Characterising relational view updates using relative information capacity

Nigel Stanger
Department of Information Science, University of Otago
Dunedin, New Zealand
nigel.stanger@otago.ac.nz

ABSTRACT

In 2013, Chris Date published a book about the view update problem in the context of the Relational Model. He presented several detailed examples of different varieties of view updates and characterised their behaviour, in the process deriving a set of principles for updating views based on the notion of *information equivalence* of view schemas. In this paper we discuss work in progress that examines how Date's operational definition of information equivalence can be formally characterised using Hull's concept of *relative information capacity*. As a proof of concept, we use an extension of Miller's *schema intension graph* formalism to model the information equivalence of Date's restriction view example.

Keywords

view update problem, information equivalence, relative information capacity, schema intension graph, schema transformation

1. INTRODUCTION

The view update problem is a long-standing problem of great practical significance in the database field. Date [10] likens it to significant unsolved problems in other fields such as the Riemann Hypothesis in mathematics or "$P = NP$?" in computational complexity theory. In his 2013 book [10] Date developed a set of principles for updating relational views, based on the notion of *information equivalence*. However, that notion was defined more operationally than formally.

View updating is a special case of schema translation: we have one schema comprising a set of views, another schema comprising a set of base relations, and we need to translate updates from one to the other.[1] It therefore makes sense that formalisms developed to characterise schema translations are also applicable to view updates. We propose to use Hull's notion of *relative information capacity* [13] as a means to formally characterise the information equivalence of the schemas that make up a given view configuration.

[1]These schemas may be actually separate, or more likely may be sub-schemas of a larger database schema, with effective separation achieved via the access privilege mechanism.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACSW '17, January 31-February 03, 2017, Geelong, Australia

© 2017 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-4768-6/17/01…$15.00

DOI: http://dx.doi.org/10.1145/3014812.3014847

In particular, we develop an extension of Miller's *schema intension graph* (SIG) formalism, which was originally designed to facilitate the comparison of relative information capacity across schemas. We first extend the SIG formalism with relevant relational concepts such as relation types, tuple types, and key constraints. We then show a proof of concept of how this extended formalism can be used to model the information equivalence of a view configuration, by applying it to Date's example of disjoint restriction views [10]. Our results are consistent with Date's operational characterisation of information equivalence for this example.

In Section 2, we briefly review the view update problem and relevant research, and in Section 3 we introduce some core concepts that are used throughout the remainder of the paper. In Section 4 we discuss relative information capacity and the SIG formalism, then explain in Section 5 how to extend the SIG formalism to more precisely model a relational schema. In Section 6 we analyse Date's restriction view example [10] in detail, and characterise the information equivalence of its various sub-schemas. We discuss conclusions and future work in Section 7.

2. THE VIEW UPDATE PROBLEM

The view update problem was first discussed in the relational context by Codd in 1974 [6]. When an update operation is applied to a view, it needs to be translated into corresponding updates on the base relations from which the view is derived [14]. Sometimes there are multiple translations that can produce the desired view update, but produce different—possibly incompatible—changes in the base relations. Worse, there may be translations that have side effects (e.g., for some views based on joins), which produce a different result in the view than if the updates were applied directly to an equivalent base relation [14]. The problem then is how to choose the most appropriate translation. As yet there has been no solution that completely and automatically resolves this ambiguity.

There has been considerable research into this problem over the last four decades, and numerous different approaches have been suggested in the context of relational, non-relational (e.g., [21, 29, 31, 33]), and deductive databases (e.g., [2, 3, 7, 24]). We shall now briefly review some key developments in the relational context. We leave consideration of non-relational and deductive contexts for future work.

Banchilhon and Spyratos proposed the concept of *constant complement* [1] in 1981. The *complement* of a view describes all the information from the original database that does not appear in the view, so composing a view with its complement provides enough information to reconstruct the original database. Keeping the complement invariant under view update facilitates translation construction. This is a useful approach that has been developed further by other authors [12, 19], but it has been shown that there

are reasonable translations that cannot be carried out under constant complement [16]. Computing the complement of a view is also difficult, and several authors have worked on more efficient algorithms [8, 18, 20]. Finally, this approach is purely structural in nature and does not consider the semantics of the schema, which is necessary to properly resolve translation ambiguity [15].

Gottlob et al. generalised the idea of constant complement into the class of *consistent views* [11]. They define consistent views as having the property that "if the effect of a view update on a view state is determined, then the effect of the corresponding database update is unambiguously determined" [11]. However, there are still reasonable views that are not consistent under this definition, and this approach again does not consider schema semantics.

Keller noted that semantic information is essential in order to choose an appropriate translation [15], and suggested that the database administrator (DBA) could provide this at view definition time [14]. Masunaga argued, however, that this would be insufficient to completely resolve ambiguity, and suggested collecting further semantic information from end users at update time [23]. Shu [32] discussed an interesting approach to this kind of information gathering, by treating the problem of generating suitable view translations as a constraint satisfaction problem. This enabled constraints to be added incrementally at view update time. However, while collecting such information from the DBA seems reasonable, collecting it from the end user at update time could be seen as too burdensome, and could impact the usability of query interfaces. It is also unclear which information needs to be captured, and how best to capture it.

Kotidis et al. [17] discussed a different approach to avoiding side effects in view translations. They proposed creating a materialised copy of the view that could be updated directly, rather than mapping updates from the (virtual) view onto its underlying base tables. Clearly this enables the view to be updated without translation side effects, but does imply that the physical manifestation (extension) of the view may no longer match its defining query (intension).

Finally, Date has published several books on the theoretical foundations of the Relational Model over the last decade, including one specifically targeting the view update problem [10] (indeed, the book is ambitiously subtitled "Solving the View Update Problem"). He developed a comprehensive set of rules for updating views in a relational context, supported by an array of worked examples. Date advocated explicitly stating all relevant constraints at view creation time, similar to the information gathering approaches discussed above [14, 23, 32]. He argued that if a complete set of constraints was available, then ambiguity would be diminished or eliminated, and it would then be possible to automatically generate a set of appropriate *compensatory rules* to automatically manage the view update process. (Compensatory rules are similar in principle to database triggers, except that they are intended to be declarative so that their definitions can made explicitly visible to end users.)

Date's work is interesting and compelling, and raises several questions worthy of further investigation. However, there are still situations where the correct view translation may be ambiguous, e.g., views based on certain types of join, as previously noted [14]. Somewhat disappointingly, given his stated goal of "solving" the view update problem, Date took a pragmatic approach in such cases. While his justification was well argued and logical, even he acknowledged that it involved "a certain degree of pragma" [10].

Date characterises the ability to define an appropriate and effective view translation using the notion of *information equivalence*. Two schemas $\mathcal{S}_1$ and $\mathcal{S}_2$ are said to be information equivalent "if and only if the constraints that apply to $[\mathcal{S}_1]$ and $[\mathcal{S}_2]$ are such that every proposition that can be represented by $[\mathcal{S}_1]$ can be represented by $[\mathcal{S}_2]$ and vice versa" [10]. Date's information equivalence is a formal concept, but its definition and characterisation in [10] are more operational in nature. This is what we explore in this paper.

3. RELATIONS AND TYPES

We generally follow Date's [9] notation and terminology regarding the Relational Model. We do not use Date's Tutorial D notation, however, preferring instead a more classical algebraic style.

In this paper, A represents an attribute (comprising a *name*:*type* pair), R represents a relation variable (*relvar*), and v represents a value of an attribute, tuple, relation, etc. Attribute values may be disambiguated where necessary by prefixing the value with the attribute name, i.e., *name*: *value*.

Date defines a *type* T_i as "essentially a *named, finite set of values*" [9], representing every possible value of some specific kind, e.g., every possible ASCII string of length 5, every possible city name, or every possible relation value with a particular heading. That is:

$$T_i = \{v_1, v_2, \ldots, v_m\}, 0 < m < k \tag{1}$$

where k is some finite upper bound. Thus, the type of attribute A_i (denoted T_{A_i}) is by definition the set of all possible values of A_i.

Consider a relvar R with heading $\{A_1, A_2, \ldots, A_n\}$. Following from equation 1, the *relation type* of R, which we denote here as T_R, is the set of all possible relation values with R's heading. Similarly, the *tuple type* of R's tuples, which we denote here as $T_{\{R\}}$, is the set of all possible tuple values with R's heading, which equates to the Cartesian product of R's attribute types, i.e.:

$$T_{\{R\}} = T_{A_1} \times T_{A_2} \times \cdots \times T_{A_n}. \tag{2}$$

(This construction can be applied to any tuple type, including projections of other tuple types.)

By definition any given value of a relvar R must comprise a set (possibly empty) of tuples $\{t_1, t_2, \ldots, t_m\} \subseteq T_{\{R\}}$. T_R therefore comprises every possible k-combination of the elements of $T_{\{R\}}$, for $0 \leq k \leq \left|T_{\{R\}}\right|$.

For example, Date's usual suppliers-and-parts schema [10] includes a relvar S with heading $\{Sno, Sname, Status, City\}$, which contains details of suppliers. From equation 2, the tuples of S have type $T_{\{S\}} = T_{Sno} \times T_{Sname} \times T_{Status} \times T_{City}$.

This notion of type as a set of possible values is synonymous with Codd's original definition of domains in the Relational Model [4, 9], and also with the notion of domains as "sets of values" in schema intension graphs [27, 28]. The implications of the latter point will be explored further in Section 5, after we have introduced relative information capacity and schema intension graphs in Section 4.

3.1 Nulls

We also follow Date's prohibition of nulls. Philosophical objections aside, however, this makes little practical difference to our approach, as we are dealing exclusively with types rather than attributes anyway. While an attribute could be null, it makes little sense to say that a *type* could be "null". It could be claimed that a type "includes" nulls, but it is unclear what this would even mean in practice, and runs the risk of treating nulls like "normal" values rather than the special markers that they are. In any case, this would effectively be no different from the situation without nulls, so we omit further consideration of nulls in the interest of clarity.

4. RELATIVE INFORMATION CAPACITY

A schema conveys information about the phenomenon it models. Hull [13] defined the concept of *relative information capacity* to describe the information content of different schemas. The relative information capacity of a schema determines the set of all valid

instances of that schema [13, 25, 26]. Relative information capacity, as its name implies, is not a quantitative measure; rather it is an indicator of the relative information content of different schemas.

Relative information capacity provides a means to identify the differences between schema instances, and can therefore be used as an aid to schema integration and translation [25]. As noted earlier, view updating is a special case of this, so it follows that information capacity can also be used to characterise the information content of views compared to the base relvar(s) from which they are derived.

Miller et al. [26] defined relative information capacity in terms of two types of mapping. An *information (capacity) preserving mapping* from schema $\mathcal{S}_1$ to schema $\mathcal{S}_2$ maps every element of $\mathcal{S}_1$ to an element of $\mathcal{S}_2$, but only some elements of $\mathcal{S}_2$ to elements of $\mathcal{S}_1$. $\mathcal{S}_2$ has an information capacity greater than $\mathcal{S}_1$, and is said to *dominate* $\mathcal{S}_1$, denoted $\mathcal{S}_1 \preceq \mathcal{S}_2$. An *equivalence preserving mapping* from $\mathcal{S}_1$ to $\mathcal{S}_3$ maps every element of $\mathcal{S}_1$ to an element of $\mathcal{S}_3$ and vice versa (i.e., both the mapping and its inverse are information preserving). The information capacity of $\mathcal{S}_1$ is identical to $\mathcal{S}_3$, and the schemas are said to be *equivalent*, denoted $\mathcal{S}_1 \equiv \mathcal{S}_3$.

(Note that Miller et al. [27] use stricter definitions of equivalence and dominance than those used here; strictly speaking, the discussion in this section should refer to *absolute* dominance and equivalence, whereas the following discussion on SIG transformations should refer to *internal* dominance and equivalence [27]. The distinction, however, is not crucial to the discussion in this paper.)

Relative information capacity can only be used comparatively. To this end, Miller et al. [28] defined the *schema intension graph* formalism as a tool for comparing the information capacity of schemas. A schema intension graph (SIG) describes the associations between domains in a schema, and provides a way to "reason about constraints on *collections of entities* in an instance of the entity set, rather than about the internal structure of a single entity" [28]. In particular, SIGs provide a useful abstract formalism within which to perform schema transformations, as transforming the SIG for a schema is effectively the same as transforming the schema itself.

4.1 Constructing SIGs

A SIG is a graph $G = (N, E)$ [27]. The nodes $n_i \in N$ represent typed domains (i.e., sets of values). Nodes may be *simple* (atomic) or *constructed* (composite). Simple nodes are denoted by the name of the domain that they represent. Constructed nodes represent a collection of domains and are built using the $\times$ (Cartesian product) and + (union) operators.

The edges $e_i \in E$ represent binary relations (in the mathematical sense) between domains. Edges are denoted by lines with an optional label, and may be composed to form a *path* between two nodes (the notation $e_2 \circ e_1$ means "edge e_1 followed by edge e_2"). Some edges may be designated as *selection* edges, labelled σ, or *projection* edges, labelled π. These correspond to the relational algebra operators of the same names [5].

Edges may also be annotated with up to four properties denoting simple constraints on the binary relation represented by the edge: totality, surjectivity, functionality and injectivity. These properties are defined as follows [27]:

> Let A and B be two sets. A binary relation $r : A \multimap B$ is *total* (denoted $e : A \mathrel{+\!\!-} B$) if it is defined for all elements of A; *surjective* ($e : A \mathrel{-\!\!+} B$) if it is defined for all elements of B; *functional* ($e : A \longrightarrow B$) if an element of A determines at most one element of B; and *injective* ($e : A \longleftarrow B$) if an element of B determines at most one element of A.

A *bijection* is a binary relation that has all four properties [28].

The use of the term "binary relation" here, while mathematically correct, is potentially confusing in any discussion that also involves the Relational Model. From now on, we shall therefore refer to the binary relations represented by SIG edges as "associations".

A selection edge $e : A \xrightarrow{\sigma} B$ indicates that the elements of B comprise a subset or specialisation of the elements of A. If $B = A$, we have what we term a *trivial* selection. Every element of B must appear in A, and vice versa, and every element of A is associated with exactly one element of B, and vice versa. Trivial selection edges are therefore bijective, i.e., $A \xleftrightarrow{\sigma} B$. If $B \subset A$, we have what we term a *non-trivial* selection. Every element of B must appear in A, but not vice versa. The mapping remains one-to-one, as before. From the perspective of the superset A, non-trivial selection edges are therefore surjective, functional, and injective, i.e., $A \xleftarrow{\sigma} B$.

A projection edge $e : A \xrightarrow{\pi} B$ indicates that the elements of B are a projection of A, where A is a constructed node of the form $C \times D \times \cdots$ (i.e., a tuple type). A bijective projection edge $A \xleftrightarrow{\pi} B$ indicates that B is unique with respect to A. For non-unique projections, every element of A must match an element in B, and vice versa. Every element of A is also associated with at most one element of B, but not vice versa. Non-unique projection edges are therefore total, surjective, and functional, i.e., $A \xrightarrow{\pi} B$.

4.2 Comparing SIGs

It is possible to determine the relationship between the relative information capacities of two schemas by comparing SIGs for the two schemas. Consider two schemas $\mathcal{S}_1$ and $\mathcal{S}_2$, with corresponding SIGs $G_{\mathcal{S}_1}$ and $G_{\mathcal{S}_2}$. Intuitively, if $G_{\mathcal{S}_1}$ and $G_{\mathcal{S}_2}$ are isomorphic, then $\mathcal{S}_1$ and $\mathcal{S}_2$ are equivalent [28]. $G_{\mathcal{S}_1}$ and $G_{\mathcal{S}_2}$ are considered isomorphic if they have identical structure and semantics, i.e., the same nodes, edges, node types, constraints, and annotations.

Miller et al. [28] identify three possible results from comparing two SIGs $G_{\mathcal{S}_1}$ and $G_{\mathcal{S}_2}$:

1. $G_{\mathcal{S}_1}$ and $G_{\mathcal{S}_2}$ are either immediately isomorphic, or one can be manipulated by a series of equivalence preserving transformations (described below) until it is isomorphic to the other. The two schemas are equivalent, i.e., $\mathcal{S}_1 \equiv \mathcal{S}_2$.

2. $G_{\mathcal{S}_1}$ and $G_{\mathcal{S}_2}$ are not immediately isomorphic, but one can be manipulated by a series of information capacity augmenting transformations (described below) until it is isomorphic to the other. The transformed schema is dominated by the other, e.g., if we transform $G_{\mathcal{S}_1}$, then $\mathcal{S}_1 \preceq \mathcal{S}_2$.

3. $G_{\mathcal{S}_1}$ and $G_{\mathcal{S}_2}$ are not immediately isomorphic and cannot be transformed to be isomorphic. The information capacities of $\mathcal{S}_1$ and $\mathcal{S}_2$ cannot be compared.

Miller et al. [27] describe the following equivalence preserving and information capacity augmenting transformations on SIGs:

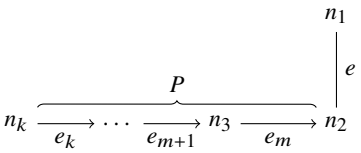
Annotation transformations allow the removal of annotations or projection and selection constraints from edges (e.g., $n_1 \longleftrightarrow n_2 \Rightarrow n_1 \longrightarrow n_2$, or $n_3 \xleftrightarrow{\sigma} n_4 \Rightarrow n_3 \longleftrightarrow n_4$). If an annotation transformation on $G_{\mathcal{S}_1}$ makes it isomorphic to $G_{\mathcal{S}_2}$, then the information capacity of $\mathcal{S}_2$ dominates that of $\mathcal{S}_1$ (i.e., $\mathcal{S}_1 \preceq \mathcal{S}_2$).

Composition transformations allow edges to be moved and copied around the graph. Consider an edge $e : n_1 \multimap n_2$, as shown in Figure 1(a). If there is a path $P = e_m \circ \cdots \circ e_k$ of functional edges from some other node n_k to n_2, e may be copied to a new edge $e' : n_1 \multimap n_k$, as shown in Figure 1(b).

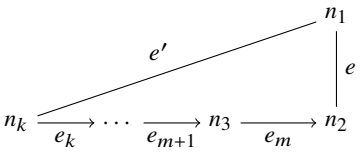
The original e can then be deleted if desired, effectively moving e to e' as shown in Figure 1(c).

The annotation of e' is determined by composing P with e, i.e., $e \circ P$. If an edge with a particular annotation (such as surjectivity) is composed with an edge without that annotation, the resulting edge also does not have the annotation. For example, the composition of the edges $\longrightarrow$ and $\rightarrowtail$ would be an edge with no annotations ($\multimap$), whereas the composition of $\longrightarrow$ and $\twoheadrightarrow$ would be $\longrightarrow$.

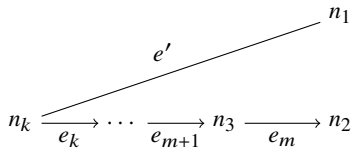
If a composition transformation on $G_{\mathcal{S}_1}$ makes it isomorphic to $G_{\mathcal{S}_2}$, then $\mathcal{S}_1 \preceq \mathcal{S}_2$.



(a) Original state



(b) Copy edge e to e' across path P



(c) Move edge e to e' across path P

Figure 1: Composition transformations.

Selection transformations allow the creation and deletion of new nodes and edges as follows:

1. An existing node n may be duplicated to create a new node n'. A trivial selection edge $e : n \overset{\sigma}{\leftarrowtail\!\!\!\!\twoheadrightarrow} n'$ is added to enforce the constraint that n' is a duplicate of n.
2. A node n_1 may be deleted if it is associated with exactly one other node n_2 by a single trivial selection edge $e : n_1 \overset{\sigma}{\leftarrowtail\!\!\!\!\twoheadrightarrow} n_2$ (implying n_1 is a duplicate of n_2). This also deletes the trivial selection edge.
3. A trivial selection edge $e : n \overset{\sigma}{\leftarrowtail\!\!\!\!\twoheadrightarrow} n$ may be created that connects a node n to itself.
4. A trivial selection edge $e : n_1 \overset{\sigma}{\leftarrowtail\!\!\!\!\twoheadrightarrow} n_2$ may be deleted, removing the association between n_1 and n_2.

If we can make $G_{\mathcal{S}_1}$ isomorphic to $G_{\mathcal{S}_2}$ using any of the first three selection transformations, then $\mathcal{S}_1 \equiv \mathcal{S}_2$. If we can make $G_{\mathcal{S}_1}$ isomorphic to $G_{\mathcal{S}_2}$ using the fourth selection transformation, then $\mathcal{S}_1 \preceq \mathcal{S}_2$. The latter may seem somewhat counter-intuitive [30], but is nonetheless correct [27]. A trivial selection edge $e : n_1 \overset{\sigma}{\leftarrowtail\!\!\!\!\twoheadrightarrow} n_2$ denotes that n_1 and n_2 correspond to the same domain, so removing the edge also removes this constraint. The information capacity of $\mathcal{S}_2$ must therefore be greater than $\mathcal{S}_1$.

These SIG transformations and their effects are summarised in Table 1.

Transformation	**Type**	**Effect on information capacity**
Delete annotation or π/σ constraint	annotation	augmented
Move or copy edge	composition	augmented
Create or delete duplicate node	selection	no change
Create trivial selection edge	selection	no change
Delete trivial selection edge	selection	augmented

Table 1: Summary of SIG transformations.

5. EXTENDING SIGS TO MODEL RELATIONAL CONCEPTS

The direct correspondence between relational types and SIG domains noted in Section 3 enables us to use SIGs to represent the structure and constraints of relational schemas. Equating tuple types to the Cartesian product of their attributes (equation 2) means that these can also be mapped directly to a constructed SIG node.

Let us again consider the relvar $S\{Sno, Sname, Status, City\}$ from Date's suppliers-and-parts schema [10]. In addition to type constraints on the attributes, tuples, and the relvar as a whole, S is subject to a (primary) key constraint on *Sno*. Taken together with the collection of types T_{Sno}, T_{Sname}, T_{Status}, T_{City}, $T_{\{S\}}$, and T_S, and assuming that S is the only relvar in the schema, we can construct the SIG shown in Figure 2 as explained below.

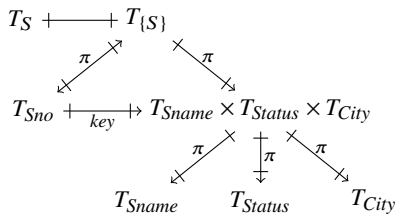


Figure 2: SIG for relvar $S\{Sno, Sname, Status, City\}$.

As discussed in Section 3, any given value of S must comprise a subset of $T_{\{S\}}$, so the association between T_S and $T_{\{S\}}$ is total. Similarly, every tuple in $T_{\{S\}}$ must appear in at least one value of S, so the association is also surjective. The association is neither functional nor injective, however, as a value of S in general may contain multiple tuples, and each tuple in $T_{\{S\}}$ can appear in multiple values of S. This gives us $T_S \mathrel{+\!\!+} T_{\{S\}}$.

The key constraint on *Sno* implies that the functional dependency $\{Sno\} \rightarrow \{Sname, Status, City\}$ holds in S. From equation 2, we can infer the type of $\{Sname, Status, City\}$ as $T_{Sname} \times T_{Status} \times T_{City}$. The association is functional and not injective by definition. Every supplier number (when used in a value of S) must be associated with an element of $T_{Sname} \times T_{Status} \times T_{City}$, and vice versa. This implies that the association is both total and surjective. Indeed, this holds for every edge that represents a key constraint [28]. We introduce the edge label *key* to denote an edge representing a key constraint, which gives us $T_{Sno} \overset{key}{\twoheadrightarrow\!\!\!\!+} T_{Sname} \times T_{Status} \times T_{City}$ in Figure 2.

Both T_{Sno} and $T_{Sname} \times T_{Status} \times T_{City}$ are projections of $T_{\{S\}}$, so we can add a non-unique projection edge $T_{\{S\}} \overset{\pi}{\twoheadrightarrow\!\!\!\!+} T_{Sname} \times T_{Status} \times T_{City}$ and a unique projection edge $T_{\{S\}} \overset{\pi}{\leftarrowtail\!\!\!\!\twoheadrightarrow} T_{Sno}$. Similarly, T_{Sname}, T_{Status}, and T_{City} are all projections of $T_{Sname} \times T_{Status} \times T_{City}$, so

we can add further non-unique projection edges to these nodes, as shown in Figure 2.

Looking at Figure 2, it could be argued that the $T_{...}$ type notation is redundant and unnecessarily complicates the diagram. Why not just use, e.g., S instead of T_S, Sno instead of T_{Sno}, and $Sno \times Sname \times Status \times City$ instead of $T_{\{S\}}$? It is important to remember, however, that SIGs like that shown in Figure 2 refer to multiple elements of quite different kinds, including relations, tuples, and various combinations of attributes. Discarding the type notation might be reasonable in informal situations, but could lead to confusion between the different kinds of elements represented. The explicit type notation protects the semantics of the various elements of the schema and makes it clear what is being represented.

6. CHARACTERISING VIEW UPDATES

In this section we show how the extended SIG formalism can be used to model the information equivalence of relational view updates, using Date's restriction view example as a proof of concept [10].

Views and the base relvars they are derived from normally exist within the same "base" database schema (although this is not strictly required), which we shall refer to as $\mathcal{S}_0$. However, an end user may be granted access only to the views, and thus not be aware of the existence of the base relvar(s), or vice versa. This effectively partitions the schema into logical sub-schemas $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_n$.

There are explicit and implicit constraints in $\mathcal{S}_0$ that should also apply to the sub-schemas [10]. Ignoring these constraints in a sub-schema $\mathcal{S}_k$ will affect its information capacity, and could negatively impact the information equivalence of view updates involving $\mathcal{S}_k$. In the next section we will show how such constraints can be propagated into sub-schemas by rewriting them in terms of the sub-schema alone, which enables us to explicitly incorporate into the sub-schema semantic information implicit in $\mathcal{S}_0$ that would otherwise be lost. We will then discuss the construction of SIGs for Date's restriction view example (Section 6.2), and show how to transform and compare these (Section 6.3).

6.1 Constraint derivation and propagation

Date's motivating example in Chapter 4 of [10] explores what is arguably the simplest case of view updating: disjoint restriction views. The example includes a base relvar S and two views LS and NLS derived[2] from S:

$$LS = \sigma_{City='London'} S \tag{3}$$

$$NLS = \sigma_{City\neq'London'} S \tag{4}$$

Informally, LS and NLS contain details of London and non-London suppliers, respectively. They represent disjoint restrictions of S with the same heading, and have the following tuple types:

$$T_{\{LS\}} = T_{Sno} \times T_{Sname} \times T_{LS} \times \{City\colon \text{`}London\text{'}\}$$
$$T_{\{NLS\}} = T_{Sno} \times T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$$

The base schema for this example is $\mathcal{S}_0 = \{S, LS, NLS\}$. This is subject to type constraints as specified by the various types noted previously, and the following additional constraints:

C_1 Sno is a key for each of S, LS, and NLS, therefore the functional dependency $\{Sno\} \rightarrow \{Sname, Status, City\}$ also holds in each of (a) S; (b) LS; and (c) NLS.

C_2 $\pi_{\{Sno\}} LS \subseteq \pi_{\{Sno\}} S$ [foreign key from LS to S]

C_3 $\pi_{\{Sno\}} NLS \subseteq \pi_{\{Sno\}} S$ [foreign key from NLS to S]

C_4 $S = LS \cup NLS$ [from (3) and (4) above]

C_5 $\sigma_{City\neq'London'} LS = \emptyset$ [from (3) above]

C_6 $\sigma_{City='London'} NLS = \emptyset$ [from (4) above]

C_7 $LS \cap NLS = \emptyset$ [from C_5 and C_6]

C_8 (a) $T_S = T_{LS} \cup T_{NLS}$; (b) $T_{\{S\}} = T_{\{LS\}} \cup T_{\{NLS\}}$ [from C_4]

C_9 (a) $T_{LS} \subset T_S$; (b) $T_{\{LS\}} \subset T_{\{S\}}$ [from (3) above]

C_{10} (a) $T_{NLS} \subset T_S$; (b) $T_{\{NLS\}} \subset T_{\{S\}}$ [from (4) above]

C_{11} $\{City\colon \text{`}London\text{'}\} \subset T_{City}$

C_{12} $(T_{City} \setminus \{City\colon \text{`}London\text{'}\}) \subset T_{City}$

C_{13} $T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\} \subset T_{Sname} \times T_{Status} \times T_{City}$

C_{14} $T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\}) \subset T_{Sname} \times T_{Status} \times T_{City}$

Constraints C_4–C_7 enforce that LS and NLS are disjoint restrictions of S, while C_8–C_{14} are additional type constraints that enforce specialisation associations implied by the subtypes and supertypes occurring in $\mathcal{S}_0$.

In most typical database schemas, only constraints C_1, C_2, and C_3 will be explicitly declared. Some database schemas may also explicitly declare constraints C_4 and C_7, but it is more likely that these and all the remaining constraints will be implicit in the definition of $\mathcal{S}_0$. It is important for the purposes of this analysis to clearly identify all such implicit constraints, so that nothing is missed when constructing corresponding SIGs.

Let us partition $\mathcal{S}_0$ into three sub-schemas: $\mathcal{S}_1 = \{S\}$, $\mathcal{S}_2 = \{LS, NLS\}$, and $\mathcal{S}_3 = \{LS\}$. We wish to compare the relative information capacities of $(\mathcal{S}_1, \mathcal{S}_2)$ and $(\mathcal{S}_1, \mathcal{S}_3)$, in order to characterise the view update compatibility of the schemas in each pair.

Note that in the following analysis, we omit the foreign key constraints C_2 and C_3, as we have yet to determine the best way to represent inclusion dependencies of this nature in a SIG[3], or indeed whether such dependencies are significant.

6.1.1 Schema $\mathcal{S}_1$

A user granted access only to $\mathcal{S}_1$ can see only relvar S. Constraint C_1(a) can be propagated directly from $\mathcal{S}_0$ into $\mathcal{S}_1$ as it is already expressed in terms of S only. Constraints C_{11}–C_{14} do not include any named relvars in $\mathcal{S}_0$, and thus can also be propagated directly from $\mathcal{S}_0$ into $\mathcal{S}_1$. Constraints C_1(b), C_1(c), and C_4–C_{10} cannot be propagated directly from $\mathcal{S}_0$ into $\mathcal{S}_1$, but can all be rewritten in terms of S only, by substituting the definitions of LS (equation 3) and NLS (equation 4). To simplify expressions in the following discussion, we shall use L to denote $\sigma_{City='London'} S$ (3) and N to denote $\sigma_{City\neq'London'} S$ (4). (This also clearly distinguishes the substitutions from the LS and NLS relvars in $\mathcal{S}_0$, $\mathcal{S}_2$, and $\mathcal{S}_3$.) A constraint that has been rewritten so that it can be propagated from $\mathcal{S}_0$ into $\mathcal{S}_1$ is denoted by, e.g., $C_4^{\mathcal{S}_1}$. Consequently we can derive the following set of rewritten constraints for $\mathcal{S}_1$:

$C_1^{\mathcal{S}_1}$ Sno is a key for each of L and N, therefore the functional dependency $\{Sno\} \rightarrow \{Sname, Status, City\}$ also holds in each of (b) L; and (c) N. [The fact that we are dealing with two disjoint subsets of $City$ does not change this.]

[2] Although as Date notes [10], we could just as easily define LS and NLS as base relvars and then derive $S = LS \cup NLS$, or even define all three as base relvars. The constraints remain the same regardless.

[3] Miller et al. [28] briefly mention inclusion dependencies, but do not discuss how to represent them in a SIG.

Constraint	Implied edge(s)
$C_1^{\mathcal{S}_1}$(b)	$T_{Sno} \overset{key}{\twoheadrightarrow} T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\}$
(c)	$T_{Sno} \overset{key}{\twoheadrightarrow} T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$
$C_9^{\mathcal{S}_1}$	$T_L \mathbin{+\!\!+} T_{\{L\}}$ $\quad T_S \overset{\sigma}{\longleftarrow\!\!\shortmid} T_L$ $\quad T_{\{S\}} \overset{\sigma}{\longleftarrow\!\!\shortmid} T_{\{L\}}$
$C_{10}^{\mathcal{S}_1}$	$T_N \mathbin{+\!\!+} T_{\{N\}}$ $\quad T_S \overset{\sigma}{\longleftarrow\!\!\shortmid} T_N$ $\quad T_{\{S\}} \overset{\sigma}{\longleftarrow\!\!\shortmid} T_{\{N\}}$
C_{11}	$T_{City} \overset{\sigma}{\longleftarrow\!\!\shortmid} \{City\colon \text{`}London\text{'}\}$
C_{12}	$T_{City} \overset{\sigma}{\longleftarrow\!\!\shortmid} (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$
C_{13}	$T_{Sname} \times T_{Status} \times T_{City} \overset{\sigma}{\longleftarrow\!\!\shortmid} T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\}$ $T_{\{L\}} \overset{\pi}{\twoheadrightarrow} T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\}$ $T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\} \overset{\pi}{\twoheadrightarrow} T_{Sname}$ $T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\} \overset{\pi}{\twoheadrightarrow} T_{Status}$ $T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\} \overset{\pi}{\twoheadrightarrow} \{City\colon \text{`}London\text{'}\}$
C_{14}	$T_{Sname} \times T_{Status} \times T_{City} \overset{\sigma}{\longleftarrow\!\!\shortmid} T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$ $T_{\{N\}} \overset{\pi}{\twoheadrightarrow} T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$ $T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\}) \overset{\pi}{\twoheadrightarrow} T_{Sname}$ $T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\}) \overset{\pi}{\twoheadrightarrow} T_{Status}$ $T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\}) \overset{\pi}{\twoheadrightarrow} T_{City} \setminus \{City\colon \text{`}London\text{'}\}$

Table 2: Constraints and the SIG edges they imply for schema $\mathcal{S}_1$.

$\mathbf{C_4^{\mathcal{S}_1}}$ $S = L \cup N$ [or even just $S = S$]

$\mathbf{C_5^{\mathcal{S}_1}}$ $\sigma_{City \neq \text{`}London\text{'}} L = \emptyset$

$\mathbf{C_6^{\mathcal{S}_1}}$ $\sigma_{City = \text{`}London\text{'}} N = \emptyset$

$\mathbf{C_7^{\mathcal{S}_1}}$ $L \cap N = \emptyset$

$\mathbf{C_8^{\mathcal{S}_1}}$ (a) $T_S = T_L \cup T_N$; (b) $T_{\{S\}} = T_{\{L\}} \cup T_{\{N\}}$ [or even just $T_S = T_S$ and $T_{\{S\}} = T_{\{S\}}$]

$\mathbf{C_9^{\mathcal{S}_1}}$ (a) $T_L \subset T_S$; (b) $T_{\{L\}} \subset T_{\{S\}}$ [from (3) above]

$\mathbf{C_{10}^{\mathcal{S}_1}}$ (a) $T_N \subset T_S$; (b) $T_{\{N\}} \subset T_{\{S\}}$ [from (4) above]

6.1.2 Schema $\mathcal{S}_2$

A user granted access only to $\mathcal{S}_2$ can see only relvars *LS* and *NLS*. Constraints C_1(b), C_1(c), and C_5–C_7 can be propagated directly from $\mathcal{S}_0$ into $\mathcal{S}_2$ as they are defined only in terms of *LS* and *NLS*. C_{11}–C_{14} can be propagated directly into $\mathcal{S}_2$ as they are not expressed in terms of any named relvars in $\mathcal{S}_0$. C_1(a), C_4, C_8, C_9, and C_{10} can be rewritten in terms of *LS* and *NLS* only, giving us:

$\mathbf{C_1^{\mathcal{S}_2}}$ *Sno* is a key for *LS*∪*NLS*, therefore the functional dependency $\{Sno\} \rightarrow \{Sname, Status, City\}$ also holds in (a) $LS \cup NLS$.

$\mathbf{C_4^{\mathcal{S}_2}}$ $LS \cup NLS = LS \cup NLS$

$\mathbf{C_8^{\mathcal{S}_2}}$ (a) $T_{LS} \cup T_{NLS} = T_{LS} \cup T_{NLS}$;
(b) $T_{\{LS\}} \cup T_{\{NLS\}} = T_{\{LS\}} \cup T_{\{NLS\}}$

$\mathbf{C_9^{\mathcal{S}_2}}$ (a) $T_{LS} \subset T_{LS} \cup T_{NLS}$; (b) $T_{\{LS\}} \subset T_{\{LS\}} \cup T_{\{NLS\}}$

$\mathbf{C_{10}^{\mathcal{S}_2}}$ (a) $T_{NLS} \subset T_{LS} \cup T_{NLS}$; (b) $T_{\{NLS\}} \subset T_{\{LS\}} \cup T_{\{NLS\}}$

6.1.3 Schema $\mathcal{S}_3$

A user granted access only to $\mathcal{S}_3$ can see only relvar *LS*. Constraints C_1(b) and C_5 can be propagated directly into $\mathcal{S}_3$, as they are defined in terms of *LS* only. C_{11}–C_{14} can also be propagated directly into $\mathcal{S}_3$ as previously discussed. The remaining constraints cannot be rewritten in terms of *LS* only, and so cannot be propagated into $\mathcal{S}_3$. As we were able to propagate all constraints into $\mathcal{S}_1$, this indicates a qualitative difference between $\mathcal{S}_1$ and $\mathcal{S}_3$. It therefore seems likely that the relative information capacities of $\mathcal{S}_1$ and $\mathcal{S}_3$ will also differ.

6.2 Constructing the SIGs

6.2.1 Schema $\mathcal{S}_1$

The SIG for $\mathcal{S}_1$ will not just be that for relvar *S* as shown in Figure 2, as that schema did not incorporate all the constraints implicit in $\mathcal{S}_0$. The process for constructing the SIG will be similar to that for the earlier example, however. The constraints appearing in $\mathcal{S}_1$ are C_1(a), $C_1^{\mathcal{S}_1}$(b), $C_1^{\mathcal{S}_1}$(c), $C_4^{\mathcal{S}_1}$–$C_{10}^{\mathcal{S}_1}$, and C_{11}–C_{14}.

The SIG will include all of the edges that appear in Figure 2. The constraints within the schema also imply the existence of several subtype nodes (e.g., the two subtypes of T_{City} implied by C_{11} and C_{12}), along with corresponding selection and projection edges, which are listed in Table 2 above. $C_4^{\mathcal{S}_1}$, $C_8^{\mathcal{S}_1}$(a), and $C_8^{\mathcal{S}_1}$(b) are effectively tautologies that add no new information, and can thus be safely ignored. The completed SIG for $\mathcal{S}_1$ is shown in Figure 3.

6.2.2 Schema $\mathcal{S}_2$

The constraints appearing in $\mathcal{S}_2$ are $C_1^{\mathcal{S}_2}$(a), C_1(b), C_1(c), $C_4^{\mathcal{S}_2}$, C_5–C_7, $C_8^{\mathcal{S}_2}$–$C_{10}^{\mathcal{S}_2}$, and C_{11}–C_{14}. This is essentially the same set of constraints as $\mathcal{S}_1$, but with different rewritings. This means that the SIG for $\mathcal{S}_2$ can be built in the same manner as for $\mathcal{S}_1$, with some nodes named differently. In particular, $C_8^{\mathcal{S}_2}$ implies that there will be nodes $T_{LS} + T_{NLS}$ and $T_{\{LS\}} + T_{\{NLS\}}$ instead of T_S and $T_{\{S\}}$, respectively. There will also be T_{LS} instead of T_L, etc. The complete SIG for $\mathcal{S}_2$ is shown in Figure 4.

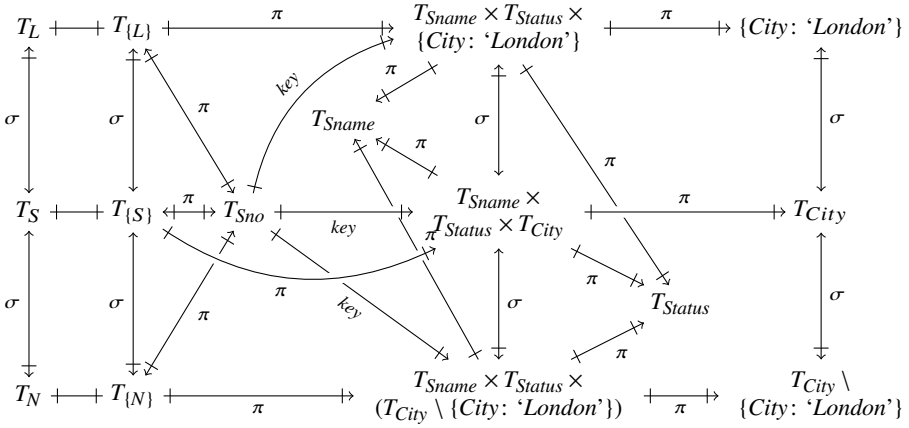


Figure 3: SIG for schema $\mathcal{S}_1 = \{S\}$.

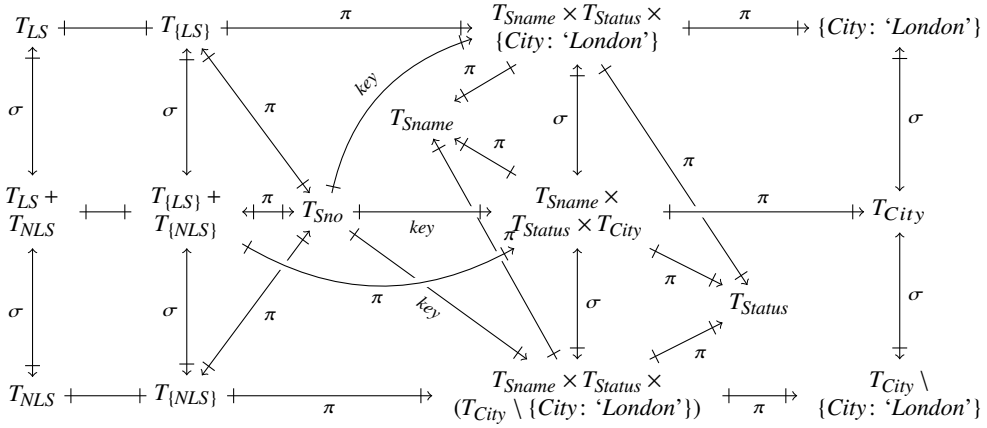


Figure 4: SIG for schema $\mathcal{S}_2 = \{LS, NLS\}$.

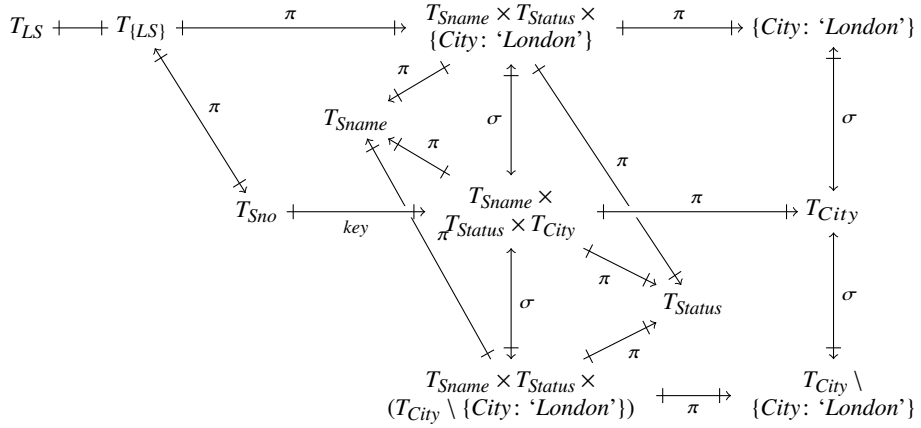


Figure 5: SIG for schema $\mathcal{S}_3 = \{LS\}$.

6.2.3 Schema $\mathcal{S}_3$

The constraints appearing in $\mathcal{S}_3$ are C_1(b), C_5, and C_{11}–C_{14}. The SIG will only include edges and nodes that relate either directly to *LS* or are independent of any relvar, such as the subtype node $T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$. The complete SIG for $\mathcal{S}_3$ is shown in Figure 5 on the previous page.

6.3 Transforming the SIGs

Now that we have constructed SIGs for all three schemas, we can compare the relative information capacities of the schema pairs $(\mathcal{S}_1, \mathcal{S}_2)$ and $(\mathcal{S}_1, \mathcal{S}_3)$. We do this by attempting to transform the SIG for one of the schemas until it is isomorphic with the other.

Looking at the SIGs for $\mathcal{S}_1$ and $\mathcal{S}_2$, we can see almost immediately that both SIGs have the same node and edge structure; indeed the only difference is in the names of some of the nodes. However, we already know from the discussion in Sections 6.1.1 and 6.1.2 that these differently named nodes are just rewritings that represent the same domains. We therefore have an immediate isomorphism between the two SIGs and can conclude that the information capacities of $\mathcal{S}_1$ and $\mathcal{S}_2$ are equivalent, i.e., $\mathcal{S}_1 \equiv \mathcal{S}_2$.

The comparison between $\mathcal{S}_1$ and $\mathcal{S}_3$ is more interesting. There are some shared structures, but the two SIGs are clearly different. Since the SIG transformations discussed in Section 4.2 only ever preserve or enhance information capacity, intuitively we should attempt to transform the "smaller" SIG for $\mathcal{S}_3$ into the "larger" SIG for $\mathcal{S}_1$. SIG transformations can generally be carried out in two broad phases (using the $\mathcal{S}_3 \to \mathcal{S}_1$ transformation as an example):

Duplicate nodes in $\mathcal{S}_3$ (producing $\mathcal{S}_3'$) that correspond to supertypes in $\mathcal{S}_1$, in order to create nodes corresponding to the extra nodes that $\mathcal{S}_1$ has over $\mathcal{S}_3$. The goal is to produce a node structure similar to $\mathcal{S}_1$. The duplicated nodes are connected to their original nodes by trivial selection edges. These transformations (if applicable) do not alter the information capacity of $\mathcal{S}_3'$ (i.e., $\mathcal{S}_3 \equiv \mathcal{S}_3'$).

Copy and move edges around the SIG to produce an edge structure similar to $\mathcal{S}_1$. The annotations of each moved or copied edge are determined by the path it is composed with, as discussed in Section 4.2. These transformations (if applicable) *increase* the information capacity of $\mathcal{S}_3'$ (i.e., $\mathcal{S}_3 \preceq \mathcal{S}_3'$).

These two phases generally occur in order, but may be interspersed throughout with the following:

Delete duplicate nodes in $\mathcal{S}_3'$ that are no longer needed, especially those that do not exist in $\mathcal{S}_1$. These transformations (if applicable) do not alter the information capacity of $\mathcal{S}_3'$ ($\equiv$).

Remove totality annotations from trivial selection edges in $\mathcal{S}_3'$ to change them into non-trivial selection edges. This decouples duplicate nodes into subtypes and supertypes, ensuring that they represent different domains. These transformations (if applicable) *increase* the information capacity of $\mathcal{S}_3'$ ($\preceq$).

For example, we might duplicate some nodes, remove some totality annotations, move or copy some edges, delete some no longer needed nodes, move or copy some more edges, then remove some more totality annotations.

Applying this process to $\mathcal{S}_3$, we first duplicate node T_{LS} to T_S and node $T_{\{LS\}}$ to $T_{\{S\}}$. To obtain a node structure similar to that of $\mathcal{S}_1$, we also need to further duplicate T_S to T_{NLS} and $T_{\{S\}}$ to $T_{\{NLS\}}$. The result of these duplications is shown in Figure 6. The transformed SIG now has the same node structure as that for $\mathcal{S}_1$.

The next phase is to copy and move edges. To produce the edge $T_S \mathbin{+\!\!+} T_{\{S\}}$, we can first *copy* $T_{LS} \mathbin{+\!\!+} T_{\{LS\}}$ across the path $T_{\{S\}} \overset{\sigma}{\leftarrow\!\!\!\!\rightarrow} T_{\{LS\}}$ (giving $T_{LS} \mathbin{+\!\!+} T_{\{S\}}$), then *move* it across $T_S \overset{\sigma}{\leftarrow\!\!\!\!\rightarrow} T_{LS}$. We can carry out a similar series of transformations to copy and move $T_S \mathbin{+\!\!+} T_{\{S\}}$ to $T_{NLS} \mathbin{+\!\!+} T_{\{NLS\}}$ (see Figure 7).

We next copy and move the unique projection edge $T_{\{LS\}} \overset{\pi}{\leftarrow\!\!\!\!\rightarrow} T_{Sno}$ to produce the edges $T_{\{S\}} \overset{\pi}{\leftarrow\!\!\!\!\rightarrow} T_{Sno}$ and $T_{\{NLS\}} \overset{\pi}{\leftarrow\!\!\!\!\rightarrow} T_{Sno}$. Similarly, we copy and move $T_{\{LS\}} \overset{\pi}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\}$ to produce $T_{\{S\}} \overset{\pi}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times T_{City}$ and $T_{\{NLS\}} \xrightarrow{\pi} T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$. Note that the first edge in the latter pair of transformations loses its totality annotation, and the second edge its surjectivity annotation, as a result of composition with the pre-existing non-bijective selection edges between the constructed tuple types running vertically down the centre of the SIG.

Finally, we carry out similar copy and move transformations to copy the key edge $T_{Sno} \overset{key}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times T_{City}$ to $T_{Sno} \overset{key}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\}$, and $T_{Sno} \overset{key}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$. Both new edges lose their totality annotations for the same reason as the projection edges above. The final result of all these edge transformations is shown in Figure 7.

If we compare Figure 3 with Figure 7, we can see that the transformed SIG for $\mathcal{S}_3'$ now has the same topology as the SIG for $\mathcal{S}_1$. We still need to remove the totality annotations from the newly created trivial selection edges between the relation and tuple types on the left, but even when this is done, we do not achieve the desired isomorphism, as the edges $T_{\{S\}} \overset{\pi}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times T_{City}, T_{\{NLS\}} \xrightarrow{\pi} T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$, $T_{Sno} \overset{key}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times \{City\colon \text{`}London\text{'}\}$, and $T_{Sno} \overset{key}{\rightarrow\!\!\!\!\rightarrow} T_{Sname} \times T_{Status} \times (T_{City} \setminus \{City\colon \text{`}London\text{'}\})$ all have different annotations from the corresponding edges in the SIG for $\mathcal{S}_1$. This clearly shows that the information capacities of $\mathcal{S}_1$ and $\mathcal{S}_3$ are incomparable, and that view updates based on this pair of schemas will be problematic.

If we were to create a fourth sub-schema $\mathcal{S}_4 = \{NLS\}$ and compare this with $\mathcal{S}_1$, the result would be similar to that for $\mathcal{S}_3$.

7. CONCLUSION & FUTURE WORK

In this paper we have shown how to use the concept of relative information capacity and an extension of the schema intension graph (SIG) formalism to more formally model Date's notion of information equivalence with respect to relational view updates. Our analysis of Date's restriction view example [10] using these techniques is consistent with his findings, so the approach shows promise. The information capacities of $\mathcal{S}_1 = \{S\}$ and $\mathcal{S}_2 = \{LS, NLS\}$ are equivalent, so it should be possible to effectively propagate view updates between $\mathcal{S}_1$ and $\mathcal{S}_2$, regardless of the configuration of views and base relvars. Conversely, the information capacity of $\mathcal{S}_3 = \{LS\}$ is not comparable with that of $\mathcal{S}_1$, implying that there are updates that cannot be propagated between $\mathcal{S}_1$ and $\mathcal{S}_3$.

Our analysis reinforces the importance of explicitly identifying all constraints that can be propagated from the base schema $\mathcal{S}_0$ to its sub-schemas. Omission of constraints could produce a different SIG structure that might lead to an erroneous conclusion about a schema's information capacity relative to another.

There are several avenues for future work. First, the proof of concept discussed in this paper only deals with disjoint restriction views, which are arguably the simplest case of view updating. Further work needs to be done to characterise other types of view configuration, such as projection views, union views, and in particular, join views. This will enable us to test and refine the application of SIGs in this context.

The discussion in this paper focused only on the relational context, but relative information capacity and the SIG formalism are data

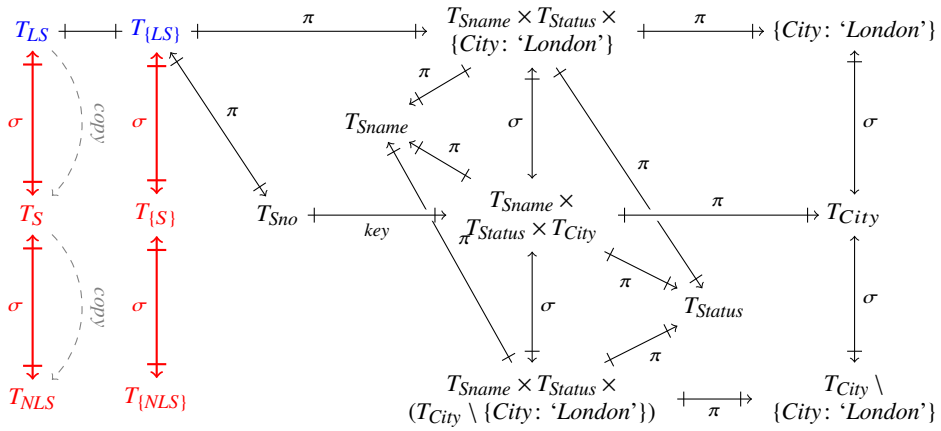


Figure 6: SIG for transformed schema S_3' after duplicating T_{LS} and $T_{\{LS\}}$ to produce analogues of the additional nodes in S_1. Inputs to the transformations are coloured blue, outputs are coloured red. [≡]

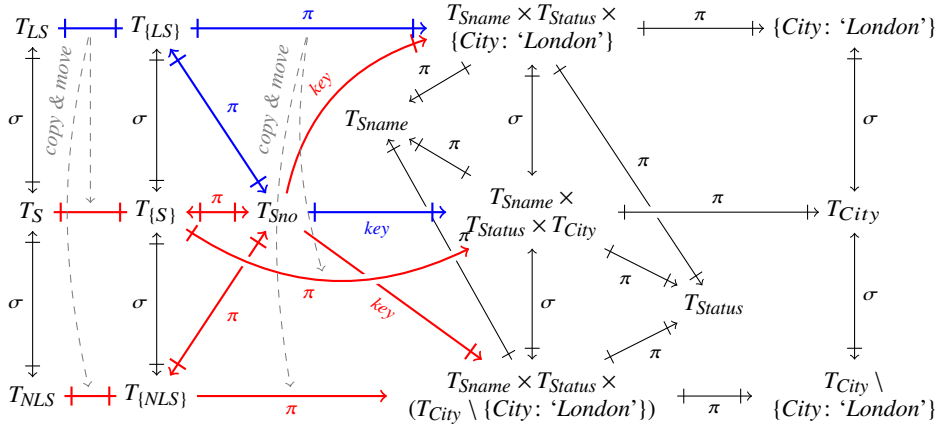


Figure 7: SIG for transformed schema S_3' after copying and moving various edges. [≤]

model agnostic. It would therefore be interesting to investigate the application of these techniques to view updating contexts beyond the Relational Model (e.g., deductive databases).

It is unclear at present whether the omission of inclusion dependencies such as foreign keys is significant. If they are significant, the question then is how best to represent them in the SIG formalism. Adding such constraints will clearly change the structure of a SIG, but we suspect that this will not change the relationship between the relative information capacities of different schemas.

It is interesting to note that with the equivalent schemas $\mathcal{S}_1$ and $\mathcal{S}_2$, we were able to propagate all of the constraints from the base schema $\mathcal{S}_0$ (Section 6.1.2), whereas with the incomparable schemas $\mathcal{S}_1$ and $\mathcal{S}_3$, only a subset of the constraints could be propagated (Section 6.1.3). A reasonable conjecture is that the degree to which constraints can be propagated from $\mathcal{S}_0$ to its sub-schemas could be a direct indicator of information equivalence among the sub-schemas, but this needs to be further investigated.

Finally, we have recently discovered further work that explores and expands upon the concepts of information capacity and meaning within data and database schemas (e.g., [22, 34, 35]). Some of these ideas may be applicable to our work, and need to be investigated in more detail.

8. REFERENCES

[1] F. Bancilhon and N. Spyratos. Update semantics of relational views. *ACM Transactions on Database Systems*, 6(4):557–575, Dec. 1981.

[2] A. Behrend and R. Manthey. A transformation-based approach to view updating in stratifiable deductive databases. In S. Hartmann and G. Kern-Isberner, editors, *Proceedings of the 5th International Symposium on Foundations of Information and Knowledge Systems (FoIKS 2008)*, volume 4932 of *Lecture Notes in Computer Science*, pages 253–271, Pisa, Italy, Feb. 11–15 2008. Springer.

[3] L. Caroprese, I. Trubitsyna, M. Truszczyński, and E. Zumpano. The view-update problem for indefinite databases. In L. F. del Cerro, A. Herzig, and J. Mengin, editors, *Proceedings of the Thirteenth European Conference on Logics in Artificial Intelligence (JELIA 2012)*, volume 7519 of *Lecture Notes in Computer Science*, pages 134–146, Toulouse, France, Sept. 26–28 2012.

[4] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[5] E. F. Codd. Relational completeness of data base

sublanguages. In R. Rustin, editor, *Data Base Systems*, Courant Computer Science Symposia Series 6, pages 65–98. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1972.

[6] E. F. Codd. Recent investigations in relational data base systems. In J. L. Rosenfeld, editor, *Proceedings of the IFIP Congress '74 (Information Processing '74)*, pages 1017–1021, Stockholm, Sweden, Aug. 5–10 1974. North-Holland.

[7] L. Console, M. L. Sapino, and D. T. Dupré. The role of abduction in database view updating. *Journal of Intelligent Information Systems*, 4(3):261–280, May 1995.

[8] S. S. Cosmadakis and C. H. Papadimitriou. Updates of relational views. *Journal of the ACM*, 31(4):742–760, Sept. 1984.

[9] C. J. Date. *SQL and Relational Theory: How to Write Accurate SQL Code*. Theory in Practice. O'Reilly, second edition, 2012.

[10] C. J. Date. *View Updating & Relational Theory: Solving the View Update Problem*. Theory in Practice. O'Reilly, 2013.

[11] G. Gottlob, P. Paolini, and R. Zicari. Properties and update semantics of consistent views. *ACM Transactions on Database Systems*, 13(4):486–524, Dec. 1988.

[12] S. J. Hegner. An order-based theory of updates for closed database views. *Annals of Mathematics and Artificial Intelligence*, 40(1–2):63–125, Jan. 2004.

[13] R. Hull. Relative information capacity of simple relational database schemata. *SIAM Journal on Computing*, 15(3):856–886, Aug. 1986.

[14] A. M. Keller. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In *Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS '85)*, pages 154–163, Portland, Oregon, USA, Mar. 25–27 1985. ACM.

[15] A. M. Keller. The role of semantics in translating view updates. *IEEE Computer*, 19(1):63–73, Jan. 1986.

[16] A. M. Keller. Comments on Bancilhon and Spyratos' "Update semantics and relational views". *ACM Transactions on Database Systems*, 12(3):521–523, Sept. 1987.

[17] Y. Kotidis, D. Srivastava, and Y. Velegrakis. Updates through views: A new hope. In L. Liu, A. Reuter, K.-Y. Whang, and J. Zhang, editors, *Proceedings of the 22nd International Conference on Data Engineering (ICDE 2006)*, page 2, Atlanta, Georgia, USA, 3–8 Apr. 2006. IEEE Computer Society.

[18] D. Laurent, J. Lechtenbörger, N. Spyratos, and G. Vossen. Monotonic complements for independent data warehouses. *The VLDB Journal*, 10(4):295–315, Dec. 2001.

[19] J. Lechtenbörger. The impact of the constant complement approach towards view updating. In F. Neven, C. Beeri, and T. Milo, editors, *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2003)*, pages 49–55, San Diego, California, USA, June 9–12 2003. ACM.

[20] J. Lechtenbörger and G. Vossen. On the computation of relational view complements. *ACM Transactions on Database Systems*, 28(2):175–208, June 2003.

[21] H. Liefke and S. B. Davidson. View maintenance for hierarchical semistructured data. In Y. Kambayashi, M. K. Mohania, and A. M. Tjoa, editors, *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2000)*, volume 1874 of *Lecture Notes in Computer Science*, pages 114–125, London, UK, Sept. 4–6 2000. Springer.

[22] R. Mantri and J. Feng. Data semantic conflicts: An approach based on the notion of 'information bearing capability'. *Nirma University Journal of Engineering and Technology*, 1(2):26–30, July–Dec. 2010.

[23] Y. Masunaga. A relational database view update translation mechanism. In U. Dayal, G. Schlageter, and L. H. Seng, editors, *Proceedings of the Tenth International Conference on Very Large Data Bases (VLDB 1984)*, pages 309–320, Singapore, Aug. 27–31 1984. Morgan Kaufmann.

[24] E. Mayol and E. Teniente. Consistency preserving updates in deductive databases. *Data & Knowledge Engineering*, 47(1):61–103, Oct. 2003.

[25] R. J. Miller. *Managing Structural Heterogeneity*. PhD thesis, University of Wisconsin-Madison, Madison, Wisconsin, USA, 1994.

[26] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. The use of information capacity in schema integration and translation. In R. Agrawal, S. Baker, and D. A. Bell, editors, *Proceedings of the Nineteenth International Conference on Very Large Data Bases (VLDB)*, pages 120–133, Dublin, Ireland, Aug. 24–27 1993. Morgan Kaufmann.

[27] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. Schema equivalence in heterogeneous systems: Bridging theory and practice. *Information Systems*, 19(1):3–31, Jan. 1994.

[28] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. Schema intension graphs: A formal model for the study of schema equivalence. Technical report CS-TR-94-1185, Department of Computer Sciences, University of Wisconsin-Madison, Madison, Wisconsin, USA, Jan. 1994.

[29] W.-W. Pan and W.-P. Yang. An object model at conceptual level to support updatable views on object-oriented databases. *Information Sciences*, 95(1–2):29–48, Nov. 1996.

[30] X. Qian. Correct schema transformations. Technical report SRI-CSL-95-08, Computer Science Laboratory, SRI International, Menlo Park, California, USA, July 1995.

[31] M. H. Scholl, C. Laasch, and M. Tresch. Updatable views in object-oriented databases. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *Proceedings of the 2nd International Conference on Deductive and Object-Oriented Databases (DOOD'91)*, volume 566 of *Lecture Notes in Computer Science*, pages 189–207, Munich, Germany, Dec. 16–18 1991. Springer.

[32] H. Shu. Using constraint satisfaction for view update. *Journal of Intelligent Information Systems*, 15(2):147–173, Sept. 2000.

[33] L. Wang, E. A. Rundensteiner, and M. Mani. Updating XML views published over relational databases: Towards the existence of a correct update mapping. *Data & Knowledge Engineering*, 58(3):236–298, Sept. 2006.

[34] X. Wang, J. Feng, and H. Xu. A study of the schema intension graphs model (SIG) approach to the relative information capacity (RIC) of data schemas. *Computing and Information Systems*, 9(3):article 9, Oct. 2005.

[35] K. Xu, J. Feng, and M. Crowe. Defining the notion of 'information content' and reasoning about it in a database. *Knowledge and Information Systems*, 18(1):29–59, Jan. 2009.