Teaching Concurrency Control and Recovery in Relational Databases

Luciano Leggieri Julian Berlín
University of Buenos Aires University of Buenos Aires
lleggieri@dc.uba.ar jberlin@dc.uba.ar

Advisor
Alejandro Eidelsztein
University of Buenos Aires
ae0n@dc.uba.ar

Abstract. We present a RDBMS Kanon, to provide an educational but fully functional database engine. It provides continuous visual feedback on the internal dynamics of those components responsible for the engine's transactional behavior. The recovery manager engine is based in ARIES, a popular family of recovery algorithms which focuses on a steal / no-force approach. We describe the features of Kanon and its technicalities.

1 Introduction

1.1 Goals

Kanon is intended to provide an educational database engine that shows those components responsible for the transactional behavior (Transaction Manager; Lock Manager) of any query asked to the database. It also shows the recovery work for those transactions that finished, but their changes were not made persistent yet.

An additional purpose of this work is to give to the Department of Computer Sciences [4] of our University [2] a tool able to show and teach how a relational database works, one which supports transactions and recovery techniques.

For a deeper understanding refer to the available Kanon DBMS Technical Report [7].

1.2 Our contribution

Today's databases do not usually show the concurrency process of their actions to the standard user, because they do not find that information to be interesting or useful. Some databases provide support to know what objects are being locked by each transaction, so that if a deadlock takes place then it is easy to know which the involved transactions were.

Kanon is a relational database engine with educational purposes; a tool that allows to see ***how*** a DBMS works.[1] This project focuses on showing the behavior of concurrency and transaction recovery. The recovery manager engine is based on the ARIES family of algorithms [8], which has desirable properties for a safe and efficient recovery method.

These aims gave us the requirement to build the engine as simple as possible, to document the implementation and to show, when the DBMS is being used, what it is currently doing, as well as the internal state of the system structures.

To program the basic components and modules, we considered what was learned during the databases course [3], and read from the course main texts ([12], [13] and [1]).

The Kanon Buffer Manager uses a **steal / no-force** scheme [12]. This scheme makes the manager implementation simpler, as there is no need for shadow copies. The locking system is based on Two Phase Locking and supports the four standard isolation levels defined in ANSI SQL/92.

The DBMS also includes simple hash based indexes to provide greater concurrence between queries executed in parallel. These queries are written in standard SQL, and the engine analyses and decomposes the query to execute it on the data base records.

In order to visualize the engine operation, and to provide a way to execute queries and see their results, a special client was built [6], which allows many concurrent connections to the server, and to execute several queries at the same time. This client also reports the state of whatever object is locked or released during the course of each transaction, and shows the log events of each operation within those transactions.

For the engine design, we departed from classic procedural algorithms showed in several papers, to try an object-oriented approach. This includes the use of design patterns and a modular architecture for a greater understanding of each component, thus allowing component modifications to be independant of each other.

The chosen language to develop the RDBMS was Java for the great quantity of plugins existing to make graphical interfaces, as well as the simplicity of its language, power that is propitious for the accomplishment of educational works of this style. Also we emphasize to have made the implementation using Design Patterns among which we can mention: Factory, Strategy, Decorator, Singleton and Abstract factory [5].

2 Learning about Databases

2.1 Why?

Nowadays, relational databases systems have a very important role in the information management that is handled by millions of people everyday in their

[1] For greater detail on each topic of this paper, full documentation and implementation source code are available.

works and/or homes. With these systems it is possible to store information in a structured way so that later those data can be queried efficiently. Queries can be performed in order of millions per second, so it is necessary for the system to be capable of supporting these volumes of data in an efficient way. It also should provide mechanisms to maintain the data integrity and be able to recover data easily and quickly after a system crash, because today the information is critical in the majority of the systems used by either people or other machines.

For this reason, in almost all universities that offer courses or topics related to Computer Sciences or Software Engineering, they usually have a course dedicated exclusively to the study of databases engines and all the related theory behind them. The depth with which the course program is dictated depends on each university, but it is necessary to have the largest number of possible tools to allow anyone interested to learn every aspect of a database engine.

There is not much about database engines that are focused in learning how a transactional database system internally works in the current literature. This is the main reason why we believe that this project could contribute in an original way for any educator interested in offering a course of transactions and concurrency with a higher level of depth.

2.2 Introducing Kanon

In this paper, we introduce Kanon, a relational and transactional database system engine, that presents a way to know and learn how the integrated transaction manager and recovery manager works. Both components were designed and developed with a basis on the ARIES algorithm. ARIES is a widely used family of algorithms for transaction management and data recovery currently used in the industry. Hence, the students are able to learn modern and original transactional and recovery methods currently used by most modern database engines present in the industry.

This contribution will show the student how the access to database tables is done by the system, and how it is possible to perform concurrent modifications to those tables without getting any data corruption and knowing that each change done by a transaction will never be lost.

Kanon was designed to be easy for those interested in their source code to analyze it and understand its operation; it presents a modular architecture in which each component is well specified and encapsulated.

Concurrency Control The program displays the acquisition and subsequent release of locked objects to access any requested record, and what the difference is between each isolation level. The system shows how each of them (READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ and SERIALIZABLE) stands up for its definition by handling the releases of those locks taken before writing records. It can be seen how each rule found in the books holds, setting a straight order that does not influence in existing concurrent transactions. For example, READ COMMITTED isolation level says that a transaction

must not read any data written by any transaction that has not yet committed. To fulfill this requirement, before a transaction writes a record, it acquires an exclusive lock on that record. Then if another transaction wants to read that record's value, it needs to acquire a shared lock first. The lock manager makes that request to succeed after the exclusive lock has been released, and that happens when its owner transaction commits or rolls back. This procedure ensures that a transaction can only read a value that has been modified by a committed transaction.

Kanon allows the user to check this by executing two concurrent queries on the same record (and make sure that the writing happens first), and the look in the locking table how those requests are ordered and executed according to the established isolation level.

This is how the system behaves according to each isolation level:

- READ UNCOMMITTED: Does not acquire a shared lock before reading a record. It does acquire exclusive locks before insertions or updates. These locks are released when the transaction ends (or reassigned to the parent transaction if it exists).
- READ COMMITTED: Acquires a shared lock before reading a record and releases it right after the operations finishes. Also acquires exclusive locks before insertions or updates. These exclusive locks are released when the transactions ends (or reassigned to the parent transaction if it exists).
- REPEATABLE READ: Acquires shared locks before reading records and exclusive locks before insertions or updates. All locks are released when the transactions ends (or reassigned to the parent transaction if it exists).
- SERIALIZABLE: Like Repeatable Read, but also before reading a record it locks the related query index or the entire table if no one was used. This is in order to avoid "phantom readings".

In every isolation level, before making an insertion or update, all indexes are locked, so as readings made in SERIALIZABLE isolation level be aware of those modifications and are able to avoid the previously mentioned "phantom readings".

Remaining transaction properties A rollback is a must to guarantee a transaction atomicity. The system shows how each operation writes in the Transaction log what did it do, so as to be able to undo every change if the transaction must be rolled back. The GUI also shows that operations made during rollback phase are also saved in the log. This allows to avoid repeating steps if the system crashes during a rollback process.

Here lie ARIES fundamentals, which establish the system behavior when realizing update operations and their logging. Then the engine will be able to redo them to guarantee transaction durability. Figure 1 shows Kanon in action: the client is running and it shows in its main windows the tables that record the lock and log events happening in real-time.

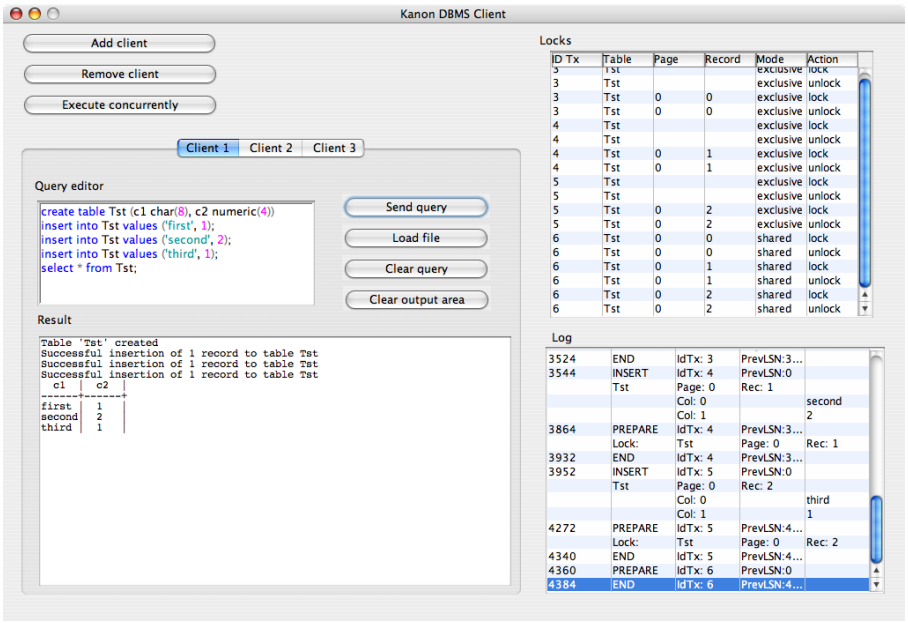

Fig. 1. A client screenshot showing the way how locks are acquired and released, and each operation gets saved into the log

Visualization example Table 1 shows the result of a transaction execution that inserts two new records in a table (named providers) and then executes a rollback. The SQL queries are:

```
begin transaction;
insert into providers values('Johnson', '1st avenue 45', 46843221);
insert into providers values('Peterson', '5th street 2310', 44327000);
rollback transaction;
```

Then the log table in the client GUI shows the changes occured to the tables as presented in Table 1.

These are the steps executed by the server:

- Begin operation initializes PrevLSN to 0. The first two events are related to the insertion of two new records on Providers table. Next it shows the Rollback event of transaction 6 and it begins to analyze it by reading the log file in a backwards fashion, from the last operation up to the first one (the index events can also be seen here).
- For each insertion operation that it is rolled back, a CLR_INSERT event is written in the long (as per ARIES specifications). This event show its owning transaction, UndoNextLSN field and which record is being removed.
- Finally it shows the END event related to the ending of transaction 6.

4604	INSERT	IdTx: 6	PrevLSN:0	
	providers	Pag: 0	Reg: 1	
		Col: 0		Johnson
		Col: 1		1st avenue 45
		Col: 2		46843221
5100	INSERT	IdTx: 6	PrevLSN:4984	
	providers	Pag: 0	Reg: 2	
		Col: 0		Peterson
		Col: 1		5th street 2310
		Col: 2		44327000
5596	ROLLBACK	IdTx: 6	PrevLSN:5480	
Reading: 5596	ROLLBACK	proxLSN:5616		
Reading: 5480	INSERT_INDEX	proxLSN:5596		
Reading: 5364	INSERT_INDEX	proxLSN:5480		
Reading: 5248	INSERT_INDEX	proxLSN:5364		
Reading: 5100	INSERT	proxLSN:5248		
5868	CLR_INSERT	IdTx: 6	PrevLSN:5784	
		4984		
	providers	Pag: 0	Reg: 2	
Reading: 4984	INSERT_INDEX	proxLSN:5100		
Reading: 4868	INSERT_INDEX	proxLSN:4984		
Reading: 4752	INSERT_INDEX	proxLSN:4868		
Reading: 4604	INSERT	proxLSN:4752		
6196	CLR_INSERT	IdTx: 6	PrevLSN:6112	
		0		
	providers	Pag: 0	Reg: 1	
6272	END	IdTx: 6	PrevLSN:6196	

Table 1. Transaction log display of a transaction execution

User interaction The user can use the system by loading Kanon's server first and then executing the client (minimum requirement is to have installed Java jdk 5.0 or later). The Client UI is easy to use and with it the user can see the progress of the transactions that are currently being executed in the server. It is possible to simulate more than one client to concurrently execute several transactions.

Friendly source code As mentioned before, viewing the code or debugging it allows the users to follow the recovery process implemented in Kanon. Another way to do this is by reading the ARIES paper to verify that the theory explained there matches with the Kanon's own implementation of ARIES algorithms; with the goal of obtaining a deeper knowledge of the theoretical concepts and their implementations.

3 Conclusions

This project pretends to serve as an aid to understanding and observing the inner-work of a RDBMS. It was thought with educational purposes, displaying to the user the internal engine operation in a simple way. It is showed how to build, from scratch,

the ARIES family of algorithms with a simple, modular and object-oriented academic model. Whoever interested in the subject is able to look at the ARIES machinery in a fully functional way for a better comprehension of its inner-working.

The methodology that the ARIES papers establishes fits well with complex subjects like transaction nesting, recovery from user errors and the use of save points.

4 Future work

This implementation does not take into account some interesting topics such as distributed transactions or long-duration ones. These are open issues and research subjects in many universities.

For Database teaching purposes, the project can be extended by showing the state of other components of the engine, like the Buffer Manager or the query analyzer. For the former, it is possible in the client to show the load and release of different pages into the Buffer Manager by currently active transactions. And for the latter, how a SQL sentence becomes a relational algebraic tree, and how the engine processes it.

An issue of improvement could be located in the Analysis phase executed in the recovery stage at system startup. One change would consist of parallelizing steps in the recovery method in order to gain performance and efficiency. An added change could be to create a Selective Redo: not to redo those transactions that will be undone in the Undo phase (faster start-up).

An alternative is to make the system listen to incoming connections (analyze queries, create and process new transactions) at the same moment that the recovering stage is being executed.

Support media recovery (backup copies and recovery methods when the data disk is faulty).

Improve the index system with the processes described in [9] and [10] and [11].

Provide methods for deadlock detection instead of methods for deadlock prevention. Details for this issue can be found at section 19.2.2 of [12].

An interesting one is to program the ability to suspend an active transaction and resume it at some other point in time (and maybe in a different thread).

5 Acknowledgments

Special thanks are due to Alejandro Vaisman who provided some helpful advice during preparation of this paper. Grateful thanks are extended to Prof. Avi Silberschatz for kindly encouraging us.

References

[1] Bernstein, P., Hadzilacos, V., Goodman, N. *Concurrency Control and Recovery in Database Systems*, 1987.

[2] Facultad de Ciencias Exactas y Naturales (Universidad de Buenos Aires) *http://www.fcen.uba.ar*

[3] Base de Datos (Databases) undergraduated Course *http://www-2.dc.uba.ar/materias/bd*

[4] Department of Computer Sciences *http://www.dc.uba.ar*

[5] Gamma, E., Helm, R., Johnson, R.,Vlissides, J. *Design Patterns, Elements of Reusable Object-Oriented Software*, 1995.
[6] Leggieri, L., Berlin, J., Cabs, V., Pippia, F. *KANON DBMS Operations manual*, 2007.
[7] Leggieri, L., Berlin, J., Cabs, V., Pippia, F. *KANON DBMS Technical Report*, 2007.
[8] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., Schwarz, P. *ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging*, 1989.
[9] Mohan, C. *ARIES/KVL A Key-Value Locking Method for Concurrency Control of Multiaction Transactions Operating on B-Tree Indexes*, 1990.
[10] Mohan, C. *ARIES/LHS: A Concurrency Control and Recovery Method Using Write-Ahead Logging for Linear Hashing with Separators*, 1993.
[11] Mohan, C., Levine, F. *ARIES/IM: An Efficient and High Concurrency Index Management Method Using Write-Ahead Logging*, 1992.
[12] Ramakrishnan, R., Gherke, J. *Database Management Systems, 3rd Ed.*, 2003.
[13] Ullman, J. *Principles of Database and Knowledge Base Systems*, 1988.
[14] Gray, J., Reuter, A. *Transaction processing: concepts and techniques*, 1993.