Dexter: An Overview

Abhijeet Mohapatra, Sudhir Agarwal, and Michael Genesereth

1. Introduction

Dexter (*http://dexter.stanford.edu*) is a browser-based, domain-independent data explorer for the everyday user. Dexter enables users to plug-n-play with heterogeneous, Web-accessible, structured data sources in a unified framework.

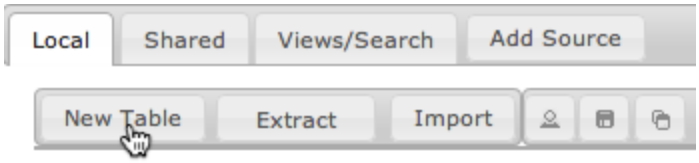
In particular, Dexter allows users to:
1. Create, edit and query tables locally in the browser.
2. Export and share local tables.
3. Import the data from local and remote CSV, XML and JSON files into local table(s).
4. Extract the contents of Web-pages into local tables.
5. Dynamically connect to remote MySQL databases and JSON(P)-APIs, and query them as well as shared tables.

Dexter respects users' privacy by storing users' local data and evaluating queries client-sided inside their browsers.

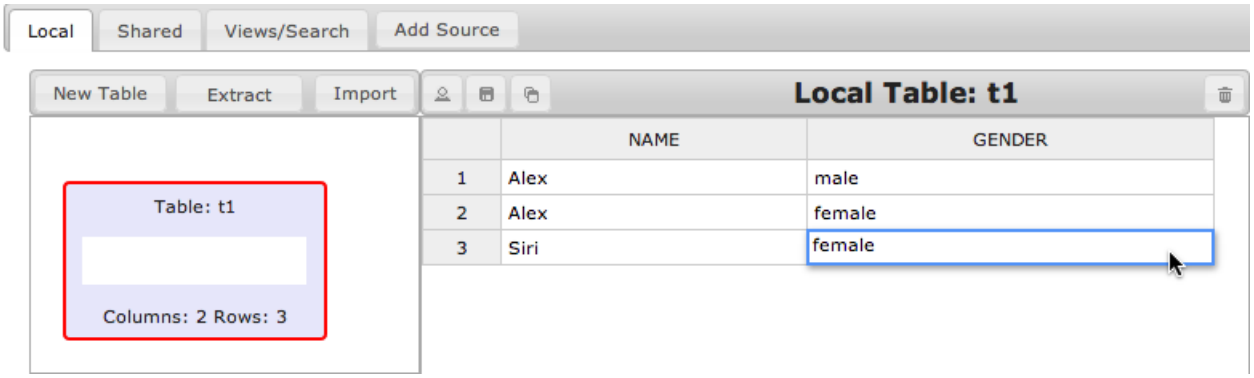
In this document, we present an overview of the main features of Dexter and how they can be used.

2. Creating and Editing Local Tables

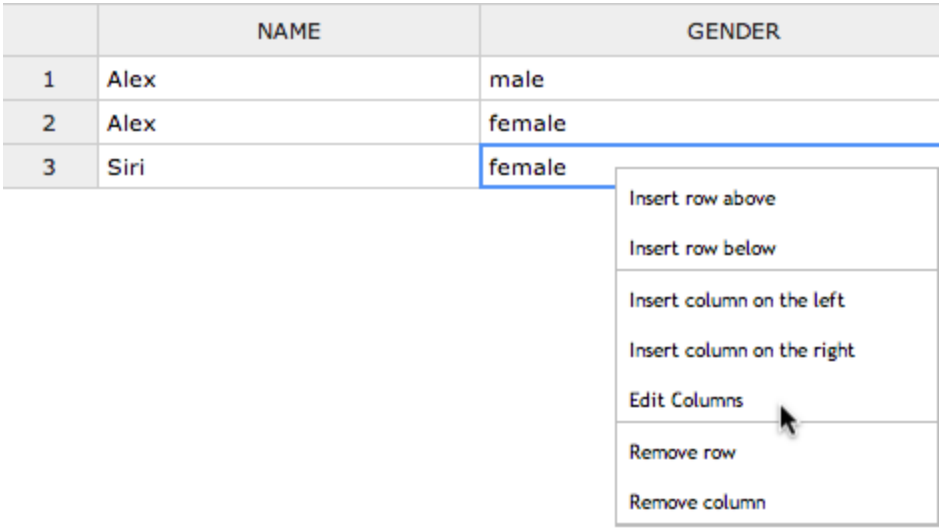
Users can create local tables by pressing the "New Table" button under the 'Local' tab. Local tables of a user are stored persistently in the local store of the user's browser.



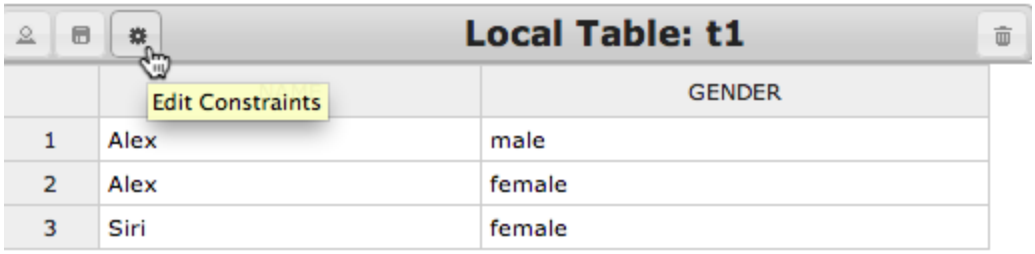
When a user creates a table, it appears as a thumbnail inside the user's repository. When a user clicks such a thumbnail, the contents of the table are presented to the user as an editable table. The user can modify table cells either by manually editing or pasting new values into the cells.



In addition to editing a table's contents, a user can edit the structure (or *schema*) of a local table either by (a) changing the table's name, (b) changing the column names, or (c) inserting / deleting rows and columns. The schema editing operations can be invoked through the context-menu that opens with a right-click on the table.



Data validation: To ensure the validity of data that is entered into local tables, Dexter allows users to impose *constraints* on local tables.



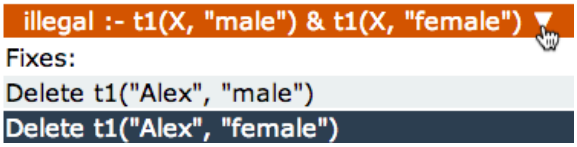
Constraints characterize the scenarios under which a violation occurs. Constraints are of the form, illegal :- <condition>, where <condition> is the body of a Datalog rule (see Section 3 for more details on the Datalog language). Consider for example the following constraint on table t1.

illegal :- t1(X, “male”) & t1(X, “female”)

As per the above constraint, the gender of a person is uniquely characterized by his / her name. In other words, if two people with the same name (X) have different genders, then a violation occurs. In table t1, X = “Alex” causes a violation.

Unlike traditional data management systems where the data must always satisfy the underlying constraints, Dexter allows the contents of local tables to violate the constraints. However, when violations occur, user is made aware of the violations by highlighting the constraints that are violated.

For every violated constraint, Dexter suggests actions (for e.g. inserting or deleting rows from a table) that could *fix* the violation as shown below.



3. Querying Tables

Dexter allows users to define *computed tables* using Datalog^{Agg} rules. A computed table can be created by clicking on “New View”. The answers of a computed table can be examined by clicking on “Evaluate”. We note that unlike local tables, users *cannot edit* the content of computed tables.

Datalog^{Agg} is a variant of Datalog that is extended using sets, tuples, aggregates, and built in arithmetic and comparison operators. The vocabulary of Datalog^{Agg} consists of:

- *Object Constants*: (double-quote-escaped) strings that are enclosed within quotes ("). E.g. "1.23", and "\"Why?\""
- *Relation Constants*: refer to names of local and computed tables, and are strings of the form [a-z][a-zA-Z0-9_]* except strings that start with "p_", "n_", "s_temp_", or "s_body_". E.g. r1, and r_.
- *Variables*: strings of the form [A-Z][a-zA-Z0-9_]*. E.g. Name, X and Xy_.
- *Sets*: are enclosed within braces and can either be empty {} or a collection of object constants, sets, or tuples which are separated using comma (","). E.g. {"1", "2", "3"}, and {{"1", "a"}, {"2"}}.
- *Tuples*: (a.k.a ordered sets) are collection of one or more object constants, sets or tuples that are separated using commas (",") and enclosed within square brackets. E.g. ["1", "2", "3"], and [{"1", "2"}, "a", {["3", "4"]}].

Computed tables are defined in Datalog^{Agg} using one or more rules. A rule has two parts, a head and a body, that are separated by ":-". The head consists of a name followed by a list of arguments (object constants, sets, tuples, or variables) that are enclosed within round-brackets. The body consists of one or more heads (each of which could be prefixed by a tilde "~") that are separated using ampersand "&". For further details regarding Datalog^{Agg}, the reader is referred to the Datalog^{Agg} manual.

Example 3.1 Projecting Columns: A user can query for the *distinct values* of one or more columns of a table by modifying the head arguments in a rule.

q1(X) :- people(X, Y)
Answers of q1 = {X | ∃Y s.t. people(X, Y)}

people

X	Y
Alex	male
Alex	female
Siri	female

q1

X
Alex
Siri

Example 3.2 Filtering Columns: A user can query for the answers from a table that *satisfy a supplied condition*.

q2(X) :- people(X, "male")
Answers of q2 = {X | people(X, “male”)}

people

X	Y
Alex	male
Alex	female
Siri	female

q2

X
Alex

Example 3.3 Joining Tables: A user can query for answers *across 2 or more tables*.

q3(Z) :- q2(X) & address(X, Z)
Answers of q3 = {Z | ∃X s.t. q2(X) ∧ address(X, Z)}

q2

X
Alex

address

X	Z
Alex	Stanford
Siri	Cupertino

q3

X
Stanford

Example 3.4 Disjunction: Computed tables can also be defined using multiple rules. In such a case, the set-union of the answers of all the rules is computed.

q4(X) :- address(X, Y)
q4(X) :- address(Y, X)
Answers of q4 = {X | ∃Y s.t. address(X, Y)} ∪ {X | ∃Y s.t. address(Y, X)}

address

X	Y
Alex	Stanford
Siri	Cupertino

q4

X
Alex
Siri
Stanford
Cupertino

Example 3.5 Negation: Consider the following query.

q5(X) :- q1(X) & ~q2(X)

Datalog^{Agg} rules are evaluated under the closed-world assumption using *negation-as-failure*. In the above query, X is an answer of q5 iff X is an answer of q1 but *not* of q2.

q1

X
Alex
Siri

q2

X
Alex

q5

X
Siri

Example 3.6 Aggregation: A user can perform aggregation using the “setof” operator.

q6(X, S) :- setof(Y, people(X, Y), S)
The tuple [X, S_x] is an answer of q6 *iff* S_x = {Y | people(X, Y)}

people

X	Y
Alex	male
Alex	female
Siri	female

q6

X	S
Alex	{“male”, “female”}
Siri	{“female”}

For details regarding the semantics of “setof”, the reader is referred to the Datalog^{Agg} manual.

Example 3.7 Recursion: A computed table can also be defined recursively.

ancestor(X, Y) :- parent(X, Y)
ancestor(X, Y) :- parent(X, Z) & ancestor(Z, Y)

parent

X	Y
Alex	Mary
Mary	Tom

ancestor

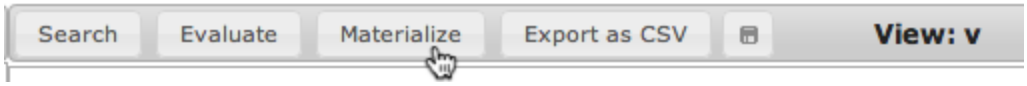
X	Y
Alex	Mary
Mary	Tom
Alex	Tom

The first rule establishes the *base case* of recursion i.e. all parents are ancestors. The second rule recursively computes ancestors.

Built-in Predicates: A function f(X1, X2, ..., Xn) that returns a value other than true or false, is represented in Datalog^{Agg} as a predicate f(X1, X2, ..., Xn, O) where O = f(X1, X2, ..., Xn). Otherwise, the function is represented as is. The variables X1, X2..., and Xn must be bound before evaluating the built-in predicate. Examples of built-in predicates include arithmetic operators (for e.g. *same*, *distinct*, *greater*, *plus*, *mod*), string-manipulation operators (for e.g. *length*, *substr*, *concat*), set-based operators (for e.g. *member*, *union*) and tuple-based operators (*attr*). For the complete list of the available built-in predicates including their description, the reader is referred to the Datalog^{Agg} manual.

As discussed previously, users can query their local tables in Dexter by creating computed tables. The definitions of a computed tables are automatically stored in the local storage of a user’s browser and can be edited by the user. Changing the schema of a computed or a local table can potentially render the dependent tables unsafe or semantically invalid. Currently, Dexter relies on users to manually repair the definition of such corrupted tables.

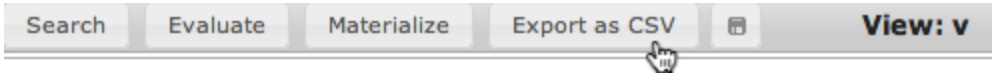
Materialization of Computed Tables: Dexter also allows users to *materialize* computed tables in their browser's local storage.



A computed table's materialization is used to answer queries over the computed table instead of re-evaluating it. Therefore, materialization can significantly speed up the evaluation time of queries over computed tables. However, when the data in the underlying sources is updated, materializations of computed tables could potentially become outdated. Currently, Dexter does not automatically update the materializations of computed tables e.g. by synchronizing with the underlying data sources. Rather, Dexter allows users to re-evaluate and re-materialize the computed tables any time they wish to do so.

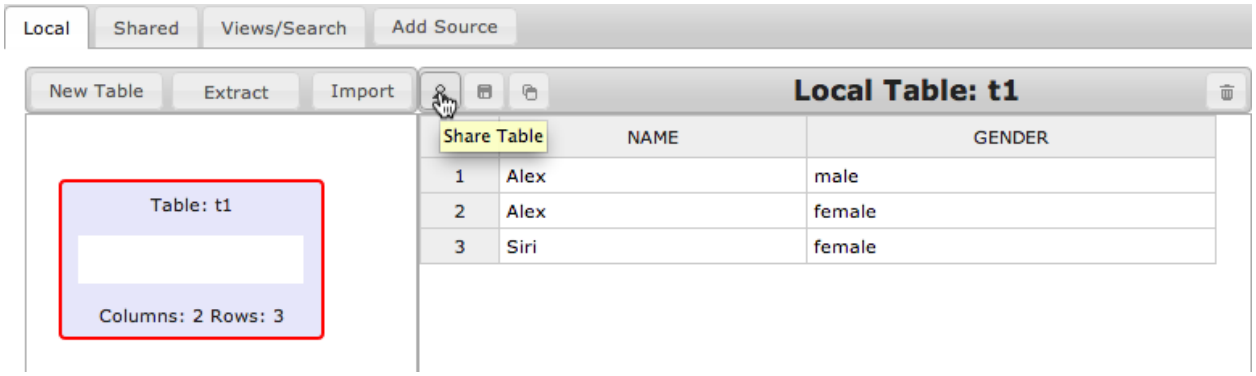
4. Exporting and Sharing

Dexter allows users to *export* local and computed tables to their file-system in CSV format.



Dexter uses comma (,) as default separator, and allow users to specify a delimiter if they wish to use a different delimiter. The data is exported from a user's browser, instead of, first, transferring the data to a remote server and subsequently, initiating a download from the server.

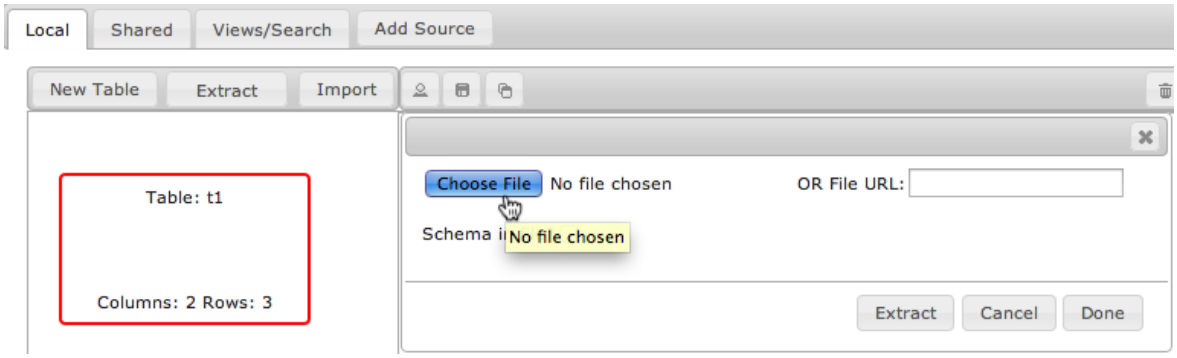
Users also can *share* their local table(s) with other users through Dexter. A local table can be shared by clicking on the table and subsequently clicking on the "Share Table" option as shown below.



The shared tables are hosted on Dexter's data-sharing server. Two tables that are shared by different users can potentially have the same name. Therefore, to avoid ambiguity between names of the shared tables, a shared table's name is prefixed with the table owner's username followed by an underscore ("_"). For example if a user 'sa' shares his local table, say 't5, then the shared table is stored as 'sa_t5'.

5. Importing CSV, XML and JSON data

In addition to creating and populating local tables manually, users can import CSV, XML and JSON files as local tables by clicking on “Import”. These files can either be located in the user's file-system or accessible via a URL.



A CSV file is imported into Dexter as a single table. Rows of the table correspond to lines in the CSV file. Dexter automatically detects the delimiter (e.g. ‘,’, ‘|', ‘\t’ and ‘ ’) used to separate adjacent fields in the CSV file. When users import CSV files, they may specify whether the first line of the file contains the column names or not. In the latter case, default column names (X1, X2, ..., and Xn for a table with n columns) are used. The following example illustrates the import of a CSV file 'students.csv' into a local table 'students'.

“Abhijeet”, “27”
“Eric”, “28”

⇒

X1	X2
Abhijeet	27
Eric	28

Users can also import JSON and XML files into Dexter. The process of importing a *flat* JSON or XML file is similar to that of a CSV file. An example is shown below.

```
[
  {“name”: “Abhijeet”, “age”: “27”},
  {“name”: “Eric”, “age”: “28”}
]
```

⇒

Name	Age
Abhijeet	27
Eric	28

When *nested* JSON and XML files are imported into Dexter, their contents (JSON objects or XML nodes) are *flattened*. A nested XML file ‘article.xml’ shown below is imported into Dexter as follows.

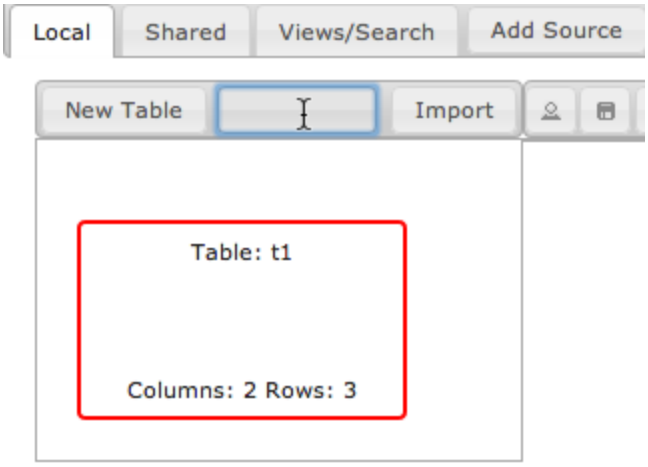
```
<article>
  <name>Dexter - An
Overview</name>
  <author>
    <fName>Abhijeet</fName>
    <lName>Mohapatra</lName>
  </author>
  <author>
    <fName>Sudhir</fName>
    <lName>Agarwal</lName>
  </author>
</article>
```

⇒

Article_name	AuthorFName	AuthorLName
Dexter - An Overview	Abhijeet	Mohapatra
Dexter - An Overview	Sudhir	Agarwal

6. From Web-pages To Local Tables

Structured data from Web-pages constitutes a large fraction of structured that is publicly available. A user can import fragments of a Web-page into a local table by first, copying the desired fragment into the clipboard and then pasting the copied fragment in the “Extract” field as shown below.



These fragments could correspond to a HTML table, few rows of a HTML table, or even non-table block elements that are present in a Web-page's DOM. We illustrate the extraction of data from a Web-page into a local table in Dexter using the following examples.

Source: http://en.wikipedia.org

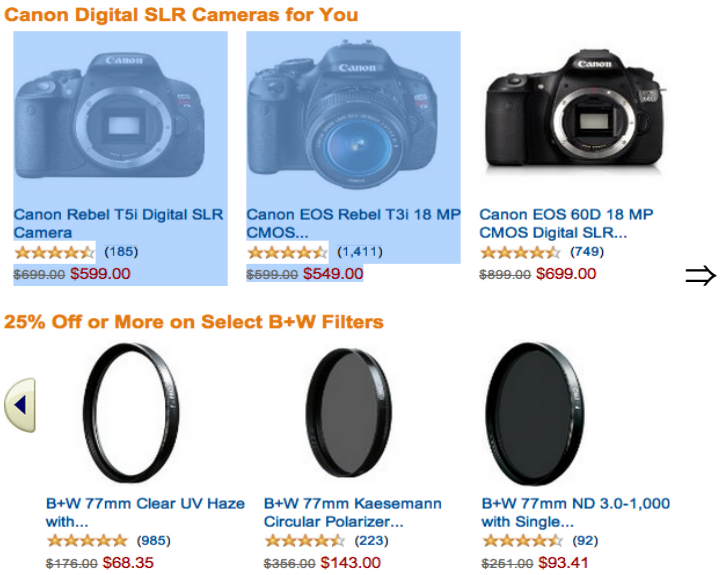
Territories of the United States with no permanent population

Territory	Acquired[10]	Territorial status[17]	Land area in mi² (km²)[L]
Baker Island	1856	Unincorporated, unorganized	0.9 (2.3)[1]
Howland Island	1858	Unincorporated, unorganized	0.6 (1.6)[1]
Jarvis Island	1856	Unincorporated, unorganized	2.2 (5.7)[1]

⇒

X1	X2	X3	X4
Baker Island	1856	Unincorporated, unorganized	0.9 (2.3)[18]
Howland Island	1858	Unincorporated, unorganized	0.6 (1.6)[18]
Jarvis Island	1856	Unincorporated, unorganized	2.2 (5.7)[19]

Source: http://amazon.com/



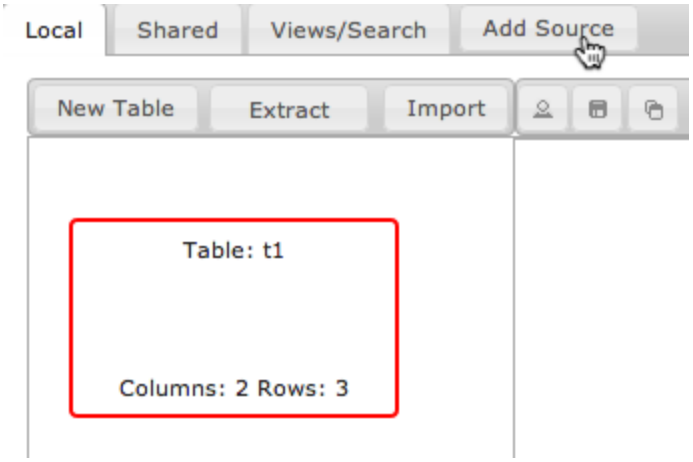
⇒

X1	X2	X3	X4
Canon Rebel T5i Digital SLR Camera	(185)	$699.00	$599.00
Canon EOS Rebel T3i 18 MP CMOS...	(1,411)	$599.00	$549.00

7. Remote Tables

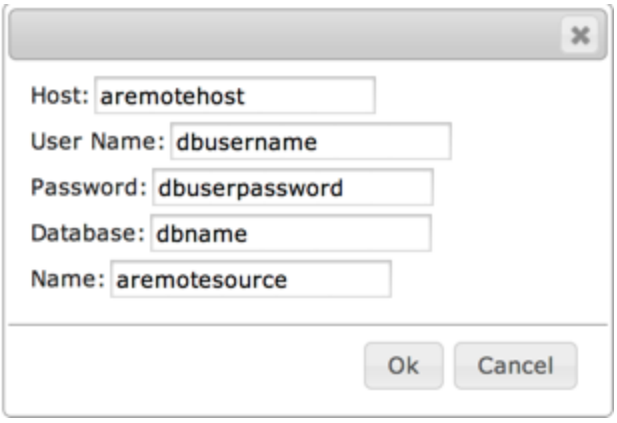
In addition to local tables, Dexter allows users to connect to three types of remote data sources.
(a) Shared tables (which are hosted on Dexter's data-sharing server)
(b) MySQL databases
(c) JSON(P) APIs

The data from remote sources is available to users in the form of *remote tables*. Users can connect a MySQL database or a JSON(P) API by clicking on "Add Source".



A user can connect to a MySQL database through Dexter by supplying the following parameters.
a. *Source Details*: Hostname (IP or domain name of MySQL Server) and Database Name
b. *Authentication Details*: Username and Password

c. *Connection Name*: The source name that will be used to refer to the MySQL database connection when querying the database's tables using Datalog^{Agg} rules.



Similarly, a user can also connect to a generic JSON(P) API through Dexter by supplying the following parameters.
a. *Connection Name*: The source name that will be used in Datalog^{Agg} rules.
b. *Querying capability*: Can either be 'filter' or 'datalog' based on whether the api allows filters or datalog queries.
c. *Client-side Accessibility*: Whether the API call can be invoked on the client side or whether it requires Dexter's server to act as a proxy.
d. *FnLowering*: Function that takes the object type, offset (say *o*), and *limit* (say *n*) for rate-limiting APIs, and returns the arguments that are required to invoke the API for fetching *n* answers starting at the offset *o*.
e. *FnLifting*: Function that converts the JSON answers to a table.
URL: The URL to be invoked to obtain data from the source of type `datalog'
f. ***O***: Set of specifications of object types. An object type *O* in ***O*** is a tuple (*name*,*properties*,*URL*), where *name* is the name of object type *O*, *properties* are the properties *O*, and *URL* is the URL to be used to fetch instances of *O* from a datasource of type 'filter'.

When the connection to a remote source succeeds, Dexter creates a new tab with the provided name and presents the data in the remote source as tables inside the that tab. A user can immediately browse through the data in the remote tables.

The above mentioned connection parameters for MySQL databases and JSON(P) APIs are stored locally inside user's browser and thus do not need to be re-entered when the user revisits Dexter. Note that Dexter does not store the data from remote sources persistently in a user's local store due to various legal or technical reasons.

Querying Remote Tables: A remote table can be queried by prefixing the table name with the *source name* followed by a dot (".").

Suppose, our example remote MySQL database contains a table ‘person’ with columns ‘Name’, ‘Affiliation’, and ‘Office’. The table can be used in a query for all distinct offices e.g. as follows.

v(O):-aremotesource.person(N, A,O)

The source name ‘shared’ is used to refer to shared tables. For example, suppose that a table, say sa_t1(X, Y) is shared on Dexter’s data-sharing server. This table can be projected on its 2nd column as follows.

v(Y) :- shared.sa_t1(X, Y)

Currently, Dexter does not allow users to edit the contents of remote tables. However, users can copy the contents of a remote table into a local table and then edit the local copy.