

Preface

The Concept

The processing of transactions on databases is a well-established area, and many of its foundations were already laid in the late 1970s and early 1980s, most notably in a series of articles by the architects of the System R relational database management system. Most of the textbooks on database management contain chapters on transaction processing, concurrency control, and recovery from failures. Textbooks devoted solely to transaction processing include the classic works by Bernstein et al. [1987] and Weikum and Vossen [2002] on the theoretical side and that by Gray and Reuter [1993] on the implementation side.

Transactions are a concept related to the logical database as seen from the point of view of a database application programmer. A transaction is a sequence of actions on the logical database that, if declared as committed, should have its effects recorded permanently in the database and otherwise have no permanent effect on the logical database. The database system must be able to ensure that updates by committed transactions persist and that updates by uncommitted transactions are rolled back. System R also introduced the important database-programming paradigm of partial rollbacks, with which a transaction can roll back only the most recently executed actions, back to a preset savepoint, and then continue as a forward-rolling transaction.

The actions of transactions are executed by concurrently running server-process threads on pages in the underlying physical database, permanently stored on disk but parts of it buffered in main memory during access. A delicate interplay is needed between the management of transactions at the logical database level and the management of page accesses at the physical database level. In many cases the keys of the data items to be accessed by a transaction are determined only after accessing and latching the pages that hold them, after which appropriate logical key-range locks can be acquired to protect the logical action against forbidden accesses by other concurrently running transactions.

Index structures such as B-trees are used to accelerate key-based access to data pages that store the data items of the logical database. The consistency and balance of the indexes must be maintained under insertions and deletions by concurrent multi-action transactions as well as in the event of process failures and system crashes. The management of recoverable index structure modifications, although documented in the research literature, is a topic usually ignored or treated inadequately in database textbooks.

The Focus

Our aim in this textbook is to bridge the gap between the theory of transactions on the logical database and the implementation of the actions on the underlying physical database. In our model, the logical database is composed of a dynamically changing set of data items (tuples) with unique keys, and the underlying physical database is a set of fixed-size data and index pages on disk, where the data pages hold the data items. We include total and partial rollbacks explicitly in our transaction model and assume that the database actions of transactions are tuple insertions, tuple deletions, tuple updates, and key-range scans on a relational database (or any database conforming to our model) and that server-process threads execute these actions on the data pages.

For efficient access to the tuples in the database, we usually assume that the logical database is stored in the leaf pages of a sparse primary B-tree index. Fine-grained (tuple-level) concurrency control is used to synchronize actions of transactions and short-duration latching for synchronizing page accesses, so that a database page can contain uncommitted updates by several concurrently running active transactions at the same time and uncommitted updates by an active transaction can migrate from one page to another due to structure modifications (page splits and merges) caused by concurrent transactions.

The B-tree is the most widely used index structure, because of its ability to accelerate key-range scans and to retain its balance under insertions and deletions. In addition to the traditional B-tree, we also consider index structures optimized for write-intensive workloads. Such workloads are more and more common in modern web services, and for such situations the basic update-in-place strategy of B-trees may not be efficient enough.

Unlike the abstract read-write model of transactions used in many textbooks, our logical database model more closely represents actions generated from SQL statements. This model allows for a rigorous treatment of isolation anomalies, such as unrepeatable reads (especially phantoms), and of fine-grained concurrency control protocols, such as key-range locking, that prevent such anomalies. As the basis of presentation of transactional isolation, we adopt the isolation-anomaly-based approach of Berenson et al. [1995], rather than the classical serializability theory, thus embracing also the cases in which transactions are run at isolation levels lower than serializability. The use of lower isolation levels such as “read committed”

(which permits unrepeatable reads) may be essential for the performance of very-high-rate transaction processing when full isolation (serializability) is not absolutely necessary.

The treatment of transaction processing in this book builds on the “do-redo-undo” recovery paradigm used in the ARIES family of algorithms designed by Mohan et al. [1992a] and nowadays employed by virtually all database engines. With this paradigm, correct and efficient recovery from a system crash is achieved by repeating first the recent transaction history by redoing physically the missing updates by all transactions from the log records saved on disk and only then undoing (physically or logically) the updates by the transactions that were active at the time of the crash.

The standard physiological write-ahead logging (WAL) protocol and the steal-and-no-force buffering policy are assumed. The forward-rolling insert, delete, and update actions of transactions are logged with redo-undo log records and their undo actions with redo-only log records, both containing the page identifier of the updated page besides the logical tuple update to make possible physical redo and physical or logical undo. In this setting, with fine-grained concurrency control, correct recovery could not be achieved using the opposite “do-undo-redo” recovery paradigm in which the active transactions are first rolled back followed by the redoing of missing updates by committed transactions.

All the methods and algorithms presented in this book are designed carefully to be compatible with do-redo-undo recovery, write-ahead logging, steal-and-no-force buffering, and fine-grained concurrency control, even though in some cases we adopt an approach different from those previously proposed for ARIES-based algorithms. Most notably, in order to make the interplay between the logical and physical database levels as simple and rigorous as possible, we prefer redo-only structure modifications over redo-undo ones: we define each structure modification such as a page split or merge as a small atomic action that retains the structural consistency and balance of an initially consistent and balanced index.

The Topics

Chapters 1–6 constitute the minimum of topics needed to fully appreciate transaction processing on a centralized database system within the context of our transaction model: ACID properties, fixing and latching of pages, database integrity, physiological write-ahead logging, LSNs (log sequence numbers), the commit protocol, steal-and-no-force buffering, partial and total rollbacks, checkpoints, the analysis, redo and undo passes of ARIES recovery, performing and recovering structure modifications, isolation anomalies, SQL isolation levels, concurrency control by key-range locking, and prevention of deadlocks caused by interplay of logical locks and physical latches. In these chapters, we do not yet give algorithms for index management but only explain the principle of atomically executed redo-only structure modifications that commit independently of the triggering transactions.

Chapters 7 and 8 include detailed deadlock-free algorithms for reading, inserting, and deleting tuples in a relation indexed by a sparse B-tree, under the key-range locking protocol, and for performing structure modifications such as page splits and merges triggered by tuple insertions and deletions. The B-tree variant considered is the ordinary one without sideways linkings. Saved paths are utilized so as to avoid repeated traversals. Together with the preceding six chapters, these chapters minimally cover the topic of the subtitle of the book: management of the logical database and the underlying physical structure.

The remaining chapters cover additional material that we consider important and is in line with the topics of the preceding chapters. In Chap. 9 we extend the tuple-wise key-range locking protocol to more general hierarchies of different granularities (multi-granular locking) and consider methods for avoiding deadlocks arising from lock upgrades (update-mode locking) and for avoiding the acquisition of many read locks. In Chap. 10 we extend our transaction model with bulk actions that read, insert, delete, or update a larger set of tuples and show how these actions are performed efficiently on a B-tree. In Chap. 11 we consider an application of bulk insertions: constructing a secondary index for a large relation while allowing the relation to be updated by transactions at the same time.

Several database management systems of today use transient versioning of data items in order to avoid read locks on items to be read. In Chap. 12 we review issues related to versioning, such as snapshot isolation and versioned B-trees. In Chap. 13 we consider the management of transactions that access data partitioned or replicated at several database servers in a distributed system. We also briefly discuss the requirements that have led to the design of systems called “key-value stores” or “NoSQL data stores” used to implement large-scale web applications. In Chap. 14 we consider a database system model, called the page server, in which transaction processing occurs at client machines, on cached pages fetched from the server.

Chapter 15 is devoted to issues related to the management of write-intensive transactions. In many Internet applications and embedded systems, relatively short updating transactions arrive in high frequency; in such systems, the database may reside in main memory, but log records are still written to disk for durability. Group commit is used to accelerate logging, and random disk writes are avoided by collecting updates into large bulks to be written with single large sequential writes onto a new location on disk.

The Audience

This book is primarily intended as a text for advanced undergraduate or graduate courses on database management. A half-semester (6-week) course on transaction processing can be taught from Chaps. 1 to 6, possibly with selected topics from Chaps. 9 and 12 (such as basics of multi-granular locking and snapshot isolation), and the course can be extended to a full-semester (12-week) course by including all

of Chaps. 7–15. The students are assumed to have basic knowledge of data structures and algorithms and of relational databases and SQL.

Acknowledgements

This textbook has grown out from lecture notes on courses on database structures and algorithms, transaction processing, and distributed databases given by the authors for many years since the late 1990s through the 2010s at the Department of Computer Science, University of Helsinki, and at the Department of Computer Science and Engineering, Aalto University School of Science (formerly Helsinki University of Technology).

The lecture notes were originally written in Finnish. Earlier versions of the notes were commented and translated into English by Riku Saikkonen; much of his text still remains in this book. The \LaTeX figures were prepared by Aino Lindh. We also borrow somewhat from our articles written together with Tuukka Haapasalo, Ibrahim Jaluta, Jonas Lehtonen, Timo Lilja, and Riku Saikkonen. The contributions of Aino, Ibrahim, Jonas, Riku, Timo, and Tuukka are gratefully acknowledged. We also thank Satu Eloranta, Sami El-Mahgary, and Otto Nurmi for comments on the lecture notes.

The work was financially supported by the Academy of Finland.

Helsinki, Finland
Espoo, Finland
August 2014

Seppo Sippu
Eljas Soisalon-Soininen

Transaction Processing
Management of the Logical Database and its
Underlying Physical Structure
Sippu, S.; Soisalon-Soininen, E.
2014, XV, 392 p. 64 illus., Hardcover
ISBN: 978-3-319-12291-5