Heuristics for Selecting Robust Database Structures with Dynamic Query Patterns

Andrew N. K. Chen
School of Accountancy and Information Management
College of Business
Arizona State University
Tempe, Arizona 85287-3606
andrew.chen@asu.edu
(480) 965-2687

Paulo B. Goes
Department of Operations and Information Management
School of Business
University of Connecticut
Storrs, CT 06269-2041

Alok Gupta
Information and Decision Sciences Department
Carlson School of Management
University of Minnesota
Minneapolis, MN 55455

James R. Marsden
Department of Operations and Information Management
School of Business
University of Connecticut
Storrs, CT 06269-2041

March, 2004
Heuristics for Selecting Robust Database Structures with Dynamic Query Patterns

ABSTRACT

The success of a company increasingly depends on timely information (internal or external) being available to the right person at the right time for crucial managerial decision-making. Achieving such a “right time/right place” duet depends directly on database performance. A database system has been a core component that supports modern business system such as enterprise resource planning (ERP) system that integrates and supports all enterprise processes including product designing and engineering, manufacturing, and other business functions to achieve highest efficiency and effectiveness of operations. We develop and demonstrate through a proof-of-concept case study, a new “query-driven” heuristics for database design that seeks to identify database structures that perform robustly in dynamic settings with dynamic queries. Our focus is the design of efficient structures to process read-only queries in complex environments. Our heuristics begins with detailed analysis of relationships between diverse queries and the performance of different database structures. These relationships are then used in a series of steps that identify “robust” database structures that maintain high performance levels for a wide range of query patterns. We conjecture that our heuristics can facilitate efficient operations and effective decision making of companies in today’s dynamic environment.

Keywords – Heuristics, Computing Science, Database Management, Simulation
I. Introduction

The amount of data and information that a company must keep track of is tremendous and increasingly complicated. At the same time, these data and information need to be processed quickly and available timely. In today’s business world, the success of a company increasingly depends on timely information (internal or external) being available to the right person at the right time for crucial managerial decision-making. This is especially true for companies involved in high-tech and engineering processes. Achieving such a “right time/right place” duet depends directly on database performance. A database system has been a core component that supports modern business system such as enterprise resource planning (ERP) system that integrates and supports all enterprise processes including product designing and engineering, manufacturing, and other business functions to achieve highest efficiency and effectiveness of operations. The task is complicated by the dynamic activities that have become an integral part of a firm’s operations.

As companies participate in electronic commerce and Internet economy, it is common for a firm to maintain multiple web portals, each with multiple linked sites. Customers, business partners, and employees access the sites and linked content seeking different information for different uses. For the most part in this environment, reading the data is the principal database activity, with writing comprising a much smaller proportion of database operations. Provision of these customized web sites may offer competitive advantage when dealing with customers or business partners, but adds complexity and uncertainty to the situation. To provide such customization requires extraction of information from various backend applications such as databases in ERP systems. A decision to utilize customization implies that the customers and business partners who access the sites are “significantly different,” that is, they seek different
information. Our purpose here is to set forth database design procedures that identify robust database structures that perform well across diverse and uncertain query sets, i.e., situations such as that just described.

Traditional database design focuses on data properties and associated relationships [5, 12, 13, 14, 15, 29, 35, 41, 44, 45]. Queries and query sets are assumed known or are predictable. Relational database design methodologies that generate third normal form tables are suited to such predictable, transaction oriented environments. It is well recognized, however, that third normal form database structures are not efficient for processing \textit{ad hoc} read queries in a decision-oriented environment. \textit{The reality is that no single database structure, regardless of how carefully it might have been designed and operationally tuned, is able to provide acceptable performance levels to every possible query in today’s e-business environment of several integrated applications.} This fact has prompted some organizations to adopt the following two general solutions based on maintaining additional structures to better process important read-only queries:

\textbf{The data warehousing solution:} This solution establishes a dichotomy of database structures and design approaches: one structure is designed for efficient processing of transactions and another set of structures, such as a data warehouse, is designed for the processing of \textit{ad hoc} decision-oriented queries. Today the design of data warehouse structures is still based on anticipating (fixing) the decision-oriented queries that will be most prevalent with the common practice of identifying and pre-establishing specific subsets of data (views) to be materialized for efficiency of processing purposes [4, 24].

\textbf{Application-specific materialized views solution:} Another important example of making available additional database structures for read-only access in parallel with the
transaction-oriented structure comes from today’s implementation of web-based applications. For obvious security and performance reasons, these applications commonly maintain a separate set of materialized views of the production database as the source of data for web access.

Replicating certain parts of the transaction database for read-only purposes allows users of web-based applications to browse large volumes of data without interfering with the transaction system. Delta Airlines’ new DNS (Delta Nervous System) implements this concept through various read-only data stores that are fed through a middleware layer [43]. All major database vendors currently provide replication solutions aimed at read-only snapshots, which tend to be created to support specific web applications.

As noted earlier, today’s business world environment is anything but stable or predictable. Customization is becoming a user expectation. Information demands and the queries necessary to meet those information demands can change rapidly. In this paper we take a step further and consider the design of a database system that can consistently answer well a variety of read-only queries in highly dynamic environments. This database concept, termed flexible database system (FDS) is based on keeping various database structures in parallel, which are designed to answer a wide range of queries in diverse dynamic conditions. Notice that materialized views can be seen as special cases of the alternative structures that are kept. The FDS approach is overviewed in Section II; details can be found in Chen et al [11].

We develop heuristics to identify the alternative structures that will be used by the FDS. Our heuristics begin with detailed analysis of relationships between diverse queries and the performance of different database structures. These relationships are then used in a series of steps that identify “robust” database structures that maintain high performance levels for a wide

---

range of query sets. Figure 1 juxtaposes our approach with the traditional database design approach.

Figure 1. Traditional database design and robust database design

In a conceptual way, our heuristics can be viewed in the same mold as robust optimization [3, 31, 34]. Effort is focused on obtaining “best performers” or “best practices” for a segment of alternative scenarios. Here, the scenarios are query sets and best performance is least processing time. Our heuristics also mimics conceptual idea of Chaturvedi and Choubey [7]. They utilize knowledge about the data usage patterns for each node in a distributed database environment to design the invariant fragments and schedule its allocation and selective update for a specific time period. Here, we use knowledge of query patterns and relevant performance
for each database structure to prepare the "right" database structure for query processing as queries change dynamically.

Our presentation is organized as follows. Section II reviews the relevant database literature and describes the flexible database system concept in more detail. Section III provides the conceptual description of the query-driven database design heuristics. Our experiment setup and evaluation procedures are also described in detail. Section IV depicts four query classification approaches, and, through careful comparison, the selection of the most effective approach to classify queries into different complexity levels. Section V introduces our techniques that group database structures in order to reduce problem dimension for database design. Section VI presents our approach for selecting robust database structure(s). Section VII summarizes our study and discusses limitations and future research directions.

II. Relevant Database Literature

Traditionally, relational database design mainly focuses on modeling data in a way that the database is implemented through a schema that is non-redundant and well-structured [12, 13, 14, 35, 41, 44, 45]. In order to improve performance of databases, many techniques such as query optimization [18, 19, 26], database tuning [37, 38, 39], and adaptive query processing [2, 23, 25] have been proposed. However, such databases are prone to unsatisfactory performance with dynamic (ad hoc) queries or sets of queries of very different types and processing needs. In particular, large and frequent variations of queries are very likely to reduce the effectiveness of query optimization techniques and consequently degrade database performance. Recognizing this fact, materialized views [1, 6, 8, 21, 40] and the data warehouse concept [4, 9, 24, 26] have been proposed and have gained commercial popularity. The goal is to provide the means to
timely answer the most frequently posed queries and decision-oriented queries that the operational database system is poor at answering. However, because these techniques need to "predict" or "anticipate" specific queries that will occur in the future, significant limitations remain in today's operationalization of these techniques, especially in the range of queries that can be answered efficiently. There are substantial performance issues associated with answering ad hoc queries whose requested data items have not been appropriately materialized. That is, these techniques do not cope well with uncertainty of query-processing needs.

Further, there are different join techniques (e.g., nested-loops join, sort-merge join, hash join, pointer-based join) that perform differently in various kinds of computing environments [33, 36]. There are also many performance tuning and query optimization techniques (e.g., on query representation, query transformation, access plan) that can be implemented for query processing [26, 37]. The specific relational database management systems (RDBMS) used in a specific environment has its own mechanism to incorporate and implement these techniques.

**The Flexible Database System Concept**

Additional database structures in the form of data warehouses or materialized views to handle read-only queries are already common practices. Recognizing this together with the increasing need to deal with unanticipated queries in very dynamic business environments, Chen et al. [11] proposed an architecture and methodology for the design and management of flexible database systems. In such systems, a transaction-oriented database structure will be maintained along with the alternative read-only structures. As a complex read-only query is posed to the system, it is assigned to the read-only structure that is best suited to handle it. The original transaction-oriented database structure can be derived through the traditional database design process with, for example, E-R modeling and normalization. Alternatively, this structure may
already be in place, and will be supplemented with additional read-only robust structures, which
will be chosen to process a wide range of queries. Notice that special cases of these alternative
structures are materialized views of parts of the production database.

The read-only structures are fed with updates received by a separate transaction-oriented
database structure that is specifically designed and implemented to capture the update activity of
the system. Updates will be reflected onto the read-only structures through periodic refresh
operations. The database administrator (DBA) can also trigger updates to the read-only
structures at any point in time or they can be automatically executed when specific conditions are
met. Please refer to Chen et al. [11] for a detailed description of the flexible database system and
the overall methodology proposed for its design and implementation. In this paper we propose
solutions for identifying a robust set of candidate read-only structures that together with the
transaction-oriented structure will constitute the flexible database system.

III. Conceptual description and proof of concept of the query-driven database design
heuristics

To design the set of database structures that will cover the widest range of possible
queries, one needs to first have a thorough understanding of the underlying information content.
The starting point for this is the construction of an accurate conceptual/logical representation of
the underlying database application. Entity-Relationship or UML approaches that capture the
information content of the database environment are essential.

The first goal is to list possible categories of queries that can be posed to a database with
such information content. These queries vary from straightforward and predicted operational
queries to queries that increasingly integrate different portions of the information content. It is
useful to first generate queries using the reference (normalized) structure that can achieved from
the conceptual model. As will be discussed later in this section, for each possible information
source of the normalized structure (every table, every feasible join of tables), queries of different
complexity levels are created.

The second goal is to create a set of feasible candidate read-only structures, which have
equivalent information content of the conceptual representation under the assumption that the
natural join operations of forming these database structures are lossless joins. That is, these
database structures may have different numbers of denormalized tables but they have the same
actual information content. All these database structures should have "equal setup" - the same
storage scheme, indices, and information content. The variation in structure (i.e., the number of
tables and tables combined from different base tables) is the only difference.

Query performance of a database structure depends on many different factors. Setting
indices on non-key attributes and applying specific storage schemes can improve query-
processing speed. However, in this study, we do not attempt to find the "best" index
arrangement or storage scheme to "reduce" query processing time for each database structure,
procedures that may be important when the query set and arrival rates are fixed and known. Our
goal is to investigate a design process that yields robust performance in the face of changing
query patterns.

As explained in significant detail in Chen [10], our approach is based on the traditional
normalized data structure designed for the environment. We construct alternative structures of
equivalent information content by exhaustively joining two tables at a time. Figure 2 provides a
general description of our query-driven heuristics for database design.
Through comprehensive experiments, we observe performance (speed of query processing) for a complete enumeration of combinations of query complexity level/database structure alternatives. We then classify database structures into groups that have very similar performance in processing queries across different given complexity levels. In each such group of database structures, we select a database structure as the representative of database structures for that group. Finally, from the set of representative database structures, we identify efficient database structure(s) for processing various queries. For each individual query, we measure processing times for each alternative structure. For each structure, deviation from fastest processing time among the set of alternative structures yields a measure of relative performance. These individual values are then used to construct mean deviation and mean squared deviation.
from best performing database structure processing time (BPDSPT). A BPDSPT of a query is the shortest time of all measured processing times for that query of each database structure.

$$BPDSPT_i = \text{Min}\{PT_{ij} : \forall j\}$$

where \(j = 1, ..., J\) representing database structures generated for the experiment and \(PT_{ij}\) is the processing time of query \(i\) by database structure \(j\).

Therefore, the mean deviation from BPDSPT for a database structure \(j\) is calculated as

$$\frac{\sum_{i=1}^{I} (PT_{ij} - BPDSPT_i)}{I}$$

where \(i = 1, ..., I\) representing queries run in the experiment.

The squared mean deviation from BPDSPT for a database structure \(j\) is calculated as

$$\frac{\sum_{i=1}^{I} (PT_{ij} - BPDSPT_i)^2}{I}$$

where \(i = 1, ..., I\) representing queries run in the experiment.

For example, consider a simple example with three structures (S1, S2, and S3) and three queries (q1, q2, and q3) with the following processing time results:

<table>
<thead>
<tr>
<th></th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>q1</td>
<td>0.5 seconds</td>
<td>0.6 seconds</td>
<td>0.6 seconds</td>
</tr>
<tr>
<td></td>
<td>(BPDSPT for q1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>q2</td>
<td>0.8 seconds</td>
<td>0.5 seconds</td>
<td>0.6 seconds</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(BPDSPT for q2)</td>
<td></td>
</tr>
<tr>
<td>q3</td>
<td>1.0 seconds</td>
<td>1.5 seconds</td>
<td>2.0 seconds</td>
</tr>
<tr>
<td></td>
<td>(BPDSPT for q3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thus the mean deviation from BPDSPT for structure S1 would be \(((0.5 - 0.5) + (0.8 - 0.5) + (1.0 - 1.0))/3\) or \((0 + 0.3 + 0)/3\) or 0.1. The mean squared deviation from BPDSPT for structure S1 would be \(((0.5 - 0.5)^2 + (0.8 - 0.5)^2 + (1.0 - 1.0)^2)/3\) or \((0 + 0.09 + 0)/3\) or 0.03. The mean deviation from BPDSPT for S2 would be \(((0.6 - 0.5) + (0.5 - 0.5) + (1.5 - 1.0))/3\) or \((0.1 + 0 + 0.5)/3\) or 0.2.

In our analysis, we investigate the grouping of queries into "complexity classes or levels" and analyze the performance of individual database structures on each query complexity class or
level. We identify structures that perform the best for queries in each specific complexity level. We then determine whether there are database structure(s) that yield robust query-processing performance across the array of different complexity levels.

III.1. Proof of Concept

A generic database was constructed for our proof-of-concept experiments that utilized the popular commercial database management system, Microsoft SQL Server. Key elements of our experimental process, including: database construction, query formulation, computing environment, and simulation process, are outlined in the following subsections.

Database

We started with a database application whose conceptual schema was initially mapped into a relational schema in third normal form consisting of seven tables. Each table has ten to forty attributes and 625 to 6,250 tuples (records). Values of numerical fields in tables were randomly generated within the fields’ domain. Contents of textual fields in tables are fixed sixteen-byte text. The database layout is presented in Figure 3 and the database schema is presented in Table1.
A computer program was written to populate the seven tables of the original normalized database structure (named "T1"). For each table we set its first field as the primary key (e.g. Field101 of Table1 and Field201 of Table2). Field216, Field406, Field605, and Field708 are foreign keys associated with Table1 and linked with Field101. Field308 is a foreign key associated with Table2 and links with Field201. Field416 and Field705 are foreign keys associated with Table5 and links with Field501. All primary keys in tables were indexed as clustered and unique. All foreign keys in tables were also indexed but as non-clustered and non-unique.

<table>
<thead>
<tr>
<th></th>
<th>Table1</th>
<th>Table2</th>
<th>Table3</th>
<th>Table4</th>
<th>Table5</th>
<th>Table6</th>
<th>Table7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of Attributes</strong></td>
<td>40</td>
<td>30</td>
<td>15</td>
<td>20</td>
<td>35</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td><strong>Number of Tuples</strong></td>
<td>625</td>
<td>2,500</td>
<td>6,250</td>
<td>3,750</td>
<td>1,250</td>
<td>1,875</td>
<td>5,000</td>
</tr>
<tr>
<td><strong>Table Size (Reserved, K Bytes)</strong></td>
<td>160</td>
<td>526</td>
<td>668</td>
<td>608</td>
<td>270</td>
<td>318</td>
<td>494</td>
</tr>
<tr>
<td><strong>Data Size (Actual, K Bytes)</strong></td>
<td>140</td>
<td>456</td>
<td>570</td>
<td>442</td>
<td>250</td>
<td>268</td>
<td>304</td>
</tr>
<tr>
<td><strong>Index Size (K Bytes)</strong></td>
<td>2</td>
<td>36</td>
<td>78</td>
<td>134</td>
<td>2</td>
<td>30</td>
<td>134</td>
</tr>
<tr>
<td><strong>Construction Time (seconds)</strong></td>
<td>95</td>
<td>507</td>
<td>1278</td>
<td>1218</td>
<td>375</td>
<td>336</td>
<td>1019</td>
</tr>
<tr>
<td><strong># of Links</strong></td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td><strong>Tables Linked</strong></td>
<td>Table2, 4, 6, 7</td>
<td>Table1, 3</td>
<td>Table2</td>
<td>Table1, 5</td>
<td>Table4, 7</td>
<td>Table1</td>
<td>Table1, 5</td>
</tr>
</tbody>
</table>

**Database Size:** 3,824,000 bytes reserved, 2,546,000 bytes as data, 466,000 bytes as indices.

**Machinery:** Dell Dimension, Pentium II 233MHz

Table 1. Relational Database Schema in Third Normal Form

In principle, the determination of the set of candidate structures can be a daunting task because of the unlimited ways we could construct “materialized views” out of the original tables. This process is of course complicated by the fact that the query patterns that the database will face can vary dramatically. For proof of concept of our work, we chose to focus on candidate...
structures that enumerate all possible combinations of complete joins of the original normalized tables, resulting in 96 different database structures (including the original normalized structure). For simplicity, we did not consider structures containing materialized views of aggregation queries within the same table or partial joins involving selections within tables.

The various database structures were constructed using the seven basic relations presented in Table 1. For example, joining Table1 and Table2 yields Table12. Table12 has all fields from Table1 and Table2. Table12 also has indices on any field that is a primary key or a foreign key of Table1 and Table2. Table12 along with original Table3, Table4, Table5, Table6, and Table7 is treated as one new database structure named "T12". Table 2 presents examples of different database structures used in the experiment.

<table>
<thead>
<tr>
<th>Name of the database structure</th>
<th>Difference(s) from the original normalized database structure</th>
<th>Tables in the database structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>T17,23</td>
<td>Joining Table1 and Table7 as a new table named Table17 and joining Table2 and Table3 as a new table named Table23</td>
<td>Table17, Table23, Table4, Table5, Table6</td>
</tr>
<tr>
<td>T12367</td>
<td>Joining Table1, Table2, Table3, Table6, and Table7 all together as a new table named Table12367</td>
<td>Table12367, Table4, Table5</td>
</tr>
<tr>
<td>T457,1236</td>
<td>Joining Table4, Table5, and Table7 as a new table named Table457 and joining Table1, Table2, Table3, and Table6 as a new table named Table1236</td>
<td>Table457, Table1236</td>
</tr>
</tbody>
</table>

Table 2: Examples of different database structures and names for different database structures and tables

Queries

A computer program was developed to randomly generate 520 queries for the database, ten queries from each of 52 possible query forms. A query form specifies from what table(s) a query retrieves data. For example, a query that extracts data only from Table1 is one query form
(e.g. SELECT Table1.Field105 FROM Table1 WHERE Table1.Field108 > 3). A query that extracts data only from Table5 is another query form (e.g. SELECT Table5.Field513 FROM Table5 WHERE Table5.Field502 <= 100). A query that extracts data from both Table1 and Table6 is another query form (e.g. SELECT Table1.Field111, Table6.Field606 FROM Table1, Table6 WHERE Table1.Field101 = Table6.Field605 AND Table6.Field608 = 25). A query that extracts data from all seven tables is another query form. There are 52 query forms that provide an exhaustive set for our database design. Ten queries (SQL statements) were generated for each possible query form. Each query has one to ten requested attributes and one to five selection criteria in an SQL statement. In each query, the number of requested attributes, the number of selection criteria, and the comparison operators and values in the selection criteria were all randomly generated by our custom-written simulation program. All requested attributes in a query are unique; that is, no duplicated attributes are present in any query. While data processing in a real-world ERP environment certainly could involve queries with any possible logical and/or comparison operators in the selection criteria, it is generally known that (1) for logical operators queries with "OR" or a mix of "AND" and "OR" operators are more expensive to be processed than those with only "AND" operator and (2) for comparison operators queries with "<>" or other wild card operators (e.g., “like” or “*”) are more expensive to be processed than those with only “>”, “<”, “=”, “=”, or “=” operators [20, 32]. In the experiment that we conducted for this study, in order not to unnecessarily extend experimental processing time, we design queries that do not contain "OR" logical operator and "<>" (not equal) or "Like" comparison operators. We argue that for demonstration purpose without losing potential generalization the queries used in our experiment can exhibit our technique for selecting robust database structures with dynamic query patterns. For more extensive experiments, we can
include queries with all kind of logical and comparison operators to further validate our conjecture.

Since our focus was on information extraction rather than database maintenance, all queries generated were information retrieval queries. All 520 queries generated were run on each of 96 database structures. For this proof-of-concept experiment, it took less than 5 seconds for our program to randomly generate these 520 queries. Time needed to generate query sets will increase when the number of tables in a database structure increases (i.e., number of possible query forms increases). However, in general, not all tables in a database will be "linked" and the increase of time will not be exponential in relation to the number of tables.

Computing Environment

Ten identical Dell Dimension computers (Pentium II, 233 MHz, 64 MB RAM, running MS Windows NT 4.0) were used. The database management system was Microsoft SQL Server (with latest service pack). The Visual Basic was used to develop the custom experiment programs.

Experiment procedures

A computer program was written to create new table(s) in SQL Server for each of the 96 different database structures. Another computer program was written to run all 520 queries sequentially through each database structure. The results of all query-processing times were recorded (in millisecond precision) and written to text files. These data were then used to develop and test our query-driven database design heuristics.

We analyze results from our experiment in two dimensions – (1) query dimension, and (2) database structure dimension. In the query dimension, we develop a methodology to classify query complexity levels (Section IV). Based on the properties and nature of a query, we
investigate four possible ways, detailed in the next section, to classify queries into meaningful complexity levels: (1) query type (number of base relation(s) that a query needs to access for information), (2) number of attributes, (3) number of selection criteria, and (4) selectivity factor. As explained in Section V, in the database structure dimension, we develop a methodology to group together “similar” database structures based on performance correlation among database structures. The goal of this process is to reduce the dimensionality and to help demonstrate the potential applicability of our process. After classifying all possible database structures into groups, we select a representative database structure for each group.

**IV. Query classification**

Recall from Figure 2 that our query-driven approach to identify the set of robust database structures hinges on simultaneously considering the universe of feasible potential queries and the universe of feasible database structures. Our objective is to reduce the universe of feasible structures to a handful of robust structures, which will be well suited to efficiently process a wide range of potential queries. To reduce the dimensionality of the problem on the query side we propose a classification procedure that groups queries into different complexity levels. Working with a few query complexity levels allows us to explore the correlation among structures with respect to the time they take to process categories of queries. This is an important step in the robust optimization that identifies the set of alternative structures, as outlined in the next section.

We sought to develop a methodology of classifying queries into complexity levels where performance (processing time) would be expected to be similar across queries within a group but would be expected to vary across groups. The objective of the classification scheme is to ex-ante evaluate query properties and be able to categorize queries into different complexity levels.
The guidelines used to select a good query classification approach are as follows:

1) choose a classification approach that shows a clear correlation between average query processing times and query complexity levels. That is, when the query complexity level goes up, there should be a consistent trend (up or down) for the average query processing time;

2) choose a classification approach that results in reasonably small standard deviation of query processing time for each complexity level; and,

3) choose a classification approach that has semantic reasoning or theory background justifications.

While there are many potential query properties that might be considered for developing query complexity groupings, we chose to consider four that are commonly found in the literature [16, 17, 27, 28, 42]. The four factors are described in detail below.

*Number of base relations (query type)*

This factor is the number of base relation(s) that must be accessed to process a query. When a query extracts data from only one table in the original normalized database, we classify this query as belonging to complexity level 1. When a query extracts data from two tables in the original normalized database (i.e. we need to join two tables to answer this query), we classify this query as belonging to complexity level 2. We expect that the complexity level for a query is positively related to processing time because there could be longer join time and search time for those queries that need to access more tables. The rational shortcoming of this approach is when we pose the same query to a different database structure, the complexity level of the query can change. For example, a query that extracts data from both Table1 and Table2 in the original normalized database belongs to complexity level 2, but the same query that extracts data from Table12 in a database structure that has Table1 and Table2 already joined together belongs to complexity level 1 by definition.
**Number of attributes**

The number of attributes is the number of attribute(s) of base relation(s) that a query requests for information. When a query retrieves data for only one attribute in its view, this query would be classified as belonging to complexity level 1. When a query retrieves data for two attributes in its view, this query would be classified as belonging to complexity level 2. We expect that the higher the complexity level is for a query, the longer processing time would be because there is likely longer information extraction time for queries with more attributes in their views. This approach has the shortcomings of not counting the size of attributes or the size of views.

**Number of selection criteria**

The third potential factor is the number of selection criteria. When a query's SQL statement has only one selection criterion, this query would be classified as belonging to complexity level 1. When a query's SQL statement has two selection criteria, this query would be classified as belonging to complexity level 2. We expect that the higher the complexity level is for a query, the shorter processing time would be for a query because there is likely lesser output data for queries with more selection criteria. This approach has the shortcoming of not counting the magnitude of expected results (i.e., degree of restriction of the corresponding selection criteria). For example, a query with one very tight selection criterion would produce only small amounts of output. Another query with two very loose selection criteria would produce large amounts of output.

**Selectivity factor**

Selectivity factor is the expected proportion of tuples from a table to be selected for the output for a query. Selectivity factor for a query is defined as a continuous variable that
combines the cardinality of base relation(s) from which a query extracts information, the
domain(s) of attribute(s) that the query requests for information, and the degree of restriction of
selection criteria in the query. If a query has one selection criterion, the proportion of data that
would be extracted from an attribute's domain (data range) is the query's selectivity factor.
When a query has more than one selection criterion, the product of the proportions of data that
would be extracted from the corresponding attribute's domain for each selection criterion is the
query's selectivity factor. Table 3 provides examples of the calculation of queries' selectivity
factors.

<table>
<thead>
<tr>
<th>Partial SQL Statement</th>
<th>Attribute Domain</th>
<th>Selectivity Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHERE Field1 &lt; 4</td>
<td>Field1: [1, 30]</td>
<td>0.1</td>
</tr>
<tr>
<td>WHERE Field1 &gt; 6 AND</td>
<td>Field1: [1, 30] Field2: [1, 80]</td>
<td>0.8 × 0.25 = 0.2</td>
</tr>
<tr>
<td>Field2 ≤ 20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Examples of selectivity factors

Though selectivity factor is a continuous variable, we use five ranges to transform the
measure to a discrete form for use in our query complexity grouping process. When a query has
a selectivity factor that falls in a specific range (e.g. [0, 0.1]), this query would be classified as
belonging to a specific complexity level. A query with a low selectivity factor would be
classified to a low query complexity level and a query with a high selectivity factor to a high
query complexity level. We expect that the higher the complexity level is for a query, the longer
the processing time would be for a query because there is likely longer data processing and
output time for queries with higher proportion of data output. In addition, it is generally true that
the stronger the restrictions applied to the individual relations prior to carrying out the join, the
less expensive is the overall process [30].
One problem in using this factor to classify queries is the need to observe or assume the underlying distribution of queries’ selectivity factors in query population in order to set up ranges that have approximately the same number of queries in each complexity level. This is important because we do not want to have the majority of queries clustered in one or two ranges. For example, if 70% of queries have selectivity factors that fall in the range of \([0, 0.1]\), we would like to have sub-ranges (e.g. \([0, 0.001]\), \([0.001, 0.05]\), \([0.05, 0.1]\)) to represent different complexity levels. This is also important for us to get enough query samples in each query complexity level in order to identify the processing performance of a specific database structure with queries in different complexity level (with different selectivity factors). As a result, irregular intervals for ranges might be used (e.g. \([0, 0.0001]\), \([0.0001, 0.1]\), \([0.1, 0.4]\)). In short, there is no clear boundary for constructing complexity levels. We tried 5, 10, and 15 discrete intervals for query classification but the five-interval approach resulted in better alliance with our guidelines and objective as described above.

We now consider the experiment outcomes of applying for each of the four query complexity classification approaches to the 520 queries (52 query forms, 10 queries in each form).

**Analysis of classifying query complexity levels with “query type”**

Our data agree with what we expect on trend of query processing time among different complexity levels. We expect that the higher the query type complexity level is for a query, the longer processing time would be for that query. Generally, corresponding standard deviations of query processing time are consistently smaller than average query processing times. Using the "query type" classification approach would result in reasonable representations for query complexity levels. However, as mentioned earlier, a serious problem with this approach is that
when the same query is posed to different database structures, the complexity level of the query is often different by definition.

**Analysis of classifying query complexity levels with “number of attributes”**

Our data do not really agree with what we expect on trend of query processing time among different complexity levels. We expect that the higher the "number of attribute" complexity level be for a query, the longer processing time would be for that query. However, average query processing times do not show any consistent ascending or descending trend as the "number of attribute" complexity level goes up. In most occasions, corresponding standard deviations of query processing time are larger than average query processing times. In addition, as we mentioned earlier, another problem with this approach is that it does not count the size of attributes and the size of views for a query. Using the "number of attributes" classification approach apparently would not result in good representations for query complexity levels.

**Analysis of classifying query complexity levels using “number of selection criteria”**

Using number of selection criteria, our data agree with what we expect on trend of query processing time among different complexity levels. We expect that the higher the query type complexity level is for a query, the shorter processing time would be for that query (i.e. more selection criteria will result smaller size of query result). That is, there is a negative correlation between query complexity level and expected query processing time. Generally, corresponding standard deviations of query processing times are quite consistently smaller than average query processing times. Using the "number of selection criteria" classification approach would result in reasonable representations for query complexity levels. However, as we mentioned earlier, a big problem with this approach is that it does not consider the magnitude of expected result according to the selection criteria.
Analysis of classifying query complexity levels using “selectivity factor”

The data consistently agree with what we expect on trend of query processing time among different complexity levels. We expect that the higher the query type complexity level is for a query, the longer processing time would be for that query. Generally, corresponding standard deviations of query processing time are consistently smaller than average query processing times.

Table 4 summarizes the results for the four classification methods.

<table>
<thead>
<tr>
<th>Classification Scheme</th>
<th>Correlation Between Query Complexity Level and Expected Query Processing Time</th>
<th>Deviation among Expected Query Processing Time in a Query Complexity Level</th>
<th>Supporting Semantic Reasoning or Theory Background</th>
<th>Major Shortcoming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query Type</td>
<td>Positive</td>
<td>Small</td>
<td>Moderate</td>
<td>Derived complexity level becomes inconsistent when database structure alters</td>
</tr>
<tr>
<td>Number of Attributes</td>
<td>Inconclusive</td>
<td>Large</td>
<td>Weak</td>
<td>Does not count size of attributes and size of view for a query</td>
</tr>
<tr>
<td>Number of Selection Criteria</td>
<td>Negative</td>
<td>Small</td>
<td>Moderate</td>
<td>Does not count magnitude of expected query result</td>
</tr>
<tr>
<td>Selectivity Factor</td>
<td>Positive</td>
<td>Small</td>
<td>Strong</td>
<td>Does not have a clear boundary of different ranges</td>
</tr>
</tbody>
</table>

Table 4. Comparison of classification schemes for query complexity levels

In Table 4 desirable characteristics of classification schemes are bolded and underlined. The only scheme that satisfies all three criteria enumerated above is the “Selectivity Factor” scheme. First, when the query complexity level classified by the "selectivity factor" approach goes up, there is a consistent trend for the average query processing time to go up. Second, corresponding standard deviations of query processing time within each complexity level are
consistently small. Finally, using the "selectivity factor" approach takes the magnitude of expected results into account. When we pose a query to a different database structure, the complexity level of the query remains the same. In addition, as pointed out by [30], the selectivity factor is an important factor in evaluating join processing methods such as merging scan and nested loop. As a result, we conjecture that using the "selectivity factor" classification approach performs best for classifying query complexity levels.

We note that more sophisticated classification procedures can be used to achieve our objective. For instance, one can think of neural network approaches, machine learning, Logit regression, etc., where several query properties are considered simultaneously. We do not discard the applicability of such techniques to the problem of database structure selection. However, for this proof-of-concept exercise, we opted for simplicity, and for the specific application we investigated, grouping queries based on the selectivity factor alone, worked very well.

V. Database structure grouping

Based on correlation among different database structures, we investigated ways to classify database structures into groups. The reason for using correlation among different database structures is that we sought groupings where query processing will be very similar with database structures in the same group. We calculated correlation figures among the 96 possible database structures based on average processing times of different query complexity levels with the “selectivity factor” discussed in the previous section. We then grouped database structures together that have specific minimum correlation level among them. Table 5 presents the result of our grouping by using selectivity factor as classification base.
For example, when query complexity classifications were based on selectivity factor, requiring at least a 95% correlation coefficient over processing times across database structures in a group would require use of only 14 groups. Choosing a "best" representative from each group would reduce the problem from the original 96 options to 14 options.

The smaller the number of groups utilized, the smaller the dimensionality of the problems to be investigated. At the same time, we do not want to cluster a large number of database structures in only one or two big groups by using low correlation figures (e.g. 92.5% in Table 5) since this would result in less similarity among members in the same group. Using a correlation level of 0.95 provides satisfactory results in the sense of a balance with high within-group similarity in queries and a reasonable number of groups (14).

Let $p_{ij}$ represent processing time from query $i$ ($i = 1, \ldots, 520$) with database structure $j$ ($j = 1, \ldots, 96$). Also, let $S_k$ represent the set of databases structures in group $k$ ($k = 1, \ldots, 14$) according to the classification described above. We define the minimum (best) processing time for each query $i$, over all the database structures as following:

$$p_{ib} = \min_j P_{ij} \quad \forall \ i, j$$

We then select “representative” database structure from each of the 14 groups in the following ways:

i) for each group, select the representative database structure using the traditional optimization paradigm that involves selecting the database structure having the
minimum total processing time, i.e., for each group \( k \), choose a database structure \( j \in S_k \) according to the following criterion:

\[
\min_{j \in S_k} \sum_{i=1}^{520} p_{ij}
\]

Alternatively, under a robust optimization paradigm, the underlying objective would typically be one of the following two common alternatives:

(ii) choose the database structure that has the minimum of maximum deviations of processing times from the best processing time for each of the 520 queries; i.e., for each group \( k \), choose a database structure \( j \in S_k \) according to the following criterion:

\[
\min_{j \in S_k} \max_i (p_{ij} - \bar{p}_{ib})
\]

(iii) choose the database structure that minimizes the total deviations of processing times from best processing times across the 520 queries. The term “deviations” could refer to linear deviations or higher order (e.g., squared deviations) distance measures.

For example, for each group \( k \), the choice of a representative database structure \( j \in S_k \) can be made according to either of the following two objective functions:

\[
\min_{j \in S_k} \sum_{i=1}^{520} (p_{ij} - \bar{p}_{ib})
\]

or,

\[
\min_{j \in S_k} \sum_{i=1}^{520} (p_{ij} - \bar{p}_{ib})^2
\]

The basic idea of robust optimization is to shield against potentially significant impacts from extreme variations in query sets. While such extremes may be unlikely, their impact might significantly cripple the database operation. While robust solutions may not provide minimum
processing time and may not be the best solution for a particular query set, they are chosen since they work well across possible query sets.

We consider the traditional optimization first. For each group, the selected representative database structure was the one with the shortest total processing time (see (2) above). Since all database structures in the same group have 0.95 or higher correlation among them, the representative database structure should be a good illustration of how queries will be processed and how queries will behave with other database structures in a group.

To illustrate the application of the robust optimization approach, we used three objectives to select the “best representative” from each group: 1) minimize the maximum deviation (see (3) above), 2) minimize the sum of linear deviations (see (4a) above), and 3) minimize the sum of squared deviations (see (4b) above).

Table 6a and 6b provide the results of applying the traditional optimization objective and the results of applying each of three robust optimization alternatives.

These tables provide information on the following:

i) number of database structures in each group (Column 2 in Table 6a);

ii) the number of queries for which database structures within each group have fastest processing time (Column 3 in Table 6a);

iii) identification of representative database structure (Column 4 to 7 in Table 6a):

1) approach 1 selects a representative database structure which has the shortest total processing time
2) approach 2 selects a representative database structure which has a minimal maximum deviation from best processing time
3) approach 3 selects a representative database structure which has a minimal sum of linear deviations from best processing times
4) approach 4 selects a representative database structure which has a minimal sum of squared deviations from best processing times;

iv) for each representative database structure, the number of queries (of the 520 total) for which it has fastest processing time (Column 3 in Table 6b);
v) total processing time of 520 queries for the representative database structures (Column 4 in Table 6b); and,

vi) overall rank (For example, T1 has the shortest total processing time for 520 queries in the simulation among all possible database structures and T1 has the rank as 1.) for the representative database among 96 database structures (Column 5 in Table 6b).
<table>
<thead>
<tr>
<th>Group #</th>
<th>Number of database structures in the group</th>
<th>Number of queries which have shortest processing times with any database structure in the group</th>
<th>Approach 1</th>
<th>Approach 2</th>
<th>Approach 3</th>
<th>Approach 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>44</td>
<td>103</td>
<td>T12,45</td>
<td>T12,45</td>
<td>T12,45</td>
<td>T12,45</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>306</td>
<td>T1</td>
<td>T57</td>
<td>T1</td>
<td>T1</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
<td>45</td>
<td>T457</td>
<td>T126,457</td>
<td>T457</td>
<td>T123,457</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>2</td>
<td>T23,1567</td>
<td>T23,1567</td>
<td>T23,1567</td>
<td>T23,1567</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>56</td>
<td>T45</td>
<td>T45</td>
<td>T45</td>
<td>T45</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>0</td>
<td>T45,1237</td>
<td>T45,1237</td>
<td>T45,1237</td>
<td>T45,1237</td>
</tr>
<tr>
<td>7</td>
<td>4</td>
<td>8</td>
<td>T17,45</td>
<td>T17,45</td>
<td>T17,45</td>
<td>T17,45</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>T12345</td>
<td>T12345</td>
<td>T12345</td>
<td>T12345</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0</td>
<td>T45,12367</td>
<td>T45,12367</td>
<td>T45,12367</td>
<td>T45,12367</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>0</td>
<td>T123457</td>
<td>T123457</td>
<td>T123457</td>
<td>T123457</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>T12467</td>
<td>T12467</td>
<td>T12467</td>
<td>T12467</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0</td>
<td>T124567</td>
<td>T124567</td>
<td>T124567</td>
<td>T124567</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0</td>
<td>T1234567</td>
<td>T1234567</td>
<td>T1234567</td>
<td>T1234567</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0</td>
<td>T123467</td>
<td>T123467</td>
<td>T123467</td>
<td>T123467</td>
</tr>
</tbody>
</table>

Table 6a. Database structure groups and representative database structures

<table>
<thead>
<tr>
<th>Group #</th>
<th>Representative database structure</th>
<th>Number of queries which have shortest processing time with this database structure</th>
<th>Total processing time of 520 queries for the representative structure in the group (hour)</th>
<th>Overall rank for the representative structure in the group among 96 structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T12,45</td>
<td>15</td>
<td>0.717</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>T1</td>
<td>142</td>
<td>0.514</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>T57</td>
<td>35</td>
<td>0.638</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>T457</td>
<td>11</td>
<td>0.829</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>T126,457</td>
<td>6</td>
<td>0.973</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>T123,457</td>
<td>3</td>
<td>0.962</td>
<td>23</td>
</tr>
<tr>
<td>7</td>
<td>T23,1567</td>
<td>1</td>
<td>1.438</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>T45</td>
<td>39</td>
<td>0.592</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>T45,1237</td>
<td>0</td>
<td>2.154</td>
<td>69</td>
</tr>
<tr>
<td>10</td>
<td>T17,45</td>
<td>8</td>
<td>0.922</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>T12345</td>
<td>0</td>
<td>2.377</td>
<td>71</td>
</tr>
<tr>
<td>12</td>
<td>T45,12367</td>
<td>0</td>
<td>5.131</td>
<td>84</td>
</tr>
<tr>
<td>13</td>
<td>T123457</td>
<td>0</td>
<td>13.062</td>
<td>92</td>
</tr>
<tr>
<td>14</td>
<td>T12467</td>
<td>0</td>
<td>13.901</td>
<td>93</td>
</tr>
<tr>
<td>15</td>
<td>T1234567</td>
<td>0</td>
<td>16.266</td>
<td>94</td>
</tr>
<tr>
<td>16</td>
<td>T123467</td>
<td>0</td>
<td>42.568</td>
<td>95</td>
</tr>
<tr>
<td>17</td>
<td>T123467</td>
<td>0</td>
<td>86.369</td>
<td>96</td>
</tr>
</tbody>
</table>

Table 6b. Comparison of representative database structures
VI. Robust database structure(s) identification/selection

We first took a further look at query processing times for all 520 queries processed with the representative database structure of 14 groups. We identified the shortest query processing time for each of the 520 queries among 14 representative database structures. We called this shortest query processing time the "Best Performing Database Structure Processing Time" (BPDSPT) (defined in Section III in detail).

For each database structure we calculated the extra processing time for that structure, that is, the amount above the BPDSPT that the individual structure took to process the indicated query set. These time differences are called "deviation from BPDSPT". For each of five query complexity levels, we calculated average extra time (i.e., average deviation from BPDSPT) for each representative database structure. The result was called "mean deviation from BPDSPT". We also calculated squared extra times and called them "squared deviation from BPDSPT". We present these results in Table 7 and Table 8.
## Mean Deviation from BPDSPT

<table>
<thead>
<tr>
<th>Groups</th>
<th>Representative database structure for the group</th>
<th>Query Complexity Level</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Level 1</td>
</tr>
<tr>
<td>G1 (103)†</td>
<td>T12,45 (15)‡</td>
<td>0.45</td>
</tr>
<tr>
<td>G2 (306)</td>
<td>T1 (142)</td>
<td>0.15*</td>
</tr>
<tr>
<td></td>
<td>T57 (35)</td>
<td>0.34</td>
</tr>
<tr>
<td>G3 (45)</td>
<td>T457 (11)</td>
<td>3.13</td>
</tr>
<tr>
<td></td>
<td>T126,457 (6)</td>
<td>4.64</td>
</tr>
<tr>
<td></td>
<td>T123,457 (3)</td>
<td>3.41</td>
</tr>
<tr>
<td>G4 (2)</td>
<td>T23,1567 (1)</td>
<td>3.60</td>
</tr>
<tr>
<td>G5 (56)</td>
<td>T45 (39)</td>
<td>0.32</td>
</tr>
<tr>
<td>G6 (0)</td>
<td>T45,1237 (0)</td>
<td>9.85</td>
</tr>
<tr>
<td>G7 (8)</td>
<td>T17,45 (8)</td>
<td>0.61</td>
</tr>
<tr>
<td>G8 (0)</td>
<td>T1234 (0)</td>
<td>10.26</td>
</tr>
<tr>
<td>G9 (0)</td>
<td>T45,12367 (0)</td>
<td>105.62</td>
</tr>
<tr>
<td>G10 (0)</td>
<td>T123457 (0)</td>
<td>122.75</td>
</tr>
<tr>
<td>G11 (0)</td>
<td>T12467 (0)</td>
<td>91.99</td>
</tr>
<tr>
<td>G12 (0)</td>
<td>T124567 (0)</td>
<td>100.11</td>
</tr>
<tr>
<td>G13 (0)</td>
<td>T1234567 (0)</td>
<td>281.68</td>
</tr>
<tr>
<td>G14 (0)</td>
<td>T123467 (0)</td>
<td>309.13</td>
</tr>
</tbody>
</table>

**Note:**

† The number in the parentheses with each group name is the total number of queries (out of 520 queries) that have shortest processing time with any database structure in that group.

‡ The number in the parentheses with each representative database structure name is the total number of queries (out of 520 queries) that have shortest processing time with the representative database structure in that group.

* A number in bold shows that a representative database structure has minimal mean deviation among all representative database structures.

Table 7. Mean deviation from BPDSPT for all representative database structures (in seconds)
<table>
<thead>
<tr>
<th>Groups</th>
<th>Representative database structure for the group</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T1 (142)</td>
<td>0.07*</td>
<td>0.10</td>
<td>277.23</td>
<td>40.57</td>
<td>3123.37</td>
</tr>
<tr>
<td></td>
<td>T57 (35)</td>
<td>0.22</td>
<td>0.41</td>
<td>296.00</td>
<td>48.28</td>
<td>3236.06</td>
</tr>
<tr>
<td></td>
<td>T457 (35)</td>
<td>0.43</td>
<td>0.64</td>
<td>0.68</td>
<td>376.36</td>
<td></td>
</tr>
<tr>
<td></td>
<td>T162,457 (6)</td>
<td>28.26</td>
<td>48.09</td>
<td>359.83</td>
<td>760.20</td>
<td>3654.72</td>
</tr>
<tr>
<td></td>
<td>T123,457 (3)</td>
<td>17.98</td>
<td>35.36</td>
<td>45.04</td>
<td>87.87</td>
<td>639.94</td>
</tr>
<tr>
<td></td>
<td>T23,1576 (1)</td>
<td>16.32</td>
<td>18.95</td>
<td>22.92</td>
<td>50.44</td>
<td>1268.31</td>
</tr>
<tr>
<td>G2 (306)</td>
<td>T45 (39)</td>
<td>0.26</td>
<td>0.46</td>
<td>0.57</td>
<td>9.44</td>
<td>413.90</td>
</tr>
<tr>
<td>G3 (45)</td>
<td>T4512,37 (0)</td>
<td>113.19</td>
<td>186.60</td>
<td>322.21</td>
<td>377.59</td>
<td>972.56</td>
</tr>
<tr>
<td>G4 (2)</td>
<td>T17,45 (8)</td>
<td>0.59</td>
<td>1.77</td>
<td>5.67</td>
<td>38.24</td>
<td>813.12</td>
</tr>
<tr>
<td>G5 (56)</td>
<td>T12345 (0)</td>
<td>111.93</td>
<td>156.70</td>
<td>125.22</td>
<td>167.52</td>
<td>497.24</td>
</tr>
<tr>
<td>G6 (0)</td>
<td>T12345 (0)</td>
<td>3050.1135</td>
<td>4015.96</td>
<td>3334.39</td>
<td>4375.51</td>
<td>9111.37</td>
</tr>
<tr>
<td>G10 (0)</td>
<td>T123457 (0)</td>
<td>37273.01</td>
<td>61518.93</td>
<td>41052.49</td>
<td>35287.26</td>
<td>58198.80</td>
</tr>
<tr>
<td>G11 (0)</td>
<td>T12467 (0)</td>
<td>38944.88</td>
<td>53202.07</td>
<td>39939.52</td>
<td>326438.84</td>
<td>114113.00</td>
</tr>
<tr>
<td>G12 (0)</td>
<td>T124567 (0)</td>
<td>20751.78</td>
<td>34868.38</td>
<td>28613.70</td>
<td>74179.05</td>
<td>55633.27</td>
</tr>
<tr>
<td>G13 (0)</td>
<td>T1234567 (0)</td>
<td>122474.49</td>
<td>443087.76</td>
<td>359018.31</td>
<td>377818.02</td>
<td>177372.19</td>
</tr>
<tr>
<td>G14 (0)</td>
<td>T123467 (0)</td>
<td>465481.20</td>
<td>1761559.34</td>
<td>983117.43</td>
<td>3355577.28</td>
<td>11298233.93</td>
</tr>
</tbody>
</table>

Note:
† The number in the parentheses with each group name is the total number of queries (out of 520 queries) that have shortest processing time with any database structure in that group.
‡ The number in the parentheses with each representative database structure name is the total number of queries (out of 520 queries) that have shortest processing time with the representative database structure in that group.
* A number in bold shows that a representative database structure has minimal mean squared deviation among all representative database structures.

Table 8. Mean squared deviation from BPDSPT for all representative database structures (in seconds)

From Table 7, we see that T1 and T45 dominate other representative database structures; that is, T1 dominates in the first two complexity levels and T45 dominates in the last three complexity levels. From Table 8, we see that T12,45, T1, and T45 dominate other representative database structures; that is, T1 dominates in the first two complexity levels, T45 dominates in the third complexity level, and T12,45 dominates in the last two complexity levels. It took longer, in
general and on average, for a group of queries to be processed on a database structure other than T12,45, T1, or T45. In other words, based on our observations, it would not be beneficial to process queries with any database structure other than T12,45, T1, or T45 if we would like to obtain fast query processing time.

Our experimental results suggest that we should build a specific database structure for a database system based on the types and frequency of queries that the database system will process. For example, T1 is the efficient database structure to use when majority of queries processed in a database system is in the first two complexity levels. T45 is the efficient database structure to use when majority of queries processed in a database system is in the third complexity level.

Consider the situation when queries to be processed are evenly distributed across complexity levels. Database structure T45, and not T1 (the normalized database structure), is the best performer. That is, following the conventional wisdom of use of the normalized database design fails to yield the best result even in this case (since normalization is not for query processing but for minimizing data redundancy). In a more skewed query set situation, the relative performance of the normalized database structure is even worst.

Our results also suggest that it may be risky to use only one database structure because no single dominant database structure performs the best for all different types of queries. At the same time, maintaining all possible database structures or too many database structures is not likely beneficial because most queries get satisfactory processing speed with only selected database structures. Therefore, we recommend maintaining only selected robust database structures identified by using our approach.
VII. Conclusion

We began this paper by emphasizing that the success of a company increasingly depends on timely information (internal or external) being available to the right person at the right time. Achieving such a “right time/right place” duet depends directly on database performance. A database system has been a core component that supports modern business system such as enterprise resource planning (ERP) system that integrates and supports all enterprise processes including product designing and engineering, manufacturing, and other business functions to achieve highest efficiency and effectiveness of operations. The task is complicated by these dynamic activities that have become an integral part of a firm’s operations. Our purpose here is to set forth database design procedures that identify robust database structures that perform well across diverse and uncertain queries, i.e., situations such as that just described.

Delivering “right time/right place” accurate data is a challenge. It is that challenge that our work begins to address by developing and implementing a new “query-driven” heuristics for database design that seeks to identify database structures that perform robustly in dynamic settings. Our heuristics begins with detailed analysis of relationships between diverse query sets and the performance of different database structures. These relationships are then used in a series of steps that identify robust database structures that maintain high performance levels for a wide range of query sets.

Adapting Wiederhold’s six-step process model [44], we can summarize our procedures as follows:

1) Specify the desired behavior of an object.
   Design a database system that matches query-processing need and provides timely query responses to assist crucial decision-making by explicitly considering queries and query properties.
2) **Select reasonable building blocks and tools.**
   Develop different query classification schemes and database structure grouping methodologies.

3) **Use the model to evaluate which combinations produce the desired result.**
   Set guidelines to select the best query classification approach (Section IV) and apply correlation analysis to group database structures (Section V).

4) **Select the most effective combination of building blocks and tools.**
   Identify "selectivity factor" approach as the most reasonable approach to classify queries into different complexity levels and identify "0.95 correlation level among database structure using the selectivity factor classification" method as the most reasonable way to group database structures.

5) **Build it as well as possible.**
   Identify and build robust efficient database structure(s) for a database system that can timely process queries with a wide range of complexity.

6) **Observe and monitor its behavior.**
   Continuously observe and monitor query changes and database performance to seek improvement and modifications.

We are performing measurement experiments that serve as inputs into our analysis and identification of robust database structures. It is important to point out that we can also use this experimental, measure-based approach to identify which views to materialize. As described in [46], the VM problem is:

- given a set of known queries;
- given a set of possible views to materialize;
- given well defined processing costs and maintenance costs;
- select the views to materialize to minimize total costs.

Instead, because of our focus on highly dynamic environment, we don’t start with a given set of queries and well-defined costs. We look at a much wider universe of queries and directly conduct cost measurements through extensive ex ante experiments. Similarly to [46] we look at materializing views that include JOINS. In the approach we have begun here we (1) attempt to look at query complexity classes that span the possible set of queries that pertain to an
information context; (2) consider all feasible materializations of table JOINS in that information context; and, (3) instead of assuming cost structures for the processing of queries, we begin by directly measuring them experimentally.

While we have emphasized the detailed measuring of alternative processing costs, we also realize the importance of delving further to incorporate costs of maintenance and restructuring. In fact, our ongoing work includes the formal modeling and analysis of such critical cost relationships beyond assumption of [22] and [45] that the cost of answering a query can be represented by the number of rows present in the table that used to construct the query.

The results in this paper provide the initial steps necessary to develop a dynamic database system capable of effectively accommodating significant query set shifts. This suggests the next research question on which we are currently working, that is, can we develop intelligent learning tools that identify important shifts in query sets and identify opportunities where a firm can gain significantly by altering the maintained data structure (i.e., changing the maintained database design)? A dynamic database system should be able to detect changes in complexity of query sets and adopt a new database structure that is most efficient for the new query-processing needs.

Alternatively, would maintaining a set of parallel database structures be preferable? Would there be net gains from assigning arriving queries to the best of the maintained structures? In either case, analysis of options would include query-driven analysis as presented here. Choice of implementation mechanism would depend on several factors, including implementation context (web, intranet, etc.) and implementation/operating costs. Our current research involves the development of a dynamic database restructuring approach that includes and relies upon formal modeling of the critical cost relationships.
References:


