WIKIPEDIA

Database transaction

A **transaction** symbolizes a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in a database. Transactions in a database environment have two main purposes:

1. To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
2. To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

A database transaction, by definition, must be atomic, consistent, isolated and durable.[1] Database practitioners often refer to these properties of database transactions using the acronym ACID.

Transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a database must either complete in its entirety or have no effect whatsoever. Further, the system must isolate each transaction from other transactions, results must conform to existing constraints in the database, and transactions that complete successfully must get written to durable storage.

Contents

Purpose

Databases and other data stores which treat the integrity of data as paramount often include the ability to handle transactions to maintain the integrity of data. A single transaction consists of one or more independent units of work, each reading and/or writing information to a database or other data store. When this happens it is often important to ensure that all such processing leaves the database or data store in a consistent state.

Examples from double-entry accounting systems often illustrate the concept of transactions. In double-entry accounting every debit requires the recording of an associated credit. If one writes a check for $100 to buy groceries, a transactional double-entry accounting system must record the following two entries to cover the single transaction:

1. Debit $100 to Groceries Expense Account
2. Credit $100 to Checking Account

A transactional system would make both entries pass or both entries would fail. By treating the recording of multiple entries as an atomic transactional unit of work the system maintains the integrity of the data recorded. In other words, nobody ends up with a situation in which a debit is recorded but no associated credit is recorded, or vice versa.

Transactional databases

A **transactional database** is a DBMS where that provides the ACID properties for a bracketed set of database operations (begin-commit). All the write transactions within a transaction have an all-or-nothing effect, that is, either the transaction succeeds and all updates take effect, or otherwise, the database is brought to a state that does not include any of the updates of the transaction. Transactions also take care that the effect of concurrent transactions satisfies certain guarantees known as isolation level. The highest isolation level is serializability that guarantees that the effect of concurrent transactions is equivalent to a serial (i.e. sequential) execution of them.

Most modern relational database management systems fall into the category of databases that support transactions. A new category of data stores known as NoSQL data stores has emerged during the last decade. NoSQL data stores pursue scalability and due to the lack of scalable solutions for transactional processing they renounced to provide transactions and therefore to guarantee data consistency in the advent of updates and concurrent accesses.

In a database system a transaction might consist of one or more data-manipulation statements and queries, each reading and/or writing information in the database. Users of database systems consider consistency and integrity of data as highly important. A simple transaction is usually issued to the database system in a language like SQL wrapped in a transaction, using a pattern similar to the following:

1. Begin the transaction
2. Execute a set of data manipulations and/or queries
3. If no errors occur then commit the transaction and end it
4. If errors occur then roll back the transaction and end it

If no errors occurred during the execution of the transaction then the system commits the transaction. A transaction commit operation applies all data manipulations within the scope of the transaction and persists the results to the database. If an error occurs during the transaction, or if the user specifies a rollback operation, the data manipulations within the transaction are not persisted to the database. In no case can a partial transaction be committed to the database since that would leave the database in an inconsistent state.

Internally, multi-user databases store and process transactions, often by using a transaction ID or XID.

There are multiple varying ways for transactions to be implemented other than the simple way documented above. Nested transactions, for example, are transactions which contain statements within them that start new transactions (i.e. sub-transactions). *Multi-level transactions* are a variant of nested transactions where the sub-transactions take place at different levels of a layered system architecture (e.g., with one operation at the database-engine level, one operation at the operating-system level). [2] Another type of transaction is the compensating transaction.

In SQL

Transactions are available in most SQL database implementations, though with varying levels of robustness. (MySQL, for example, began supporting transactions from version 5.5, with the switch to the InnoDB storage engine. The previously used storage engine, MyISAM did not support transactions.)

A transaction is typically started using the command `BEGIN` (although the SQL standard specifies `START TRANSACTION`). When the system processes a `COMMIT` statement, the transaction ends with successful completion. A `ROLLBACK` statement can also end the transaction, undoing any work performed since `BEGIN TRANSACTION`. If autocommit was disabled using `START TRANSACTION`, autocommit will also be re-enabled at the transaction's end.

One can set the isolation level for individual transactional operations as well as globally. At the READ COMMITTED level, the result of any work done after a transaction has commenced, but before it has ended, will remain invisible to other database-users until it has ended. At the lowest level (READ UNCOMMITTED), which may occasionally be used to ensure high concurrency, such changes will be visible.

Object databases

Relational databases traditionally comprise tables with fixed size fields and thus records. Object databases comprise variable sized blobs (possibly incorporating a mime-type or serialized). The fundamental similarity though is the start and the commit or rollback.

After starting a transaction, database records or objects are locked, either read-only or read-write. Actual reads and writes can then occur. Once the user (and application) is happy, any changes are committed or rolled-back atomically, such that at the end of the transaction there is no inconsistency.

Distributed transactions

Database systems implement distributed transactions[3] as transactions accessing data over multiple nodes. A distributed transaction enforces the ACID properties over multiple nodes, and might include systems such as databases, storage managers, file systems, messaging systems, and other data managers. In a distributed transaction a there is typically an entity coordinating all the process to ensure that all parts of the transaction are applied to all relevant systems.

Transactional filesystems

The Namesys Reiser4 filesystem for Linux[4] supports transactions, and as of Microsoft Windows Vista, the Microsoft NTFS filesystem[5] supports distributed transactions across networks.

See also

- Concurrency control

References

1. A transaction is a group of operations that are atomic, consistent, isolated, and durable (ACID). (http://msdn.microsoft.com/en-us/library/aa366402(VS.85).aspx)
2. Beeri, C., Bernstein, P.A., and Goodman, N. A model for concurrency in nested transactions systems. Journal of the ACM, 36(1):230-269, 1989
3. Özsu, M. Tamer; Valduriez, Patrick. *Principles of Distributed Database Systems, Third Edition - Springer* (https://link.springer.com/book/10.1007/978-1-4419-8834-8). doi:10.1007/978-1-4419-8834-8 (https://doi.org/10.1007%2F978-1-4419-8834-8).
4. namesys.com (http://namesys.com/v4/v4.html#committing)
5. "MSDN Library" (https://msdn.microsoft.com/en-us/library/ff361664(v=vs.110).aspx). Retrieved 16 October 2014.

Further reading

- Philip A. Bernstein, Eric Newcomer (2009): *Principles of Transaction Processing*, 2nd Edition (http://www.elsevierdirect.com/product.jsp?isbn=9781558606234), Morgan Kaufmann (Elsevier), ISBN 978-1-55860-623-4
- Gerhard Weikum, Gottfried Vossen (2001), *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*, Morgan Kaufmann, ISBN 1-55860-508-8

External links

- c2:TransactionProcessing
- https://docs.oracle.com/database/121/CNCPT/transact.htm#CNCPT016
- https://docs.oracle.com/cd/B28359_01/server.111/b28318/transact.htm

Retrieved from "https://en.wikipedia.org/w/index.php?title=Database_transaction&oldid=803410500"