To be bijective, *f* must be onto as well as one-to-one:
f is onto because each concatenated predicate, *b* in B is in the image of *f* . That is,

$$\forall\, b \in B, \exists\, a \in A \text{ and } f(a) = b \quad \ldots$$

f is one-to-one because for a concatenated predicate, b ∈ B there is at most one *a* ∈ A such that *f*(a)= b . That is,

$$\forall\, a, a` \in A \text{ and } f(a) = f(a`) \text{ implies } a = a`$$

where *a`* is the inverse of *a* going by the concatenation operation implied by *f.*
Given that (3.10) and (3.11) holds, we conclude that
f: A → B is a bijective function on the set of concatenated equality predicates to the set of conjunctive predicates because *f* is both one-to-one and onto.
⇒ A ⇔ B and hence they return the same number of records for a compatible database

c) Proof of Optimisation

The generic optimisation model of a relational database query is described in terms of a relational algebra expression. The relational algebra corresponding to a query describes a set of operators whose number can be determined and their cost estimated. Based on either number of operators or estimated cost, two queries can be compared to ascertain that one optimizes the other. A relational algebra expression *e`* optimises another relational algebra, *e* if the following conditions are satisfied (1) *e`* is equivalent to *e* given a compatible database (2) the query time of *e`* is less than that of *e*. *e`* is optimal if a relational algebra expression that optimises *e`* does not exists.

Theorem 3: A predicate, *C`* is optimal relative to the conjunctive equality predicate, *C,* if (1) *C`* *is* equivalent to the conjunctive equality predicate, *C (2) C`* has fewer occurrences of equality predicates than *C,* and (3) there exists no other predicate, *p* that is equivalent to *C`* and has fewer occurrences of equality predicate than *C`*.

Proof:

The proof consists of a lemma and a proof by induction on $n(\sigma)$, the number of equality atomic conditions in the predicate, σ. The lemma establishes the equivalence of *C* to *C`*, while the poof by induction establishes the optimality of *C`* compared to *C* .

Lemma: Two query predicates (conditions) are said to be equivalent if they each *return the same records from a compatible database* [11].

Induction hypothesis: Consider the query execution tree in Figure 3 and let $n(\sigma)$= number of atomic predicates. Each atomic predicate in *C* corresponds to a predicate operator (σ_c=σ_1, σ_2, .. ., σ_m). In all circumstances, $n(\sigma)$=1 for *C`* since *C`* is a product of the terms of *C*. Being a product, the number of terms in *C`* is $m = 1$ and so $n(\sigma) = 1$ for *C`*.

Initial Induction Step: The height of the tree corresponding to *C* = number of atomic predicates in *C* = $n(\sigma_c) = m$, where *m* is the number of atomic predicates in *C* . Assuming $n(\sigma_c) = m = 5$ as the initial induction step, *m* is defined in subsequent induction steps as *m-i*, where *i* is the subsequent induction step, defined as 1,2, .., *m.* That is, in each subsequent induction step, we decrease the number of atomic predicates, *m,* by one at a time.

Subsequent Induction Steps:
When i=1, then m-i=5-1=4 implying $n(\sigma) = 4$
When i=2, then m-i=5-2=3 implying $n(\sigma) = 3$
When i=4, then m-i=5-4=1 implying $n(\sigma) = 1$

When i=5, then m-i=5-5=0 implying $n(\sigma) = 0$
When m approaches 0, $n(\sigma)$ approaches 0

This means that the number of operators, $n(\sigma)$ decreases in proportion to the number of atomic predicates, m. Mathematically,

$$n(\sigma) = m,\ m = 1, 2, .. , \infty$$

The query does not work and is undefined for when $m = 0$, $n(\sigma) = 0$. The query processes the least number of selection operators and hence does the least amount of work when $m = 1$, $n(\sigma) = 1$. Recall that $n(\sigma) = 1$ for predicate $C`$ . This means that $C`$ is optimal since any other reduction of the atomic predicates in C will result in a predicate that has $n(\sigma)<1$ and by the lemma, the equivalence of C since $C`$ is proved`

The proof assumes that the cost of execution of a predicate is directly proportional to the number of atomic predicates that makes it up. This was proved by previous experiments.

One of two equivalent predicates optimizes the other if the query time associated with the optimising predicate is lesser and both predicates are equivalent given a compatible database.

Proof: The relational algebra of the concatenate predicate and the sargable conjunctive equality predicate are made up of the same operator, the selection operation. The proof that the concatenate predicate, $C = \prod_{i=1}^{m} P_i$ has fewer occurrences of operators than *CP* the conjunctive predicate and hence optimizes the sargable conjunctive equality predicate is as follows

Let the compatible database be R and the relational algebra corresponding to the sargable conjunctive equality predicate be e and the algebra of the concatenated predicate be $e`$. Then $e= \sigma_{P1 \text{ and } P2, \text{ and...} \text{ and } Pm}(R)$ and $e`= \sigma_p(R)$ where the number of atomic predicates in e, $|e| = m$, $m>1$. Given that $e`$ is a product, it follows that $|e`| = 1$. Given $|e| = m$, $m>1$ and $|e`| = 1$, we can assume the minimum value of m=2 for $e$ resulting in $e= \sigma_{P1 \text{ AND } P2}(R)$. The relational algebra of $e$ and $e`$ consist of the selection operation, σ whose implementation uses either the sequential search (table scan) or the binary search (indexed scan). Since both e and $e`$ are made up of the same operation, it is convenient to assume that table scan has been used to implement them in the following algorithmic procedure.

The algorithmic steps corresponding to $e = \sigma_{P1 \text{ and } P2}(R)$ are:

1. Apply the selection operator σ_{P1} to get the intermediate relation, I_1
2. Apply the selection operator σ_{P2} to get the intermediate relation, I_2, the final result Analysis:

Assume the arity of the intermediate results, I_1 and I_2 to be approximately of the uniform value, n respectively. Let the total query time of e be $f_e(n)$. then $f_e(n)$ = query time of I_1 + query time of $I_2 = n+n = 2n$
The algorithmic steps corresponding to $e` = \sigma_p(R)$ are:

1. Apply the selection operator σ_p to get the intermediate relation, I the final result

Analysis

Assume the arity of the intermediate results, I_1 and I_2 to be approximately of the uniform value, n respectively. Let the total query time of $e`$ be $f_{e`}(n)$. then $f_{e`}(n)$ = query time of $I = n$
Clearly, $f_e(n) > f_{e`}(n)$, meaning that the query time of $e`$ is less than that of e implying that $e`$ optimises e.
The proof of optimality follows.

d) Proof of Optimality

Theorem 4: Given two equivalent relational algebras where one optimises the other, the one that optimises is optimal if a relational algebra expression that optimises it does not exist

Proof: The proof that the concatenated predicate, C that optimizes the conjunctive equality predicate, *CP* is optimal is proved by induction on m, the number of predicates in both C and *CP*.

Induction hypothesis: Each atomic predicate in *CP* corresponds to a predicate operator (σ) on the logical plan tree. Assuming m to be the number of atomic predicates in *CP* and $n(\sigma)$ be the number of predicate operators on the corresponding logical plan tree, then for every additional atomic predicate, $n(\sigma)$ increases by 1 such that $m = n(\sigma)$.

Induction Step: Assuming *CP* has a single atomic predicate, then $m = 1$ and $n(\sigma) = 1$. Proceeding with the induction steps, we increase the number of atomic predicates, m, by one at a time to get,
When $m = 2$, $n(\sigma) = 2$
When $m = 2$, $n(\sigma) = 3$
...When m approaches ∞, $n(\sigma)$ approaches ∞

This means that the number of operators, $n(\sigma)$ grows in proportion to the number of atomic predicates, m. Mathematically,
$n(\sigma) = m,\ m = 1, 2, .. , \infty$
But C is a product, implying that the number of terms in C is m = 1 and so $n(\sigma) = 1$ for every C corresponding to CP

For there to exist a predicate that optimises C, the occurrence of operators in such a predicate, m, must be zero, that is $m < 1$. If $m = 0$, then $n(\sigma)$ will also be zero. $n(\sigma) = 0$ defines a predicate that has no atomic predicate which is non-existent and hence a predicate that optimises C is non-existent. This means C having one operator has the least number of operators and hence it is optimal.

In this section, the proof of correctness of the concatenated predicate model has demonstrated. Table

3.4 shows the equations used to model the various components of the model.

Table 4: Summary of Model Equations

Model Component	Equation
Conjunctive Predicate	$\bigwedge_{i=1}^{m} \theta_i = P_1 \; AND \; P_2 \; AND \; \ldots AND \; P_m$
Conjunctive Predicate as a product of terms	$\prod_{i=1}^{m} P_i$
Concatenate Predicate	$C = \prod_{i=1}^{m} v_i$
Surrogate Index	$t(S) = \prod_{i=1}^{m} vs_i{}^{pa}$
Model Evaluation	$\theta_i(t) = \begin{cases} true, & if \; t(S_i) = C \\ false, & otherwise \end{cases}$

V. Conclusion and Suggestion for Further Work

The optimization of queries where complexity is due to a large number of joins has received a lot of attention in the database literature, but the optimization of complex selection predicates involving multiple ANDs (conjunctive predicates) and ORs (disjunctive predicates) has not been widely addressed [24]. In lieu of the dearth of research into selection predicates, the contribution to knowledge of this research effort can be said to be significant. Enhancing the query times of sargable conjunctive equality predicates is significant in the following ways:

1. The optimisation of the sargable conjunctive equality predicates within the context of big data minimises query time which tend to increase with big data
2. Sargable conjunctive equality predicates are widely used in applications involving data extraction, mining, matching and resolving data entities [56]. Enhancing these predicates directly improves the running times of applications designed to automate these operations.
3. Considering the very many other areas in which an improved query time can be of use, the research is of significance to software architects, software developers, the software industry and researchers.

The concatenated predicate model works very much like an index hence we can refer to it as a surrogate index. In our subsequent work, the concatenated predicate model will be experimentally validated and work on how to integrate the surrogate index into existing DBMS architecture studied.

References Références Referencias

1. Storey VC, Song I. Big data technologies and management: What conceptual modeling can do? Data & Knowledge Engineering. 2017; 108:50–67
2. Obilikwu, P., Ogbuju, E. (2020) A data model for enhanced data comparability across multiple organizations. *J Big Data* **7,** 95 (2020). https://doi.org/10.1186/s40537-020-00370-1
3. Khan MA, Uddin MF, Guptam N. Seven V's of Big Data: Understanding Big Data to extract value. Proceedings of 2014 Zone 1 Conference of the American Society for Engineering Education (ASEE Zone 1). 2014.
4. Patgiri R, Ahmed A. Big Data: The V's of the game changer paradigm. International Conference on High-Performance Computing and Communications. 2016.
5. Matalqa H. and Mustafa S.H. (2016): "The Effect of Horizontal Database Table Partitioning on Query Performance", The International Arab Journal of Information Technology, Vol. 13, No. 1A, 2016
6. Das, T.K. & Mohapatro, Arati. (2014). A Study on Big Data Integration with Data Warehouse. International Journal of Computer Trends and Technology. 9. 188-192. 10.14445/22312803/IJCTT-V9P137.
7. Tailor, U., and Patel, P. (2016). A Survey on Comparative Analysis of Horizontal Scaling and Vertical Scaling of Cloud Computing Resources. IJSART - Volume 2 Issue 6, ISSN [ONLINE]: 2395-1052.
8. Stonebraker, M., & Çetintemel, U. (2018). One size fits all: an idea whose time has come and gone. In Proceedings of the International Conference on Data Engineering (ICDE), 2-11. 10.1145/3226595.3226636.
9. Selinger, P. G, Astrahan, M.M, Chamberlin, D.D, Lorie, R.A, Price, T.G (1979): "Access Path Selection in a Relational Database Management System",

SIGMOD Conference 1979, Boston, Massachusetts, May 30 - June 01, pp. 23-34.
10. Obilikwu P.O., Kwaghtyo K.D and Ogbuju E. (2021), Enhancing Query Time Using a Volume-Adaptive Big Data Model OF Relational Databases, *The Journal of Basic Physical Research.* Department of Geological Sciences, Nnamdi Azikiwe University, Awka, Anambra State, Nigeria
11. Chandra, A. K. and Merlin, P. M.(1977): "Optimal implementation of conjunctive queries in Relational Databases", Proceedings of the 9th ACM Symposium of Theory of Computing, Boulder, Colorado, USA, May 4th, pp. 77-90.
12. Levy, A.Y., Mumick, I. S. and Sagiv Y. (1994): "Query Optimization by Predicate Move-Around", Proceedings of the 20th VLDB Conference, Santiago, Chile, September 12-15, pp. 96-107.
13. Abiteboul, S., Hull R. and Vianu V.(1995): "Foundations of Databases", Addison_Wesley, Reading, MA. pp. 35 – 65.
14. Gottlob, G., Lee, S.T. and Valiant, G. (2012): "Size and Tree width Bounds for Conjunctive Queries", Journal of the ACM, Volume 59 Issue 3, Article No. 16
15. Swami, A. and Scheifer, K.B. (1993): "On Estimation of Join Result Sizes", Technical Report, IBM Research division, IBM Research Report RJ9569
16. Grohe, M., Schwentick, T. and Segoufin, L. (2001): "When is the evaluation of Conjunctive Queries Tractable", Proceeding of the 33rd Annual ACM symposium on Theory of Computing, Hersonissos, Greece, July 6-8, pp. 657 – 666
17. Mohan, C., Haderle, D. J., Wang, Y., and Cheng, J. M. (1990): "Single Table Access using Multiple Indexes: Optimization, execution, and concurrency control techniques", International Conference on Extending Database Technology (EDBT), Venice Italy, March 26-30, Volume 416 of LNCS, pp. 29–43.
18. Elmasri, R. and Navathe, S. B. (2011): "Fundamentals of Database Systems", 6th Edition, Pearson Education Inc, pp. 679 - 723
19. Garg, V.K. and Waldecker, B. (1994):"Detection of weak unstable predicates in distributed programs", IEEE Transactions on Parallel and Distributed Systems, Volume: 5, Issue: 3, Pp: 299 – 307
20. Mugnier M., Rousset M. and Ulliana F. (2016): "Ontology-Mediated Queries for NOSQL Databases", Association for the Advancement of Artificial Intelligence
21. Munir, K. and Anjum, M.S. (2017): "The use of ontologies for effective knowledge modelling and information retrieval", Applied Computing and Informatics (2017),http://dx.doi.org/10.1016/j.aci. 2017.07.003
22. Heimel, M., Markl, V. and Murthy, K. (2009): "A Bayesian Approach to Estimating the selectivity of Conjunctive Predicates", Proceedings of Datenbanken und Informationssysteme (DBIS), Münster, Germany, March 2-6, pp 47-56
23. Yu, X., Koudas, N., and Zuzarte, C. (2006): "HASE: A Hybrid Approach to Selectivity Estimation for Conjunctive Predicates", Advances in Database Technology - EDBT 2006, Springer International Publishing, AG, Volume 3896 of the series Lecture Notes in Computer Science pp. 460-477.
24. Chaudhuri, S., Ganesan, P. and Sarawagi, S. (2003): "Factorizing Complex Predicates in Queries to Exploit Indexes", ACM SIGMOD 2003, June 9-12, San Diego, CA. pp. 361-372
25. Kemper, A., Moerkotte, G., Peithner, K., and Steinbrunn, M. (1994): "Optimizing disjunctive queries with expensive predicates", ACM Intl. Conference on Management of Data (SIGMOD), Minneapolis, Minnesota, May 24-27, pp. 336–347.
26. Lohman, G. (2014): "Is Query Optimization a 'Solved' Problem?", ACM Special Interest Group on Management of Data blog, http://wp.sigmod.org/
27. Chaudhuri, S. (2012): "What next?: a half-dozen data management research goals for big data and the cloud", Proceeding PODS '12 Proceedings of the 31st ACM symposium on Principles of Database Systems, Scottsdale, Arizona, USA — May 21 - 23, pp. 1-4
28. Codd, E. F. (1970): "A Relational Model of Data for Large Shared Data Banks". Communications ACM 13(6): 377-387
29. Chamberlin, D.D., Astrahan, M.M., Blasgen, M. W., Gray, J. N., King, W. F., Lindsay B. G., Lorie, R., Mehl, J. W., Price, T, G., Putzolu, F., Selinger, P. G., Schkolnick, M., Slutz, D.R., Traiger, I. L., Wade, B. W., Yostet, R. A. (1981): "A History and Evaluation of System R", Communications of ACM 24(10): Pp. 632-646
30. Clough, L., Haseman, W.D. and So, Y.H.(1976): "Designing Optimal Data Structures", AFIPS national computer conference and exposition, New York, New York — June 07 - 10, pp. 829-837.
31. Codd, E. F. (1975): "Implementation of Relational Database Systems", Panel Discussion, NCC (AFIPS) 75, Anaheim.
32. Chaudhuri, S. (1998): "An Overview of Query Optimization in Relational Systems", Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems
33. Ioannidis, Y. (2003): "The History of Histograms", Proceedings of the 29th VLDB Conference, Berlin, Germany, September 9-12, pp. 19-30.
34. Cao, B. and Badia, A. (2005): "A Nested Relational Approach to Processing SQL Subqueries", SIGMOD 2005 June 14 - 16, 2005, Baltimore, Maryland, USA, pp. 191 – 202

35. Vellev, S. (2009): "Review of Algorithms for the Join Ordering Problem in Database Query Optimisation", Information Technologies and Control, pp. 32 – 40
36. Bamnote, G. R. and Agrawal, S.S. (2013): "Introduction to Query Processing and Optimization ", International Journal of Advanced Research in Computer Science and Software Engineering, Volume 3, Issue 7, Pp. 53 – 56
37. Ullman, J. (1989): "Principles of Database and Knowledge Base System", Volume 1, Computer Science Press Inc., New York, p. 631.
38. Chimenti, D., Gamboa, R. and Krishnamurthy R. (1989): "Towards an open architecture for LDL", Proceedings of the fifth International VLDB Conference, Amsterdam, the Netherlands, August 22-25, pp 195-203
39. Chaudhuri, S. and Shim, K. (1993): "Query optimization in the presence of foreign functions", Proceeding of the 19th International VLDB Conference, Dublin, Ireland, August 24 – 27, pp. 529 – 542.
40. Hellerstein, J. M. and Stonebraker, M. (1993): "Predicate Migration: Optimising queries with Expensive Predicates", SIGMOD Conference, Washington DC, May 25-28, pp. 267–276
41. Chaudhuri, S. and Gravano, L (1996): "Optimizing queries over multimedia repositories", ACM International Conference on Management of Data (SIGMOD), Montreal, Quebec, Canada, June 4-6, pp 91–102.
42. Chaudhuri, S. and Shim, K. (1999): "Optimization of queries with user-defined predicates", ACM Transactions on Database Systems (TODS), 24(2), June 1-3, Seattle, Washington, USA, pp. 177–228.
43. Kemper, A., Moerkotte, G., and Steinbrunn, M.(1992): "Optimizing Boolean expressions in object Bases", Proceedings of the VLDB Conference, Vancouver, Canada, August 23-27, pp 79-90
44. Balasubramanian, P. and Arisaka R. (2007): "A Set Theory Based Factoring Technique and Its Use for Low Power Logic Design", World Academy of Science, Engineering and Technology, 3, pp. 446 – 456
45. Brayton, R.K., Rudell R. and Sangiovanni-Vincentelli, A. and Wang, A. (1987): "MIS: A multiple-level logic optimization system", IEEE Transactions. on CAD of Integrated Circuits and Systems, Vol 6, Issue 6, pp. 1062-1081.
46. Reinwald, L.T. and Soland, R.M. (1966): "Conversion of Limited-Entry Decision Tables to Optimal Computer Programs: Minimum Average Processing Time", JACM, 13(3), Pp 339-358
47. Muralikrishna, M. and DeWitt, D. J. (1988): "Optimization of multiple-relation multiple-disjunct queries, In Proceedings of the Seventh ACMSIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Austin, Texas, March 21 – 23, pp 263-275.
48. Hellerstein, J. M. (1994): "Practical Predicate Placement", Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, May 24-27, pp. 325-335
49. Deen, S. M. (1982): "An implementation of impure surrogates", International Conference on Very Large Databases, Mexico City, September 8-10, pp. 245-256.
50. Deen, S. M., Amin, R. R. and Taylor, M. C. (1994): "A Strategy for Decomposing Complex Queries in a Heterogeneous DDB", Proceedings of the Tenth International Conference on Very Large Databases, Singapore, August 27-31, pp. 397-400.
51. Oracle (2017): "Oracle Sharding Linear Scalability, Fault Isolation and Geo-distribution for Web-scale OLTP Applications", ORACLE White Paper, April 2017.
52. Sander-Bruggink, H.J., Konig, B. and Kupper S. (2013): "Concatenation and other Closure Properties of Recognizable Languages in Adhesive Categories", Proceedings of the 12th International Workshop on Graph Transformation and Visual Modeling Techniques, Mar 23 – Mar 24 2012, Rome, Italy
53. Lynch, C. and Stonebraker M. (1988): "Extended User-Defined Indexing with Application to Textual Databases", Proc. 14th International Conference on Very Large Datsbases, Los Angeles, August 29 – September 1, pp. 306 – 317
54. Harkins, S. (2011): "10 Tips for Choosing between a Surrogate and Natural Primary Key". Retrieved from www.techrepublic.com, pp 1-2
55. Valduriez, P. (1987): "Join Indices", ACM Transactions on Database Systems, Vol. 12, No. 2, June 1987, pp. 218-246.
56. Getoor, L. and Machanavajjhala, A. (2012): "Entity Resolution: Theory, Practice and Open Challenges", Proceedings of the VLDB Endownment, Istanbul, Turkey, August 27-31, Vol 5, No. 12, pp 2018 – 2019.