2005

A Sin of Omission: Database Transactions

David Hansen
George Fox University, dhansen@georgefox.edu

A Sin of Omission: Database Transactions

David M. Hansen
George Fox University
Newberg, OR 97132
dhansen@georgefox.edu

Abstract

Database courses are under increasing pressure to include new topics which inevitably leads to a decision about which topics are essential to the course and which can be omitted to make room for others. Recent surveys have indicated that many instructors are reducing or eliminating coverage of "transaction management" from their introductory database courses. As both an academic and a practitioner, I believe that this is a mistake. This paper discusses why the semantics of transaction management should be an integral topic in any introductory database management system course.

1 The Squeeze Is On

The introductory database management systems (DBMS) course, like many courses in the undergraduate curriculum, is being forced to change. Databases are an integral component in most every information system and, with the explosive growth of web-accessible information systems, courses in database management are increasingly incorporating topics related to managing and accessing data via the web. A recent survey of topics taught in the introductory DBMS course shows that the topics receiving the greatest increase in coverage include the Internet, client-server, web interfaces, and client tools. At the same time, topics such as system architecture, database design, and functional dependencies are seeing greatly decreased coverage[7].

While "transaction management" is not among those topics seeing greatly reduced coverage, Robbert and Ricardo's survey indicates that transaction management was already a topic taught by just 54% of the faculty surveyed in 2001 and only 45% of the faculty surveyed in 2002[7, p. 140] — a figure I find worrisome.

But if transaction management isn't being displaced as significantly as other topics, why the concern about transaction management in particular? The reason is that *the use of transactions is essential to the correct use of a DBMS*. One can ignore many of the underlying DBMS implementation details and formal database theory yet still be a capable DBMS user. However, students must be

taught how to properly understand and use transactions; failure to do so leads to common programming errors that we would never tolerate in other subjects and that continue to wreak havoc in deployed information systems.

2 Transaction Management

The transaction management component of a DBMS insures that a logical group of database operations are executed together in such a way as to preserve four properties:

1. **A**tomicity — all operations are completed or none are completed
2. **C**onsistency — the database moves from one "consistent state" to another
3. **I**solation — concurrent transactions do not affect one another
4. **D**urability — changes committed to the database can not be lost

These properties, known as the *ACID* properties of database transactions, are critical to the correct use of a DBMS. The most important guarantee of transactions is a consistent database state. To use a DBMS correctly in any non-trivial application, transactions must be properly understood and used[1].

2.1 Ignoring Transactions

The most common problem arising from the misuse of (or simply ignoring) database transactions is a race condition that can lead to an inconsistent database. As a simple example, suppose my wife and I both visit ATMs concurrently. Each of us withdraws $100 from our checking account. During my transaction, the ATM reads my balance b in one transaction and subtracts $100 from b. Meanwhile, the ATM my wife is using also reads the balance b and subtracts $100 from the balance. Both ATMs now write back the updated balance in update transactions. At this point the database is inconsistent as the balance *should* reflect the sum of all deposits minus all withdrawals. But our balance does not reflect all withdrawals as one has been lost. The savvy reader may argue that the problem that arises in this example can easily be fixed if we simply amend the update transaction to set the balance equal to the current balance minus the withdrawal:

[1]Compounding the problem, transaction management has been made transparent by many DBMSs. Some (unfortunately) popular DBMSs simply do not have multi-statement transaction semantics as they are not designed for concurrent access. As a practitioner, I find the use of these sorts of single-user "DBMS" products for course projects inappropriate; students fail to deal with and understand the ramifications of concurrent access to a shared database. Other DBMSs have made transactions transparent by supporting "automatic" transactions where, by default, each individual database query or update statement is treated as a single self-contained transaction; the user does not need to explicitly "begin" and "commit" transactions as the DBMS implicitly begins and commits a transaction for each statement. In both cases, the importance and correct use of transactions are being hidden from users of the DBMS.

```
update account set balance = balance - 100 where ...
```

While this is true for the trivial example above, it is not true of all, or even most sequences of database queries and updates. Many database updates include more substantial computations and thus the boundaries of the transaction **must** include retrieving the data from the database, performing computations in the application, and updating the database.

In addition to race conditions, many DBMSs allow users to relax concurrency constraints in ways that can lead to non-serializable transaction schedules. For example, users can choose to allow "dirty reads" which, when used properly, can be helpful in limited situations where precise answers are not critical and whose computation would adversely impact the operation of the database. However, users must understand that allowing dirty reads in general can likewise lead to an inconsistent database state. Suppose transaction T_1 reads uncommitted data written by transaction T_2 and uses that data to update the state of the database. If T_2 should abort instead of commit, then the data written by T_1 was based on a state of the database that technically never existed. DBMSs that allow dirty reads do so with the tacit understanding that the user will not use the data read to update the state of the database. This approach could be used by a bank, for example, to generate a report that includes a relatively close approximation for the sum of all account balances without impacting the ability of other transactions to continue to update the database while the report transaction was running over the entire database.

Concerns about the correctness of database access may seem somewhat academic, yet as a practitioner I have seen non-trivial, mission-critical database accessing application programs that have tolerated race-conditions. In one instance, developers I worked with knowingly sacrificed the guaranteed correctness of their applications to avert the *potential* for concurrency conflicts[2]; they were understandably concerned about the impact of a long-running update transaction on concurrent access to their database. Yet rather than find a way to address the concern while maintaining consistency, they coded in such a way as to introduce a race condition into their application. The developers felt that the occurrence of a race condition was *unlikely* to arise because of the way they *expected* users to access their database, yet they conceded that lost updates as the result of concurrent access to the same data would be a serious problem.

2.2 Ignorance is Not Bliss

As academics we spend a good deal of time asserting that algorithms **must** be correct. An algorithm that is incorrect, no matter how fast or elegant, is useless and may be worse than no solution at all. Most Operating Systems courses cover policies and mechanisms that ensure correct operation in the midst of concurrent access to shared resources. Yet when it comes to concurrent access to shared resources in database management systems, it seems that many instructors may

[2]I'm reminded of Knuth's observation that "premature optimization is the root of all evil."

be treating correctness, achieved via transaction management, as an optional topic.

To correct this deficiency, the semantics of transaction management should be taught and course-projects should be undertaken using a **real** DBMS with full support for explicit multi-statement transactions. Students should assume that most databases are deployed in multi-user environments where application programs concurrently query and update the database. Assignments should include the implementation of application programs that demonstrate the correct use of multi-statement database transactions. Students should also be aware of the potential costs and conflicts of database transactions and acquainted with strategies to deal with conflicts without sacrificing correctness.

2.3 Textbook Troubles

One difficulty in teaching students about transactions is that many textbooks cover the semantics of transactions together with mechanisms for transaction management and concurrency control[4, 8, 6]. This muddies the waters some by combining the fundamental concept of transaction semantics, which all users of a DBMS ought to know, with implementation details which are somewhat incidental. In one introductory textbook that is largely devoid of implementation details, Ullman and Widom still cover the semantics of transactions[9].

Instructors who do not wish to cover details about mechanisms (e.g., locking) and policies (e.g., serializability) must still cover transaction semantics. The semantics of transactions can be covered with little or no reference to mechanism and policy, focusing on the correctness of application programs that are guaranteed to take the database from one consistent state to another.

2.4 Compounding the Problem

Ironically, instructors that fail to cover transaction management while adding topics related to the Internet and web-data management are compounding the difficulties of managing transactions by introducing the complexity of stateless database interaction. While traditional application programs with embedded database access can group a logical sequence of database and program operations into a single database transaction, simple back-end web applications execute in a stateless environment where one program may read data from the database to be returned to the user while a subsequent program accepts input from the user and updates the database. The traditional single-transaction approach does not work in this environment[2]; other approaches, such as the introduction of middleware transaction-processing monitors[3], must be used to guarantee correctness in web-accessible information systems.

3 In Conclusion

I've worn a number of hats in my career, both as a practitioner in industry and now as an academic. As a practitioner **and** an academic, I conclude by urging every instructor teaching an introductory DBMS course to include coverage of the basics of transaction management. One need not delve into implementation or policy details, but every student should emerge with a solid understanding of the semantics of transactions and how they relate to the overall semantics of application programs that access the DBMS.

Yet the question remains of how to deal with the increasing pressure to include new topics into the introductory database curriculum — especially topics related to the Internet and web-accessible information systems. At George Fox University we recognized some years ago that we needed to address the broad topic of integrated web-accessible information systems. Our approach was to develop a hybrid course we call "Client-Server Systems" that introduces students to three-tier information systems. This course includes elements of networking via the Internet, web-interfaces, and client-server programming, along with limited coverage of DBMS topics. The course serves a number of purposes. First, it includes a non-trivial programming project that includes the construction of an end-to-end information system currently implemented using HTML, JavaScript, PHP, and SQL. Along the way we cover basic elements of web-specific Human Computer Interaction. While not a "database" course, we do include an introduction to SQL and the essentials of both database transactions and higher-level transactions that encapsulate user-interaction in the stateless environment of the web. This course has proved to be very popular and is often taken by Sophomores before they continue on to more traditional DBMS and Networking courses.

Whatever approach is taken to the incorporation of new topics into the curriculum, it is essential that students learn how to guarantee the correctness of application programs that concurrently access a DBMS. The semantics of transactions and transaction management must be an integral topic in any introductory database management system course.

References

[1] Elizabeth S. Adams, Mary Granger, Don Goelman, and Catherine Ricardo. Managing the introductory database course: what goes in and what comes out? In *SIGCSE '04: Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 497–498. ACM Press, 2004.

[2] Roger Barga, David Lomet, German Shegalov, and Gerhard Weikum. Recovery guarantees for internet applications. *ACM Trans. Inter. Tech.*, 4(3):289–328, 2004.

[3] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Branbilla, Sara Comai, and Maristella Matera. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 2003.

[4] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems*. Addison Wesley, 2004.

[5] CORPORATE The Joint Task Force on Computing Curricula. Computing curricula 2001. *J. Educ. Resour. Comput.*, 1(3es):1, 2001.

[6] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw Hill, 3rd edition, 2003.

[7] Mary Ann Robbert and Catherine M. Ricardo. Trends in the evolution of the database curriculum. In *ITiCSE '03: Proceedings of the 8th annual conference on Innovation and technology in computer science education*, pages 139–143. ACM Press, 2003.

[8] Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill, fourth edition, 2001.

[9] Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. Prentice Hall, 2nd edition, 2002.