MONDRIAN: Annotating And Querying Databases Through Colors And Blocks

DOI: 10.1109/ICDE.2006.102
Handle: http://hdl.handle.net/1942/1421

MONDRIAN: Annotating and querying databases through colors and blocks

Floris Geerts
Hasselt University &
University of Edinburgh
fgeerts@inf.ed.ac.uk

Anastasios Kementsietsidis
School of Informatics
University of Edinburgh
akements@inf.ed.ac.uk

Diego Milano
Dipartimento di Informatica
e Sistemistica
Università di Roma "La Sapienza"
diego.milano@dis.uniroma1.it

Abstract

Annotations play a central role in the curation of scientific databases. Despite their importance, data formats and schemas are not designed to manage the increasing variety of annotations. Moreover, DBMS's often lack support for storing and querying annotations. Furthermore, annotations and data are only loosely coupled. This paper introduces an annotation-oriented data model for the manipulation and querying of both data and annotations. In particular, the model allows for the specification of annotations on sets of values and for effectively querying the information on their association. We use the concept of block to represent an annotated set of values. Different colors applied to the blocks represent different annotations. We introduce a color query language for our model and prove it to be both complete (it can express all possible queries over the class of annotated databases), and minimal (all the algebra operators are primitive). We present MONDRIAN, a prototype implementation of our annotation mechanism, and we conduct experiments that investigate the set of parameters which influence the evaluation cost for color queries.

1. Introduction

From biology to astronomy, scientific databases play a central role in the advancement of science by providing access to large collections of data. At the same time, these databases are of particular interest to computer scientists due to the data management challenges that they pose [9]. Apart from the often staggering amounts of data, two additional characteristics make the management of scientific databases challenging. First, scientific data come in a variety of formats which range from flat-formatted files to images and electronic publications. Thus, the challenge here is to *integrate* [15], *annotate* [7] and *cross-reference* [20] such diverse collections of data. Second, scientists often analyze data that are collected from a variety of sources and, in turn, this analysis results in new data which are used by other scientists, resulting a continuous feedback of data. In such a setting, it is important to maintain *data provenance* [10, 22], i.e., where the data that a scientist is using come from.

In this paper, we offer a new model to annotate databases and a new language to query annotated databases. Our work is motivated by the pressing needs of biological databases and our examples are drawn from this domain. Figures 1(a), (b) and (c) show three sample relations taken from GDB [2] (a human gene database), Swissprot [3] (a protein database), and PIR [1] (a protein sequence database). Each relation stores, correspondingly, pairs of identifiers and names of genes, proteins and protein sequences.

Two key points distinguish our work from other mechanisms proposed for annotation management:

- First, we argue that for an annotation mechanism to be useful in practice, it should support the annotation of *sets of values*. In the literature, existing mechanisms assume that each annotation is attached to a particular value of a specific attribute (e.g., see [6]). So, one can annotate, for example, the gene name *NF1*, in Figure 1(a), with the name of the researcher that discovered this gene. A possible implementation of such a mechanism requires adding a column, say *annot_gname*, in the GDB relation and use it to store the annotations of the *gname* column values.

However, in a number of domains, including scientific databases, it becomes increasingly important to annotate sets of values. As an example, consider the relation shown in Figure 1(d). The relation associates GDB gene identifiers to SwissProt protein and PIR sequence identifiers. The semantics of this association is that the specified gene is related to the indicated protein (and protein sequence). Such relations are widely used in the biological domain and offer a quick way to cross-reference and establish associations between independent biological sources [20, 22]. In this setting, it is useful to record, for each stored association, what evidence exists for its validity [13]. In the figure, we show possible annotations of the relation in the form of blocks and block labels. In more detail, a block is used to indicate the set of values for which an annotation exists, while

Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)
8-7695-2570-9/06 $20.00 © 2006 IEEE

gid	gname
120231	NF1
120232	NF2
120233	NGFB
120234	NGFR
120235	NHS

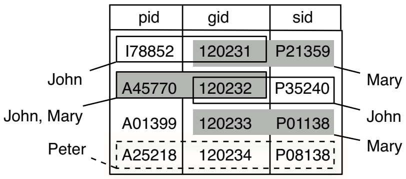
(a) GDB relation

sid	sname
P01138	Beta-NGF
P08138	TNR16
P14543	Nidogen
P21359	Neurofibromin
P35240	Merlin

(b) Swissprot relation

pid	pname
A01399	Nerve growth factor
A25218	Tumor necrosis factor
A45770	Merlin
I78852	Neurofibromatosis
Q6T45	Nancy-Horan syndrome

(c) PIR relation



(d) An integrated relation

Figure 1. Three biological sources and their integrated relation

block labels are used to indicate the annotations for this block. In our example, the annotations indicate the names of the curators who verified that a particular association holds. So, in the first tuple, a block indicates Mary's belief that the gene with gid *120231* produces protein with sid *P21359*. Although conceptually easy to illustrate, complex annotations like those shown here pose interesting challenges in terms of how they can be *implemented*. In this paper, we propose an implementation that has two desirable properties: first, it does not require any restructuring of the existing schema of the database to be annotated (only extra tables need to be added); and second, it is such that the annotation of the database imposes minimum overhead both in terms of space, and in terms of query execution time when compared to its unannotated version.

• Second, we believe that annotations should be treated as first-class citizens of the database, that is, we should be able to query values and annotations alike (in isolation or in unison). Currently, query languages cannot *see* the annotations and they can only transmit them [11]. However, for curators, annotations are of equal or even greater importance than values. A curator using the relation in Figure 1(d) might want to find which tuples are annotated by either John or Mary. This is a value-based query, but it refers to an annotation value (rather than a data value) which, as the figure shows, spans attribute boundaries, and it can appear over distinct attribute sets, in different tuples. As another example, the curator might be interested in finding which gene-protein sequence (*gid, sid*) pairs are annotated, and by whom. This is not a value-based annotation query. Rather, it refers to the attributes on which annotations are applied on each tuple. Finally, we note that sometimes the lack of annotations might also be of interest to a curator. In a heavily curated database a curator might want to find which gene-protein (*gid, pid*) pairs are not annotated. All these operations assume that we have a query language capable of expressing queries over annotated databases.

Desirable properties of such a query language are that it is at a level of abstraction that is independent of the chosen representation of annotations and that it is user friendly. Any relational representation of annotated databases that we can think of offers the relational algebra or, on a practical level, SQL as candidate query languages. However, each query posed in these languages is not annotation-representation independent. A change in the representation of annotations requires us to reformulate all our queries. Furthermore, while writing such queries, we should make sure that the result of each query must be interpretable as an annotated database again. It is clear that one needs to pose severe syntactic and semantic conditions on such queries in order to achieve this goal. It is not clear what these conditions should be.

We opt for a different approach and introduce a new query language hereafter referred to as the *color algebra* (since we use colors to represent annotations). By definition, our algebra is annotation-representation independent. Furthermore, any query in this language produces a color (annotated) relation on every input color relation. Moreover, the semantics of colors and blocks is transparent in each algebra operator. Our algebra facilitates the querying of color databases and can easily express queries such as the ones described in the previous paragraphs.

The contributions of this paper are as follows:

• We introduce the first annotation mechanism for relational databases that is capable of annotating both single values and the associations between multiple values.

• We introduce an algebra to query values and annotations alike. Our algebra includes well-known operators, like selection and projection, properly re-defined to account for colors and blocks, along with new operators that are particular to the querying of annotations. We formalize the notion of annotation in relational databases and we prove that our algebra is both *complete* (it expresses all possible queries over annotated databases) and *minimal* (every operator is primitive, and thus necessary).

• We present MONDRIAN[1] which is an implementation of our annotation mechanism over a relational DBMS. We investigate the space overhead of our representation, and we study the cost of evaluating queries over annotated databases and the parameters that influence this cost.

The remainder of this paper is organized as follows. First, we review related work. Section 2 introduces the ba-

[1] Piet Mondrian: Dutch painter whose paintings mainly consist of color blocks.

sic notations and presents the color algebra while Section 3 describes the relational representation and presents the completeness result. Section 4 introduces MONDRIAN and offers a description of our implementation and experiments. Section 5 concludes with a summary of the results and a discussion on future work.

1.1. Related work

Most existing annotation systems focus on text and HTML documents (e.g., Annotea [18]) and are often specialized to support annotations for a particular kind of data, e.g., genomic sequences [19, 7]. Research on these systems has been focused on scalability, distributive support of annotations, and other features.

Bhagwat et al. [6] propose an annotation mechanism for relational databases where annotations are stored in extra annotation attributes. The authors extend the Select-Project-Join-Union fragment of SQL with a PROPAGATE clause which allows the user to specify how annotations should propagate. The focus is thus on the propagation of the annotations through queries, and the issue of how to query the annotations themselves is not addressed. Also, only single values are annotated. The DBNotes system [12] extends this framework and offers limited support of querying annotations over single values.

An extensive literature exists regarding the computation of provenance. Annotations provide a solid way of keeping track of provenance. Indeed, computing provenance by forwarding annotations along data transformations has been proposed in various forms [5, 18, 21, 6]. The data provenance problem without the use of annotation is studied by Cui et al. [14], Buneman et al. [10, 11], and Widom [23]. In this work, a "reverse" query is generated to compute data provenance. In this paper, we will not address the issue of provenance. Instead, we provide a foundation on which both provenance information and other forms of annotations can be managed.

An unrelated work (although the title suggests otherwise) regards "Colorful XML" [17]. A new XML data model, called multi-colored trees, is proposed and colors are used to add semantic structure over the XML data nodes.

2. Colors and Blocks

2.1. Basic notation

As already mentioned, our aim is to provide a mechanism for annotating sets of attribute values. We refer to such a set of attribute values as a *block*. As an example, in Figure 1(d), there are six different blocks, and each block has an associated annotation. In the remainder of the paper, for ease of presentation and notational convenience, we assume that each annotation is represented by a *color*. Therefore, instead of talking about annotations and annotated blocks we talk about *colors* and *color blocks*, respectively. Similarly, we talk about *color databases* (databases that are annotated) and *color queries* (queries on annotated databases) that are written using a *color algebra* (an algebra that accounts for annotations).

Let **D** be a standard relational database consisting of the relations $R_1, \ldots, R_k$. For each relation R_i, we denote its set of attributes by $\mathit{sort}(R_i)$, while we use r_i to denote an instance of the relation. We use upper-case letters early in the alphabet $(A, B, \ldots)$ to denote attribute names while upper-case letters late in the alphabet $(X, Y, \ldots)$ are used to denote sets of attributes. Accordingly, lower-case letters early in the alphabet $(a, b, \ldots)$ are used to denote attribute values, while those late in the alphabet $(x, y, \ldots)$ are used to denote sets of attributes values. Finally, $\mathcal{C}$ denotes a set of colors.

Let r be an instance of relation R and let t be a tuple in r. The annotation, or *coloring*, of a tuple t is performed through a coloring function χ. Function χ accepts as input a tuple t and a non-empty set of attributes $Y \subseteq \mathit{sort}(R)$ and assigns a set of colors to the values in $t[Y]$. For a tuple t, the triplet $(t, Y, \chi(t, Y))$ defines a *color block* which consists of the attribute values in $t[Y]$ along with their assigned colors. If $\chi(t, Y) = \emptyset$, then the values in $t[Y]$ are not within a color block. Hereafter, we use $\langle r, \chi \rangle$ to denote a relation r whose tuples are colored through function χ.

Example 1. *Consider the relation in Figure 1(d). Then, the coloring of each tuple in the relation is expressed through the following coloring function χ (where, t_i is the ith tuple in the relation):*

$$\begin{array}{ll}
\chi(t_1, \{\mathit{pid}, \mathit{gid}\}) = \{\mathit{John}\} & \chi(t_1, \{\mathit{gid}, \mathit{sid}\}) = \{\mathit{Mary}\} \\
\chi(t_2, \{\mathit{pid}, \mathit{gid}\}) = \{\mathit{John,\ Mary}\} & \chi(t_2, \{\mathit{gid}, \mathit{sid}\}) = \{\mathit{John}\} \\
 & \chi(t_3, \{\mathit{gid}, \mathit{sid}\}) = \{\mathit{Mary}\} \\
\chi(t_4, \{\mathit{pid}, \mathit{gid}, \mathit{sid}\}) = \{\mathit{Peter}\} &
\end{array}$$

Moreover, for every tuple t_i and all other sets of attributes Y, $\chi(t_i, Y)$ is (implicitly) defined to be the empty set.

2.2. Color algebra (CA)

In what follows, we introduce the main set of operators of the color algebra (CA) and we present a number of examples to illustrate their use. The full set of operators (available in [16]) is omitted here due to lack of space.

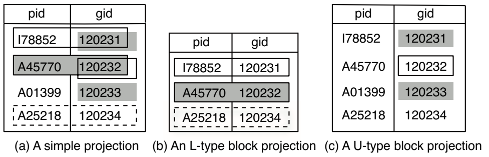
***Projection*:** We define the projection $\pi_{A_1 \cdots A_k}$ as the operator which takes as input any instance $\langle r, \chi \rangle$ of sort containing $\{A_1, \ldots, A_k\}$ and returns the instance $\langle r', \chi' \rangle$ of sort $\{A_1, \ldots, A_k\}$ such that

$$r' = \{t[A_1, \ldots, A_k] \mid t \in r\} \text{ (normal projection)}$$

and for any $t \in r$, and any $Y \subseteq \{A_1, \ldots, A_k\}$,

$$\chi'(t[A_1, \ldots, A_k], Y) = \bigcup_Z \chi(t, Y \cup Z),$$

where Z ranges over all subsets of $\mathit{sort}(R) \backslash \{A_1, \ldots, A_k\}$.



pid	gid
I78852	120231
A45770	120232
A01399	120233
A25218	120234

(a) A simple projection

pid	gid
I78852	120231
A45770	120232
A25218	120234

(b) An L-type block projection

pid	gid
I78852	120231
A45770	120232
A01399	120233
A25218	120234

(c) A U-type block projection

Figure 2. The projection operators

Example 2. *Consider the relation $\langle r, \chi \rangle$ shown in Figure 1(d). Then, expression $\pi_{pid,gid}(r)$ returns the relation r' shown in Figure 2(a). Note that the tuples t_1, t_2 and t_3 get those (projected) blocks of r involving only the $\{gid\}$ attribute, while tuple t_4 gets a (projected) block of r involving the $\{pid,gid\}$ attributes.* □

The projection operator removes parts of a relational instance based on schema-level information. Here, we introduce a corresponding operator that uses schema-level information to remove blocks.

***Block Projection*:** We offer two types of block projection that allow for the projection of blocks based on whether blocks contain or are contained in a specified set of attributes. Specifically, the L-type (Lower) block projection operator $\Pi^L_{A_1,\ldots,A_k}$ takes as input an instance $\langle r, \chi \rangle$ of sort containing $\{A_1, \ldots, A_k\}$, and returns the instance $\langle r', \chi' \rangle$ of the same sort defined by

$$r' = \{t \mid t \in r \text{ and there exists a block}(t, Y, \chi(t, Y)) \text{ with } A_1, \ldots, A_k \subseteq Y\}$$

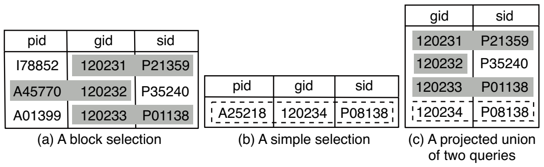
and for any $t \in r'$, and any set of attributes $Y \subseteq sort(R')$,

$$\chi'(t, Y) = \begin{cases} \chi(t, Y) & \text{if } \{A_1, \ldots, A_k\} \subseteq Y, \chi(t, Y) \neq \emptyset; \\ \emptyset & \text{otherwise.} \end{cases}$$

The U-type (Upper) projection operator $\Pi^U_{A_1,\ldots,A_\ell}$ is defined similarly, except that $r' = r$ and in the definition of $\chi'(t, Y)$, $Y \subseteq \{A_1, \ldots, A_k\}$ must hold. We also define $\Pi^L_\emptyset = \text{Id}$, while $\Pi^U_\emptyset$ only returns the unannotated tuples.

Example 3. *Consider the relation $\langle r, \chi \rangle$ in Figure 2(a). Assume that we want to find all the tuples with at least one annotation that involves the protein identifier (pid) attribute. Expression $\Pi^L_{pid}(r)$ returns the desired result, shown in Figure 2(b). Operator Π^L sets a lower bound on the set of attributes that must participate in a selected block. Thus, in our example, both tuples that do not have an annotation involving the pid attribute, and blocks that do not annotate the attribute, are removed.*

On the other hand, we may want all the tuples of relation r that might have an annotation only on the gid attribute. Then, the expression $\Pi^U_{gid}(r)$ finds all such tuples. The resulting relation is shown in Figure 2(c). Operator Π^U sets an upper bound on the set of attributes that may participate in a selected block. Blocks only involving a subset of this upper bound are also selected. In our example, any block that involves an attribute other that gid is removed.



pid	gid	sid
I78852	120231	P21359
A45770	120232	P35240
A01399	120233	P01138

(a) A block selection

pid	gid	sid
A25218	120234	P08138

(b) A simple selection

gid	sid
120231	P21359
120232	P35240
120233	P01138
120234	P08138

(c) A projected union of two queries

Figure 3. The selection and union operators

We note that an unannotated tuple always satisfies the condition of the Π^U operator while it always violates the condition of the Π^L operator. Thus, it is always preserved by the former and dropped by the latter.

By combining Π^L and Π^U, we can find all tuples that have a block on a specific attribute set (and only this set). Expression $\Pi^L_{gid}(\Pi^U_{gid}(r))$ returns all tuples with a block on gid alone. These are the first three tuples in Figure 2(c). □

***Selection*:** On input $\langle r, \chi \rangle$, operator $\sigma_{A=a}$ returns the instance $\langle r', \chi' \rangle$ of the same sort defined by $r' = \{t \mid t \in r, t[A] = a\}$ and χ' is the restriction of χ to r'.

On input $\langle r, \chi \rangle$ of sort containing $\{A, B\}$, operator $\sigma_{A=B}$ returns the instance $\langle r', \chi' \rangle$ of the same sort defined by $r' = \{t \mid t \in r, t[A] = t[B]\}$. Concerning χ', for any $t \in r'$, and any $Y \subseteq sort(R')$ we have that

$$\chi'(t, Y) = \begin{cases} \chi(t, Y) & A, B \notin Y; \\ \chi(t, Y) \cap \beta(t, A) \cap \beta(t, B) & \text{otherwise.} \end{cases}$$

where $\beta(t, A)$ (resp. $\beta(t, B)$) is the set of colors of all blocks in t containing attribute A (resp. B). So, for tuples that satisfy the selection condition at the value level, only those blocks containing A (resp. B) for which there exists a block, of the same color, containing B (resp. A), are selected. If no such blocks exist, the selection attributes become unannotated. Thus, selection requires an agreement on both the data and block level, as one of our following examples illustrates (Example 5).

Similar to the selection operator, that identifies tuples with a particular data value, we offer a block selection operator that identifies blocks which have a specific color.

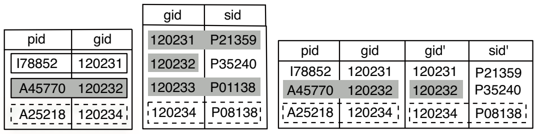
***Block Selection*:** The operator Σ_c, where $c \in \mathcal{C}$, takes as input any instance $\langle r, \chi \rangle$ and returns the instance $\langle r', \chi' \rangle$ of the same sort defined by

$$r' = \{t \mid t \in r \text{ and there exists a block in } t \text{ of color } c\},$$

and for any $t \in r'$ and any set of attributes $Y \subseteq sort(R)$, $\chi'(t, Y) = \chi(t, Y) \cap \{c\}$.

Example 4. *Consider again relation $\langle r, \chi \rangle$ in Figure 1(d). Assume that we want to find all the tuples that have a block annotated by Mary, or concern the protein with sid P08138. Also, assume that we are only interested in keeping the $\{gid, sid\}$ attributes from these tuples. Then, the expression*

$$\pi_{gid,sid}((\Sigma_{Mary}(r)) \cup (\sigma_{sid=\text{“}P08138\text{”}}(r)))$$



pid	gid
I78852	120231
A45770	120232
A25218	120234

(a) A relation

gid	sid
120231	P21359
120232	P35240
120233	P01138
120234	P08138

(b) Another relation

pid	gid	gid'	sid'
I78852	120231	120231	P21359
A45770	120232	120232	P35240
A25218	120234	120234	P08138

(c) Cartesian-product followed by selection

Figure 4. Selection and product operators

returns the desired result. Due to space limitations, we haven't introduced formally the union operator whose definition is rather straightforward (see [16] for all the definitions). Figure 3 shows both the intermediate results for each part of the union, and the final result of the query. Notice that the block selection operator maintains only the tuples that have at least one annotation from Mary. At the same time, from these tuples, the operator keeps only the blocks belonging to Mary. On the other hand, a selection predicate of the form $A = a$ focuses on values without altering the block structure. Our next example shows that this is not the case for selections of the form $A = B$.

The next two operators are of particular importance both for the completeness of our algebra and because, along with the selection operator, they define the color join.

Product: Given two instances $\langle r, \chi_r \rangle$ and $\langle s, \chi_s \rangle$ of disjoint sorts, the product operator × returns the instance $\langle r', \chi' \rangle$ with $sort(R') = sort(R) \cup sort(S)$ defined by $r' = r \times s$ (normal product). For any tuple $t \in r'$ and $Y \subseteq sort(R')$,

$$\chi'(t, Y) = \begin{cases} \chi_r(\pi_{sort(R)}(t), Y) & \text{if } Y \subseteq \mathit{sort}(R); \\ \chi_s(\pi_{sort(S)}(t), Y) & \text{if } Y \subseteq \mathit{sort}(S); \\ \emptyset & \text{otherwise.} \end{cases}$$

Merge: A natural operation on blocks is merging. The merge operator $\mu_{Y,Z}$, with Y, Z being sets of attributes such that $Y \cap Z = \emptyset$, takes as input instances $\langle r, \chi \rangle$ of sort $sort(R)$ containing $Y \cup Z$ and returns the instance $\langle r', \chi' \rangle$ of the same sort defined by $r' = r$. For any $t \in r'$ and any $X \subseteq sort(R)$,

$$\chi'(t, X) = \chi(t, X_1) \cap \chi(t, X_2),$$

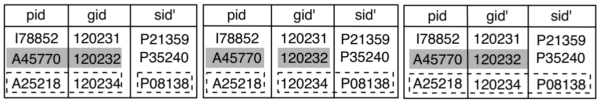
where $X = X_1 \cup X_2$, $X_1 \subseteq Y$, $X_2 \subseteq Z$ and $\chi(t, X) = \emptyset$. Intuitively, the merge operator considers each tuple t and it identifies pairs of blocks that are contained in Y and Z, respectively, and have the same color. Then, it replaces two equi-colored blocks with a new block that is the result of their merging. Blocks that are contained in Y and Z but cannot be merged, are dropped, as are the blocks not contained in Y and Z.

Example 5. *Consider relation $\langle r, \chi \rangle$ from Figure 2(b) and relation $\langle r', \chi' \rangle$ from Figure 3(c) (which are copied in Figures 4(a) and (b) for convenience). Figure 4(c) shows the relation that results from taking the product of r and r' and applying an equality condition on the gid attribute, that is,*

$$\sigma_{gid=gid'}(r \times \delta(r'))$$

where δ is a renaming function such that $\delta(gid) = gid'$ and $\delta(sid) = sid'$. Again, due to space limitations, we omit the formal definition of the renaming operator (see [16] for all the definitions). Notice that the selection only selects those blocks containing gid which have a equi-colored block containing gid', and vice versa. If no such blocks exists, as is the case for gid 120231, the resulting tuple is unannotated.

Now, what if we want to remove the duplicate gid column from Figure 4(c)? It turns out that, depending on which of the two gid columns we project out, we end up with a different block structure, as shown in Figures 5(a) and (b). We can avoid this by using the $\mu_{\{pig,\, gid\},\{gid',\, sid'\}}$ operator. After the merge operator is applied, irrespectively of which of the two attributes is projected out, the resulting relation is the same. This is shown in Figure 5(c).



pid	gid	sid'
I78852	120231	P21359
A45770	120232	P35240
A25218	120234	P08138

(a) Projecting out gid'

pid	gid'	sid'
I78852	120231	P21359
A45770	120232	P35240
A25218	120234	P08138

(b) Projecting out gid

pid	gid'	sid'
I78852	120231	P21359
A45770	120232	P35240
A25218	120234	P08138

(c) Projection after the merge operator is applied

Figure 5. The merge operator

The presented operators, along with the operators of union, renaming, and recoloring (which changes the color of a block), which are not presented here, constitute the whole set of operators of our algebra. Since the result of each operator on a color relation is again a color relation, we can compose all operators.

Definition 1. *The color algebra (CA) consists of all expressions obtained by composing a finite number of the operators mentioned above.* □

Our first theoretical result shows that we cannot hope to reduce the set of operators in our algebra and that all the operators are necessary. In the next section, Theorem 3 shows that our operators are also sufficient for our needs.

Theorem 1 (Minimality)**.** *The set of operators in the color algebra is minimal.*

Proof Sketch The proof considers each operator in turn and shows that it cannot be expressed in terms of the other operators. (see [16] for the full details) □.

We conclude this section with a few observations. The first observation concerns the color queries that are written using the color operators that have a relational algebra counter-part (e.g. selection (σ), projection (π), product ($\times$)). Each such color query results in the same set of tuples as the one we would get by applying the corresponding relational algebra query on an unannotated database. The difference between the two queries is the presence of blocks. Thus, by using the CA algebra, we don't *lose* any data.

Our second observation is that we can define a color *join* in CA as well. This join identifies attributes based on both their values and block structure and merges the "common" blocks. More specifically, we define $\langle r, \chi_r\rangle \bowtie \langle s, \chi_s\rangle$ by the expression (assuming we join on attributes A_i and A_j)

$$\pi_{sort(r)\cup sort(s)\setminus\{A_j\}}(\mu_{sort(r),sort(s)}(\Pi^L_{A_i}(\sigma_{A_i=A_j}(r \times s)) \cup (\Pi^L_{A_j}\sigma_{A_i=A_j}(r \times s))))$$

As noted in Example 5, we obtain an equivalent expression when projection includes A_j instead of A_i.

3. Connection with relational model

3.1. Relational representation

In this section, we provide a relational representation of color databases. In what follows, we define a mapping rep from color databases to a special type of relational databases. The inverse mapping rep^{-1} can also be defined but we omit the details due to lack of space.

Given a relation schema R with $sort(R) = \{A_1, \ldots, A_k\}$ we define the relation S_R of sort $\{A_1, \ldots, A_k, B_1, \ldots, B_k, \gamma\}$, where the attributes B_i are of type *Boolean*, the γ attribute is of type *color*, and there exist a bijection $assoc(A_i, B_i)$ mapping attribute A_i to attribute B_i and viceversa. We denote the class of relational databases satisfying the above schema constraints by $\mathcal{CR}ep$.

Let $\mathcal{CD}$ denote the class of color databases. The mapping is a function $\mathsf{rep} : \mathcal{CD} \mapsto \mathcal{CR}ep$ such that $\mathsf{rep}(R) = S_R$. Let $\langle r, \chi\rangle$ be a color relation instance over R. For each tuple $t \in r$, the representation $\mathsf{rep}(\langle r, \chi\rangle)$ contains the tuple $(t, 0, \ldots, 0, c)$, where c is a designated "blank" color not appearing in r. Furthermore, for each *annotated* tuple $t \in r$ and each $Y \subseteq sort(R)$ such that $\chi(t, Y) \neq \emptyset$, the relation $\mathsf{rep}(\langle r, \chi\rangle)$ contains the set of tuples

$$\{(t, B_1, \ldots, B_k, c) \mid c \in \chi(t, Y)\},$$

where $B_i = 1$ if $A_i \in Y$ and $assoc(A_i, B_i)$ holds, and $b_i = 0$ otherwise.

Note that the Boolean attributes in the representation are used to determine which of the corresponding data attributes belong to a block.

This concludes the definition of mapping rep. The extension to color databases, i.e., a set of color relations, is defined analogously.

Example 6. *Consider again the color relation $\langle r, \chi\rangle$ given in Figure 1(d). The following relation contains the three tuples in $\mathsf{rep}(\langle r, \chi\rangle)$ which correspond to the representation of the first tuple in $\langle r, \chi\rangle$. We assume that $assoc(pid, bpid)$, $assoc(gid, bgid)$, and $assoc(sid, bsid)$.*

pid	gid	sid	$bpid$	$bgid$	$bsid$	γ
$I78852$	120231	$P21359$	0	0	0	c
$I78852$	120231	$P21359$	1	1	0	$John$
$I78852$	120231	$P21359$	0	1	1	$Mary$

where c is a new color. The other tuples in $\mathsf{rep}(\langle r, \chi\rangle)$ are obtained similarly. □

A few words about the choice of representation. Note that we can normalize our representation so that the values of a tuple are not repeated for every block. Separating the data from the annotation representation not only saves space but also facilitates the incorporation of our mechanism to existing databases since no re-structuring of the existing schemas is necessary. We also note that our representation has several advantages over alternative representations. Indeed, we explored alternative representations whose schemas are exponentially large, with respect to the size of the schema of the relation to be annotated (since each possible block, i.e., set of attributes is represented by a separate column). Such alternative representations have several disadvantages including waste of space, since we need to allocate the space of a column for each possible block, even if this block does not exist in the currently annotated tuple. Furthermore, query processing is much more expensive and complicated in such representations.

3.2. Expressiveness

The relational representation of color databases suggests another candidate query language, namely the normal relational algebra on this representation and specifically the fragment consisting of the union of conjunctive queries. In this section, we establish a link between our algebra and this fragment of the normal relational algebra.

3.2.1 Color relational algebra (CRA)

It is clear that not every relational algebra query on a representation of a color relation results in a representation of a color relation again. For example, any projection consisting only of data attributes, does not correspond to a color database. It would therefore be desirable to identify the class of algebra expressions which, when applied to any representation of a color relation, results in a representation of a color relation.

Recall the $\mathcal{CR}ep$ is the set of relational databases which represent a color database through the rep mapping.

Definition 2. A positive relational algebra query Q is *colored* if for every relational database $\mathbf{D} \in \mathcal{CR}ep$, the query result $Q(\mathbf{D}) \in \mathcal{CR}ep$ as well.

We identify the following three necessary and sufficient syntactic conditions for a positive relational algebra query

to be colored. Without loss of generality we may assume that such query is given in normal form [4]. More specifically it is the union of conjunctive queries of the form $\pi_X \sigma_F(R_1 \times \cdots \times R_k)$. Clearly, a positive query is colored if it is the union of colored conjunctive queries. It is easily verified that a conjunctive query is colored iff it satisfies the following three properties:
(i) The projection must contain a single color attribute;
(ii) Since each data attribute corresponds to a unique Boolean attribute (and vice versa), queries should "respect" this relationship. In other words, if a data attribute is part of a query result schema, then the associated Boolean attribute should also be (and vice versa);
(iii) If some new data attributes are introduced which are in the schema of the query result, also associated new Boolean attributes should be introduced (and vice versa).

The above characterization is simple and provides an easy test (which runs in linear time in the size of the expression) to check whether a query, written as a union of conjunctive queries, is colored.

Definition 3. *The color relational algebra (CRA) consists of the class of colored positive relational algebra queries.*

3.2.2 Color algebra vs Color relational algebra

The color algebra (CA) and the class of color relational algebra (CRA) queries are closely connected. First of all, there exists a translation of any CA query into a CRA query. More specifically,

Theorem 2 (Soundness)**.** For every color database $\langle \mathbf{D}, \chi \rangle$ and every CA expression Q, there exists a color relational algebra (CRA) expression P such that

$$\mathsf{rep}(Q(\langle \mathbf{D}, \chi \rangle)) = P(\mathsf{rep}(\langle \mathbf{D}, \chi \rangle)).$$

Moreover, given the CA expression Q, the CRA expression P is of polynomial size, to that of Q.

Proof Sketch The proof consists of translating each operator in CA into a CRA query. (see [16] for the translation rules) □

The previous theorem gives us a way of implementing the CA on top of existing relational DBMS. Our color DBMS, represents color databases as described in Section 3 and when a CA query is issued, it translates it first into the corresponding CRA query and then to the equivalent SQL query. Then, the SQL query is executed in a standard relational DBMS.

Our theoretical result is that the CA has the same functionality (expressive power) as the CRA. More specifically:

Theorem 3 (Completeness)**.** For every relational database $\mathbf{D}$ in $\mathcal{CR}ep$, and every color relational algebra expression P, there exists a color algebra expression Q such that

$$\mathsf{rep}^{-1}(P(\mathbf{D})) = Q(\mathsf{rep}^{-1}(\mathbf{D})).$$

Proof Sketch The proof consists of a translation of any CRA query into a CA query and relies heavily on the fact that CRA queries only have a single color attribute in their projection. See [16] for details. □

Since the CA and CRA have the same expressive power why opt for one versus the other? For one thing, CRA queries are complex to write. The simple CA expression $\sigma_{A_i=A_j}(r)$ is equivalent to the following CRA expression (which consists of a union of four CRA queries):

$$\begin{gathered}\sigma_{A_i=A_j \wedge B_i=0 \wedge B_j=0}(\mathsf{rep}(r)) \cup \sigma_{A_i=A_j \wedge B_i=1 \wedge B_j=1}(\mathsf{rep}(r)) \cup \\ \pi_{\mathit{sort}(\mathsf{rep}(r))}(\sigma_{A_i=A_j \wedge B_i=1 \wedge B_j=0}(\mathsf{rep}(r)) \bowtie \\ \delta(\sigma_{A_i=A_j \wedge B_i=0 \wedge B_j=1}(\mathsf{rep}(r)) \cup \\ \pi_{\delta(\mathit{sort}(()r))}(\sigma_{A_i=A_j \wedge B_i=1 \wedge B_j=0}(\mathsf{rep}(r)) \bowtie \\ \delta(\sigma_{A_i=A_j \wedge B_i=0 \wedge B_j=1}(\mathsf{rep}(r))\end{gathered}$$

Clearly, a user cannot be expected to write such CRA expressions. The CA is not only simple syntactically but also independent of the underlying representation of annotations. Any change on the representation of annotations leaves CA queries unaffected. On the other hand, such a change would probably necessitate a re-writting of the equivalent CRA query.

4. The MONDRIAN System

The current status of the MONDRIAN system includes the following components. MONDRIAN is implemented on top of the MySQL relational DBMS. The MySQL server runs on a linux-based Pentium 4 PC (CPU 1.8GHz, 2GB RAM). On top of MySQL, we have implemented a module that accepts text-based CA queries. The module translates each such query first to its equivalent CRA query and then to an equivalent SQL query. The resulting SQL query is then sent to the MySQL server and is executed against the representation of an annotated database.

In our experiments, we used real biological data from the Swissprot [3] database. The relational representation of the Swissprot data was based on the schema of the UCSC Genome Browser database [19]. From this relational representation, we extracted two relations for our experiments, namely, relation *Protein* containing 200,000 protein tuples (560MB in size), and relation *Public* that contained four million tuples that concern publications related to proteins (750MB in size). Relation *Protein* has eight attributes in its schema, while relation *Public* has five. We used the above two relations as pools from which we generated three different experimental data sets. Each set contained five unannotated relations for each of the two pools. The sizes of these five relations varied from 10,000 to 50,000 tuples (in 10,000 tuple increments). Thus, the total number of created relations was 30. Each experimental data set allowed for executing experiments with different relation sizes. By using different data sets, we avoided any possible bias in the measurements due to characteristics of the underlying data.

A second module of the implementation was responsible for annotating unannotated relations. The annotation process is influenced by three user-specified parameters. The first parameter, called *MaxNo*, limits the number of blocks that can appear in each annotated tuple. The second parameter, called *AvgNo*, specifies the average size of each generated block. The last parameter is the cardinality of $\mathcal{C}$ (the number of available colors). In general, $\mathcal{C}$ can vary between one (all blocks have the same color) and the number of blocks in the relation (each block has its own color).

In this setting, we conducted two sets of experiments that (a) prove *feasibility*, i.e., that our annotation mechanism can be implemented efficiently in practice, and (b) study *performance*, i.e., the space/time overhead of our mechanism.
***(1)*:** In the first set, we compared the cost of executing CA queries over annotated databases to the cost of executing *equivalent* CRA queries over the corresponding unannotated databases.
***(2)*:** In the second set, we investigated how the annotation parameters influence the evaluation cost of CA queries.
All reported times are averaged over five runs of each experiment, over each of the different sets of (un)annotated relation instances (to rule out CPU interference and any bias from using a single instance). For annotated relations, the reported size is the number of annotated tuples and not the number of representation tuples. The cumulative size of the (un)annotated data sets used in these experiments is 26GB.

4.1. Costs of using colors and blocks

In this experiment, we investigate the additional cost, in terms of time and space, in querying an annotated database instead of an unannotated one. The experiment has two parts. Initially, we considered three types of queries, all written in relational algebra, where each type uses one of the operators of selection, projection and join. For example, one query projects on the id and description of a protein, while another selected only proteins of tomatoes. For the third type, we joined the *Protein* relation with the *Public* relation resulting proteins along with their associated publications. For each query type, we measured the evaluation time over our experimental data sets.

In the second part, we annotated the relations used in the first part. While generating the annotated relations, we used what we consider to be representative parameter values. We assumed that both *MaxNo* and *AvgNo* are equal to three, since tuples are expected to involve a small number of blocks with a few attributes in each one. Furthermore, we assumed that each color represents a curator and thus we set $\mathcal{C}$ to 100. Note that indeed the number of curators in Swissprot, one of the most heavily curated databases, is close to 40.

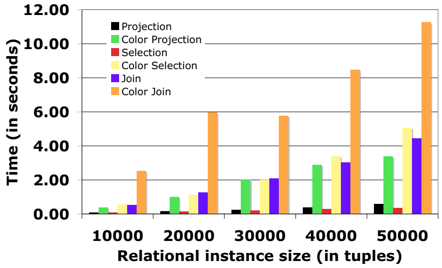


Figure 6. Color vs. normal algebra

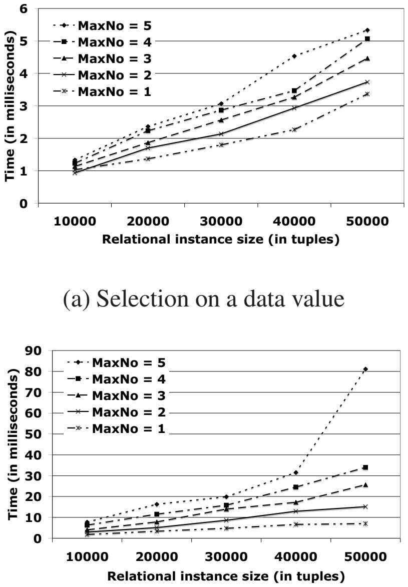
Given the annotated relations, we considered queries that, syntactically, are identical to the queries in the first part of the experiment. However, instead of the normal relational algebra operators, the queries used their color counterparts, i.e., they contained a CA projection instead of a projection, a CA selection instead of a selection and a color join instead of a normal join (note that a color join can be expressed through the select, merge and project CA operators). So, for example, the query that projects on the id and description of proteins was *translated* to a query that projects on the data *and* blocks of these two attributes. Given the queries, we measured their evaluation time over the annotated databases, and we compared these times with the times collected from the first part.

Figure 6 shows the results of this comparison for various relation sizes. Next to the cost of each relational operator, we show the cost of its color counter-part. In general, each color operator costs from three to five times as much as its relational counterpart. There are two main reasons for this. First, remember that color operators are actually applied on the representation of the annotated relation. This representation is bigger in size than the corresponding unannotated relation since it includes one tuple for each color of each block. With *MaxNo* equal to three, annotating a relation with 10,000 tuples results in a relation that is close to 30,000 tuples (assuming single colored blocks). Thus, while a normal projection is applied on 10,000 tuples, a color projection is actually applied on 30,000. Second, remember that color operators perform extra processing since they also consider Boolean attributes and operate on them.

Note that the overhead of supporting annotations is not prohibitive and it is more than balanced by the added value of being able to represent and query complex annotations. We expect that further optimizations of our implementation (e.g. use of specialized indexes) would further reduce the above costs making our solutions even more attractive.

4.2. Query evaluation cost parameters

In what follows, we investigate the relationship between evaluation cost of color queries and the three annotation parameters. Fo these experiments, we considered three different types of color queries. The first type included color queries that involved only the selection operator where the



(a) Selection on a data value

(b) Block selection and projection

Figure 7. Varying the *MaxNo* parameter

selection was on a data value. Note that the size of the result set of such queries is expected to be independent of blocks. The second type of color queries involved operators that are mostly block-dependent, namely, the operators of block projection and block selection. Finally, the third type of queries involved both block-independent and block-dependent operators. We used three different parameter configurations to annotate relations. For each resulting annotated relation, in each configuration, we executed queries of all three types and measured the corresponding evaluation times. In what follows, we present each parameter configuration and we review our key findings.

***Configuration 1*:** In this configuration, we annotated the relations in our experimental data sets once for each value of *MaxNo* between one and five. The *AvgNo* parameter was set to three and the $\mathcal{C}$ was 100. In Figure 7, we show the running times for the first and second type of queries, for different relation sizes (the third type exhibits the same running time trends as the second type). The main conclusion from these experiments is that the evaluation cost of most CA operators, with the exception of selections on data values, is heavily influenced by the maximum number of blocks per tuple. This is because this number increases the number of tuples in the underlying representation. The trend shown in Figure 7(b) is explained as follows. In the representation, there is one tuple for each color of each block. Assuming single-colored blocks, for an annotated relation with X tuples, there are $(\mathit{MaxNo} + 1) \times X$ representation tuples. As the figure shows, evaluation time increases sharply, when both *MaxNo* and X increase. In spite of the increase representation size, operations on the data side, like selections on data values, are not influenced by the variance of *MaxNo*, as Figure 7(a) illustrates.

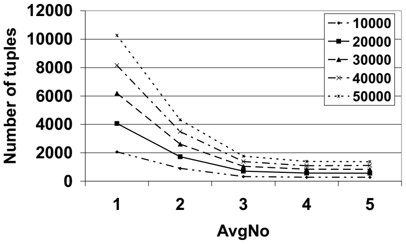


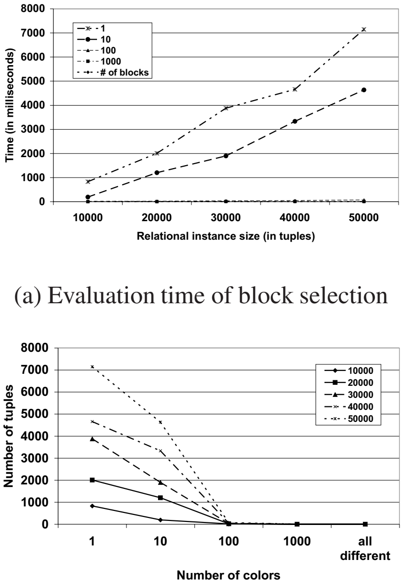
Figure 8. Result size of the Π^U operator

***Configuration 2*:** For our second configuration, we annotated relations of various sizes by varying, this time, the *AvgNo* parameter between the values of one and five. Parameter *MaxNo* was set to three and $\mathcal{C}$ to 100. Again, we executed color queries from all the types and recorded their evaluation times. Our experiments showed that varying the *AvgNo* parameter had negligible effects on the running times of various queries, for a fixed number of annotated tuples. To a large extent, this is to be expected since any variance of *AvgNo* does not influence the number of tuples in the underlying representation. As *AvgNo* increases, the only change, representation-wise, is that more Boolean attributes are set to 1, instead of 0. It is interesting to note the interaction between the value of *AvgNo* and the size of the result of block projections. As an example, in Figure 8 we show that, for a relation of fixed size, as we increase *AvgNo* we decrease the number of tuples in the result size of a U-type block projection on three attributes. This is because as we increase *AvgNo*, increasingly less blocks are contained within the projected attribute set.

***Configuration 3*:** Our last configuration considered annotated relations where both *MaxNo* and *AvgNo* were set to three. Here, five different values where considered for $\mathcal{C}$, namely, 1, 10, 100, 1000 and as many colors as there are blocks. Our experiments showed that the parameter has negligible effects on the evaluation times of algebra operators (since again availability of colors does not affect the representation size) with the exception of the block selection operator. As Figure 9(a) shows, when $\mathcal{C}$ is 10, there is a sharp increase on the evaluation time of the operator. The reason for this is shown in Figure 9(b). With less available colors, there is a large number of blocks sharing a color.

5. Conclusions and future work

In this paper, we proposed a new model for data annotations which, unlike previous works, it allows annotating not only values but also sets of values. Furthermore, our work is novel in that it also considers the importance of query-



(a) Evaluation time of block selection

(b) Result tuples of block selection

Figure 9. Block selection as $\mathcal{C}$ varies

ing annotations. To this end, we introduced a color algebra, and we proved that it is both complete and minimal. We presented the MONDRIAN annotation management system and we performed experiments on the feasibility and performance of our solutions.

We believe that colored databases provide the right framework to answer data provenance questions. Future work will be directed towards showing this. In terms of implementation, we are considering migrating our implementation from the MySQL server to MonetDB [8]. MonetDB offers a vertical fragmentation storage model and initial investigation shows that this model is particularly suitable for the processing performed by CA algebra operators. Finally, an interesting topic is the extension of our algebra to account for negation and the to allow blocks across multiple tuples.

Acknowledgments Floris Geerts is a postdoctoral researcher of the FWO Vlaanderen and is supported in part by EPSRC GR/S63205/01.

References

[1] Cathy H. Wu, Lai-Su L. Yeh, et al. The Protein Information Resource. Nucleic Acids Research, 31: 345-347, 2003.

[2] GDB(tm) Human Genome Database [database online]. url http://www.gdb.org/.

[3] The SWISS-PROT Protein Knowledgebase. URL: http://www.ebi.ac.uk/swissprot/.

[4] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[5] P. A. Bernstein and T. Bergstraesser. Meta-data support for data transformations using microsoft repository. *IEEE Data Eng. Bull.*, 22(1):9–14, 1999.

[6] D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An Annotation Management System for Relational Databases. In *VLDB*, pages 900–911, 2004.

[7] biodas.org. http://biodas.org.

[8] P. A. Boncz. *Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications*. Ph.d. thesis, Universiteit van Amsterdam, Amsterdam, The Netherlands, May 2002.

[9] P. Buneman. The two cultures of digital curation. In *SSDBM*, pages 7–, 2004.

[10] P. Buneman, S. Khanna, and W. C. Tan. Why and where: A characterization of data provenance. In *ICDT*, pages 316–330, 2001.

[11] P. Buneman, S. Khanna, and W. C. Tan. On propagation of deletions and annotations through views. In *ACM PODS*, pages 150–158, 2002.

[12] L. Chiticariu, W. C. Tan, and G. Vijayvargiya. Dbnotes: a post-it system for relational databases based on provenance. In *ACM SIGMOD*, pages 942–944, 2005.

[13] T. G. O. Consortium. The gene ontology (go) database and informatics resource. *Nucl. Acids Res*, 32:258–261, 2004.

[14] Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM Trans. Database Syst.*, 25(2):179–227, 2000.

[15] S. Davidson, G. C. Overton, and P. Buneman. Challenges in Integrating Biological Data Sources. *Journal of Computational Biology*, 2(4):557–572, 1995.

[16] F. Geerts, A. Kementsietsidis, and D. Milano. MONDRIAN: Annotating and querying databases through colors and blocks. Technical report. Available at http://www.lfcs.inf.ed.ac.uk/research/database/annotation.html.

[17] H. V. Jagadish, L. Lakshmanan, M. Scannapieco, D. Srivastava, and N. Wiwatwattana. Colorful xml: One hierarchy isn't enough. In *ACM SIGMOD*, pages 251–262, 2004.

[18] J. Kahan, M.-R. Koivunen, E. Prud'hommeaux, and R. R. Swick. Annotea: an open rdf infrastructure for shared web annotations. *Computer Networks*, 39(5):589–608, 2002.

[19] D. Karolchik, R. Baertsch, M. Diekhans, T. Furey, A. Hinrichs, Y. Lu, K. Roskin, M. Schwartz, C. Sugnet, D. Thomas, R. Weber, D. Haussler, and W. Kent. The UCSC Genome Browser Database. *Nucl. Acids Res*, 31:51–54, 2003.

[20] A. Kementsietsidis, M. Arenas, and R. J. Miller. Data Mapping in Peer-to-Peer Systems: Semantics and Algorithmic Issues. In *ACM SIGMOD*, pages 325–336, 2003.

[21] W. C. Tan. Containment of relational queries with annotation propagation. In *9th Int'l Workshop on Database Programming Languages (DBPL)*, pages 37–53, 2003.

[22] W.-C. Tan. Research Problems in Data Provenance. *IEEE Data Engineering Bulletin*, 27(4):45–52, Dec 2004.

[23] J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.