An Effective Framework for Enhancing Query Answering in a Heterogeneous Data Lake

Anonymous Author(s)

ABSTRACT

There has been a growing interest in cross-source searching to gain rich knowledge in recent years. A data lake collects massive raw and heterogeneous data with different data schemas and query interfaces. Many real-life applications require query answering over the heterogeneous data lake, such as e-commerce, bioinformatics and healthcare. In this paper, we propose LakeAns that semantically integrates heterogeneous data schemas of the lake to enhance the semantics of query answers. To this end, we propose a novel framework to efficiently and effectively perform the cross-source searching. The framework exploits a reinforcement learning method to semantically integrate the data schemas and further create a global relational schema for the heterogeneous data. It then performs a query answering algorithm based on the global schema to find answers across multiple data sources. We conduct extensive experimental evaluations using real-life data, and our approach outperforms existing solutions in terms of effectiveness and efficiency.

	name	location	buy
u_1	Smith	Germany	p_1
u_2	Franklin	Italy	p_1
u_3	Miller	Washington	p_2

(a) D_1: **User**

	title	color
p_1	Backpack	black
p_2	Shoes	white

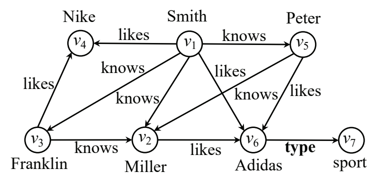
(b) D_1: **Product**

User: (name, location, buy)
Product: (title, color)

(c) Relation schema of D_1

o_1: { customer: Franklin, totalprice: 723.88,
item:[
{ p_1 : { name: Backpack, color: black,
brand: Adidas } },
{ p_2 : { ... } }] }
o_2: { customer: Smith, totalprice: 318,
item:[{ p_1: { ...}}] }

(d) D_2: a json fragment



(e) D_3: graph data

Figure 1: Example of heterogeneous data lake.

```
SELECT  Customer.name, Customer.knows, Product.title, Brand.name
FROM    Customer, Product, Brand
WHERE   Customer.buy = Product.title AND
        Product.brand = Brand.name AND
        Brand.type = 'Sports'
```

(a) A SQL query Q

	name	knows	title	brand
t_1	Smith	Miller	Backpack	Adidas
t_2	Franklin	Miller	Backpack	Adidas

(b) An answer of Q

Figure 2: Example of SQL query and its answer.

1 INTRODUCTION

The increasing diversity of information and products has led to the evolution of search and recommendations from primarily unidirectional and text-based to multi-model and heterogeneous data sources [29, 46]. Many industrial search systems have emerged interested in discovering a heterogeneous data lake to enhance the answer semantics and the search quality [14, 22]. A data lake includes multiple data models (e.g., structured, semi-structured, and unstructured) with different data schemas and query interfaces [24, 33]. For instance, an enterprise data lake as shown in Figure 1 naturally organizes transaction data in different data formats and models. Both user and product data are maintained in relational tables (see Figures 1(a)-(b)), which associates with the relation schema as shown in Figure 1(c). Orders and social networks are stored as a JSON document and a graph, respectively (see Figures 1(d)-(e)).

Exploring these data to obtain more enriching and complete knowledge is an important task for data management and applications, such as pharmacy, bioinformatics and healthcare [10, 18]. Consider an analysis task over the social e-commerce lake as shown in Figure 1, analysts would like to determine whether to recommend a product to a user Miller. This requires inspecting the holistic view of the customer information, i.e. (i) the personalized features and (ii) the social relationship of Miller. It is necessary to check the user and product information in Figures 1(a) and (b) for condition (i), and the social graph in Figure 1(e) for condition (ii).

For a holistic insight into the heterogeneous data lake, we focus on performing cross-source searching using a general query. Benefiting from the theories and techniques of relational database management systems (RDBMS), we utilize the SQL query to search for answers across multiple heterogeneous data sources. The following example motivates our work.

EXAMPLE 1. *To retrieve the customers and their friends who have bought the products typed "Sports", a data scientist would like to submit a SQL query Q illustrated in Figure 2(a) over the lake shown in Figures 1. We can see that Smith and Franklin both bought some products with the brand "Adidas" in the type of "Sports". However, neither the relational tables D_1, the JSON D_2 nor the graph D_3 can completely reflect this information.*

In this paper, we expect to find an answer like $R(Q)$ reported in Figure 2(b), which is created by the following three aspects: (i) Considering the attribute and instance overlap, we think that "Customer" and "User" refer to the same concept, similar for "Product" and "Item". (ii) For the source D_1 shown in Figures 1 (a)-(b), a set of tuples with attributes "name" and "title" associated with the concepts "Customer" and "Product" can be obtained based on D_1. As for D_2 shown in Figure 1 (d), the structural relationship associated with the concepts "Product" and "Brand" can be obtained. For D_3 shown in Figure 1 (e), the "type" information of "Brand" can be obtained. (iii) The answer $R(Q)$ can be calculated by joining these results returned from the different sources based on holistic relation schemas for the concepts "Customer", "Product" and "Brand".

Clearly, $R(Q)$ spans three different data models and completely captures the relational information for customers, which is exactly what the user wants or could be interested in.

Cross-source query answering also occurs in medicine discovery scenarios where biologists would check the basic pharmacological features of drugs and the interactions among drugs and protein molecules in the human body. This paper aims to create a global relational schema for querying across heterogeneous data sources. We will support SQL queries across multiple data sources, and to the best of our knowledge, this is the first paper to consider a global relational schema for this purpose.

The query answering in a data lake faces many challenges presented as follows. Firstly (C1), how to create a global relational schema that bridges the schema associations between different sources is the first concern due to the heterogeneity and autonomy of the data lake. Some relational schema learning approaches have been proposed for a single data model such as for key-value stores [8, 12, 45], for uncertain data [30]. These cannot be adapted to the lake with multi-model data sources because they fail to extract the same or similar semantics from heterogeneous concepts or classes in different sources. A recent work [50] focuses on storing multi-model data in RDBMS, which differs from our goal of supporting SQL queries and exploring richer cross-source answers. Secondly (C2), how to efficiently perform query answering based on an integrated relational schema to enhance the answer semantics of a SQL query over the data lake. Most works have been studied to augment data lake tables [13, 27], such as Infogather [48] is designed to obtain more complete answers via entity augmentation and attribute discovery. These are restricted to relational data and do not enrich semantics with a global schema. Additionally, some engines have been proposed to process distributed SQL, such as MuSQLE [20], Presto [38], CockroachDB [43] and Polaris [5], but do not consider that in a data lake with heterogeneous data models and query engines.

Main idea. To tackle these challenges, we present a schema integration-based query answering framework. Every source D_i maintains a local schema C_i, and a mediator M manages a global relational schema C_g that provides the schema associations between different sources, and the mappings from C_g to C_i. A SQL query is formulated based on C_g and submitted to the mediator M. The framework consists of *schema integration* and *query answering*, detailed in Section 3. We present classification-based method and an edge table-based approach to break semantic gap between different data sources, and then build the global relational schema to unify the semantics between the local schemas using a reinforcement learning approach (for C1). The global schema generation is considered a Markov decision process and is motivated by three types of actions and a novel reward function based on a Q-learning algorithm, introduced in Section 4. And then we present a query answering approach based on schema integration to enhance the answer semantics of SQL queries (for C2). Specifically, we enrich logical search spaces by bridging the schema gap between the query and the source schemas based on F_g. Semantic joining is proposed to establish semantic associations between data instances of different data sources to compute the final answers of the query, illustrated in Section 5. Finally, we experimentally verify that LakeAns can find the answers with greater coverage rate spending almost the same time with the methods performing local evaluation in a single data source, detailed in Section 6.

2 PROBLEM DEFINITION

2.1 Preliminary

Data lake. A data lake denoted by $D = \bigcup_{i=1}^{n} D_i$, collects a set of heterogeneous data sources, such as JSON documents, graphs, relational data and multimedia data (e.g., videos, images). Here we mainly concern three general data models JSON, graphs, and relational databases presented as follows. Unstructured multimedia data can be converted into structured scene graphs [47, 51] by extracting all objects and their relations from images and videos, and will be concerned in our future works.

JSON. A JSON object is a collection of key-value pairs $\{k_1 : a_1, ..., k_l : a_l\}$ that maps the key k_i to the value a_i. The JSON value can be a boolean value, a numeric value, a string value, an array and an object.

Graph. A graph is defined by $G = (V, E, L)$, where V is a set of nodes, and $E \subseteq V \times V$ is a set of edges. $L(v)$ and $L(e)$ represent the labels of $v \in V$ and $e \in E$, respectively.

Relational database. A *relation schema* R is associated with a set of *attributes* of the form $R(A_1, ..., A_k)$, where A_i is an attribute with *domain* $dom(A_i)$, and the attribute A_i of R is written as $R[A_i]$, $i \in [1, k]$. A *relation* I_R *of schema* R is a set of tuples with attributes $A_i(i \in [1, k])$ of R. A *database schema* is a pair $(\mathcal{R}, \Sigma)$, where $\mathcal{R} = (R_1, ..., R_n)$ is a finite set of *relation schemas*, and Σ is a set of *constraints*. A *relational database* $\mathcal{D}$ of $\mathcal{R}$ is $(D_1, ..., D_n)$ that satisfies all constraints in Σ, where D_i is a relation of schema R_i for $i \in [1, n]$.

These data models have heterogeneous schema semantics and query interfaces. Below we discuss the key challenges of unifying the data schemas and performing a general SQL query.

2.2 Problem Statement

Schema integration is an effective approach to bridge heterogeneous schema semantics and provide a unified query interface for the data lake. For example, C_1, C_2 and C_3 shown in Figure 3 are the local schemas of the relational database D_1, the JSON document D_2 and the graph D_3 in Figure 1, respectively. We formally present a relational schema graph as follows.

Relational schema graph. A *relational schema graph* is modeled as a property graph $S = (V_s, E_s, L_s, F_s)$, where V_s and E_s are a set of nodes and edges, respectively. L_s is a labeling function such that for each node $v \in V_s$, $L_s(v)$ is a node label. $F_s(v)$ is a set of attributes $\{A_1, ..., A_k\}$ for each node v.

In practice, a relational schema graph illustrates the relation schemas and their reference constraints in a database schema. The relation schema $R(A_1, ..., A_k)$ is specified by a node $v \in V_s$ whose label $L_s(v)$ is the relation name R, and $F_s(v)$ captures its attributes A_i for $i \in [1, k]$. Each edge indicates the semantic relationships among nodes. In this paper, an edge from u to v indicates that there is a reference from R_1 to R_2, where R_1 and R_2 are relation schemas encoded by u and v, respectively.

The local schema graph C_1 of D_1 shown in Figure 3 describes the relational database schema in Figure 1(c), which includes an edge connecting two relation schemas "User(name, location, buy)"

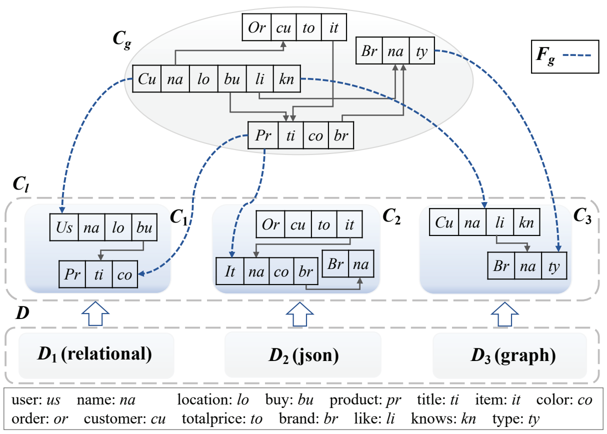


Figure 3: Architecture of the schema integration $\mathcal{S}$ for D consisting of C_g, C_l and F_g.

and "Product(title, color)". In Section 4.2, we will introduce how to create a local schema for different data sources.

To integrate heterogeneous data schemas and effectively support query answering across multiple data sources, we create a global relational schema graph, namely C_g shown in Figure 3. C_g is generalized from the local schemas of the heterogeneous data lake and stored in a mediator M, which will be detailed in Section 4.3. How to create an effective global schema C_g for D and efficiently support SQL queries based on it is our main concern in this paper.

To uniformly query the heterogeneous data lake, we study the query answering of SQL queries with SELECT-FROM-WHERE clauses. A SQL query Q is formulated based on the global relational schema C_g, and its answer after performing Q over the data lake is formally defined as follows.

Query answer. Given a SQL query Q over the data lake D, an answer for Q is a set of tuples with attributes $(A_1, ..., A_k)$, denoted by $T(Q)$, where (1) each A_i $(1 \leq i \leq k)$ corresponds to an attribute in the SELECT clause of Q; and (2) each attribute values is an instance value of D.

Note that (i) A_i may be an attribute of the relational database, a key of the JSON object and an edge label of the graphs; and (ii) the answer of Q follows a relational schema $R(A_1, ..., A_k)$, which may span multiple data sources of D and can be deduced from C_g.

EXAMPLE 2. *Figure 2 shows the answer of Q over the data lake shown in Figure 1. The answer $T(Q)$ spans three data sources, i.e., D_1, D_2 and D_3, and follows a schema R with attributes A = {"name", "knows", "title", "brand"} specified by the SELECT clause of Q. The schema R is deduced by joining the relation schemas "Customer" and "Product" based on C_g and projecting into the attribute set A.*

3 QUERY ANSWERING FRAMEWORK

Figure 4 illustrates our query answering framework (namely LakeAns), which aims at exploring the complete answers for SQL queries across multiple heterogeneous data sources in the data lake.

The LakeAns consists of two components: offline schema integration and online query answering.

Schema integration. As for a data lake D, we formulate a schema integration paradigm $\mathcal{S}$, which consists of a *global schema* C_g, a

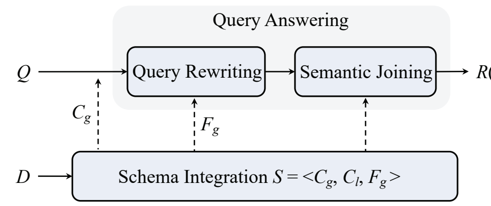


Figure 4: Overview of query answering framework.

set of *local schemas* C_l extracted from D, and a *mapping functor* F_g. C_l is built to bridge the schema gap between heterogeneous data sources. C_g provides global relational schema and uniformly general query interface for data scientists, which is created by integrating the local schemas based on a reinforcement learning approach. The F_g maps the concepts from C_g to C_l and benefits to query rewriting and across-sources access over the data lake. Detains will be presented in Section 4.

Query answering. Given a SQL query Q formulated based on C_g, it is answered based on the integrated schema to enhance the query semantics, as introduced in Section 5. We enrich logical plans of Q based on the local relational schema C_l and the mapping functor F_g. Thereafter, the local results computed from different data sources are required to perform *semantic joining* to deduce a relational schema based on the local schema C_l to obtain the final answers for Q.

4 SCHEMA INTEGRATION

In this section, a schema integration approach is developed to create a global relational schema graph to establish semantic associations across different data sources in the data lake.

4.1 Method Overview

Given a data lake $D = \{D_1, ..., D_m\}$, we define a schema integration paradigm as $\mathcal{S} = \langle C_l, C_g, F_g \rangle$. As shown in Figure 3, we have $C_l = \bigcup_{i=1}^{m} C_i$ and $F_g = \bigcup_{i=1}^{m} F_i$.

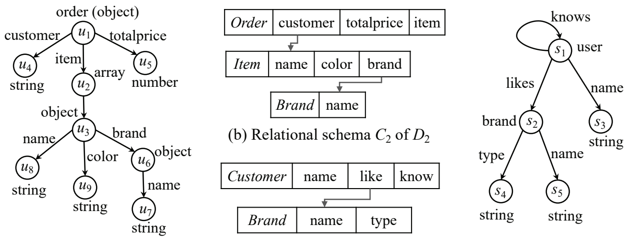
As for C_l, a classification-based method and an edge table-based approach are introduced to create the local relational schema for the JSON documents and the graphs, respectively. Details will be described in Section 4.2.

To unify the schema semantics among the local schemas, a global relational schema C_g is required. We consider the generation of C_g as a Markov decision process that aims to find the best relational schema (i.e., *state*) for D. This is done by performing a sequence of transformations (i.e., *actions*)) on the local schemas to efficiently support cross-source answering of SQL queries. Starting from the local schemas C_l, C_g is created utilizing a reinforcement learning approach motivated by three types of actions and a new reward function. Details will be introduced in Section 4.3.

In the integration paradigm $\mathcal{S}$, a query can be uniformly formulated by using C_g and completely answered based on F_g and C_l over the heterogeneous data sources D.

4.2 Local Schema Construction

For each heterogeneous data of D, local relational schema is constructed by doing the following two steps: (1) extracting a data schema, and (2) transforming it into relational database schema.



(a) Schema of C_2 (b) Relational schema C_2 of D_2 (c) Relational schema C_3 of D_3 (d) Schema of D_3

Figure 5: Example of local schema construction.

As for a relational database D_i, C_i is described by its database schema where nodes are specified by the relation schemas of D_i and edges represent the reference constraints among the relation schemas. C_1 shown in Figure 3 illustrates the relational schema of the data source D_1.

<u>For JSON documents.</u> Given a JSON document D_i, a classification-based approach is used to construct its corresponding relational schema. Firstly, we extract a tree-structured schema for D_i based on the JSON schema specification[1]. Secondly, we iteratively identify collection-like keys and tuple-like keys from the JSON schema. At each iteration, a unique relational schema is created by treating those tuple-like keys as attributes, and there is a reference to the relation schema encoded by the collection-like key.

Figure 5(a) illustrates the tree-structured JSON schema of D_2 shown in Figure 1(d), and its relational schema is shown in Figure 5(b). Starting from an object u_1, three tuple-like keys are identified, i.e., “customer", “totalprice" and “item", so that a relation schema is created as "Order(customer, totalprice, item)". Similarly, a schema “Item(name, color, brand)" is created by another object u_3 and its parent node u_2 labeled “item". There is an edge from the schema “Order" to “Item" because u_2 is a collection-like key represented by an array. Similar for the object u_6.

<u>For graphs.</u> Given a graph database D_i, an edge table-based approach is presented to create its relational schema graph C_i. Firstly, we summarize a graph schema by merging vertices with the same type and summarizing the relationships. Secondly, we extract a set of edge tables, each of which associates with a non-leaf vertex and a set of its adjacent edges. An edge table naturally forms a relational schema where the label of the vertex is relational name, and the labels of adjacent edges are attribute set. If there is an edge connecting two vertices, there is a reference between the two relational schemas encoded by them.

For example, Figure 5(d) is a graph schema of D_2 shown in Figure 1(e), and its relational schema is shown in Figure 5(c).

4.3 Global Schema Generation

In this section, we introduce a reinforcement learning method (namely LakeGrsg) to generate a global relation schema, which requires to augment the semantic associations across multiple data sources and to further efficiently support relational queries in the data lake.

4.3.1 Idea of LakeGrsg. Given a set of SQL queries and the local schema C_l of D, LakeGrsg aims to create a global relational schema graph C_g based on a Q-learning algorithm, such that these

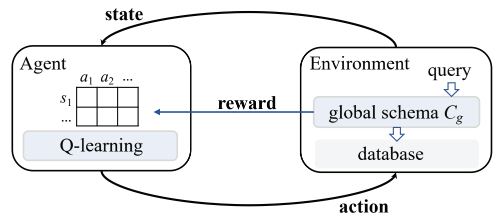


Figure 6: Overview of LakeGrsg for C_g over D.

queries can obtain the most richest answer semantics with the least cumulative query time.

We consider the generation of C_g as a *Markov decision process* defined by a quadruple (S, A, T, R), where S is a state space, A is a set of actions, T is a transition function linking state-action pairs to new states, and R is a reward function that reports a reward value for a state-action pair. In this paper, we design three types of *actions*, i.e., *linking*, *merging* and *splitting*. The *state* is a relational schema graph generated by taking the action. This process generates C_g by interacting an *agent* with an *environment*. Figure 6 illustrates the overview of LakeGrsg where the environment is a RDBMS. A Q-learning algorithm is used to take actions on the environment to maximize the expected rewards, which is recorded in a Q-table for action-state pairs. In each step, the agent greedily select the action with maximum expected rewards according to the Q-table and change the state of the environment. The state and reward are fed back into the agent to perform the next action.

Specifically, taking C_l as an initial state, the agent works in the following steps: for each iteration, (1) it selects an action a_t and creates a new state s_t; (2) the given SQL queries are performed based on the new state; and (3) it computes a reward r and updates the value of Q-table based on the generated state. This process is repeated until the maximum number of iterations is reached or a finite number of actions are tried, and we can finally obtain a global relational schema C_g for D.

The multi-model data is stored as relational tables based on C_l to support the execution of the given SQL queries. Furthermore, the functor F_g that records the mappings of concepts (i.e., relation names and attributes) between C_g and C_l would be updated when an action is taken in each iteration.

To enable to calculate the semantics of relations and instances, we project the string values into an embedding space based on the fastText database [26], which can handle out-of-vocabulary words by learning character embeddings. As for a relation schema $R(A_1, ..., A_k)$, $\vec{R}$ represents the embedding vector projected by the value of R. We utilize Π_R to represent the feature space associated with the instance values of R. Let $dom(A_i)$ be the instances of A_i in R, and $\Pi_R(A_i)$ be the embedding set of $dom(A_i)$. A *pivot vector* is used to represent the sample mean of $\Pi_R(A_i)$, which is calculated by the embedding mean of the values in $dom(A_i)$, denoted by $\vec{\mu}(A_i)$.

We next introduce the action identification and a Q-learning based state update.

4.3.2 Actions. Action space is defined by a set of transformations (i.e., actions) about the relational schemas motivated by the unique attributes in C_l. This means that when an attribute is selected, the

[1] https://json-schema.org

agent will determine how to transform the relational schema associated with it. Based on the transformation, we further define three types of actions: *linking*, *merging* and *splitting* to integrate the initial relation schema.

Linking actions are introduced to complement implicit references across multiple local schemas and unify the schema semantics of different data sources. Splitting and merging actions are developed to normalize schemas and make them to effectively support relational query processing. Before detailing these actions, we first describe how to identify whether an attribute A_i of relation schema R_1 can join with the attribute B_j in the relation schema R_2.

Given two relation schemas $R_1(A_1, ..., A_k)$ and $R_2(B_1, ..., B_l)$, R_1 and R_2 are semantically *joinable* on attributes A_i and B_j, $i \in [1, k]$, $j \in [1, l]$, if (a) A_i and B_j refer to the same or similar concept; and (b) A_i and B_j have similar value domains. Specifically, we first check how close are $\vec{A_i}$ and $\vec{B_j}$ in the embedding space. If the similarity is greater than a threshold α, we measure the distance of $\vec{\mu}(A_i)$ and $\vec{\mu}(B_j)$ using the cosine similarity. We say that R_1 and R_2 are semantically joinable on A_i and B_j if $cos(\vec{A_i}, \vec{B_j}) \geq \alpha$ and $cos(\vec{\mu}(A_i), \vec{\mu}(B_j)) \geq \theta$ where α and θ are the thresholds for concept similarity and pivot vector distance, respectively.

Linking action. There is a linking action on two joinable attributes $R_1[A_i]$ and $R_2[B_j]$, which inserts an edge connecting A_i of R_1 and B_j of R_2. Essentially, the linking action corresponds to complementing a foreign key reference between R_1 and R_2. Accordingly, F_g stores a pair of attributes $R_1[A_i]$ and $R_2[B_j]$, which are semantically equivalent concepts.

Merging action. To reduce schema redundancy and avoid frequent expensive join operations, we introduce merging action to combine the relation schemas. As for $R_1(A_1, ..., A_k)$ and $R_2(B_1, ..., B_l)$, R_1 is *contained* in R_2, denoted by $R_1 \sqsubseteq R_2$, if the following conditions are satisfied: (a) R_1 and R_2 have the same or similar relation name; and (b) for each attribute A_i of R_1, there is an attribute B_j in R_2 that is similar to its concept and domain. That is, we have $R_1 \sqsubseteq R_2$ if $cos(\vec{R_1}, \vec{R_2}) \geq \lambda$, $cos(\vec{A_i}, \vec{B_j}) \geq \alpha$ and $cos(\vec{\mu}(A_i), \vec{\mu}(B_j)) \geq \theta$, where λ is a distance threshold for the relation name.

In this case, a merging action is required to combine R_1 and R_2, which inserts all of the foreign key references of R_1 into the corresponding attributes of R_2. Therefore, F_g updates the mappings associated with R_1 to those of R_2.

Splitting action. To effectively support query processing and improve schema readability, we perform a splitting action when there are too many references included in a relation schema. Given a schema R, splitting action corresponds to a projection of R, which divides R as a schema set $P = \{P_1, ..., P_s\}$ such that (1) $A = \bigcup_{1 \leq i \leq s} P_i$ where A_i is the attribute set of P_i; (2) $\pi_{A_i}(I_R) = I_{P_i}$ for $i \in [1, s]$, and $I_{P_1} \bowtie ... \bowtie I_{P_s} = I_R$ where $\bowtie$ is the join operations of $I_{P_1}, ..., I_{P_s}$ in the embedding space Π_R; (3) There is a foreign key reference from $P_i[A_p]$ to $P_j[A'_p]$, $1 \leq i, j \leq s$, where A_p and A'_p are the joinable attributes for P_i and P_j.

The first and third conditions aim at dependency preservation and lossless joining for relation schemas, respectively. The second condition ensures that the join operation for each relation P_i of P does not lose the instance information in R. Accordingly, F_g updates the mappings associated with R to those of P_i, $1 \leq i \leq s$.

Algorithm 1: LakeGrsg

Input: a set of queries P, an action space A, the Q-table T_Q, and an initial state s_t;
Output: a new state s_{t+1}.
$m \leftarrow 0$; /** an episode **/
Select an action a_i;
while *A is not empty* **do**
  Compute $R(s_t, a_i)$ based on P;
  Obtain the next state s_{t+1} based on s_t and a_i;
  Compute $Q_{t+1}(s_t, a_i)$ and update $T_Q[t, i]$;
  if $m < Q_{t+1}(s_t, a_i)$ **then**
    $a_t \leftarrow a_i$;
    $m \leftarrow Q_{t+1}(s_t, a_i)$;
  $A \leftarrow A \setminus \{a_i\}$;
Update s_{t+1} based on a_t and s_t;
return s_{t+1};

4.3.3 State Update.

At the t-th step, the agent starts with the state s_t and greedily selects the action a_t with the maximum expected reward to generate a new state s_{t+1}.

A Q-table is used to record the expected reward for each action-state pair, which is a $(N \times N)$-dimensional table whose rows are states represented by all attributes associated with the relation schemas and columns are actions defined by A, where N is the size of A. At each step t, the agent selects an action a_t and update Q-value based on the reward r_t for a_t and s_t.

As for an action a_t and a state s_t, the reward function is designed to motivate the agent not only to explore the optimal global relational schema, but also to weight the semantic diversity of answers and the query complexity for the given query set. Consider the answers of query set returned by the environment, we define the reward function to be negatively correlated with the query time and positively correlated with the number of crossing answers. The reward $R(a_t, s_t)$ for a_t and s_t is computed as follows:

$$R(a_t, s_t) = \eta(\frac{d_t}{d_{t-1}} - 1) + (1 - \eta)(\frac{w_{t-1}}{w_t} - 1), \tag{1}$$

where (i) d_t (resp. d_{t-1}) is the number of crossing answers at the t-th (resp. $(t-1)$-th) iteration; (ii) w_t is the query time performing the given query set in the RDBMS at the t-th iteration; and (iii) η is a weight parameter, $0 \leq \eta \leq 1$.

The update of Q-value is based on the Bellman optimality equation, which is a weighted average of the current value and the new information, as introduced below:

$$Q_{t+1}(s_t, a_t) = (1-\beta) \cdot Q_t(s_t, a_t) + \beta \cdot [R(s_t, a_t) + \gamma \max_{a \in A} Q(s_{t+1}, a))], \tag{2}$$

where (i) $\beta \in [0, 1]$ is the learning rate that determines the ratio of accepting newly learned information; (ii) $R(s_t, a_t)$ is the reward from s_t to s_{t+1} by taking the action a_t; (iii) $\gamma \in [0, 1]$ is a discount factor that determines the influence of future reward, where the smaller γ the more short-term benefits are considered; and (iv) $\max_{a \in A} Q(s_{t+1}, a)$ is the maximum reward that can be obtained from state s_{t+1}, which is weighted by the learning rate β and the discount factor γ.

Algorithm 1 illustrates the process of state update and the details is described by the following example.

5 QUERY ANSWERING WITH INTEGRATION PARADIGM

In this section, we first present an overview of query answering pipeline, and then introduce how to attack the heterogeneity of the data lake to enhance the query semantics of Q based on the schema integration paradigm $\mathcal{S}$.

5.1 Query Answering Pipeline

Generally, distributed query optimizers [1, 21, 39] handle a SQL query in the following steps: (1) parsing the input query string and creates an abstract syntax tree; (2) compiling the created syntax tree to generate the logical search space where contains all logical equivalent alternative plans to execute the query; and (3) enumerating all physical distributed execution plans of these logical plans and choosing one with the least estimated cost.

To perform query answering over D, there are two problems should to be solved, as described below: (1) **heterogeneity of data schemas**, which may lead to incomplete logical plans and further produce incorrect answers; and (2) **semantic heterogeneity of data instances**, which will lead to incomplete query results computed by the physical plans.

Therefore, we present two key modifications to break the heterogeneity of schemas and instances in the data lake. Firstly, **logical plan enrichment** enriches the logical search space by bridging the schema gap between the query and the source schemas. Secondly, **semantic joining** builds semantic associations among data instances of different data sources.

5.2 Semantic Enhancement

To enhance the query semantics of Q, we present our logical plan enrichment and semantic joining based on the schema integration paradigm $\mathcal{S}$ as follows.

Logical plan enrichment. We bridge the semantic gap between the query and the local relational schema based on the mapping functor F_g. This is done by replacing the concept in the logical tree of Q as that of the local schemas.

For example, the class "Customer" in the query Q shown in Figure 2 should be projected into "User" in C_1 and "Customer" of C_3, as illustrated in Figure 3.

Semantic joining. After performing the distributed physical query plan over the naive database engines of D, a semantic join operation is presented to progressively join these local results. We need to address the heterogeneous *matches* that refer to the same entity, and further deduce a relation schema for them in terms of the local schema C_l.

Given a set of tuples T with relation schema $R(b_1, ..., b_m)$ in D_k and a subgraph G in D_l, we identify a match (u, t) for a tuple $t \in T$ and vertex $u \in G$ by checking whether they have the same or similar concepts and instance values. Recall that there is an edge from $R_1[A_i]$ to $R_2[B_j]$ in C_g if there is a reference from the foreign key A_i of R_1 to the primary key B_j of R_2. Let $F_g(u)$ represent the global concept of u in C_g. Therefore, u conceptually matches t if $F_g(u)$ and an attribute b of t in C_g are reachable. We then use the embeddings $\vec{L(u)}$ and $\vec{t.b}$ to compute their instance similarity for the condition (2). If the similarity is larger than a threshold, we call that t and u can be semantically joined via the joining key $t.b$.

Table 1: Datasets.

	Relational	JSON	Graph-vertex	Graph-relationship
Unibench	150 000	142 257	9 949	357 620
IMDB	494 295	84 390	113 858	833 178
DrugBank	940 812	-	648 415	1 487 091
DBLP	1 182 391	435 469	512 768	3 492 502

As for the match (u, t), we deduce a relation schema R_G with attributes $(b_1, ..., b_m, b_{m+1}, ..., b_n)$, such that t can be enriched with additional attributes A' of u extracted from the subgraph G, where $A' = \{b_i | m+1 \leq i \leq n\}$. Specifically, we first extract the relation schema R' with attributes A' for G based on C_l, and then semantically join R with R' via the joining key b to obtain R_G, i.e., $R_G = R \bowtie_b R'$. Thereafter, we populate R_G with the corresponding instance values to compute $T \bowtie G$.

6 EVALUATION

Using real-life datasets, we experimentally evaluate LakeAns for its efficiency and scalability. Our highlights are described as follows:

(1) Our schema integration approach LakeGrsg outperforms its competitors in terms of query time, the number of crossing answers and convergence speed.

(2) LakeAns can find the answers cross multiple sources spending almost the same time with the methods performing local evaluation in a single data source.

(3) LakeAns can efficiently perform cross-source queries based on the integrated relational schema in LakeGrsg, and has better scalability than its competitors in terms of data size and number of sources.

6.1 Experimental Setting

Datasets. We use four real-life datasets shown in Table 1: (1) Unibench, a multi-model e-commerce benchmark [52], in which the customer and product information are modeled as two relational tables including 150,000 tuples. Order information and transaction relationships are represented by a JSON document and a graph, respectively. The JSON contains 142,257 objects, and graph has 9,949 vertices and 375,620 edges. (2) IMDB: movie dataset in [10]. The relational data includes performing and rating information. The JSON stores film information and the graph is about cooperative information that indicates two actors have ever worked for the same movie and they are linked. (3) DrugBank, is extracted from DrugBank Central [3] and ChEMBL database [2], which includes 8 tables to present drug, target, ChEMBL information, etc. The interaction data of drugs and targets is modeled as a graph. (4) DBLP: publication data consisting of bibliography records in computer science. The publication records are presented in relational data. A subset of the paper information is represented in JSON. We also construct a co-authorship graph where two authors are connected if they publish at least one paper together.

Every dataset derive a local schema for each data source and generates a global relational schema that is maintained in the mediator using LakeGrsg. SQL queries are formulated based on the global schema and submitted at the mediator.

Query workload. We first generate a set of SQL queries for each dataset by randomly selecting classes from the global relational schema. The queries have SELECT and FROM clauses. We then extend these queries by adding references or filter conditions into the WHERE clause that relate to the classes of the SELECT clause. Here we denote the query set as Q_1, Q_2, Q_3, Q_4, Q_5 and Q_6, corresponding to 1J, 2J, 3J, 3J1S, 3J2S and 4J, respectively, where the query iJjS has i join conditions and j filter operations in the WHERE clause. The default one is Q_3.

Implementation. We develop a prototype system of LakeAns where the mediator employs a SQL server to maintain the global relational schema, and MongoDB, Neo4j and SQL server as the local database for the JSON, graph and relational data, respectively. As we discuss in Section 3, our algorithm mainly performs on the mediator and benefits from the technology of these local databases. We conduct our experiments on CentOS Linux 5.4 equipped with 3 Intel(R) Xeon(R) Silver 4110 CPUs, 30 GB memory.

Baselines. Most relational query baselines are limited in their ability to perform cross-source querying because they are restricted to a single data model, such as structured relational data. To ensure fair comparisons, we modify several existing methods to accommodate cross-source querying over the data lake as described below. (1) Naive performs the input SQL query on each local relational schema of the data source and then returns the final answer by directly matching the returned local results with the same attributes. Naive is barely able to find those results cross heterogeneous data sources and is mainly used to measure the semantic fusion capability of other methods. (2) InfoGather [48] considers indirect and direct matching, and exploits a holistic matching framework based on entity augmentation and attribute discovery to find the answer cross multiple web tables. InfoGather cannot support the query answering over the data lake including semi-structured and unstructured data. To adapt it to heterogeneous data lake, we load the dataset into relational tables complying with the local relational schemas created in Section 4.2. (3) Snowflake [9] is a shared-data distributed architecture that provides SQL extensions to traverse semi-structured and unstructured data. Snowflake cannot support global schema-based computation and it bridge heterogeneous semantics between different data sources using schema discovery. Similar to InfoGather, Snowflake focuses only on the enhancements of query semantics but neglects the optimal joining order of intermediate results, which is constrained by the SQL extensions.

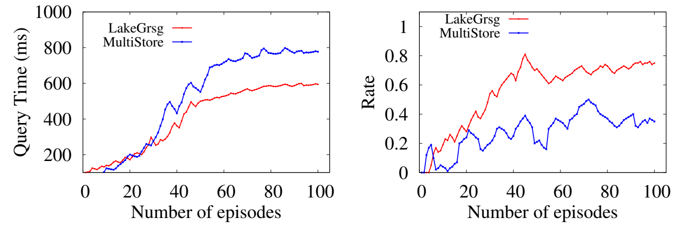
These baselines differ from our approach in that LakeAns requires to enhance query semantics over the data lake with multiple data models and heterogeneous query engines. To this end, LakeAns considers the global relational schema to bridge the schema heterogeneity of multiple data sources and the optimal join order of intermediate results to improve the query efficiency.

There is no baseline considers schema integration of heterogeneous data to efficiently support SQL queries across sources. We design the following baselines. (1) MultiStore [50] learns a relational schema to store multi-model data into RDBMS using reinforcement learning. It denotes all data as a set of two-column tables and aims to find the optimal join sequences to generate relational schema. MultiStore expects to obtain the schema with minimal query time but ignores the query semantics over the generated schema. (2) Two variants of LakeGrsg are designed by modifying the agent of LakeGrsg. LakeR_l is the agent removing merging and splitting actions. LakeR_{lm} is the agent removing the splitting action.

Metrics. To evaluate the performances of schema integration quantitatively, we use the following aspects: (1) the query time (2) the relative coverage rate of crossing answers in the final results, and (3) the expected reward of the given SQL queries. The relative coverage rate is computed as $Rate = |T_i - T_0|/T_0$, where T_i and T_0 are the answer set at the i-th iteration and the initial state, respectively.

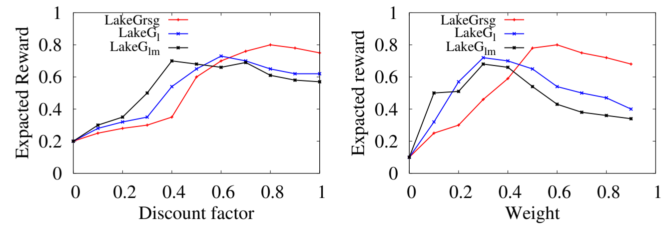
The evaluated metrics for query answering contain the query time and the relative coverage rate of crossing answer in terms of the answers computed by Naive.

Parameter selection. As for the LakeGrsg algorithm, we search hyper-parameters in the following values: the learning rate β in {0.0001, 0.0005, 0.001, 0.005, 0.01}, discount factor γ in {0, 0.2, 0.4, 0.6, 0.8, 1}, reward weight parameters η in {0, 0.2, 0.4, 0.6, 0.8, 1}. The similarity thresholds in actions are set λ in {0.2, 0.6, 0.8, 1}, α in {0.5, 0.6, 0.7, 0.8, 0.9} and θ in {0.5, 0.6, 0.7, 0.8, 0.9} for relation names (i.e., classes), attribute names and pivot vectors of attribute domains, respectively. The selected setting is that $\beta = 0.001$, $\gamma = 0.8$, $\eta = 0.6$, $\lambda = 0.6$, $\alpha = 0.8$, $\theta = 0.7$.



(a) Query time (b) Coverage rate

Figure 7: The changes in query time and coverage rate as the increasing of episodes.



(a) Discount factor (b) Weight parameter

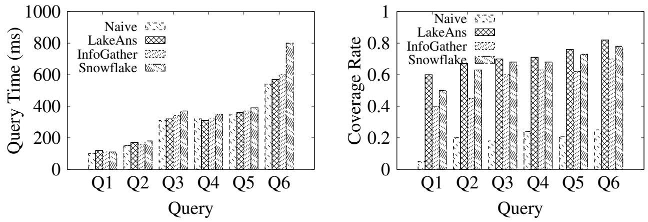
Figure 8: The expected reward with respect to the discount factor γ and weight parameter η.

6.2 Experimental Results

We next report our findings.

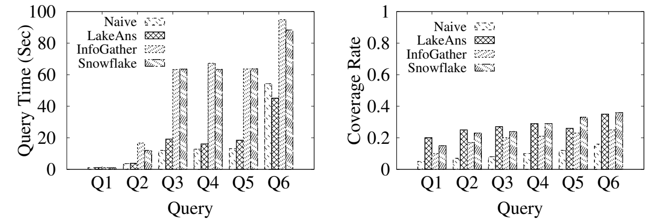
Exp-1: performance of schema integration. To evaluate the performance of schema integration algorithm, we concern two aspects: (1) the changes in query time and coverage rate as the increasing of episodes; and (2) the expected reward with respect to the discount factor γ and weight parameter η.

As reported in Figure 7(a), the query time of LakeGrsg is always smaller than that of MultiStore especially when the episode number is large. The reason is that (i) MultiStore creates a relational schema by finding a sequence of joins, which results in some relation schema including much foreign keys to increase the query cost. (2) Considering the foreign key of schemas, LakeGrsg defines merging and



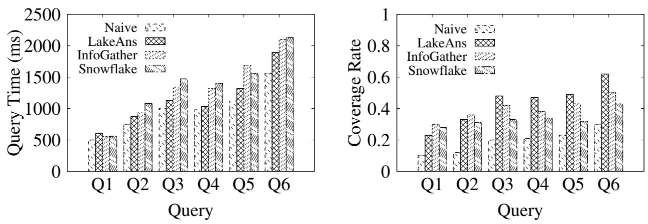
(a) Query time (b) Coverage rate

Figure 9: Impact of query in Unibench dataset.



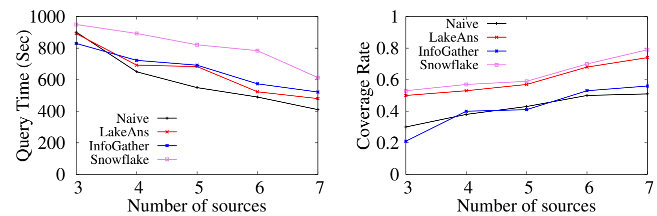
(a) Query time (b) Coverage rate

Figure 11: Impact of query in DrugBank dataset.



(a) Query time (b) Coverage rate

Figure 10: Impact of query in IMDB dataset.



(a) Query time (b) Coverage rate

Figure 12: Impact of source number.

splitting actions to simplify the relational schema to further avoid excessive joins on it. Figure 7(b) illustrates that LakeGrsg has higher coverage rate and converges earlier than MultiStore because our agent is powered by the query time and the number of crossing answer. However, the agent of MultiStore does not consider the answer semantics of cross-source queries.

Therefore, our schema integration approach LakeGrsg outperforms MultiStore in terms of query time, coverage rate and convergence speed.

Figure 8 shows that with the increasing of γ and η, LakeGrsg has more stable expected reward than LakeG$_l$ and LakeG$_l m$. This is because that (i) LakeGrsg exploits linking action to bridge the semantics between different data sources, so that the crossing answers can be found, and (ii) it performs merging and splitting actions to simplify relational schema, so that the query time can be reduced by avoiding excessive joins. That means that the three actions in the agent of LakeGrsg helps it to obtain more stable long-term benefits.

Exp-2: performance of query answering. To evaluate the impact of different queries, we vary the query from $Q1$ to $Q6$ and then apply it to LakeAns as well as the comparison algorithms. As for each query, we both report the impact on query time and coverage rate on different datasets.

Figure 9(a) shows that the execution time of LakeAns, InfoGather and Snowflake increases with the increasing of join condition number, while the execution time of LakeAns is close to that of Naive and smaller that of than InfoGather and Snowflake. More importantly, LakeAns has much smaller upward trend than them. Especially, LakeAns performs much better than InfoGather and Snowflake on large datasets, as we can see from the IMDB and DrugBank datasets shown in Figures 10(a) and 11(a). This is because that: (i) InfoGather aims to find an holistic match for a query by considering entity enhancement and attribute discovery. However, LakeAns is a schema-based approach that answers queries based on a global relational schema and the mappings from the global to the local. (ii) Snowflake utilizes SQL extensions to answer cross-source queries, which progressively traverses the bindings of attributes and entities. LakeAns benefits form an integrated schema and can effectively bridge the semantic associations between different data sources to perform cross-source querying.

Figure 9(b) illustrates the following two aspects. On the one hand, the coverage rate of LakeAns and Snowflake are larger than that of Naive and InfoGather. In most cases, the coverage rate of LakeAns is greater than or equal to that of the other methods, as we can see from the IMDB and DrugBank datasets shown in Figures 10(b) and 11(b). On the other hand, the coverage rate of LakeAns, InfoGather and Snowflake all increase with the increasing of join condition number in the query, while LakeAns significantly has a much larger upward trend than that of Naive and InfoGather in different datasets.

Totally, LakeAns spends almost the same time with Naive that performs local evaluation in a single data source to find the answers with greater coverage rate than other methods.

Exp-3: impact of source numbers. To evaluate the scalability, we vary the source number from 3 to 7, which is done by partitioning the DBLP dataset into several fragments.

As illustrated in Figure 12, LakeAns outperforms other methods in terms of scalability, yielding much smaller execution time than InfoGather and Snowflake, and greater coverage rate than Naive and InfoGather. With the increasing of source number, all execution times decrease and coverage rate improve. As the number of sources increases, all execution times decrease and the coverage rate slowly increases. Furthermore, LakeAns can obtain twice the coverage rate of the answers obtained by Naive and InfoGather, as reported in Figure 12(b). This is because that LakeAns finds crossing answers in the data lake as much as possible by leveraging an integrated relational schema.

All the above experimental results justify that our LakeAns framework is efficient and effective. It scales well on all metrics and has high coverage rate of answers across multiple data sources. Moreover, the integrated relational schema learned by LakeGrsg can effectively and efficiently support cross-source querying and enhance the answer semantics for a heterogeneous data lake.

7 RELATED WORK

We categorize the related work as follows.

Schema Matching. Some problems in data integration, such as entity resolution [16, 31], ontology alignment [28, 35], schema matching [17, 40] and dataset linking [19] have attracted significant interests in recent years.

The most related work to our schema integration is schema matching. Given a set of datasets and their schemas, schema matching is the problem of discovering potential correspondences between concepts of different datasets and it is one of the most important prerequisite steps for analyzing heterogeneous data collections [40]. State-of-the-art schema matching algorithms use simple schema- or instance-based similarity measures to struggle with finding matches beyond the trivial cases. Semantics-based algorithms require the use of domain-specific knowledge encoded in a knowledge graph or an ontology. For example, Data Tamer [41] match attributes using a variety of similarity measures and algorithms called experts. The Data Civilizer system [10] uses a linkage graph to support data discovery, while Aurum [18] builds knowledge graphs where different datasets are correlated with respect to their content or schema. All of these methods rely on similarity computation, e.g. Jaccard Similarity, value distribution. [53] and [19] build relationships between concepts of different databases using cluster-based matching algorithm and a given ontology. As a result, schema matching depends on external knowledge [19] such as domain-specific ontologies, and still remains a largely manual process.

These efforts differ from ours in that we generate a global relational schema for heterogeneous data sources by using a reinforcement learning method. This does not require external knowledge, but rather interaction with a relational database to find the schema with maximum expected reward.

Relational Schema Mapping. There has been a growing interest in mapping various data types into relational database to be able to reuse mature robust relational database technology. The choice of relational schema design plays a crucial role for efficiency. Different designs not only imply different kinds of physical data partitions, but also different translations to SQL operations. Our global relational schema generation relates closely to store the multi-model data to relational databases [50], but it is not designed for supporting the unified SQL query. Additionally, relational mapping has been studied in semi-structured data [11, 12, 44], RDF [4, 37] and uncertain data [15, 30].

The most straightforward of semi-structured to relational mapping earlier efforts provide generic mapping rules that do not necessitate upfront analysis of the input dataset [11]. Some other works present a mapping strategy based on structural analysis of the input dataset [11, 12, 49]. They design schemas for XML datasets by analyzing a graph of the DTD elements present in the input data and a set of heuristics is used to dictate whether an element should be materialized as its own table or linked within a parent element's table. Investigations of various schema designs form RDF data to relational have been started by [4, 36]. Recently, for a single-node system, the analysis provided in [37] gives arguments for using an emergent schema. [6] obtains the relational schemata using Apache Hadoop as the distributed processing platform and mapping SPARQL queries into Spark SQL. There are two major and complementary approaches to dealing with uncertainty in data [15]. In general, probability theory is quantitative with more precise outcomes, but these come at the price of acquiring actual probabilities and high computational complexities in managing them [42]. However, if data uncertainty does come with meaningful probability values, a probabilistic model is more appropriate if it can be managed with feasible resources. In recent work, a survey of practical methods for constructing possibility distributions was given [15]. Research on probabilistic databases has focused on queries [42]. Typical is the desire to extend trusted relational technology to handle uncertainty.

Data Lake Discovery. [33] identifies the challenges and opportunities in a data lake. Aurum [18] discovers syntactic relationships between datasets in a graph data structure and supports keyword search and similar content search. [34] defines the table union search problem on open data, which proposes value set, class semantic and embedding similarity to determine the attribute union-ability. Skluma [7] extracts diverse embedded metadata from files based on a probabilistic pipeline and allows the topic-based discovery. Constance [23] exploits the semantic annotations of data sources to enriches metadata, which can be accessed by a template-based query. Some approaches also navigate dataset based on the linkage graphs [10] or version graphs [25]. Recently, a new organization structure based on the Markov probabilistic model is proposed such that users can navigate a data lake more effectively [32]. Josie [55] finds some datasets in the data lake that can be joined with a given table, which is transformed into a overlap set similarity problem. Juneau [54] finds additional tabular or nested data for training or validation from computational notebooks (e.g., Jupyter), workflows and cells. More related works for knowledge discovery in a data lake can be found in surveys [24, 33].

These works have different goals with our work, and we aim at exploring complete answers for SQL queries across multiple heterogeneous data sources.

8 CONCLUSION

In this paper, we study query answering over a heterogeneous data lake, which aims to enhance answer semantics of cross-source relational querying. To this end, a novel schema-based framework LakeAns is proposed. LakeAns integrates local schemas to create a global relational schema based on a reinforcement learning approach to effectively support the query answering and to find answers across multiple data sources. Our experimental evaluation verify that this framework is promising for finding more answers cross multiple data sources with minimum execution time. In future works, we will extend our work to handle unstructured multimedia data in a heterogeneous data lake.

REFERENCES

[1] [n.d.]. Azure Synapse Analytics. https://azure.microsoft.com/en-us/services/synapse-analytics/.

[2] 2016. ChEMBL: A Database of Bioactive Drug-like Small Molecules. https://www.ebi.ac.uk/chembl.

[3] 2017. DrugCentral. http://drugcentral.ca.

[4] Daniel J Abadi, Adam Marcus, Samuel R Madden, and Kate Hollenbach. 2007. Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd international conference on Very large data bases*. 411–422.

[5] Josep Aguilar-Saborit, Raghu Ramakrishnan, Krish Srinivasan, Kevin Bocksrocker, Ioannis Alagiannis, Mahadevan Sankara, Moe Shafiei, Jose Blakeley, Girish Dasarathy, Sumeet Dash, et al. 2020. POLARIS: the distributed SQL engine in azure synapse. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3204–3216.

[6] Victor Anthony Arrascue Ayala, Polina Koleva, Anas Alzogbi, Matteo Cossu, Michael Färber, Patrick Philipp, Guilherme Schievelbein, Io Taxidou, and Georg Lausen. 2019. Relational schemata for distributed SPARQL query processing. In *Proceedings of the International Workshop on Semantic Big Data*. 1–6.

[7] Paul Beckman, Tyler J. Skluzacek, Kyle Chard, and Ian Foster. 2017. Skluma: A Statistical Learning Pipeline for Taming Unkempt Data Repositories.

[8] Craig Chasseur, Yinan Li, and Jignesh M Patel. 2013. Enabling JSON Document Stores in Relational Systems.. In *WebDB*, Vol. 13. 14–15.

[9] Benoit Dageville, Thierry Cruanes, Marcin Zukowski, Vadim Antonov, Artin Avanes, Jon Bock, Jonathan Claybaugh, Daniel Engovatov, Martin Hentschel, Jiansheng Huang, et al. 2016. The snowflake elastic data warehouse. In *Proceedings of the 2016 International Conference on Management of Data*. 215–226.

[10] Dong Deng, Raul Castro Fernandez, Ziawasch Abedjan, Sibo Wang, Michael Stonebraker, Ahmed K Elmagarmid, Ihab F Ilyas, Samuel Madden, Mourad Ouzzani, and Nan Tang. 2017. The Data Civilizer System.. In *Cidr*.

[11] Alin Deutsch, Mary F. Fernández, and Dan Suciu. 1999. Storing Semistructured Data with STORED. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, Alex Delis, Christos Faloutsos, and Shahram Ghandeharizadeh (Eds.). ACM Press, 431–442.

[12] Michael DiScala and Daniel J Abadi. 2016. Automatic generation of normalized relational schemas from nested key-value data. In *Proceedings of the 2016 International Conference on Management of Data*. 295–310.

[13] Yuyang Dong, Kunihiro Takeoka, Chuan Xiao, and Masafumi Oyamada. 2021. Efficient Joinable Table Discovery in Data Lakes: A High-Dimensional Similarity-Based Approach. In *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*. IEEE, 456–467.

[14] Huizhong Duan and ChengXiang Zhai. 2015. Mining coordinated intent representation for entity search and recommendation. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*. 333–342.

[15] Didier Dubois and Henri Prade. 2016. Practical methods for constructing possibility distributions. *International Journal of Intelligent Systems* 31, 3 (2016), 215–239.

[16] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1454–1467.

[17] Jérôme Euzenat, Pavel Shvaiko, et al. 2007. *Ontology matching*. Vol. 18. Springer.

[18] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A data discovery system. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 1001–1012.

[19] Raul Castro Fernandez, Essam Mansour, Abdulhakim A Qahtan, Ahmed Elmagarmid, Ihab Ilyas, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2018. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 989–1000.

[20] Victor Giannakouris, Nikolaos Papailiou, Dimitrios Tsoumakos, and Nectarios Koziris. 2016. MuSQLE: Distributed SQL query execution over multiple engine environments. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 452–461.

[21] Goetz Graefe. 1995. The Cascades Framework for Query Optimization. *IEEE Data Eng. Bull.* 18, 3 (1995), 19–29.

[22] Yangyang Guo, Zhiyong Cheng, Liqiang Nie, Yinglong Wang, Jun Ma, and Mohan Kankanhalli. 2019. Attentive long short-term preference modeling for personalized product search. *ACM Transactions on Information Systems (TOIS)* 37, 2 (2019), 1–27.

[23] Rihan Hai, Sandra Geisler, and Christoph Quix. 2016. Constance: An Intelligent Data Lake System.

[24] Rihan Hai, Christoph Quix, and Matthias Jarke. 2021. Data lake concept and systems: a survey. *arXiv preprint arXiv:2106.09592* (2021).

[25] Joseph M. Hellerstein, Vikram Sreekanti, Joseph E. Gonzalez, James Dalton, Akon Dey, Sreyashi Nag, Krishna Ramachandran, Sudhanshu Arora, Arka Bhattacharyya, Shirshanka Das, Mark Donsky, Gabriel Fierro, Chang She, Carl Steinbach, Venkat Subramanian, and Eric Sun. 2017. Ground: A Data Context Service. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*.

[26] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. 2017. Bag of Tricks for Efficient Text Classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, Mirella Lapata, Phil Blunsom, and Alexander Koller (Eds.). 427–431.

[27] Aamod Khatiwada, Roee Shraga, Wolfgang Gatterbauer, and Renée J. Miller. 2022. Integrating Data Lake Tables. *Proc. VLDB Endow.* 16, 4 (2022), 932–945.

[28] Prodromos Kolyvakis, Alexandros Kalousis, and Dimitris Kiritsis. 2018. Deepalignment: Unsupervised ontology matching with refined word vectors. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. 787–798.

[29] Lizi Liao, Le Hong Long, Zheng Zhang, Minlie Huang, and Tat-Seng Chua. 2021. MMConv: an environment for multimodal conversational search across multiple domains. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 675–684.

[30] Sebastian Link and Henri Prade. 2016. Relational database schema design for uncertain data. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1211–1220.

[31] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *Proceedings of the 2018 International Conference on Management of Data*. 19–34.

[32] Fatemeh Nargesian, Ken Q. Pu, Erkang Zhu, Bahar Ghadiri Bashardoost, and Renée J. Miller. 2020. Organizing Data Lakes for Navigation. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier, Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). 1939–1950.

[33] Fatemeh Nargesian, Erkang Zhu, Renée J Miller, Ken Q Pu, and Patricia C Arocena. 2019. Data lake management: challenges and opportunities. *Proceedings of the VLDB Endowment* 12, 12 (2019), 1986–1989.

[34] Fatemeh Nargesian, Erkang Zhu, Ken Q Pu, and Renée J Miller. 2018. Table union search on open data. *Proceedings of the VLDB Endowment* 11, 7 (2018), 813–825.

[35] Lorena Otero-Cerdeira, Francisco J Rodríguez-Martínez, and Alma Gómez-Rodríguez. 2015. Ontology matching: A literature review. *Expert Systems with Applications* 42, 2 (2015), 949–971.

[36] Zhengxiang Pan and Jeff Heflin. 2004. *Dldb: Extending relational databases to support semantic web queries*. Technical Report. LEHIGH UNIV BETHLEHEM PA DEPT OF COMPUTER SCIENCE AND ELECTRICAL ENGINEERING.

[37] Minh-Duc Pham and Peter Boncz. 2016. Exploiting emergent schemas to make RDF systems more efficient. In *The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part I 15*. Springer, 463–479.

[38] Raghav Sethi, Martin Traverso, Dain Sundstrom, David Phillips, Wenlei Xie, Yutian Sun, Nezih Yegitbasi, Haozhun Jin, Eric Hwang, Nileema Shingte, et al. 2019. Presto: SQL on everything. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1802–1813.

[39] Srinath Shankar, Rimma V. Nehme, Josep Aguilar-Saborit, Andrew Chung, Mostafa Elhemali, Alan Halverson, Eric Robinson, Mahadevan Sankara Subramanian, David J. DeWitt, and César A. Galindo-Legaria. 2012. Query optimization in microsoft SQL server PDW. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*, K. Selçuk Candan, Yi Chen, Richard T. Snodgrass, Luis Gravano, and Ariel Fuxman (Eds.). ACM, 767–776.

[40] Roee Shraga, Avigdor Gal, and Haggai Roitman. 2020. Adnev: Cross-domain schema matching using deep similarity matrix adjustment and evaluation. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1401–1415.

[41] Michael Stonebraker, Daniel Bruckner, Ihab F Ilyas, George Beskales, Mitch Cherniack, Stanley B Zdonik, Alexander Pagan, and Shan Xu. 2013. Data curation at scale: the data tamer system.. In *Cidr*, Vol. 2013.

[42] Dan Suciu, Dan Olteanu, Christopher Ré, and Christoph Koch. 2011. Probabilistic databases. *Synthesis lectures on data management* 3, 2 (2011), 1–180.

[43] Rebecca Taft, Irfan Sharif, Andrei Matei, Nathan VanBenschoten, Jordan Lewis, Tobias Grieger, Kai Niemi, Andy Woods, Anne Birzin, Raphael Poss, et al. 2020. Cockroachdb: The resilient geo-distributed sql database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1493–1509.

[44] Daniel Tahara, Thaddeus Diamond, and Daniel J. Abadi. 2014. Sinew: a SQL system for multi-structured data. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 815–826.

- [45] Igor Tatarinov, Stratis D Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. 2002. Storing and querying ordered XML using a relational database system. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*. 204–215.
- [46] Xiaohui Xie, Jiaxin Mao, Yiqun Liu, Maarten de Rijke, Haitian Chen, Min Zhang, and Shaoping Ma. 2020. Preference-based evaluation metrics for web image search. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 369–378.
- [47] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. 2017. Scene Graph Generation by Iterative Message Passing. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 3097–3106.
- [48] Mohamed Yakout, Kris Ganjam, Kaushik Chakrabarti, and Surajit Chaudhuri. 2012. Infogather: entity augmentation and attribute discovery by holistic matching with web tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. 97–108.
- [49] S Hassas Yeganeh, Oktie Hassanzadeh, and Renée J Miller. 2011. Linking semistructured data on the web. *Interface* (2011).
- [50] Gongsheng Yuan, Jiaheng Lu, Shuxun Zhang, and Zhengtong Yan. 2021. Storing Multi-model Data in RDBMSs based on Reinforcement Learning. In *CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, Virtual Event, Queensland, Australia, November 1 - 5, 2021*, Gianluca Demartini, Guido Zuccon, J. Shane Culpepper, Zi Huang, and Hanghang Tong (Eds.). ACM, 3608–3611.
- [51] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. 2018. Neural Motifs: Scene Graph Parsing With Global Context. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*. Computer Vision Foundation / IEEE Computer Society, 5831–5840.
- [52] Chao Zhang. 2018. Parameter Curation and Data Generation for Benchmarking Multi-model Queries. In *Proceedings of the VLDB 2018 PhD Workshop co-located with the 44th International Conference on Very Large Databases (VLDB 2018), Rio de Janeiro, Brasil, Aug 27-31, 2018*, Senjuti Basu Roy and Altigran Soares da Silva (Eds.), Vol. 2175.
- [53] Meihui Zhang, Marios Hadjieleftheriou, Beng Chin Ooi, Cecilia M Procopiuc, and Divesh Srivastava. 2011. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 109–120.
- [54] Yi Zhang and Zachary G. Ives. 2020. Finding Related Tables in Data Lakes for Interactive Data Science. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*. ACM, 1951–1966.
- [55] Erkang Zhu, Dong Deng, Fatemeh Nargesian, and Renée J. Miller. 2019. JOSIE: Overlap Set Similarity Search for Finding Joinable Tables in Data Lakes. In *Proceedings of the 2019 International Conference on Management of Data, SIGMOD Conference 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*, Peter A. Boncz, Stefan Manegold, Anastasia Ailamaki, Amol Deshpande, and Tim Kraska (Eds.). 847–864.