Open Issues in Semantic Query Optimization in relational DBMS

Bryan Genet, Annika Hinze
Department of Computer Science
University of Waikato, New Zealand
{b.genet,hinze}@cs.waikato.ac.nz

Abstract

After two decades of research into Semantic Query Optimization (SQO) there is clear agreement as to the efficacy of SQO. However, although there are some experimental implementations there are still no commercial implementations. We first present a thorough analysis of research into SQO. We identify three problems which inhibit the effective use of SQO in Relational Database Management Systems (RDBMS). We then propose solutions to these problems and describe first steps towards the implementation of an effective semantic query optimizer for relational databases.

Contents

1 Introduction

After two decades of research into Semantic Query Optimization (SQO) there is still no commercial implementation. Despite this, there is clear agreement in the literature as to the worthwhileness of SQO [YHPM99, Dat95, HK94]. We make the observation from our experience in the commercial database industry that, in the case of relational database management systems (RDBMS), SQO is neither mainstream nor widely employed. Other writers report similar observations [CGK$^+$99, GGXZ01] and note the emphasis of much research has been deductive databases (DD). In this paper we specifically address the role of SQO in relational databases.

We identify three problems:

1. In the context of RDBMS, previous research does not make clear the unique role that database integrity constraints[1] can play in SQO. In addition, the close relationship between SQO and the enforcement of database integrity constraints at query time has not been established.

2. While many writers consider SQO in the context of deductive databases (DD), very little work specifically considers the efficacy of SQO for relational databases (RDB) and even less describe actual implementations.

3. Previous research does not make clear under what conditions it is likely to be worthwhile implementing SQO or what types of SQO are likely to be effective for a given database schema.

In this paper we focus on each of the above problems in turn, first describing each problem against the background of previous research. We then propose solutions to the above problems. Our solutions emphasize both the increased potential of SQO in current database environments and the importance of considering all available sources of semantic information when implementing SQO. Our approach follows that of Godfrey *et al* [GGM96, GGGM98, GGXZ01] and we explore some of their results in the specific context of RDBMS.

In this paper, the context of any examples we use is RDBMS. We introduce the remainder of the paper with a motivating example in Section 2. The remaining sections are organized as follows.

In Section 3 we describe related research in the field of SQO. Our description is built around the definitions of some important terms used by other researchers. Our goal is to clarify and to some extent simplify some of this terminology. We extend this terminology by introducing some new terms. We define some important preliminary

[1]We use this term for now because it is used more generally in the literature. However, in Definition 3.9 we introduce the term *schema constraint*. Schema constraints subsume integrity constraints.

terms before defining the central term *semantic query optimization*. We then consider SQO in the context of commercial RDBMS and report some of the reasons suggested by other researchers as to why SQO is not routinely employed. We conclude this section by noting some important changes to hardware typically employed in the database industry.

In Section 4 we look in more detail at the three problems identified above. We begin by showing that it is often unjustified to assume that data actually stored in the database conforms to known database semantics. We then describe the benefits that accrue from enforcing schema constraints at query time. We argue that it is extremely unlikely that utilizing database integrity constraints for SQO will result in exponentially increasing numbers of candidate queries that are semantically equivalent[2]. In the context of RDBMS, we question the common assumption that effective SQO requires computationally expensive mining[3] of data to discover complex semantic rules. We note that, for a given database schema, attempting to exploit SQO without knowing what types of queries are actually made on the database, may result in little or no query optimization.

In Section 5 we describe some preliminary results of our research to address the problems described in Section 4. We argue that the enforcement of schema constraints at query time adds a data robustness that significantly enhances the validity of information derived from the database. We then sketch how a simple but effective semantic query optimizer can readily be constructed from schema constraints alone. We highlight the ubiquity of rich semantic information and the relative ease with which an effective semantic query optimizer can be constructed. In the case of RDBMS, we describe how an effective semantic query optimizer may be built from schema constraints alone. We reiterate a key observation that there is no semantic reasoning engine built in to SQL optimizers.

In Section 6 we summarize the main contributions of this paper.

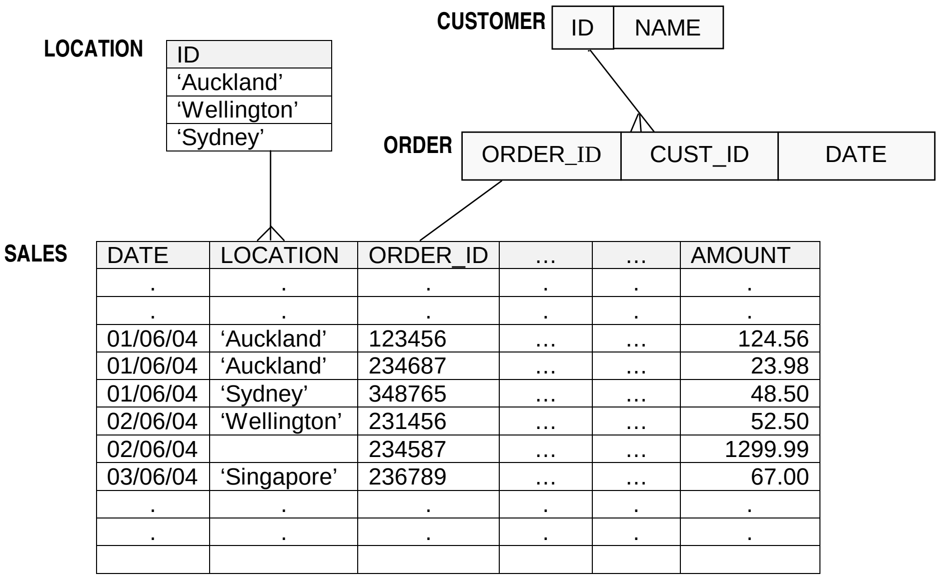
2 Motivating Example

Refer to Figure 1 which shows a simplified fragment of a Customer Order schema. Three tables from the transactional database are depicted: `LOCATION`, `CUSTOMER` and `ORDER`.

- The `LOCATION` table is a reference table whose primary key (PK) is the unique `ID` if each company Office.
- The `CUSTOMER` table contains the unique `ID` of each Customer (its PK), along with various Customer details such as `NAME`.
- The `ORDER` table contains header information for each customer order. Its PK is `ORDER_ID`. In addition a foreign key (FK) points back to the Customer `ID` in the `CUSTOMER` table.

In addition to the above transactional tables, Sales information is periodically summarized in the `SALES` table. Columns in this table include the FKs `LOCATION` and `ORDER_ID` which point back to tables `LOCATION` and `ORDER` respectively.

[2]See Definition 3.1 for a precise definition of *semantic equivalence*.
[3]We use this term in the sense of data mining.



ID
‘Auckland’
‘Wellington’
‘Sydney’

ID	NAME

ORDER_ID	CUST_ID	DATE

DATE	LOCATION	ORDER_ID	…	…	AMOUNT
.	.	.	.	.	.
.	.	.	.	.	.
01/06/04	‘Auckland’	123456	…	…	124.56
01/06/04	‘Auckland’	234687	…	…	23.98
01/06/04	‘Sydney’	348765	…	…	48.50
02/06/04	‘Wellington’	231456	…	…	52.50
02/06/04		234587	…	…	1299.99
03/06/04	‘Singapore’	236789	…	…	67.00
.	.	.	.	.	.
.	.	.	.	.	.

Figure 1: *A simplified fragment of a Customer Order schema. Three tables from the transactional database are depicted: `LOCATION`, `CUSTOMER` and `ORDER`. In addition to the above transactional tables, Sales information is periodically summarized in the `SALES` table.*

An Employee prepares a simple report showing the Total Sales for the company during the previous month. The company has offices in `Auckland`, `Wellington` and `Sydney` so sales figures are fetched for these three locations, as well as the total sales. On receiving the report, the Manager points out that the total sales exceed the sum of the sales for the three offices. Investigation reveals that the Database Administrator (DBA) has disabled the check constraint on the `LOCATION` column to speed data insertion during a batch operation to populate the `SALES` table. Further investigation reveals that data from the company's new office in `Singapore` has already been inserted while other sales do not have their location recorded.

We conclude this section by summarizing the main point of our example. Our example illustrates how easily the assumed schema semantics can differ from the actual semantics of the data. In particular, it is often quite unjustified to assume that database constraints have been enforced. This is considered in detail in Section 4.1.

3 Related Work

In this section we consider the work of other researchers in the field of SQO. We first consider this research in the context of the some important terminology which we clarify and then employ to describe our own contribution. We then briefly consider SQO in the context of commercial RDBMS.

3.1 Definition of important terms

We now describe related research in the field of SQO by considering the definitions of some important terms used in the research literature. Our goal is to clarify and to some extent simplify some of this terminology. We extend this terminology by introducing some new terms. We will define the central term *semantic query optimization* in due course, but in order to do so we first define some preliminary terms.

Definition 3.1 ***Semantic equivalence****: Two database queries are* semantically equivalent *if they return the same answer, for a given database state.*

The aim of semantic optimization is to use semantic knowledge about the database domain to transform a query into one which is semantically equivalent but which can be executed more efficiently[4] [CGM90, GGM96, HZ80, Kin81]. Our definition of semantic equivalence follows [SSS92].

Note that the queries in question cannot meaningfully be considered equivalent *per se* but must be semantically equivalent *with respect to some specific set of schema semantics*. So it is more correct to say that *semantic equivalence* means the transformed query always produces the same answer as the original query for any database state *satisfying a given set of constraints* [SSS92].

For RDBMS, semantic equivalence means, literally, that the result sets returned by the two queries are exactly the same, although the tuples are not necessarily presented in the same order. Since the contents of a database will typically change over time, whenever we speak of the *answer* to a query we are referring to the tuples returned at a particular point in time for a particular database state.

Definition 3.2 ***Query rewrite****: This is the process of recasting a database query into a semantically equivalent query with a different syntax.*

This notion is intrinsic to SQO and we use this term in the same way as other researchers in the field [AF94, Kin81, HZ80]. [5]

Definition 3.3 ***SQL optimizer****: This is the engine that takes as its input the SQL query text and outputs the specific execution path to be followed in order to compute the query answer.*

All SQL language interpreters require an optimizer because SQL is a declarative language [Dat95]. A critical feature of the optimizer is that it calculates a metric[6] which is a measure of how costly the query is expected to be, without actually executing the query. This metric is readily available in all commercial RDBMS. This makes it straightforward to *estimate* the relative computational cost of semantically equivalent queries, without actually executing the queries many times and comparing average query times.

Definition 3.4 ***Query cost****: The* cost *of a query is the expenditure of computing resources required to answer that query.*

This term is used very generally by researchers, hence the generic definition. Various writers emphasize the amount of disk activity required to answer a query [SSS92],

[4]See Definition 3.5 for a precise definition of *query efficiency*.

[5]Note that the term *syntax* here refers to the actual textual expression of the query statement. We do not mean the abstract syntax of the query language e.g. SQL.

[6]typically dimensionless

the total time required to answer the query [Zhu92] and delays due to communication costs [JK84].

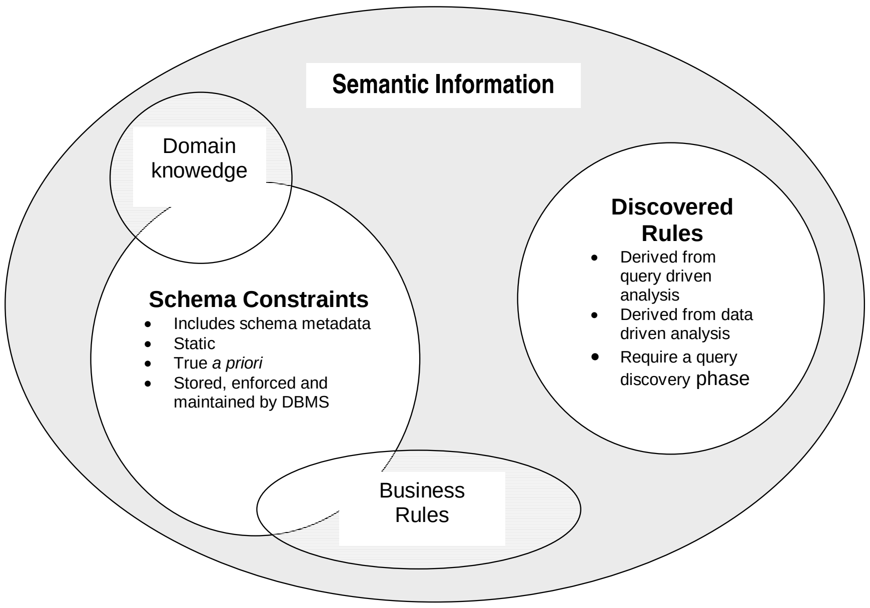
In this paper, query cost specifically refers to *the average time required to answer that query*. This is the only criterion we use to judge query cost. However, *relative* query cost may be *estimated* by examining the cost metric returned by the SQL optimizer[7].

Definition 3.5 ***Query efficiency****: This is the relative cost of two semantically equivalent queries.*

Intuitively, a more efficient query is one that on average requires fewer resources to execute (e.g. fewer disk reads) and therefore is executed more quickly [GG96], given a particular set of computer resources.

One can only meaningfully compare the efficiencies of two *semantically equivalent* queries. Consider two semantically equivalent queries Q1 and Q2. Query Q1 is *more efficient* than query Q2 if it takes on average less time to answer.

We now consider definitions arising from the notion of *semantic information*. We begin with a very general definition which we refine as we consider different sources of semantic information. Refer to Figure 2 for clarification.



Semantic Information
Domain knowedge
Schema Constraints
Includes schema metadata
Static
True *a priori*
Stored, enforced and maintained by DBMS
Discovered Rules
Derived from query driven analysis
Derived from data driven analysis
Require a query discovery phase
Business Rules

Figure 2: *Semantic information can be drawn from a number of different sources including schema constraints and discovered rules. Schema constraints originate from human practioners, while discovered rules are found by the execution of software.*

Definition 3.6 ***Semantic information****: This is any logical statement or data which* describes *or* constrains *the data currently stored in the database and the data that may be stored in the database.*

[7]See Definition 3.3

Semantic information includes schema meta-data (such as the table and view definitions in a relational database), domain knowledge (such as might be held by *domain experts*) as well as various constraints defined, stored and enforced by the database management system (DBMS). A prerequisite for query rewriting is obtaining *valid* semantic information i.e. semantic information which is true of the target database at this particular point in time.

Definition 3.7 ***Semantic rule*** *A semantic rule is a sentence in first order predicate calculus which expresses semantic information.*

This notion is intrinsic to SQO. The nuance of the term *rule* in this context is that the semantic information is captured by a formal logical sentence. Such logical sentences may be utilized by a reasoning system. In contrast, the terms *business rules* and *domain knowledge* do not necessarily refer to formal logical sentences and may include the informal knowledge of *domain experts*. Such knowledge may be difficult or impossible to express in first order predicate calculus [GGM96].

Definition 3.8 ***Rule relevance****: A semantic rule is* relevant *if it is able to be utilized by the semantic query optimizer to increase query efficiency.*

Many semantic rules may be discovered in a mechanical knowledge discovery process which are of no practical value to the semantic query optimizer [AF94, CGK$^+$99, GGXZ01]. For example, no matter how ”strong” (in some sense) the rule discovered, it will nevertheless have no impact whatsoever if the rule is never actually invoked.

Example 3.1 *A knowledge discovery exercise identifies a strong correlation between an employee's* bank*, the* make of their car *and the* gender of their manager*. Such a correlation might be uncovered by a mechanical analysis which carries out an exhaustive search for such relationships. Yet this semantic knowledge, although valid, is of little practical value. A cursory examination of the actual query profile reveals that queries incorporating the three employee attributes:* bank*,* make of car *and* gender of manager *are never actually made. So the (possibly expensive) knowledge discovery exercise is of no value.*

In this paper we specifically identify a number of important sources of semantic rules. We classify these semantic rules firstly by observing whether or not they may be formally encoded within the database as *constraints*.

Definition 3.9 ***Schema constraint****: A schema constraint is a rule which is stored, maintained and enforced by the DBMS and which constrains the legal values that may be stored in the database.*

The critical role of schema constraints as a rich and stable source of semantic information is considered by [YS89] and [GGXZ01] in the context of knowledge discovery and deductive databases respectively. In RDBMS, schema constraints therefore consist exactly of the schema meta-data plus the *integrity constraints* [8] plus user defined constraints [9] defined as part of the particular RDBMS schema.

Various writers have identified other sources of semantic information (which may or may not be expressible in the language of the particular DBMS) such as specialized domain knowledge held by domain experts [GGXZ01] and application business

[8]For example *not null, primary key, foreign key, check*. We use this term in the conventional way [Dat95].

[9]In RDBMS, complex user defined constraints (possibly arising from the implementation of business rules) may be defined as *triggers* or implemented procedurally.

rules [GGM96, GD98, Dat95, SHKC93, SO87]. In this paper we assume that such semantic information is true *a priori* and that it is unchanging. Such an environment is quite realistic as in the case of a data warehouse.

We add the important observation that schema constraints are intended to constrain data only at insert or modification time and are not utilized at query time. Furthermore, while schema constraints remain valid statements about the domain of interest, this is no guarantee that the actual data stored in the database conforms to these rules. This contradictory situation arises with great regularity in commercial DBMS because constraints are often relaxed during data insert or modification. This is considered in more detail in Section 4.

Some writers classify rules considered for semantic optimization as *static* or *dynamic* [IBG94, CGM90]. Included in static constraints are such rules that do not change or evolve over time as the state of the database changes. Therefore schema constraints are *static*.

Some authors specifically concentrate on rules that are derived from the database [Che96, SSS92]. These studies use a variety of techniques to detect correlations in data which are then used to formulate rules.

Definition 3.10 ***Rule discovery****: Rule discovery is the search for patterns, regularities and correlations in the target database, utilizing every available information source.*

Our definition follows [GSZZ01, HK98, SSS92, SHKC93], but for clarity specifically refers to the uncovering of semantic information which is *additional* to the schema constraints described above. [SHKC93] use the term rule discovery *phase* to emphasize the uncovering of previously unknown information.

All the researchers in this field identify two broad rule categories: static and dynamic. (See for example [IBG94, CGM90]). Static rules do not change over time i.e. we may assume their validity for the lifetime of the schema. Therefore schema constraints are *static*. When discovered rules are static, they need only be compiled once [CGM90].

However, *dynamic* rules may change as the database state changes. In particular, dynamic rules may be rendered invalid by updates to the database. When dynamic rules are used, the cost of checking to see if the rules are still valid after a database update must be taken into account. We consider the question of revalidating temporary or dynamic constraints as quite separate from the central issue of SQO.

Rule discovery is typically *query driven* or *data driven* [YS89, SSS92, SHKC93]. In query driven rule discovery, rules are inferred from the restriction clauses of queries arriving at the database and the results they produce. In its simplest form, the method notes when two syntactically different queries produce the same answer. The more efficient query will then be substituted whenever appropriate.[10] Such processing may incur performance penalties at run time. Another more subtle problem is that the rules produced may only optimize queries which are the same (or similar to) previous ones received and analyzed. This leaves many potential semantic optimizations unexplored [SHKC93].[11]

[10]A trivial example of detecting semantic equivalence is the case of three queries, identical except that one is uppercase, another is lower case while the third is a mixture of cases. The SQL optimizer of at least one major commercial RDBMS does no textual reformatting such as moving to upper case or white space compression. All three queries are judged as being different from one another and each will be separately parsed.

[11]It is a truism in the real world of RDBMS management that 90% of query activity is based around 10% of the tables. A more extreme real world example can be found in data warehouses where one large aggregated table is the target of all queries. In these contexts query driven rule discovery can be effective.

In data driven rule discovery, we look primarily at data distribution [SHKC93]. Data is analyzed off line in order to discover patterns or correlations that may be formulated into semantic rules. A trivial example is the discovery that all values of a particular table column are null. Being independent of any queries, rules may be compiled incrementally without affecting run time performance. Examples include inference rules derived from subsets of data which have typically been clustered or partitioned in some way.

We note that data driven rule discovery is in many respects identical to the traditional tasks undertaken by the Database Administrator (DBA) with respect to data reorganization. For example, inspection of data distribution may yield candidate columns for indexing.

When a large rule set exists which may potentially be used to semantically optimize a query, the problem arises as to which ones are the best to use. The number of transformations suggested by a semantic optimizer can quickly become combinatorially explosive [SSS92]. Most writers suggest the use of heuristics to guide the choice of rules and to prune the number of possible transformations to the best ones [Kin81, SO87, SSS92, SSD92].

[GGXZ01] differentiate between rules which are assumed to be always true and have been defined using existing DBMS mechanisms (for example RDBMS *check* constraints) and *soft constraints* (SC) which are discovered rules assumed to be true ”most” of the time. A SC may be an *absolute soft constraint* meaning the current state of the database contains no data that violates the rule and *statistical soft constraints* to which a majority of the data comply and a small exception do not. Therefore we have a three level hierarchy defined by rule’s reliability i.e. the probability that it is valid at a particular time:

1. Schema constraints: static, true for all data, rule applies for all time;
2. Absolute soft constraints: dynamic, true for all data, rule applies at this particular time only;
3. Statistical soft constraints: dynamic, true for most data, rule applies at this particular time only.

SCs are therefore temporary constraints which we would classify along with other dynamic rules.

[HK98] pursue a similar theme of the reliability of a discovered rule. They claim that while most approaches to SQO assume that database semantics are static, in practice they are dynamic. They propose a quantitative metric called *robustness* which is the probability that a discovered rule is consistent with a database state. A rule has high robustness if it is unlikely to become inconsistent after database updates. Robustness of a rule may be estimated from readily available DBMS meta-data. Only rules with high robustness are used for semantic optimization, thereby limiting the cost of re-validating rules.

One intrinsic property of all schema constraints is that they are created and encoded into the database by human practioners. It is reasonable then to assume such rules are robust. In contrast, discovered rules are found by the execution of software and we cannot in general assume their robustness.

We summarize three important approaches to the discovery of semantic information in Figure 3 where we make explicit the rich contribution of schema constraints to discovery of useful semantic information.

We can now define SQO. The next three definitions use Figure 4 for clarification.

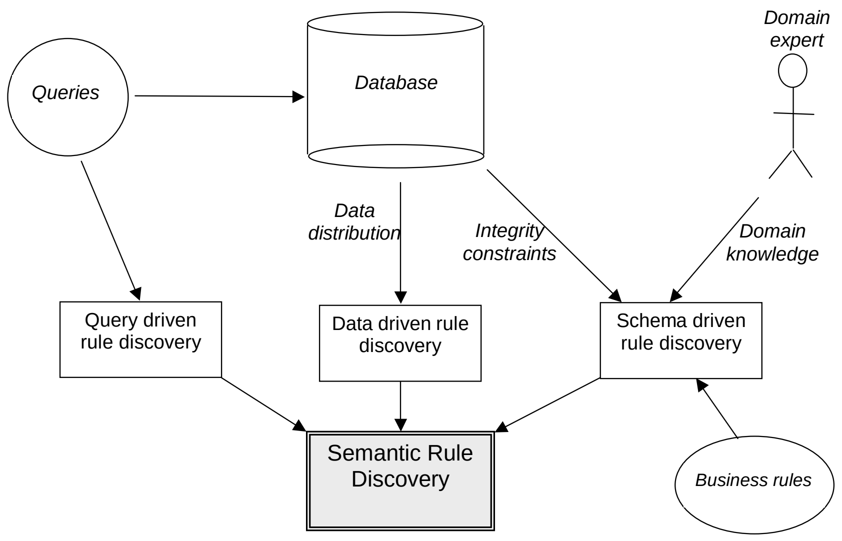


Figure 3: *Semantic Rule Discovery: Semantic information is harvested from an analysis of 1. queries 2. data distribution 3. schema constraints.*

Definition 3.11 ***Semantic query optimization**: SQO is the process of uncovering semantic information (from all available sources) plus* query rewrite, *where the aim is to transform the original query into one which is semantically equivalent but more efficient.*

Most researchers in the field [AF94, CGM90, GGM96, GSZZ01, YS89] use this term to refer to query rewrite. However, we specifically include the activity associated with uncovering semantic information such as rule discovery, in addition to actual query rewrite. Query rewrite is therefore a necessary but not sufficient condition for SQO. Whatever methods are employed to derive semantic information, ultimately this activity results in the actual transformation of the query into a syntactically different but semantically equivalent query.

Definition 3.12 ***Data reorganization**: This is the physical relocation of data plus the creation of auxiliary data structures such as clusters, indexes or materialized views, for the purpose of increasing query efficiency.*

One undertakes data reorganization with the aim of optimizing access to data, primarily data which is stored on disk.[12] The discovery of certain semantic information provides compelling evidence for the creation of (for example) clusters, indexes or materialized views.

Example 3.2 *A simple analysis of queries made against a particular database reveals that the most expensive queries are joins between three tables* `A`, `B` *and* `C`*. The DBA decides to co-locate* `A`, `B` *and* `C` *and checks that the join column is indexed. The tables themselves are now subject to further scrutiny with a view to discovering rules to be utilized in SQO.*

[12]This is a well researched topic, beyond the scope of this paper. We simply note that disk access times are typically orders of magnitude greater than memory access times.

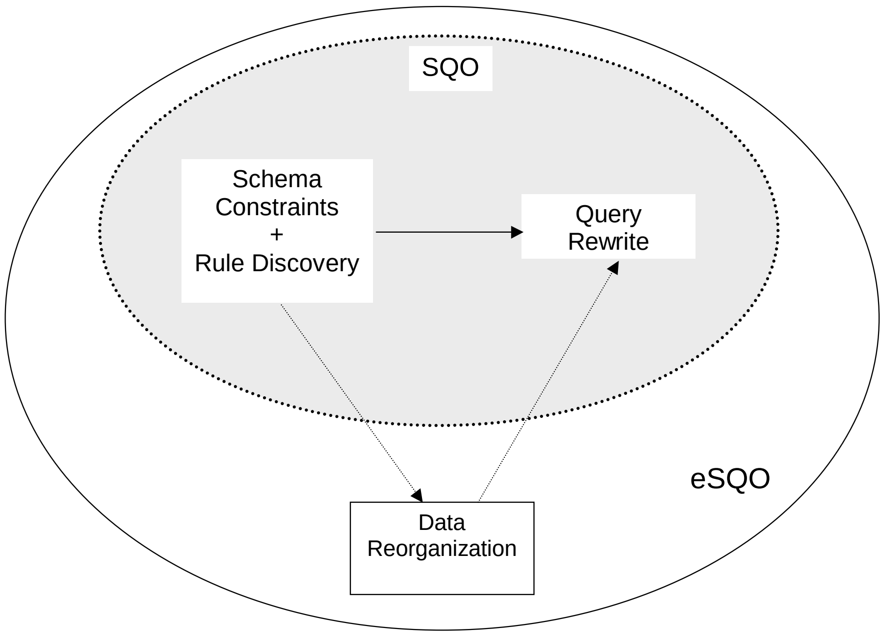


Figure 4: *Semantic query optimization (SQO) and Extended semantic query optimization (eSQO). Rule discovery drives both query rewrite and data reorganization.*

Clearly, this is a type of query optimization and depends on semantic information for its success. The extraction of this kind of information is traditionally associated with normal DBA duties.

Definition 3.13 ***Extended semantic query optimization (eSQO)***: *Extended semantic query optimization is SQO plus data reorganization.*

We use the new term *extended* SQO because we note that the discovery of semantic information not only leads to query rewrite but can provide compelling reasons for data reorganization.

Definition 3.14 ***Query profile (QP)***: *A query profile is a high level description of queries actually made against the target database.*

We use this new term to refer to a query analysis whose aim is not to identify a particular result set, but rather to identify eSQO strategies which are likely to enhance query efficiency. For example, at its simplest level, a QP notes which objects have actually been queried. This is valuable information; we now know what objects should be targeted for optimization.[13] Note that finding a QP is distinctly different from the *query driven rule discovery* defined in Definition 3.10.

Discovery of QP is analogous to the rule discovery phase[14] advocated by [SHKC93] where it is envisaged that semantic knowledge is discovered from the database and converted into semantic rules which may then be utilized by the semantic query optimizer.

[13]We argue later that SQO without a QP may be of limited value.
[14]See Definition 3.10.

While other writers suggest knowledge discovery might be guided by what queries are actually made, we make the stronger claim that discovery of the query profile ought to be a pre-requisite for rule discovery and strongly influence its focus. This is because one may infer suitable starting points or *parent* nodes[15] in the search for *relevant* semantic information. This is equivalent to an initial heavy pruning of the space of possible rules, making it much more likely discovered rules are relevant. We note that the capture of a query profile is already a normal part of DBA activities and it is easy to capture a simple QP using available software. We give a specific example of a simple QP in Example 5.2. We focus on the critical role of QP in Section 5 where we argue that SQO without a knowledge of QP may ultimately be futile.

3.2 SQO and commercial RDBMS

Our background in the IT industry has led to the observation that SQO is a largely unutilized technique, despite the prevailing view that SQO is useful. We now look at some of the reasons advanced by other researchers as to why SQO is not routinely employed. We then note some important changes to hardware typically employed in the database industry. We conclude the section by reiterating that current SQL optimizers lack even a basic semantic reasoning engine. We employ two examples which foreshadow some of the main points presented in Section 5.

3.2.1 Identified impediments to the implementation of SQO in RDBMS

SQO is known to be useful [YHPM99, Dat95, HK94]. For example [HK94] reported in 1994 that SQO was achieving average speedups of 20–40%. Many writers have claimed significant benefits from their own flavour of SQO [HK96, GD98, SL96] and there are some experimental implementations [CGK$^+$99, GGXZ01][16]. In [Dat95], the author comments that semantic optimization could potentially provide much greater performance improvements than more traditional algebraic optimizers, but that few commercial products, if any, do much in the way of semantic optimization.

In [CGK$^+$99], the authors put forward two reasons why SQO has never caught on in the commercial world where most databases are RDBMS:

- SQO is designed for deductive databases where the relatively high cost of applying complex rules (in comparison to much less complex rules in relational databases) is more likely to make the extra computational effort of implementing SQO worthwhile;
- CPU speeds are not high enough for the extra computational cost of SQO to be acceptable.

In [GGXZ01], the authors consider the role of schema constraints[17] in capturing *business rules* and identify four reasons for SQO techniques not being employed:

- The potential for using schema constraints to capture business rules is only now being realized, so opportunities for SQO have until now seemed limited;

[15]We use the word *parent* in this context to mean the root of a search tree consisting of interesting nodes.

[16]These papers report a restricted implementation of *join elimination* and *predicate introduction* in the DB2 commercial RDBMS implementation.

[17]These authors use the term *integrity constraint* but the meaning is the same.

- The expense of checking schema constraints at data insert or update time has limited the use of schema constraints, so opportunities for SQO have until now seemed limited;
- Many semantic rules which could potentially be used as schema constraints are simply not discovered;
- Even if a semantic rule is discovered there may be no justification for making it a schema constraint.

The third point point reinforces the notion of a *rule discovery phase*. Without such a phase, only rules that are known *a priori* can be employed. The last point addresses the notion of the *relevance* of the discovered rule. A discovered rule may reflect a true correlation between data and is therefore valid, but it may address a part of the domain which is of no interest (for example, because the rule antecedent never appears in a query).

3.2.2 Significant changes in computer hardware

We now make some observations concerning changes in computer hardware typically employed in the database industry. We argue that these changes have made SQO significantly more attractive than in the years 1980–2000 when much foundational work in SQO was done.

- Average data volumes have increased by several orders of magnitude, driven by the rising use of data warehouses and the falling cost of disk storage. Therefore even small increases in query efficiency offered by SQO may now be worthwhile.
- Available RAM has increased, typically by a factor of three or four, driven by the falling cost of RAM[18]. In DBMS, the impact of increasing available main memory is seen in the increasing proportion of the database that runs in memory. Ultimately, when sufficient main memory is made available, most of the database runs in memory most of the time and, crucially, disk activity is minimized. In RDBMS, this equates to most queried table data plus most procedural and SQL code being held in cache most of the time. In this environment, any query optimization that reduces disk activity is likely to be significant.
- Distributed databases, where a single logical database comprises several geographically distant nodes, are now commonly deployed by businesses across their WANs[19]. Distributed databases introduce delays in query answering, primarily because of the cost of transporting data between physical nodes. In this environment, data is typically partitioned across physical nodes according to simple semantic rules. These rules may then be utilized by a simple semantic query optimizer to minimize communication costs.

Example 3.3 *We refer to the motivational example in Section 2. Suppose the database is distributed across three physical nodes located in Auckland, Wellington and Sydney. Each node holds data strictly for its own Sales Office. A simple semantic query optimizer is constructed which determines which Sales Office is being queried and routes the query to the correct nodes while preventing the query from being passed to the remaining nodes.*

[18]PC based databases are now routinely endowed with several Gb of RAM. Larger database systems are routinely endowed with tens of Gb.

[19]*W*ide *A*rea *N*etwork

3.2.3 Current SQL optimizers lack a semantic reasoning engine

Current SQL optimizers cannot utilize semantic information because they lack a semantic reasoning engine [GGM96, HK00]:

- they are unable to detect inconsistent queries;
- they are unable to resolve schema constraints with query restrictions.

We now illustrate this with two simple examples. Our objective is to demonstrate that although the semantic information we require is available, it cannot be utilized

Example 3.4 *We refer to the motivating example in Section 2. Consider the SQL query which is submitted in error:*

```
select * from LOCATION where 1 = 2;
```

Clearly this query will return no rows, independent of table `LOCATION`*. Yet in general SQL optimizers will blindly submit such a query to the database.*

Example 3.5 *We refer to the motivating example in Section 2. Suppose the* `ORDER` *table contains a check constraint on column* `DATE` *to capture the domain knowledge that the company was formed on 01 April 2004 and that Orders can never be post-dated*[20]*:*

```
check DATE between '01-APR-2004' and SYSDATE;
```

and we then query the database with a where clause which includes the restriction:

```
and DATE < '01-APR-2004'
```

If the constraints defined by the database architect have any meaning at all, it follows that we can add to the above restriction a second restriction which simply reiterates the check constraint:

```
and DATE < '01-APR-2004' and DATE between '01-APR-2004' and
SYSDATE;
```

This query therefore returns no rows and at a stroke we might have realized an important optimization, namely we might have prevented a null query from ever being submitted to the database.

[20]`SYSDATE` is a function that returns the current system date.

4 Problem statement

In this section we consider in more detail the problems we listed in Section 1:

1. In the context of RDBMS, previous research does not make clear the unique role that schema constraints can play in SQO. In addition, the close relationship between SQO and the enforcement of database integrity constraints at query time has not been established.

2. While many writers consider SQO in the context of deductive databases (DD), very little work focusses on the efficacy of SQO for relational databases (RDB) and even less describe actual implementations.

3. Previous research does not make clear under what conditions it is likely to be worthwhile implementing SQO or what types of SQO are likely to be effective for a given database schema.

We begin with Problem 1 in Section 4.1 by highlighting some important aspects of schema constraints which are generally ignored by researchers. We then focus on Problem 2 in Section 4.2 and argue that, while acknowledging that the relational database model is subsumed by the deductive database model, it is nevertheless highly advantageous to consider SQO specifically in the context of RDBMS. We conclude this section by focussing on Problem 3 in Section 4.3 where we describe why, for a given database schema, attempting to exploit SQO without knowing what types of queries are actually made on the database, may result in little or no query optimization.

Although we may *conjecture* these problems have impeded the uptake of SQO, we are primarily concerned here with highlighting the fact that SQO provides a credible methodology which is particularly relevant to RDBMS. However, SQO can be expensive [AF94, Che96, GGMR97, GSZZ01] and we therefore argue it is unlikely to be implemented unless there can be confidence it will be worthwhile.

For each of the problems we considered in this section, we propose a resolution in Section 5.

4.1 Problem 1: RDBMS schema constraints and SQO

In the context of RDBMS, previous research does not make clear the unique role that database integrity constraints can play in SQO. Our argument proceeds as follows. In this section we show that it is often unjustified to assume that data actually stored in the database conforms to known database semantics, as defined by the schema constraints. We conclude this section by asserting that the search for rules to utilize for SQO is meaningless when performed on data stored in a database schema for which schema constraints have not been enforced. In the corresponding Section 5.1 we propose a solution to this problem.

We note that schema constraints are applied to data at the time of insertion or modification i.e. at insert, update and delete time. At all other times, these constraints are inactive. In particular, they play no part whatsoever in the querying of data, in the sense that no constraint is enforced when a query is submitted. This point is touched on by [GGGM98] in the context of deductive databases.

This would not present a problem were it always the case that constraints were enforced throughout the lifetime of the schema. Then we could *assume* the conformance of stored data to the schema semantics. Yet in reality it is often the case that various constraints are relaxed during the lifetime of the schema. This practice is in fact pervasive in the industry and may occur

- for performance reasons. Active constraints in the database may be quite expensive to enforce. Database Administrators (DBAs) may disable various constraints during, for example, the loading of large amounts of data simply because of the computational burden of checking that constraints are satisfied. This is widely practiced in the industry and all commercial RDBMS allow constraints to be selectively relaxed while retaining the constraint definition. [21]

- because referential integrity is enforced procedurally. A database architect may decide to enforce various data relationships via procedures defined and stored in the database if these relationships are complex and/or costly.

- because business rules are enforced in a layer (typically the middle of three layers) above the database. The N-tier model of data processing is very popular and it is usual in this methodology to consider the database as simply a persistence mechanism[].

It is easy to imagine the consequences of *assuming* the conformance of data to the known schema semantics, as our example in Section 2 illustrates for a typical transactional database. However, the consequences for a data warehouse which is being mined for information are even more serious. Any conclusions inferred from this mining can only be meaningfully interpreted with respect to the schema semantics. But if one cannot be sure the data accurately reflects these semantics, then the robustness of any conclusions must be questioned. [22]

We conclude this section by asserting that the search for rules to utilize for SQO is meaningless when performed on data stored in a database schema for which schema constraints have not been enforced. We propose a resolution to this problem in Section 5.1.

4.2 Problem 2: Implementing SQO specifically for RDBMS

While many writers consider SQO in the context of deductive databases (DD), very little work focusses on the efficacy of SQO for relational databases (RDB) and even less describe actual implementations.

The DD model subsumes the RDB model [GGGM98]. So it is generally assumed that what is true for DD models is also true for RDB models. Currently, commercial RDBMS[23] implement the SQL–2 specification [DD93] which (for example) describes the standard for encoding schema constraints and does not allow recursion. However, we note with [GGGM98] that the SQL–3 specification [DD97] does support recursion so that any RDBMS that supports the SQL–3 standard will in fact be a DD.

[21]In fact a ”best practice” has emerged in the industry whereby all appropriate constraints are defined in the database but disabled immediately. This allows the data semantics to be available to various tools while escaping the performance penalty. For example, the database may be ”reverse engineered” to discover its Entity-Relationship (ER) diagram. Such a tool will query the defined but unenforced constraints.

[22]This is in fact what drives the *data conditioning* stage in a typical data warehouse design. Data is conditioned or cleansed to ensure it conforms to the expected format and that the various data fields contain meaningful values before it is actually loaded into the warehouse tables. The real aim of data conditioning is to enhance the robustness of knowledge inferred from the data warehouse. This is a well researched area and beyond the scope of this paper.

[23]such as *DB2*, *Oracle*, *Postgres*

One must be careful that arguing in this manner does not obscure the opportunities for SQO in RDBMS. We now look at several assumptions commonly made about SQO, but examine them specifically in the context of RDBMS.

4.2.1 Schema constraints are finite

It is commonly assumed that the search for relevant semantic rules to apply in SQO leads to a combinatorial explosion of rule choice [Kin81, SO87, SSS92]. In Section 3 we made the (apparently facile) observation that schema constraints are implemented by human practitioners. Yet this allows us to safely conclude that we only have to deal with a *finite* number of semantic rules. Even if multiple constraints may be applied to some data items, clearly we still avoid the combinatorial explosion that many writers report may result from a mechanical analysis of data. Thus we eliminate the need for heuristics to guide the search for relevant rules and the need to limit the analysis with various stopping rules, as described in [SSD92].

4.2.2 After schema constraints, what then?

It is commonly assumed that any effective implementation of SQO will include a *rule discovery phase* during which unknown semantic knowledge will be uncovered which will then be formulated into rules to be utilized by a semantic query optimizer [SHKC93].

Yet following through this assumption in the context of RDBMS leads to some curious conclusions. Consider a RDB schema for which the application business rules have been captured (in as much as the RDBMS system allows) and expressed as schema constraints. Suppose that we undertake a rule discovery phase and uncover some (previously unknown) semantic information which allows us to write semantic rule `R1`. According to our own definitions (see Section 3) `R1` is either relevant or irrelevant. If it is irrelevant, we discard it. If it is relevant, it must be a valid business rule. But this can only have occurred because the business rule has been overlooked by domain experts.[24] While it seems reasonable to suppose domain experts have missed a small subset of business rules, conversely it seems quite unreasonable to assume a significant number of business rules have somehow been missed.

Let us assume that one would only undertake a rule discovery phase on the assumption that there was a *significant amount of relevant semantic knowledge that remained undiscovered*. A key notion here is *relevance*. Recall that we consider a semantic rule to be relevant if (and only if) it may be utilized by the semantic query optimizer to make a query more efficient. Recall also that research into SQO typically envisages a semantic query *pre*-processor where the raw query is first submitted to the semantic query optimizer and the output then submitted to the normal (syntactic) query optimizer. (See Figure 5.) So whatever query transformation is made by the SQO stage, ultimately any advantage bestowed by that transformation must in fact be realized by the normal syntactic query optimizer. This is equivalent to saying that the increased efficiency of the semantically optimized query must ultimately be reflected by the ability to fetch the answer tuples more efficiently. Typically, this implies that a helpful index exists. Indexing in RDBMS is a well studied topic, beyond the scope of this paper.

[24]Much AI literature reports the difficulty of capturing commonsense, background or contextual knowledge. For example, suppose the `EMPLOYEE` table contains a reference to maternity leave. Then a correct schema must specifically capture the fact that only female employees can be pregnant, in addition to noting which employees are on maternity leave.

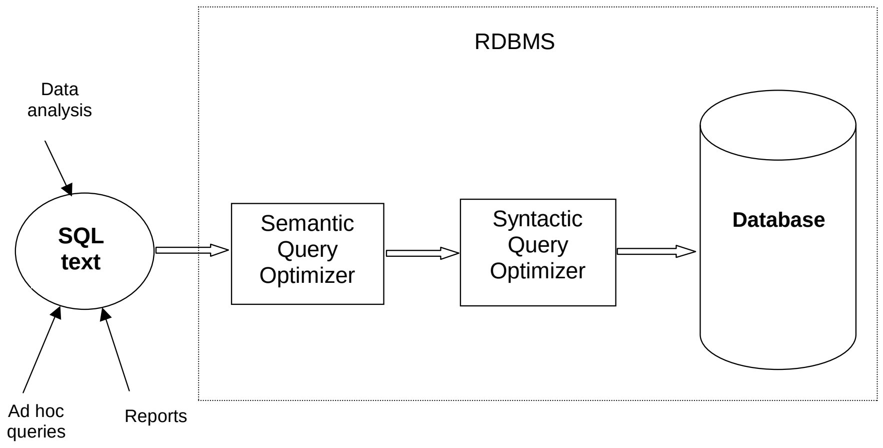


Figure 5: *Most research envisages SQO taking place as a preprocessing stage. SQL query text is first semantically optimized then passed to the conventional (syntactic) optimizer. Any advantage bestowed by the semantic optimizer can only be manifested by the syntactic optimizer. The syntactic optimizer will typically look to indexes to enhance query efficiency.*

However, we note that the conditions for effective indexing are well understood and that it seems very unlikely indeed that a large number of candidate indexes are lying undiscovered in the schema.

We conclude this section by asserting that while it may be reasonable to assume that *a small subset* of relevant schema constraints might be missed by domain experts, conversely it is unreasonable to assume a significant number of business rules have somehow been missed and remain to be discovered by a mechanical analysis. We propose a solution to this problem in Section 5.1.

4.3 Problem 3: Determining when SQO is worthwhile

Previous research does not make clear under what conditions it is likely to be worthwhile implementing SQO or what types of SQO are likely to be effective for a given database schema. In this section we show that, for a given database schema, attempting to exploit SQO without knowing what types of queries are actually made on the database, may result in little or no query optimization. We employ two helpful Examples 4.1 and 4.2 which both refer to the motivational example in Section 2.

Example 4.1 *Scenario One: Inspection of the* `ORDERS` *table reveals that most orders are made by a subset of active customers. In fact the distribution of customer orders is approximately normal. However, various reports which interrogate the* `ORDERS` *table mechanically query over the entire range of customers.*

The situation described in Example 4.1 is pictured in Figure 6 which shows the relative number of Sales made to each Customer. On the same graph we plot the the relative number of sales queries made on each customer. This distribution is rectangular because we are supposing the vast majority of queries arise from reports which mechanically interrogate the `ORDERS` table over the entire range of customers. All

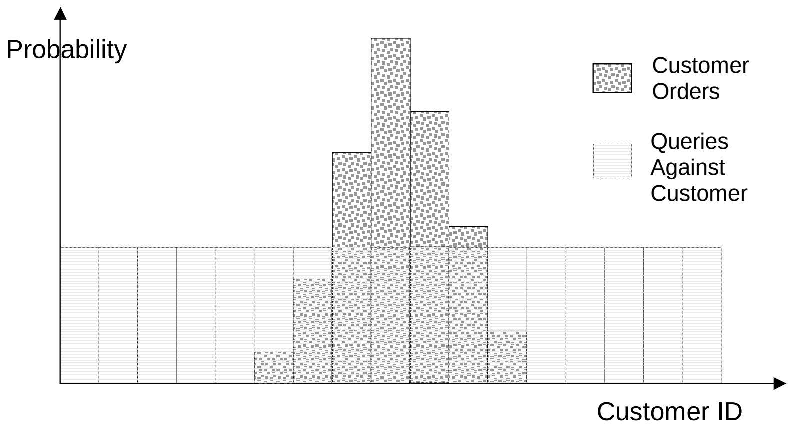


Figure 6: *Scenario One: Referring to Example 4.1, the plot sketches the relative number of Orders made by each Customer, on which is superimposed the relative number of* queries *made on each Customer. In this scenario, the distribution of customer orders is approximately normal. However, various reports mechanically query over the entire range of customers.*

queries that ask for Customer Sales outside of the range of active customers are in fact *null queries* i.e. queries which return no rows. The prevention of null queries being submitted to the database is seen as a major advantage by many writers because a 100% saving is made against the potential cost of the query (neglecting the effort required to deduce the logical contradiction) [IBG94, HK96, GG96, ZO97, GD98, YHPM99]. To our knowledge, no commercial RDBMS implementation performs this optimization.

It is clear in Example 4.1 that it is very likely to be worthwhile constructing a semantic query optimizer that utilizes knowledge about the distribution of Customer Orders. Yet it is easy to imagine the opposite situation i.e. queries on customers outside of the active subset of customers are rarely if ever made. In this case, it is equally clear that this type of SQO is unlikely to be worthwhile.

Example 4.2 *Scenario Two:The same employee infers (incorrectly) from the existence of Sales data pertaining to the new `Singapore` office that many queries are asking for `Singapore` data. Anxious to prevent another employee from making a mistake with Sales totals, she suggests intercepting each query and returning null to those that ask for `Singapore` data. The DBA responds that all users are aware of the location of the company's sales offices and it is very unlikely that anyone at all is querying this data. Instead, the DBA proposes that the check constraint on the `LOCATION` column be added to each and every query on the `SALES` table.*

In Example 4.2 we imagine the opposite situation to Example 4.1 i.e. SQO is unlikely to be worthwhile. Suppose we construct a semantic optimizer that checks the value of the Sales Office for every query that asks for sales data, returning null for any unknown Sales Office. But this effort is futile since no queries are ever made against unknown Sales Office data. While it may be futile to attempt SQO here, it may still be highly advantageous to add the check constraint at query time, as described in Section 4.1. Figure 7 depicts a more general situation where the distribution of query constraints on a particular column variable approximately matches the distribution of column values.

In other words, out of range queries are seldom if ever made. Therefore, semantically optimizing for out of range queries is unlikely to be effective in this situation.

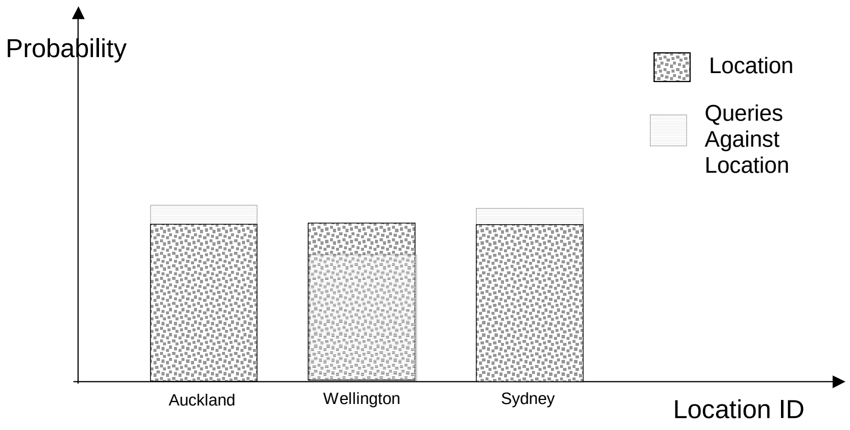


Figure 7: *Scenario Two: Referring to Example 4.2, the plot sketches the relative number of* queries *made on each company Office. In this scenario, each company Location is queried with approximately the same probability.* Out of range *queries are seldom if ever made. Therefore, semantically optimizing for out of range queries is unlikely to be effective.*

We conclude this section by referring to the latter part of Example 4.2. The DBA has proposed adding the check constraint on the `LOCATION` column to *every* query made on the `SALES` table. This proposal is driven by the intuition that any decrease in query efficiency that might result from this will be negligible in comparison to the execution cost of the query. [CGK$^+$99] report that their semantic query optimizer typically spent less than 1% of the total query time on the query rewrite phase. This alludes to a solution we propose in Section 5.2.

5 Preliminary results of our research

In this section we describe some preliminary results of our research to address the problems described in Section 4. We argue that the enforcement of schema constraints at query time adds a data robustness that significantly enhances the validity of information derived from the database. We then sketch how a simple but effective semantic query optimizer can readily be constructed from schema constraints alone. We reiterate a key observation that there is no semantic reasoning engine built in to SQL optimizers.

5.1 Proposed solution to Problem 1: RDBMS schema constraints and SQO

In Section 4.1 we focussed on some important characteristics of schema constraints that are generally ignored by researchers. We now describe how we propose to utilize schema constraints to construct an effective semantic query optimizer. Our argument proceeds by first showing that schema constraints may be easily enforced at query time in order to guarantee query answers actually conform to the known schema semantics.

We then argue that, having enforced schema constraints at query time, we are but one step away from a simple but effective semantic query optimizer. We then summarize the distinctive features of schema constraints and conclude that it seems axiomatic that schema constraints are utilized early in any effective optimization methodology.

5.1.1 Enforcing schema constraints at query time

We now describe the benefits that accrue from enforcing schema constraints at query time. We firstly reiterate that schema constraints are *static* i.e. we may assume their validity for the lifetime of the schema. This is equivalent to saying that we may add any relevant schema constraint to any query at any time and the meaning of the query will not change, with respect to the known schema semantics.

Example 5.1 *We refer to the motivating example in Section 2. The employee concerned now corrects the erroneous report by adding an additional restriction to the SQL query that calculates Total Sales:*

```
and LOCATION in ('Auckland', 'Wellington', 'Sydney');
```

The added query restriction is simply a reiteration of the check constraint and ensures that only sales known to have originated from the three target offices are actually included in the total.

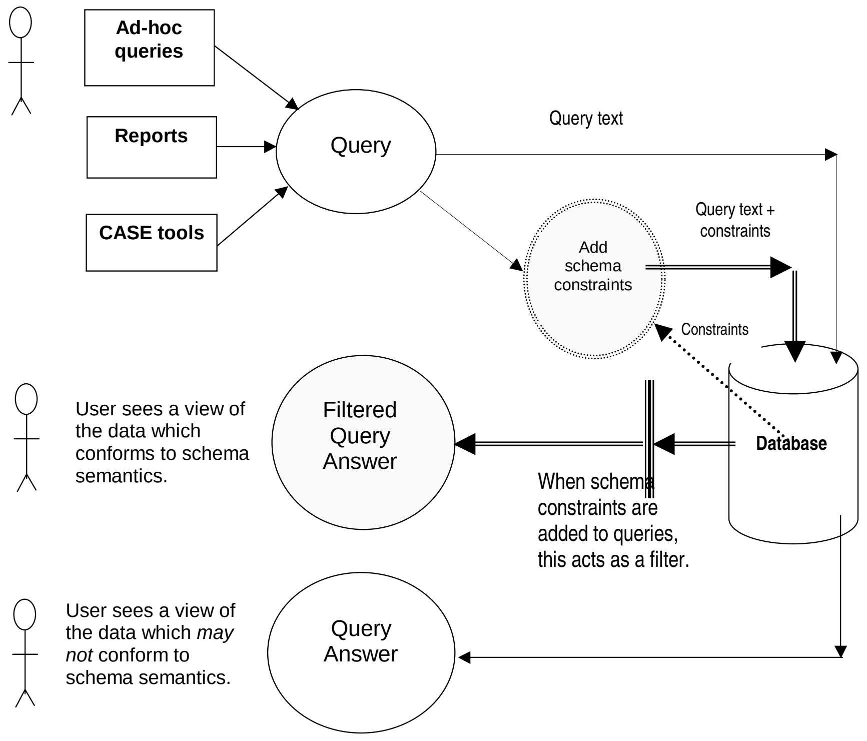
The above example illustrates the enforcement of a schema constraint simply by adding it to the query concerned. When all relevant schema constraints are added to the query, we can be sure that the answer returned will always conform to the known schema semantics. Crucially, this is true irrespective of whether or not all data stored in the database actually conforms to those schema semantics. This is particularly important when information is being inferred from the database, as in the case of data mining in data warehouses. Thus, enforcement of schema constraints at query time adds to the robustness of any knowledge that may be inferred from the data. It effectively filters any answers returned by the query such that only data which conforms to the known schema semantics is actually considered. We illustrate the unique role of schema constraints in Figure 8.

5.1.2 SQO follows from the enforcement of schema constraints at query time

So far in Section 5.1 we have highlighted the potential role of schema constraints in enforcing schema semantics at query time. We now illustrate how, having enforced schema constraints at query time, we are but one step away from a simple but effective semantic query optimizer. We refer back to examples 3.4 and 3.5.

We make the reasonable assumption that domain experts have identified all but a small subset of the application's business rules and that these have been captured and enforced by schema constraints. In Example 3.5 we illustrated how a null query results from the resolution of the check constraint with the SQL query restriction. Yet current SQL optimizers do not perform even elementary semantic reasoning, so the (possibly expensive) query will be blindly submitted to the database.

We argue that if SQO is ever to be employed routinely in RDBMS, an efficient reasoning engine will need to be implemented which is compatible with current RDBMS technology. We note that object oriented (OO) programming methodologies and complex data types are now part of the SQL–2 and SQL–3 definitions and are available in



Ad-hoc queries
Reports
CASE tools
Query
Query text
Add schema constraints
Query text + constraints
Constraints
Database
User sees a view of the data which conforms to schema semantics.
Filtered Query Answer
When schema constraints are added to queries, this acts as a filter.
User sees a view of the data which *may not* conform to schema semantics.
Query Answer

Figure 8: *The Role of Schema Constraints: When schema constraints are enforced at query time, this effectively filters any query answer such that it reflects the correct schema semantics. This enhances the robustness of any knowledge that may be inferred from query answers.*

all the major commercial RDBMS implementations. We expect the OO paradigm to provide the mechanism to implement an efficient reasoning engine. This is the subject of a forthcoming paper.

We conclude this section by reiterating that the semantic rules required for a simple but effective semantic query optimizer are already available in RDB schemas where all but a small subset of the relevant application business rules have been captured and enforced by schema constraints. We argued that a key impediment to implementation of SQO in RDBMS is the lack of an efficient reasoning engine.

5.1.3 Summary of distinctive features of schema constraints

We now summarize the distinctive features of schema constraints.

- Schema constraints are static i.e. they are valid for the lifetime of the schema. Thus, static constraints require no re-validation when the state of the database changes (for example, when a database table is updated).
- Schema constraints are true *a priori* i.e.they require no discovery. This is important because much of the computational effort required to implement SQO is

expended in the discovery of semantic rules [Kin81, HZ80]. Schema constraints require no such effort to discover[25].

- Schema constraints typically incorporate the application business rules and hence are relevant[26].
- Schema constraints are finite. We avoid any combinatorial explosion of relevant semantic rules to apply during SQO.
- If relevant schema constraints are added back in to a query, just one more step is required to perform effective SQO. This final step requires a reasoning engine.

It seems axiomatic that schema constraints are utilized early in any effective optimization methodology. We illustrate the role of schema semantics in SQO in Figure 9.

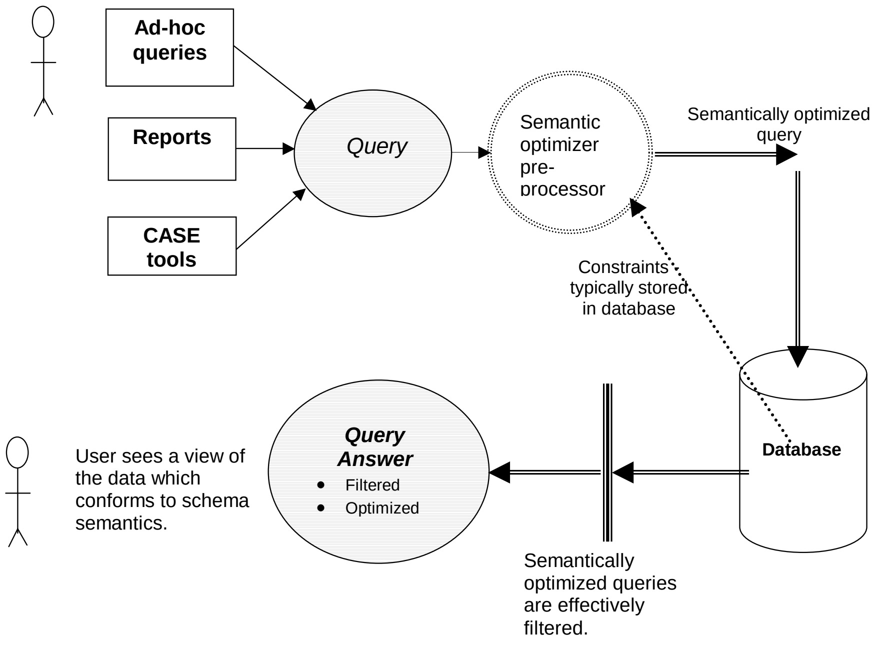

Ad-hoc queries
Reports
CASE tools
Query
Semantic optimizer pre-processor
Semantically optimized query
Constraints typically stored in database
Database
Query Answer
Filtered
Optimized
User sees a view of the data which conforms to schema semantics.
Semantically optimized queries are effectively filtered.

Figure 9: *Utilizing Schema Constraints for SQO: If schema constraints were utilized for SQO, queries would not only be optimized but query answers would correctly reflect schema semantics. Current syntactic SQL optimizers lack even an elementary semantic reasoning engine, so are unable (for example) to detect inconsistent queries.*

5.2 Proposed solution to Problem 2: Implementing SQO specifically for RDBMS

[CGK$^+$99, GGXZ01] describe an experimental implementation of a semantic query optimizer in the DB2 commercial RDBMS. Their optimizer is a highly restricted im-

[25]We neglect the effort required to gather domain knowledge and/or business rules and incorporate these into the DBMS.

[26]See Definition 3.8.

plementation but they report ”dramatic query performance improvements”. A key feature of their implementation is that it does not rely on complex integrity constraints, but uses only a subset of referential and check constraints. We now describe a simple semantic query optimizer for RDBMS that incorporates a much larger set of database constraints and subsumes the optimization described by [CGK$^+$99].

5.2.1 Assumptions

Our proposal for a semantic query optimizer makes the following assumptions:

1. All but a small subset of the application business rules have been captured by the database constraints.
2. The optimizer is implemented as a pre-processor, taking the raw SQL query as input and outputting one semantically equivalent query to the normal syntactic optimizer. (See Figure 5)
3. We utilize only the integrity constraints defined within the RDBMS.
4. We have a semantic reasoning engine which is at least an order of magnitude faster than the time required to actually retrieve query results. [27]
5. We do not allow foreign keys to be null. This is primarily to simplify the discussion in this paper.

5.2.2 Optimizer Design

Our proposed semantic query optimizer consists of three parts:

1. ***A constraint compiler***. The constraint compiler retrieves all constraints for each table and formulates a single SQL string incorporating all constraint information. Note that this need only be performed once since the constraints are static. This operation is therefore effectively costless.
2. ***A concatenate procedure***. This determines which tables are invoked in the SQL query and mechanically appends the appropriate constraint string.
3. ***A reasoning engine***. This is able to resolve query restrictions and detect null queries. [28]

Note that the implementation of the first two parts above is sufficient to guarantee that query results conform to the known schema semantics. We do *not* require that constraints be enforced during data insert or modification. We point to the compelling simplicity of the method we describe and comment that it is difficult to find reasons not to perform this, if one’s goal is to infer semantic rules or uncover semantic knowledge.

We do not attempt, in this simple design, to find any optimal combination of constraints. We are content (at this stage) with the assumption that the SQO pre-processor will not add significantly to the total query processing time. It is always possible of

[27]This is not an unreasonable assumption. Current syntactic optimizers are typically several orders of magnitude faster than average query retrieval times.

[28]In fact we can do a lot better than that. Our optimizer also performs simplification and generalization functions which greatly increase the chance that the query result is already cached. This is beyond the scope of this paper but is described in a forthcoming paper.

course that mechanically adding the constraint string to the SQL query text will *reduce* the query efficiency. But this is unlikely, since appending the constraint string can never enlarge the domain under consideration beyond what is defined by the application business rules. However, we argue that even if this is sometimes the case, it is still negligible in comparison with the total execution cost.

5.2.3 Constraint types utilized by our optimizer

Our proposed design captures and incorporates the following types of RDBMS constraints:

- ***not null***: Any column `COL` (of table `TAB`) declared to be not null results in the following restriction being added to every query on `TAB`:

  ```
  COL is not null
  ```

- ***check*** (including implication): Check constraints allow RDBMS to capture a powerful set of business rules, including range constraints:

  e.g: `check SALARY between 15000 and 150000`

  Most RDBMS allow any column in the data row to be referenced. This allows quite complex constraints to be expressed:

  e.g: `check PURCHASE_DATE > SHIP_DATE`

  Any implication `A -> B` may be expressed as `(not A or B)`

  e.g: `if LOCATION in ('Heathrow', 'Newham', 'Notting Hill')`
  `then CITY = 'London'`

  may be written as

  ```
  check LOCATION not in ('Heathrow', 'Newham', 'Notting Hill')
  or CITY = 'London'
  ```

- ***referential***: Any foreign key column must logically be at least as restricted as its parent. So it inherits the parents' restrictions, in addition to any restrictions of its own.

Note that we are not concerned with the cost of enforcing the constraints at data insert or modification time. This is a quite separate problem. We require only that the constraints be defined within the RDBMS so they may be harvested by the constraint compiler.

5.3 Proposed solution to Problem 3: Determining when SQO is worthwhile

In Section 4.3 we showed that attempting to exploit SQO without knowing what types of queries are actually made on the database, may result in little or no query optimization. For example, if null queries are never made against the database then attempting to detect them is of no value. On the other hand, the potential impact of a query from an unsophisticated user on a large data warehouse might make it a high priority to detect null queries.

Our research advocates the discovery of a *query profile* from which we may infer suitable starting points in the search for semantic information. This is equivalent to an initial heavy pruning of the space of possible rules making it much more likely discovered rules are relevant. We note that the capture of a query profile is already a normal part of DBA activities and it is easy to capture a simple profile using available software.

Example 5.2 *We refer to the motivating example in Section 2. The DBA decides to gather a simple query profile for the Sales database schema. She collects the following information by running a query analysis tool supplied as a standard part of the RDBMS.*

- *tables that are actually queried;*
- *columns that are restricted and the logical expression of that restriction;*
- *tables that are joined and the join column.*

The DBA infers from the query profile what parts of the schema are likely to be worth examining more closely for the purpose of query optimization.

We conclude this section by listing three compelling reasons to collect a query profile:

1. A query profile defines a set of potential targets for data reorganization.
2. A query profile defines a focus for further SQO.
3. A query profile reveals what SQO strategies are likely to be worthwhile.

6 Conclusion

In this section we first summarize the main contributions of this paper. We then briefly describe some next steps for our research.

This paper focusses on the efficacy of SQO in the context of RDBMS. Our ultimate goal is the construction of an effective semantic query optimizer for RDBMS.

- We present a thorough analysis of the current state of SQO. We propose definitions which clarify and to some extent simplify the terminology used by other researchers. We utilize these to form our own definiton of SQO which incorporates both the uncovering of semantic information (from all available sources) plus query rewrite.
- We identify three open problems which we argue inhibit the effective use of SQO in RDBMS. We focus on the unique role of schema constraints in SQO, the efficacy of SQO in the specific context of RDBMS and the worthwhileness of SQO for a given query profile.
- We propose solutions to these problems and describe first steps towards the implementation of an effective semantic query optimizer for relational databases. We describe how schema constraints may be enforced at query time to enforce

the known schema semantics. The addition of a reasoning engine completes (in principle) the design of an effective semantic query optimizer. We propose the capture of a query profile as a prerequisite for effective SQO and describe how this heavily prunes the potential search space for relevant semantic information.

There is no semantic reasoning engine built in to SQL optimizers. Next steps in our research will include the completion of the logical design of our Reasoning Engine (RE), followed by its implementation in a standard commercial RDBMS. Our RE is based on a generalization of an *interval* data type [GD98].

References

[AF94] K. Aberer and G. Fischer. *Semantic Query Optimization for Methods in Object-Oriented Database Systems*. Gesellschaft fuer Mathematik und Datenverarbeitung (GMD), Darmstadt, Germany, 1994.

[CGK$^+$99] Q. Cheng, J. Gryz, F. Koo, T. Y. Leung, L. Liu, X. Qian, and K. B. Schiefer. Implementation of two semantic query optimization techniques in db2 universal database. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 687–698. Morgan Kaufmann Publishers Inc., 1999.

[CGM90] S. Chakravarthy, J. Grant, and J. Minker. Logic based approach to semantic query optimization. *ACM Transactions on Database Systems*, 15(2):162–207, June 1990.

[Che96] I. A. Chen. Query answering using discovered rules. In *Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, pages 402–411. IEEE, 1996.

[Dat95] C. J. Date. *An introduction to database systems*. Addison-Wesley, 6 edition, 1995.

[DD93] C. J. Date and Hugh Darwen. *A Guide to The SQL Standard*. Addison-Wesley, 3 edition, 1993.

[DD97] C. J. Date and Hugh Darwen. *A Guide to The SQL Standard*. Addison-Wesley, 4 edition, 1997.

[GD98] B. Genet and G. Dobbie. Is semantic optimisation worthwhile? In *Proceedings of the 21st Australasian Computer Science Conference*, pages 245–256, Perth, Australia, February 1998.

[GG96] P. Godfrey and J. Gryz. A framework for intensional query optimization. In *Workshop on Deductive Databases and Logic Programming*, 1996.

[GGGM98] P. Godfrey, J. Grant, J. Gryz, and J. Minker. Integrity constraints: Semantics and applications. In *Logics for Databases and Information Systems*, pages 265–306, 1998.

[GGM96] P. Godfrey, J. Gryz, and J. Minker. Semantic query optimization for bottom-up evaluation. In *Proceedings of the Ninth International Symposium on Foundations of Intelligent Systems*, volume 1079 of *LNAI*, pages 561–571, Berlin, June 9–13 1996. Springer.

[GGMR97] J. Grant, J. Gryz, J. Minker, and L. Raschid. Semantic query optimization for object databases. In *ICDE*, pages 444–453, 1997.

[GGXZ01] P. Godfrey, J. Gryz, H. Xu, and C. Zuzarte. Exploiting constraint-like data characterizations in query optimization. In *SIGMOD Record*, 2001.

[GSZZ01] J. Gryz, K. B. Schiefer, J. Zheng, and C. Zuzarte. Discovery and application of check constraints in db2. In *ICDE*, pages 551–556, 2001.

[HK94] C. Hsu and C. A. Knoblock. Rule induction for semantic query optimization. In *Proc. 11th International Conference on Machine Learning*, pages 112–120. Morgan Kaufmann, 1994.

[HK96] C. Hsu and C. A. Knoblock. Using inductive learning to generate rules for semantic query optimization. In *Advances in Knowledge Discovery and Data Mining*, pages 425–445. 1996.

[HK98] C. Hsu and C. A. Knoblock. Discovering robust knowledge from databases that change. *Data Mining and Knowledge Discovery*, 2(1):69–95, 1998.

[HK00] C. Hsu and C. A. Knoblock. Semantic query optimization for query plans of heterogeneous multidatabase systems. *Knowledge and Data Engineering*, 12(6):959–978, 2000.

[HZ80] M. Hammer and S. B. Zdonik. Knowledge based query processing. In *Proceedings of the 6th International Conference Very Large Data Bases*, pages 137–147, February 1980.

[IBG94] A. Illarramendi, J. M. Blanco, and A. Goni. Making knowledge base systems more efficient: a method to detect inconsistent queries. *IEEE Transactions on Knowledge and Data Engineering*, 6(4), August 1994.

[JK84] Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Comput. Surv.*, 16(2):111–152, 1984.

[Kin81] J. J. King. A system for semantic query optimization in relational databases. In *Proceedings of 7th VLDB Conference*, pages 510–517, 1981.

[SHKC93] S. Shekhar, B. Hamidzadeh, A. Kohli, and M. Coyle. Learning transformation rules for semantic query optimization: A data-driven approach. In *Special Issue on Learning and Discovery in Knowledge-Based Databases*, number 5(6), pages 950–964. Institute of Electrical and Electronics Engineers, Washington, U.S.A., 1993.

[SL96] A. Sayli and B. Lowden. The use of statistics in semantic query optimisation, 1996.

[SO87] S. T. Shenoy and Z. M. Ozsoyoglu. A system for semantic query optimization. In *Association for Computing Machinery Special Interest Group on Management of Data 1987 Annual Conference, SIGMOD Record*, volume 6, pages 181–195, December 1987.

[SSD92] S. Shekhar, J. Srivastava, and S. Dutta. A formal model of trade-off between optimization and execution costs in semantic query optimization. *Data and Knowledge Engineering (North-Holland)*, 8(2):131–151, 1992.

[SSS92] M. Siegel, E. Sciore, and S. Salveter. A method for automatic rule derivation to support semantic query optimisation. *ACM Transactions on Database Systems*, 17(4):563–600, December 1992.

[YHPM99] S. Yoon, L. J. Henschen, E. K. Park, and S. Makki. Using domain knowledge in knowledge discovery. In *Proceedings of the eighth international conference on Information and knowledge management*, pages 243–250. ACM Press, 1999.

[YS89] C.T. Yu and W. Sun. Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Transactions on Knowledge and Data Engineering*, 1(3), September 1989.

[Zhu92] Qiang Zhu. Query optimization in multidatabase systems. In *Proceedings of the 1992 conference of the Centre for Advanced Studies on Collaborative research*, pages 111–127. IBM Press, 1992.

[ZO97] X. Zhang and Z. M. Ozsoyoglu. Implication and referential constraints: A new formal reasoning. *Knowledge and Data Engineering*, 9(6):894–910, 1997.