Query Processing in a System for Distributed Databases (SDD-1)

PHILIP A. BERNSTEIN and NATHAN GOODMAN
Harvard University
EUGENE WONG
University of California at Berkeley
CHRISTOPHER L. REEVE
Massachusetts Institute of Technology
and
JAMES B. ROTHNIE, Jr.
Computer Corporation of America

This paper describes the techniques used to optimize relational queries in the SDD-1 distributed database system. Queries are submitted to SDD-1 in a high-level procedural language called Datalanguage. Optimization begins by translating each Datalanguage query into a relational calculus form called an *envelope*, which is essentially an aggregate-free QUEL query. This paper is primarily concerned with the optimization of envelopes.

Envelopes are processed in two phases. The first phase executes relational operations at various sites of the distributed database in order to delimit a subset of the database that contains all data relevant to the envelope. This subset is called a *reduction* of the database. The second phase transmits the reduction to one designated site, and the query is executed locally at that site.

The critical optimization problem is to perform the reduction phase efficiently. Success depends on designing a good repertoire of operators to use during this phase, and an effective algorithm for deciding which of these operators to use in processing a given envelope against a given database. The principal reduction operator that we employ is called a *semijoin*. In this paper we define the semijoin operator, explain why semijoin is an effective reduction operator, and present an algorithm that constructs a cost-effective program of semijoins, given an envelope and a database.

Key Words and Phrases: distributed databases, relational databases, query processing, query optimization, semijoins
CR Categories: 3.70, 4.33

This research was supported by the Advanced Research Projects Agency of the Department of Defense and was monitored by the Naval Electronic System Command under Contract N00039-77-C-0074, ARPA Order 3175-6.

Authors' addresses: P.A. Bernstein and N. Goodman, Center for Research in Computing Technology, Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138; E. Wong, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA 94720; C.L. Reeve, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; J.B. Rothnie, Jr., Computer Corporation of America, 675 Massachusetts Avenue, Cambridge, MA 02139.

1. INTRODUCTION

SDD-1 is a distributed database system developed by the Computer Corporation of America [23]. SDD-1 permits a relational database to be distributed among the sites of a computer network, yet accessed as if it were stored at a single site. Users interact with SDD-1 by submitting queries coded in a high-level procedural language called Datalanguage [20]. Figures 1 and 2 illustrate an SDD-1 database and a Datalanguage query. This paper is concerned with efficient execution of such queries. Other aspects of SDD-1 are discussed in [5, 6, 14, 23].

Our objective is to process queries with a minimum quantity of intersite data transfer. That is, we assume network bandwidth to be the system bottleneck and seek to minimize use of this resource; all other resources are assumed to be free.[1] This assumption is appropriate in SDD-1 because the network is the slowest system component by two orders of magnitude.[2] This assumption has been

Database D

S(s#,	name,	location)
1,	Acme,	MA
2,	Best,	MA
3,	Mid,	NY
4,	Nadir,	CA

Y(s#,	p#,	qty)
1,	1,	20
1,	2,	50
3,	3,	50
4,	1,	10
4,	5,	75

P(p#,	name,	type)
1,	LSI,	micro
2,	P11,	mini
3,	360,	main
4,	CRI,	huge
5,	8080,	micro

S describes suppliers.
P describes parts.
Y tells which suppliers supply which parts and in what quantity.

Assume that S, Y, and P are sorted at sites 1, 2, and 3, respectively.

Fig. 1. Example database.

Description of query
List the supplier name, part name, and quantity supplied for all parts supplied by a Massachusetts supplier. Also, print how many of these are minis.

Query Q

```
Begin
  Count := 0;
  For S
    If S.location = "MA" then for Y
      If S.s# = Y.s# then for P
        If Y.p# = P.p# then begin
          Print S.name, P.name, Y.qty;
          If P.type = "mini" then Count := Count + 1; end;;;
  Print "Number of minis is", Count ;
End.
```

Fig. 2. Example datalanguage query. Datalanguage used in this example is defined in the appendix.

[1] In practice, database processing within sites is considered as a secondary objective. For expository clarity, we shall not treat this issue.
[2] Sites in SDD-1 are mainframe computers (PDP-10s), while the network is a packet-switched long-distance network (Arpanet). Sustainable bandwidth on the network is at most 10 kbits per second (see [24–26]).

Envelope E

Retrieve (S.s#, S.name, S.location)	where *qualification*
Retrieve (Y.s#, Y.p#, Y.qty)	where *qualification*
Retrieve (P.p#, P.name, P.type)	where *qualification*

qualification: S.location = "MA" $\wedge$ S.s# = Y.s# $\wedge$ Y.p# = P.p#

Fig. 3. Envelope for query of Figure 2. Envelopes are defined in Section 2. Intuitively, an envelope specifies a subset of each relation in the database. We express envelopes in a relational calculus similar to QUEL [15].

The result of envelope E is to retrieve *any superset* of the data specified by E. For example,

S(s#,	name,	location)
1,	Acme,	MA
2,	Best,	MA

Y(s#,	p#,	qty)
1,	1,	20
1,	2,	50

P(p#,	name,	type)
1,	LSI,	micro
2,	P11,	mini
3,	360,	main
4,	CRI,	huge
5,	8080,	micro

The specific superset retrieved is determined by efficiency considerations.

The retrieved relations are also transmitted to a single site, for example, site 3.

Fig. 4. Processing envelope of Figure 3.

adopted by other researchers [7, 8, 13, 16, 17, 33, 34], although naturally it is not appropriate in every system [10, 19, 27]. Section 5 discusses the impact of this assumption on our approach.

Our algorithm has three main steps. Step 1 maps a Datalanguage query Q into a relational calculus form (an *envelope*) that specifies a *superset* of the database needed to answer Q (see Figure 3). Step 1 depends on details of Datalanguage and is of general interest only insofar as Datalanguage resembles other procedural query languages. This step is described in [11].

Step 2 *evaluates the envelope*. This step retrieves a *superset* of the database specified by the envelope, assembling the result at a single site S_a (see Figure 4). (The specific superset retrieved and the "assembly site" S_a are determined by efficiency considerations.) Step 2 is accomplished by translating the envelope into a program P containing relational operations (a *reducer*), followed by commands to move the results of P to S_a (see Figure 5). The goal is to construct a reducer P and select a site S_a such that the cost of computing P and moving the results to S_a is minimum over all reducers and sites. This optimization problem constitutes the core of the SDD-1 query processing algorithm and is the focus of this paper.

Step 3 executes Q at S_a using the data assembled by Step 2. Since Step 3 only involves local query processing, it will not be discussed further. Steps 1–3 are outlined in Figure 6.

The paper has five sections. Section 2 defines envelopes and the operations used to process envelopes. Section 3 presents techniques for estimating the cost and effect of a reducer composed of these operations. Section 4 presents a heuristic algorithm that compiles envelopes into efficient (though not necessarily optimal) reducers. Section 5 discusses related work and suggests directions for

Program P

```
1. S := S[location = "MA"]     ; restrict S to MA suppliers
2. Y := Y⟨s# = s#]S            ; this operation is semijoin
                               —it computes the set of Y
                               tuples that corresponds to
                               MA suppliers.
end
```

Figure 4 shows the result of applying P to the database of Figure 1.

Fig. 5. Program for envelope of Figure 3.

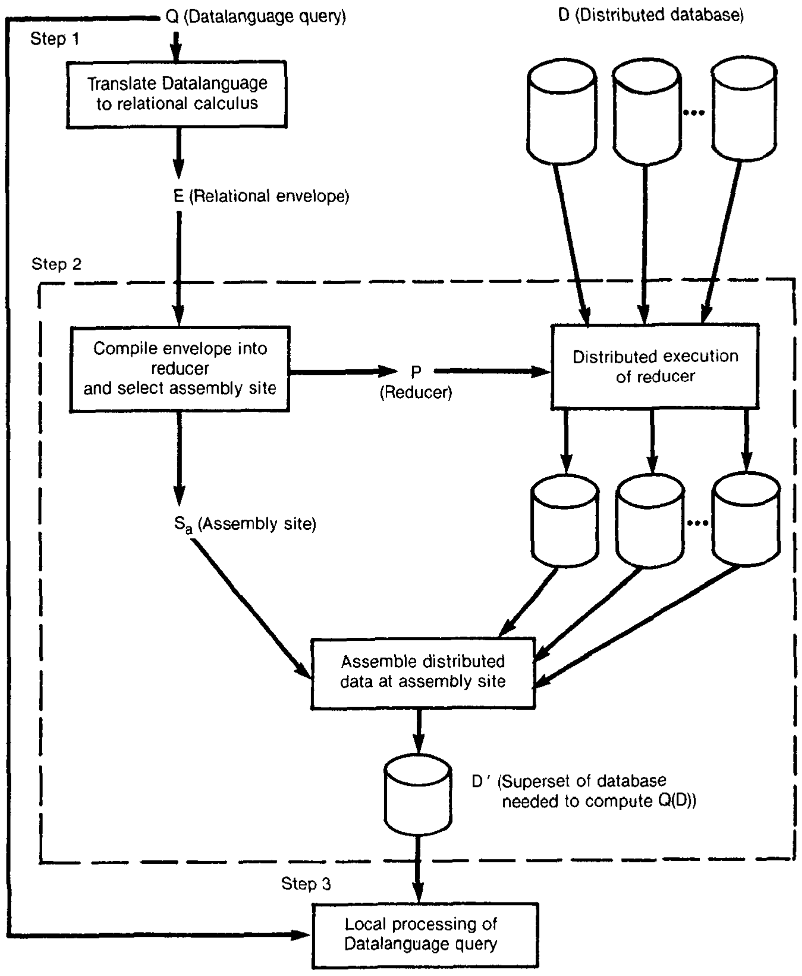
Q (Datalanguage query)
D (Distributed database)
Step 1
Translate Datalanguage to relational calculus
E (Relational envelope)
Step 2
Compile envelope into reducer and select assembly site
P (Reducer)
Distributed execution of reducer
S_a (Assembly site)
Assemble distributed data at assembly site
D′ (Superset of database needed to compute Q(D))
Step 3
Local processing of Datalanguage query

Fig. 6. Main steps of query processing algorithm.

(a) Relational Data Objects

Term	Definition
Domain	A set of values
Attribute	An alternate name for a domain
Relation schema	A description of a relation, consisting of a relation name and list of attributes
Relation	A subset of the Cartesian product of the domains of the attributes of the corresponding relation schema
Tuple	An element (or row) of a relation
Database	A set of relations
attr(R)	Attributes of relation R

(b) Relational Algebraic Operations

Restriction:	$R[A = k] = \{r \in R \mid r.A = k\}$ where r.A is the value of the A-domain in tuple r
Also	$R[A = B] = \{r \in R \mid r.A = r.B\}$
Projection:	$R[A_1, A_2, \ldots, A_n] = \{\langle r.A_1, r.A_2, \ldots, r.A_n\rangle \mid r \in R\}$
Join:	$R[A = B]S = \{\langle r, s\rangle \mid r \in R, s \in S, \text{and } r.A = s.B\}$
Semijoin:	$R\langle A = B]S = (R[A = B]S)\,[\text{attr}(R)] = \{r \mid r \in R \wedge r.A \in s[B]\}$

Fig. 7. Relational terminology.

future research. We assume reader familiarity with relational databases at the level of [9]. A review of relational terminology appears in Figure 7.

2. QUERY PROCESSING STRATEGY

2.1 Envelopes

The attributes of relation R are denoted attr(R). Relation R_i' is a *subrelation* of relation R_i, if $\text{attr}(R_i') \subseteq \text{attr}(R_i)$ and $R_i' \subseteq R_i[\text{attr}(R_i')]$. Let $D = \{R_1, \ldots, R_n\}$ and $D' = \{R_1', \ldots, R_n'\}$ be databases. D′ is a *subdatabase* of D, denoted $D' \leq D$, if R_i' is a subrelation of R_i for $i = 1, \ldots, n$. An *envelope* is a relational calculus expression that maps a database into a subdatabase. We express envelopes in a language similar to QUEL [15].

An *envelope* E consists of a *qualification* q and *target lists* $t_1, \ldots, t_n$. The term q is a Boolean formula with clauses of the form $R_i.A = R_j.B$ or $R_i.A = k$.[3] The terms $R_i.A$ and $R_j.B$ are called *indexed variables.* Each t_i is a set of variables indexed by R_i: that is, t_i is of the form $\{R_i.A_{i1}, \ldots, R_i.A_{il}\}$. Envelope E maps database D into subdatabase D′ defined by the following collection of QUEL queries.

Retrieve into $R_1'(t_1)$ where q.
⋮
Retrieve into $R_n'(t_n)$ where q.

We limit the form of envelopes in two additional ways. One, qualifications are

[3] Note that we avoid tuple variables. These can be accommodated by (conceptually) duplicating a relation, thereby having two relation names ranging over the same relation.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.