# A QUORUM-BASED COMMIT PROTOCOL

Dale Skeen

TR 82-483
February 1982

Department of Computer Science
Cornell University
Ithaca, New York 14853

# A QUORUM-BASED COMMIT PROTOCOL

Dale Skeen

Computer Science Department
Cornell University
Ithaca, New York

## Abstract

Herein, we propose a commit protocol and an associated recovery protocol that is resilient to site failures, lost messages, and network partitioning. The protocols do not require that a failure be correctly identified or even detected. The only potential effect of undetected failures is a degradation in performance. The protocols use a weighted voting scheme that supports an arbitrary degree of data replication (including none) and allows unilaterally aborts by any site. This last property facilitates the integration of these protocols with concurrency control protocols. Both protocols are centralized protocols with low message overhead.

## 1. Introduction

A transaction is, by definition, an atomic operation on a distributed database system. Either all changes by the transaction are permanently installed in the database, in which case the transaction is said to be committed, or no changes persist, in which case the transaction is said to be aborted. It is the task of a commit protocol to ensure that a transaction is atomically executed.

In this paper we propose a commit protocol that is resilient to multiple occurrences of the following classes of benevolent failures: arbitrary site failures, lost messages, and network partitioning. It does not require that the type of failure be correctly determined, in fact, resiliency is guaranteed even if failures go undetected.

The protocol uses a weighted voting scheme to resolve conflicts during failures. When failures occur, a transaction is committed only if a minimum number of votes, called a commit quorum and denoted $V_C$, are cast for committing. Similarly, in the presence of failures, a transaction will be aborted only if a minimum number of votes, called an abort quorum and denoted $V_A$, are cast for aborting. A commit quorum does not have to equal an abort quorum, but their sum must exceed the total number of votes.

Voting schemes have been proposed previously for transaction management. Thomas introduced a majority voting scheme to ensure consistency in a fully replicated database ([THOM79]). Gifford extended the scheme by assigning weights to sites and using quorums rather than a simple majority ([GIFF79]). The proposed protocol differs from the previous work in several important ways:

(1) It is a commit protocol, not a concurrency control scheme. It provides atomicity at a per transaction basis. Nonetheless, it is straightforward to integrate any type of concurrency control protocol into this protocol.

(2) It allows unilateral aborts during the first phase of the transaction. A site may decide to abort because of several reasons, for example, a deadlock is detected locally.

(3) It is primarily intended for partially replicated distributed databases where a transaction can read from any copy but must update all copies.

In addition, the protocol exhibits the following properties:

(1) It is a centralized protocol and, thus, benefits from the economy of centralized protocols.

(2) In the absence of failures it is no more expensive than previously proposed protocols that are resilient only to coordinator failures (and not to a partitioning of the network).

(3) If all failures are eventually repaired, then the protocol will eventually terminate.

(4) It is a blocking protocol -- operational sites must occasionally wait until a failure is repaired. This is an undesirable but necessary property exhibited by any protocol that is resilient to network partitioning ([SKEE81a]). However, the protocol can be tuned so that the

frequency of blocking is low.

This paper is divided into six sections. The second section states our assumptions and defines the terminology used in the remainder of the paper. The third section develops a resilient quorum-based commit protocol, and the fourth section develops a resilient quorum-based recovery protocol. The recovery protocol is invoked whenever a group of sites can no longer communicate with the original coordinator (either it has failed or the network has partitioned). Like the commit protocol, it is a centralized protocol. The fifth section discusses performance, and the sixth section concludes the paper.

Although the protocols proposed are resilient to many classes of failures, this paper will focus on the problem of network partitioning. This class of failures is generally agreed to the most difficult class to handle. The other two classes, site failures and lost messages, can be cast as special cases of a partitioned network. In a site failure, a single site is isolated (partitioned) from the remainder of the network. A lost message can be viewed as a very short lived partitioning. In all cases, the protocols work without modifications.

## 2. Background

We assume that an underlying communications network provides point-to-point communication between any pair of sites. We also assume that it generates no spontaneous messages, and that garbled messages are detected and deleted. We do _not_ assume that messages arrive in order nor that it detects lost messages.

A _partitioned network_ occurs when there are two or more disjoint groups of sites such that no communication is possible between the groups. Each of the disjoint groups is called a _partition_.

A distributed transaction T is decomposed into subtransactions $T_1$, $T_2$, ..., $T_N$, where a subtransaction is executed at one of the N participating sites. Any subtransaction can be unilaterally aborted, which results in the abortion of the entire transaction. Hence, for transaction T to be committed, _all_ sites must agree to commit their subtransaction. We assume that a subtransaction can be atomically executed by a local transaction management system ([GRAY79,LIND79]).

It is the responsibility of a _commit protocol_ to ensure that all subtransactions are consistently committed or aborted. One of the simplest commit protocols is the two-phase protocol ([GRAY79, LAMP76]) depicted in Figure 1. The protocol uses a central site, the coordinator, to direct the execution of the transaction at the other sites. Each slave has a chance to abort the transaction by replying with a "no" in the first round.

A commit protocol can be conveniently described by a set of state diagrams, one for each participating site ([SKEE81a]). The diagram for Site i describes the processing of subtransaction $T_i$. A state in the diagram is called a _local transaction state_.

In the two-phase commit protocol, a single state diagram (illustrated in Figure 2.) suffices to describe processing at all sites. For both the coordinator and the slaves, there are four distinct and easily identified

| **COORDINATOR** | **SLAVE** |
|---|---|
| (1) Transaction is received. *Subtransactions* are sent to each slave. | |
| | *Subtransaction* is received. A reply is sent: *yes* to commit, *no* to abort. |
| (2) If all sites respond *yes* then *commit* is sent; else, *abort* is sent. | |
| | Either *commit* or *abort* is received and processed. |

**Figure 1.** The two-phase commit protocol.

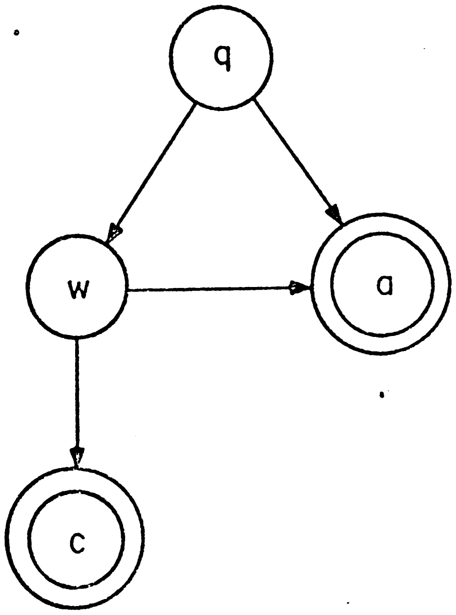

**Figure 2.** The state diagram for the two-phase commit protocol.

local transaction states: the *initial state* (state q in the diagram), the _wait state_ (w), the _abort state_ (a), and the _commit state_ (c). A site occupies the _initial_ state until it decides whether to unilateral abort the transaction. If the site decides against an abort, then the _wait_ state is entered. This state represents a period of uncertainty for the site, where it has agreed to proceed with the transaction but does not yet know its outcome (i.e. committed or aborted). The _commit_ and _abort_ states are self-explanatory.

The local transaction states of any protocol form two disjoint subsets: the _committable_ states and the _noncommittable_ states. A site occupies a committable state only if all sites have agreed to proceed with the transaction. For example, the only committable state in the two-phase commit protocol is the commit state. A state that is not a committable state is a noncommittable state.

## 3. A Resilient Commit Protocol

The two-phase commit protocol is not a very robust protocol. Whenever the coordinator fails or becomes partitioned from the slaves, the slaves must block until the failure can be repaired.

In this section we develop a very resilient commit protocol that allows recovery from both of these types of failures. The section develops the commit protocol in detail; the next section discusses the associated recovery protocols for handling coordinator failures and partitioning.

Each site is assigned an integral nonnegative number of votes. (The number can be 0, in which case the site is a passive participant.) The basic idea is that whenever a group of communicating sites establishes a quorum, they are allowed to proceed. There are two distinct types of quorums - a commit quorum and an abort quorum.

Let V, $V_C$, and $V_A$ represent the total number of votes, the number required for a commit quorum, and the number required for an abort quorum. A resilient quorum-based protocol must obey the following properties ([SKEE81c]):

(1) $V_C+V_A>V$ where $0<V_C,V_A<=V$

(2) When any site is in the commit state, then at least a _commit quorum_ of sites are in _committable_ states.

(3) When any site is in the abort state, then at least an _abort quorum_ of sites are in _noncommittable_ states.

These requirements are sufficient to ensure that a quorum-based protocol terminates in a consistent state -- if it does terminate ([SKEE81c]). The requirements are very similar to those for _k-resiliency_ where a protocol can tolerate upto k arbitrary site failures (see [ALSB76] for a definition of k-resiliency and [SKEE81b] for a set of sufficient conditions ensuring k-resiliency in a commit protocol). In both cases a minimum number of sites must agree before an _irreversible_ decision is made by any site.

The second requirement can be viewed as two subrequirements:

(2.1) Before the first site commits, a commit quorum of sites in committable states must be obtained, and

```
COORDINATOR                                          SLAVE'S RESPONSE

(1) Transaction is received.
    Subtransactions are
      sent to each slave.

                                                        Yes to commit
                                                         No to abort

(2) If all sites respond yes
      then
        prepare to commit is sent;
        continue to phase (3)
      else
        abort is sent;
        stop.

                                                              Ack

(3) If the sum of the weights
    of the responding sites equals
    or exceeds $V_C$
      then
        send commit to all
      else
        block (wait until a "merge").

                                                              --
```

**Figure 3.** The quorum based commit protocol.

(2.2) After any site has committed, a commit quorum must be maintained.

As a consequence of (2.2), a site can safely move from a committable state to a noncommittable state if and only if it can be shown that no site has committed the transaction, or it can be shown that this will not destroy a commit quorum.

The third requirement, concerning abort quorums, is analogous to (2). Hence, there exists (3.1) and (3.2) which are the analogs of (2.1) and (2.2).

The two-phase commit protocol does not satisfy the second rule, nor can it be simply extended to satisfy it. Moreover, any protocol which has a single committable state (which must be the commit state) cannot satisfy the rule. Hence, in a quorum-based commit protocol, we need to introduce a new committable state, the *prepared to commit* (pc) state. This state will substantially increase the cost of the protocol, but unfortunately, it is necessary.

The new protocol is described in Figure 3 and its state diagram is given in Figure 4. It requires three phases to commit, two to abort. The new phase is the second phase, where all sites move into the _prepared to commit_ state. The only explicit mention of quorums is in the third phase where the transaction is committed only if a commit quorum of sites advance to the _prepared to commit_ state. Even though abort quorums are not explicitly mentioned, the third requirement is still satisfied. In fact, if any site unilaterally aborts (including the coordinator), then no site ever enters a committable state and the third rule is trivially satisfied.

The protocol is a pessimistic protocol -- if any site fails or a partition occurs during the first phase, then the coordinator immediately aborts the transaction.

## 4. Recovery

There are two aspects of recovery. When a group of sites is partitioned from the rest of the sites, they will execute a protocol that

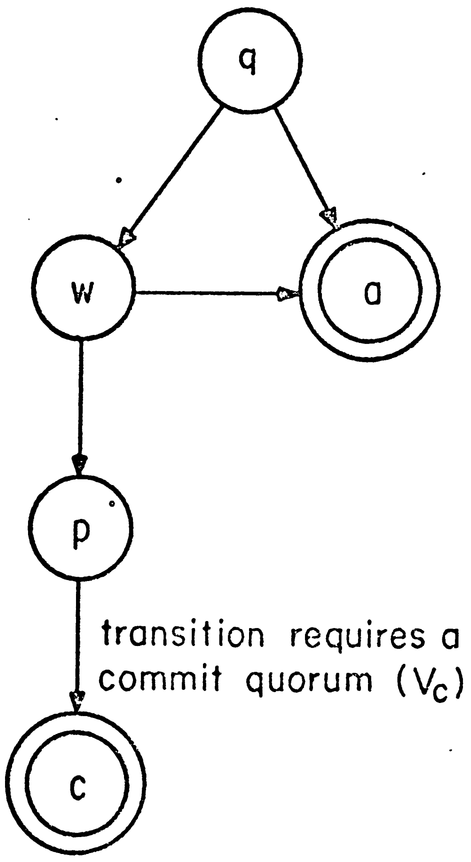

**Figure 4.** State diagram for the quorum based commit protocol.

attempts to form a quorum and terminate the transaction. These protocols, called _termination protocols_ are discussed in the first part of this section. If a quorum can not be achieved within the partition, then the sites must block until communication between partitions is restored. Once this is achieved, the sites within the new partition can execute a _merge protocol_ and reattempt terminating the transaction.

## Termination Protocol

As with the commit protocol, the major emphasis in the proposed protocol is on successful termination. Partially executed transaction will be aborted, when necessary, to achieve this goal.

When a group of sites detect that they are partitioned from the remainder of the network, they execute a two part termination protocol. The first part consists of electing a _surrogate coordinator_ and the second part consists of an attempt to form a quorum.

There are several possible election protocols. We will not explicitly discuss election protocols except to note that it is possible to elect a unique coordinator at linear cost ([GARC81,HAMM79]). The resiliency of a quorum-based protocol is not dependent on the uniqueness of the outcome of the election. Even if two surrogates are chosen, resiliency is guaranteed but performance suffers.

When the election completes the surrogate executes a protocol similar to the commit protocol in the previous section. The termination protocol is slightly more complex for two reasons. First, a surrogate works with less knowledge than the original coordinator, specifically, the surrogate may not know if a transaction is committable. Second, whereas there was a single coordinator originally, there many be many surrogates each operating in different partitions.[1]

For the first problem, a surrogate can attempt to form a commit protocol only if a site within the partition is in the committable state. For the second problem, a surrogate must explicitly form abort quorums. A site indicates its willingness to participate in an abort quorum by moving into a _prepared to abort_ state.

The termination protocol is given in Figure 5. Like the commit protocol, it consists of three phases. In the first phase the surrogate coordinator polls the sites about their local state, and these replies determine the action taken in the next two phases. If any site has committed (aborted), then the transaction is immediately committed (aborted) at all sites. Otherwise, the surrogate will attempt to establish a quorum.

A commit quorum is possible if at least one site is in the _prepared to commit_ state and the sum of the weights of the sites occupying the _prepared to commit_ state and the _wait_ states is at least $V_C$. If this is the case, the surrogates will attempt to move all sites in the _wait_ state into the _prepared to commit_ state. Barring additional failures, the surrogate will then commit the transaction.

---

[1]Or even in the same partition if the election protocol fails to uniquely elect a surrogate.

**COORDINATOR**

(1) Request local state.

(2) **slave responses** — **coordinator's actions**

| **slave responses** | **coordinator's actions** |
|---|---|
| $\geq 1$ commit | send commit; terminate |
| $\geq 1$ abort | send abort; terminate |
| $\geq 1$ prepared to commit and weights of wait and prepared to commit $\geq V_C$ | send prepare to commit continue with (3a) |
| weights of wait and prepared to abort $\geq V_A$ | send prepare to abort continue with (3b) |

(3a) if $\geq V_C$ ack's then send commit
else block

(3b) if $\geq V_A$ ack's then send abort
else block

(Slaves respond with their local state in Phase 1 and with an acknowledgement in Phase 2).

**Figure 5.** The quorum-based termination protocol.

However, additional failures may prevent sites either from making the transition or from acknowledging the transition. If an insufficient number of acknowledgements is received, then the protocol blocks.

An abort quorum is possible if the sum of the weights of sites occupying the _wait_ state and the _prepared to abort_ state is at least $V_A$. Unlike a commit quorum, an abort quorum does not require any sites to occupy the _prepared to abort_ state. Again, the surrogate attempts to move an abort quorum of sites into the _prepared to abort_ state -- aborting the transaction if it is successful, blocking otherwise.

The state transition diagram is given in Figure 6. A heavy line indicates the normal movement of the site into a "prepared" state and then into the corresponding final state. A dashed line indicates a path taken when the site is not a participant in the formation of the quorum.

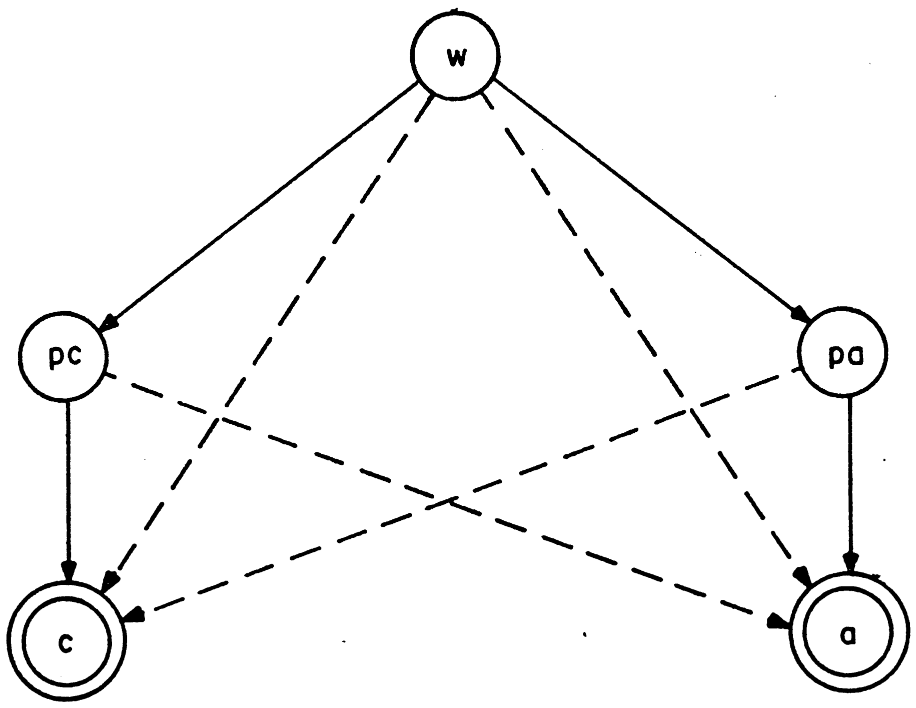

**Figure 6.** State diagram for the termination protocol.

## Merging

Partition merging occurs whenever a failure is repaired and communication is established between two or more partitions. We assume that the re-establishment of communication paths is detectable[2].

The recovery strategy for merging is simple: execute the termination protocol described in the last section. In this case the election process can be streamlined -- the new coordinator can be chosen from among the old coordinators, e.g. let the coordinator with lowest site number become the new coordinator. The new coordinator then executes the three phases in the second part of the termination protocol.

Site recovery is equally simple -- it is a special case of merging where one partition contains a single site.

## 5. Performance

It is very difficult to analyze the expected performance of quorum-based protocols, even if very simple and independent probability distribution functions are used to describe site failures. For nonzero failure probabilities, it is clear that the worst case performance is unbounded, which is expected from the results of the Two Generals Problem (see [GRAY79] for an description of this problem and its ramifications).

---

[2]A low level protocol can periodically attempt communication with other sites. Eventually it will detect the repair of the partition.

However, we argue that if all partitions are eventually resolved, then the protocols will eventually terminate. They are acyclic, hence every state transition moves a site closer to termination, and they are deadlock free. This latter property is assured by the choice for the quorum sizes -- after the merging of all partitions, it must be the case that either an abort quorum or a commit quorum can be formed.

In environments where failures are rare, the most important cost measure is the cost of the commit protocol in the absence of failures. The quorum-based commit protocol requires 3 phases, 5 end-to-end message delays, and about 5N messages (where N is the number of participants). This cost is substantially higher than the cost of the two phase commit protocol -- higher by approximately 50%. However, the two-phase protocol is not very resilient. A more resilient protocol, specifically one that is resilient to a coordinator failure, requires at least three phases. While several three phase protocols are known ([GARC79, SKEE81b]), the quorum-based protocol is the only one resilient to network partitioning.

There are two sets of parameters that determine the performance of the protocol in the presence of failures: the weights assigned to individual sites, and the values for $V_C$ and $V_A$.

The assignment of weights is often influenced by policy considerations external to implementation of the system. However, some factors that are relevant to performance are percentage downtime, failure rate, and percentage of data stored at the site. The most intuitive rule is to assign weights inversely proportional to the percentage downtime.

In choosing quorum sizes, it is not necessary for $V_C$ to equal $V_A$. In fact, there are several strong arguments for choosing $V_C>V_A$. One argument concerns protocols allowing unilateral aborts: if a significant number of transactions are unilaterally aborted, then clearly $V_A$ should be smaller. A stronger argument is that most site failures are expected to occur during Phase 1 of the commit protocol since most of the transaction execution time is spent in Phase 1. This phase is time consuming because the majority of the data processing takes place during it; whereas, Phase 2 and Phase 3 synchronize state information among the sites and require very little local processing. If sites fail during Phase 1, then the transaction must be aborted -- hence, it should be easy to abort.

An interesting heuristic for choosing $V_A$ is based on a rough estimate of the failure distribution of the sites. This heuristic is useful in environments where site failures, rather than network partitions, predominate. Let $P(V_A)$ be the probability that at least an abort quorum is operational. $P(V_A)$ is a decreasing function in $V_A$. The point is to choose the maximum $V_A$ such that $V_A<=V_C$ and $P(V_A)$ exceeds a minimum level of desired availability.

As mentioned before, the weight of a site can be zero, in which case the site contributes nothing toward forming a quorum. (However, such a site can still unilaterally abort the transaction.) When designing a protocol, a zero-weighted site can be eliminated from all phases requiring the formation of a quorum. In the extreme case, where only a single site has a non-zero weight, a quorum based commit protocol degenerates into the standard two-phase protocol with all of its disadvantages. Specifically, all sites must block on the failure of the only nonzero weighted site (which is normally the coordinator).

## 6. Conclusion

The use of quorums is a standard recovery technique for handling network partitioning (even primary site schemes, e.g. [STON79], are a degenerate case of using quorums). We have presented a very general quorum-based commit protocol that can be used with both replicated and nonreplicated data. Unlike previous schemes it allows a single site to unilaterally abort the transaction.

Quorum-based protocols are resilient because a site is allowed to participate in only one type of quorum. Quorum sizes are carefully chosen such that the formation of both a commit and an abort quorum requires the participation of a common site. In this way mutual exclusion is assured -- only one type of quorum can be formed during the execution of a transaction. (However, it is possible for multiple occurrences of a single type of quorum to be formed. For example, since abort quorums are usually small, more than one can be formed concurrently.) In such a scheme the concurrent execution of several coordinators, even if they are within the same partition, does not destroy consistency.

When a new coordinator is elected in the proposed recovery protocol, it polls all sites about their current local state. In making a commit decision, only the replies from the latest poll is used -- information obtained in earlier polls is ignored. Less conservative approaches which uses previous information can be found in [SKEE81c].

## REFERENCES

[ALSB76] Alsberg, P. and Day, J., "A Principle for Resilient Sharing of Distributed Resources," *Proc. 2nd International Conference on Software Engineering*, San Francisco, Ca., October 1976.

[GARC79] Garcia-Molina, Hector, Ph.D. Thesis, Stanford University, 1979.

[GARC81] Garcia-Molina, Hector, "Elections in a Distributed Computing System," TR No. 280, Princeton University, December, 1980.

[GIFF79] Gifford, David, "Weighted Voting for Replicated Data" *Operating Systems Review*, 13, 5, Dec., 1979, pp. 150-9.

[GRAY79] Gray, J. N., "Notes on Database Operating Systems," in *Operating Systems: An Advanced Course*, Springer-Verlag, 1979.

[HAMM79] Hammer, M. and Shipman, D., "Reliability Mechanisms for SDD-1: A System for Distributed Databases," Computer Corporation of America, Cambridge, Mass., July 1979.

[LAMP76] Lampson, B. and Sturgis, H., "Crash Recovery in a Distributed Storage System," Tech. Report, Computer Science Laboratory, Xerox Parc, Palo Alto, California, 1976.

[LIND79] Lindsay, B.G. et al., "Notes on Distributed Databases," IBM Research Report, no. RJ2571 (July 1979).

[SKEE81a] Skeen, D. and M. Stonebraker, "A Formal Model of Crash Recovery in a Distributed System," _IEEE Transactions on Software Engineering_, (to appear).

[SKEE81b] Skeen, D., "Nonblocking Commit Protocols," _SIGMOD International Conf. on Management of Data_, Ann Arbor, Michigan, 1981.

[SKEE81c] Skeen, D., "Crash Recovery in a Distributed Database System," Ph.D. Thesis, University of California, Berkeley (in preparation).

[STON79] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies in Distributed INGRES," _IEEE Transactions on Software Engineering_, May 1979.

[THOM79] Thomas, Robert, "A Majority Consensus Approach to Concurrency Control," _Transactions on Database Systems_, 4, 2, June 1979.