Conceptual Level Concurrency Control of Relational Update Transactions
(Extended Abstract)

Victor Vianu †
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093, USA
and
Gottfried Vossen ‡
Lehrstuhl fuer angewandte Mathematik
Technische Hochschule Aachen
Ahornstr. 55, D-5100 Aachen, FRG

Abstract
The concurrency control problem is examined for transactions as they appear at the conceptual level in a concrete database model. Specifically, a class of widely accepted update transactions in relational databases is studied with respect to concurrency control. It is shown how static serializability testing, as well as dynamic scheduling, can be improved by making use of the semantic information available at the conceptual level on transactions and database constraints.

1. Introduction
The concurrency control problem for database systems is traditionally investigated using a simple model for transactions which reflects the sequence of read and write operations performed at the internal level. Schedulers based on such a model allow limited concurrency, because they do not have available any information on the meaning of the transactions. Recently, there has been considerable interest in looking at transaction models capturing more semantic information, and using this information to increase the amount of concurrency allowed by schedulers. The work in this area includes widely different approaches, such as enriching the read/write model with additional operations [12, 5], using models based on abstract data types [18, 21, 17], and using semantic information provided by the users [8]. However, none of these approaches considers transactions as they appear at the conceptual level in a concrete database model. This paper is a first effort in this direction. Specifically, we investigate the concurrency control problem for a widely accepted class of update transactions in relational databases.

The model for transactions used in this paper is that developed in [1, 2]. A transaction is viewed as a sequence of insertions, deletions, or modifications, forming a semantic unit. The selection of tuples (to be deleted or modified) involves the inspection of individual attribute values for each tuple. Several features of this model are particularly desirable in the context of concurrency control. First, equivalence of transactions can be tested effectively and efficiently (see [1, 2]). This enables us to look at serializability of schedules in semantic terms, rather than syntactic, and leads to a scheduling algorithm that allows increased concurrency. Second, efficient techniques are available for simplifying transactions [1, 2, 10] and extracting internal parallelism from

† This author was supported in part by the National Science Foundation, under grant number IST-8511558.
‡ This author was supported by the NATO Science Committee under a Postdoctoral Fellowship, Grant No. DAAD 300/402/508/7; this work was done while the author was visiting at UCSD.
transactions [10, 11]. These techniques can be used in conjunction with scheduling algorithms to obtain simpler schedules and to further increase concurrency.

We start by looking at various notions of schedules and serializability, which are natural in the context of our model. Schedules are distinguished based on the connection between the updates occurring in the schedule and those in the transactions. The simplest type of schedule is an interleaving of all updates occurring in the transactions. However, other types of schedules are considered, where the updates in the schedule are connected to those in the transactions in a less straightforward fashion. Serializability of a schedule is defined entirely semantically: a schedule for a set of transactions is serializable if it has the same effect as the execution of the transactions in some serial order. Our results concern static serializability testing, as well as dynamic scheduling. We first look at the complexity of testing serializability and show that it is NP-complete. However, we exhibit an infinite sequence of increasingly powerful polynomial-time testing algorithms which approximate, in some sense, exact serializability testing. Intuitively, the exact algorithm requires looking at the effect of the schedule as a whole. The approximate algorithms are obtained by restricting, in various ways, the "amount" of context of the schedule examined at a time. The least powerful of the approximate algorithms examines only conflicts between pairs of updates. (Note that this corresponds to the use of compatibility tables [9, 12] for our updates.) We also show that efficient, exact serializability testing algorithms exist for less powerful transactions. For instance, serializability can be tested in polynomial time if the transactions contain only inserts and deletes (no modifications).

Finally, we briefly look at concurrency control using constraint information. Specifically, we show how our serializability testing algorithms can use information on the functional dependencies satisfied by the database to allow more concurrency in schedules.

The paper consists of six sections. The model for transactions is briefly outlined in section 2. In section 3, the notions of schedule and serializability are discussed. Our results on testing serializability are presented in section 4. Section 5 contains our results on concurrency control in the presence of functional dependencies. In Section 6, we summarize our work and review some problems that deserve further study.

This paper is an extended abstract of [20]; due to space limitations, some concepts and results are presented informally, and many details are omitted.

2. The Model for Transactions

The model for transactions used in this paper is that developed in [1, 2]. In this section we review the model and some previously obtained results. We assume familiarity with basic terminology for relational databases, as in [14, 19].

Informally, a transaction is a sequence of instructions viewed as a semantic unit. As in most commercial database management systems, three types of atomic instructions are used to build up (update) transactions: insertion, deletion, and modification. We focus on a tractable and widely used class of transactions. Specifically, we consider the important class of "domain-based" transactions, where the selection of tuples (to be deleted or modified) only involves the inspection of each individual attribute value of a tuple, independently of other attribute values in the tuple and of other tuples in the instance. To formally define these instructions, we need the concept of a condition:

2.1 Definition. Let $U$ be a set of attributes. A condition over $U$ is an expression of the form $A=a$ or $A\neq a$, where $A \in U$ and $a \in \text{dom}(A).$ A tuple $\mu$ over $U$ satisfies a condition $A=a$ ($A\neq a$) iff $\mu(A) = a$ ($\mu(A) \neq a$). A

1 The model can easily be extended to include other comparison operators as well.