On Schema Discovery*

Renée J. Miller   Periklis Andritsos
Department of Computer Science
University of Toronto
{miller,periklis}@cs.toronto.edu

Abstract

Structured data is distinguished from unstructured data by the presence of a schema describing the logical structure and semantics of the data. The schema is the means through which we understand and query the underlying data. The schema permits the more sophisticated structured queries that are not possible over schema-less data. Most systems assume that the schema is predefined and is an accurate reflection of the data. This assumption is often not valid in networked databases that may contain data originating from many sources and may not be valid within legacy databases where the semantics of data have evolved over time. As a result, querying and tasks that depend on structured queries (including data integration and schema mapping) may not be effective. In this paper, we consider the problem of discovering schemas from data. We focus on discovering properties of data that can be exploited in querying and transforming data. Finally, we very briefly consider the suitability of mining approaches to the task of schema discovery.

1 Introduction

As the size, number, and complexity of databases continues to grow, understanding their structure and semantics gets more difficult. This, together with other associated problems such as the lack of documentation or the unavailability of the original database designers can make the task of understanding the structure and semantics of databases a very difficult one. At the same time, data sources may contain errors and may contain data that was integrated from many sources. This integration may introduce additional anomalies or duplicate data values and records. Fewer and fewer databases contain only raw data that has been put in electronic form for the first time and that has been carefully designed and structured for a well-defined application task. Rather, modern databases contain data that has been integrated and transformed, often many times, from other source(s).

No matter how principled or sophisticated the integration approach, we simply cannot always guarantee a perfect integration. Similarly, no matter how carefully a database was designed in the past, the data semantics may still evolve or errors may be introduced into the data. Hence, the data may be inconsistent or incomplete with respect to its schema (where the schema may include structuring primitives such as table declarations along with constraints). There are many forms of data inconsistency including incorrect or erroneous data values. A

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*Funding provided by NSERC, MITACS, a Premier's Research Excellence Award, and a gift from Microsoft.

data instance is typically said to be inconsistent if it does not conform to the constraints of the schema [ABC99]. Similarly, there are many forms of incompleteness including missing or unknown values or missing records [Gra02]. As with inconsistency, incompleteness is usually studied with respect to a schema. Notice that the assumption is that we trust the schema, that is, the schema is an accurate model of the time-invariant properties of the data. However, in both legacy databases (where the semantics of the data may have evolved since the schema was designed) and in integrated data (where data with different semantics may be added) this may not be a valid assumption.

In this paper, we briefly consider a different approach. Rather than viewing the data as being inconsistent or incomplete with respect to a given schema, we consider the schema to be potentially inconsistent or incomplete with respect to a given data instance. With this view, we consider whether we can propose techniques for discovering a better schema for the data. Of course, given a single data instance, we cannot be sure that the properties we find will hold for all possible instances (that is, we cannot be sure that these are time invariant properties). But we believe that automated techniques can be used to suggest potential schemas and that such suggestions can be incorporated into physical and logical data design and integration tools.

Below, we consider how having a schema that is inaccurate with respect to a given data instance can hamper both querying and the integration process itself. We then consider, in closer detail, the properties of schemas with an eye toward developing algorithms for schema discovery. We present one application of these ideas involving the clustering of tuples and values based on their structure.

2 Schema Properties

We use the term schema to refer to a set of structuring primitives (predicates), for example, relations or nested-relations, and a set of constraints (logical statements) over these primitives. Examples of constraints include key constraints and referential constraints. Constraints play an important role in data design and, as we will argue below, they are integral in understanding what forms of learning approaches are most useful for schema discovery. Before discussing the discovery problem, we examine in closer detail the role of schemas in providing structural and semantic information about data. We also consider the role schemas have played in the integration process.

Structured queries are by definition typed, that is, they take instances of a fixed schema (or type) and create an instance of a fixed schema [AHV95].[1] When schemas are inaccurate, queries can produce unexpected results. Consider the following schema and instance.

title	director	actor	genre	release date
Godfather II	M. Scorsese	R. De Niro	Crime	1974
Good Fellas	F. F. Coppola	R. De Niro	Crime	1998
Vertigo	A. Hitchcock	J. Stewart	Thriller	1958
N by NW	A. Hitchcock	C. Grant	-	1959
Alexander the Great	Baz Luhrmann	-	-	-
Water	Deepa Mehta	-	-	-
My Life Without Me	Isabel Coixet	-	-	-

Table 1: An instance of the movie relation

Intuitively, the first four tuples of this relation correspond to movies that have already been released and the last three tuples to movies that are currently being shot. We could imagine a situation where Table 1 has

[1]Of course, this analysis excludes work on higher-order or polymorphic queries [LSS01] which produces instances conforming to a family of schemas.

been produced after integrating two separate relations, one for released movies and another for pre-release films (which may include movies in production or movies that may never be shot or released) or a situation where the original data requirements included only released movies, but later applications required information about pre-release movies. In either case, the schema asserts that movies may have release dates and genres. A query analyzing trends in the genre of movies may give very misleading results as this information is systematically missing for pre-released films. Similarly, schemas in which constraints hold but are not expressed can lead to query results that are much different than anticipated. In our movie database, films may be associated with many companies which are responsible for various aspects of their production, financing, and release. However, films produced under the old studio system may all be associated with exactly one company. We can view such schemas as being incomplete (or more generally inaccurate) in that they fail to model important logical distinctions.

In addition to the effect on queries, inaccurate schemas can influence the integration process. Traditional integration approaches were very schema-centric [BLN86]. Given a set of schemas, these approaches would create an integrated schema or view. In the above example, it is not apparent from the schema that the movie relation contains two different types of movies. Hence, the integration of this database with another in which movies are separated by their production stage may result in all movies from our database being incorrectly assumed to belong to a single production stage (for example, they may all be assumed to be released movies). In more recent work, the focus of integration methods has shifted away from the problem of creating integrated schemas, to the problem of creating queries between independently created schemas [MHH00, HMN$^+$99]. Such approaches have been called query-centric [Ull03], to distinguish them from view-centric approaches which use views as a means of integrating data and modeling the query capabilities of different sources [LRO96, Hal01]. Query-centric and view-centric approaches have been combined in approaches that create more general mappings (sometimes called GLAV, global-and-local-as-view, or BAV, both-as-view, mappings) [Len02, PVM$^+$02, MP02]. However, because these structured queries and views are inherently typed, the effectiveness of all of these approaches depends to a large extent on the quality of the schemas being used. Logical semantics that are not modeled in a schema cannot be exploited. Mappings created over inaccurate schemas, such as the schema in Table 1, may incorrectly integrate or transform data (for example, by treating release and pre-released movies as a homogeneous collection). These mappings may produce unexpected or undesired results including (possibly) instances that do not conform to the target schema of the mappings.

In schema discovery, we will seek to discover logical structure that may be useful for integration or querying. Before presenting examples of discovery techniques, we consider one additional property of schemas that will be useful in designing and understanding schema discovery methods.

Schemas, like structured query languages that use them, treat data values largely as uninterpreted objects. This property has been called *genericity* [AHV95] and is closely tied to *data independence*, the concept that schemas should provide an abstraction of a data set that is independent of the internal representation of the data. That is, the choice of a specific data value (perhaps "Coppola" or "F.F.C." or "francis ford coppola" in our example) has no inherent semantics and no influence on the schema used to structure director values. The semantics captured by a schema is independent of such choices.[2]

For query languages, genericity is usually formalized by saying that a query must commute with all possible permutations of data values (where the permutations may be restricted to preserve a distinguished set of constants) [AHV95]. Similarly, we can formalize the genericity of schemas in the following way.

Definition 1: A schema S is said to be *generic* if for all instances I of S and for all permutations π of values in I, $\pi(I)$ is also an instance of S.

It is possible in SQL and other languages used in practice, to write non-generic queries (for example, using

[2]The term *generic* is sometimes used to mean *data model independent*. In this paper, we will use the adjective *generic* to refer to schemas that exhibit the genericity property.

user-defined functions) and non-generic constraints (for example, `check release-date < 2004`). However, such constraints are not foundational to structuring data and when used, are typically restricted to domains with a known, application-specific semantics (such as the ordering of years in this example). A similar constraint on movie titles (`check title < 2001`) is likely to be meaningless with respect to the semantics of movie data. Our thesis is not that such application-specific information is unimportant, but rather that it is often unavailable or inconsistent across different data sources which may use different conventions for encoding data values. Our techniques will therefore focus on the discovery of generic structure and constraints.

3 Schema Discovery Techniques

Schema discovery is not a new problem. Early approaches were motivated by the observation that an important source of inaccuracy in relational schemas is incompleteness, particularly in the constraints. Techniques for enumerating constraints that hold on a relational database include algorithms for finding dependencies (such as functional and multivalued dependencies) [BB95, Bel95, Bel97, KMRS92, WBX98]. Given an instance of a schema, the idea is to find all constraints within a specified class that describe the data. Of course, from a single database instance, we cannot determine whether the constraint is a dependency, that is, whether it will continue to hold as the data changes. However, these techniques can find a set of candidate dependendencies. Furthermore, several approaches consider the constraint mining problem over dirty data and propose techniques to find *approximate* dependencies that hold for most records [HKPT98]. The candidate dependencies found by such algorithms can be examined by a user or provided as input to a data design tool.

Constraint mining has applications beyond schema discovery (for example, to semantic query optimization). But certainly discovered constraints can be used to find "better" schemas. If additional constraints are found, then we can apply normalization methods to produce a (vertically) decomposed schema. Such methods are well-known to produce better schemas by reducing redundancy in the data and removing (or lessening) such evils as update anomalies. Notice that such a discovery approach operates by finding redundancy (specifically the redundancy that can be characterized by traditional relational dependencies) and removing this redundancy. Also, all of the constraint mining techniques we have mentioned are generic in that they do not rely on any application-specific semantics for the representation of data values.

Constraint mining looks to find a set of constraints (logical statements) that hold on a given data set and structure. The set of predicates used in these constraints is fixed – where the set of predicates corresponds to the set of tables defined in the schema. In our example, this means that the algorithms will search for constraints that hold over all movies and will not consider separating movie tuples into different tables and finding constraints that hold within each table. The re-structuring that is done is with respect to attributes. Attributes are regrouped into (potentially overlapping) sets, each representing a new table.

Now consider the problem of producing better schemas even when the set of predicates is not fixed. Conceptually, we would like to find a clustering of tuples such that each cluster can be described by a good schema. So we wish to find redundancy in the data, but redundancy that can be removed by partitioning tuples horizontally rather than vertically. Our approach is based on a clustering method called LIMBO [ATMS03]. Clustering produces a partitioning of objects where objects within the same cluster are similar and objects in different clusters are dissimilar. To apply clustering to this problem, we have to define a distance metric for relational tuples that is generic. In our movie example, it is not obvious how to measure the similarity (or dissimilarity) of the values "Coppola", and "Scorsese". Nor is it obvious how to define the distance between tuples "Vertigo" and "Good Fellas". If we are seeking to remove redundancy, intuitively we need a measure of similarity that reflects the redundancy in these tuples. However, even with such a measure, it is not obvious how to define a quality measure for the clustering. Yet, for humans there is an intuitive notion of quality for clustering. A good clustering is one where the clusters are *informative* about the tuples they contain. Since tuples are expressed over attribute values, we require that the clusters be informative about the attribute values of the tuples they hold. That is, given a cluster, we should be able to predict the attribute values of the tuples in the cluster to the greatest degree possible. Hence, highly redundant tuples will tend to be grouped together. The quality measure of the clustering is the information that the clusters hold about the attributes. Since a clustering is a summary of the data, some information may be lost. Our objective is to minimize this loss, or equivalently to minimize the increase in uncertainty.

Consider again the data in Table 1 and the partitioning of tuples into two clusters. Clustering C groups the first four movies together into one cluster, c_1, and the remaining three into another, c_2. Note that cluster c_2 preserves all information about the actor, genre and release date of the movies it holds. For cluster c_2, we know with certainty that these values are missing (or null). For cluster c_1, we have lost some information about these attributes so we have less certainty. But we do know, for example, that there is a 50% chance that the genre is "Crime". In this example, any other clustering will result in greater information loss. To see this, consider a clustering D again with two clusters. The first cluster d_1 contains the first three tuples and the second cluster d_2 contains the last four (so unlike in C, the fourth tuple has be assigned to the second cluster d_2). In d_2, we maintain certainty about the value of genre, but we lose information about actor and release date.

We have formalized this intuition and developed a scalable algorithm for clustering large categorical data sets [ATMS03]. We have applied our techniques to clustering both relational data and web data. Furthermore, LIMBO is a hierarchical algorithm that produces clusterings for a large range of k values (where k is the number of clusters). This property is very important for schema discovery since we can evaluate different k values (among the many clusterings produced in a single application of the algorithm) to determine one that achieves the best schema design.

We have argued that schema discovery involves characterizing (and eliminating) redundancy in the data. Our approach to characterizing redundancy is similar in spirit to recent work on using information theory to both characterize good schema designs and to compare the quality of different designs [AL03] (and to work characterizing the information theoretic properties of dependencies [DR00]). However, this work defines the information content of database elements with respect to a fixed set of constraints. Furthermore, the measure is restricted to valid database instances that satisfy these constraints. In our work, we have characterized the information content independent of constraints since such constraints are often unknown (or only partially known) and the data may contain errors. Nonetheless, there are important synergies between these results.

4 Conclusions

Schemas are typically the result of a data design process (performed by a human designer or by an automated tool). The choices made in data design are known to be highly subjective. A schema is inherently one out of many possible choices for modeling a data set. To understand and reconcile heterogeneous data, we may need to understand (and explicitly represent) some of the alternative design choices. Our current research focuses on the development of schema discovery techniques for finding such alternative designs. Our long term research goal is to find generic methods for discovering schemas that fit the data. Our work on LIMBO is a first step toward this goal. This work is part of a broader research agenda designed to develop solutions for creating, managing and transforming structure and meta-data, including schemas and mappings between schemas [AFF+02].

References

[ABC99] M. Arenas, L. Bertossi, and J. Chomicki. Consistent Query Answers in Inconsistent Databases. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 68–79, 1999.

[AFF+02] P. Andritsos, R. Fagin, A. Fuxman, L. M. Haas, M. Hernandez, C.-T. Ho, A. Kementsietsidis, R. J. Miller, F. Naumann, L. Popa, Y. Velegrakis, C. Vilarem, and L.-L. Yan. Schema Management. *Data Engineering Bulletin*, 25(3), September 2002.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Wesley, 1995.

[AL03] M. Arenas and L. Libkin. An Information-Theoretic Approach to Normal Forms for Relational and XML Data. In *PODS*, pp. 15-26, 2003.

[ATMS03] P. Andritsos, P. Tsaparas, R. J. Miller, and K. C. Sevcik. Limbo: Scalable Clustering of Categorical Data. Hellenic Database Symposium, 2003.

[BLN86] C. Batini, M. Lenzerini, and S. B. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[BB95] Siegfried Bell and Peter Brockhausen. Discovery of constraints and data dependencies in relational databases. In Nada Lavrač and Stefan Wrobel, editors, *Proceedings of the 8th European Conference on Machine Learning*, volume 912 of *LNAI*, pages 267–270, Berlin, April 1995. Springer.

[Bel95] S. Bell. Discovery and maintenance of functional dependencies by independencies. In *Proceedings of the Workshop on Knowledge Discovery in Databases*, pages 27–32. AAAI Press, 1995.

[Bel97] Siegfried Bell. Dependency mining in relational databases. In Dov M. Gabbay, Rudolf Kruse, Andreas Nonnengart, and Hans Jürgen Ohlbach, editors, *Proceedings of the First International Joint Conference on Qualitative and Quantitative Practical Reasoning*, volume 1244 of *LNAI*, pages 16–29, Berlin, June9–12 1997. Springer.

[DR00] M. M. Dalkilic and E. L. Robertson. Information Dependencies. In *PODS*, Dallas, TX, USA, 2000.

[Gra02] G. Grahne. Information Integration and Incomplete Information. *IEEE Data Engineering Bulletin*, 25(3):46–52, 2002.

[HMN+99] L. M. Haas, R. J. Miller, B. Niswonger, M. Tork Roth, P. M. Schwarz, and E. L. Wimmers. Transforming Heterogeneous Data with Database Middleware: Beyond Integration. *IEEE Data Engineering*, 22(1):31–36, 1999.

[Hal01] A. Y. Halevy. Answering Queries Using Views: A Survey. *The Int'l Journal on Very Large Data Bases*, 10(4):270–294, 2001.

[HKPT98] Y. Huhtala, J. Kärkkäinen, P. Porkka and H. Toivonen. Efficient Discovery of Functional and Approximate Dependencies Using Partitions. *Int'l Conf. on Data Engineering*, 392–401, 1998.

[KMRS92] M. Kantola, H. Mannila, K.-J. Rih, and H. Siirtola. Discovering Functional and Inclusion Dependencies in Relational Databases. *International Journal of Intelligent Systems*, 7(7):591–607, September 1992.

[Len02] M. Lenzerini. Data Integration: A Theoretical Perspective. In *Proc. of the ACM Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[LRO96] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 251–262, Bombay, India, 1996.

[LSS01] Laks V. S. Lakshmanan, Fereidoon Sadri, and Subbu N. Subramanian. Schemasql: An extension to sql for multidatabase interoperability. *ACM Trans. on Database Sys. (TODS)*, 26(4):476–519, 2001.

[MHH00] R. J. Miller, L. M. Haas, and M. Hernández. Schema Mapping as Query Discovery. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 77–88, Cairo, Egypt, September 2000.

[MP02] Peter McBrien and Alexandra Poulovassilis. Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach. In *Conference on Advanced Information Systems Engineering (CAISE)*, pages 484–499, 2002.

[PVM+02] L. Popa, Y. Velegrakis, R. J. Miller, M. A. Hernandez, and R. Fagin. Translating Web Data. In *Proc. of the Int'l Conf. on Very Large Data Bases (VLDB)*, pages 598–609, Hong Kong SAR, China, August 2002.

[Ull03] J. D. Ullman. Where Is Database Research Headed? Keynote. In *IEEE Conference on Database Systems for Advanced Applications (DASFAA)*, Kyoto, Japan, 2003.

[WBX98] S. K. M. Wong, C. J. Butz and Y. Xiang. Automated Database Schema Design Using Mined Data Dependencies. *Journal of the American Society for Information Science*, 49(5), pages 455–470, April 1998.