

Intelligent Query Answering by Knowledge Discovery Techniques *

Jiawei Han[†]

Yue Huang[†]

Nick Cercone^{†‡}

Yongjian Fu[†]

[†] School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada V5A 1S6.

[‡] Department of Computer Science, University of Regina, Regina, Saskatchewan, Canada S4S 0A2.

Abstract

Knowledge discovery facilitates querying database knowledge and intelligent query answering in database systems. In this paper, we investigate the application of discovered knowledge, concept hierarchies, and knowledge discovery tools for intelligent query answering in database systems. A knowledge-rich data model is constructed to incorporate discovered knowledge and knowledge discovery tools. Queries are classified into data queries and knowledge queries. Both types of queries can be answered directly by simple retrieval or intelligently by analyzing the intent of query and providing generalized, neighborhood or associated information using stored or discovered knowledge. Techniques have been developed for intelligent query answering using discovered knowledge and/or knowledge discovery tools, which includes generalization, data summarization, concept clustering, rule discovery, query rewriting, deduction, lazy evaluation, application of multiple-layered databases, etc. Our study shows that knowledge discovery substantially broadens the spectrum of intelligent query answering and may have deep implications on query answering in data- and knowledge-base systems.

Index Terms – Database and knowledge-base systems, knowledge discovery in databases, knowledge-rich data model, intelligent query answering, multiple layered databases, query analysis and query processing.

1 Introduction

Huge amounts of data are already being and will continue to be collected in a large number of databases by various kinds of data gathering tools, which creates both a need and an opportunity for extracting knowledge from databases. Knowledge discovery in databases (KDD), (or data mining), which refers to the *nontrivial extraction of implicit, previously unknown, and potentially useful infor-*

mation from the data stored in databases [8], has become an active area in both database and machine learning researches [27, 23].

With the rapid development of knowledge discovery techniques, it is natural to study the applications of the knowledge discovery technology in querying database knowledge and the impact of the technology to the development of intelligent query answering mechanisms in database systems.

In this paper, we investigate the application of concept hierarchies, discovered rules, and knowledge discovery tools to intelligent query answering in database systems. The study is based on the premise that efficient, relatively sophisticated knowledge discovery tools will soon become available for large database systems. Such a view has been reinforced by some preliminary studies and experimental results [8, 23, 9, 1, 3, 18, 11]. For example, in our previous studies [4, 9, 11], an attribute-oriented induction method has been developed for knowledge discovery in databases. The method integrates a machine learning paradigm, especially *learning-from-examples* techniques, with database operations and provides an efficient way for extraction of generalized data from actual data in databases. A knowledge discovery system prototype, DBLearn [11], has been constructed based on this methodology and has been experimented on several large relational databases with satisfactory performance. Other studies, such as [8, 16, 23], have also developed interesting knowledge discovery techniques and systems/prototypes, such as INLEN [18], KDW+ [24], Quest [1], IMACS [3], Datalogic/R [31], 49er [32], etc., which demonstrates the promising future of knowledge discovery.

Different from most of the previous studies on cooperative query answering [26, 25, 15, 7, 6, 30] and querying database knowledge [20] which emphasize on the application or inquiry of deduction rules and integrity constraints in relational or deductive databases, this study extends the domain of study from a deductive database to a knowledge-rich database assisted with generalized knowledge and knowledge discovery tools. A knowledge-rich data model is constructed which consists of not only the components from a deductive database (including database schemas expressed by an extended deductive entity-relationship model, data relations, deduction rules and integrity con-

*The work was supported in part by the research grants from the Natural Sciences and Engineering Research Council of Canada, IRIS-2 Grant HMI-5 from IRIS/PRE-CARN (Networks of Centres of Excellence Program of the Government of Canada), and a research grant from the Centre for Systems Science of Simon Fraser University.

straints) but also the components relevant to knowledge discovery processing, including concept hierarchies, generalized knowledge, and knowledge discovery tools. With the availability of concept hierarchies and generalized knowledge, queries can be posed and answered at levels higher than that of primitive concepts, and knowledge about general characteristics of data can be inquired or utilized in query answering. Furthermore, knowledge discovery tools can be used to extract general knowledge dynamically, when necessary, from any set of interested data in the database. A unified framework is established for answering data and knowledge queries in a knowledge-rich database. A systematic study is performed on intelligent query answering of data queries in a database system associated with discovered knowledge and knowledge discovery tools.

The remaining of the paper is organized as follows. In Section 2, a data model is constructed for knowledge-rich databases, which consists of both deductive database and knowledge discovery components. In Section 3, four basic categories of query answering mechanisms in a knowledge-rich database are introduced, according to the combinations of *data* vs. *knowledge* queries and *direct* vs. *intelligent* query answering mechanisms. In Section 4, knowledge discovery methods and tools associated with intelligent query answering are presented. Applications of knowledge discovery techniques for intelligent answering of data queries are studied in Section 5. A summary of our study and a discussion of the future research issues are provided in Section 6.

2 A Data Model for Knowledge-Rich Databases

To study intelligent query answering using knowledge discovery techniques, it is often necessary to distinguish data, knowledge and queries defined at the primitive data level from those defined at a high concept level. Data in a knowledge-rich database are classified into **primitive data** and **high-level data**. The former are actual data stored in data relations and, if appearing in some concept hierarchies, correspond to the primitive level (i.e., leaf) nodes of the hierarchies; whereas the latter are nonprimitive data subsuming primitive ones and residing at the nonprimitive level of concept hierarchies. Correspondingly, a **primitive-level query** is a query whose constants involve only primitive data; whereas a **high-level query** is a query whose constants involve high-level data. Similarly, rules (notice that *integrity constraints* can be viewed as a special kind of rules) can be classified into **primitive-level** and **high-level rules** based on whether they reference high-level data. Many deduction rules in *traditional* deductive databases are primitive-level rules because they do not reference high-level data. However, a deduction rule can also be a high-level one if it references high-level data. For example, “*every young faculty member in computing science has a Ph.D. degree in the field*” is a high-level rule if the concepts “*young*” and “*faculty*” are high-level concepts.

Moreover, since a rule can be defined explicitly or be extracted by a knowledge discovery process, it is important to differentiate a deduction rule from a generalized one. A *rule which is explicitly defined by a user or an expert* is a **deduction rule**; whereas a *rule which is generalized from a database state* is a **generalized rule**. Both deduction rules and generalized rules can be primitive-level or high-level rules. However, since a generalized rule summarizes data from a database state, it reflects a *general fact* in the current database state but does not enforce a *constraint* on the possible database states. This contrasts with a deduction rule or an integrity constraint which states a rule (or a constraint) that a *potential* database state *must* follow. For example, a generalized rule, “*all of the teaching assistants are graduate students*”, states a fact in the current database but does not claim that all the teaching assistants *must be* graduate students nor reject the *possibilities* of hiring an undergraduate student as a teaching assistant. However, the same rule, if stated as a deduction rule or an integrity constraint, will reject the possibility of allowing an undergraduate student to serve as a teaching assistant in a consistent database.

As an extension to the logic data model proposed in deductive database research [29], a *knowledge-rich data model* is constructed for databases with both deduction and knowledge discovery capabilities.

Definition 2.1 A **knowledge-rich database (KRDB)** consists of six components: (1) *Schema*, a knowledge-rich database schema; (2) *EDB*, an *extensional database*; (3) *IDB*, an *intensional database*; (4) *H*, a set of *concept hierarchies*; (5) *GDB*, a *generalized database*; and (6) *KDT*, a set of *knowledge discovery tools*, i.e., $KRDB = \langle Schema, EDB, IDB, H, GDB, KDT \rangle$. It is defined as follows.

1. **Schema**, a *knowledge-rich database schema*, describes the general structure and organization of KRDB including (i) *physical* and *virtual* entities, attributes and relationships, and (ii) the organization of rules, integrity constraints and concept hierarchies, based on a deductive entity-relationship data model [13].
2. **EDB**, an *extensional database*, consists of a set of extensional data relations.
3. **IDB**, an *intensional database*, consists of a set of *deduction rules* and *integrity constraints (ICs)*.
4. **H**, a set of *concept hierarchies*, specifies taxonomies of concepts on top of primitive data in extensional and intensional databases.
5. **GDB**, a *generalized database*, consists of a set of *generalized rules* which summarize the regularities of the data at a high level.
6. **KDT**, a set of *knowledge discovery tools*, performs knowledge discovery efficiently in databases, when necessary. \square

The first component, *Schema*, follows from a deductive entity-relationship model [13] which extends an entity-relationship model [5, 28] to incorporate rules, integrity constraints and complex data objects for deductive databases. Based on this model, a database schema consists of a set of entities and relationships, each of which (including their attributes) could be defined by *physical* data (i.e., data relations), *virtual* data (i.e., deduction rules), or their mixtures. Furthermore, physical or virtual entities and relationships are organized into an entity-relationship diagram, some of which may also form generalization or specialization hierarchies, with some of the properties of the lower level components (entities or relationships) inherited from their corresponding higher level ones. The details of such a model are described in [13].

The second and third components, *EDB* and *IDB*, are the same as in deductive databases [29] except that *IDB* rules can be defined by some nonprimitive data as well. Notice that a rule (or an integrity constraint) in the *IDB* can be first discovered by a knowledge discovery process and then be *recognized* and stored in the *IDB* as a regular rule or integrity constraint. However, once a *discovered* regularity is *recognized* and stored, it will play the same role as the *originally defined* one. Thus, we assume that all of the rules in *IDB* are defined ones.

The last three components, *H*, *GDB* and *KDT*, are the newly introduced knowledge discovery components which are used to incorporate discovered knowledge and knowledge discovery processes.

H, a set of *concept hierarchies*, represents the relationships among concepts at different levels. The information about concept hierarchies can be *provided* by knowledge engineers or domain experts or be *discovered* automatically or semi-automatically using knowledge discovery tools based on the statistics of data distribution in databases and the relationships among different attributes [10]. Many concept hierarchies are implicitly stored in the database. For example, the hierarchical relationship among “*city*”, “*province*” and “*country*” attributes are usually stored in the database and can be made explicit at the schema level by indicating a part-of-hierarchy: “*city* \subset *province* \subset *country*”. It is realistic to have some concept hierarchies provided by knowledge engineers or domain experts even in a large database system since a concept hierarchy registers *only* the *distinct* discrete attribute values or *ranges* of numerical values for an attribute, which is, in general, not very large. Further, by providing different concept hierarchies, users or experts may have preference to control the knowledge discovery or intelligent query answering processes.

GDB, the *generalized database*, is another important component in the knowledge-rich database. Since there are usually a very large set of generalized rules which can be extracted from any interesting subset of data in a database by performing generalization in different directions, it is unrealistic to store all of the possible generalized rules. However, it is often useful to store some generalized rules or intermediate generalized relations in the *GDB* based upon

the importance of the knowledge and the frequency of inquiries. The stored generalized rules are useful for querying database knowledge and semantic query optimization. Notice that a stored generalized rule should be updated incrementally in response to the updates of the relevant set of data to preserve its correctness. This can be performed by an incremental learning algorithm provided in knowledge discovery tools [8, 9].

The last component, *KDT*, consists of a set of *knowledge discovery tools*, which could be a set of knowledge discovery algorithms or a database-oriented knowledge discovery subsystem, such as INLEN [16], KDW++ [8], DBLearn [9], etc. Since a knowledge-rich database stores only a small portion of all of the possible generalized knowledge, it is often necessary to evoke a knowledge discovery process dynamically and extract general regularity from a specific set of data relevant to the query. The *KDT* tools can be used for on-line knowledge discovery and intelligent query answering.

An illustrative example of such a knowledge-rich database is presented in Example 2.1.

Example 2.1 Let a *university* database be modeled by a deductive entity-relationship model in which the extensional database (*EDB*) is mapped to a relational-like schema presented in Figure 1, where *Cnum* stands for *course number*, *TA* for *teaching assistant*, and *GPA* for *grade point average*.

Course (*Cnum*, *Title*, *Semester*, *Department*, *Instructor*,
TA, *Enrollment*, *Time*).
Professor (*Pname*, *Department*, *Salary*).
Student (*Sname*, *Status*, *Sex*, *Major*,
Birth_place(*City*, *Province*, *Country*), *GPA*).
Grading (*Student*, *Course*, *Grade*).

Figure 1: Schema of the University database

The concept hierarchies defined in the database are shown in Figure 2. The first three lines imply that the primitive data for *Status* is {*freshman*, ..., *Ph.D.*}, and their corresponding high-level data is *undergraduate* or *graduate* respectively. The entry “*Birth_place*(*City* \subset *Province* \subset *Country*)” indicates that the concept hierarchy for the attribute *Birth_place* is given by the data stored in the relation *Student* according to the *part_of* hierarchy: *City*, *Province* and *Country*. For example, a tuple,

Student(*Tom_Jackson*, ..., *Birth_place*(*Vancouver*,
BC, *Canada*), ...),

indicates that *Vancouver* is a part of *British Columbia* (*BC*), which is in turn a part of *Canada*, in the concept hierarchy for *Birth_place*.

Notice that there are many different kinds of hierarchical relationships among data in a database, such as *part_of*, *is_a*, *subset_of*, etc., which may play different roles in conceptual analysis. The different semantics among concept

hierarchies are not essential in the knowledge discovery algorithm itself since different concepts are generalized to their corresponding higher level concepts by following their corresponding concept hierarchies in a similar manner in the generalization process. However, such semantic differences will be important in the analysis of query intent and provision of intelligent answers.

```

{ freshman, sophomore, junior, senior } ⊂ undergraduate
{ M.S., M.A., Ph.D. } ⊂ graduate
{ undergraduate, graduate } ⊂ ANY (status)
{ biology, chemistry, computing, . . . , physics } ⊂ science
{ literature , music, . . . , painting } ⊂ art
{ science, art } ⊂ ANY (major)
{ 0.0 — 1.99 } ⊂ poor
{ 2.0 — 2.99 } ⊂ average
{ 3.0 — 3.49 } ⊂ good
{ 3.5 — 4.0 } ⊂ excellent
{ poor, average, good, excellent } ⊂ ANY (GPA)
Birth_place (City ⊂ Province ⊂ Country).
Birth_date (Day ⊂ Month ⊂ Year).

```

Figure 2: A concept hierarchy table of the database.

IDB rules are defined on top of EDB predicates. For example, *award_candidate* and *pre_requisite* are two IDB predicates defined in Figure 3.

$$award_candidate(P) \leftarrow status(P) = \text{“graduate”} \wedge gpa(P) \geq 3.75. \quad (2.1)$$

$$award_candidate(P) \leftarrow status(P) = \text{“undergraduate”} \wedge gpa(P) \geq 3.5. \quad (2.2)$$

$$pre_requisite(Course, Pre_requisite_course) \leftarrow edb_pre_requisite(Course, Pre_requisite_course). \quad (2.3)$$

$$pre_requisite(Course, Pre_requisite_course) \leftarrow edb_pre_requisite(Course, Required_course), pre_requisite(Required_course, Pre_requisite_course). \quad (2.4)$$

Figure 3: A set of deduction rules in IDB.

Figure 4 shows a set of generalized rules extracted from EDB and stored in GDB.

Our study below on intelligent query answering mechanisms will reference this database frequently. □

3 Four Categories of Query Answering Mechanisms in Knowledge-Rich Databases

In a knowledge-rich database system, there may exist two kinds of queries, *data queries* and *knowledge queries*, where

1. All of the teaching assistants are graduate students.

$$s \in Student \wedge c \in Course \wedge c.TA = s.Sname \rightarrow s.Status = \text{“graduate”}.$$

2. Every teaching assistant has a good or excellent grade point average.

$$s \in Student \wedge c \in Course \wedge c.TA = s.Sname \rightarrow s.GPA = \{\text{“good”}, \text{“excellent”}\}$$

Figure 4: A set of generalized rules stored in GDB.

Category of queries: $\left\{ \begin{array}{l} \text{Data query} \\ \text{Knowledge query} \end{array} \right.$

Query answering mechanisms: $\left\{ \begin{array}{l} \text{Direct query answering} \\ \text{Intelligent query answering} \end{array} \right.$

Figure 5: Categories of queries and query answering mechanisms.

a **data query** is to find concrete data stored in a database, which corresponds to a basic retrieval statement in a database system; whereas a **knowledge query** is to find rules and other kinds of knowledge in the database, which corresponds to querying database knowledge [20] including deduction rules, integrity constraints, generalized rules and other regularities. For example, “retrieving all of the students who took the course *CMPT-454* in 1994” is a data query; whereas “describing the general characteristics of those students” is a knowledge query.

Furthermore, it is often desirable to provide intelligent and assisted answers to queries *besides* (or *instead of*) direct retrieval of data and knowledge. Thus, query answering mechanisms in a knowledge-rich database can be classified based on their responses to queries into two categories: *direct query answering* and *intelligent (or cooperative) query answering*. **Direct query answering** is a direct, simple retrieval of data or knowledge from the knowledge-rich database; whereas **intelligent query answering** consists of analyzing the intent of query and providing generalized, neighborhood or associated information relevant to the query [7]. For example, simple retrieval of the names of the students who take the designated course is direct query answering to the above data query; whereas summarizing the characteristics of those students, such as “90% of them majored in computing science and took *CMPT-354* as prerequisites”, provides an intelligent answer to the same data query.

Based on such classifications (as shown in Figure 5), query answering mechanisms can be categorized into the following four combinations, each of which will be examined in this section.

1. (data query and direct answering): direct answering of data queries;
2. (data query and intelligent answering): intelligent answering of data queries;
3. (knowledge query and direct answering): direct answering of knowledge queries; and
4. (knowledge query and intelligent answering): intelligent answering of knowledge queries.

Notice that in many cases, a database user may not be able to distinguish between primitive and high-level data and between information that is data and information that is knowledge. Thus it is difficult for a user to explicitly indicate to which category a query belongs. A knowledge query can often be viewed as a *follow-up* to a data query when further explanation, reasoning or summarization are needed besides the answers to a data query. Therefore, it is important to provide a single, coherent framework to handle data and knowledge queries and to handle direct query answering and intelligent query answering.

3.1 Direct answering of data queries

Direct answering of data queries corresponds to direct data retrieval in knowledge-rich databases. Traditional query processing in relational and deductive databases belongs to direct answering of data queries. A *primitive-level data query* can be processed directly using relational and deductive query processing techniques.

A *high-level data query* can be processed in two steps. First, a query rewriting process can be performed to rewrite the query into one or a set of equivalent *primitive-level data queries* by substituting each high-level concept in the query with a set of or a range of its subordinate primitive-level concepts by consulting concept hierarchies in the KRDB. Second, each rewritten query is then fed into a relational or deductive query processor for processing. Answers should be returned at the primitive level. Presentation of answers at a nonprimitive level, when desired, is considered as a task of *intelligent* query answering and will be discussed in the next subsection. One example is illustrated below.

Example 3.1 To find the graduate students born in Canada, majoring in science, and with excellent GPAs, the query can be formulated in a syntax similar to SQL as follows.

```

retrieve Name
from Student
where Status = "graduate" and Major = "science"
      and Birth_place = "Canada"
and GPA = "excellent"

```

Notice that "graduate", "science" and "excellent" are high-level concepts which are not stored in the relation *Student*. Using the information stored in the concept hierarchy *H*, the query can be reformulated by substituting *graduate* with {*M.S.*, *M.A.*, *Ph.D.*}, and *GPA = "excellent"* with $GPA \geq 3.5$ and $GPA \leq 4.0$, etc. The rewritten query can be answered by direct data retrieval. \square

3.2 Intelligent answering of data queries

Intelligent answering of data queries refers to the mechanisms which answer data queries cooperatively and intelligently. Intelligent query answering is accomplished by analyzing the intent of a query and providing some generalized, neighborhood, or associated answers. There are many ways for a data query to be answered intelligently, including generalization and summarization of answers, explanation of answers or returning intensional answers, query rewriting using associated or neighborhood information, comparison of answers with those of similar queries, etc.

Example 3.2 The data query in Example 3.1 can be answered intelligently in many ways as illustrated below.

1. *Query rewriting using associated or neighborhood information.* For example, instead of printing only the names of the students satisfying the query condition, one may print other information as well, such as age, major, advisor, etc. associated with each student name. Such a technique is referred to as "width extension" [7].
2. *Generalization and summarization of the answers to a query.* For example, one may also print the summative information for the students satisfying the query condition, such as 45% of those students major in "computing science" and 20% major in "physics", etc.
3. *Comparison of the answer set with those under similar situations.* For example, one may print the comparative information for the referred students, e.g., there are 35 such graduate students (born in Canada, majoring in science, and with excellent GPAs) in comparison with 150 such undergraduate students, etc. \square

Mechanisms for implementations of intelligent answering of data queries using knowledge discovery techniques will be examined in detail in Section 4.

3.3 Direct answering of knowledge queries

A knowledge query is a statement which inquires about database knowledge, including concept hierarchies, deduction rules, integrity constraints and general characteristics of a particular set of data in a database. Direct answering of knowledge queries means that a query processor receives a knowledge query and answers it directly by returning the inquired knowledge. Since IDB knowledge and concept hierarchy information are stored in the database according to our assumption, a query on such knowledge can be answered by direct retrieval. However, the situation is different at querying generalized knowledge. A generalized database (GDB) usually stores only a small, but frequently used portion of generalized knowledge. Thus, an inquiry on general knowledge should be answered by direct retrieval only if the knowledge is available in GDB. Otherwise, the

knowledge should be discovered dynamically by a knowledge discovery process, which will be described in Section 4. In general, a knowledge query can be answered by consulting concept hierarchy, retrieving stored rules (if available) or triggering a discovery process.

Different syntactic specifications can be adopted to distinguish knowledge queries from data queries. A data query is to *retrieve the data elements that satisfy a condition Φ* ; whereas a knowledge query is to *describe the data elements that satisfy Φ* . Following the notion proposed by Motro and Yuan [20], data queries and knowledge queries are distinguished in syntax by starting the former with **retrieve** but the latter with **describe**. Further, to distinguish different types of knowledge being inquired, concrete keywords such as *generalized rule*, *deduction rule*, *concept hierarchy*, *integrity constraint*, etc. can be used after the keyword **describe**. Moreover, to query a discriminant rule which distinguishes the general characteristics of one class (*target class*) from others (*contrasting classes*), the following syntax is adopted:

describe generalized rule for relation
which distinguishes *target_class* from *contrasting_class*
where *condition Φ* .

Several knowledge queries are presented in the following examples.

Example 3.3 To find the deduction rule *award_candidate* for Canadian graduate students, a query can be formulated as below.

describe deduction rule *award_candidate(candidate)*
where *Status(candidate) = "graduate"*
and *Birth_place(candidate) = "Canada"*

This query can be answered by direct retrieval of deduction rules. Suppose that there are only two rules, (2.1) and (2.2), in the IDB defined for *award_candidate*. Notice that only (2.1) matches the condition, *Status(candidate) = "graduate"*. Furthermore, there is no further distinction on *birth place* in the condition for an award candidate in (2.1). Thus the rule (2.1) is presented as the answer to the query. \square

Example 3.4 To *describe the characteristics of the graduate students in computing science who were born in Canada with excellent GPA*, the query can be formulated as below.

describe generalized rule
for *Student*
where *Status = "graduate" and Major = "cs"*
and *Birth_place = "Canada"*
and *GPA = "excellent"*

Notice that the query represents a high-level knowledge query since *"graduate"*, *"Canada"* and *"excellent"* are not stored as primitive data in the *University* database. The query can be answered by directly retrieving the discovered rule, if available, or by performing induction on the relevant data set [9]. \square

Example 3.5 To *distinguish the characteristics of the graduate students from undergraduate students in computing science, born in Canada with excellent GPA*, the query can be formulated as below.

describe generalized rule for *Student*
which distinguishes *Status = "graduate"*
from *Status = "undergraduate"*
where *Major = "cs" and Birth_place = "Canada"*
and *GPA = "excellent"*

Notice that the query wishes to find a discriminant rule which contrasts the general properties of the two classes. The rule can be discovered dynamically by a knowledge discovery process from the primitive data [9] or from an intermediate generalized relation [12]. \square

3.4 Intelligent answering of knowledge queries

Intelligent answering of knowledge queries means that a knowledge query is answered in an intelligent way by analyzing the intent of the query and providing generalized, neighborhood or associated information. Similar to the intelligent answering of data queries, a knowledge query can be answered in many ways, such as generalization and summarization of answers, explanation of answers, query rewriting using associated or neighborhood information, comparison of answers with those of neighborhood queries, etc. The availability of database knowledge and knowledge discovery tools enhances the power and efficiency of intelligent query answering of knowledge queries, as illustrated in the following examples.

Example 3.6 The knowledge query of Example 3.3, which is to find the deduction rule *award_candidate*, can be answered intelligently not only by returning the *award_candidate* rule eligible to Canadian graduate students but also by (i) providing an explanation that both Canadian and foreign graduate students share the same condition for the award, (ii) returning the *award_candidate* rule eligible for undergraduate students as well, or (iii) returning other associated information, such as award titles, amounts, application deadlines, regulations, summary of award history, or statistical information, etc. \square

Example 3.7 The knowledge query of Example 3.4, which is to find the characteristics of designated graduate students, can be answered intelligently by returning the characteristic rule for Canadian graduate students with excellent GPA's, together with (i) the characteristics of Canadian graduate students with different majors or weaker GPAs for comparison, or (ii) an explanation of the reasons why such students got excellent GPA's, etc. \square

Intelligent answering of knowledge queries may involve greater complexity in query intent analysis and require more sophisticated implementation techniques than intelligent answering of data queries. Due to space limitation, this paper is focused on the efficient methods for intelligent answering of data queries and leave the in-depth discussion of the mechanisms for knowledge queries to future studies.

4 Knowledge Discovery Methods and Tools for Intelligent Query Answering

Many knowledge discovery methods have been developed in recent studies for mining knowledge from data [8], ranging from database search [1], generalization [9, 18], knowledge representation [2], to mathematical or statistical modeling [22, 31], etc. Based on the kinds of knowledge to be discovered from data, knowledge discovery tools can be classified into two major classes: (1) generalization-based discovery, and (2) discovering knowledge without generalization, as shown in Fig. 6. In each class, further classifications can be performed based on the kinds of rules or the form of knowledge to be discovered, including knowledge rules (characteristic, discriminant, clustering, dependency or association rules), generalized relations and multiple layered databases, etc. These discovery methods and the discovered knowledge will contribute to intelligent query answering.

Since the rules discovered without going through generalization processes represent data regularities at the primitive concept level, their roles in intelligent query answering are similar to that of deduction rules and/or integrity constraints defined at the primitive concept level, which have been discussed in intelligent query answering in relational or deductive databases [26, 25, 15, 7, 6]. Therefore, this study will emphasize more on the impact of **generalization-based discovery** to intelligent query answering. The techniques studied here are based on one generalization method: **attribute-oriented induction** developed in our previous studies [9, 11], with an emphasis on the derivation of prime relations, extraction of generalized feature tables, and construction of multiple layered databases. Nevertheless, the techniques studied here can be easily integrated with other generalization-based knowledge discovery methods.

4.1 Generalization and extraction of prime generalized relations

Data generalization, statistics summarization and generalized rule extraction are essential techniques for intelligent query answering. Generalization can be classified into two categories: (1) *attribute generalization*, and (2) *relation generalization*.

The first category refers to the generalization of certain attributes in one or a small set of tuples (e.g., the answer set) to certain high level concepts, which may help summarize data and expressing data at a high concept level. For example, instead of stating that “*Tom is a junior student, born in Vancouver on July 15, 1971*”, the statement can be generalized to a more summative statement, “*Tom is an undergraduate student, born in Canada in 1971*”. Attribute generalization can be performed by simply substituting some low level data in the answer set by the corresponding superordinate concepts at an appropriate level based on a query intent analysis (see Section 5).

The second category refers to the generalization of a relatively large data relation, which is usually a query-relevant portion of a database, resulted from query processing. The generalization can be performed efficiently by an attribute-oriented induction method [9, 11]. Here we present a similar process which extracts a special intermediate generalized relation, *prime relation*, to facilitate the extraction of different feature tables and the generation of various generalized rules for different purposes of intelligent query answering.

It is often desirable to express a generalized result using a small number of distinct (generalized) values for each attribute in the generalized relation. Also, a user or an expert may sometimes like to specify explicitly a designated concept level as the desirable level for an attribute, such as the level “country” for a birth_place, etc. Otherwise, a small integer is usually specified (or taken by default) as a **desirability threshold** for an attribute. An attribute is at the *desirable* level if it contains no more distinct values than its *desirability threshold*. It is **generalizable** if the attribute contains a relatively large number of distinct values in the relation but there exist higher level concepts (e.g., in a specified concept hierarchy) which subsume these attribute values. Otherwise, if there exist no such higher level concepts, it is **nongeneralizable**. A **prime relation** is a generalized relation in which each nongeneralizable attribute is removed and each generalizable attribute is generalized to the *desirable* level.

The extraction of a prime relation and the mapping of such a relation to interesting rules can be performed by the attribute-oriented induction in the following three steps.

1. **Relevant data collection.** A set of task-relevant data is collected by execution of a (direct) data query.
2. **Prime relation generation.** The collected data is generalized by (1) removal of nongeneralizable attributes, and (2) performing concept-tree ascension (replacing lower-level attribute values in a relation by their corresponding superordinate values using the concept hierarchy) on each generalizable attribute to its desirable level. By removal of nongeneralizable attributes and generalization of the values in the generalizable attributes to the desirable level, some generalized tuples in the relation may become identical. The identical generalized tuples are merged into one tuple, associated with a special internal attribute, *count*, which registers the number of original tuples generalized to the current one. The generalized relation obtained at this stage is the **prime relation**.
3. **Generalized rule extraction.** This step can be performed in two different directions. One direction is to derive a final generalized relation by further application of attribute-oriented induction. The other direction is to derive a generalized feature table for intelligent query answering, which is presented in the next subsection. The derivation of a final generalized relation is performed by further generalization on the

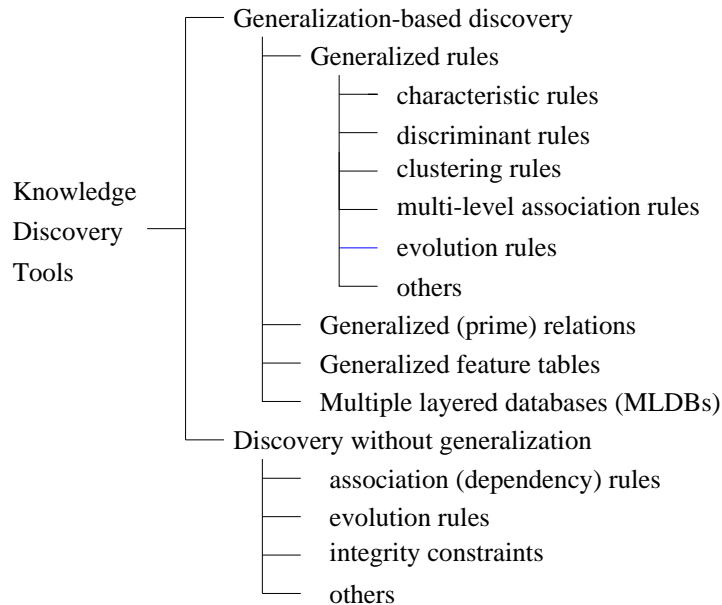


Figure 6: A classification of knowledge discovery tools.

attributes in the prime relation based on preferred rule forms, tuple reduction ratio, preferred concept levels of the attributes, etc. and then merge of identical generalized tuples with count accumulation. The final generalized relation, which consists of only a small number (usually controlled by a *generalization threshold*) of distinct generalized tuples, can be expressed in a logic form with the association of statistical information.

The core of the attribute-oriented induction is the concept-tree ascension on generalizable attributes, which relies heavily on the information stored in the concept hierarchy (H) in the KRDB. The concept hierarchy for numerical values can be constructed dynamically without prior knowledge based on the value range distribution in the database. For other hierarchies found in the KRDB, modification can be performed dynamically based on the statistics of current relevant data sets and user preference in order to extract interesting rules [10]. For example, to extract the interesting relationships between *GPA* and *Birth_place*, a given hierarchy can be modified dynamically to allow more detailed distributions of *Birth_places* in nearby provinces or countries than remote ones.

A prime relation maintains the relationships among generalized data in different attributes for a frequently inquired data set at a reasonable cost. The prime relation can be used for extraction of various kinds of generalized rules. Therefore, the prime relations for frequently inquired data sets can be stored in the GDB to facilitate intelligent query answering.

The algorithm for the extraction of a prime relation is presented as follows.

Algorithm 4.1 *Extraction of a prime relation \mathcal{R}_p from a frequently inquired data set \mathcal{R} .*

Input. (i) A frequently inquired data set \mathcal{R} , which is a relation of arity m with a set of attributes A_i ($1 \leq i \leq m$); (ii) a set of concept hierarchies, H_i , where H_i is a hierarchy on the generalizable attribute A_i , if available; and (iii) a set of *desirability thresholds* T_i for each attribute A_i .

Output. The prime relation \mathcal{R}_p .

Method.

1. Collect the statistics of the initial relation \mathcal{R} .
2. Compute the minimum desired level, determine the mapping pairs (v, v') for each attribute, where v is the (original) attribute value and v' the generalized one, and perform concept tree climbing.

$\mathcal{R}' := \mathcal{R}$; /* \mathcal{R} is a frequently inquired data set, obtained by a data query. */

```

for each attribute  $A_i$  in  $\mathcal{R}'$  do {
  if  $A_i$  is not at the desirable level and is nongeneralizable
  then remove  $A_i$ ;
  if  $A_i$  is not at the desirable level but generalizable
  then generalize  $A_i$  to the desirable level;
}
  
```

3. Derive the prime relation, \mathcal{R}_p , by merging identical tuples in \mathcal{R}' (with the number of identical tuples registered in *count*). \square

The generalization is essentially based on the generalization rules, *dropping conditions* and *climbing generalization trees*, in *learning-from-examples* [17]. However, it is performed in attribute-oriented fashion, which substantially reduces the computational complexity when performed in large databases.

Theorem 4.1 *The worst-case time complexity of Algorithm 4.1 is $O(n \log p)$ where n is the number of tuples of the initial relation \mathcal{R} and p is the number of tuples in the prime relation \mathcal{R}_p .*

Proof. The total processing cost is the accumulation of the cost of the following three parts.

- Step 1 collects the statistics of the initial relation, which scans the copied initial relation \mathcal{R}' , collects the distinct attribute values in \mathcal{R}' , and registers the number of occurrences of each distinct value in \mathcal{R}' . This step takes $O(n)$ time since the process scans \mathcal{R}' exactly once.
- Step 2 computes the minimum desired level and determines the mapping pairs (v, v') for each attribute. Suppose there are m relevant attributes and the average number of distinct values for each attribute is d , the average number of levels to be climbed up is l , and the average cost of tree-climbing is c . Then the cost of Step 2 should be $m \times d \times l \times c$, which is in the order of $O(n)$, since $d \leq n$, and m , l , and c are small constants.
- Step 3 derives the prime relation by merging equivalent tuples, which is performed as follows. For each tuple t in \mathcal{R}' , substitute its attribute values based on the value mapping-pairs derived in Step 2. This results in a generalized tuple t' . If t' is not already in the prime relation \mathcal{R}_p , insert it into \mathcal{R}_p and set its *count* to 1. Otherwise, increment the *count* of t' in \mathcal{R}_p . Since there are total p tuples in \mathcal{R}_p , the search takes $O(\log p)$ time if a B-tree structure is used. The total cost will be $O(n \log p)$.
- Summing-up the cost of the three portions, it is clear that the worst case time complexity should be $O(n \log p)$. \square

Notice that if the production of the thresholds, $\prod_{i=1}^m T_i$, is small, a multi-dimensional array may be used to register the number of tuple occurrences in each array element in the derivation of the prime relation by mapping the generalized tuples into corresponding array elements. In this case, the time complexity is $O(n)$. However, when $\prod_{i=1}^m T_i$ is very large, such an array could be too big and too sparse to be useful, and the B-tree structure mentioned above should still be used to keep the worst case time complexity to $O(n \log p)$.

The attribute-oriented induction technique avoids inefficient search in tuple-oriented generalization, explores the integration of well-implemented relational operations with the learning-from-examples algorithm, and leads to an easily implementable and highly efficient algorithm for induction in large databases [9].

Example 4.1 Suppose a frequently inquired data set in the University database collects the information about computing science students and their associated teaching assistant information. Let the prime relation be generalized

Status	Sex	Age	Birth_country	GPA	IsTA	count
grad	M	25_30	Canada	good	Y	8
grad	F	25_30	Canada	excellent	Y	2
...
underg	M	16_25	Asia	good	N	6

Table 1: A prime relation for the interested set of students.

from a data set obtained by a selection from the relation *Student* using, *Major* = “*computing_science*”, plus an attribute *IsTA* to indicate whether the student is a teaching assistant (i.e., whether the tuple is joinable with the *Course* relation on the teaching assistant attribute). The scheme for the initial data relation is,

Student(*Sname*, *Status*, *Sex*, *Birth_date*(*Day*, *Month*, *Year*),
Birth_place(*City*, *Province*, *Country*), *GPA*, *IsTA*).

Following Algorithm 4.1, the attributes *Sname*, *Day*, *Month*, *City*, and *Province* are removed, and the prime relation has the scheme,

Student(*Status*, *Sex*, *Age*, *Birth_country*, *GPA*, *IsTA*),

where *Age* is from *Birth_date*(*Day*, *Month*, *Year*), and *Birth_country* is from *Birth_place*(*Country*). After generalization, the prime relation contains only a small number of distinct values in each attribute as follows: *Status*: {*grad*, *undergrad*}, *Sex*: {*male*, *female*}, *Age*: {16_25, 26_30, >30}, *Birth_Country*: {*Canada*, *USA*, *Asia*, *Europe*}, *GPA*: {*poor*, *average*, *good*, *excellent*}, and *IsTA*: {*Yes*, *No*}. The prime relation is shown in Table 1. \square

4.2 Extraction of generalized feature tables and generalized rules

A prime generalized relation, though compressed substantially from the original relevant set of data, may still not be compact enough to disclose interesting relationships among different attributes. To facilitate intelligent query answering, a prime generalized relation can be mapped into several generalized feature tables from which a variety of interesting generalized rules can be extracted.

A **generalized feature table** is a two-dimensional table generated from the prime relation. It represents the occurrence frequency of a set of generalized features in relevance to one or a set of reference attributes in the prime relation.

The following algorithm extracts a feature table from a prime relation.

Algorithm 4.2 *Extraction of the feature table T_A for an attribute A from the prime relation \mathcal{R}_p .*

Input. A prime relation \mathcal{R}_p consists of (i) an attribute A with distinct values a_1, \dots, a_m , (ii) k other attributes B_1, \dots, B_k (suppose different attributes have distinct values), and (iii) a special attribute, *count*.

Output. The feature table T_A for the attribute A .

Method.

1. The feature table T_A consists of $m + 1$ rows and $l + 1$ columns, where m is the number of distinct values in the attribute A , and l is the total number of distinct values in all of the other k attributes. Each slot of the table is initialized to 0.
2. Each slot in T_A (except the last row) is filled by the following procedure,

```

for each row  $r$  in  $R_p$  do {
  for each attribute  $B_i$  in  $R_p$  do
     $T_A[r.A, r.B_i] := T_A[r.A, r.B_i] + r.count;$ 
     $T_A[r.A, count] := T_A[r.A, count] + r.count;$ 
}

```

3. The last row p in T_A is filled by the following procedure:

```

for each column  $s$  in  $T_A$  do {
  for each row  $t$  (except the last row  $p$ ) in  $T_A$  do
     $T_A[p, s] := T_A[p, s] + T_A[t, s];$ 
}

```

Theorem 4.2 *The worst-case time complexity of Algorithm 4.2 is $O(a \times p)$, where a is the number of attributes and p is the number of rows in the prime relation \mathcal{R}_p .*

Proof. There are a attributes and p rows in \mathcal{R}_p , and each attribute in each row in \mathcal{R}_p will be examined exactly once in the computation according to the nested loop structure of the algorithm. Moreover, each such examination will take only a few steps of calculations, i.e., bounded to a constant time c . Thus the total processing cost is $O(a \times p)$. \square

Since the number of attribute and the number of rows in the prime relation \mathcal{R}_p is in general small, Algorithm 4.2 is fairly efficient. The algorithm is optimal in the sense that we have to look at each attribute value of each row in \mathcal{R}_p at least once.

Different feature tables can be extracted from a prime relation based on the interest of particular reference attributes. The following example shows how the feature table in relevance to one reference attribute, *Status*, is extracted from the prime relation of Example 4.1.

Example 4.2 The feature table in relevance to the attribute *Status*, called *Status* feature table (Table 7), is extracted from the prime relation of Example 4.1. It is useful when comparing students who possess different status (graduates vs. undergraduates).

The *Status* feature table consists of 3 rows: each of the two distinct *Status* values in the prime relation {"*grad*", "*undergrad*"} corresponds to one row, and the last row (*total*) is the summation of information in the previous rows. The table consists of 5 major columns, each corresponding

to one attribute in the prime relation, plus one special column for *count*. Each major column in the table is further divided into k subcolumns (where k is the number of distinct general values in the corresponding major column), each corresponding to one distinct value in the attribute. For example, *GPA* is divided into 4 subcolumns: {*poor*, *avg*, *good*, *exclnt*}, each corresponding to one distinct value in *GPA*.

The table contents are derived from the prime relation as follows. Each slot in the table (except for the last row) corresponds to the number of occurrences of the corresponding values in the prime relation. For example, the slot for "*grad*" and "*good (GPA)*" corresponds to the number of grad's with good GPAs, that is, the summation of all of the count of those rows with *Status* = "*grad*" and *GPA* = "*good*" in the prime relation. The special column *count* registers the number of occurrences of the corresponding class in the relation. For example, 50 in "*grad*" indicates that there are 50 graduates in total in the prime relation. The special row *total* summarizes the total number of occurrences with each feature in all of the classes. For example, total = 160 in the column "*Sex = M*" indicates that there are totally 160 male students computed in the prime relation.

The method can be easily extended to generate feature tables in relevance to more than one reference attribute. \square

Since a feature table registers the number of occurrences for each general feature in the prime relation, with two specific properties: *total* and *count*, the quantitative relationships between the corresponding general features can be easily referenced and transformed into a set of interesting general rules to facilitate intelligent query answering.

The following algorithm extracts generalized rules from a feature table.

Algorithm 4.3 *Extraction of generalized rules from the feature table T_A .*

Input. A feature table T_A for the attribute A , where A has a set of distinct generalized values a_1, \dots, a_m . Another attribute B in the table has a set of distinct generalized values b_1, \dots, b_n . The slot of the table corresponding to the row with the value a_i and the column with the value b_j is referenced by $T_A[a_i, b_j]$.

Output. A set of generalized rules relevant to A and B extracted from the feature table.

Method.

1. For each row a_i , the following rule is generated in relevance to attribute B , which presents the distribution of different generalized values of B in class a_i .

$$a_i(x) \rightarrow b_1[p_{i_1}] \vee \dots \vee b_n[p_{i_n}],$$

where p_{i_j} is the probability that the value b_j of B is in class a_i , which is computed by,

$$p_{i_j} = T_A[a_i, b_j] / T_A[a_i, count].$$

Status	Sex		Age			Birth_country				GPA				IsTA		count
	M	F	16_25	26_30	>30	Canada	USA	Asia	Europe	poor	avg	good	exclnt	Y	N	
grad	40	10	10	20	20	30	5	10	5	1	1	30	18	30	20	50
underg	120	80	140	60	0	130	40	30	0	15	90	70	25	0	200	200
total	160	90	150	80	20	160	45	40	5	16	91	100	43	30	220	250

Figure 7: A *Status* feature table mapped from the prime relation.

- For each column b_j , the following rule is generated in relevance to all of the classes, which presents the distribution of the generalized value b_j of B among all of the classes.

$$b_j(x) \rightarrow a_1[q_{1j}] \vee \dots \vee a_m[q_{mj}],$$

where q_{ij} is the probability that the value b_j of B is distributed in class a_i among all of the classes, which is computed by,

$$q_{ij} = T_A[a_i, b_j] / T_A[total, b_j].$$

□

Theorem 4.3 *The worst-case time complexity of Algorithm 4.3 is $O(r \times c)$, where r is the number of rows and c the number of columns of the feature table T_A .*

Proof. Algorithm 4.3 extracts generalized rules from each column and each row in the feature table by computing the proportion of the number of occurrences of each generalized feature vs. its corresponding total number of occurrences in each row or each column. In Step 1, it computes total r rows with each computing c columns and thus takes total $O(r \times c)$ time. Similarly for Step 2. Thus, we have the theorem. □

Example 4.3 The extracted feature table is useful for derivation of the relationships between the classification attribute and other attributes at a high level. For example, the generalized rule, *all of the teaching assistants are graduate students*, can be extracted from Table 7 based on the fact that the class *grad* takes all of the *IsTA* count.

The feature table is especially useful for generation of rules associated with quantitative (statistical) information. For example, the following quantitative rule can be derived from the first row *grad* and the first major column *Sex*,

$$grad(x) \rightarrow male(x)[80\%] \vee female(x)[20\%].$$

The rule indicates that *80% of (computing_science) graduate students are male and 20% are female*. Obviously, the rules extracted from the feature table are useful for presenting summary and neighborhood information in intelligent query answering. □

By extraction of generalized feature tables and generalized rules, many of the intelligent query answering mechanisms which require knowledge discovery tools can be implemented efficiently. For example, summarization of

statistics of the answer set, comparison of the answer set with those of neighborhood queries, etc. can be realized by extraction of different rules from the corresponding generalized feature table.

4.3 Construction of multiple-layered databases

Besides the extraction of characteristic rules, discriminant rules and prime (generalized) relations, an important technique to facilitate intelligent query answering is the construction of multiple-layered databases [12].

Although the extraction of the rules or prime relations provides a flexible means for intelligent query answering, it has two drawbacks: (1) the discovered knowledge is often too task-relevant to be readily applied to other situations, and (2) it is often too costly to extract such knowledge dynamically. For example, a prime relation about the general characteristics of the graduate students majoring in computing science, with GPA greater than 3.5, though takes some processing effort, may hardly be useful for answering similar queries about senior students majoring in engineering. It is infeasible to store the generalized data of all the different combinations of the relevant sets of data, but it is too costly to generalize each such data set dynamically. A good compromise is to store, based on the query accessing frequency, some generalized relations in the GDB as higher layered data. This leads to the construction of a multiple layered database.

A **multiple layered database (MLDB)** is a database composed of several layers of information, with the lowest layer corresponding to the primitive information stored in a conventional database, and with higher layers storing more general information extracted from lower layers.

Similar to the derivation of prime relations by attribute-oriented induction, the algorithm for construction of an MLDB is presented below.

Algorithm 4.4 Construction of an MLDB.

Input. A relational database, a set of concept hierarchies, and a set of frequently referenced attributes and frequently used query patterns.

Output. A multiple layered database.

Method. An MLDB is constructed in the following steps.

- Determine the multiple layers of the database based on the frequently referenced attributes and frequently used query patterns.

2. Starting with the most specific layer, generalize the relation step-by-step (using the given concept hierarchies) to form multiple layered relations (according to the layers determined in Step 1).
3. Merge identical tuples in each generalized relation and update the *count* of each generalized tuple.
4. Construct a new schema by recording all the primitive and generalized relations, their relationships and the generalization paths. \square

Rationale of Algorithm 4.4.

Step 1 indicates that the layers of an MLDB should be determined based on the frequently referenced attributes and frequently used query patterns. This is reasonable since to ensure the elegance and efficiency of an MLDB, only a small number of layers should be constructed, which should provide maximal benefits to the frequently accessed query patterns. Obviously, the frequently referenced attributes should be preserved in higher layers, and the frequently referenced concept levels should be considered as the candidate concept levels in the construction of higher layers. Steps 2 and 3 are performed in a way similar to the attribute-oriented induction, studied previously [9, 4]. Step 4 constructs a new schema which records a route map and the generalization paths for database browsing and cooperative query answering. \square

Example 4.4 Figure 8 illustrates the route map and the generalization paths of a multiple layered database constructed on top of the University database in Example 2.1.

The schema of the higher layer generalized relations are illustrated in Fig. 9, where *Student'* is a generalized relation with *Sname* (the key) preserved, *Birth_place* removed, *Birth_date* generalized to *Age*, etc.; *Student''* is a generalized relation with the key removed, which summarizes the general relationship among *Status*, *Major*, *Age* and *GPA*; and *SGC'* is obtained by firstly joining three relations, *Student'*, *Grading*, and *Course'*, and then generalizing the joined relation to obtain the general statistics of the performance of a course.

Student' (*Sname*, *Status*, *Sex*, *Major*, *Age*, *GPA*).
Student'' (*Status*, *Major*, *Age*, *GPA*).
Course' (*Cnum*, *Title*, *Semester*, *Department*, *Instructor*, *Time*).
SGC' (*Cnum*, *Semester*, *Department*, *Instructor*, *Grade*).

Figure 9: Schema of the University database

This multiple layered database facilitates the extraction, summarization and comparison of data at a generalized level in intelligent query answering, as illustrated in the next section. \square

5 Intelligent Answering of Data Queries Using Knowledge Discovery Methods

With the availability of knowledge discovery tools, a query can be answered intelligently by interacting query intent analysis and intelligent query answering processes with the major components of a knowledge-rich database as illustrated in Fig. 10.

The intelligent query answering consists of three major processes: *query intent analysis*, *query rewriting*, and *answer transformation and answer explanation*. Query intent analysis analyzes the intent of the query and determines whether it is necessary to provide assisted answers, and if it is, what kind of assistance should be provided. A query rewriting process transforms, when desired, a query into a generalized, specialized or neighborhood rewritten query using associative and neighborhood information. Finally, answer transformation and answer explanation process, when desirable, transforms the answers into a summarized or general form, compares them with a neighborhood query, and explains the answers based on rules and/or generalized knowledge. The analysis and the transformation processes are based on the available or discovered knowledge about databases, queries, and users. Therefore, they interact closely with the major components of the KRDB.

5.1 Analysis of the intent of a query

Several interesting methods for query intent analysis have been developed in the studies of intelligent query answering [14, 15, 19, 7, 26]. Such analyses are based on the notions of generalization, association, aggregation, concept clustering, etc. Semantic data modeling, classification of *topics of interests*, and *plan analysis and formation* are powerful techniques for query intent analysis [15, 19, 7, 26].

When posing a query, different users often have different intentions. For example, when asking the highest monthly balance of an account, a customer and a bank manager are likely to have different intentions. Therefore, an important task of query intent analysis is *user modeling*, which analyzes the user's background and intention and constructs different models for different classes of users.

Moreover, a large volume of knowledge may exist or can be discovered in a database. One may often find that there exist too many "intelligent" ways to associate a query with the available or discoverable database knowledge. It is crucial to have knowledge about user's background and the role that he/she plays in order to understand user's intention, avoid superfluous answers, and provide users with quality assistance.

Since the knowledge-rich database is constructed based on an extended deductive entity-relationship model and enhanced with knowledge discovery components, it provides naturally the above mentioned notions, and thus a powerful support for query intent analysis.

Taking the university database as an example, we ana-

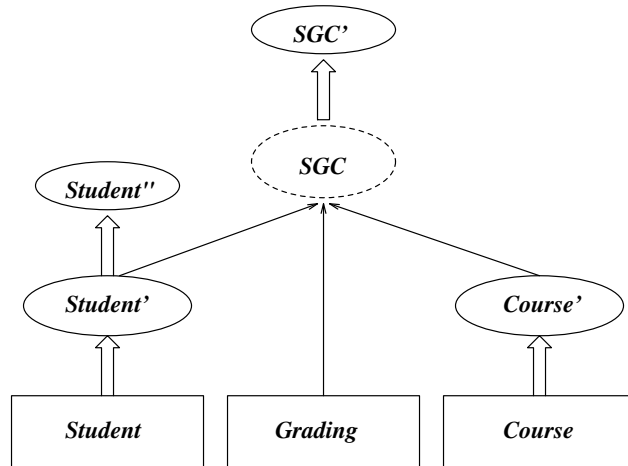


Figure 8: A multiple level database (MLDB) for the University Database.

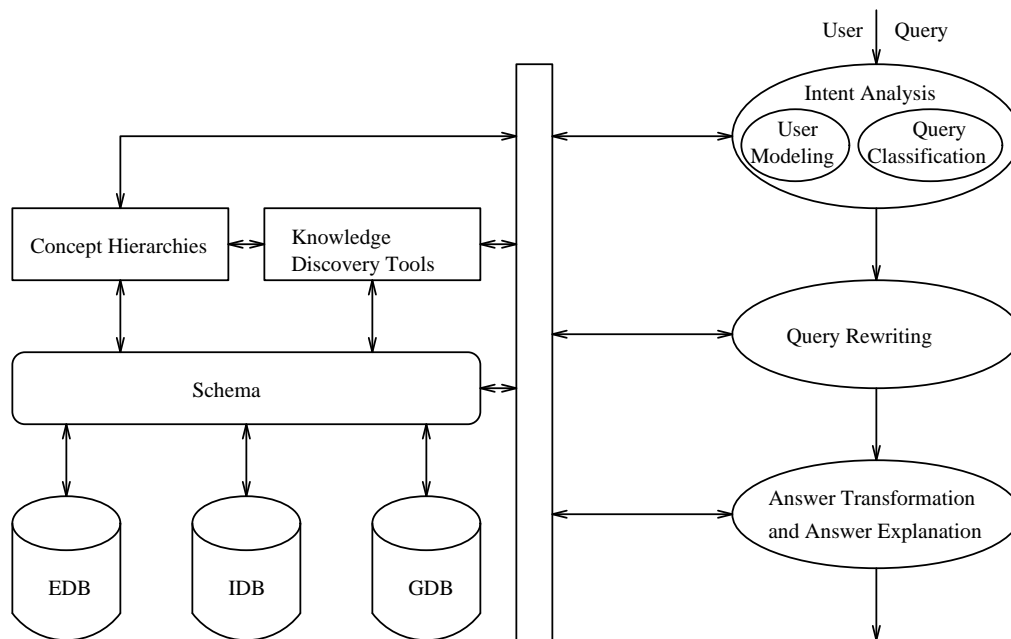


Figure 10: Intelligent query answering under the KRDB model.

lyze how knowledge discovery components may help query intent analysis.

1. User modeling:

Based upon a user's profession/position (e.g., university administrator, professor, student, etc.), security level (e.g., eligibility of accessing some sensitive data, such as a professor's salary), accessing history (e.g., new student, new professor, etc.) or other related information, a user can be associated with a particular user category built in the system. The linkage between a category of users and a class of preferred concepts or queries is constructed by experts in the development of intelligent query answering system, based on the summarization of the accessing history of each class of users, etc. With the available knowledge-rich data model and knowledge discovery components, users can be categorized into some high-level user classes (e.g., graduate program applicants, experienced finance secretary) and be associated with a set of high-level concepts (e.g., the major interests of a new student being expressed at a high concept level) to assist query intent analysis.

2. Query classification:

A query can also be classified into different categories according to the query condition and the information to be inquired. For example, queries on a course plan can be categorized into long term course plan, semester course plan, etc. according to the conditions given in the query (notice that a long term course plan may ignore many levels of details, such as classroom, class time, etc. while a short term one may not) or be categorized into general browsing, detailed examination, or course registration according to query actions. A query class can be linked with certain user categories, generalized concept classes and transformation rules to guide appropriate intelligent query answering for particular classes of queries.

3. Data classification and concept clustering:

Based upon the extended entity-relationship model, data which includes entities, relationships, attributes and specific conditions can be classified and clustered. For example, a set of courses in a particular subarea (e.g., Databases, Graphics, etc.) or at a particular level (e.g., senior, M.Sc., etc.) can be clustered together. The data classification and clustering task is facilitated by the availability of concept hierarchies and knowledge discovery tools.

4. Heuristic rule specification:

A set of heuristic rules can be specified by experts based upon user category, query category, concept hierarchies and the relationships among high-level entities, attributes, and conditions. For example, if a user is in the category of freshman student, the detailed information about course sequence, workload,

general course description could be of a major concern at course registration, etc. Such heuristics can be specified as rules to guide intelligent query answering.

Query intent analysis is performed by systematically applying the above techniques. Furthermore, the constructed models and transformation rules should be testified by experiments and be tuned according to their effectiveness in intelligent query answering and the feedbacks from users. Note that query intent analysis and user modelling can also be enhanced by a graphical use interface component which communicates with users to determine the desired and concrete types of intelligent responses.

5.2 Query rewriting using associated or neighborhood information

Direct data retrieval may not always find enough answers for a user. Furthermore, a user may like to know more information than the direct answers to a query for decision making. Therefore, it is often useful to provide associated or neighborhood answers to a query. Directed by query intent analysis, associated query answering can be performed by (1) *presenting the information about some additional attributes which are not directly inquired but are relevant to the query*; (2) *relaxation of certain query conditions*; and (3) *adding an alternative query which is closely related to the original one* [7].

Let the answer set be viewed as a relation table. The above three mechanisms can find their corresponding relational transformations: *width-extension*, *height-extension*, and *table-extension*.

1. **Width-extension:** The first case (addition of relevant attributes) can be viewed as extension of the width of the answer table by adding some closely related attributes to the table. For example, an inquiry on the course number for a course can be answered by returning the course offering time and location as well.
2. **Height-extension:** The second case (relaxation of certain query conditions) can be viewed as an extension of the height of the table. For example, an inquiry on the information for a particular course can be answered by relaxation of the query condition to provide information about other similar courses as well.
3. **Table-extension:** The third case (answering an alternative query) can be viewed as an extension of the answer table or a switch to a similar table. For example, an inquiry on the available teaching assistant positions can be answered by returning information about research assistant and project assistant positions in the same department (possibly in other relation tables) if the user is a graduate student and the time is at the beginning of a new semester (a job hunting season).

Query rewriting redirects a query according to the intent of the query. The success of query rewriting depends on the query intent analysis and the availability of associated, generalized and neighborhood information which may exist in concept hierarchies or discovered knowledge rules or can be discovered by knowledge discovery tools. Query rewriting can be implemented by mapping query constants to an appropriate level via generalization or specialization hierarchies and mapping a query to a neighborhood one by providing with additional, associated or neighborhood information. The knowledge discovery components, which specify or discover generalization, aggregation, neighborhood, or association relationships among data in the database, provide important assistance in the analysis of query intent and in the rewriting of queries into their alternatives based on hierarchical or neighborhood relationships.

Example 5.1 Suppose a student poses a query about the course numbers of the available database courses for juniors offered by the School of Computer Science in the Fall of 1994 to the University database of Example 2.1. The intelligent query processor will first examine the user model of “student”, which may suggest to extend the “width of the table” by association of the closely related attribute information with the course number. Suppose that a set of attributes associated with the course information are as follows, which can be obtained based upon the deductive entity-relationship model and the clustering of concepts,

$$course_information \left\{ \begin{array}{l} course_number \\ course_name \\ instructor \\ location \\ time \\ course_outline \\ other_info \end{array} \right.$$

For concise presentation of the associative information (the “conciseness” can be determined based upon the schema definition), *course_outline* and *other_info* will not be presented in the “width extension”. Thus the width-extended table will carry the following header,

course_number	course_name	instructor	location	time
---------------	-------------	------------	----------	------

Alternatively, the intelligent query processor may suggest to extend the “height of the table” by association of other closely related courses with the inquired courses. The information about other closely related courses could be obtained by the available conceptual hierarchy of the courses. Suppose that available conceptual hierarchy for courses are as follows.

$$All\ Courses \left\{ \begin{array}{l} Computing\ Science \left\{ \begin{array}{l} Artificial\ Intelligence \{ \dots \\ Database\ Systems \left\{ \begin{array}{l} CMPT-354 \\ CMPT-454 \\ CMPT-459 \end{array} \right. \\ Programming\ Languages \{ \dots \\ Operating\ Systems \{ \dots \\ \dots \end{array} \right. \\ Physics \{ \dots \\ \dots \end{array} \right.$$

Since the lowest conceptual hierarchy node higher than *database systems* is *computing science*, the “height extension” may print the course numbers of the available *computing science* courses for juniors offered by Computing Science in the same semester. Notice that other alternatives exist, such as relaxation of the constraint of *junior* to *undergraduate* using the conceptual hierarchy related to *junior*, etc.

Therefore, the selection of associated additional attributes (as width-extension) or the relaxation of query constraints (as height-extension) can be performed by analyzing the semantics of the query and using the conceptual hierarchy information related to query constraints or semantic structures of the database. □

5.3 Answer transformation and answer explanation

Besides query rewriting, the set of answers may also be transformed, explained, compared or summarized in different ways for intelligent query answering.

5.3.1 Generalization and summarization of answers

A database user may be interested in general descriptions or overall statistics of the answer set but not the detailed answer set itself. Thus, a data query can be answered by generalization and summarization of the answer set, that is, by presenting generalized data only, a combination of generalized and primitive data, or a summarization of concrete answers (possibly together with the presentation of concrete answers) using generalized data and database statistics. Such an **answer transformation** process can be realized by the following techniques.

1. *Presentation of higher level concepts in the answer.* Based upon the analysis of a user model, a query may be answered by presenting higher level concepts in the answer set using the available concept hierarchies or data generalization tools. By association of general information with concrete answers, answers to a query can be presented in a general and concise manner, thus making the implications of the answers better understood.

Example 5.2 A query which inquires about the information of a professor Tom Smith in the University database can be answered as “*Tom Smith is a Computer Science professor (not mentioning the specific fields), born in Europe (not mentioning the specific city, province, and country) in 1940s (not mentioning the specific date and year)*”. This is meaningful if the user (such as a university administrator) is concerned of the general information but not the detailed one. □

2. *Generalization and summarization of the set of answers.* This process can be realized by extraction

of the prime generalized relations and characteristic rules, as described in the last section.

Example 5.3 A query which inquires “*who have good or excellent GPAs in computing science?*” can be answered intelligently in several ways: (1) “*100% graduate students, 55% senior students, and 25% junior students*” (general, statistical information only), (2) “*all of the graduate students and the following undergraduate students . . .*” (a combination of generalized and primitive data), (3) concrete answer (student names) plus a summarization of the answers at a high level, etc. Obviously, the general information can be obtained dynamically using knowledge discovery tools or retrieved directly from the corresponding higher level relation stored in a multiple-level database. □

3. *Lazy evaluation using generalized rules or relations.* As an intelligent query answering mechanism, lazy evaluation presents the deduction or generalized rules related to the query without accessing the database or presenting the full answers set. The detailed and concrete answers are provided only by further requests. Lazy evaluation by deduction rules has been studied in deductive database research [15, 20, 26, 25]. With the availability of multiple layered databases or stored generalized rules, it is interesting to examine how to perform lazy evaluation using generalized rules and multiple layered database.

Example 5.4 Suppose that a user poses a query: “*print the information about the graduate students in computing science*”. Obviously, it is too clumsy to print all such information. A lazy evaluation method can be performed by presenting the generalized rule or the relevant portion of the generalized (prime) relation stored in a high-layered database without evaluation of the query against the database. For example, suppose a database stores a higher layered relation H which describes the general characteristics of all the graduate students in the university. Since the graduate students majoring in Computing Science is a proper subset of all the graduate students in the university, the query-relevant portion should be extracted from the generalized relation H by enforcing a selection condition such as “*major = computing science*”. The extracted generalized relation can be simplified or mapped into feature tables, etc., and can be used as a “lazy” answer to the query, such as, 25% of them has GPAs greater than 3.5, etc. Notice that the processing is performed on a higher layered relation without accessing large primitive-level relations. □

Notice that it is important to perform concept mapping between different levels of data based upon the concept hierarchies. Constants in a query or answers to a query

can be mapped up or down along a concept hierarchy depending on the semantics and the intent of the query. A high-level query can be rewritten into a primitive-level one by mapping the high-level data in the query to a set of primitive data using concept hierarchies. Similarly, a low-level answer set can be transformed into a high-level one by mapping a set of primitive data in the answer set to a set of corresponding high-level ones according to user’s need. The interactions between query conditions and rule bodies (conditions) also need the data/constant mapping among different levels. For example, to examine whether a query is relevant to a certain generalized relation, the query can be restated at the same concept level as that in the rule.

5.3.2 Answer explanation

Another intelligent query answering method is **answer explanation**, which explains the answers to a query by presentation of the associated rules, demonstration of the reasoning process, or illustration of the general information.

The summarization of the statistics of an answer set discussed above can also be employed as a technique for answer explanation. The following example demonstrates that it is often necessary to provide explanations to the answers when the query condition follows or contradicts a rule or an integrity constraint.

Example 5.5 If a query condition follows or contradicts a rule or an integrity constraint, the query can be answered by presentation of the knowledge rules rather than primitive data (a form of lazy evaluation), or by presentation of the rules together with the answer set as a means of explanation.

For example, suppose there is a generalized rule, “*all of the teaching assistants are graduate students*”. The query “*find all of the undergraduate students who are teaching assistants*” can be answered by returning an empty set without accessing extensional database. Moreover, it is cooperative to associate the rule as an explanation.

Similarly, if “*all of the teaching assistants have good or excellent GPAs*” is a generalized rule, the query “*find all of the teaching assistants whose GPAs are greater than 2.5*”, may return “*all of the teaching assistants*”, together with the rule. Specific teaching assistant names are presented only when the user requests for more details.

The above processes can be implemented by testing the query condition against the (generalized) rule for containment or contradiction. If the query passes the test, lazy evaluation can be applied rather than returning detailed answers. □

5.3.3 Answer comparison

Queries can also be answered intelligently by **answer comparison**, which compares or contrasts the general characteristics of its answers with some similar queries. Answer comparison may involve two steps: (i) rewriting a query into a neighborhood query, and (ii) generalization, summarization and comparison of two answer sets, one to the

original query and one to the neighborhood query, at a general level. The first step, rewriting a query into a neighborhood query, can be performed by query intent analysis and substitution of some query constant(s) in the original query by the closest higher level concept(s) using the concept hierarchies. The second step involves learning characteristic and discriminant rules using knowledge discovery tools [9] or MLDBs [12].

Example 5.6 In answering the query, “*find all of the graduate students with excellent GPAs*”, it is interesting to find the undergraduate students with similar characteristics or the graduate students with weaker GPAs and compare the general characteristics and statistics between these answers. Such comparisons may lead to some interesting observations, such as “*36% graduate students vs. only 12.5% undergraduate students have excellent GPAs*”, as shown in Table 7. □

5.4 Layer mapping for intelligent query answering in multiple layered databases

A multiple layered database (MLDB) may facilitate intelligent query answering because some dynamic, relatively costly knowledge discovery process(es) can be saved by storing associated or summarized information in the MLDB.

However, there may arise two new problems in intelligent query answering in MLDBs: (1) how to locate the most appropriate layer in an MLDB in response to a particular query? and (2) how to perform information transformation if the information at a certain layer does not quite match that in the query?

The following method can be applied for layer mapping and information transformation for intelligent query answering.

Algorithm 5.1 *Layer mapping and information transformation for intelligent query answering in multiple layered databases.*

Input. A query and a multiple layered database.

Output. An intelligent answer to the query using the MLDB.

Method.

1. *Mapping the query to an appropriate layer.* Map the constants in the query condition and inquired information to an appropriate layer in the MLDB. The appropriate layer for intelligent query answering should be the lowest layer which covers all the information provided in the query, and each of its concept level is not higher than the information provided in the query. This can be done by marking the attributes in the lowest layer which covers the corresponding attribute concept in the query and finding the lowest layer which covers all the marks. It is possible that such a layer may involve more than one relation.
2. *Transforming the information in the layer for intelligent query answering.* Assume that the information provided in the selected layer is in the form of a prime relation. First, extract the query-relevant information from the layer. Second, generalize the information in the layer so that each attribute resides at the same concept level as those in the query. Notice that if a join needs to be performed on more than one generalized relation, each of the joining relations must be a key-preserving relation for the join attributes [12]. Otherwise, the join has to be performed on its corresponding highest key-preserving layer.
3. *Perform corresponding query answering mechanism based on the user’s requirement and the query intent analysis.* For example, if the intelligent query answering is to compare the answer with the neighborhood information, the obtained generalized tuples are compared with other corresponding tuples at the same layer. □

Rationale of Algorithm 5.1.

Step 1 is to find the most appropriate layer for intelligent query answering. First, a layer must be high enough to cover the concept in the query. However, a too high layer is too general to make the answer intelligent. Thus the appropriate layer should be the lowest layer which covers all the information provided in the query. Step 2 is to extract and transform the information in the layer for intelligent query answering. A generalized relation often covers wider scope than a query does. Thus the query-relevant information should be extracted from the layer. Some information in the layer could be lower than that in the query. Thus generalization should be performed to make each attribute reside at the same concept level as those provided in the query. Moreover, joining on non-key-preserving attributes may lead to information loss [12]. Thus, lossless join should be performed. Step 3 performs appropriate query answering based on our discussion in the previous subsections. □

Example 5.7 Suppose a student is interested in the grade distribution of senior-level science courses. The query is in relevance to the attributes *Cnum*, *Department*, *Grade* in two data relations, *Course* and *Grade*, in the *University* database. Since a higher layered relation *SGC'* (as shown in Example 4.4 and Figure 8) contains the generalization of the join of the three relations, *Course'*, *Grade*, and *Student'*, and each of its concept levels is not higher than those provided in the query, the query can be answered intelligently by examining *SGC'* only. Note that *SGC'* could not have been used if the query were to ask the grade distribution of a particular student since *SGC'* stores only the information at a level higher than that of particular students. Since the concept “science” is more general than individual departments (“physics”, “computer science”, etc.), generalization should be performed on *SGC'*

to generalize the attribute values in Department. Irrelevant attributes such as Semester and Instructor can be removed and identical tuples be merged to derive a generalized relation. If the query is to be answered intelligently by presenting more specific information about the grade distributions, some attributes (such as Semester) or lower level concepts (such as individual departments) may also be retained. More detailed information should be printed by accessing lower layered data only if further requirements are expressed explicitly.

Notice that as indicated in Step 2 of Algorithm 5.1, if a join needs to be performed on a generalized relation at answering a query, each of the join relation must be a key-preserving relation. For example, to answer a query about *course information* and the grade distribution of senior-level science courses, SGC' can still be used since SGC' is a key-preserving relation on the join attribute $Cnum$. However, SGC' cannot be used to answer a query about the grade distribution of *female* students since SGC' is not a key-preserving relation on the join attribute $Sname$. In this case, join should be performed on $Student'$, $Grading$ and $Course'$. \square

6 Discussion and Conclusion

A framework has been presented for intelligent query answering in a knowledge-rich database composed of deductive and knowledge discovery components. The availability of generalized rules, concept hierarchies and knowledge discovery tools greatly enhance the power of intelligent query answering. First, it expands the spectrum of knowledge queries from inquiring deduction rules to inquiring general regularity of data, such as characteristic rules, discriminant rules, data evolution regularities, etc. [9]. Second, it facilitates the query intent analysis since the notions of generalization, aggregation, neighborhood, similarity, etc. can be studied systematically using the generalized knowledge and concept hierarchies. Third, it facilitates intelligent query answering since answers can be presented in general terms, summarized by statistical information, and compared with similar groups of data at a high level. Finally, the intelligent query answering can be implemented efficiently using generalized rules and knowledge discovery tools such as prime relations, feature tables, multiple layered databases, and other implementation techniques.

The enhanced power of intelligent query answering may lead to two problems: *superfluous "intelligent" answers* and the *risk on database security*.

The first problem indicates that one may suffer from obtaining too many superfluous "interesting" answers to a query because there are many ways for a query to be answered intelligently. Techniques should be developed to control the answer generation process in intelligent query answering. In general, one may assume that an appropriate knowledge level is associated with each user. A user usually poses queries at his/her corresponding knowledge level and expects the answers to be presented at the same

level. If the contents of the answer set are not at such level, generalization or specialization should be performed on the answer set as concept level adjustment. Further, with user modeling and query intent analysis, only those answers which match the query intent and the user model will be presented. More desirably, an intelligent query answering process can be triggered or directed by interaction with users. For example, after obtaining the preliminary set of answers to a query, some following-up questions can be raised by users, such as "*in more detail?*", "*in summary?*", "*why?*", "*other options?*", "*comparing with others?*", etc. These questions indicate what kind of intelligent answers are expected. Then the corresponding intelligent query-answering mechanisms can be evoked.

The second problem indicates that with the extended power of intelligent query answering, some sensitive or confidential information could be disclosed inappropriately to someone who should not know it [21]. One technique which may enhance the database security is to associate with a user model certain kinds of constraints on accessing rights. For example, if the user is a student (easily known from the login name), the constraints on intelligent answering of his/her query in a university database will be quite different from the same query posed by a university administrator. Sensitive information will not be disclosed to the users who do not have appropriate access rights. However, because of the power and complexity of deduction and knowledge discovery, it is difficult to tell to what extent that accessing certain piece of information may eventually lead to the disclosure of sensitive information by a sequence of deduction and induction. Therefore, more study should be performed on ensuring database security in intelligent query answering in databases augmented with deduction and knowledge discovery components.

Based upon our previous study on knowledge discovery in databases [4, 9], a prototyped experimental database learning system, **DBLearn**, has been constructed [11]. Based on the principles of attribute-oriented induction, experimentation has been performed for knowledge discovery in large databases, such as the NSERC (*Natural Sciences and Engineering Research Council of Canada*) **Grant Information System**. Our experimental results on direct answering of data and knowledge queries are successful with satisfactory performance. Further studies are being performed on intelligent query answering using the provided concept hierarchies and knowledge discovery tools.

The innovations of this paper can be summarized as follows. First, a knowledge-rich data model is constructed which consists of an extended entity-relationship schema, an extensional database, an intensional database, a set of concept hierarchies, a set of generalized rules, and a set of knowledge discovery tools. Second, a systematic study has been performed on intelligent query answering with the focus on the application of discovered knowledge, concept hierarchies, and knowledge discovery tools to intelligent query answering in database systems. Query answering mechanisms are classified into (1) direct answering of data queries, (2) intelligent answering of data queries, (3)

direct answering of knowledge queries, and (4) intelligent answering of knowledge queries. Third, techniques have been developed for implementation of such mechanisms using discovered knowledge and/or knowledge discovery tools, which include generalization, data summarization, answer transformation, rule discovery, concept clustering, query rewriting, lazy evaluation, etc. The implementation of these methods using multiple layered databases, prime relations and feature tables are investigated.

Our study shows that knowledge discovery substantially broadens the spectrum of intelligent query answering and may have deep implications on query processing in data- and knowledge-base systems. There are many interesting issues which should be studied further. The systematic development of language primitives for intelligent query answering, new techniques for implementation of intelligent query answering mechanisms using knowledge discovery techniques, mechanisms for intelligent answering of knowledge queries, security problems in intelligent query answering, and the integration of different intelligent query answering techniques are interesting topics for future research.

Acknowledgements

The authors are grateful to the anonymous referees for their many valuable comments and suggestions that improved the quality of this paper.

References

- [1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 207–216, Washington, D.C., May 1993.
- [2] A. Borgida and R. J. Brachman. Loading data into description reasoners. In *Proc. 1993 ACM-SIGMOD Int. Conf. Management of Data*, pages 217–226, Washington, D.C., May 1993.
- [3] R. J. Brachman. Viewing data from a knowledge representation lens. In *Proc. Int. Conf. Building and Sharing of Very Large-Scale Knowledge Bases'93*, pages 117–120, Tokyo, Japan, December 1993.
- [4] Y. Cai, N. Cercone, and J. Han. Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 213–228. AAAI/MIT Press, 1991.
- [5] P. Chen. The entity-relationship model : Toward a unified view of data. *ACM Trans. Database Syst.*, 1:9–36, 1976.
- [6] W. W. Chu and Q. Chen. Neighborhood and associative query answering. *Journal of Intelligent Information Systems*, 1:355–382, 1992.
- [7] F. Cuppens and R. Demolombe. Cooperative answering: A methodology to provide intelligent access to databases. In *Proc. 2nd Int. Conf. Expert Database Systems*, pages 621–643, Fairfax, VA, April 1988.
- [8] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases: An overview. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 1–27. AAAI/MIT Press, 1991.
- [9] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.
- [10] J. Han and Y. Fu. Dynamic generation and refinement of concept hierarchies for knowledge discovery in databases. In *Proc. AAAI'94 Workshop on Knowledge Discovery in Databases (KDD'94)*, pages 157–168, Seattle, WA, July 1994.
- [11] J. Han and Y. Fu. Exploration of the power of attribute-oriented induction in data mining. In U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 399–421. AAAI/MIT Press, 1996.
- [12] J. Han, Y. Fu, and R. Ng. Cooperative query answering using multiple-layered databases. In *Proc. 2nd Int. Conf. Cooperative Information Systems*, pages 47–58, Toronto, Canada, May 1994.
- [13] J. Han and Z. N. Li. Deductive-ER: Deductive entity-relationship model and its data language. *Information and Software Technology*, 34:192–204, 1992.
- [14] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Comput. Surv.*, 19:201–260, 1987.
- [15] T. Imielinski. Intelligent query answering in rule based systems. *J. Logic Programming*, 4:229–257, 1987.
- [16] K. A. Kaufman, R. S. Michalski, and L. Kerschberg. Mining for knowledge in databases: Goals and general description of the INLEN system. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 449–462. AAAI/MIT Press, 1991.
- [17] R. S. Michalski. A theory and methodology of inductive learning. In Michalski et al., editor, *Machine Learning: An Artificial Intelligence Approach, Vol. 1*, pages 83–134. Morgan Kaufmann, 1983.
- [18] R. S. Michalski, L. Kerschberg, K. A. Kaufman, and J.S. Ribeiro. Mining for knowledge in databases: The INLEN architecture, initial implementation and first results. *J. Int. Info. Systems*, 1:85–114, 1992.
- [19] A. Motro. Using integrity constraints to provide intensional responses to relational queries. In *Proc. 15th Int. Conf. Very Large Data Bases*, pages 237–246, Amsterdam, Netherlands, Aug. 1989.
- [20] A. Motro and Q. Yuan. Querying database knowledge. In *Proc. 1990 ACM-SIGMOD Int. Conf. Management of Data*, pages 173–183, Atlantic City, NJ, June 1990.
- [21] D. E. O'Leary. Knowledge discovery as a threat to database security. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 507–516. AAAI/MIT Press, 1991.
- [22] G. Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 229–238. AAAI/MIT Press, 1991.

- [23] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.
- [24] G. Piatetsky-Shapiro and C.J. Matheus. Knowledge discovery workbench for exploring business databases. *Int. J. Intell. Syst.*, 7:675–686, 1992.
- [25] A. Pirotte and D. Roelants. Constraints for improving the generation of intensional answers in a deductive database. In *Proc. 5th Int. Conf. Data Engineering*, pages 652–659, Los Angeles, CA, Feb. 1989.
- [26] C-D. Shum and R. Muntz. Implicit representation for extensional answers. In *Proc. 2nd Int. Conf. Expert Database Systems*, pages 497–522, Vienna, VA, April 1988.
- [27] A. Silberschatz, M. Stonebraker, and J. D. Ullman. Database systems: Achievements and opportunities. *Comm. ACM*, 34:94–109, 1991.
- [28] T. J. Teorey, D. Yang, and J. P. Fry. A logical design methodology for relational databases using the extended entity-relationship model. *ACM Comput. Surv.*, 18:197–222, 1986.
- [29] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Vols. 1 & 2*. Computer Science Press, 1989.
- [30] C. T. Yu and W. Sun. Automatic knowledge acquisition and maintenance for semantic query optimization. *IEEE Trans. Knowledge and Data Engineering*, 1:362–375, 1989.
- [31] W. Ziarko, R. Golan, and D. Edwards. An application of Datalogic/R knowledge discovery tool to identify strong predictive rules in stock market data. In *Proc. AAAI-93 Workshop on Knowledge Discovery in Databases*, pages 93–101, Washington DC, July 1993.
- [32] J. Zytkow and J. Baker. Interactive mining of regularities in databases. In G. Piatetsky-Shapiro and W. J. Frawley, editors, *Knowledge Discovery in Databases*, pages 31–54. AAAI/MIT Press, 1991.