International Journal of Data Science and Analytics

FACTORBASE: Multi-Relational Structure Learning with SQL All The Way

--Manuscript Draft--

Manuscript Number:	
Full Title:	FACTORBASE: Multi-Relational Structure Learning with SQL All The Way
Article Type:	Regular Article
Funding Information:	Canadian Network for Research and Innovation in Machining Technology, Natural Sciences and Engineering Research Council of Canada — Dr. Oliver Schulte China Scholarship Council (CN) — Dr. Zhensong Qian
Abstract:	We describe FactorBase, a new SQL-based framework that leverages a relational database management system to support multi-relational model discovery. A multi-relational statistical model provides an integrated analysis of the heterogeneous and interdependent data resources in the database. We adopt the BayesStore design philosophy: statistical models are stored and managed as first-class citizens inside a database. Whereas previous systems like BayesStore support multi-relational inference, FactorBase supports multi-relational learning. A case study on six benchmark databases evaluates how our system supports a challenging machine learning application, namely learning a first-order Bayesian network model for an entire database. Model learning in this setting has to examine a large number of potential statistical associations across data tables. Our implementation shows how the SQL constructs in FactorBase facilitate the fast, modular, and reliable development of highly scalable model learning systems.
Corresponding Author:	zhensong qian Simon Fraser University CANADA
Corresponding Author Secondary Information:	
Corresponding Author's Institution:	Simon Fraser University
Corresponding Author's Secondary Institution:	
First Author:	Oliver Schulte
First Author Secondary Information:	
Order of Authors:	Oliver Schulte Zhensong Qian
Order of Authors Secondary Information:	
Author Comments:	We got an exclusive invitation from Dr. Longbing Cao, the Editor-in-Chief of the Int. J. Data Science and Analytics with Springer, to submit an extended version of our DSAA'15 long paper: Zhensong Qian and Oliver Schulte FACTORBASE: Multi-Relational Structure Learning with SQL All The Way. Compared to the conference version, the current submission is almost twice the length due to adding two sections.
Suggested Reviewers:	Chris Mayfield Assistant Professor cmayfiel@cs.purdue.edu He is working in the related area and I attached one of his recent papers. "ERACER: A Database Approach for Statistical Inference andDataCleaning"https://www.cs.purdue.edu/homes/neville/papers/mayfield-

	sigmod2010.pdf
	Jennifer Neville neville@cs.purdue.edu She is working in the related area and I attached one of her recent papers. "ERACER: A Database Approach for Statistical Inference andDataCleaning"https://www.cs.purdue.edu/homes/neville/papers/mayfield-sigmod2010.pdf
	Jan Motl jan.motl@fit.cvut.cz He is working in the related area. https://datalab.fit.cvut.cz/people/55-jan-motl
	Samuel Madden madden@csail.mit.edu He is working in the related area and I attached one of his recent papers. "Demonstration of ModelDB: a system for managing machine learning models" https://goo.gl/QD4ExV

Int. J. Data Science and Analytics manuscript No.
(will be inserted by the editor)

FACTORBASE: Multi-Relational Structure Learning with SQL All The Way

Oliver Schulte · **Zhensong Qian**

Received: date / Accepted: date

Abstract We describe FACTORBASE , a new SQL-based framework that leverages a relational database management system to support multi-relational model discovery. A multi-relational statistical model provides an integrated analysis of the heterogeneous and interdependent data resources in the database. We adopt the BayesStore design philosophy: statistical models are stored and managed as first-class citizens inside a database [34]. Whereas previous systems like BayesStore support multi-relational inference, FACTORBASE supports multi-relational learning. A case study on six benchmark databases evaluates how our system supports a challenging machine learning application, namely learning a first-order Bayesian network model for an entire database. Model learning in this setting has to examine a large number of potential statistical associations across data tables. Our implementation shows how the SQL constructs in FACTORBASE facilitate the fast, modular, and reliable development of highly scalable model learning systems.

Keywords Relational Learning · Bayesian Networks · Model Selection

1 Introduction

Data science brings together ideas from different fields for extracting value from large complex datasets. The system described in this paper combines advanced analytics from multi-relational or *statistical-relational* machine learning (SRL) with database systems. The power of combining machine learning with database systems has been demonstrated in several systems [13,18,2]. The novel contribution of FACTORBASE is supporting machine learning for *multi-relational* data, rather than for traditional learning where the data are represented in a *single* table or data matrix. We discuss new challenges raised by multi-relational model learning compared to single-table learning, and how FACTORBASE solves them using the resources of SQL (Structured Query Language). The name FACTORBASE indicates that our system supports learning factors that define a log-linear multi-relational model [17]. Supported new database services include constructing, storing, and transforming complex statistical objects, such as factor-tables, cross-table sufficient statistics, parameter estimates, and model selection scores.

Multi-relational data have a complex structure, that integrates heterogeneous information about different types of entities (customers, products, factories etc.) and different types of relationships among these entities. A statistical-relational model provides an integrated statistical analysis of the heterogeneous and interdependent complex data resources maintained by the database system. Statistical-relational models have achieved state-of-the-art performance in a number of application domains, such as natural language processing, ontology matching, information extraction, entity resolution, link-based clustering, query optimization, etc. [3, 23,10]. Database researchers have noted the usefulness of statistical-relational models for knowledge discovery and for representing uncertainty in databases [31,34]. They have developed a system architecture where statistical models are stored as first-class citizens *inside a database.* The goal is to seamlessly integrate query processing and statistical-relational inference. These systems focus on inference *given* a statistical-relational model, not on *learning* the model from the data

Oliver Schulte
Simon Fraser University, Canada
Tel.: +1-778-782-3390
Fax: +1-778-782-3045
E-mail: oschulte@sfu.ca

Zhensong Qian
Simon Fraser University, Canada
Tel.: +1-778-782-7008
Fax: +1-778-782-3045
E-mail: zqian@sfu.ca

stored in the RDBMS. FACTORBASE complements the in-database probabilistic inference systems by providing an in-database probabilistic model learning system.

1.1 Evaluation

We evaluate our approach on six benchmark databases. For each benchmark database, the system applies a state-of-the-art SRL algorithm to construct a statistical-relational model. Our experiments show that FACTORBASE pushes the scalability boundary: Learning scales to databases with over 10^6 records, compared to less than 10^5 for previous systems. At the same time it is able to discover more complex cross-table correlations than previous SRL systems. We report experiments that focus on two key services for an SRL client: (1) Computing and caching sufficient statistics, (2) computing model predictions on test instances. For the largest benchmark database, our system handles 15M sufficient statistics. SQL facilitates block-prediction for a set of test instances, which leads to a 10 to 100-fold speedup compared to a simple loop over test instances.

1.2 Contributions

FACTORBASE is the first system that leverages relational query processing for learning a multi-relational log-linear graphical model. Whereas the in-database design philosophy has been previously used for multi-relational inference, we are the first to adapt it for multi-relational model structure learning. Pushing the graphical model inside the database allows us to *use SQL as a high-level scripting language for SRL*, with the following advantages.

1. Extensibility and modularity, which support rapid prototyping. SRL algorithm development can focus on statistical issues and rely on a RDBMS for data access and model management.
2. Increased scalability, in terms of both the size and the complexity of the statistical objects that can be handled.
3. Generality and portability: standardized database operations support "out-of-the-box" learning with a minimal need for user configuration.

A previous version of this paper appeared in DSAA '15. Compared to the conference version, the current submission is almost twice the length due to adding two sections: one describing how the model manager supports structure learning, and an appendix with the details of how the random variable database is constructed using metadata from the SQL system catalog.

1.3 Paper Organization

We provide an overview of the system components and flow. For each component, we describe how the component is constructed and managed inside an RDBMS using SQL scripts and the SQL view mechanism. We show how the system manages sufficient statistics and test instance predictions. The evaluation section demonstrates the scalability advantages of in-database processing. The intersection of machine learning and database management has become a densely researched area, so we end with an extensive discussion of related work.

2 Background on Statistical-Relational Learning

We review enough background from statistical-relational models and structure learning to motivate our system design. The extensive survey by Kimmig *et al.* [17] provides further details. The survey shows that SRL models can be viewed as log-linear models based on par-factors, as follows.

2.1 Log-linear Template Models for Relational Data

Par-factor stands for "parametrized factor". A par factor represents an interaction among parametrized random variables, or par-RVs for short. We employ the following notation for par-RVs [17, 2.2.5]. Constants are expressed in lower-case, e.g. *joe*, and are used to represent entities. A type is associated with each entity, e.g. *joe* is a person. A first-order variable is also typed, e.g. *Person* denotes some member of the class of persons. A functor maps a tuples of entities to a value. We assume that the range of possible values is finite. An *atom* is an expression of the form $r(\tau_1,\ldots,\tau_a)$ where each τ_i is either a constant or a first-order variable. If all of $\tau_1,\ldots,\tau_a$ are constants, $r(\tau_1,\ldots,\tau_a)$ is a *ground atom* or random variable (RV), otherwise a *first-order atom* or a **par-RV**. A par-RV is instantiated to an RV by grounding, i.e. substituting a constant of the appropriate domain for each first-order variable.

A **par-factor** is a pair $\Phi = (\mathsf{A}, \phi)$, where A is a set of par-RVs, and ϕ is a function from the values of the par-RVs to the non-negative real numbers.[1] Intuitively, a grounding of a par-factor represents a set of random variables that interact with each other locally. SRL models use *parameter tying*, meaning that if two groundings of the same par-factor are assigned the same values, they return the same factor value. A set of parfactors $\mathscr{F}$ defines a joint probability distribution over the ground par-RVs as follows. Let $\mathscr{I}(\Phi_i)$ denote the set of *all* ground par-RVs in par-factor Φ_i. Let $\mathbf{x}$ be a joint assignment of values to all ground random variables. Notice that this assignment determines the values of

[1] A par-factor can also include constraints on possible groundings.

all ground atoms. An assignment $\mathbf{X} = \mathbf{x}$ is therefore *equivalent to a single database instance*. The probability of a database instance is given by the log-linear equation [17, Eq.7]:

$$P(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} \prod_{\Phi_i \in \mathscr{F}} \prod_{\mathbf{A} \in \mathscr{I}(\Phi_i)} \phi_i(\mathbf{x}_\mathbf{A}) \quad (1)$$

where $\mathbf{x}_\mathbf{A}$ represents the values of those variables in $\mathbf{A}$ that are necessary to compute ϕ_i. Equation 1 can be evaluated without enumerating the ground par-factors, as follows.

1. For each par-factor, for each possible assignment of values, find the number of ground factors with that assignment of values.
2. Raise the factor value for that assignment to the number of ground factors that instantiate it. The number of ground factors with the same assignment of values is known as a **sufficient statistic**.
3. Multiply together the exponentiated factor values.

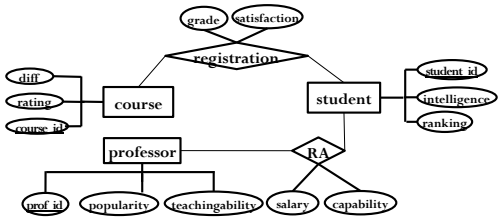

Fig. 1 A relational ER design for a university domain.

Table 1 Tables in an example database instance

Student

s_id	intelligence	ranking
Jack	3	1
Kim	2	1
Paul	1	2

Professor

p_id	popularity	teachingability
Jim	2	1
Oliver	3	1
David	2	2

RA

s_id	p_id	salary	capability
Jack	Oliver	High	3
Kim	Oliver	Low	1
Paul	Jim	Med	2
Kim	David	High	2

2.2 Examples

SRL has developed a number of formalisms for describing par-factors [17]. First-order probabilistic graphical models

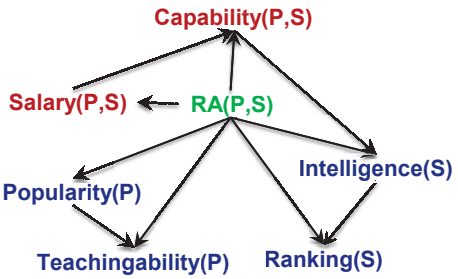

Fig. 2 Bayesian network for the University domain. We omit the *Registered* relationship for simplicity. The network was learned from the University dataset [27].

Table 2 Conditional Probability table $Capability(\mathbb{P},\mathbb{S})\_CPT$, for the node $Capability(\mathbb{P},\mathbb{S})$. Only value combinations that occur in the data are shown. This is an example of a factor table. Both CP and CT tables are stored in an RDBMS.

Capa(P,S)	RA(P,S)	Salary(P,S)	CP
n/a	F	n/a	1
4	T	high	0.45
5	T	high	0.36
3	T	high	0.18
3	T	low	0.2
2	T	low	0.4
1	T	low	0.4
2	T	med	0.22
3	T	med	0.44
1	T	med	0.33

Table 3 Contingency Table $Capability(\mathbb{P},\mathbb{S})\_CT$ for the node $Capability(\mathbb{P},\mathbb{S})$ and its parents. Both CP and CT tables are stored in an RDBMS.

Count	Capa(P,S)	RA(P,S)	Salary(P,S)
203	n/a	F	n/a
5	4	T	high
4	5	T	high
2	3	T	high
1	3	T	low
2	2	T	low
2	1	T	low
2	2	T	med
4	3	T	med
3	1	T	med

are popular both within SRL and the database community [17,34]. The model structure is defined by edges connecting par-RVs. For instance, a **parametrized Bayesian network structure** is a directed acyclic graph whose nodes are par-RVs. Figure 2 shows a Bayesian network for a University domain.

We use the university example as a toy running example throughout the paper. The schema for the university domain is given in Figure 1. This schema features only one relationship for simplicity; FACTORBASE learns a model for any number of relationships. While we describe FACTOR-

BASE abstractly in terms of par-factors, for concreteness we illustrate it using Bayesian networks. The system takes as input a database instance like that shown in Table 1, and produces as output a graphical model like that shown in Figure 2.

A par-factor in a Bayesian network is associated with a **family** of nodes [17, Sec.2.2.1]. A family of nodes comprises a child node and all of its parents. For example, in the BN of Figure 2, one of the par-factors is associated with the par-RV set $\mathsf{A} = \{\mathit{Capability}(\mathbb{P},\mathbb{S}), \mathit{Salary}(\mathbb{P},\mathbb{S}), \mathit{RA}(\mathbb{P},\mathbb{S})\}$. For the database instance of Table **??**, Table **??** and Table **??**, there are $3 \times 3 = 9$ possible ground factors associated with this par-RV, corresponding to the Cartesian product of 3 professors and 3 students. The value of the factor ϕ is a function from an assignment of family node values to a non-negative real number. *In a Bayesian network, the factor value represents the conditional probability of the child node value given its parent node values.* These conditional probabilities are typically stored in a table as shown in Table 2. This table represents therefore the function ϕ associated with the family par-factor. Assuming that all par-RVs have finite domains, a factor can always be represented by a **factor table** of the form Table 2: there is a column for each par-RV in the factor, each row specifies a joint assignment of values to a par-RV, and the factor column gives the value of the factor for that assignment (cf. [17, Sec.2.2.1]).

To evaluate a joint probability $P(\mathbf{X} = \mathbf{x})$ over all ground par-RVs using Equation 1, we must obtain the sufficient statistics: count the number of times that each row in the CP-table is instantiated in the joint assignment $\mathbf{X} = \mathbf{x}$. The sufficient statistics for the $\mathit{Capability}(\mathbb{P},\mathbb{S})$ family can be represented in a contingency table as shown in Table 3. For example, the first row of the contingency table indicates that the conjunction

$$\mathit{Capability}(\mathbb{P},\mathbb{S}) = n/a, \mathit{Salary}(\mathbb{P},\mathbb{S}) = n/a, \mathit{RA}(\mathbb{P},\mathbb{S}) = \mathrm{F}$$

is instantiated 203 times in the University database (publicly available at [27]). This means that for 203 professor-student pairs, the professor did not employ the student as an RA (and therefore the salary and capability of this RA relationship is undefined or n/a).

2.3 SRL Structure Learning

Algorithm 1 shows the generic format of a statistical-relational structure learning algorithm (adapted from [17]). The instantiation of procedures in lines 2, 3, 5 and 8 determines the exact behavior of a specific learning algorithm. The structure algorithm carries out a local search in the hypothesis space of graphical relational models. A set of candidates is generated based on the current model (line 3), typically using a search heuristic. For each candidate model, parameter values are estimated that maximize a model selection score function chosen by the user (line 5). A model selection score is computed for each model given the parameter values, and the best-scoring candidate model is selected (line 7). We next discuss our system design and how it supports model discovery algorithms that follow the outline of Algorithm 1. Figure 3 outlines the system components and dependencies among them.

Algorithm 1: Structure learning algorithm template

Input: Hypothesis space $\mathscr{H}$ (describing graphical models), training data $\mathscr{D}$ (assignments to random variables), scoring function score $(\cdot, \mathscr{D})$
Output: A graph structure G representing par-factors.

1: $G \leftarrow \emptyset$
2: **while** CONTINUE(G, $\mathscr{H}$, score $(\cdot, \mathscr{D})$) **do**
3: $\quad\mathscr{R} \leftarrow$ REFINECANDIDATES($G, \mathscr{H}$)
4: $\quad$**for each** $R \in \mathscr{R}$ **do**
5: $\quad\quad R \leftarrow$ LEARNPARAMETERS(R,score $(\cdot, \mathscr{D})$)
6: $\quad$**end for**
7: $\quad G \leftarrow \mathrm{argmax}_{G' \in \mathscr{R} \cup \{G\}}$ score(G', $\mathscr{D}$)
8: **end while**
9: **return** G

3 The Random Variable Database

Statistical-relational learning requires various metadata about the par-RVs in the model. These include the following.

Domain the set of possible values of the par-RV.
Types Pointers to the first-order variables in the par-RV.
Data Link Pointers to the table and/or column in the input database associated with the par-RV.

The metadata must be machine-readable. Following the in-database design philosophy, we store the metadata in tables so that an SRL algorithm can query it using SQL. The schema analyzer uses an SQL script that queries key constraints in the system catalog database and *automatically* converts them into metadata stored in the random variable database *VDB*. In contrast, existing SRL systems require users to specify information about par-RVs and associated types. Thus FACTORBASE utilizes the data description resources of SQL to facilitate the "setup task" for relational learning [33]. We illustrate the general principles with the entity-relationship (ER) diagram of the University domain (Figure 1). A full description is provided in the Appendix 12.

The translation of an ER diagram into a set of functors converts each element of the diagram into a functor, except for entity sets and key fields [12]. Table 4 illustrates this translation. In terms of database tables, attribute par-RVs correspond to *columns*. Relationship par-RVs correspond to *tables*, not columns. Including a relationship par-RV in a

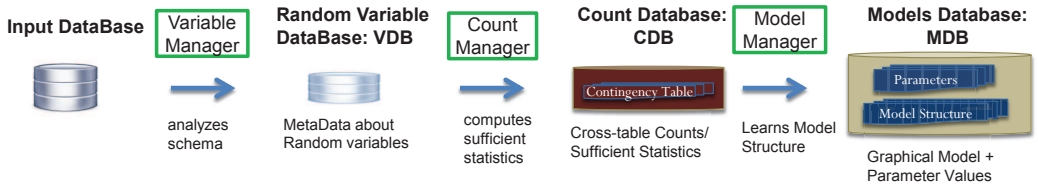

Fig. 3 System Flow. All statistical objects are stored as first-class citizens in a DBMS. Objects on the left of an arrow are utilized for constructing objects on the right. Statistical objects are constructed and managed by different modules, shown as boxes.

Table 4 Translation from ER Diagram to Par-RVs

ER Diagram	Example	par-RV equivalent
Entity Set	Student, Course	$\mathbb{S},\mathbb{C}$
Relationship Set	RA	RA($\mathbb{P}$,$\mathbb{S}$)
Entity Attributes	intelligence, ranking	Intelligence($\mathbb{S}$), Ranking($\mathbb{S}$)
Relationship Attributes	capability, salary	Capability($\mathbb{P},\mathbb{S}$), Salary($\mathbb{P},\mathbb{S}$)

statistical model allows the model to represent uncertainty about whether or not a relationship exists between two entities [17]. The values of descriptive attributes of relationships are undefined for entities that are not related. We represent this by introducing a new constant n/a in the domain of a relationship attribute [21]; see Table **??**. Table 5 shows the schema for some of the tables that store metadata for each relationship par-RV, as follows. par-RV and FO-Var are custom types.

Relationship The associated input data table.
Relationship_Attributes Descriptive attributes associated with the relationship and with the entities involved.
Relationship_FOVariables The first-order variables contained in each relationship par-RV.[2]

Table 5 Selected Tables In the Variable Database Schema.

Table Name	Column Names
Relationship	RVarID: par-RV, TABLE_NAME: string
Relationship_Attributes	RVarID: par-RV, AVarID: par-RV, FO-ID: FO-Var
Relationship_FOvariables	RVarID: par-RV, FO-ID: FO-Var, TABLE_NAME: string

While we have described constructing the variable database for an ER model, different structured data models can be represented by an appropriate first-order logic vocabulary [17], that is, an appropriate choice of functors. For example, in a star schema, facts can be represented in the form $f(\mathbb{D}_1,\ldots,\mathbb{D}_k)$, where the first-order variable $\mathbb{D}_i$ ranges over the primary key of dimension table i. Attributes of dimension i can be represented by a unary functor $a(\mathbb{D}_i)$. FACTORBASE can perform structure learning for different data models after the corresponding data format has been translated into the *VDB* format.

[2] The schema assumes that all relationships are binary.

Table 6 The metadata about attributes represented in *VDB* database tables. The table *Domain* lists the domain for each functor. The table *AttributeColumns* specifies which tables and columns contain the functor values observed in the data. The column name is also the functor ID.

Domain

Column_Name	Value
capability	1
capability	2
capability	3
capability	n/a
diff	1
diff	2
grade	1
grade	2
grade	3
grade	n/a

AttributeColumns

Table_Name	Column_Name
course	diff
course	rating
prof	popularity
prof	teachingability
RA	capability
RA	salary
registration	grade
registration	sat
student	intelligence
student	ranking

4 The Count Manager

The **count database** *CDB* stores a set of *contingency tables*. Contingency tables represent sufficient statistics as follows [22]. Consider a fixed list of par-RVs. A **query** is a set of $(variable = value)$ pairs where each value is of a valid type for the variable. The **result set** of a query in a database $\mathscr{D}$ is the set of instantiations of the logical variables such that the query evaluates as true in $\mathscr{D}$. For example, in the database of Table 1, the result set for the query $RA(\mathbb{P},\mathbb{S}) = \mathrm{T}$, $Capability(\mathbb{P},\mathbb{S}) = 3$, $Salary(\mathbb{P},\mathbb{S}) = high$ is the singleton $\{\langle jack, oliver\rangle\}$. The **count** of a query is the cardinality of its result set.

Every set of par-RVs $\mathbf{V} \equiv \{V_1,\ldots,V_n\}$ has an associated **contingency table** (CT) denoted by $CT(\mathbf{V})$. This is a table with a row for each of the possible assignments of values to

the variables in $\mathbf{V}$, and a special integer column called *count*. The value of the *count* column in a row corresponding to $V_1 = v_1, \ldots, V_n = v_n$ records the count of the corresponding query. Table 2 shows a contingency table for the par-RVs $RA(\mathbb{P},\mathbb{S})$, $Capability(\mathbb{P},\mathbb{S})$, $Salary(\mathbb{P},\mathbb{S})$. The **contingency table problem** is to compute a contingency table for par-RVs $\mathbf{V}$ and an input database $\mathscr{D}$.

SQL Implementation With Metaqueries. We describe how the contingency table problem can be solved using SQL. This is relatively easy for a *fixed* set of par-RVs; the challenge is a general construction that works for different sets of par-RVs. For a fixed set, a contingency table can be computed by an SQL count(*) query of the form

```
CREATE VIEW CT-table(<VARIABLE-LIST>) AS
SELECT COUNT(*) AS count, <VARIABLE-LIST>
FROM TABLE-LIST
GROUP BY VARIABLE-LIST
WHERE <Join-Conditions>
```

FACTORBASE uses SQL itself to construct the count-conjunction query. We refer to this construction as an SQL **metaquery**. We represent a count(*) query in four kinds of tables: the Select, From, Where and Group By tables. Each of these tables lists the entries in the corresponding count(*) query part. Given the four metaquery tables, the corresponding SQL count(*) query can be easily constructed and executed in an application to construct the contingency table. Given a list of par-RVs as input, the metaquery tables are constructed as follows from the metadata in the database *VDB*.

FROM LIST Find the tables referenced by the par-RV's. A par-RV references the entity tables associated with its first-order variables (see VDB.Relationship_FOvariables). Relational par-RV's also reference the associated relationship table (see VDB.Relationship).

WHERE LIST Add join conditions on the matching primary keys of the referenced tables in the WHERE clause. The primary key columns are recorded in VDB.

SELECT LIST For each attribute par-RV, find the corresponding column name in the original database (see VDB.AttributeColur Rename the column with the ID of the par-RV. Add a *count* column.

GROUP BY LIST The entries of the Group By table are the same as in the Select table without the *count* column.

Figure 4 shows an example of a metaquery for the university database. This metaquery defines a view that in turn defines a contingency table for the random variable list associated with the relationship table *RA*. This list includes the entity attributes of professors and of students, as well as the relationship attributes of the *RA* relationship. The Bayesian network of Figure 2 was learned from this contingency table. The contingency table defined by the metaquery of Figure 4 contains only rows where the value of *RA* is true. The Möbius Virtual Join [28] can be used to extend this contingency table to include counts for when *RA* is false, like the table shown in Table 3.

Metaqueries	Entries

CREATE VIEW Select_List AS
SELECT RVarID, CONCAT('COUNT(*)',' as "count"') AS Entries
FROM VDB.Relationship
UNION DISTINCT
SELECT RVarID, AVarID AS Entries
FROM VDB.Relationship_Attributes;
	COUNT(*) as "count"
	`Popularity(P)`
	`Teachingability(P)`
	`Intelligence(S)`
	`Ranking(S)`

CREATE VIEW From_List AS SELECT RVarID, CONCAT('@database@.',TABLE_NAME) AS Entries FROM VDB.Relationship_FOvariables UNION DISTINCT SELECT RVarID, CONCAT('@database@.',TABLE_NAME) AS Entries FROM VDB.Relationship;	@database@.prof AS P @database@.student AS S @database@.RA AS `RA`
CREATE VIEW Where_List AS SELECT RVarID, CONCAT(RVarID,'.',COLUMN_NAME,' = ', FO-ID,'.', REFERENCED_COLUMN_NAME) AS Entries FROM VDB.Relationship_FOvariables;	`RA`.p_id = P.p_id `RA`.s_id =S.s_id

Fig. 4 Example of metaquery results based on university database and the par-RV metadata (Table 5).

5 The Model Manager: Parameter Learning

The Model Manager provides three key services for statistical-relational structure learning. In terms of Algorithm 1:

1. Estimating and storing parameter values (line 5).
2. Computing one or more model selection scores (line 7).
3. Generating, scoring, and storing candidate model structures (line 3).

FACTORBASE uses a *store+score* design for these services, which is illustrated in Figure 5. A **model structure table** represents a candidate model. When a candidate model structure is inserted, a view uses the sufficient statistics from a contingency table to compute a table of parameter values. Another view uses the parameter values and sufficient statistics together to compute the score for the candidate model.

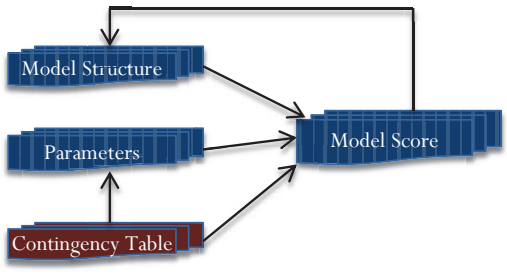

Fig. 5 Dependencies Among Key Components of the Model Manager.

5.1 *MDB tables for Parameter Learning*

The relational schema for the Parameter Manager is shown in Table 7. The @par-RVID@ parameter refers to the ID of a par-RV, for instance *Capability*$(\mathbb{P},\mathbb{S})$. The model manager stores a set of factor tables (cf. Section 2.2). In a graphical model, each factor is defined by the local topology of the model template graph. For concreteness, we illustrate how factor tables can be represented for Bayesian networks. The graph structure can be stored straightforwardly in a database table *BayesNet* whose columns are *child* and *parent*. The table entries are the IDs of par-RVs. For each node, the *MDB* manages a conditional probability table. This is a factor table that represents the factor associated with the node's family (see Table 2). In a Bayesian network, model selection scores are decomposable. This means that there is a local score associated with each family, such that the total score for the BN model is the sum of the local scores. For each family, the local score is stored in the *Scores* table indexed by the family's child node.

5.2 Parameter Manager

Deriving predictions from a model requires estimating values for its parameters. Maximizing the data likelihood is the basic parameter estimation method for Bayesian networks. The maximum likelihood estimates equal the observed frequency of a child value given its parent values.

SQL Implementation With Natural Join. Given the sufficient statistics in a contingency table, a conditional probability table containing the maximum likelihood estimates can be computed by aggregation using SQL as in the example below.

```
SELECT count/temp.parent_count as CP,
Capability(P,S), RA(P,S), Salary(P,S)
FROM Capability(P,S)_CT
NATURAL JOIN
(SELECT sum(Count) as parent_count,
RA(P,S), Salary(P,S)
FROM Capability(P,S)_CT
GROUP BY  RA(P,S), Salary(P,S) ) as temp
```

5.3 Model Score Computation

A typical model selection approach is to maximize the likelihood of the data, balanced by a penalty term. For instance, the Akaike Information Criterion (AIC) is defined as follows

$$AIC(G,\mathscr{D}) \equiv ln(P_{\widehat{G}}(\mathscr{D})) - \#par(G)$$

where $\widehat{G}$ is the BN G with its parameters instantiated to be the maximum likelihood estimates given the database $\mathscr{D}$, and $\#par(G)$ is the number of free parameters in the structure G. The number of free parameters for a node is the product of (the possible values for the parent nodes) $\times$ (the number of the possible values for the child node -1). Given the likelihood and the number of parameters, the AIC column is computed as $AIC = loglikelihood - \#par$. Model selection scores other than AIC can be computed in a similar way given the model likelihood and number of parameters.

5.3.1 Parameter Number Computation

To determine the number of parameters of the child node @parVar-ID@, the number of possible child and parent values can be found from the *VDB.Domain* table in the Random Variable Database.

5.3.2 Likelihood Computation

As explained in Section 2.1, the log-likelihood can be computed by multiplying the instantiation counts of a factor by its value. Assuming that instantiation counts are represented in a contingency table and factor values in a factor table, this multiplication can be elegantly performed using the Natural Join operator. For instance, the log-likelihood score associated with the *Capability*$(\mathbb{P},\mathbb{S})$ family is given by the SQL query below.

```
SELECT Capability(P,S),  SUM
(MDB.Capability(P,S)_CPT.cp *
CDB.Capability(P,S)_CT.count)
AS loglikelihood
FROM MDB.Capability(P,S)_CPT
NATURAL JOIN CDB.Capability(P,S)_CT
```

The aggregate computation in this short query illustrates how well SQL constructs support complex computations with structured objects.

6 Model Manager: Structure Learning

For learning the structure of a parametrized Bayesian network, we used FACTORBASE to implement the previously existing learn-and-join algorithm (LAJ) [15,29]. The model search strategy of the LAJ algorithm is an iterative deepening search for correlations among attributes along longer and longer chains of relationships; a similar strategy was proposed by Friedman *et al.* [6]. We describe the LAJ algorithm, then discuss how FACTORBASE implements the algorithm by leveraging SQL capabilities. The previous implementation of the LAJ algorithm posted at [27], limits the par-factors so they contain at most *two* relationship par-RVs; FACTORBASE overcomes this limitation.

Table 7 The main tables in the Models Database *MDB*. For a Bayesian network, the *MDB* stores its structure, parameter estimates, and model selection scores.

BayesNet(child:par-RV,parent:par-RV)
@par-RVID@_CPT(@par-RVID@:par-RV,parent$_1$:par-RV,...,parent$_k$:par-RV,cp:real)
Scores(child:par-RV,loglikelihood:real,#par:int,aic:real)

6.1 The learn-and-join algorithm

The algorithm takes as input a database and a lattice of relationship chains. A chain represents a path template of connected entities (a metapath in the terminology of [32]). The algorithm learns a Bayesian network for each chain the lattice. The presence or absence of edges learned for shorter chains is propagated to longer chains. The final output is the Bayesian network associated with the longest relationship chain. Figure 6 illustrates the learning strategy for our running example. Algorithm 2 presents pseudocode; the following sections discuss the different components of the algorithm in detail.

Algorithm 2: Learn-and-Join Structure Learning

Input: Database $\mathscr{D}$; parametrized random variables F; relationship chain lattice $\mathscr{L}$ with maximum chain length m.
Output: A Bayes multi-net $\mathbb{B}_{\mathbf{R}}$ for relationship chains in $\mathscr{L}$.
Calls BNL: Any propositional Bayes net learner that accepts edge constraints and a single table of cases as input.

Notation: $BNL(T, Econstraints)$ is the output DAG of the Bayes net learner given data table T and edge constraints.

1: **for each** entity type E **do** {compute BN for each entity type}
2: T_E := the contingency table for the attribute nodes of E
3: $\mathbb{B}_E := BNL(T_E, \emptyset)$
4: **end for**
5: **for each** relationship node R **do** {compute BN for each relationship node}
6: Find $constraints_R$ propagated from entity BNs {Constraint 1}
7: $T_R := CT(Vars(R)$ {the contingency table for the nodes associated with R}
8: $\mathbb{B}_R := BNL(T_R, constraints_R)$
9: **end for**
10: **for** chain length $\ell \leftarrow 2, 3, \ldots, m$ **do**
11: **for** each chain $\mathbf{R}$ of length ℓ **do**
12: Find $constraints_{\mathbf{R}}$ propagated from shorter chains $\mathbf{R}^* \subset \mathbf{R}$ {Constraint 2}
13: $T_{\mathbf{R}} := CT(Vars(\mathbf{R})$ {the contingency table for the nodes associated with $\mathbf{R}$}
14: $\mathbb{B}_{\mathbf{R}} := BNL(T_{\mathbf{R}}, constraints_{\mathbf{R}})$
15: **end for**
16: **end for**

6.2 The Lattice of Relationship Chains

We represent sets of relationship functors by lists without repeating elements. Assuming an ordering of relationship par-RVs, a **relationship set** $\mathbf{R} = \{R_1(\tau_1), \ldots, R_k(\tau_k)\}$ translates into a **relationship list** $[\mathbf{R}] = [R_1(\tau_1), \ldots, R_k(\tau_k)]$. For order-independent concepts we refer to sets rather than to lists. A relationship list $[R_1(\tau_1), \ldots, R_k(\tau_k)]$ is a **chain** if each functor $R_{i+1}(\tau_{i+1})$ shares at least one population variable with the preceding terms $R_1(\tau_1), \ldots, R_i(\tau_i)$.[3] In the following we use the set notation $\mathbf{R}$ for both chains and the associated relationship set. For instance, in the University schema of Figure 1, a relationship chain of length 2 is the list

$$[RA(\mathbb{P}, \mathbb{S}), Registered(\mathbb{S}, \mathbb{C})]. \tag{2}$$

A three-element chain is

$$[RA(\mathbb{P}, \mathbb{S}), Registered(\mathbb{S}, \mathbb{C}), TA(\mathbb{C}, S)]. \tag{3}$$

A relationship chain $\mathbf{R}$ is a **subchain** of another chain $\mathbf{R}'$, written $\mathbf{R} \sqsubseteq \mathbf{R}'$, if every relationship par-RV in $\mathbf{R}$ occurs also in $\mathbf{R}'$. For example, the chain (2) is a subchain of the chain (3). Two chains are equivalent in case they contain the same relationship variables. The **relationship lattice** contains a representative chain from each equivalence class. A representative chain for a set of relationship variables can be generated using any fixed order on relationship variables. In the following we do not distinguish between a relationship chain and its equivalence class unless there is risk of confusion. The subchain relation $\sqsubseteq$ defines a lattice on (equivalence classes of) relationship chains. Figure 7 illustrates the lattice for the relationship nodes in the University schema of Figure 2. For reasons that we explain below, entity tables are also included in the lattice and linked to relationships that involve the entity in question.

6.3 The Bayesian Multinet

Each chain in the lattice corresponds to a subset $\mathbf{R}$ of relationship variables. Associated with the chain is a set of par-RVs $Vars(\mathbf{R})$, comprising the following:

- All relationship par-RVs in $\mathbf{R}$.
- Each attribute par-RV associated with a relationship par-RV in $\mathbf{R}$.

For each chain $\mathbf{R}$, the learn-and-join algorithm learns a Bayesian network $\mathbb{B}_{\mathbf{R}}$ whose nodes comprise the set $Vars(\mathbf{R})$. This network is learned from the contingency table $CT(Vars(\mathbf{R}))$.

[3] Essentially the same concept is called a slot chain in PRM modelling [8].

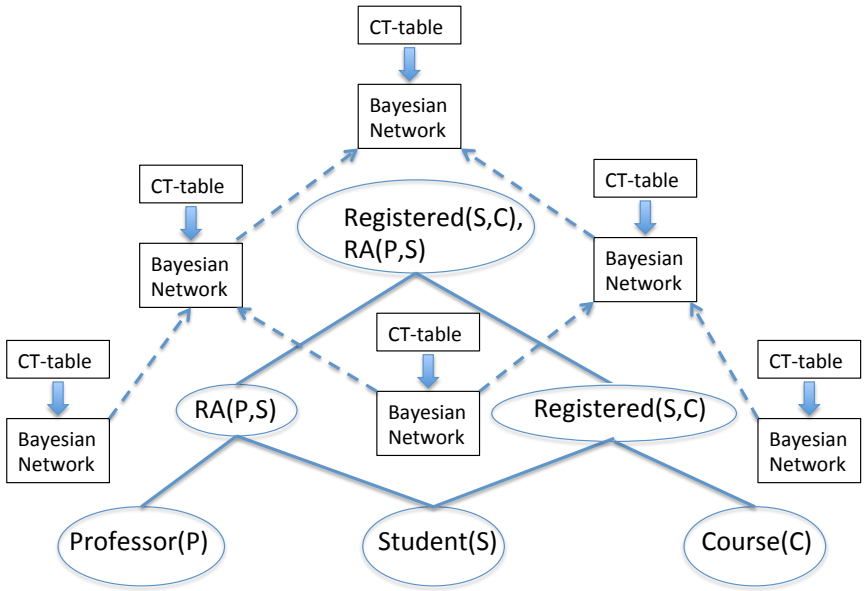

Fig. 6 Overview of the learn-and-join hierarchical structure learning method. The hierarchy is shown for two relationships, *Registered* and *RA*. For each relationship chain, SQL meta queries compute a contingency table. Solid block arrows: For each relationship chain, a single table Bayesian network learner constructs a Bayesian network, given the contingency table for each relationship chain. Dashed arrows: The absence and presence of Bayesian network edges learned for a shorter relationship chain are propagated as constraints for learning for a longer relationship chain.

The lattice structure defines a *multinet* rather than a single Bayes net. Multinets are a classic Bayes net formalism for modelling context-sensitive dependencies among variables. Geiger and Heckerman contributed a standard reference article for the multinet formalism [7]. In the learn-and-join algorithm, the context of a multinet is defined by a chain of relationship functor nodes. Distinguishing these different contexts allows us to represent that the existence of certain dependencies among attributes of entities depend on which kind of links exist between the entities. The final output of the learn-and-join algorithm is a single Bayes net derived from the multinet.

6.4 Edge Inheritance In the Relationship Lattice

These constraints state that the presence or absence of edges in graphs associated with join tables lower in the lattice is inherited by graphs associated with join tables higher in the lattice. The motivation for these constraints is that dependencies should be assessed in the most specific context possible. The first constraint states that edges from an entity table are inherited by relationship tables that involve the entity in question.

Constraint 1 *Let $\mathbb{A}$ be the population variable for an entity type associated with entity table E. Let $\mathbf{R}$ be any relationship set that contains the first-order variable $\mathbb{A}$. Then the Bayes net associated with $\mathbf{R}$ contains an edge $f(\mathbb{A}) \rightarrow g(\mathbb{A})$ connecting two descriptive attributes of $\mathbb{A}$ if and only if the Bayes net associated with E contains the edge $f(\mathbb{A}) \rightarrow g(\mathbb{A})$.*

The second constraint states that edges learned on smaller relationship sets are inherited by larger relationship sets. If the smaller sets are ambiguous with regard to the direction of an adjacency, the larger relationship set must contain the adjacency; the direction is then resolved by applying Bayes net learning to the larger relationship set.

Constraint 2 *Suppose that nodes $f(\tau), g(\tau')$ appear in the variables $Vars(\mathbf{R})$. Then*

1. *If $f(\tau)$ and $g(\tau')$ are not adjacent in any DAG $\mathbb{B}_{\mathbf{R}^*}$ associated with a relationship subset $\mathbf{R}^* \subset \mathbf{R}$, then $f(\tau)$ and $g(\tau')$ are not adjacent in the graph associated with the relationship set $\mathbf{R}$.*
2. *Else if all subset graphs agree on the orientation of the adjacency $f(\tau) - g(\tau')$, the graph associated with the relationship set $\mathbf{R}$ inherits this orientation.*
3. *Else the graph associated with the relationship set $\mathbf{R}$. must contain the edge $f(\tau) \rightarrow g(\tau')$ or the edge $f(\tau) \leftarrow g(\tau')$.*

Examples. Constraint 1. The Bayes net for the entity type *Prof* contains the edge $Popularity(P) \rightarrow Teaching\_ability(P)$. Therefore the length 1 chain $RA(P,S)$ is *required* to contain this edge as well. The Bayes net graph for the entity type *Course* does not contain the edge $Difficulty(C) \rightarrow Level(C)$, so the Bayes net for the length 1 chain $Registered(S,C)$ must *not* contain the edge $Difficulty(C) \rightarrow Level(C)$.

Table 8 Tables supporting structure learning

LatticeMember(Member:par-RV, Chain: rchain)
LatticeOrder(Subchain:rchain,SuperChain:rchain)
ChainBayesNets(Chain:rchain,child:par-RV,parent:par-RV)
RequiredEdges(Chain:rchain,child:par-RV,parent:par-RV)
ForbiddenEdges(Chain:rchain,child:par-RV,parent:par-RV)

Constraint 2. The Bayes net for the length 1 chain $RA(P,S)$ contains an edge $Salary(P,S) \rightarrow Capability(P,S)$, and does not contain $Popularity(P) \rightarrow Salary(P,S)$. So for the length 2 chain $Registered(S,C), RA(P,S)$, the edge $Salary(P,S) \rightarrow Capability(P,S)$ is required. The edge $Difficulty(C) \rightarrow Level(C)$ is forbidden. Figure 7 presents a trace of the LAJ algorithm for part of our running example. We next discuss how FACTORBASE leverages SQL to implement the algorithm.

6.5 SQL Implementation

Table 8 shows the main tables that support Bayesian network structure learning. These comprise two groups: Lattice tables related to relationship chains and graph tables related to edges among par-RVs.

6.5.1 Lattice Tables

LatticeMember

Member	Chain
RA(P,S)	[RA(P,S),Registered(Student,Course)]
Registered(Student,Course)	[RA(P,S),Registered(Student,Course)]

LatticeOrder

Subchain	SuperChain
RA(P,S)	[RA(P,S),Registered(Student,Course)]
Registered(Student,Course)	[RA(P,S),Registered(Student,Course)]

The lattice tables support SQL access to the internal structure of a relationship chain: The *LatticeMember* table lists, for each valid relationship chain, the relationship nodes that are members of this chain. The *LatticeOrder* table lists for each relationship chain, its immediate subchain. Table 6.5.1 shows the lattice tables for our running example. Here *rchain* is a custom ID type for relationship chains. Concatenating the IDs for a relationship par-RV defines an ID for a relationship chain. The relationship par-RVs are listed in the Variable Database.

We generate IDs for valid relationship chains using an application language outside of SQL. (Java in our case). The space of possible relationship chains can be constructed in different approaches to reflect domain knowledge. In our experiments, we follow the suggested default for the LAJ algorithm [29]: include all relationship chains, of any length, that contain at most 3 first-order variables.

6.5.2 Graph Tables

For each relationship chain, a Bayesian network is stored using the tabular representation described in Section 5. Table 9 illustrates this representation in our running example.

Table 9 Tabular representation of the Bayesian multi-net from Figure 6. Top: Edges learned for the relationship chain RA(P,S). Bottom: Edges learned for first-order variables Prof and Student. Bayes net learning found no edges (correlations) for attributes of Courses. *P* ranges over professors, *S* over students.

ChainBayesNets

Rchain	child	parent
RA(P,S)	Capability(P,S)	*RA(P,S)*
RA(P,S)	Intelligence(S)	*RA(P,S)*
RA(P,S)	Popularity(P)	*RA(P,S)*
RA(P,S)	Popularity(P)	Teachingability(P)
RA(P,S)	Ranking(S)	Intelligence(S)
RA(P,S)	Salary(P,S)	*RA(P,S)*
RA(P,S)	Salary(P,S)	Capability(P,S)

EntityBayesNets

FOVariable	child	parent
Prof	popularity(P)	teachingability(P)
Student	ranking(S)	intelligence(S)

For each relationship node (e.g. RA(P,S)), the *Required Edges* are the edges learned for the associated FOVariables (contained in the *EntityBayesNets* table). For each relationship chain, the required edges are the union of learned edges for shorter chains. Similarly for *Forbidden Edges*. Required and forbidden edges are exported as constraints for Bayesian network learning. Table 10 illustrates this representation.

The RequiredEdges table is implemented as a view shown in Figure 8. The view mechanism automatically adds new required edges when new learned edges are added to the ChainBayesNets table. There is a similar view (not shown) that adds forbidden edges, based which edges are *not* added to the ChainBayesNets table.

This completes our description of how the modules of FACTORBASE are implemented using SQL. We next show how these modules support a key learning task: computing the predictions of an SRL model on a test instance.

7 Test Set Predictions

Computing probabilities over the label of a test instance is important for several tasks. 1) Classifying the test instance, which is one of the main applications of a machine learning system for end users. 2) Comparing the class labels predicted against true class labels is a key step in several approaches to model scoring [17]. 3) Evaluating the accuracy

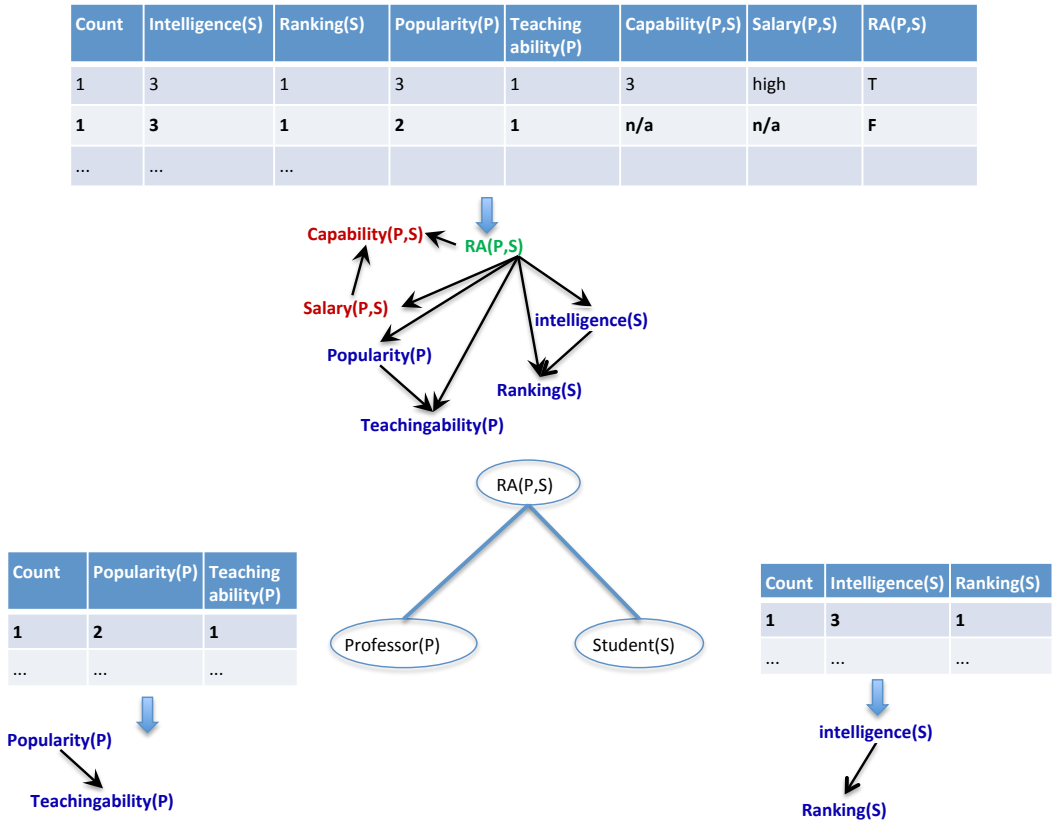

Fig. 7 Trace of the learn-and-join hierarchical structure learning method of Figure 6 for the University domain. The trace is shown for the *RA* relationship only. For each relationship chain, the figure shows the complete Bayesian network structures learned, and excerpts from the contingency tables.

Table 10 Required and Forbidden edges for the relationship chain RA(P,S). Horizontal lines separate constraints for different relationship chains. *P* ranges over professors, *S* over students, *C* over courses.

	RequiredEdges	
Rchain	child	parent
RA(P,S)	Popularity(P)	Teachingability(P)
RA(P,S)	Ranking(S)	Intelligence(S)
Reg.(S,C), RA(P,S)	Capability(P,S)	Salary(P,S)
Reg.(S,C), RA(P,S)	Popularity(P)	Teachingability(P)
Reg.(S,C), RA(P,S)	Popularity(P)	Teachingability(P)
	ForbiddenEdges	
Reg.(S,C)	Level(C)	Difficulty(C)
Reg.(S,C)	Difficulty(C)	Level(C)
Reg.(S,C), RA(P,S)	Capability(P,S)	Salary(P,S)
Reg.(S,C), RA(P,S)	Level(C)	Difficulty(C)
Reg.(S,C), RA(P,S)	Difficulty(C)	Level(C)

of a machine learning algorithm by the train-and-test paradigm, where the system is provided a training set for learning and then we test its predictions on unseen test cases. We first discuss how to compute a prediction for a single test case, then how to compute an overall prediction score for a set of test cases. Class probabilities can be derived from Equation 1 as follows [17, Sec.2.2.2]. Let Y denote a ground par-RV to be classified, which we refer to as the **target variable**. For example, a ground atom may be $Intelligence(jack)$. In this example, we refer to jack as the **target entity**. Write $\mathbf{X}_{-Y}$ for a database instance that specifies the values of all ground par-RVs, except for the target, which are used to predict the target node. Let $[\mathbf{X}_{-Y}, y]$ denote the completed database instance where the target node is assigned value y. The log-linear model uses the likelihood $P([\mathbf{X}_{-Y}, y])$ as the joint score of the label and the predictive features. The conditional probability is proportional to this score:

$$P(y|\mathbf{X}_{-\mathbf{Y}}) \propto P([\mathbf{X}_{-Y}, y]) \tag{4}$$

where the joint distribution on the right-hand side is defined by Equation 1, and the scores of the possible class labels need to be normalized to define conditional probabilities.

```
CREATE VIEW RequiredEdges AS
SELECT DISTINCT
        LatticeOrder.Superchain AS Rchain,
        ChainBayesNets.child AS child,
        ChainBayesNets.parent AS parent
FROM
        ChainBayesNets,
        LatticeOrder
WHERE
       LatticeOrder.Subchain =
       ChainBayesNets.Rchain
UNION
       SELECT DISTINCT
        RNodes_pvars.rnid AS Rchain,
        Entity_BayesNets.child AS child,
        Entity_BayesNets.parent AS parent
        FROM
        RNodes_pvars, Entity_BayesNets
        WHERE
        RNodes_pvars.pvid = Entity_BayesNets.pvid
```

Fig. 8 SQL for a view that computes required edges from learned edges stored in *ChainBayesNets.*

SQL Implementation. The obvious approach to computing the log-linear score would be to use the likelihood computation of Section 5.3 for the entire database. This is inefficient because only instance counts that involve the target entity change the classification probability. This means that we need only consider query instantiations that match the appropriate logical variable with the target entity (e.g., $\mathbb{S} = \mathit{jack}$).

Table 12 Target contingency tables for target = jack and for target = jill.

jack_Capability_(P,S)_CT

sid	Count	Cap.(P,S)	RA(P,S)	Salary(P,S)
Jack	5	N/A	N/A	F
Jack	5	4	high	T
...	...	...	...	...

jill_Capability_(P,S)_CT

sid	Count	Cap.(P,S)	RA(P,S)	Salary(P,S)
Jill	3	N/A	N/A	F
Jill	7	4	high	T
...	...	...	...	...

Assuming that for each node with ID @parRVID@, a target contingency table named *CDB.target_@parRVID@_CT* has been built in the Count Database *CDB*, the log-likelihood SQL is as in Section 5.3. For instance, the contribution of the $\mathit{Capability}(\mathbb{P},\mathbb{S})$ family is computed by the SQL query shown, but with the contingency table jack_Capability(P,S)_CT in place of Capability(P,S)_CT. The new problem is finding the target contingency table. SQL allows us to solve this easily by restricting counts to the target entity in the WHERE clause. To illustrate, suppose we want to modify the contingency table query of Figure 4 to compute the contingency table for $\mathbb{S} = \mathit{jack}$. We add the student id to the SELECT clause, and the join condition $S.s\_id = \mathit{jack}$ to the WHERE clause; see Table 11. The FROM clause is the same as in Figure 4. The metaquery of Figure 4 is easily changed to produce these SELECT and WHERE clauses.

Next consider a setting where a model is to be scored against an entire test set. For concreteness, suppose the problem is to predict the intelligence of a set of students $\mathit{Intelligence}(\mathit{jack})$, $\mathit{Intelligence}(\mathit{jill})$, $\mathit{Intelligence}(\mathit{student}_3)$, $\ldots, \mathit{Intelligence}(\mathit{student}_m)$. An obvious approach is to loop through the set of test instances, repeating the likelihood query above for each single instance. Instead, SQL supports *block access* where we process the test instances as a block. Intuitively, instead of building a contingency table for each test instance, we build a single contingency table that stacks together the individual contingency tables (Table 12). Blocked access can be implemented in a beautifully simple manner in SQL: we simply add the primary key id field for the target entity to the GROUP BY list; see Table 11.

8 Evaluation

Our experimental study describes how FACTORBASE can be used to implement a challenging machine learning application: Constructing a Bayesian network model for a relational database. Bayesian networks are a good illustration of typical challenges and how RDBMS capabilities can address them because: (1) Bayesian networks are widely regarded as a very useful model in machine learning and AI, that supports decision making and reasoning under uncertainty. At the same time, they are considered challenging to learn from data. (2) Database researchers have proposed Bayesian networks for combining databases with uncertainty[34]. (3) A Bayesian network with par-RVs can be easily converted to other first-order relational models, such as a Markov Logic Network; see [3, **?**].

We describe the system and the datasets we used. Code was written in MySQL Script and Java, JRE 1.7.0. and executed with 8GB of RAM and a single Intel Core 2 QUAD Processor Q6700 with a clock speed of 2.66GHz (no hyperthreading). The operating system was Linux Centos 2.6.32. The MySQL Server version 5.5.34 was run with 8GB of RAM and a single core processor of 2.2GHz. All code and datasets are available on-line [27].

8.1 Datasets

We used six benchmark real-world databases. For detailed descriptions and the sources of the databases, please see [27] and the references therein. Table 13 summarizes basic information about the benchmark datasets. IMDb is the

Table 11 SQL queries for computing target contingency tables supporting test set prediction. <Attribute-List> and <Key-Equality-List> are as in Figure 4.

Access	SELECT	WHERE	GROUP BY
Single	COUNT(*) AS count, <Attribute-List>, S.sid	<Key-Equality-List> AND S.s_id = jack	<Attribute-List>
Block	COUNT(*) AS count, <Attribute-List>, S.sid	<Key-Equality-List>	<Attribute-List>, S.sid

largest dataset in terms of number of total tuples (more than 1.3M tuples) and schema complexity. It combines the MovieLens database[4] with data from the Internet Movie Database (IMDb)[5] following [25].

Table 13 Datasets characteristics. #Tuples = total number of tuples over all tables in the dataset.

Dataset	#Relationship Tables/ Total	# par-RV	#Tuples
Movielens	1 / 3	7	1,010,051
Mutagenesis	2 / 4	11	14,540
UW-CSE	2 / 4	14	712
Mondial	2 / 4	18	870
Hepatitis	3 / 7	19	12,927
IMDb	3 / 7	17	1,354,134

Table 13 provides information about the number of par-RVs generated for each database. More complex schemas generate more random variables.

8.2 Bayesian Network Learning

We applied to each dataset our new SQL-based implementation of the LAJ algorithm described in Section 6.

A major design decision is how to make sufficient statistics available to the LAJ algorithm. In our experiments we followed a *pre-counting* approach where the count manager constructs a **joint contingency table** for *all* par-RVs in the random variable database. An alternative would be *on-demand* counting, which computes many contingency tables, but only for factors that are constructed during the model search [20]. Pre-counting is a form of data preprocessing: Once the joint contingency table is constructed, local contingency tables can be built quickly by summing (Group By). Different structure learning algorithms can therefore be run quickly on the same joint contingency table. For our evaluation, pre-counting has several advantages. (1) Constructing the joint contingency table presents a maximally challenging task for the count manager. (2) Separating counting/data access from model search allows us to assess separately the resources required for each task.

[4] www.grouplens.org, 1M version
[5] www.imdb.com, July 2013

8.3 Results

Table 14 reports the number of sufficient statistics for constructing the joint contingency table. This number depends mainly on the number of par-RVs. The number of sufficient statistics can be quite large, over 15M for the largest dataset IMDb. Even with such large numbers, constructing contingency tables using the SQL metaqueries is feasible, taking just over 2 hours for the very large IMDb set. The number of Bayesian network parameters is much smaller than the number of sufficient statistics. The difference between the number of parameters and the number of sufficient statistics measures how compactly the BN summarizes the statistical information in the data. Table 14 shows that Bayesian networks provide very compact summaries of the data statistics. For instance for the Hepatitis dataset, the ratio is $12,374,892/569 > 20,000$. The IMDb database is an outlier, with a complex correlation pattern that leads to a dense Bayesian network structure.

Table 14 Count Manager: Sufficient Statistics and Parameters

Dataset	# Database Tuples	# Sufficient Statistics (SS)	SS Computing Time (s)	#BN Parameters
Movielens	1,010,051	252	2.7	292
Mutagenesis	14,540	1,631	1.67	721
UW-CSE	712	2,828	3.84	241
Mondial	870	1,746,870	1,112.84	339
Hepatitis	12,927	12,374,892	3,536.76	569
IMDb	1,354,134	15,538,430	7,467.85	60,059

Table 15 shows that the graph structure of a Bayesian network contains a small number of edges relative to the number of parameters. The parameter manager provides fast maximum likelihood estimates for a given structure. This is because computing a local contingency table for a BN family is fast given the joint contingency table.

Figure 9 compares computing predictions on a test set using an instance-by-instance loop, with a separate SQL query for each instance, vs. a single SQL query for all test instances as a block (Table 11). Table 16 specifies the number of test instances for each dataset. We split each benchmark database into 80% training data, 20% test data. The test instances are the ground atoms of all descriptive attributes of entities. The blocked access method is 10-100 faster depending on the dataset. The single access method did not scale to the large IMDb dataset (timeout after 12 hours).

Table 16 # of Test Instances

Dataset	Movielens	Mutagenesis	UW-CSE	Mondial	Hepatitis	IMDb
#instance	4,742	3,119	576	505	2,376	46,275

Table 15 Model Manager Evaluation.

Dataset	# Edges in Bayes Net	# Bayes Net Parameters	Parameter Learning Time (s)
Movielens	72	292	0.57
Mutagenesis	124	721	0.98
UW-CSE	112	241	1.14
Mondial	141	339	60.55
Hepatitis	207	569	429.15
IMDb	195	60,059	505.61

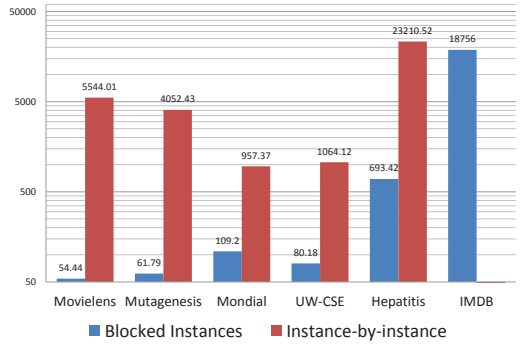

Fig. 9 Times (s) for Computing Predictions on Test Instances. The right red column shows the time for looping over single instances using the Single Access Query of Table 11. The left blue column shows the time for the Blocked Access Query of Table 11.

Table 17 reports result for the complete learning of a Bayesian network, structure and parameters. It benchmarks FACTORBASE against functional gradient boosting, a state-of-the-art multi-relational learning approach. MLN_Boost learns a Markov Logic Network, and RDN_Boost a Relational Dependency Network. We used the Boostr implementation [16]. To make the results easier to compare across databases and systems, we divide the total running time by the number of par-RVs for the database (Table 13). Table 17 shows that structure learning with FACTORBASE is fast: even the large complex database IMDb requires only around 8 minutes/par-RV. Compared to the boosting methods, FACTORBASE shows excellent scalability: neither boosting method terminates on the IMDb database, and while RDN_Boost terminates on the MovieLens database, it is almost 5,000 times slower than FACTORBASE. Much of the speed of our implementation is due to quick access to sufficient statistics. As the last column of Table 17 shows, on the larger datasets FACTORBASE spends about 80% of computation time on gathering sufficient statistics via the count manager. This suggests that a large speedup for the boosting algorithms could be achieved if they used the FACTORBASE in-database design.

We do not report accuracy results due to space constraints and because predictive accuracy is not the focus of this paper. On the standard conditional log-likelihood metric, as defined by Equation 4, the model learned by FACTORBASE performs better than the boosting methods on all databases. This is consistent with the results of previous studies [29].

Table 17 Learning Time Comparison (sec) with other statistical-relational learning systems. NT = non-termination

Dataset	RDN_Boost	MLN_Boost	FB-Total	FB-Count
MovieLens	5,562	N/T	1.12	0.39
Mutagenesis	118	49	1	0.15
UW-CSE	15	19	1	0.27
Mondial	27	42	102	61.82
Hepatitis	251	230	286	186.15
IMDb	N/T	N/T	524.25	439.29

Conclusion. FACTORBASE leverages RDBMS capabilities for scalable management of statistical analysis objects. It efficiently constructs and stores large numbers of sufficient statistics and parameter estimates. The RDBMS support for statistical-relational learning translates into orders of magnitude improvements in speed and scalability.

9 Related Work

The design space for combining machine learning with data management systems offers a number of possibilities, several of which have been explored in previous and ongoing research. We selectively review the work most relevant to our research. Figure 10 provides a tree structure for the research landscape.

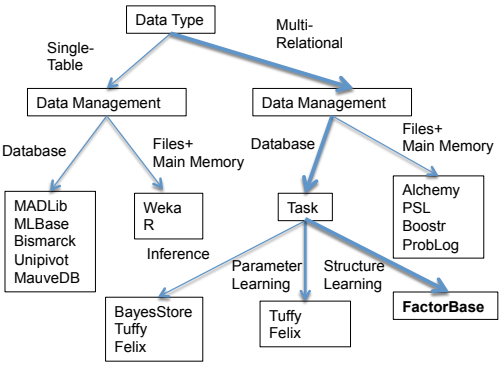

Fig. 10 A tree structure for related work in the design space of machine learning × data management

9.1 Single-Table Machine Learning

Most machine learning systems, such as Weka or R, support learning from a single table or data matrix only. The single-table representation is appropriate when the data points represent a homogeneous class of entities with similar attributes, where the attributes of one entity are independent of those of others [17]. The only way a single-table system can be applied to multi-relational data is after a preprocessing step where multiple interrelated tables are converted to a single data table. When the learning task is classification, such preprocessing is often called propositionalization [17]. This "flattening" of the relational structure typically involves a loss of information.

9.1.1 RDBMS Learning

Leveraging RDBMS capabilities through SQL programming is the unifying idea of the recent MADLib framework [13]. An advantage of the MADLib approach that is shared by FACTORBASE is that in-database processing avoids exporting the data from the input database. The Apache Spark [1] framework includes MLBase and SparkSQL that provide support for distributed processing, SQL, and automatic refinement of machine learning algorithms and models [18]. Other RDBMS applications include gathering sufficient statistics [11], and convex optimization [5]. The MauveDB system [2] emphasizes the importance of several RDBMS features for combining statistical analysis with databases. As in FACTORBASE, this includes storing models and associated parameters as objects in their own right, and using the view mechanism to update statistical objects as the data change. A difference is that MauveDB presents model-based views of the *data* to the user, whereas FACTORBASE presents views of the *models* to machine learning applications.

9.1.2 RDBMS Inference

Wong *et al.* applied SQL operators such as the natural join to perform log-linear inference with a single-table graphical model [36] stored in an RDBMS. Monte Carlo methods have also been implemented with an RDBMS to perform inference with uncertain data [14,35]. The MCDB system [14] stores parameters in database tables like FACTORBASE .

9.2 Multi-Relational Learning

For overviews of multi-relational learning please see [9,3, 17]. Most implemented systems, such as Aleph and Alchemy, use a logic-based representation of data derived from Prolog facts, that originated in the Inductive Logic Programming community [4].

9.2.1 RDBMS Learning

The ClowdFlows system [19] allows a user to specify a MySQL database as a data source, then converts the MySQL data to a single-table representation using propositionalization. Singh and Graepel [31] present an algorithm that analyzes the relational database system catalog to generate a set of nodes and a Bayesian network structure. This approach utilizes SQL constructs as a data description language in a way that is similar to our Schema Analyzer. Differences include the following. (1) The Bayesian network structure is fixed and based on latent variables, rather than learned for observable variables only, as in our case study. (2) The RDBMS is not used to support learning after random variables have been extracted from the schema.

Qian *et al.* [28] discuss work related to the contingency table problem and introduce contingency table algebra. Their paper focuses on a Virtual Join algorithm for computing sufficient statistics that involve negated relationships. They do not discuss integrating contingency tables with other structured objects for multi-relational learning.

9.2.2 RDBMS Inference

Database researchers have developed powerful probabilistic inference algorithms for multi-relational models. The BayesStore system [34] introduced the principle of treating all statistical objects as first-class citizens in a relational database as FACTORBASE does. The Tuffy system [23] achieves highly reliable and scalable inference for Markov Logic Networks (MLNs) with an RDBMS. It leverages inference capabilities to perform MLN parameter learning. RDBMS support for local search parameter estimation procedures, rather than closed-form maximum-likelihood estimation, has also been explored [5,23,24].

10 Conclusion and Future Work

Compared to traditional learning with a single data table, learning for multi-relational data requires new system capabilities. In this paper we described FACTORBASE , a system that leverages the existing capabilities of an SQL-based RDBMS to support statistical-relational learning. Representational tasks include specifying metadata about structured parametrized random variables, and storing the structure of a learned model. Computational tasks include storing and constructing sufficient statistics, and computing parameter estimates and model selection scores. We showed that SQL scripts can be used to implement these capabilities, with multiple advantages. These advantages include: 1) Fast program development through high-level SQL constructs for complex table and count operations. 2) Managing large and complex statistical objects that are too big to fit in main

memory. For instance, some of our benchmark databases require storing and querying millions of sufficient statistics. While FACTORBASE provides good solutions for each of these system capabilities in isolation, the ease with which large complex statistical-relational objects can be integrated via SQL queries is a key feature. Empirical evaluation on six benchmark databases showed significant scalability advantages from utilizing the RDBMS capabilities: Both structure and parameter learning scaled well to millions of data records, beyond what previous multi-relational learning systems can achieve.

Future Work. Further potential application areas for FACTORBASE include managing massive numbers of aggregate features for classification [26], and collective matrix factorization [30,31]. There are opportunities for optimizing RDBMS operations for the workloads required by statistical-relational structure learning. These include view materialization and the key scalability bottleneck of computing multi-relational sufficient statistics. NoSQL databases can exploit a flexible data representation for scaling to very large datasets. However, SRL requires count operations for random complex join queries, which is a challenge for less structured data representations. An important goal is a single RDBMS package for both learning and inference that integrates FACTORBASE with inference systems such as BayesStore and Tuffy.

11 Acknowledgments

This research was supported by a Discovery grant to Oliver Schulte by the Natural Sciences and Engineering Research Council of Canada. Zhensong Qian was supported by a grant from the China Scholarship Council.

12 Conflict of Interest

On behalf of all authors, the corresponding author states that there is no conflict of interest.

A Appendix: The Random Variable Database Layout

We provide details about the Schema Analyzer. Table 18 shows the relational schema of the Random Variable Database. Figure 11 shows dependencies between the tables of this schema.

B MySQL script for creating the Random Variable Database

This script creates default random variables and their metadata, based on the schema information in the system catalog.

Table 18 Schema for Random Variable Database

Table Name	Column Names
AttributeColumns	TABLE_NAME, COLUMN_NAME
Domain	COLUMN_NAME, VALUE
Pvariables	Pvid, TABLE_NAME
1Variables	1VarID, COLUMN_NAME, Pvid
2Variables	2VarID, COLUMN_NAME, Pvid1, Pvid2, TABLE_NAME
Relationship	RVarID, TABLE_NAME, Pvid1, Pvid2, COLUMN_NAME1, COLUMN_NAME2

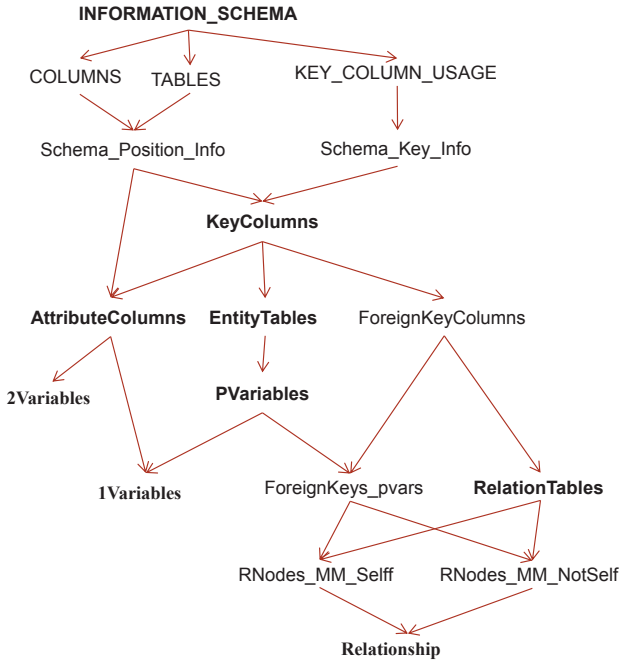

Fig. 11 Table Dependencies in the Random Variable Database *VDB*.

```
/*AchemaAnalyzer.sql*/
DROP SCHEMA IF EXISTS @database@_AchemaAnalyzer;
CREATE SCHEMA  @database@_AchemaAnalyzer;

CREATE SCHEMA  if not exists @database@_BN;
CREATE SCHEMA  if not exists @database@_CT;

USE @database@_AchemaAnalyzer;
SET storage_engine=INNODB;

CREATE TABLE Schema_Key_Info AS SELECT TABLE_NAME, COLUMN_NAME,
REFERENCED_TABLE_NAME, REFERENCED_COLUMN_NAME, CONSTRAINT_NAME
FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
WHERE (KEY_COLUMN_USAGE.TABLE_SCHEMA =
'@database@') ORDER BY TABLE_NAME;

CREATE TABLE Schema_Position_Info AS
SELECT COLUMNS.TABLE_NAME,
COLUMNS.COLUMN_NAME,
COLUMNS.ORDINAL_POSITION FROM
INFORMATION_SCHEMA.COLUMNS,
INFORMATION_SCHEMA.TABLES
WHERE
(COLUMNS.TABLE_SCHEMA = '@database@'
    AND TABLES.TABLE_SCHEMA = '@database@'
    AND TABLES.TABLE_NAME = COLUMNS.TABLE_NAME
    AND TABLES.TABLE_TYPE = 'BASE TABLE')
ORDER BY TABLE_NAME;
```

```
CREATE TABLE NoPKeys AS SELECT TABLE_NAME FROM
Schema_Key_Info
WHERE
TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
        Schema_Key_Info
    WHERE
        CONSTRAINT_NAME LIKE 'PRIMARY');

CREATE table NumEntityColumns AS
SELECT
    TABLE_NAME, COUNT(DISTINCT COLUMN_NAME) num
FROM
    Schema_Key_Info
WHERE
    CONSTRAINT_NAME LIKE 'PRIMARY'
        OR REFERENCED_COLUMN_NAME IS NOT NULL
GROUP BY TABLE_NAME;

CREATE TABLE TernaryRelations as SELECT TABLE_NAME FROM
NumEntityColumns
WHERE
num > 2;


CREATE TABLE AttributeColumns AS
SELECT TABLE_NAME, COLUMN_NAME FROM
Schema_Position_Info
WHERE
(TABLE_NAME , COLUMN_NAME) NOT IN (SELECT
        TABLE_NAME, COLUMN_NAME
    FROM
        KeyColumns)
    and TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
        NoPKeys)
    and TABLE_NAME NOT IN (SELECT
        TABLE_NAME
    FROM
        TernaryRelations);

ALTER TABLE AttributeColumns
ADD PRIMARY KEY (TABLE_NAME,COLUMN_NAME);

CREATE TABLE InputColumns AS SELECT * FROM
KeyColumns
WHERE
CONSTRAINT_NAME = 'PRIMARY'
ORDER BY TABLE_NAME;

CREATE TABLE ForeignKeyColumns AS SELECT * FROM
KeyColumns
WHERE
REFERENCED_COLUMN_NAME IS NOT NULL
ORDER BY TABLE_NAME;

ALTER TABLE ForeignKeyColumns
ADD PRIMARY KEY (TABLE_NAME,COLUMN_NAME,
REFERENCED_TABLE_NAME);

CREATE TABLE EntityTables AS
SELECT distinct TABLE_NAME, COLUMN_NAME
FROM
KeyColumns T
WHERE
1 = (SELECT
        COUNT(COLUMN_NAME)
    FROM
        KeyColumns T2
    WHERE
        T.TABLE_NAME = T2.TABLE_NAME
            AND CONSTRAINT_NAME = 'PRIMARY');

ALTER TABLE EntityTables
ADD PRIMARY KEY (TABLE_NAME,COLUMN_NAME);

CREATE TABLE SelfRelationships AS
SELECT DISTINCT RTables1.TABLE_NAME
AS TABLE_NAME,
RTables1.REFERENCED_TABLE_NAME AS REFERENCED_TABLE_NAME,
RTables1.REFERENCED_COLUMN_NAME AS REFERENCED_COLUMN_NAME FROM
KeyColumns AS RTables1,
KeyColumns AS RTables2
WHERE
(RTables1.TABLE_NAME = RTables2.TABLE_NAME)    AND
(RTables1.REFERENCED_TABLE_NAME = RTables2.REFERENCED_TABLE_NAME)
 AND
(RTables1.REFERENCED_COLUMN_NAME = RTables2.REFERENCED_COLUMN_NAME)
 AND
(RTables1.ORDINAL_POSITION < RTables2.ORDINAL_POSITION);

ALTER TABLE SelfRelationships ADD PRIMARY KEY (TABLE_NAME);

CREATE TABLE Many_OneRelationships AS
SELECT KeyColumns1.TABLE_NAME FROM
KeyColumns AS KeyColumns1,
KeyColumns AS KeyColumns2
WHERE
(KeyColumns1.TABLE_NAME , KeyColumns1.COLUMN_NAME) IN (SELECT
        TABLE_NAME, COLUMN_NAME
    FROM         InputColumns)
    AND (KeyColumns2.TABLE_NAME , KeyColumns2.COLUMN_NAME) IN
  (SELECT  TABLE_NAME, COLUMN_NAME
    FROM    ForeignKeyColumns)
    AND (KeyColumns2.TABLE_NAME , KeyColumns2.COLUMN_NAME)
   NOT IN (SELECT  TABLE_NAME, COLUMN_NAME
    FROM     InputColumns);

CREATE TABLE PVariables AS
SELECT CONCAT(EntityTables.TABLE_NAME, '0') AS Pvid,
EntityTables.TABLE_NAME,
0 AS index_number FROM
EntityTables
UNION
SELECT
CONCAT(EntityTables.TABLE_NAME, '1') AS Pvid,
EntityTables.TABLE_NAME,
1 AS index_number
FROM
EntityTables,
SelfRelationships
WHERE
EntityTables.TABLE_NAME = SelfRelationships.REFERENCED_TABLE_NAME
AND
EntityTables.COLUMN_NAME = SelfRelationships.REFERENCED_COLUMN_NAME;

ALTER TABLE PVariables ADD PRIMARY KEY (Pvid);

CREATE TABLE RelationTables AS
SELECT DISTINCT ForeignKeyColumns.TABLE_NAME,
ForeignKeyColumns.TABLE_NAME IN (SELECT
        TABLE_NAME
    FROM
        SelfRelationships) AS SelfRelationship,
ForeignKeyColumns.TABLE_NAME IN (SELECT
        TABLE_NAME
    FROM
        Many_OneRelationships) AS Many_OneRelationship FROM
ForeignKeyColumns;

ALTER TABLE RelationTables
ADD PRIMARY KEY (TABLE_NAME);

CREATE TABLE 1Variables AS
SELECT CONCAT('`', COLUMN_NAME, '(', Pvid, ')', '`') AS 1VarID,
COLUMN_NAME,
Pvid,
index_number = 0 AS main FROM
PVariables
    NATURAL JOIN
AttributeColumns;

ALTER TABLE 1Variables ADD PRIMARY KEY (1VarID);
ALTER TABLE 1Variables ADD UNIQUE(Pvid,COLUMN_NAME);

CREATE TABLE ForeignKeys_pvars AS
SELECT ForeignKeyColumns.TABLE_NAME,
ForeignKeyColumns.REFERENCED_TABLE_NAME,
ForeignKeyColumns.COLUMN_NAME,
Pvid,
index_number,
ORDINAL_POSITION AS ARGUMENT_POSITION FROM
```

```
ForeignKeyColumns,
PVariables
WHERE
PVariables.TABLE_NAME = REFERENCED_TABLE_NAME;

ALTER TABLE ForeignKeys_pvars
ADD PRIMARY KEY (TABLE_NAME,Pvid,
ARGUMENT_POSITION);

CREATE table Relationship_MM_NotSelf AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars1.TABLE_NAME,
            '(',
            ForeignKeys_pvars1.Pvid,
            ',',
            ForeignKeys_pvars2.Pvid,
            ')',
            '`') AS orig_RVarID,
    ForeignKeys_pvars1.TABLE_NAME,
    ForeignKeys_pvars1.Pvid AS Pvid1,
    ForeignKeys_pvars2.Pvid AS Pvid2,
    ForeignKeys_pvars1.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars2.COLUMN_NAME AS COLUMN_NAME2,
    (ForeignKeys_pvars1.index_number = 0
        AND ForeignKeys_pvars2.index_number = 0) AS main
FROM
    ForeignKeys_pvars AS ForeignKeys_pvars1,
    ForeignKeys_pvars AS ForeignKeys_pvars2,
    RelationTables
WHERE
 ForeignKeys_pvars1.TABLE_NAME = ForeignKeys_pvars2.TABLE_NAME
 AND RelationTables.TABLE_NAME = ForeignKeys_pvars1.TABLE_NAME
        AND ForeignKeys_pvars1.ARGUMENT_POSITION <
        ForeignKeys_pvars2.ARGUMENT_POSITION
        AND RelationTables.SelfRelationship = 0
        AND RelationTables.Many_OneRelationship = 0;

CREATE table Relationship_MM_Self AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars1.TABLE_NAME,
            '(',
            ForeignKeys_pvars1.Pvid,
            ',',
            ForeignKeys_pvars2.Pvid,
            ')',
            '`') AS orig_RVarID,
    ForeignKeys_pvars1.TABLE_NAME,
    ForeignKeys_pvars1.Pvid AS Pvid1,
    ForeignKeys_pvars2.Pvid AS Pvid2,
    ForeignKeys_pvars1.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars2.COLUMN_NAME AS COLUMN_NAME2,
    (ForeignKeys_pvars1.index_number = 0
        AND ForeignKeys_pvars2.index_number = 1) AS main
FROM
    ForeignKeys_pvars AS ForeignKeys_pvars1,
    ForeignKeys_pvars AS ForeignKeys_pvars2,
    RelationTables
WHERE
ForeignKeys_pvars1.TABLE_NAME = ForeignKeys_pvars2.TABLE_NAME
AND RelationTables.TABLE_NAME = ForeignKeys_pvars1.TABLE_NAME
AND ForeignKeys_pvars1.ARGUMENT_POSITION <
ForeignKeys_pvars2.ARGUMENT_POSITION
AND
ForeignKeys_pvars1.index_number < ForeignKeys_pvars2.index_number
AND RelationTables.SelfRelationship = 1
AND RelationTables.Many_OneRelationship = 0;

CREATE table Relationship_MO_NotSelf AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars.REFERENCED_TABLE_NAME,
            '(',
            PVariables.Pvid,
            ')=',
            ForeignKeys_pvars.Pvid,
            '`') AS orig_RVarID,
    ForeignKeys_pvars.TABLE_NAME,
    PVariables.Pvid AS Pvid1,
    ForeignKeys_pvars.Pvid AS Pvid2,
    KeyColumns.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars.COLUMN_NAME AS COLUMN_NAME2,
    (PVariables.index_number = 0
        AND ForeignKeys_pvars.index_number = 0) AS main
FROM
    ForeignKeys_pvars,
    RelationTables,
    KeyColumns,
    PVariables
WHERE
    RelationTables.TABLE_NAME = ForeignKeys_pvars.TABLE_NAME
        AND RelationTables.TABLE_NAME = PVariables.TABLE_NAME
        AND RelationTables.TABLE_NAME = KeyColumns.TABLE_NAME
        AND RelationTables.SelfRelationship = 0
        AND RelationTables.Many_OneRelationship = 1;

CREATE table Relationship_MO_Self AS
SELECT
    CONCAT('`',
            ForeignKeys_pvars.REFERENCED_TABLE_NAME,
            '(',
            PVariables.Pvid,
            ')=',
            ForeignKeys_pvars.Pvid,
            '`') AS orig_RVarID,
    ForeignKeys_pvars.TABLE_NAME,
    PVariables.Pvid AS Pvid1,
    ForeignKeys_pvars.Pvid AS Pvid2,
    KeyColumns.COLUMN_NAME AS COLUMN_NAME1,
    ForeignKeys_pvars.COLUMN_NAME AS COLUMN_NAME2,
    (PVariables.index_number = 0
        AND ForeignKeys_pvars.index_number = 1) AS main
FROM
    ForeignKeys_pvars,
    RelationTables,
    KeyColumns,
    PVariables
WHERE
    RelationTables.TABLE_NAME = ForeignKeys_pvars.TABLE_NAME
        AND RelationTables.TABLE_NAME = PVariables.TABLE_NAME
        AND RelationTables.TABLE_NAME = KeyColumns.TABLE_NAME
AND PVariables.index_number < ForeignKeys_pvars.index_number
        AND RelationTables.SelfRelationship = 1
        AND RelationTables.Many_OneRelationship = 1;

CREATE TABLE Relationship AS SELECT * FROM
Relationship_MM_NotSelf
UNION SELECT
*
FROM
Relationship_MM_Self
UNION SELECT
*
FROM
Relationship_MO_NotSelf
UNION SELECT
*
FROM
Relationship_MO_Self;

ALTER TABLE Relationship ADD PRIMARY KEY (orig_RVarID);
ALTER TABLE `Relationship`
ADD COLUMN `RVarID` VARCHAR(10) NULL ,
ADD UNIQUE INDEX `RVarID_UNIQUE` (`RVarID` ASC) ;


CREATE TABLE 2Variables AS SELECT CONCAT('`',
        COLUMN_NAME,
        '(',
        Pvid1,
        ',',
        Pvid2,
        ')',
        '`') AS 2VarID,
COLUMN_NAME,
Pvid1,
Pvid2,
TABLE_NAME,
main FROM
Relationship    NATURAL JOIN  AttributeColumns;

ALTER TABLE 2Variables ADD PRIMARY KEY (2VarID);</p>
```

References

1. Contributors, A.S.P.: Apache Spark. http://spark.apache.org/
2. Deshpande, A., Madden, S.: MauveDB: supporting model-based user views in database systems. In: SIGMOD, pp. 73–84. ACM (2006)
3. Domingos, P., Lowd, D.: Markov Logic: An Interface Layer for Artificial Intelligence. Morgan and Claypool Publishers (2009)
4. Dzeroski, S., Lavrac, N.: Relational Data Mining. Springer, Berlin (2001)
5. Feng, X., Kumar, A., Recht, B., Ré, C.: Towards a unified architecture for in-RDBMS analytics. In: SIGMOD Conference, pp. 325–336 (2012)
6. Friedman, N., Getoor, L., Koller, D., Pfeffer, A.: Learning probabilistic relational models. In: IJCAI, pp. 1300–1309. Springer-Verlag (1999)
7. Geiger, D., Heckerman, D.: Knowledge representation and inference in similarity networks and bayesian multinets. Artificial Intelligence **82**(1-2), 45–74 (1996)
8. Getoor, L., Friedman, N., Koller, D., Pfeffer, A., Taskar, B.: Probabilistic relational models. In: Introduction to Statistical Relational Learning [9], chap. 5, pp. 129–173
9. Getoor, L., Taskar, B.: Introduction to Statistical Relational Learning. MIT Press (2007)
10. Getoor, L., Taskar, B., Koller, D.: Selectivity estimation using probabilistic models. ACM SIGMOD Record **30**(2), 461–472 (2001)
11. Graefe, G., Fayyad, U.M., Chaudhuri, S.: On the efficient gathering of sufficient statistics for classification from large SQL databases. In: KDD, pp. 204–208 (1998). URL `http://www.aaai.org/Library/KDD/1998/kdd98-034.php`
12. Heckerman, D., Meek, C., Koller, D.: Probabilistic entity-relationship models, PRMs, and plate models. In: Getoor and Taskar [9]
13. Hellerstein, J.M., Ré, C., Schoppmann, F., Wang, D.Z., Fratkin, E., Gorajek, A., Ng, K.S., Welton, C., Feng, X., Li, K., Kumar, A.: The MADlib analytics library: Or MAD skills, the SQL. PVLDB **5**(12), 1700–1711 (2012). URL `http://dl.acm.org/citation.cfm?id=2367502.2367510`
14. Jampani, R., Xu, F., Wu, M., Perez, L.L., Jermaine, C.M., Haas, P.J.: MCDB: a monte carlo approach to managing uncertain data. In: SIGMOD Conference, pp. 687–700 (2008)
15. Khosravi, H., Schulte, O., Man, T., Xu, X., Bina, B.: Structure learning for Markov logic networks with many descriptive attributes. In: AAAI, pp. 487–493 (2010)
16. Khot, T., Shavlik, J., Natarajan, S.: Boostr. http://pages.cs.wisc.edu/ tushar/Boostr/
17. Kimmig, A., Mihalkova, L., Getoor, L.: Lifted graphical models: a survey. Machine Learning **99**(1), 1–45 (2015). DOI 10.1007/s10994-014-5443-2. URL `http://dx.doi.org/10.1007/s10994-014-5443-2`
18. Kraska, T., Talwalkar, A., Duchi, J.C., Griffith, R., Franklin, M.J., Jordan, M.I.: MLbase: A distributed machine-learning system. In: CIDR (2013)
19. Lavravc, N., Perovvsek, M., Vavpetivc, A.: Propositionalization online. In: ECML, pp. 456–459. Springer (2014)
20. Lv, Q., Xia, X., Qian, P.: A fast calculation of metric scores for learning Bayesian network. Int. J. of Automation and Computing **9**, 37–44 (2012). URL `http://dx.doi.org/10.1007/s11633-012-0614-8`
21. Milch, B., Marthi, B., Russell, S.J., Sontag, D., Ong, D.L., Kolobov, A.: BLOG: probabilistic models with unknown objects. In: IJCAI-05, pp. 1352–1359 (2005). URL `http://www.ijcai.org/papers/1546.pdf`
22. Moore, A.W., Lee, M.S.: Cached sufficient statistics for efficient machine learning with large datasets. J. Artif. Intell. Res. (JAIR) **8**, 67–91 (1998)
23. Niu, F., Ré, C., Doan, A., Shavlik, J.W.: Tuffy: Scaling up statistical inference in Markov Logic Networks using an RDBMS. PVLDB **4**(6), 373–384 (2011)
24. Niu, F., Zhang, C., Ré, C., Shavlik, J.: Felix: Scaling Inference for Markov Logic with an Operator-based Approach. ArXiv e-prints (2011)
25. Peralta, V.: Extraction and integration of MovieLens and IMDb data. Tech. rep., Technical Report, Laboratoire PRiSM (2007)
26. Popescul, A., Ungar, L.: Feature generation and selection in multi-relational learning. In: Introduction to Statistical Relational Learning [9], chap. 16, pp. 453–476
27. Qian, Z., Schulte, O.: The BayesBase system (2015). `www.cs.sfu.ca/~oschulte/BayesBase/BayesBase.html`
28. Qian, Z., Schulte, O., Sun, Y.: Computing multi-relational sufficient statistics for large databases. In: CIKM, pp. 1249–1258. ACM (2014)
29. Schulte, O., Khosravi, H.: Learning graphical models for relational data via lattice search. Machine Learning **88**(3), 331–368 (2012)
30. Singh, A.P., Gordon, G.J.: Relational learning via collective matrix factorization. In: SIGKDD, pp. 650–658. ACM (2008)
31. Singh, S., Graepel, T.: Automated probabilistic modeling for relational data. In: CIKM, pp. 1497–1500. ACM (2013)
32. Sun, Y., Han, J.: Mining Heterogeneous Information Networks: Principles and Methodologies, vol. 3. Morgan & Claypool Publishers (2012)
33. Walker, T., O'Reilly, C., Kunapuli, G., Natarajan, S., Maclin, R., Page, D., Shavlik, J.W.: Automating the ILP setup task: Converting user advice about specific examples into general background knowledge. In: ILP, pp. 253–268 (2010)
34. Wang, D.Z., Michelakis, E., Garofalakis, M., Hellerstein, J.M.: BayesStore: managing large, uncertain data repositories with probabilistic graphical models. In: VLDB, vol. 1, pp. 340–351 (2008)
35. Wick, M.L., McCallum, A., Miklau, G.: Scalable probabilistic databases with factor graphs and MCMC. In: PVLDB, vol. 3, pp. 794–804 (2010)
36. Wong, S.M., Butz, C.J., Xiang, Y.: A method for implementing a probabilistic model as a relational database. In: UAI, pp. 556–564 (1995)