2009

Generalization of ACID Properties

Brahim Medjahed

Mourad Ouzzani

Ahmed Elmagarmid
Purdue University, ake@cs.purdue.edu

Generalization of ACID Properties

Brahim Medjahed
Department of Computer & Information Science, The University of Michigan - Dearborn
http://www.engin.umd.umich.edu/∼brahim
Mourad Ouzzani
Cyber Center, Purdue University
http://www.cs.purdue.edu/homes/mourad/
Ahmed K. Elmagarmid
Cyber Center and Department of Computer Science, Purdue University
http://www.cs.purdue.edu/homes/ake/

SYNONYMS

Advanced Transaction Models; Extended Transaction Models

DEFINITION

ACID (Atomicity, Consistency, Isolation, and Durability) is a set of properties that guarantee the reliability of database transactions [2]. ACID properties were initially developed with traditional, business-oriented applications (e.g., banking) in mind. Hence, they do not fully support the functional and performance requirements of advanced database applications such as computer-aided design, computer-aided manufacturing, office automation, network management, multidatabases, and mobile databases. For instance, transactions in computer-aided design applications are generally of long duration and preserving the traditional ACID properties in such transactions would require locking resources for long periods of time. This has lead to the generalization of ACID properties as Recovery, Consistency, Visibility and Permanence. The aim of such generalization is to relax some of the constraints and restrictions imposed by the ACID properties. For example, visibility relaxes the isolation property by enabling the sharing of partial results and hence promoting cooperation among concurrent transactions. Hence, the more generalized are ACID properties, the more flexible is the corresponding transaction model.

MAIN TEXT

ACID is an important concept of database theory. It defines four properties that traditional database transactions must display: Atomicity, Consistency, Isolation, and Durability. *Atomicity* states that transactions must follow an “all or nothing” rule. Either all of the changes made by a transaction occur or none of them do. The two-phase commit protocol (2PC) is generally used in distributed databases to ensure that each participant in a transaction agrees on whether the transaction should be committed or not. *Consistency* means that transactions always operate on a consistent view of the database and leave the database in a consistent state. A database is said to be consistent as long as it conforms to a set of invariants, called *integrity constraints. Isolation* gives the illusion that each transaction is executed alone. It ensures that the effects of a transaction are invisible to other concurrent transactions until that transaction is committed. Concurrency control protocol are generally implemented in database systems to preserve this property. *Durability* states that once a transaction is committed, its effects are guaranteed to persist even in the event of subsequent failures. This is usually achieved using database backups and transaction logs.

The limitations inherent to the original ACID properties and the peculiarities of advanced database applications has lead to the generalization of ACID properties as Recovery, Consistency, Visibility and Permanence. *Recovery* refers to the ability to take the database to a state that is considered correct in case of failure. *Consistency* refers to the correctness of the state of the database that a committed transaction produces. *Visibility* refers to the ability of one transaction to see the results of another running transaction. *Permanence* refers to the ability of a transaction to record its results in the database. The flexibility of a transaction model depends on the way

ACID properties are generalized. An extensive coverage of advanced transaction models is presented in [1]. *Sagas*, *Nested Transactions*, and *Flex* are representative examples of models that generalize ACID properties while *ACTA* is an example of a formal framework to express these models and reason about them.
A *Saga* is a chain of transactions that is itself atomic. Isolation is relaxed at the level of a Saga and visibility is permitted at the component transaction boundaries. A saga itself is atomic but because of the relaxed visibility, it supports semantic consistency. That is, each transaction in the chain is assumed to have a semantic inverse, or compensation, transaction associated with it. If one of the transactions in the saga fails, the transactions are rolled back in the reverse order of their execution. Committed transactions are rolled back by executing their corresponding compensation transactions.
Nesting allows concurrency within a transaction and provides fine-grained and hierarchical control for failure handling. The original nested transaction model, which supports only closed sub-transactions, was extended to include open sub-transactions. Closed sub-transactions may not support consistency and durability. A closed sub-transaction commits its results to its parent. These partial results are externalized only after the top (root) transaction commits, thus ensuring atomicity and isolation of the whole transaction. Because of its relaxed visibility, open sub-transactions directly externalize their results and expose them to other transactions.
The *Flex Transaction Model* has been proposed to generalize ACID properties in multidatabase systems. It relaxes the atomicity and isolation properties of nested transactions to provide users increased flexibility in specifying their transactions. A Flex transaction may proceed and commit even if some of its sub-transactions fail. It also allows the specification of dependencies on sub-transactions as internal or external dependencies. Internal dependencies define the execution order of sub-transactions, while external dependencies define the dependency of a sub-transaction execution on events (such as the start/end execution times) that do not belong to the transaction. The Flex model also enables users to control the isolation granularity of a transaction through the use of compensating sub-transactions.
ACTA is a framework that facilitates the formal description of properties of extended transaction models. It defines constructs that facilitate the synthesis of extended transaction models by tailoring/combining existing models or starting from first principles. Different notions are introduced in ACTA to enable the generalization of ACID properties from different perspectives. For instance, the notion of delegation allows transactions to selectively abort some of the operations it has performed and yet commit.

CROSS REFERENCE

Transaction Management
Concurrency Control
Serializability
Recovery
Two-Phase Commit
ACID
Extended Transaction Models
ACTA
Distributed, Parallel and Networked Databases

RECOMMENDED READING

[1] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, 1992.
[2] J. Gray and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.