A Model of Independence and Overlap for Transactions on Database Schemata

Stephen J. Hegner

Umeå University, Department of Computing Science
SE-901 87 Umeå, Sweden
hegner@cs.umu.se   http://www.cs.umu.se/~hegner

Abstract. Traditional models of support for concurrent transactions invariably rely upon a notion of serializability, which involves not only complex scheduling, but also primitives (such as locks) for requiring transactions to wait, as well for aborting a transaction and forcing it to re-run. For batch transactions, this approach is often the most reasonable. On the other hand, for interactive transactions, only a very limited amount of waiting and aborting is tolerable, and so minimizing their occurrence, even at the cost of increased analysis of the transactions themselves, is warranted. In this work, a systematic study of independence for transactions, without any explicit serialization, is initiated. Each transaction operates on a view of the main schema, and each such view is partitioned into a write region and a read-only region. For a set of transactions to run concurrently, their views may overlap only on their read-only regions. These regions need not be specified explicitly; rather, they are defined naturally using a component-based model of the main schema. Furthermore, when two transactions do conflict, because their views overlap on write regions, the precise point of conflict is immediately identified. To illustrate the utility of the framework, the case of relational schemata governed by the most common types of constraints in practice — functional and foreign-key dependencies — is developed in detail.

1 Introduction

Support for concurrent transactions has long been a crucial feature of database-management systems. Central to all approaches is a notion of conflict. Typically, the database is modelled as a set $X = \{x_1, x_2, \ldots, x_n\}$ of data objects, each of which may be read and modified by transactions. Two transactions are in potential conflict if they operate on the same data object in certain ways. In the case of a potential conflict, the standard criterion for admissibility is that the transactions involved be interleaved in such a way that the the result is equivalent to that obtained were they not interleaved; so-called *serializability* [3, Ch. 22] [15, Ch. 2]. For enforcement, the scheduler may require a transaction to wait (e.g., via locking) for a resource which is held by another transaction, and in the worst case it may require a transaction to abort and re-run.

In recent years, the need to support interactive transactions has grown enormously. With humans in the loop, techniques which impose long waits (such as locking) are clearly undesirable, if not outright unacceptable. Therefore, such actions must be used sparingly, if at all. Of course, there is no magical way to avoid conflict of concurrent operations. On the other hand, with interactive transactions, additional overhead of a few milliseconds or even a few seconds is a reasonable tradeoff for reduced delays and aborts, so it is realistic to work with a more complex model of data objects, providing a finer analysis of how the operations of distinct transactions interact and resulting in fewer conflicts and a better model of identifying those conflicts which do occur. The goal of this work is to provide such a model. The basic idea is that data objects are not just sets of tuples, but rather are defined by objects which are structured views. Each such view-object has a number of sub-views which define a read-only region. Two objects can never entail a conflict of transactions if those objects overlap at most on their read-only regions. There is furthermore a calculus of combination of these objects which creates larger objects. If all objects which contain a given read-only region are combined, then that region becomes writable. The formal model is based database schema components, as developed in [8]. As such, it distinguishes itself from other work on similar topics, such as [18] and [17], which focus more on transaction primitives. On the other hand, it also distinguishes itself from semantic approaches, such as [12], in which increased concurrency is obtained by modelling the the data objects as abstract data types with explicitly defined operations. In the work reported here, the data objects have no structure beyond the definitions of write and read-only regions.

In describing the work in this paper, the term *claim*, rather than *lock*, will be used to describe a data object which has been assigned to a transaction. The reason is that this paper is not about locking or serialization, it is rather about modelling independence and conflict. How the latter is prevented or resolved is not the focus. Locking is one way to prevent conflict, but solutions which involve negotiation, for example, may also be appropriate in the context of interactive transactions. Before proceeding to the development of the main ideas, it is useful to illustrate the basic idea via a running example, which will also be used in other parts of the paper. The relational schema $\mathbf{E}_0$ has the three relations and integrity constraints identified in Fig. 1. The keys are underlined,

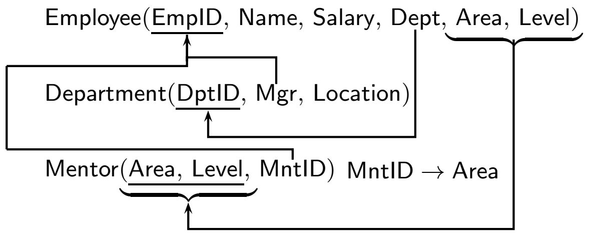
Employee(EmpID, Name, Salary, Dept, Area, Level)
Department(DptID, Mgr, Location)
Mentor(Area, Level, MntID)
MntID → Area

Fig. 1. Diagrammatic representation of the running example $\mathbf{E}_0$

and foreign keys are indicated by the arrows from the foreign key to the associated primary key. Thus, $\mathsf{Department}[\mathsf{Mgr}] \subseteq \mathsf{Employee}[\mathsf{EmpID}]$, $\mathsf{Mentor}[\mathsf{MntID}] \subseteq \mathsf{Employee}[\mathsf{EmpID}]$, and $\mathsf{Employee}[\mathsf{Area}, \mathsf{Level}] \subseteq \mathsf{Mentor}[\mathsf{Area}, \mathsf{Level}]$. The functional dependency $\mathsf{MntID} \to \mathsf{Area}$ is shown explicitly because it is not a key constraint. All relations are in 3NF (third normal form). The relation Mentor, constrained by both of the FDs (functional dependencies) $\{\mathsf{Area}, \mathsf{Level}\} \to \mathsf{MntID}$ and $\mathsf{MntID} \to \mathsf{Area}$, is a classical example of a relation which cannot be decomposed further into BCNF (so that the only FDs are key dependencies) while retaining a cover of the governing dependencies [4, Sec. 10.5].

In most approaches to defining data objects for transactions, even those which forward using so-called predicate locks such as [5] and [14], the data objects always return a set of tuples from a relation. However, it is advantageous, and also a natural feature of the theory developed here, to allow claim sets to be defined by views, since two transactions may be able to update different fields of the same tuple. Consider two simple transactions. T_1 is to give a 5% raise to each employee in the Research department, and T_2 is to change the names of employees in the set NSet (because they married, say). It is clear that these two operations may be carried out concurrently, even though they may operate on the same tuples (in the case that NSet contains employees who work in the Research department). Formally, T_1 operates on the view defined by $\sigma_{\mathsf{Dept}=\mathrm{Research}}\langle \mathsf{Employee}[\mathsf{EmpID}, \mathsf{Salary}, \mathsf{Dept}]\rangle$; i.e., the selection to the Research department of the projection $\mathsf{Employee}[\mathsf{EmpID}, \mathsf{Salary}]$, while T_2 operates on the view defined by $\sigma_{\mathsf{Name}=\mathrm{NSet}}\langle \mathsf{Employee}[\mathsf{EmpID}, \mathsf{Name}]\rangle$. These two data objects overlap on their read claims, but not on their write claims. Specifically, the write claim of T_1 involves only the values of the Salary field, while that for T_2 involves only the name field. Their read claims overlap on the EmpIDs of their common tuples. Since neither transaction may alter these EmpIDs, they may be shared. More complex operation on this example schema and others will be considered in that which follows.

The remainder of the paper is composed of two main sections. In Section 2, the formal model is developed, independently of any specific data model. In Section 3, these ideas are applied to the relational model, constrained by FDs and foreign-key dependencies (FKDs). The basic components identified via vertical decomposition (i.e., projections) are augmented with a further decomposition into horizontal components, defined by selection. Finally, Section 4 provides a summary and indication of possible further directions.

2 Component-Based Independent Updates

In this section, the fundamental ideas of the model for complex update objects for transactions are developed. As they do not depend upon any particular data model, they are developed within a general framework of set-based schemata. The basic definitions of schema, morphism, view, complement, and the like parallel those developed in [6], to which the reader is referred for details. A summary of some of these ideas may be found in [7, Sum. 2.1] as well. The basic ideas for the component-based concepts are based upon those in [8]. The relational model will nevertheless be used for some examples. Because it is so widely known, the standard notation and terminology surrounding it, as may be found in textbooks such as [4] and [13], will not be reviewed but rather assumed.

Definition 2.1 (Database schemata, views, and updates). A *set-based database schema* $\mathbf{D}$ is one for which a finite set $\mathsf{LDB}(\mathbf{D})$ of *legal database states* is given. In the relational model, $\mathsf{LDB}(\mathbf{D})$ is the set of states which satisfy the integrity constraints of the schema. A *morphism* $f : \mathbf{D}_1 \to \mathbf{D}_2$ of set-based schemata is given by a function $\mathsf{LDB}(f) : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$. Since no confusion can result, the qualifier LDB will usually be dropped; i.e., $f : \mathsf{LDB}(\mathbf{D}_1) \to \mathsf{LDB}(\mathbf{D}_2)$. For the rest of this section, unless stated specifically to the contrary, the term *database schema* will mean *set-based database schema*.

A *view* of the database schema $\mathbf{D}$ is a pair $\Gamma = (\mathbf{V}, \gamma)$ in which $\mathbf{V}$ is a database schema and $\gamma : \mathbf{D} \to \mathbf{V}$ is a database morphism for which the underlying function $\gamma : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathbf{V})$ is surjective. The surjectivity ensures that every state of the view schema $\mathbf{V}$ is the image of some state of the main schema $\mathbf{D}$. The equivalence relation $\mathsf{Congr}(\Gamma) = \{(M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D}) \mid \gamma(M_1) = \gamma(M_2)\}$ is called the *congruence* of Γ. If $\Gamma_1 = (\mathbf{V}_1, \gamma_1)$ and $\Gamma_2 = (\mathbf{V}_2, \gamma_2)$ are views with $\mathsf{Congr}(\Gamma_1) \subseteq \mathsf{Congr}(\Gamma_2)$, then Γ_2 may be thought of as a smaller view than Γ_1, in that Γ_1 preserves more information about the state of $\mathbf{D}$ than does Γ_2. The special notation $\Gamma_2 \sqsubseteq_{\mathbf{D}} \Gamma_1$ will be used to indicate that $\mathsf{Congr}(\Gamma_1) \subseteq \mathsf{Congr}(\Gamma_2)$. To make this idea more precise, given that $\Gamma_2 \sqsubseteq_{\mathbf{D}} \Gamma_1$, define the function $\lambda\langle \Gamma_1, \Gamma_2\rangle : \mathbf{V}_1 \to \mathbf{V}_2$ by $N \mapsto \gamma_1(M)$ for any $M \in \gamma_1^{-1}(N)$. This function is well defined, since if $M_1, M_2 \in \gamma_1^{-1}(N)$, then $\gamma_2(M_1) = \gamma_2(M_2)$, owing to the fact that $\mathsf{Congr}(\Gamma_1) \subseteq \mathsf{Congr}(\Gamma_2)$. Furthermore, Γ_2 may be regarded as a view $\Lambda(\Gamma_1, \Gamma_2) = (\mathbf{V}_2, \lambda\langle \Gamma_1, \Gamma_2\rangle)$ of $\mathbf{V}_1$. As a concrete example, let $\mathbf{E}_1$ be the relational schema with the single relation schema $R[ABC]$, constrained by the functional dependencies (FDs) $A \to C$ and $B \to C$. Let $\Pi_{AB}^{\mathbf{E}_1} = (\mathbf{E}_{1_{AB}}, \pi_{AB}^{\mathbf{E}_1})$ be the view which defines the projection of $R[ABC]$ onto AB; the single relation symbol of $\mathbf{E}_{1_{AB}}$ is $R[AB]$. Similarly, let $\Pi_B^{\mathbf{E}_1} = (\mathbf{E}_{1_B}, \pi_B^{\mathbf{E}_1})$ be the view which defines the projection of $R[ABC]$ onto B; the single relation symbol of $\mathbf{E}_{1_B}$ is $R[B]$. The function $\lambda\langle \Pi_{AB}^{\mathbf{E}_1}, \Pi_B^{\mathbf{E}_1}\rangle$ is just the projection $\pi_B^{\mathbf{E_{AB}}_1}$ of $R[AB]$ onto $R[B]$, with the relative view $\Pi_B^{\mathbf{E}_{1_{AB}}} = (\mathbf{E}_{1_B}, \pi_B^{\mathbf{E_{AB}}_1})$.

It is also important to note that each equivalence relation $r \subseteq \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$ defines a set-based view $\Gamma_{[r]} = (\mathbf{V}_{[r]}, \gamma_{[r]})$ with $\mathsf{LDB}(\mathbf{V}_{[r]}) = \mathsf{LDB}(\mathbf{D})/r$, the blocks of the equivalence relation r, and with $\gamma_{[r]} : M \mapsto \{M' \mid r(M, M')\}$. Indeed, the congruence of a set-based view $\Gamma = (\mathbf{V}, \gamma)$ characterizes it up to a renaming of the elements of $\mathsf{LDB}(\mathbf{V})$. More formally, a *morphism* $h : \Gamma_1 \to \Gamma_2$ of views is given by a morphism $h : \mathbf{V}_1 \to \mathbf{V}_2$ on the underlying schemata with the property that $h \circ \gamma_1 = \gamma_2$. In the case of set-based views, h is bijective (hence an isomorphism) iff $\mathsf{Congr}(\Gamma_1) = \mathsf{Congr}(\Gamma_2)$.

The *zero view* on $\mathbf{D}$, denoted $\mathsf{ZView}_{\mathbf{D}}$, is a view whose congruence is $\mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$. Thus, its schema has only one state, and so it conveys no information about the state of the main schema. Dually, the *identity view*

$\mathsf{IdView}_{\mathbf{D}} = (\mathbf{D}, \mathsf{IdMor}_{\mathbf{D}})$ on $\mathbf{D}$ is the view which is the identity on $\mathsf{LDB}(\mathbf{D})$. Each will be useful in certain constructions.

The *join* of two set-based views Γ_1 and Γ_2 is the set-based view (unique up to isomorphism) whose congruence is $\mathsf{Congr}(\Gamma_1) \cap \mathsf{Congr}(\Gamma_2)$. It is denoted $\Gamma_1 \sqcup \Gamma_2$. In the case of relational views, the join may be constructed explicitly by taking the view schema to be the (disjoint) union of the schemata of the two views. See [9, Def. 4.3] for details. Since the join is associative, the definition extends naturally to an arbitrary finite set. The join of a finite set $S = \{\Gamma_1, \ldots, \Gamma_n\}$ of views is denoted $\Gamma_1 \sqcup \ldots \sqcup \Gamma_n$, or $\bigsqcup_{i=1}^{n} \Gamma_i$, or just $\bigsqcup S$.

An *update* on the schema $\mathbf{D}$ is a pair $(M_1, M_2) \in \mathsf{LDB}(\mathbf{D}) \times \mathsf{LDB}(\mathbf{D})$, with M_1 the old state and M_2 the new state. The set of all updates on $\mathbf{D}$ is denoted $\mathsf{Updates}(\mathbf{D})$. Let $u = (N_1, N_2) \in \mathsf{Updates}(\mathbf{V})$ be an update on the view schema $\mathbf{V}$, and let $M_1 \in \mathsf{LDB}(\mathbf{D})$ with $\gamma(M_1) = N_1$. A *reflection* (or *translation*) of the view update u to $\mathbf{D}$ relative to M_1 is an update $u' = (M_1, M_2)$ on $\mathbf{D}$ with the property that $\gamma(M_2) = N_2$.

The update $(M_1, M_2) \in \mathsf{Updates}(\mathbf{D})$ is *constant on Γ* if $\gamma(M_1) = \gamma(M_2)$.

Notation 2.2. Throughout this section, unless specifically stated to the contrary, $\mathbf{D}$ will be taken to be a set-based database schema. Furthermore, $\Gamma = (\mathbf{V}, \gamma)$, as well as $\Gamma_x = (\mathbf{V}_x, \gamma_x)$ for any subscript x, will be taken to be a set-based view over $\mathbf{D}$.

If $\mathbf{E}_x$ is a relational schema with single relation $R[\mathbf{U}]$ for some set $\mathbf{U}$ of attributes, and $\mathbf{W} \subseteq \mathbf{U}$, then $\Pi^{\mathbf{E}_x}_{\mathbf{W}} = (\mathbf{E}_{x\mathbf{W}}, \pi^{\mathbf{E}_x}_{\mathbf{W}})$ is the projection view on $\mathbf{W}$ whose relation symbol is denoted by $R[\mathbf{W}]$.

Definition 2.3 (Complementary sets and pairwise definability). Let $\mathfrak{C} = \{\Gamma_1, \Gamma_2, \ldots, \Gamma_n\}$ be a finite set of views of $\mathbf{D}$. Call $\mathfrak{C}$ a *complementary set* if $\bigsqcup_{i=1}^{N} \Gamma_i = \mathsf{IdView}_{\mathbf{D}}$. If $n = 2$, $\mathfrak{C}$ is also called a *complementary pair.* The *decomposition mapping* for $\mathfrak{C}$ is $\gamma_1 \times \ldots \times \gamma_k : \mathsf{LDB}(\mathbf{D}) \to \mathsf{LDB}(\mathbf{V}_1) \times \ldots \times \mathsf{LDB}(\mathbf{V}_k)$, given on elements by $M \mapsto (\gamma_1(M), \ldots, \gamma_k(M))$. It is immediate that $\mathfrak{u}$ is a complementary set iff $\gamma_1 \times \ldots \times \gamma_k$ is injective. Thus, a complementary set of views defines a way to recover the state of the main schema from the combined states of the views. In classical terms, it defines a lossless decomposition.

Think of $\mathfrak{C}$ as representing the set X of data objects in the concurrency model, as identified in Section 1. A transaction T wishes to effect an update u which involves some subset $S_T \subseteq \mathfrak{C}$; formally, u is specified as an update on the schema of $\bigsqcup S_T$. Ideally, other transactions could then be allowed to update the views not in S_T. Unfortunately, for most realistic schemata, it is not possible to find a useful set $\mathfrak{C}$ of views whose members are *independent* in the sense that each may be updated without affecting the states of the others. Rather, the elements of such a set of views typically overlap, and the updates which are permitted must respect that overlap. The necessary additional condition is found in a classical result in the theory of database decomposition. In [2], a number of "desirable" aspects of universal relational schemata are presented. One of these is *pairwise definability.* Let $\mathbf{U}$ be a finite set of attributes, and consider $\mathbf{E}_{\mathbf{U}}$, as defined in Notation 2.2, constrained by some set $\mathcal{F}$ of dependencies. Let

$Z = \{\mathit{\Pi}^{\mathbf{E_U}}_{\mathbf{U}_i} \mid 1 \leq i \leq k\}$. be a finite set of projections of $\mathbf{E_U}$ and assume that Z is a complementary set of views. Call a sequence $S = \langle M_1, M_2, \ldots, M_k\rangle$ with $M_i \in \mathsf{LDB}(\mathbf{E}_{\mathbf{U}_i})$, $1 \leq i \leq k$, *compatible* for Z if there is an $M \in \mathsf{LDB}(\mathbf{E_U})$ with the property that $\pi^{\mathbf{E_U}}_{\mathbf{U}_i}(M) = M_i$ for $1 \leq i \leq n$, and call S *pairwise compatible* for Z if whenever $\mathbf{U}_i \cap \mathbf{U}_j \neq \emptyset$, then $\pi^{\mathbf{E}_{\mathbf{U}_i}}_{\mathbf{U}_i \cap \mathbf{U}_j}(M_i) = \pi^{\mathbf{E}_{\mathbf{U}_j}}_{\mathbf{U}_i \cap \mathbf{U}_j}(M_j)$. Call Z *pairwise definable* if every pairwise compatible set for Z is compatible for Z; that is, agreement on the overlapping columns is sufficient to ensure consistency. Within this model, there is a simple characterization of pairwise definability; namely, that a cover of $\mathcal{F}$ embed into the members of Z. For a complementary pair, a proof may be found in [6, 2.17]; the extension to larger sets is straightforward.

To extend this idea to the general case of a set $\mathfrak{C} = \{\mathit{\Gamma}_1, \mathit{\Gamma}_2, \ldots, \mathit{\Gamma}_n\}$ of views of $\mathbf{D}$, first define a sequence $S = \langle M_1, \ldots, M_n\rangle$ of states with $M_i \in \mathsf{LDB}(\mathbf{V}_i)$ for $1 \leq i \leq k$ to be *compatible for* $\mathfrak{C}$ if there is an $M \in \mathsf{LDB}(\mathbf{D})$ with the property that $\gamma_i(M) = M_i$ for $1 \leq i \leq n$. To obtain a notion of pairwise compatibility, the idea is to specify a set $\mathfrak{P}$ of views on which the elements of $\mathfrak{C} = \{\mathit{\Gamma}_1, \mathit{\Gamma}_2, \ldots, \mathit{\Gamma}_n\}$ must agree, called the set of *ports*. More precisely, say that S is *pairwise compatible for* $\mathfrak{C}$ *with respect to* $\mathfrak{P}$ if $\lambda\langle\mathit{\Gamma}_i, \mathit{\Gamma}\rangle(M_i) = \lambda\langle\mathit{\Gamma}_j, \mathit{\Gamma}\rangle(M_j)$ for every pair $\{\mathit{\Gamma}_i, \mathit{\Gamma}_j\} \subseteq \mathfrak{C}$ and every $\mathit{\Gamma} \in \mathfrak{P}$ for which $\mathit{\Gamma} \sqsubseteq_{\mathbf{D}} \mathit{\Gamma}_i$ and $\mathit{\Gamma} \sqsubseteq_{\mathbf{D}} \mathit{\Gamma}_j$. Say that $\mathfrak{C}$ is *pairwise definable via* $\mathfrak{P}$ if every pairwise compatible set is compatible. In the relational example above, the set of ports is $\{\mathit{\Pi}^{\mathbf{E_U}}_{\mathbf{W}} \mid \mathbf{W} \subseteq \mathbf{U}\}$; that is, the set of all projections. There is a useful visualization of pairwise definability, using the conventions introduced in [8]. Each main view in $\mathfrak{C}$ (corresponding to a *component* in [8] is represented using a rectangle, and each port in $\mathfrak{P}$ is represented using a circle, with lines connecting the components to the ports which they subsume. Fig. 3 shows the representation for a decomposition of the schema $\mathbf{E}_0$ of Sec. 1 (to be discussed in detail in the next section), and Fig. 2 illustrates this idea for the views $\{\mathit{\Pi}^{\mathbf{E}_3}_{AB}, \mathit{\Pi}^{\mathbf{E}_3}_{BC}, \mathit{\Pi}^{\mathbf{E}_3}_{CD}, \mathit{\Pi}^{\mathbf{E}_3}_{CE}\}$ of the relational schema $\mathbf{E}_3$ with the the single relation $R[ABCDE]$, constrained by the FDs in $\mathcal{F}_3 = \{A \to B, B \to C, C \to DE\}$. The set of ports is taken to be the three projections $\{\mathit{\Pi}^{\mathbf{E}_3}_{A}, \mathit{\Pi}^{\mathbf{E}_3}_{B}, \mathit{\Pi}^{\mathbf{E}_3}_{C}\}$.

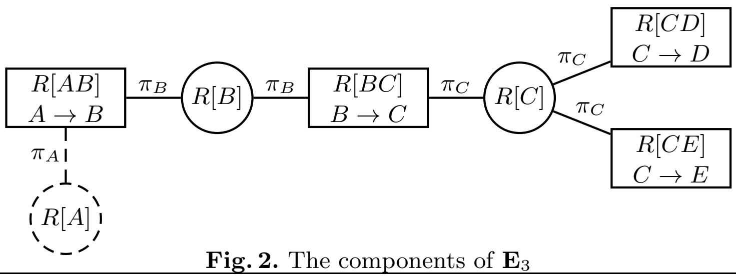

Fig. 2. The components of $\mathbf{E}_3$

In Fig. 2, the port $R[A]$ is shown with dashed lines because it is not necessary, as it is connected to only one component. Such a port is called *irrelevant*. More generally, returning to the general case of $\mathfrak{C}$ and $\mathfrak{P}$, call $\Gamma \in \mathfrak{P}$ a *port of* $\Gamma_i \in \mathfrak{C}$ if $\Gamma \sqsubseteq_{\mathbf{D}} \Gamma_i$, and call it an *essential port* of Γ_i if there is at least one other component Γ_j, distinct from Γ_i, for which it is also a port. Call $\Gamma \in \mathfrak{P}$ *relevant* for $\mathfrak{C}$ if it is an essential port for at least two distinct members of $\mathfrak{C}$, and call $\mathfrak{P}$ *completely relevant* for $\mathfrak{C}$ if every $\Gamma \in \mathfrak{P}$ is relevant for $\mathfrak{C}$. It is always possible to render $\mathfrak{P}$ completely relevant by removing elements which are not relevant. Let $\mathsf{RelRed}_{\mathfrak{C}}\langle\mathfrak{P}\rangle$ denote the subset of $\mathfrak{P}$ obtained by removing all elements which are not relevant for $\mathfrak{C}$.

Definition 2.4 (Independent updates). Using pairwise definability, it is possible to give a formal definition of what is meant by independent updates. First, let $u = \langle (N_1, N'_1), \ldots, (N_n, N'_n) \rangle \in \mathsf{Updates}(\mathbf{V}_1) \times \ldots \times \mathsf{Updates}(\mathbf{V}_n)$. Call u *initial-state compatible* (resp. *final-state compatible*) for $\mathfrak{C}$ if $\langle N_1, \ldots, N_n \rangle$ (resp. $\langle N'_1, \ldots, N'_n \rangle$) is compatible for $\mathfrak{C}$. If u is both initial-state and final-state compatible, it is called simply *compatible* for $\mathfrak{C}$. It is immediate that the set of all elements of $\mathsf{Updates}(\mathbf{V}_1) \times \ldots \times \mathsf{Updates}(\mathbf{V}_n)$ which are compatible for $\mathfrak{C}$ is in bijective correspondence with $\mathsf{Updates}(\mathbf{D})$. Of interest here, however, is the subset of the compatible updates which are independent. To this end, let $\mathfrak{u} \subseteq \mathsf{Updates}(\mathbf{V}_1) \times \ldots \times \mathsf{Updates}(\mathbf{V}_n)$ consist of compatible n-tuples. Call $\mathfrak{u}$ *independent* for $\mathfrak{C}$ if it is compatible for $\mathfrak{C}$ and the following two conditions are satisfied.

(ind1) For every $M \in \mathsf{LDB}(\mathbf{D})$, the corresponding identity n-tuple $\langle (\gamma_1(M), \gamma_1(M)), \ldots, (\gamma_n(M), \gamma_n(M)) \rangle$ is in $\mathfrak{u}$ as well.

(ind2) For every n-element set $\{\langle (N_{i1}, N'_{i1}), \ldots, (N_{in}, N'_{in}) \rangle \mid 1 \leq i \leq n\} \subseteq \mathfrak{u}$, the diagonal tuple $\langle (N_{11}, N'_{11}), \ldots, (N_{ii}, N'_{ii}), \ldots, (N_{nn}, N'_{nn}) \rangle$ is in $\mathfrak{u}$ whenever it is initial-state compatible.

Condition (ind2) is the core of the definition, allowing one to "mix and match" updates from distinct n-tuples, subject only to the condition that the result is initial-state compatible. Condition (ind1) ensures that each view update may be considered alone, by matching it with identity updates from the other views.

For $\Gamma_i \in \mathfrak{C}$, define $\mathsf{UpdFam}\langle \Gamma_i, \mathfrak{P} \rangle$ to be the set of all updates on the schema $\mathbf{V}_i$ of Γ_i on which every $\Gamma \in \mathfrak{P}$ is constant and define $\mathsf{IndUpd}\langle \mathfrak{C}, \mathfrak{P} \rangle$ to be the set of initial-state compatible pairs in $\mathsf{UpdFam}\langle \Gamma_1, \mathfrak{P} \rangle \times \ldots \times \mathsf{UpdFam}\langle \Gamma_n, \mathfrak{P} \rangle$. The following result then provides the largest independent set.

Proposition 2.5. *Let $\mathfrak{C} = \{\Gamma_1, \ldots, \Gamma_n\}$ and $\mathfrak{P}$ be finite sets of views of $\mathbf{D}$, and suppose further that $\mathfrak{P}$ is completely relevant for $\mathfrak{C}$ and that $\mathfrak{C}$ is pairwise definable via $\mathfrak{P}$. Then for any $\mathfrak{u} \subseteq \mathsf{Updates}(\mathbf{V}_1) \times \ldots \times \mathsf{Updates}(\mathbf{V}_n)$ which is independent for $\mathfrak{C}$, $\mathfrak{u} \subseteq \mathsf{IndUpd}\langle \mathfrak{C}, \mathfrak{P} \rangle$.*

Proof. Let $\mathfrak{u} \subseteq \mathsf{Updates}(\mathbf{V}_1) \times \ldots \times \mathsf{Updates}(\mathbf{V}_n)$ be independent for $\mathfrak{C}$, and let $u = \langle (N_1, N'_1), \ldots, (N_n, N'_n) \rangle \in \mathfrak{u}$. Choose $i \in \{1, \ldots, n\}$ and let $u' = \langle (N_1, N_1), \ldots, (N_{i-1}, N_{i-1}), (N_i, N'_i), (N_{i+1}, N_{i+1}), \ldots, (N_n, N_n) \rangle$. Thus, u' is obtained from u by retaining (N_i, N'_i) and replacing each other entry with the identity which keeps it initial-state compatible. In view of the complete relevance of $\mathfrak{P}$, u' is constant on every view in $\mathfrak{P}$, since every such view is contained in at least two distinct members of $\mathfrak{C}$, at least one of which is held constant by the update. In particular, (N_i, N_i') is constant on all views $\Gamma \in \mathfrak{P}$ for which $\Gamma \sqsubseteq_{\mathbf{D}} \Gamma_i$. Since i was chosen arbitrarily, it follows that all of u must be constant on every view in $\mathfrak{P}$. □

Definition 2.6 (Compound components and external ports). The ideas of Definition 2.4 and Proposition 2.5 identify the conditions under which updates may be executed independently on the components in the set $\mathfrak{C}$ of data objects. However, not all updates are so representable. In particular, any update would change the state of of an essential port of $\mathfrak{P}$ is disallowed. Thus, this framework, by itself, is not adequate. When no atomic component in $\mathfrak{C}$ is adequate to support an update, the solution is to combine several components into one complex one. For example, referring to $\mathbf{E}_3$ of Definition 2.3 and Fig. 2, suppose that a transaction wishes to update the AB projection of $R[ABCDE]$. This is not possible within any single component; indeed, it requires an update on a port. The solution is to combine the components $R[AB]$ and $R[BC]$ into a single component defined by $R[ABC]$. The new set of components is then $\{\Pi^{\mathbf{E}_3}_{ABC}, \Pi^{\mathbf{E}_3}_{CE}, \Pi^{\mathbf{E}}_{CF}\}$, and an update to the AB-projection is now possible. The port $R[B]$ of this combination becomes *internal*, and hence irrelevant.

More generally, returning to the general case of $\mathfrak{C}$ and $\mathfrak{P}$ of Definition 2.3, a *compound component* over $\mathfrak{C}$ is any join of views in $\mathfrak{C}$. For $S \subseteq \mathfrak{C}$, the essential ports of the compound component $\bigsqcup C$ are exactly those $\Gamma \in \mathfrak{P}$ which are ports for some $\Gamma_i \in S$ as well as some $\Gamma_j \in \mathfrak{C} \setminus S$. Thus, for the join $\Pi^{\mathbf{E}_3}_{AB} \sqcup \Pi^{\mathbf{E}_3}_{BC}$, the only essential port is $R[C]$.

Discussion 2.7 (The support of independent updates). Continuing with the above framework, a transaction whose task is to perform an update must identify the components in $\mathfrak{C}$ which are necessary for the operations which it is to perform. The *claim set* $S \subseteq \mathfrak{C}$ which it is to hold must satisfy the following two conditions.

(u1) The update operations which it is to perform must be expressible within the view $\bigsqcup S$.

(u2) All essential ports of $\bigsqcup S$ must be constant under these update operations; that is, they are read(-only) claims but not write claims.

For a set of transactions to proceed independently, their claim sets may overlap only on their ports, and these overlaps identify the read claims. Any update is supportable by choosing S sufficiently large; indeed, by choosing $S = \mathfrak{C}$, all updates are possible (but without any parallelism).

It might seem a suitable strategy to allow a transaction to express its goal to update an arbitrary view Γ and then to seek a subset $S \subseteq \mathfrak{C}$ which "covers" Γ. However, this is not possible in general. Consider the example schema $\mathbf{E}_0$ of Sec. 1 and Figs. 1 and 3. Suppose that the view to be updated is the projection of Employee[Salary]. It makes no sense, in general, to insert salaries. They must be associated with employees. Thus, the transaction itself must have knowledge of the set $\mathfrak{C}$ of components and determine which ones to claim for its operations.

There is one further point which should be discussed briefly, and that is read claims. If a transaction claims an object $\bigsqcup S$, it may not need to be able to update all of it. For example, considering again the example $\mathbf{E}_0$, the task of a transaction may be to update employee salaries, based upon information about the department of each employee. Such a transaction would need to read claim the department information, but it would not require write access. The extension the model presented here to such read claims is straightforward, but due to space limitations it will not be developed further.

3 The Basic Components of a Relational Schema

The examples of Sec. 2 were all based upon classical "vertical" decomposition of relational schemata, which is not by itself adequate for defining useful read or write claims. In the context of the running schema $\mathbf{E}_0$, a transaction which is to update the salary of Alice should not need to claim the entire projection $\mathsf{Employee}[\mathsf{EmpID}, \mathsf{Salary}]$. Rather, it should suffice to claim just those tuples involving Alice. Thus, a complementary theory of *horizontal decomposition* is needed. That which is required for this work is very different from the horizontal decomposition of in [16, Ch. 5], which is based upon exceptions and *afunctional dependencies.* Unfortunately, that form of decomposition does not lead to pairwise definability. Rather, what is needed is an approach to horizontal decomposition which is based upon the relational operation of selection, just as vertical decomposition is based upon projection.

The context for this section is relational schema which are constrained by functional dependencies (FDs) and foreign-key dependencies (FKDs), which are undeniably the two most important types of constraints in real-world database schemata. Since classical vertical decomposition almost never considers inclusion dependencies (of which FKDs are a special case), it is prudent to begin with a short description of how they are incorporated into a vertical decomposition. To keep the focus on the key ideas, null values will not be considered; it will be assumed that all values are non null.

Discussion 3.1 (Conventions for vertical decomposition). The overall approach is to begin with a vertical decomposition, and then decompose each component into its horizontal sub-components. Thus, it is important to begin with a clarification of exactly what properties the vertical decomposition must have. First of all, the theory only applies to acyclic decompositions, which is equivalent to the existence of pairwise definable decompositions [2, Cond. 3.7]. Although that topic is usually approached from the perspective of join dependencies, cyclicity can also arise from decompositions arising entirely from FDs [1, Thm. 4]. Fortunately, such schemata occur rarely, if ever, in practice, but in any case, they are not covered by the theory presented here.

Figure 3 depicts the vertical decomposition for the schema $\mathbf{E}_3$, introduced in Sec. 1 and Fig. 1. For compactness, the relations $\mathsf{Employee}$, $\mathsf{Department}$, and Mentor are abbreviated to E, D, and M in the ports (the circles). $\underline{\text{Keys}}$ are underlined, while $\utilde{\text{foreign keys}}$ have a wavy line beneath them. These abbreviations and conventions will also be used in that which follows.

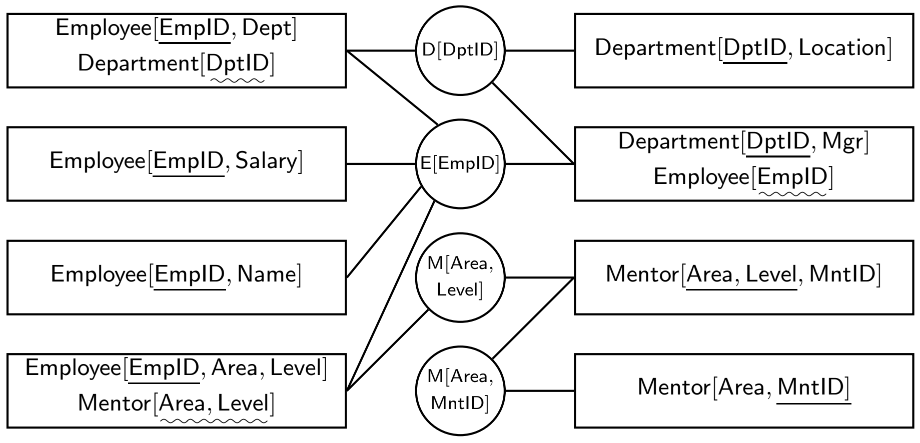

Fig. 3. The vertical components of the running example $\mathbf{E}_0$

In the classical theory of vertical decomposition, there is a tradeoff between 3NF, which always admits dependency-preserving decompositions but which may require non-key dependencies in some of the components, and BCNF, in which each component is governed only by key dependencies but for which dependency-preserving decomposition is not always possible. For this work, dependency preservation is essential. However, support for non-key dependencies within a component is also to be avoided. To address this dilemma, there is a trick which is arguably useless for classical normalization but which serves the purposes of support of independent updates very well. It is illustrated by the example of Mentor[MntID, Area, Level], governed by {Area, Level} $\rightarrow$ MntID and MntID $\rightarrow$ Area. The relation Mentor is "decomposed" into Mentor[MntID, $\underline{\text{Area, Level}}$], in which only the key dependency is enforced, and Mentor[$\underline{\text{MntID}}$, Area], in which only the local key dependency EmpID $\rightarrow$ Area is enforced. The latter FD is enforced in Mentor[MntID, $\underline{\text{Area, Level}}$] via the common port M[Area, MntID]. This approach is taken also when a relation has more than one key, which must be split into several components, one for each key. For example, if the relation Department were to have an additional attribute DName which were also a (secondary) key, it would necessary to have two additional components, Department[$\underline{\text{DptID}}$, DName] and Department[DptID, $\underline{\text{DName}}$]. Their common port would be the entire relation Department[DptID, DName], but each key would be checked separately in its component. Thus, it is always possible to arrange things so that only one key dependency need be checked in each vertical component. In Fig. 3, the key to be checked in a given component is exactly that which is underlined.

In most cases each component relation may have only one non-key attribute. The only exception is foreign keys consisting of more than one attribute, which must be grouped. An example of the latter is Employee[$\underline{\text{EmpID}}$, Area, Level]. Otherwise, for example, the relation

$\mathsf{Employee}[\mathsf{EmpID}, \mathsf{Salary}, \mathsf{Dept}]$ must be decomposed into $\mathsf{Employee}[\underline{\mathsf{EmpID}}, \mathsf{Salary}]$ and $\mathsf{Employee}[\underline{\mathsf{EmpID}}, \mathsf{Dept}]$, even though the composite is already in BCNF.

FKDs are of the form $R_1[F] \subseteq R_2[K]$, in which K is the (primary) key of R_2 and F is a set of attributes of R_1, called the *foreign key*. The case that F includes some key attributes of R_1 is not excluded. To preserve such a dependency in the component framework, both sides of the FKD must be contained in a single component. The convention that $R_2[K]$ be included in the component containing $R_1[F]$ is adopted. The satellite projection (e.g., $R_2[K]$) is then connected, via a port, to the corresponding component containing the full R_2 as the main relation. This is illustrated in three cases in Fig. 3.

Given a relational schema $\mathbf{D}$, a *simple key schema* is a set of projections of $\mathbf{D}$ with the following properties. First, it contains a *main relation* R, governed by a single key dependency; i.e., an FD which determines all other attributes. Second, it contains all projections onto their primary keys of the other relations R' in $\mathbf{D}$ for which the primary key of R' is a foreign key for R. It thus embodies the FKDs. All of the schemata in Fig. 3 are simple key schemata.

Convention 3.2 (The finite domain property). In that which follows for horizontal decomposition, it will always be assumed that each attribute A has the *finite domain property*; that is, the set $\mathsf{Dom}(A)$ of domain elements for A, the set of all possible values for attribute A, is finite. This condition is always met in real examples, and it simplifies the theory substantially by keeping the number of basic components in a horizontal decomposition finite.

Definition 3.3 (Views defined by selection). Just as projection is the defining operation for vertical decomposition, so too is selection the operation for horizontal decomposition. For a schema $\mathbf{D}$ whose relation symbols include $\{R_1, \ldots, R_k\}$, the notation $\sigma_\varphi\langle R_1, \ldots, R_k\rangle$ will be used to denote the selection φ applied to those relations. The *select view* $\Sigma_\varphi\langle R_1, \ldots, R_k\rangle$ has $\sigma_\varphi\langle R_1, \ldots, R_k\rangle$ as its underlying morphism.

The selects which will be applied to simple key schemata to obtain horizontal decompositions will always be of a particular form. The selection is defined only on the main relation, with all other relations "following" that select, based upon the embodied FKDs. The notation $\sigma_{(\varphi)^+}\langle R_1, \ldots, R_k\rangle$ will be used to denote such a selection, with $\Sigma_{(\varphi)^+}\langle R_1, \ldots, R_k\rangle$ the corresponding view. For example, considering the schema defined by the upper-left rectangle of Fig. 3, $\Sigma_{(\mathsf{EmpID}=\text{Alice})^+}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Dept}], \mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$ has selection morphism $\sigma_{((\mathsf{EmpID}=\text{Alice}))^+}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Dept}], \mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$ which is the same as the selection $\sigma_{(\mathsf{EmpID}=\text{Alice})\wedge(\mathsf{Dept}=\mathsf{DptID})}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Dept}], \mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$, while the selection $\sigma_{((\mathsf{EmpID}=\text{Alice})\wedge(\mathsf{Dept}=\text{Research}))^+}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Dept}], \mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$ is identical to $\sigma_{(\mathsf{EmpID}=\text{Alice})\wedge(\mathsf{Dept}=\text{Research})\wedge(\mathsf{Dept}=\mathsf{DptID})}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Dept}], \mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$.

If $\mathbf{D}$ is a simple key schema, *simple key select* on $\mathbf{E}$ is a select view $\Sigma_{(K=t)^+}\langle \mathbf{E}\rangle$, where $K = A_1 \ldots A_k$ is the key of the main relation of $\mathbf{D}$ and $t = (\mathrm{a}_1, \ldots, \mathrm{a}_k) \in \mathsf{Dom}(A_1) \times \ldots \times \mathsf{Dom}(A_k)$. Thus, in a simple key select, only a selection on the primary key of the main relation is allowed, with that selection extending to those parts of foreign keys which reference the primary key. For example, $\Sigma_{(\mathsf{EmpID}=\text{Alice})^{+}}\langle(\mathsf{E}[\underline{\mathsf{EmpID}},\mathsf{Dept}],\mathsf{D}[\utilde{\mathsf{DptID}}])\rangle$ is a simple key select while $\Sigma_{((\mathsf{EmpID}=\text{Alice})\wedge(\mathsf{Dept}=\text{Research}))^{+}}\langle(\mathsf{E}[\underline{\mathsf{EmpID}},\mathsf{Dept}],\mathsf{D}[\utilde{\mathsf{DptID}}])\rangle$ is not.

Simple key schemata may be decomposed on a key-by-key basis into simple key selects, with complete independence. The following observation, whose proof is immediate, recaptures this.

Observation 3.4 (Horizontal decomposition of a simple key schema). *Let* $\mathbf{D}$ *be a simple key schema with primary key* $K = A_1 \ldots A_k$*, and whose relations include* $\{R_1, \ldots, R_k\}$*. Then* $\{\Sigma_{(K=S)^{+}}\langle R_1, \ldots, R_k\rangle \mid S \in \mathsf{Dom}(A_1) \times \ldots \times \mathsf{Dom}(A_k)\}$ *is pairwise definable, with the ports the relativized zero views.* $\square$

Examples 3.5 (Combined horizontal and vertical decomposition). To visualize the nature of decomposition using simple key selects as the horizontal part of a vertical-horizontal decomposition, it is best to begin with a schema which is simpler and more regular than that running example $\mathbf{E}_0$. To that end, let $\mathbf{E}_2$ be as defined in Definition 2.3, having a vertical decomposition into the basic schema components $\{\Pi^{\mathbf{E}_2}_{AB}, \Pi^{\mathbf{E}_2}_{BC}, \Pi^{\mathbf{E}_2}_{CD}\}$, The combined horizontal and vertical decomposition is depicted in Fig. 4. Each horizontal decomposition is actually

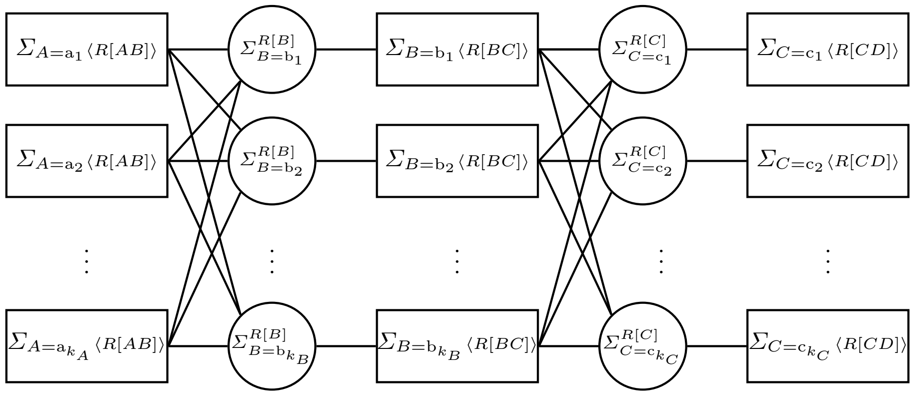
$\Sigma_{A=\mathrm{a}_1}\langle R[AB]\rangle$
$\Sigma_{A=\mathrm{a}_2}\langle R[AB]\rangle$
$\Sigma_{A=\mathrm{a}_{k_A}}\langle R[AB]\rangle$
$\Sigma^{R[B]}_{B=\mathrm{b}_1}$
$\Sigma^{R[B]}_{B=\mathrm{b}_2}$
$\Sigma^{R[B]}_{B=\mathrm{b}_{k_B}}$
$\Sigma_{B=\mathrm{b}_1}\langle R[BC]\rangle$
$\Sigma_{B=\mathrm{b}_2}\langle R[BC]\rangle$
$\Sigma_{B=\mathrm{b}_{k_B}}\langle R[BC]\rangle$
$\Sigma^{R[C]}_{C=\mathrm{c}_1}$
$\Sigma^{R[C]}_{C=\mathrm{c}_2}$
$\Sigma^{R[C]}_{C=\mathrm{c}_{k_C}}$
$\Sigma_{C=\mathrm{c}_1}\langle R[CD]\rangle$
$\Sigma_{C=\mathrm{c}_2}\langle R[CD]\rangle$
$\Sigma_{C=\mathrm{c}_{k_C}}\langle R[CD]\rangle$

Fig. 4. Visualization of the horizontal and vertical components for $\mathbf{E}_6$

represented by a vertical stack of components defined by simple key selects. Note that the ports are each selections on single elements. The horizontal components associated with $R[AB]$ and $R[BC]$ have connections to each projection on the non-key attribute.

Now return to the running example $\mathbf{E}_0$ of Fig. 3, the vertical component defined by $\mathsf{Employee}[\underline{\mathsf{EmpID}},\mathsf{Salary}]$. There is one horizontal component for each element of $\mathsf{Dom}(\mathsf{EmpID})$. To update the salary of Alice, only the component

$\Sigma_{(\mathsf{EmpID}=\text{Alice})^+}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Salary}]\rangle$ need be claimed; the salary of any other employee may be updated independently.

Next consider changing the department of Alice. The relevant component is $\Sigma_{((\mathsf{EmpID}=\text{Alice}))^+}\langle \mathsf{E}[\mathsf{EmpID}, \mathsf{Dept}], \mathsf{D}[\mathsf{DptID}]\rangle$ Again, only the tuple corresponding to Alice need be claimed; all others are available to other transactions. However, the entire set of names of departments is read claimed, since $\Sigma_{((\mathsf{EmpID}=\text{Alice}))^+}\langle \mathsf{E}[\mathsf{EmpID}, \mathsf{Dept}], \mathsf{D}[\mathsf{DptID}]\rangle$ has a port connecting to each horizontal component of $\mathsf{Department}[\underline{\mathsf{DptID}}, \mathsf{Location}]$. Thus, while it is possible to execute other updates concurrently which read the list of department names, it is not possible for any transaction to update this list. More will be said about this point in Discussion 3.6.

Finally, suppose that a transaction wishes to add $(\text{French}, 1, \text{DuBois})$ and $(\text{German}, 1, \text{Dimpflmeier})$ to $\mathsf{Mentor}[\underline{\mathsf{Area}, \mathsf{Level}}, \mathsf{MntID}]$. The constraint $\mathsf{MntID} \to \mathsf{Area}$ must be verified to support this change. This update may be effected with support from a write claim for $\Sigma_{((\mathsf{Area}\in\{\text{French},\text{German}\})\wedge(\mathsf{Level}=1))^+}\langle \mathsf{Mentor}[\underline{\mathsf{Area}, \mathsf{Level}}, \mathsf{MntID}]\rangle$, which is the join $\Sigma_{((\mathsf{Area}=\text{French})\wedge(\mathsf{Level}=1))^+}\langle \mathsf{Mentor}[\underline{\mathsf{Area}, \mathsf{Level}}, \mathsf{MntID}]\rangle$
$\sqcup\, \Sigma_{((\mathsf{Area}=\text{German})\wedge(\mathsf{Level}=1))^+}\langle \mathsf{Mentor}[\underline{\mathsf{Area}, \mathsf{Level}}, \mathsf{MntID}]\rangle$
of two simple key selects. It also effects a read claim on the port defined by $\mathsf{Mentor}[\mathsf{Area}, \underline{\mathsf{MntID}}]$. The transaction read claims this port, and hence the entire component of the same name. If DuBois is a already a mentor for French, and Dimpflmeier is already a mentor for German (as recorded in $\mathsf{Mentor}[\mathsf{Area}, \underline{\mathsf{MntID}}]$), then no further claims are necessary. However, if this is not the case, then $\mathsf{Mentor}[\mathsf{Area}, \underline{\mathsf{MntID}}]$ must be write claimed for DuBois and Dimpflmeier as well. Since this read claims all Area values in $\mathsf{Mentor}[\mathsf{Area}, \underline{\mathsf{MntID}}]$, this effectively claims the entire Mentor relation, and no parallel updates are possible (although much may still be read).

Discussion 3.6 (Range restriction). As illustrated in Examples 3.5, claiming a relation even for one key value claims every non-key value. For example, for a transaction T_1 to place new employee Alice in the Research department, the view $\Sigma_{(\mathsf{EmpID}=\text{Alice})^+}\langle \mathsf{E}[\mathsf{EmpID}, \mathsf{Dept}], \mathsf{D}[\mathsf{DptID}]\rangle$ must be claimed, which reserves every possible value for Dept as a new value for the tuple with key Alice. Thus, a second transaction T_2, whose task it is to delete the Education department, could not proceed independently because it is not known that T_1 will not change the department for Alice to Education. It might seem that a solution would be for T_1 to claim only $\Sigma_{((\mathsf{EmpID}=\text{Alice})\wedge(\mathsf{Dept}=\text{Research}))^+}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Dept}], \mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$ However, that would open the door for another transaction to put Alice into another department, say TechSupport, concurrently, since it is impossible to verify within $\Sigma_{((\mathsf{EmpID}=\text{Alice})\wedge(\mathsf{Dept}=\text{Research}))^+}\langle \mathsf{E}[\underline{\mathsf{EmpID}}, \mathsf{Dept}], \mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$ whether the FD $\mathsf{EmpID} \to \mathsf{Dept}$ is satisfied for $\mathsf{EmpID} = \text{Alice}$. Since the whole purpose of this approach is to detect and prevent such conflicts and support truly independent updates, simply ignoring this conflict is not acceptable. Fortunately, there is a solution, provided the transaction provides a bit more information. Roughly, the idea is that the transaction claims the data object

$\Sigma_{((\mathsf{EmpID}=\text{Alice})\wedge(\mathsf{Dept}=\text{Research}))^+}\langle\mathsf{E}[\underline{\mathsf{EmpID}},\mathsf{Dept}],\mathsf{D}[\utilde{\mathsf{DptID}}]\rangle$ as suggested, and for the life of that claim, the system excludes any other possibilities for tuples with EmpID = Alice. This does not reduce possible parallelism in any way, since only one transaction may have write privileges on a field of a tuple with a given key. Furthermore, it allows the transaction which is to delete the Education department to proceed independently (provided that no other employee works in that department). This approach may be used to increase the possible concurrency for the examples involving the Mentor at the end of Examples 3.5 as well.

It is a straightforward exercise to extend the framework of Sec. 2 to incorporate these additional features, called *range restriction*. Unfortunately, space limitation preclude a full presentation here.

4 Conclusions and Further Directions

A theory of updateable data objects has been presented. A key of this approach is that transactions may claim data objects which overlap on read areas. It requires more structural analysis of the schema, but in return it supports a finer grain of concurrency than in achievable with traditional models. It is particularly suited to interactive applications, which can tolerate more preprocessing for a transaction to begin but are much more sensitive to long waits and aborts.

Important further directions include the following.

Application to cooperative update: An approach to the support of cooperative updates which is based upon components has been developed recently [11], [10]. Indeed, the ideas for this paper began as a search for appropriate ways to support concurrency in such an environment, and the topic will be developed further.

Application to nondeterministic update requests: A key aspect of the approach to cooperative updates in [11] and [10] is that requests may be *nondeterministic*; that is, a user may request that one of a number of alternative updates be supported. For example, a business travel request may involve alternatives regarding hotel, flight, date, *etc.*. Some of these alternatives may become impossible as the processing of the transaction proceeds. The model of conflict developed here seems well suited to identifying and eliminating parts of the nondeterministic request which conflict with other needs and thus cannot be supported, while allowing the others to proceed.

Acknowledgement: Some of the ideas reported here have their origins in discussions with Peggy Schmidt. Her contribution is gratefully acknowledged.

References

1. A. V. Aho, C. Beeri, and J. D. Ullman. The theory of joins in relational databases. *ACM TODS*, 4(3):297–314, 1979.

2. C. Beeri, R. Fagin, D. Maier, and M. Yannakakis. On the desirability of acyclic database schemes. *JACM*, 30(3):479–513, 1983.
3. P. A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, 1987.
4. R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems.* Addison Wesley, fifth edition, 2006.
5. K. P. Eswaran, J. Gray, R. A. Lorie, and I. L. Traiger. The notions of consistency and predicate locks in a database system. *Comm. ACM*, 19(11):624–633, 1976.
6. S. J. Hegner. An order-based theory of updates for closed database views. *Ann. Math. Art. Intell.*, 40:63–125, 2004.
7. S. J. Hegner. The complexity of embedded axiomatization for a class of closed database views. *Ann. Math. Art. Intell.*, 46:38–97, 2006.
8. S. J. Hegner. A model of database components and their interconnection based upon communicating views. In H. Jakkola, Y. Kiyoki, and T. Tokuda, editors, *Information Modelling and Knowledge Systems XIX*, Frontiers in Artificial Intelligence and Applications, pages 79–100. IOS Press, 2008.
9. S. J. Hegner. Semantic bijectivity and the uniqueness of constant-complement updates in the relatiional context. In K.-D. Schewe and B. Thalheim, editors, *International Workshop on Semantics in Data and Knowledge Bases, SDKB 2008, Nantes, France, March 29, 2008, Proceedings*, volume 4925 of *Lecture Notes in Computer Science*, pages 172–191. Springer-Verlag, 2008.
10. S. J. Hegner. A simple model of negotiation for cooperative updates on database schema components. In Y. Kiyoki, T. Tokuda, A. Heimbürger, H. Jaakkola, and N. Yoshida., editors, *Frontiers in Artificial Intelligence and Applications XX11*, 2011. in press.
11. S. J. Hegner and P. Schmidt. Update support for database views via cooperation. In Y. Ioannis, B. Novikov, and B. Rachev, editors, *Advances in Databases and Information Systems, 11th East European Conference, ADBIS 2007, Varna, Bulgaria, September 29 - October 3, 2007, Proceedings*, volume 4690 of *Lecture Notes in Computer Science*, pages 98–113. Springer-Verlag, 2007.
12. M. Herlihy and W. E. Weihl. Hybrid concurrency control for abstract data types. *J. Comput. System Sci.*, 43(1):25–61, 1991.
13. A. Kemper and A. Eickler. *Datenbanksysteme.* Oldenbourg, 6. Auflage, 2006.
14. A. C. Klug. Locking expressions for increased database concurrency. *J. Assoc. Comp. Mach.*, 30(1):36–54, 1983.
15. C. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, 1986.
16. J. Paredaens, P. De Bra, M. Gyssens, and D. Van Gucht. *The Structure of the Relational Database Model.* Springer-Verlag, 1989.
17. M. C. Sampaio and S. Turc. Cooperative transactions: A data-driven approach. In *29th Annual Hawaii International Conference on System Sciences (HICSS-29), January 3-6, 1996, Maui, Hawaii*, pages 41–50. IEEE Computer Society, 1996.
18. W. Wieczerzycki. Transaction management in databases supporting collaborative applications. In W. Litwin, T. Morzy, and G. Vossen, editors, *Advances in Databases and Information Systems, Second East European Symposium, ADBIS'98, Poznan, Poland, Spetember 7-10, 1998, Proceedings*, volume 1475 of *Lecture Notes in Computer Science*, pages 107–118. Springer, 1998.