Automatically Inferring Database Schemas

Sitaram Lanka
Department of Computer ft Information Science
University of Pennsylvania
Philadelphia, PA 19104*

Abstract

The goal of this research is to investigate the possibility of automatically inferring a database schema. Our motivation is to make the task of the database designer easier. We require the designer only to provide a picture of how s/he expects the database to be used. This is provided in the form of natural language queries which the database might be expected to answer. The system synthesises a schema from this information. The above problem can be viewed as a problem in learning. The inference method we are proposing incrementally constructs the schema. The central idea of the inference mechanism is that it exploits certain features occurring in natural languages, namely, the syntactic structure of sentences.

1. Introduction

The goal of this research is to investigate the possibility of automatically inferring a database schema. The state of the art in database design is that, to create a database for a particular application, one has to express one's needs using certain concepts and notation specific to the database-management system. The person creating the database thus has to deal with concepts foreign to the application domain. Even for an expert, this is a tedious and error prone task. Our motivation is to make the task for the database designer easier. We desire that the designer should only be concerned with providing a picture of how s/he expects the database to be used and that the system should synthesis* a schema from this information.

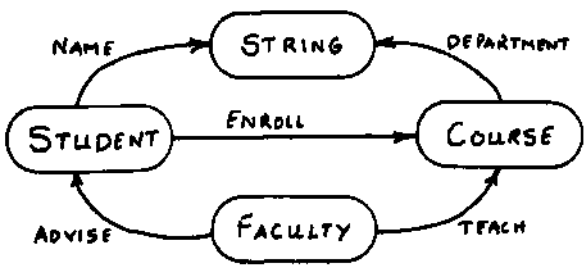


Figure above provides a pictorial illustration of a database schema. We have chosen *the functional data model* [Shipman 81][Nikhil 84] to express such schemas. A functional data model can be envisioned as a collection of database types (entities) together with a set of functions that operate on these database types. We shall represent a schema in this model by a collection of expressions of the form, $f(\alpha) \rightarrow \beta$, where f is to be thought of as a function whose domain is of type α and whose range is of type β. We will call an expression of this sort a database structure. Thus the above database schema, for example, will be represented by the following collection of expressions.

student() → entity
faculty() → entity
course() → entity
name(student) → string
enroll(student) → course
teach(faculty) → course
department(course) → string
advise(faculty) → student

We expect the database designer to provide a picture of the database by specifying sample English queries that the database system might be expected to answer. The following set of queries might serve this purpose for the above database.

1. Name the students enrolled for a course
2. Which faculty teach a course in the cis department?
3. Which faculty advise students?

From these queries we expect our system to synthesise the above database schema**.

The task of automatically inferring a database schema may be viewed as a problem in learning. Most of the work in machine learning has concentrated on inductive inference [Plotkin 71][Shapiro 81] and concept learning [Mitchell 82]. The assumption made in these methods is that the input is in some formal language and generalisations formed from them are expressed in the same language. Our work differs in that our input is in the form of parse trees of English sentences and the generalisations formed from them are expressed in the functional data model. That is, the input and the generalisations formed are expressed in different formalisms; this disparity makes it difficult to characterise the mechanism that forms the generalisation. This work also differs from those mentioned above in that they use notions such as *subsumption* to form generalisation [Plotkin 71] or impose some ordering on the hypothesis space [Shapiro 81][Mitchell 82], whereas we attempt to exploit certain features of the input, namely the syntactic structure of the sentence, to form abstractions from the input data. However, as this research develops we might take advantage of the notions such as subsumption and other techniques advanced by the above approaches to form further inferences or impose an ordering on the hypothesis space.

In the following sections we address the issue of automatically inferring a database schema in some detail. In section 2 we outline our basic approach. In section 2.1 we will explicate how information pertinent to the database structure is extracted from individual queries. Section 2.2 discusses how the information from different queries might be integrated to synthesise a database schema.

2. Inferring a database schema

The approach that we follow is that of constructing the database schema incrementally in the following sense. Each query is processed one at a time and the following two tasks are performed. First, the database structures corresponding to the query are inferred. Then an attempt is made to integrate this into the existing partial database schema. We discuss these two aspects in detail below.

2.1. Inferring database structures from individual queries

Database structures are abstracted from queries essentially

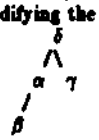
*This work is partially supported by the NSF grant #MCS81-07196 A02

**We are at this stage merely interested in inferring an appropriate schema. We are not at the moment concerning ourselves with the undoubtedly important task of associating a procedure with each of the function names.

mapping each individual query into a corresponding set of database structures. It is obviously not sensible to desire to do this by associating a rule with each query since there are far too many queries. What we require is a process that can be described in a finite manner but which is still capable of mapping a large set of queries into the appropriate structures. To achieve this goal we view a query as being composed of several smaller pieces, which are in particular its syntactic constituents. We then associate rules with specific syntactic constituents, each such rule performing the task of mapping the associated syntactic structure into a corresponding database structure. The database structures corresponding to an entire query are obtained as a cumulative effect of applying these rules to syntactic substructures of the query.

Each of the rules is motivated by the function the associated syntactic structure serves in a database setting. For example, noun phrases generally correspond to objects (database types), prepositional phrases to properties of objects, and verbs to relationships between objects. We can capture these generalities by associating rules that map, for example, noun phrases to objects, prepositional phrases to properties of objects and etc. We present below examples of some rules. Before doing so we need the following notions.

Definition 1: α dominates β if β is an immediate son of α. For example, in the tree below, α dominates β. In the case of a parse tree of an English sentence, if NP dominates PP it is to be understood as the PP modifying the NP.



Definition 2 : DB-property

i. If α is an NP and β a PP and α dominates β then

   a. if the preposition in β is 'of' then the common noun of α is a DB-property of the common noun (object) of β.

   b. otherwise, the common noun of β is a DB-property of the common noun (object) of α.

ii. If α β is a noun-noun compound and α is not an instance of β then α is a DB-property of the common noun of β.

For example, if the above definition is applied to the query 'What are the names of students in the database course?' we get *name* as a DB-property of *student*. Similarly, for the query 'Which faculty teach a course in the cis department?' we get *department* as a DB-property of *student*, and for the query 'Name the students enrolled for a course?' we get *name* as a DB-property of *student*.

Now for a few sample rules:

Syntactic Structure S_1: Noun Phrases.
Motivation for the rule: Common nouns correspond to objects (database types).
Rule R_1: *Common nouns that do not denote DB-properties (by definition 2) denote database types (entities).*

Syntactic Structure S_2: α dominates one or more structures $\beta, \gamma, \ldots$ where α is a NP and $\beta, \gamma, \ldots$ are PP's
Motivation for the rule: Preposition phrases correspond to properties of objects.
Rule R_2: *For each structure that α dominates, i.e., $\beta, \gamma, \ldots$ the common noun which is a DB-property (by definition 2) of an object denotes the function name and the object itself denotes the type of the domain.*[***]

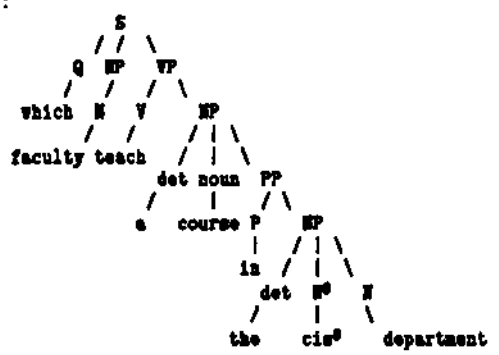
Syntactic Structure S_3: A sentence structure with α as its subject NP and one or more structures $\beta, \gamma, \ldots$ conjoined to the right of the main verb where $\beta, \gamma, \ldots$ are either NP's or PP's.
Motivation for the rule: Verbs correspond to relationships between objects.
Rule R_3: *The main verb denotes the function name. The common noun of the subject NP denotes the type of the domain of the function. The range of the function is the n-tuple $<r_1, r_2, \ldots, r_n>$, where each r_i is the common noun of $\beta, \gamma, \ldots$ conjoined to the right of the main verb*

We will now illustrate the use of these rules to infer database structures from the parse trees of queries. The assumption being made here is that parse trees of queries can be obtained and that the parsing mechanism uses a standard Chomsky transformational grammar and also filters out tense and inflectional markers. A rule is triggered if the structure associated with it matches a structure in the parse tree. Further the rules are unordered and that a single parse tree potentially triggers more than one rule.

Example 1

Q1: Which faculty teach a course in the cis department?

Parse tree:



The syntactic structure S_1 occurs in this parse tree[****], and so the associated rule R_1 is applied to yield the database structure

faculty () → entity
course () → entity

The syntactic structure S_2 also occurs in the parse tree, and so the associated rule R_2 is applied to yield the database structure

department(course) → string

Finally, the syntactic structure S_3 occurs in the parse tree and this triggers the rule R_3 to yield the database structure

teach(faculty) → course

3.3. Integrating database structures

The task here is to integrate the information inferred from individual queries to form a cogent database schema which is concise and consistent. Two database structures are conflicting if they assign different types to the same function. A database schema is consistent if no two structures in it conflict. The process of integrating can be thought of as consisting of two tasks: one task is to identify conflicting database structures and resolve them. The other is to represent concisely the database schema.

[***] The information provided by the parse tree is not adequate to infer the type of the range. Some additional information and heuristics are required for this purpose. For the present we will not concern ourselves with this issue.

[****] We assume that in a noun-noun compound, n_1 n_2, if n_1 is an instance of n_2 then n_1 is marked by the parser. Hence 'cis' is marked in the above parse tree. The motivation here is to facilitate the identification of the common nouns.

2.2.1. Resolving Conflicts

Conflicting database structures are inferred from queries and in fact they can be inferred from a single query. There are at least two circumstances that could potentially result in conflicting hypotheses[*****]. First, due to the inherent ambiguity of natural language, it is possible to generate multiple parse trees for a single query. These distinct parse trees may give rise to conflicting hypotheses. For example, consider the query

Q2: Which student is taking a course in the *cis* department?

There is ambiguity in where the PP - 'in the cis department' should be attached, thus yielding the two potential database structures,

(i) take(student) → course
(ii) take(student) → <course, department>

Second, application of different rules to a single parse tree may also lead to conflicting hypotheses. For example, consider the query

Q3: What are the grades of the students in the seminar course?

This query gives rise to the conflicting database structures

(i) grade(student) → string
(ii) grade(student, course) → string

In certain situations different functions may be inferred that actually bear a common name. For instance, from the query in Example 1, we can infer the database structure,

(i) course() → entity

From the query Q3, we infer the database structure,

(ii) course(student) → string

The functions in (i) and (ii) are actually different; the function in (i) returns all entities of type *course* whereas the function in (ii) returns the *course-name* the entity *student* is enrolled for. Functions of this sort are said to be overloaded. A set of hypotheses which include overloaded functions could also be mistakenly identified as conflicting hypotheses. It is therefore important to distinguish between these two cases.

In order to resolve conflicts we need to address two issues. First, distinguishing between function overloading and conflicting hypotheses; second, resolving conflicting hypotheses. If only conflicting hypotheses were present, then these could be resolved easily. For example, from the additional information (the existing partial database schema or the incoming data) if hypothesis, h_1, is inferred and if h_1 happens to be one of the conflicting hypotheses then we can resolve the conflicting hypotheses in favour of h_1. But the presence of overloaded functions rules out such a straightforward approach toward resolving conflicts. Alternatively, if negative examples were provided in the input data then this information can be used to rule out certain hypotheses and this may help resolve conflicting hypotheses. Since it is hard for the database designer to express in terms of what requirements the database should not satisfy we will assuming only positive instances to be provided. From the above arguments it is clear that we are unable to resolve conflicting hypotheses and indeed even to distinguish between these and overloaded functions. Therefore, as a last resort we will turn to the database designer (user) to provide some crucial information which will aid the task of distinguishing between conflicting hypotheses and overloaded functions and resolve conflicting hypotheses.

We will briefly give an overall picture of how the interaction with the user is carried and then illustrate it through an example. A common representation will be adopted to express both conflicting hypotheses and overloaded functions. They are represented by a *disjoint union*. For instance, $\mathcal{H}$: *A or B*, is a disjoint union where hypothesis $\mathcal{H}$ is either *A* or *B*. Let us assume we have such a set $\mathcal{H}$. Now from the additional information (the existing partial database schema or the incoming data) say a hypothesis, h_1, is inferred and h_1 is also part of $\mathcal{H}$. Since we have gathered more evidence on a hypothesis belonging to the disjoint union we could use this information to assess the other hypotheses in the disjoint union. At this stage we initiate the interaction with the user by asking if the other hypotheses in $\mathcal{H}$, i.e., $\{\mathcal{H} - h_1\}$, can be discarded. If the user's response is 'yes' then $\mathcal{H}$ can be thought of as conflicting hypotheses and it can be resolved in favour of h_1. If the user's response is 'no' then $\mathcal{H}$ can be construed as overloaded functions. The following example illustrates this,

Consider we have formed the disjoint union, $\mathcal{H}$, for the conflicting hypotheses inferred from query Q2. Next from the additional information if we infer the following database structure

h_1: take(student) → course[******]

Obviously the above database structure is one of the hypothesis in $\mathcal{H}$ hence at this stage the user is asked the explicit question

Can {take(student) → <course, department>} be discarded?

If user's response is 'yes' then the conflicting hypotheses $\mathcal{H}$ can be resolved in favour of h_1. If user's response is 'no' then $\mathcal{H}$ is construed as overloaded function.

2.2.2. Conciseness of representation: Type hierarchy

It is possible to encode information economically by arranging the objects of the database in a hierarchy using the notion of subtype. A database type α is a subtype of a database type β if all functions defined on β must be defined on α but α may have other functions defined on it. That is, the subtype inherits all the properties of its supertype. This provides an economy in representation since we need represent the common properties of the subtype and the supertype only once at the supertype. The information pertaining to the type hierarchy can be inferred but due to space limitations we shall not discuss them here.

The notion of *subtype* is also used for another purpose, to identify whether two functions are the same. For example, from the query - 'What are the names of the persons in the university?'; we infer the database structure,

(i) name(person) → string

and from the query - 'What are the names of the students enrolled for cis110 course?'; we infer

(ii) name(student) → string

If *student* is a subtype of *person* then both (i) and (ii) can be construed as a single function and a single procedure can be attached to both these functions.

3. Summary

In this paper we have indicated an approach to the problem of automatically inferring database schemas. Our motivation is to make the task for the database designer easier. The designer need only provide an idea of how s/he expects the database to be used and that the *system* synthesises a schema from this information. We have chosen the *functional data model* to express database schemas. The crux of the inference mechanism is to exploit certain features of natural languages, namely, the syntactic structure of sentences, to form abstractions.

4. Acknowledgments

I acknowledge gratefully the support and encouragement provided by Aravind Joshi and Rob Gerritsen. Many helpful ideas and criticisms were provided by Julia Hirschberg, Kathy McCoy and Gopalan Nadathur.

5. References

[1] Mitchell, T.M. "Generalization as search", Artificial Intelligence, Vol. 18, No. 2, (1982) 203-266.

[2] Nikhil, R.S. "An Incremental, Strongly typed, Database Query Language" PhD thesis, Dept. of Computer and Information Science, University of Pennsylvania, Philadelphia, 1984.

[3] Plotkin, G.D. "Automatic Methods of Inductive Inference" PhD thesis, Edinburgh University, 1971.

[4] Shapiro, E. "Inductive inference of theories from facts" Tech. report 192, Dept. of Computer Science, Yale University, New Haven, 1982.

[5] Shipman, D.W. "The Functional Data Model and the Data Language DAPLEX" ACM transactions on database systems 6(1), (1981) 140-173.

[*****] We will use the terms *conflicting hypotheses* and *conflicting database structures* interchangeably.

[******] This can be inferred from a query, 'which student is taking a database course?'