

# A game-playing experience in the learning of database concurrency control

Vinícius Medina Kern, Maria do Rosário Stotz, and Merilyn Bento

**Abstract** — A usual undergraduate course on Databases covers several subjects, such as modeling, query languages, and database management techniques. Given the extension and complexity of the subjects, students frequently find the textbooks arid, and failing is not uncommon. This paper briefly presents important concepts of database concurrency control and introduces a card game designed to simulate database concurrent access control. The participation in the game helps students to sharpen their skills on concurrency control, and allows the professor to build connections with other courses, such as operating systems and object-oriented programming.

**Index Terms** — Concurrency control, database management, transaction processing, educational games.

## I. INTRODUCTION

Databases is a very important course for Computer Engineering and Computer Science majors. A usual undergraduate introductory course covers several subjects such as database modeling, query languages, and database management techniques such as failure recovery and concurrent access control.

Given the extension and complexity of the subjects, students frequently find the textbooks somewhat difficult and arid. Failing is not uncommon. In addition, as observed in this study, some students don't recognize the importance of the learning of database management techniques, claiming that most professional opportunities are not related to this subject.

These students don't understand what Computer Science is. This can be explained with the help of the ANSI/SPARC three-level architecture, illustrated in fig. 1.

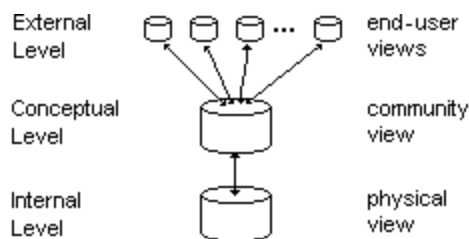


Fig.1. The ANSI/SPARC 3-level architecture

The ANSI/SPARC architecture, published in the 1970s [1], allowed for a division of work that caused a great progress in

the database area. According to it, any database can be viewed through different perspectives and their mappings:

- The **External Level** includes a number of end-user views or schemas, representing the database as perceived by each end-user.

- The **Conceptual Level** is the community view, describing the structure of the database for the whole group of users.

- The **Internal Level** is the physical view, describing “the complete details of data storage and access paths for the database” [2].

Most database-related opportunities in the area in which this experiment was conducted (Santa Catarina State, Brazil) deal with the building of the external and conceptual levels, and the mapping between them. Modelers analyze data requirements from end users and map them into the database design. A database design document is an expression of the conceptual level. Programmers and analysts design, code and implement computer solutions for user views, mapping user-view data structures from the conceptual schema.

Database administrator (DBA) positions are less common than modeler, programmer, and analyst positions. They require knowledge about the external and conceptual levels, and proficiency in database management techniques (related to the internal level) in order to make good technological decisions using a database management system (DBMS).

However, DBMSs translate conceptual schemas into physical schemas automatically. The only professional who actually builds internal level routines for **general-purpose** [2]-[4] database systems is a computer scientist who works for one of the companies that make DBMSs. These positions are scarce, available mostly in the United States. Nevertheless, database management techniques are indispensable in the education of Computing majors.

This article presents a brief introduction to the learning of database concurrency control, a critical database management technique. A card game intended for the learning of concurrency control is introduced. Results from the application of this game are reported. In the conclusion, consequences of the introduction of this game in classroom are discussed, and opportunities for improvement and extensions are pondered.

## II. THE LEARNING OF DATABASE CONCURRENCY CONTROL

Databases is usually in the middle of undergraduate courses. A typical database *curriculum* favors the most common techniques, approaching the learning of database concurrency control within the following scope:

Manuscript received on December 15, 2001. (Deadline date).

V. M. Kern, UNIVALI at São José, Brazil, Computer Engineering and Computer Science, kern@eps.ufsc.br; M. R. Stotz, UNIVALI at Biguaçu, Brazil, Psychology, mrstotz@matrix.com.br; M. Bento, UNIVALI at Biguaçu, Brazil, Psychology, merilyn@big.univali.br.

- **Interleaved concurrency** in single-processor computers, in opposition to simultaneous concurrency in multi-processor computers, and
- **Short transactions**, usual in business applications in relational databases, but not always the case in engineering and other applications.

Database textbooks such as [2]-[4] present the concept of **transaction** – a logical unit of database processing – and its properties, known as ACID: atomicity, consistency preservation, isolation, and durability. A transaction is composed of operations, being the read and write operations the most important with regard to concurrency control, since they access database items.

For instance, any money withdrawal in an automatic teller machine corresponds to a transaction with a read-item operation (to access the account balance), and a write-item operation (to update the balance). Naturally, there are several other non-database operations such as counting money, magnetic card reading, calculations, and screen display.

The concurrent access control problem arises when more than one user accesses the same database (a very common and desirable situation). Database management software has to deal with the fact that the result of one transaction may affect other transactions. A failing transaction must not affect the database because of atomicity. Its operations must be undone, and this may cause the failure of other transactions, leading to cascade failure.

An appropriate and detailed **schedule** of transactions' operations must be produced. A schedule is appropriate if data integrity is maintained in the database, according to the consistency property. A simple approach would be to process one transaction at a time, from begin to end, therefore avoiding errors resulting from the order in which reads and writes from different transactions are processed.

This **serial schedule**, however, wastes valuable time. For instance, 50 simultaneous bank transactions would have to wait in line for processor time. In this case there would be no real database concurrency problem. A failing transaction X, say, a withdrawal, could be aborted and re-submitted later without causing any problem to the other 49 transactions.

If the 50 transactions access different database items, then there is no database concurrency. Instead of waiting in line, they could share disk and processor resources. If, however, any transaction checks balance or withdraws money from the same account accessed by transaction X, then that failing withdrawal might cause inconsistency.

An approach to avoid inconsistency and, at the same time, allow for the interleaving of operations of different transactions, is to guarantee that the schedule is **serializable** [5]. The characteristic of serializable schedules is that they cause results in the database that are equivalent to results from a serial schedule.

A popular kind of serializability is conflict-serializability. A schedule is conflict-serializable if it is conflict-equivalent to a serial schedule, i. e., if all pairs of **conflicting operations** in both schedules have the same precedence order. A pair of conflicting operations is made of read- or write-item operations with three characteristics:

- They access the same database item,
- They belong to different transactions, and
- At least one of them is a write-item operation.

Fig. 2 shows three alternative schedules for the simultaneous transactions T1, T2, and T3. Only reads and writes are represented (as r(item) and w(item), respectively).

A possible interpretation for the meaning of these transactions is: T1 is a money transfer between two bank accounts whose balances are stored in database items X and Y. T2 is a money withdrawal from the account whose balance is Y. T3 is a global balance check (say, X has the balance of a checking account, and Y has the balance of a savings account of the same bank customer).

The graphs in fig. 2 represent the transactions (nodes) and pairs of conflicting operations (edges). There are six pairs of conflicting operations: three between T1 and T2, one between T2 and T3, and two between T1 and T3. Each edge represents one or more pairs of conflicting operations, oriented from the transaction in which the operation is processed in first place to the transaction in which the operation is processed after that.

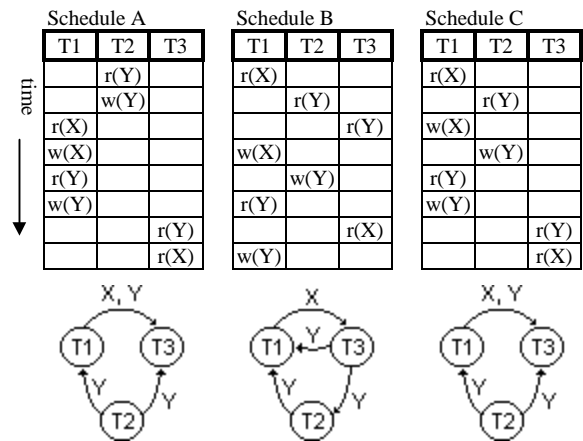


Fig. 2. Schedules for transactions T1, T2, and T3, with precedence graphs

Schedule A is serial (T2 → T1 → T3); therefore the order in which the operations are processed cannot introduce inconsistency in the database. However, performance is adversely affected since read and write operations are costly and processor time is wasted waiting for the operations to be terminated.

Schedule B favors performance. Processor time can be used to process database access operations while disk devices are still processing previous reads and writes. For instance, r(Y) in T2 may be scheduled soon after r(X) is scheduled in T1, even before it is finished on disk - a very time-consuming operation. However, schedule B is not conflict-equivalent to any serial schedule, therefore the results in the database may be inconsistent.

Schedule C is not serial, but the order in which conflicting operations are processed is the same as in schedule A. They are conflict-equivalent; schedule C is (conflict-) serializable.

It can be demonstrated that serializable schedules produce correct results in the database [5]. In summary, schedule C

produces correct results as if it was serial, and has the advantage of intercalating operations from different transactions, thus increasing concurrency and allowing for a better performance.

Once students recognize serializability as an interesting property of schedules, another important property is introduced: **recoverability** [6]. It is related to the question of how to preserve consistency in case of failure.

Every transaction ends with a COMMIT operation, unless it fails (and there are several reasons for failure). A schedule is recoverable if every transaction is committed only after all other transactions that wrote items read by the transaction about to commit are committed.

If a transaction fails, it ends with a ROLLBACK operation, implying that all operations must be undone to preserve atomicity. This may cause other transactions to fail, too. However, if those transactions are committed, they cannot be undone. That's the durability property of transactions.

Fig. 3 presents a serializable but non-recoverable schedule. The results in the database are correct if both transactions are successful. The problem is: if T1 fails (ends in ROLLBACK instead of COMMIT), then w(X) must be undone.

If w(X) in T1 is undone, then r(X) in T2 has read an incorrect value of X. T2 must be aborted, too. However, if T1 fails after T2 is committed, T2 cannot be rolled back (T2 is durable). The error is not recoverable. The only way to prevent this is to postpone the COMMIT of T2 (to after the COMMIT of T1).

	T1	T2
time ↓	w(X)	
		r(X)
		COMMIT
	w(Y)	
	ROLLBACK	

Fig. 3. A non-recoverable schedule

At this point, in classroom, students are dealing with several complex concepts without being able to respond to an important question: "How does a DBMS controls concurrent access?" They learned about serializability and recoverability, but how does the DBMS schedule concurrent transactions following these principles?

There are various approaches to concurrency control. The most commonly used are based on **locking**, i.e., the reservation of a database item for access by a single transaction, while other transactions must wait for the unlocking if they are to access the same item. In **multiple-mode locking**, an improvement of the binary locking just described, several transactions may share access to the same item if they are all reads, but write operations require exclusive locking.

Because the use of locks does not guarantee serializability, a two-phase locking protocol [7] is usually adopted. It consists of the restriction that all locks in a transaction be obtained before the first unlocking.

Fig. 4 illustrates a schedule for the same transactions of fig. 2. It can be simulated in class using the following time-sharing policy: each transaction has the opportunity to

schedule one read- or write-item operation and any number of other operations.

This time-sharing policy is not far from reality, since read- and write-item operation are costly because they usually require access to disk devices, while other operations (not represented in fig. 4) are relatively cheap in terms of processor time.

Every access to database items requires locking. Read-only access requires read locking (Rlock(item) in fig. 4); write-only or read-and-write access requires write locking (Wlock(item) in fig. 4).

	T1	T2	T3
1	Wlock(X)		
2	r(X)		
3		Wlock(Y)	
4		r(Y)	
5	w(X)		
6		w(Y)	
7		Unlock(Y)	
8		COMMIT	
9			Rlock(Y)
10			r(Y)
(T1 and T3 are in deadlock)			
11			ROLLBACK
12	Wlock(Y)		
13	r(Y)		
14	Unlock(X)		
15	w(Y)		
16	Unlock(Y)		
17	COMMIT		
18			T4
19			Rlock(Y)
20			r(Y)
21			Rlock(X)
22			r(X)
23			Unlock(Y)
24			Unlock(X)
25			COMMIT

Fig. 4. A schedule, showing locks and unlocks, according to the basic two-phase locking protocol

A **deadlock** problem occurs after line 10 in fig. 4, because T1 holds a lock on X and waits for T3 to unlock Y, and T3 holds a lock on Y and waits for T1 to unlock X. The time-sharing routine may keep allocating time to T1 and T3, but they can't continue.

The solution to break the deadlock is to abort (ROLLBACK) one of the transactions. There are a number of *criteria* [2] to make that choice. In fig. 4 the aborted transaction is T3.

A ROLLBACK operation includes the unlocking of any items the transaction may hold locks. Another transaction (T4 in fig. 4) is submitted later to do the work T3 couldn't do. It can be observed in fig. 4 that the effect on the database is the same produced by schedule C in fig. 2.

There are alternative approaches to the concurrency control technique just described. Students should learn about these, too. However, the complexity, extension, and number of technological alternatives are often a hindrance for students.

Database textbooks present the techniques in a somewhat descriptive way. Students in the experiment described here frequently ask for more examples to make sure they

understand the principles. They want to learn about “practical” aspects of databases, but some don’t recognize the subject as very “practical”. The division of work explained in Section I might give a hint about why this misperception occurs.

A reader of Date’s “Introduction to database systems” [4], one of the best-selling textbooks on Databases, says [8]:

“If you wish to quickly jump into database design (...) then this book is not for you. If, however, you want a thorough grounding in the principles and practice of database theory considered from an academic standpoint, then this book is highly recommended.”

In order to help students to get a better grasp of concurrency control theory and practice, a card game was designed. The next section describes that.

### III. LEARNING CONCURRENCY CONTROL WITH A CARD GAME

Fig. 5 shows 5<sup>th</sup>-semester students playing with cards and building a schedule for a given set of simultaneous transactions. They use their knowledge about transactions and concurrency control protocols to simulate the work of a transaction processing system.

The sequence of activities is simulated through a control card or *kanban* that is passed from student to student (this is “k”, being passed by student number 1 in fig. 5). The card exchange is similar to a procedure call. Each student has to decide what to do in his turn, based on the concepts he learned and the concurrency control protocol being used.

The game may be played by two or three students to more than ten, depending on how many tests and procedures are assigned to each player. It is possible to assign the same task to two or more students, growing the number of players involved to include all students in class.

Fig. 6 presents a flow representation for the scheduling of transactions according to the conservative two-phase locking protocol. This protocol is deadlock-free. A transaction must obtain all locks it needs before it can execute any read or write. In the experiment of fig. 5, the students were assigned the roles of time-sharing (1), operation service (2), locking execution (3), lock table maintenance (4), locking control (5), unlocking execution (6), and COMMIT and recoverability control (7).

The professor or mediator (“P” in figure 6) begins the game by offering a set of transactions. They are sequences of read- and write-item cards such as read(X), write(Z), etc., preceded by a start-transaction N. Depending on the protocol being used, ROLLBACK and other cards can also take part in the game.

The *kanban* is given to whoever performs time-sharing (student number 1 in fig. 5). The scheduling uses a “one read or write a time” policy, i. e., only one read or write can be scheduled in each turn of the *kanban*. Time-sharing begins by allocating processor time to transaction number 1.

Using their knowledge of concurrency control, students decide when to lock or unlock an item, when to COMMIT a transaction, and when to schedule a read or write operation,

disposing them adequately and producing a schedule somewhat similar to the one presented in figure 4. Fig. 5 shows the visual aspect of the scheduling.

When there are no more active transactions to schedule, the professor helps the students in a checking of accuracy, ending the game.

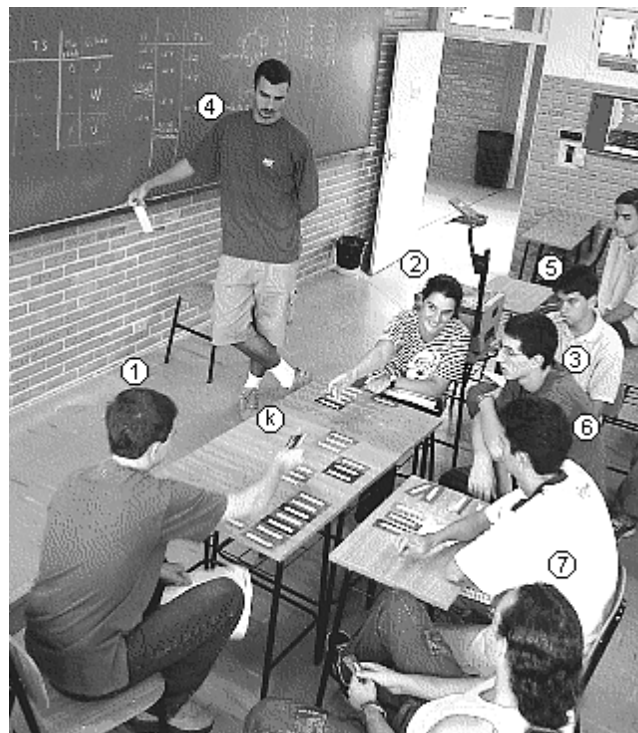


Fig. 5. Students at play

### IV. RESULTS

The introduction of the concurrency control game has impacted learning positively. Students’ average analytical skills improved with the game, according to the professor’s observation of tests and questions in classroom. It has been possible to reach a deeper understanding in the time available. The analysis and implementation of a program that simulates concurrency control, a task associated to the game, represents a significant increase in the size and complexity of the problems that the students are used to face.

The game has been used to simulate various concurrency control protocols and their variations. Locking has been simulated in its two forms: binary and multiple. Locking protocols used include the basic two-phase and variations: conservative and strict (the most popular according to [2]). The timestamp ordering protocol has been simulated, too. Simulations have been done with various types of schedules: recoverable, avoiding cascade rollback, and strict.

The specialist in Psychology in Education who assists the professor (see the acknowledgments) reported that some students think that the game is “boring after a while”. The professor regards this as a good sign – there is no fun in doing computer’s work. Incidentally, the schedule of transactions shown in figs. 2 and 4 using the conservative two-phase locking detailed in fig. 6 results not only

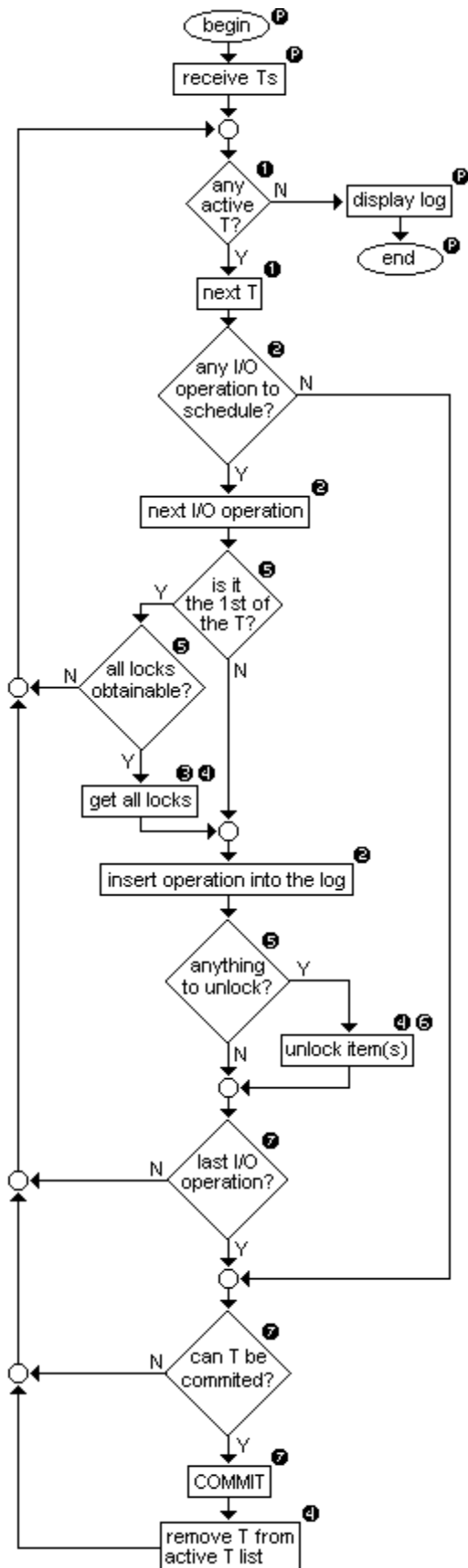


Fig. 6. A flow representation of the concurrency control game using conservative two-phase locking

serializable, but serial ( $T1 \rightarrow T2 \rightarrow T3$ ). The checking of this is left as a suggested exercise.

## V. CONCLUSION

This paper briefly discussed fundamental concepts of database concurrency control and introduced a card game for the simulation and learning of its concepts. The game's introduction promoted a faster and better learning of concurrency control techniques.

While playing with concurrency control, the professor has the opportunity to build links with other curricular courses:

- Programming languages, since there is a strong demand for good programming techniques;
- Data structures, especially when it comes to the design of data structures in the analysis and implementation assignment associated to the game.
- File organization, because the actual implementation of concurrency control routines uses file organization routines, and because the students need to understand that reads and writes are complex operations that can fail at any point of their execution, with consequences to concurrency control and failure recovery;
- Operating systems, since there is a lot in common between database and operating system concurrency. Besides, each read or write is a complex operation that demands calls to the operating system; and
- Software engineering, since the problem's complexity does not allow for an informal implementation approach.

## ACKNOWLEDGMENTS

Ms. Bento is a student of Psychology at UNIVALI at Biguaçu-SC, Brazil. She is assisting the first author as a Psychology of Education intern, under the supervising of Mrs. Stotz.

## REFERENCES

- [1] D. Tsichritzis and A. Klug (eds.), "The ANSI/X3/SPARC DBMS framework report of the study group on database management systems". *Information Systems* 3, pp. 173-191, 1978.
- [2] R. Elmasri and S. B. Navathe, *Fundamentals of database systems*, Addison-Wesley, 1994.
- [3] R. Ramakrishnan, *Database management systems*, 2<sup>nd</sup> edition, WCB/McGraw-Hill, 2000.
- [4] C. J. Date, *An introduction to database systems*, 7<sup>th</sup> edition, Addison-Wesley, 1999.
- [5] J. Gray, R. Lorie, and G. Putzulo, "Granularity of locks and degrees of consistency in a shared database", in G. Nijssen (ed.), *Modelling in data base management systems*, North-Holland, 1976.
- [6] D. Lomet and B. Salzberg, "Access method concurrency control with recovery", *ACM SIGMOD conf. on the management of data*, 1992.
- [7] K. Eswaran, J. Gray, R. Lorie, and I. Traiger, "The notions of consistency and predicate locks in a data base system", *Communications of the ACM* 19 (11), pp. 624-633, 1976.
- [8] Amazon.com. *Buying info: An introduction to database systems (Introduction to database systems, 7th ed)*. [Available online at [http://www.amazon.com/exec/obidos/ASIN/0201385902/qid=995256659/sr=1-1/ref=sc\\_b\\_1/002-9090914-7115218](http://www.amazon.com/exec/obidos/ASIN/0201385902/qid=995256659/sr=1-1/ref=sc_b_1/002-9090914-7115218), accessed 2001.07.12]