Querying the Deep Web Databases using DeepQ

A Proposal

Presented in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

with a

Major in Computer Science

in the

College of Graduate Studies

University of Idaho

by

Amal Aljohani

Major Professor: Hasan Jamil, Ph.D.

Committee Members: Clinton Jeffery, Ph.D. ; Xiaogang Ma, Ph.D. ; Jia Song,

Ph.D. Department Administrator: Terence Soule, Ph.D.

November 2021
AUTHORIZATION TO SUBMIT

DISSERTATION

This dissertation of Amal Aljohani, submitted for the degree of Doctor of Philosophy with a major in Computer Science and titled “DeepQ: A System to Peek inside the eCommerce Deep Web,” has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College of Graduate Studies for approval.

Major Professor    _______________________________ Date ____________
Hasan Jamil, Ph.D.

Committee Members

______________________________ Date ____________
Clinton Jeffery, Ph.D.

______________________________ Date ____________
Xiaogang Ma, Ph.D.

______________________________ Date ____________
Jia Song, Ph.D.

Department Administrator

______________________________ Date ____________
Terence Soule, Ph.D.
Abstract

The Deep Web is large, and its databases contain enormous amounts of high quality content that can provide great value to the user if accessed properly. Search engines such as Google can only find indexed information that are surfaced to the *Shallow Web*. In contrast, peeking into *Deep Web* databases is not possible for search engines such as Google. Search engines are not able to simulate SQL-like queries of database contents in a traditional or other intuitive human-like manner. In this work, we present a system called *DeepQ* to search the database contents behind the firewalls inside the deep web and show that these contents can be accessed using a structured query language treating them as a deep relational web. A recent proposed declarative deep web query language called DQL is leveraged and we present the contours of its implementation in the *DeepQ* system. We also believe that this work has the potential to demonstrate the Universal Relational model as the user will be able to interact freely with databases that are hidden behind firewalls without needing to know their schema.
## Table of Contents

Authorization to Submit Thesis ii  
Abstract iii  
Table of Contents iv  
List of Figures vi  

1 **Introduction** 1  
   1.1 Shallow Web and Deep Web 2  
   1.2 Why do we need to search the Deep Web? 3  
   1.3 Why is it difficult to search the Deep Web? 3  
   1.4 Research Problem 4  

2 **Related Work** 7  
   2.1 Deep Web Source Integration 8  
   2.2 Web Data Integration 9  
   2.3 UR Model for the Deep Web Search 11  

3 **Our Work** 14  
   3.1 Motivation and Objectives 14  
   3.2 Hypotheses 14  
   3.3 Work Contribution 15  
   3.4 Methodology 15  
   3.5 Leveraging the DQL features 15  
   3.6 DeepQ Architecture 17  
      3.6.1 Pick-and-Filter Interface 19  
      3.6.2 Event-Driven Resource Indexing 21
3.7 System Overview .................................................. 22
  3.7.1 Deep Web forms ............................................. 23
  3.7.2 The CGI Script and related databases ......................... 24
  3.7.3 User Interface .............................................. 26
3.8 Single Domain web databases( Simple Query)- An Example ....... 26
3.9 Results .................................................................. 28
  3.9.1 Single Domain Results - Simple Query .......................... 28
3.10 Future Plans ....................................................... 30
  3.10.1 Multiple Domains Web Databases - Complex Query Imple-
         mentation ......................................................... 30
  3.10.2 More Features .................................................. 31
3.11 Performance and Evaluation Criteria .............................. 31
3.12 Problems and limitations ......................................... 33
3.13 Conclusion ........................................................ 34

Appendix A: Name ..................................................... 35

References ............................................................. 35
List of Figures

1.1 Google response to query $Q_1$. ............................................. 5
1.2 Expansion of the top link for $Q_1$ showing a partial list of available Honda Civics. ............................................. 6
1.3 Manual query results of $Q_1$ at autotrader.com. .................. 6

3.1 DeepQ architecture. .......................................................... 17
3.2 Pick-and-Filter and Direct type deep web interfaces. .............. 20
3.3 DeepQ System Overview for a single domain. ....................... 23
3.4 Form 4 after submitting as an example ................................. 24
3.5 Form 2 after submitting as an example ................................. 25
3.6 Input/Output Attributes for each form. ............................... 25
3.7 Graphical Interface (restricted) ............................................ 27
3.8 Text Interface (restricted) .................................................. 27
3.9 Graphical Interface Query Result ....................................... 29
3.10 Text User Interface Query Result ...................................... 29
3.11 DeepQ System Overview for the multiple domain querying. .... 30
3.12 DeepQ performance measures. ......................................... 32
3.13 Databases and Search Engines. ........................................ 33
3.14 Some System Strengths and Limitations. ............................. 33
3.15 Comparison of System Capabilities, Identification (Source Identification), Classification, Integration (Interface or data Integration), UUQI(Unified User Query Interface), SM (Schema Matching), FF (Form Filling), Results Wrapping, Reusability. ............................................. 34
3.16 Operators and implementation plan. .................................. 35
Chapter 1

Introduction

Today, the World Wide Web (also referred to as the web) is the leading infrastructure for information presenting and publishing and it has become the most important platform for e-commerce and business transactions. The number of web pages is growing exponentially, and this massive growth has caused the web to evolve into a data-rich repository [28]. The web holds millions of searchable data sources, and a large portion of them are hidden behind firewalls and can only be accessed through Query Interfaces (or Web Forms). These query interfaces are the main means to retrieve web database contents; they also provide a glance into the underlying database structure. Users are allowed to submit queries through a query interface to the web database and obtain results as dynamically generated responses from those databases.

The Web contains structured, semi structured, and unstructured content. The structured content that resides in the web databases are called the Deep Web or Hidden Web [13]; thus, the deep web contains structured data that exists in dynamic generated web pages [11]. It is also important to note that obtaining the content of web databases requires that users pose structured queries through Web forms. Web databases are usually domain specific, and it is impossible for one web database to carry all information about that specific domain. Thus, users usually search in different web databases, even if they are trying to get one result from a specific domain.

We believe that ordinary users should be able to pose simple declarative queries that can obtain useful, accurate, and relevant results whether the domain is specific or not. An example will be discussed later in section 1.4 but before that it is important to answer the following questions:
• What is the difference between the shallow web and the deep web?

• Why do we need to search the Deep Web?

• Why is it difficult to search the Deep Web?

1.1 Shallow Web and Deep Web

The term *Shallow Web* (also known as *Surface Web*) refers to the web pages that are indexed by some search engines. In contrasts, the term *Deep Web* (also known as *Hidden Web*) has been used in the past by many authors to refer to non-indexed web pages. There is also the *Dark Web* that should not be confused with the Deep Web as it refers to some web pages that use encryption software such as *Tor* to hide the location and IP addresses [20]. Web contents can be categorized into two types: *static*, and *dynamic* based on their impact on crawlers. If some or all content of a web page is generated at run time, it defined as a dynamic web page. More than 80% of web content is dynamically generated [36]. In contrast, the web contents that exist on the server and are transmitted after receiving a client request are defined as static web pages. The web in general is dynamically growing, and the size of the indexable web is huge and rapidly increasing. However, the size of the hidden web is much larger [30], and in 2012 this size reached several trillions of web pages [20]. The focus in this work is on the traditional definition of the term “Deep Web” that was proposed in 2001 by Bergman [6]. Deep web refers to the dynamically generated pages that come in response to submitting a query via web forms. It contains a tremendous amount of useful and high-quality information that can only be obtained after submitting a query via web form. In fact, the dynamic query-based data access is provided through query interfaces that provides access to lots of online databases. In other words, the dynamically generated pages that have the structured data come as responses to a specific query. Deep web content can neither be seen nor retrieved
1.2 Why do we need to search the Deep Web?

The deep web carries huge amounts of hidden information that would be very beneficial if accessed and leveraged properly. Traditionally, users are restricted to accessing the content that is provided by search engines from the shallow web. The probability of finding the desired information in the deep web is much higher considering its size and richness [6]. Even though hidden information represents a large proportion of the web, finding and accessing such data through search engines is often impossible [1]. Usually, deep web crawlers reach web pages and process them via information extractors that provide structure to the information in order to facilitate its integration in an automated manner [20]. There are many useful databases on the web, but users find it difficult to find the right sources and query them. As it is not useful to query only one deep web site at a time, it is more efficient to integrate data from multiple sources [26]. Researchers are cognizant of the fact that it is difficult to query the deep web and integrate it because of access limitations [11]. Many research projects aim to empower users with the ability to access databases on the web effectively by proposing new approaches and building systems that can make the deep web more accessible and usable [17]. The fact that these databases are hidden behind firewalls does not invalidate the fact that they are actually databases that can be searched in a more structured way.

1.3 Why is it difficult to search the Deep Web?

In general, web search will only continue to grow [2]. Not only is deep web data not indexed by search engines, but also their URLs are not defined. Besides, large portions of the hidden web were not explored due to the difficulties associated with
finding and accessing hidden web data. The large scale of the dynamically generated content is one of the problems that make deep web crawling a challenge. Also, search interface restrictions make accessing the underlying databases difficult [36]. In fact, accessing hidden web data requires not only finding the right web forms as sources, but also filling those forms properly [40], processing the queries, and integrating all data. Moreover, the tasks of query translation (across different deep web sources), data retrieval, and integration should be done on-the-fly to achieve large-scale data integration. Searching the deep web is considered challenging especially when it comes to coping with the large scale since deep web integration is dynamic, and it is not configured statically for such integration [17]. It has been known that accessing the deep web is demanding as it poses many technical challenges. Additionally, it requires combining multiple techniques from different areas of computer science [10]. There exists a gap between the theoretical understanding of the problem of querying the deep web and the practical approaches to it [12]. Even the simplest query on the deep web requires the execution of recursive query plans.

1.4 Research Problem

Search engines such as Google do not allow users to peek into the databases behind web forms. These engines can only access information that resides in the Shallow Web. Consider for example that Nina is searching online for a car. She submits the following query to Google:

\[ Q_1: \text{Red Honda Civic ex 2015 less than } \$5,000 \text{ Moscow Idaho} \]

This query returns astonishing an 67,800 results, but none of them precisely responded to the query as shown in Figure 1.1. Google’s response was either completely wrong and did not meet query conditions, or partially met some of the query conditions. After clicking on the topmost recommended link as shown in Figure 1.2,
we can see that none of the cars fully satisfy the query conditions. Most responses ignored the location condition, and the other responses are actually commercial leads for businesses. These leads might be generated using text to approximate query translation APIs to find a possible match in a database that is close enough [8] and not truly responsive to the user’s intended query. User intents are very difficult to model properly [14]. In fact, search engines are unable to query deep web databases because they are unable to simulate a search of database contents like a human can do. Accessing deep web sources can only be done through search interfaces that hide those contents.

Google’s topmost recommendation as shown in Figure 1.1 was a car vendor called autotrader.com. The banner reads “Used Honda Civic for Sale Under $5,000” and purports to have about 539 rows (cars) of information. After clicking on that link, it shows the list of cars in Figure 1.2. As shown in that figure, none of the cars satisfy the query conditions. If the user manually uses the search interface and the filter functions to look for a car that has the closest specification, she will end up with list of results as shown in Figure 1.3, which is actually close to what she would like to see.

Our goal is to propose an implementation of the DQL query language [25] for
structured querying of deep web databases, and to demonstrate that such a system holds enormous promise for eCommerce applications. We present a prototype implementation called DeepQ using examples in the automobile sales industry.
Chapter 2

Related Work

The Deep Web can be accessed using different approaches such as the *Virtual Integration approach* and the *Surfacing approach*. Virtual integration is considered a data integration approach for accessing the deep web where a mediator is constructed for a specific domain [34]. Users can then pose their queries over this mediator. On the contrary, the surfacing approach works by pre-computing selected sources to index the results at the end. The majority of recent research on querying the content of the structured web has a focus on leveraging either the exposed tables in an indexable documents form, or using traditional query methods after exposing the content [25]. In addition, some deep web query approaches focus on exposing the page content first, then querying it using standard techniques. Many approaches have been used for querying deep web content by exposing the tables based on their predicted use and then querying the exposed content [16]. Also, modeling deep web sources as relational tables enables deep web querying and accessing while respecting access limitations.

Certain systems and techniques such as [17] and [29] and [33] have attempted to dynamically access and leverage deep web content that is relevant and permissible in the context of a query [26], but these attempts were focusing on specific parts such as interface integration and they do not provide a reusable framework for the research community to contribute in order to improve the field. This approach does not require prior indexing or harvesting of deep web content. It has been noted in the literature that the work on deep web integration is focused on multiple separated aspects such as schema matching. Some approaches were limited to discovering, organizing, and analyzing web forms to provide a form exploration feature [5], and other research focused on building fully integrated systems that start from HTML pages [40].
Different systems and approaches address different issues related to searching the deep web from multiple aspects. Some systems focus on interface integration and matching (web source integration); other systems focus on the data level integration (data integration). We will discuss the source integration in section 2.1, the data integration in section 2.2, and finally we will present the concept of the UR model in section 2.3 to highlight its relation with the deep web.

### 2.1 Deep Web Source Integration

Integrating Deep Web sources is most commonly handled by performing the domain-wise integration that is useful but challenging. For instance, the Visual Query Interface Integration System (VisQI)[29] was proposed as an integration system that extracts query interfaces from web pages, transforms them into a hierarchical representation, clusters them to different domains, and matches fields from different interfaces within their shared domain. The system was built in a modular way and it only addresses web integration developers. It can help developers to support their deep web integration systems and evaluate extraction and matching algorithms. The architecture of VisQI consists of different components that perform schema tree extraction, classify application domains, and match semantically equivalent nodes from different interfaces into clusters. Two different modes can be used in the system: the extraction mode and the mapping and classification mode. In general, the quality of the extracted data structures outperforms other methods such as [18] and [41].

Recently, other techniques were borrowed from the area of distributed information retrieval to be used within a framework for integration of deep web sources [11]. The proposed framework aimed to compute queries against a mediated schema that represents the structure of the web sources to perform an integration for deep web sources. Additionally, an approach was discussed for automated sampling, size estimation,
deep web source selection, and finally combining result lists.

Likewise other approaches, such as The Prudent Schema Matching Approach for web forms (PruSM) [35], focused on automating schema matching for web forms. This automated schema matcher can work effectively with a large, heterogeneous collection of forms focusing on the frequency of attributes. The availability of the information about web form interfaces is crucial for form filling and schema matching. Also, researchers have attempted to access the deep web and perform query interface integration based on incremental schema matching and merging, such as is proposed with DWQII [27]. The problem with this approach is that it gathers all results in one file without further filtering based on the needs of the user.

A very recent approach [22] was also presented to handle query interface schema extraction for the sake of enabling an effective access to the hidden web contents based on domain ontology. Even though many research projects addressed the schema extraction problem, this one is relatively new in the sense that it uses the internal code for the extraction within an ontology. This system and some others such as [19] are limited to the interface level.

2.2 Web Data Integration

There is great potential in integrating and re-purposing the enormous amount of structured data available on the Web. For instance, OCTOPUS [9] is a system that enables users to obtain new datasets gathered from those available on the Web. The idea is to automate manual tasks (e.g. data searching, extracting, and cleaning) using specific operators. The data that is manipulated by OCTOPUS is extracted from HTML tables and lists. There are three main integration operators that OCTOPUS performs:

- *The Search Operator* that takes user keywords and a set of relations and returns
a cluster of tables that are related to the user's keyword query string.

- The **Context Operator** that takes one relation and modifies it by adding a data column to the obtained table derived from the source web pages corresponding to the relation.

- The **Extend Operator** that provides more relevant data columns to the existing table by performing a join.

The only limitation with this system is that it only deals with the shallow web tables. Moreover, several strategies have been proposed for retrieving hidden web data automatically. For instance, Siphon++ is a Hidden web crawler for retrieving data that are hidden in the deep web behind keyword-based form interfaces. Some approaches use a fixed strategy for query generation, but the strategy in Siphon++ adapts query generation and selection by detecting features of the index underlying the search interface [40]. The adaptation to the features of the underlying search interface improves the crawler coverage. The proposed architecture consists of two main components: the **adaptive component** that discovers and selects index features using probe queries that are sent against the search interface, and the **heuristic component** that generates queries to retrieve the hidden content. The heuristic component has two phases: a *sampling phase* that aims to assemble a representative sample of the database, and a *crawling phase* that crawls the database using the most common word in the document of the sample. A sample of the indexes is built at the beginning, before the crawling starts, unlike other crawlers that build the sample within the process. The quality of the sample might affect the performance of Siphon++, and that kind of dependency can be considered as a weakness. Other attempts focused on attribute matching for data integration by applying data mining-based techniques [33]. The work not only targeted the input-input or output-output attribute matching, but also input-output matching.
It is important to mention fully integrated systems such as MetaQuerier [17] that were proposed for data integration. MetaQuerier was the first fully integrated system concerned with streamlining and automating web interfaces, and allows on-the-fly translation between unseen sources. Although it is a good integration system, it does not provide a reusable general framework for the community to advance the field. The system consists of a sequence of three subsystems as follows:

- An Interface Extractor takes HTML pages containing web forms as an input and provides extracted attribute information as an output.

- A Schema Matcher handles the web interface attributes and its semantic correspondences.

- An Interface Unifier generates the unified interface and the mapping that links it with the internal interfaces.

Overall, it is not appropriate for the deep web to follow the traditional access techniques; thus, there is a need to develop a useful and effective technique for accessing the deep web. Some research projects have explored a direction that aims to access the deep web upon user querying by redirecting the user to the appropriate database to be searched through its search interface. For example, MetaQuerier followed this direction [15].

### 2.3 UR Model for the Deep Web Search

The Universal Relations (UR) Model that was proposed by Ullman aims to allow the user to see the database as a simple view that is more easily queried [39]. The simplicity of the view comes from its representation as a single semantic view for the entire database [24]. In the UR concept, a relational database was viewed from a different perspective. This concept was supported by many proposed formal foundations
such as [4]. Previously however, supporting the UR model was a burden to system developers. Querying the relational database while considering the UR model was a complex task. Also, using the UR model required theoretical and system support. The above challenges made it unattractive to support the UR model. Nevertheless, it happens that the UR model offers some attractive properties that can be leveraged in the design of a structured query language for searching the Deep Web [24]. The UR as a user view is similar to a natural language interface to databases, and researchers have developed several implementations for UR interfaces over the years [31]

As web databases contain numerous amounts of structured information from different domains, the need for developing automated techniques to access the Deep Web becomes increasingly important. Web database access and integration requires combining and leveraging multiple techniques and approaches form different domains. There is a need for building an automated system that can access the Deep Web and return results as a user desires. We aim to leverage the query context to access the relevant deep web contents dynamically at run time.

Our goal is not limited to enabling effective access to the deep web, but also obtaining useful results that meet a user’s queries automatically on-the-fly and that require performing and integrating several tasks such as Schema Matching, Forms Filling, and Result Extraction. We are leveraging some previous work to perform these tasks as follows:

- **A Schema Matcher**: On-the-fly large scale applications require accurate and scalable schema matchers such as OntoMatch [7] without user intervention. OntoMatch was developed to leverage the strength of existing schema matchers in a way that is effective with improved quality matching. The beauty of this system relies not only on its effectiveness, but also on its flexibility and extensibility. It can use any number of matchers to improve the schema matching task.
• **A Form Filler:** Form filling is an important task in DeepQ as forms are our gateway to access the hidden contents inside the deep web. The form filling task needs to be done automatically and on-the-fly. There are several methods such as iForm [38] that can be implemented to perform this task. iForm can receive a *data-rich free text* input and use it to fill out the fields in form-based interfaces properly.

• **A Wrapper:** Extracting data from web pages was addressed in previous research extensively. Wrappers are software modules that perform data extraction from HTML pages. This task can be done by following several approaches; some of them use machine learning or automatic extraction; others provide specialized pattern specification languages for constructing the extracting program. FastWrap [3] is a table extraction technique that can handle dynamically generated web pages. In general, web databases generate web pages dynamically, and the content of those pages needs to be wrapped.

Most importantly, one of the recently proposed models [25] for the Deep Relational Web is a structured query model that can be used to query the deep web as a virtual database. This model also proposes a **Deep Query Language (DQL)**. DQL is a simple SQL-like language that facilitates searching the deep web as virtual database. Our work is an implementation of the DQL model.
Chapter 3

Our Work

3.1 Motivation and Objectives

This work focuses on proving that DQL is feasible and that it facilitates a programmatic access to the Deep Web. It is able to facilitate result integration from multiple sources to obtain results that are relevant to a user. Our objectives are:

- To propose an implementation of the DQL query language[25] for structured querying of deep web databases, and to demonstrate that the DeepQ system holds promise for eCommerce applications.
- To show that the UR view can be beneficial for simplifying the view of the deep web for users [24].

Our goal is to make the deep web data accessible and useful. The main objective is to show that the proposed structured query model is feasible, and that it has the potential to facilitate deep web access, enable leveraging of deep web data, and provide only the desired and useful results for the user [25]. In fact, DeepQ is built based on DQL, thus it can provide a reusable framework that the community can advance the field jointly.

3.2 Hypotheses

- An abstract view of the deep web as a schemaless universal relation is feasible and it enables the design of an SQL-like declarative query language called DQL.
• A translational semantics of DQL can be used as an implementation strategy to build a graphical user interface called DeepQ.

• A restricted subset of DQL has a faithful and intended mapping into DeepQ.

### 3.3 Work Contribution

Our contribution can be summarized as follows:

1. Presenting DeepQ as an implementation of DQL for structured querying of database contents behind firewalls. Also as a general reusable framework.

2. Implementing simple and complex query handling.

3. Adding more features to the system such as (Limit, OR) that can empower users with flexibility when querying hidden web database.

4. Demonstrating the UR model within the context of the deep web.

### 3.4 Methodology

This section briefly describes our work and the implementation flow of DeepQ [1] that is utilized to search the database contents behind the firewall inside the Deep Web. We will begin with describing how we leveraged the DQL features in section 3.5, then the DeepQ Architecture in section 3.6, and finally in section 3.7 we present an overview of the system and what has been implemented so far and published.

### 3.5 Leveraging the DQL features

The model consists of four steps that can be implemented and developed independently. The four steps are listed below:
1. Site Selection

2. Schema Mapping and Form Filling

3. Table Extraction

4. Output Processing

The second and third steps are implemented as operators: a *Transform Operator* and a *Combine Operator* respectively. The DQL syntax is simple as it is a SQL-like language that was proposed within a structured query model to query the relational web. DQL has only two clauses: **list** and **where**. The **list** clause is followed by all attributes users need to obtain in the result separated by a comma, and the **where** clause is followed by the conditions that should be satisfied by the system, using the needed relational operators such as ($=$, $>$, $<$). DQL includes a sentence that stands in parallel to SQL’s *select from where* that used to extract tables from deep web databases. The general structure of the extract statement is as follows:

```
extract Attribute List
using matcher μ filler φ wrapper ω
from φ
where θ
submit r
```

Although the model allows operations over a unstructured and semi structured data, it is completely relational. The ultimate goal is to have a uniform view of data as relations. It is important to mention that an ontology can be built and associated with the system. This can enable writing general queries without restricting users to specific keywords [37]. Also a good schema matcher can be beneficial for this task.
3.6 DeepQ Architecture

The functionality of the extract statement is leveraged to implement a graphical interface in order to facilitate search as DeepQ system interface. User can provide a query that has two parts: a list of attributes that they want to obtain as a response for their query, and a set of Boolean conditions that must be satisfied. The detailed DeepQ architecture is illustrated in Figure 3.1.

![DeepQ architecture](image)

For example, a user might want to have a list of Honda Civic EX vehicles which does not have more than 120,000 miles on it, the exterior color is red, it is priced below $5,000, and the dealer is located in Moscow, Idaho. He wants to know the dealer name and address, car’s VIN number, exact price and the dealer’s contact phone. She will formulate her DeepQ query as follows in the user interface:

**List:** DealerName, DealerAddress, VIN, Price, DealerContactPhone

**Conditions:** miles ≤ 120000 and make = Honda and model = Civic and trim = EX and color = red and price ≤ 5000 and zipcode = 83843

Once the query in the above form is submitted, the Query Rewriting unit of DeepQ then consults the Deep Web Index it maintains to find websites such as autotrader.com and cargurus.com and reformulate the queries as the following *extract* statements (similarly for cargurus.com).

*extract* DealerName, DealerAddress, VIN, Price, DealerContactPhone

*using matcher* OntoMatch *filler* iForm *wrapper* FastWrap
where miles ≤ 120000 and make = Honda and model = Civic and
        trim = EX and color = red and price ≤ 5000 and zipcode = 83843

from autotrader.com

Once the query in the above form is submitted, the Query Rewriting unit of DeepQ
then consults the Deep Web Index it maintains to find websites such as autotrader.com
and cargurus.com and reformulate the queries as the following extract statements
(similarly for cargurus.com).

extract DealerName, DealerAddress, VIN, Price, DealerContactPhone
using matcher OntoMatch filler iForm wrapper FastWrap
where miles ≤ 120000 and make = Honda and model = Civic and
        trim = EX and color = red and price ≤ 5000 and zipcode = 83843

from autotrader.com

These extract queries are then annotated and reformulated extensively to simulate
the web site interaction to be able to interrogate the deep web site. For example,
in the autotrader.com, the front-end only allows submission of vehicle make, model
name, and zip code. In the next step, it allows filters to weed out unwanted responses.
DeepQ therefore will breakdown the above query into the following two queries:

extract *
using matcher OntoMatch filler iForm wrapper FastWrap
where make = Honda and model = Civic and zipcode = 83843
from autotrader.com

extract DealerName, DealerAddress, VIN, Price, DealerContactPhone
using matcher OntoMatch filler iForm wrapper FastWrap
where miles ≤ 120000 and trim = EX and color = red and price ≤ 5000
from autotrader.com
These statements are executed in tandem and the query variables are matched with the site variables using OntoMatch schema matcher, and the manner in which they are used in the form or the filter in the next page at autotrader.com is decided by iForm, and it matches based on nearest match that will maximize the set of responses.

The interesting part is that even though DeepQ is able to execute these two extract queries successfully, the information that are gathered may still not satisfy the query conditions. For example, even though the filter in the second page has a Price Rating option, the options are “Good Price” and “Great Price”, which do not conform to the format of price condition of the query. So, DeepQ submits for both options. Also for the mileage filter, autotrader.com allows bands. The closest two were \( \leq 150,000 \) and \( \leq 100,000 \). DeepQ chose \( \leq 150,000 \) so that 120,000 falls in between. Finally, after extracting the data using FastWrap wrapper, DeepQ stores them in the Response Triage to process later.

The Query Processing unit processes these queries to collect all tentative responses as candidates. Any further filters that could not be processed at the source, are then applied at the DeepQ local database Response Triage using a simple SQL query, and the final response returned to the users.

### 3.6.1 Pick-and-Filter Interface

The design of the deep web interfaces vary widely even though there are similarities. Many of these data sources use a pre-filter in their landing page to steer the queries to a more focused exploration, e.g., autotrader.com. In its landing page it only allows to query based on make, model and zipcode, which then leads the users to second page where a plethora of filter conditions can be set to find the vehicle of interest. We term these types of deep web interfaces as Pick-and-Filter type interfaces. On the other hand, sites such as carvana.com use the landing page itself for user query submission, which we define as direct filtering-based exploration. Five major conditions are listed
In DeepQ, we support both types of interfaces – direct and pick-and-filter type. For DeepQ to successfully execute queries in both of these types of interfaces, it must know the architecture of these sites and the search parameters they allow. The Query Rewriting module of DeepQ gathers such information about the sites from the manually maintained and curated Resource Capability Description meta-data database. In this database, for each of the site, we maintain the URL, the type of interface (direct vs pick-and-filter), the set of form entries and filters for each page (names), type of each form entry and filter (e.g., text, rolodex, list, radio button, check box, drop-down, etc.), and variable type of each (e.g., term, number, date, etc.).

Given a DeepQ query in the form of a pair \((\text{List}, \text{Conditions})\), our interest is in discovering what conditions can be applied in each type of interfaces, where (first page
or the second), and how. For example, in the case of query $Q_1$, we determine that autotrader.com is a pick-n-filter type interface, and only Make, Model and Zipcode conditions can be applied to the landing page filter, and we must apply trim and color filters in the second page. However, we must pick mileage at $\leq 150,000$ and tick off all prices to collect possible candidates knowing that they do not allow exact filters. Also, this determination is made at run time by the Query Processor with the help of iForm form filler function.

The other choice is the first obtain all possible candidates without setting the mileage and price filters, and subsequently making a secondary filter query to weed out the responses that do not fit the query. However, in both cases, we extract the table of vehicle information using a table structure recognition function in FastWrap wrapper, and proceed to perform the secondary filtering.

In the event that the user query had a condition that included a specific dealer name (e.g., Moscow Motors), the processing strategy would be significantly different. The second extract query would still look like as shown, i.e., the where clause will not include the DealerName = Moscow Motors condition, in particular. This is because there is no filter available to pick this dealer. Instead, we obtain all the car information that meet the rest of the conditions, and then we filter out the cars that are not sold by Moscow Motors as a secondary query in DeepQ locally.

### 3.6.2 Event-Driven Resource Indexing

In the current edition of DeepQ, the databases Deep web Index and Resource Capability Descriptions are manually extracted, stored and maintained. However, the next edition of DeepQ will have a module to extract such meta-data fully automatically. Fortunately, many of the databases today use identical designs because many of them are designed using services such as Shopify\(^1\) and Wix\(^2\), even though the number of

\(^1\)https://www.shopify.com/

\(^2\)https://www.wix.com/
such sites are small. This will expedite the process of meta-data collection because we are able to apply almost the same set of filter conditions across various sites. One such obvious example is that Expedia.com and Orbitz.com have almost identical interfaces even though they are not designed using the likes of Shopify or Wix. The overall idea of the module is as follows.

Every DeepQ query will first be submitted in Google to explore possible identification of deep web sites that are relevant as determined by Google’s powerful search engine. We will choose top $K$ number of deep web sites to explore and possibly index. We use of schema matcher OntoMatch to find out what fraction of our query variables match the front-end and back-end filters, what variables it returns, and their types. We do so by submitting a probe query using the conditions in the DeepQ query, not necessarily to obtain a result, but just to explore the schema of the site. Once we have explored all $K$ sites, and possibly indexed, we move on to actually process the query using the information in these two databases. This is how we increase our reach into the deep web database space on the internet. Obviously, there are several choices on how we can do this because it has query processing implications. We can decide to just log the sites for now, and process them for meta-data collection during DeepQ idle times as opposed to at query processing time.

## 3.7 System Overview

The system we are implementing mimics the Structured Query Model for the Deep Relational Web [17]. A simple state of the implemented system is illustrated in Figure 3.3. It shows three main parts: a text user interface, a CGI (Common Gateway Interface) with related Databases, and deep web forms. In order to access the deep web effectively and obtain the useful information, users will interact directly with a text interface. The text interface allows users to submit a query that is written in
DQL to obtain an output that is gathered not only from one form interface, but also from all forms that can provide the desired results. The user text interface runs a CGI script that provides a dynamic web page dependent upon the submitted DQL query. The CGI script works as an intermediary between the text interface and the other deep web forms.

3.7.1 Deep Web forms

Deep web forms (or search interfaces) are designed to facilitate the Human-Computer Interactions; and in order to access the deep web effectively, this interaction needs to be simulated automatically. Users are able to fill in fields in web forms with specific parameters; usually each form is an interface for an online web database with unique schema and does not necessarily belong to one specific domain. Search forms are treated as black boxes in the DeepQ, but in order to understand the underlying web databases and leverage their contents, search forms should be understood clearly. Forms usually have selectable input requirements (access limitation), and expected output. In traditional scenarios, if users tend to access one of these forms directly, they would need to fill the form, send the query, and the result would be automatically
generated after querying its related database. For example, in Figure 3.3 there are three different forms that belong to a single domain (car sales) from three different web sites. Form4 allows users to select model, color, and price to find a car that meets the user’s selections and retrieve the available information from the database as the following: Car Id, Make, Model, Color, Price, Seller Name, and Year (see Figure 3.4), also another example is Form2 in Figure 3.5. In addition, the table in Figure 3.6 shows the input and output attributes for the three forms.

![Search Vehicles](image)

**Figure 3.4:** Form 4 after submitting as an example

It is important to mention that no dynamic scripts such as JavaScript were involved with the building process of any of our forms. Web sites that use JavaScript have been known to pose a challenge for web crawling. This is one of the unsolved issues when dealing with web site crawling [20].

### 3.7.2 The CGI Script and related databases

Creating dynamic web contents are assisted by some mechanisms and technologies like server-side programs that are the most common/oldest method for generating web pages on the fly, CGI as an example [36].
The CGI script performs the following on submit:

- Get keys from user query by identifying attributes associated with the Where clause and the list clause.

- Recommending candidate forms based on both input-output attributes. Form input considered as access limitations that enforces internal constraints. The system specifies relevant candidate forms that can be queried and premised to obtain the desired output. This interface identification task can be implemented by using a recommender system; the implemented algorithm mimics the recommender system task as the follows:

For example, if there is a form that accepts only Make and Model as inputs,
but the user query contains not only Make and model. What if she asked for also color or seller information that only can be found within the output? Considering only form input is not efficient, the information desired might exists in a specific form but will not be obtained if the system just considers form access limitations (input). It is more efficient and useful to consider also output information; see the example in section 3.8

- Schema matching and form filling
- Get all response and write them to one table on-the-fly (SQLite is used)
- Evaluate user query against that table and output processing

3.7.3 User Interface

DeepQ support both types of interfaces. The direct interface and pick-and-filter type The Graphical user Interface allows users to select the output field names and the query conditions that corresponds to (list, conditions) in DeepQ and return response. See Figure 3.7. The Free Text user Interface shown in Figure 3.8, is a simple interface that has a text box and submit button and it allows unrestricted queries. Sophisticated users can write their queries directly in DQL without worrying about manual picking and filtering, filling multiple forms, making separate decisions, and compare between results. The simple free text user interface reduces user intervention while accessing the deep web which can improve scalability [20].

3.8 Single Domain web databases (Simple Query) - An Example

The deep web sites interfaces are mostly heterogeneous. Processing deep web queries in a single domain is considered as simple querying process [24].
Assume that a user submits the following query to the Text User Interface:

\[
\text{list price, year where make = Jeep and model = Cherokee and color = Silver and price} \leq 40000
\]

This user is interested in finding a **Silver Jeep Cherokee that cost \( \leq 40000 \)**; The user wants only to see the price and year in the result. According to Figure 3.3, all
three forms can be queried. All of them are candidate forms in this situation. Keep in mind that each form has its own access limitation; and there is no form takes all user requirements listed in the where clause as input. In addition, the user asked for \textit{price} and \textit{year} in the list clause. The only form that provides \textit{year} is \textbf{Form 4}. It accepts \textit{Model}, \textit{color}, \textit{price} and provides \textit{Make} and \textit{year} (besides other output). In general, the simplest description is; when a user sends a query, the system works perform the following:

1. Determines candidates forms by determining the mandatory input keys for each form.
2. The system checks if other requirements in the query are available within the forms output. (Extras)
3. Sending queries and receiving results.
4. Filtering results based on user query.

\section{Results}

In this section we are presenting the results of Single domain querying (simple query processing); the complex query will be implemented in the future, see section 3.10.1.

\subsection{Single Domain Results - Simple Query}

In this section we will show the results from both interfaces, the graphical interface and the text user interface. After running the query in the the Text User Interface as shown in Figure 3.10, results were returned from \textbf{form 3 and 4} and stored in the \textit{storage triage}. After applying the secondary filtering process ; the only one result displayed to the user is the one that met the query condition. The Text interface is designed for more sophisticated users. On the other hand, as shown in Figure 3.9, the
user got the desired result but the query was restricted only to the conditions that are provided by the graphical interface.

![Figure 3.9: Graphical Interface Query Result](image)

![Figure 3.10: Text User Interface Query Result](image)
3.10 Future Plans

3.10.1 Multiple Domains Web Databases - Complex Query Implementation

Let us consider that a user submits the following query to the Text User Interface:

list price, year, DealerName, city, state where make = Jeep and model = Cherokee and color = Silver and price \leq 40000 and zipcode = 99163 and elevations \leq 2600

The user added a condition that does not belong to the (sales) domain. The *elevations* condition can not be found in the sales domain. Thus, a different domain will be considered and added to be part of the candidate forms to be processed as a complex query after implementing the Combine Operator[21].

An overview of the process that deals with querying the multiple domain web databases is shown in Figure 3.11.

![Figure 3.11: DeepQ System Overview for the multiple domain querying.](image)
3.10.2 More Features

We also plan to add more features such as *Limit* and *OR* as follows:

- **Limit**: for specifying maximum number of results.

- **OR**: To allow a user to pose more flexible queries such as:

  ```
  list price, year
  where make = Jeep and model = Cherokee and color = Silver and price \leq 40000
  or
  make = Jeep and model = Wrangler and color = Silver and price \leq 40000
  limit 10
  ```

  The idea is to empower users with the ability to access databases behind firewalls and also to flexibly control the desired output.

3.11 Performance and Evaluation Criteria

Generally, the performance of deep web searching systems can be captured in different ways, and several factors can be taken into account. For instance, in web crawlers, it is important to capture *effectiveness* and *coverage* [40]. Furthermore, with form submission, none of the crawling metrics capture the challenge of dealing with hidden web form submission [36]. Also, relevance is usually one of the performance measures with web crawlers. One of the goals of the implemented model is enabling users to obtain useful and to-the-point results. The DeepQ system has the potential of capturing three important factors as shown in Figure 3.12:

- **Effectiveness and Usefulness**: The user gets ONLY the desired result(s).
• **Accessibility**: The system accesses multiple forms in a single or different domains, fills them and gets results back, while respecting the form's access limitations.

Users’ expectation while interacting with the DeepQ system will be similar to the way they interact with databases. In other words, users will expect precise and complete answers [23], unlike what they get when interacting with traditional search engines. A balance between precision and recall is expected when using the DeepQ system because it reaches a compromise between users’ expectation from both search engines and databases (see Figure 3.13).

It has been known that the concept of the Universal Relation model can actually solve the *usability* problem because it allows users to interact with databases with no prior knowledge of its structure [32].

![Figure 3.12: DeepQ performance measures.](image)

Finally, the Figure 3.14 presents strengths and limitations for some systems, including DeepQ. Figure 3.15 presents further comparison between the capabilities of DeepQ and related systems.
3.12 Problems and limitations

One of the obvious problems is that most web sites use dynamic scripts such as JavaScript and Ajax for content generating[20]. In this work, we are limited to the websites that are not using such dynamic scripts.
### 3.13 Conclusion

As has been noted, Deep Web data can be accessed and leveraged if the system is implemented properly. Users can obtain the desired results in a useful and effective way without being overwhelmed with having to manually search multiple web forms. DeepQ experimental implementation demonstrates the viability of the DQL platform for dynamic and ad hoc querying of deep web databases. We believe that the development of related research can improve the functionality of the system.

Future work will focus on implementing the four main stages of the structured query model for the relational deep web, and build the operators related to it (Figure 3.16) beside the work that was mentioned in section 3.10. Also, developing a dynamic resource indexing module is one of our in progress work; this will help to improve site recommendation and provide more flexibility. Additionally, the complexities introduced by the use of a diverse set of component tools (e.g., wrappers, matchers,
form fillers, etc.) within the DeepQ system raise opportunities for optimization and questions about efficiency.

Figure 3.16: Operators and implementation plan.
Bibliography


