damvitool Documentation

Release 0.2.0

praxigento

February 17, 2015

Contents

1 Installation **3**

2 Using damvitool **5**
2.1 Run damvitool with sample database 5
2.2 Admin panel access 5
2.3 Supported databases 5

3 Authentication **7**

4 damvitool frontend usage **9**
4.1 Open frontend 9
4.2 Construct new request to database 9
4.3 Build new database request 9

5 RESTful API **13**

6 Developing damvitool **19**
6.1 Install damvitool for development 19
6.2 Running the tests 19

7 Roadmap **21**

8 Indices and tables **23**

Contents:

CHAPTER 1

Installation

Use pip to install damvitool:

```
$ pip install damvitool
```

CHAPTER 2

Using damvitool

2.1 Run damvitool with sample database

To run damvitool execute the following command:

```
$ damvitool
```

When you run damvitool from command line without parameters it connects by default to the demo Chinook Database for SQLite.

To connect to your legacy database run damvitool with your database URL as parameter, like so:

```
$ damvitool --database sqlite:///damvitool/data/Chinook_Sqlite.sqlite
```

where *sqlite:///damvitool/data/Chinook_Sqlite.sqlite* is database URL in SQLAlchemy format (http://docs.sqlalchemy.org/en/rel_0_9/core/engines.html#database-urls).

2.2 Admin panel access

Default admin panel URL is `http://localhost:8080`

2.3 Supported databases

damvitool supports the same RDBMSs as SQLAlchemy (http://docs.sqlalchemy.org/en/rel_0_9/core/engines.html#supported-databases):

- MySQL (MariaDB)
- PostgreSQL
- SQLite
- Oracle
- Microsoft SQL Server
- Firebird
- Drizzle
- Sybase

- IBM DB2
- SAP Sybase SQL Anywhere
- MonetDB

CHAPTER 3

Authentication

To use authentication you need to use the user data file as a parameter when you start damvitool:

```
damvitool --users users.txt
```

Here is the example of users.txt file content:

```
user1=password1
user2=password2
```

CHAPTER 4

damvitool frontend usage

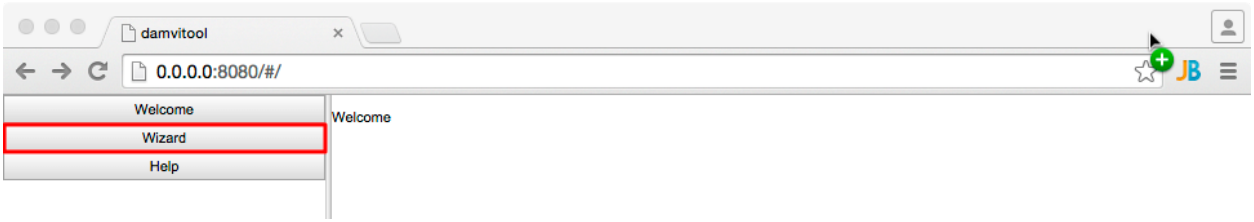
4.1 Open frontend

Open damvitool frontend in your browser. Default admin panel URL is `http://localhost:8080`

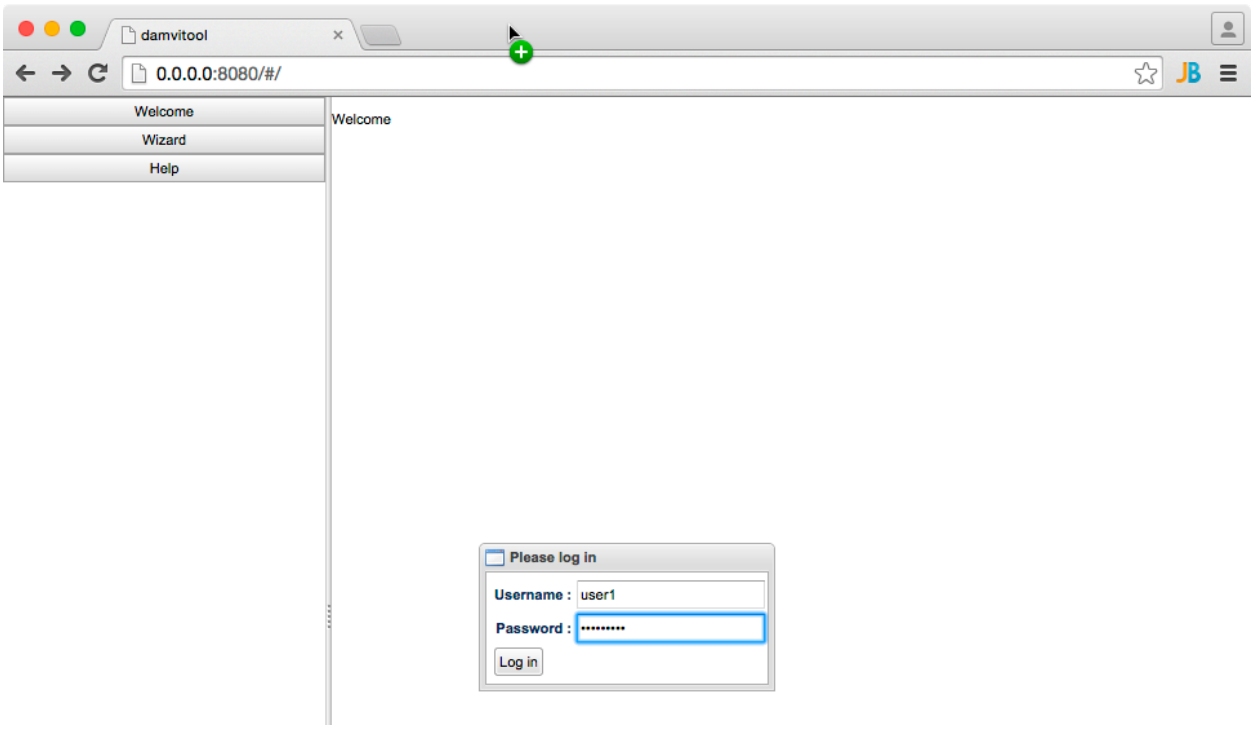
4.2 Construct new request to database

4.3 Build new database request

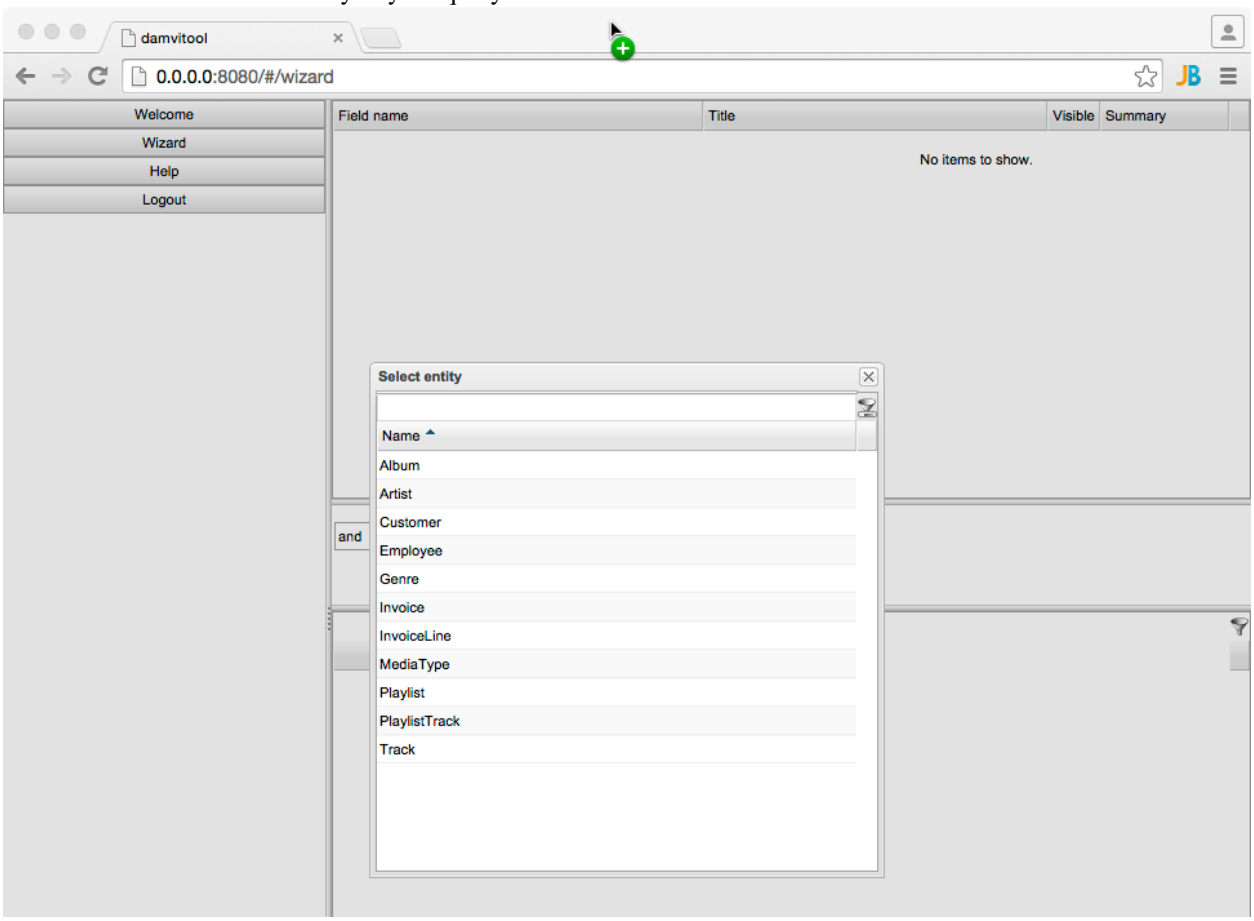
1. Click `Wizard` button.



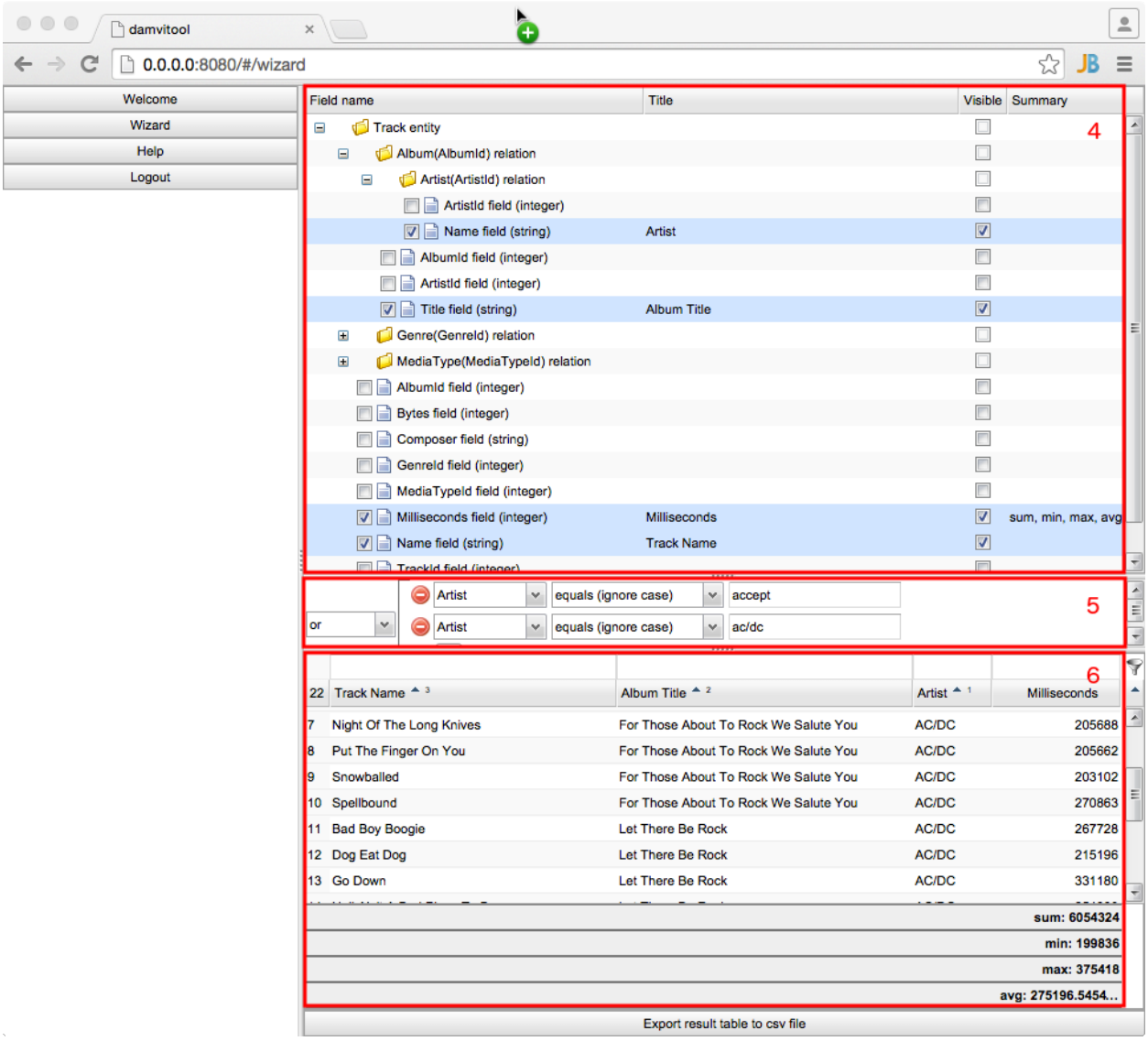
2. Login with the following credentials: *user1/password1*.



3. Choose root entity for your data query. If tables needed for your query don't have relations between them you can add another root entity to your query.



4. Choose relevant entities fields.
5. Set filter criteria.
6. View results.



CHAPTER 5

RESTful API

- **GET */api* - return login schema** Example:

  ```
  {
    "schema": "http://0.0.0.0:8080/api/database",
    "login": "http://0.0.0.0:8080/api/login"
  }
  ```

  where *login* - url for *POST* login request, *schema* - url to *GET* API schema

- **POST *http://0.0.0.0:8080/api/login* - open new session** Request data:

  ```
  {
    "username": "user1",
    "password": "password1"
  }
  ```

  Response data:

  ```
  {
    "username": "user1",
    "sessionId": "43bee700-b5ed-11e4-9596-a820662c96a1"
  }
  ```

  After login success, you must use HTTP Basic Authorization with *sessionId* instead of password.

- POST */api/logout* - close current session

  Send empty request.

- GET */api/database* - return api schema:

```
{
  "logout": "http://0.0.0.0:8080/api/logout",
  "mode": "http://0.0.0.0:8080/api/database/mode",
  "uni-grid-request": "http://0.0.0.0:8080/api/database/uni-grid-request",
  "entities": {
    "PlaylistTrack": {
      "get": "http://0.0.0.0:8080/api/database/tables/PlaylistTrack",
      "add": "http://0.0.0.0:8080/api/database/tables/PlaylistTrack/add",
      "record": "http://0.0.0.0:8080/api/database/tables/PlaylistTrack/recs/[{PlaylistId},{Track
    },
    "Invoice": {
      "get": "http://0.0.0.0:8080/api/database/tables/Invoice",
      "add": "http://0.0.0.0:8080/api/database/tables/Invoice/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Invoice/recs/[{InvoiceId}]"
```

```
    },
    "Employee": {
      "get": "http://0.0.0.0:8080/api/database/tables/Employee",
      "add": "http://0.0.0.0:8080/api/database/tables/Employee/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Employee/recs/[{EmployeeId}]"
    },
    "Artist": {
      "get": "http://0.0.0.0:8080/api/database/tables/Artist",
      "add": "http://0.0.0.0:8080/api/database/tables/Artist/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Artist/recs/[{ArtistId}]"
    },
    "MediaType": {
      "get": "http://0.0.0.0:8080/api/database/tables/MediaType",
      "add": "http://0.0.0.0:8080/api/database/tables/MediaType/add",
      "record": "http://0.0.0.0:8080/api/database/tables/MediaType/recs/[{MediaTypeId}]"
    },
    "Customer": {
      "get": "http://0.0.0.0:8080/api/database/tables/Customer",
      "add": "http://0.0.0.0:8080/api/database/tables/Customer/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Customer/recs/[{CustomerId}]"
    },
    "Track": {
      "get": "http://0.0.0.0:8080/api/database/tables/Track",
      "add": "http://0.0.0.0:8080/api/database/tables/Track/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Track/recs/[{TrackId}]"
    },
    "Album": {
      "get": "http://0.0.0.0:8080/api/database/tables/Album",
      "add": "http://0.0.0.0:8080/api/database/tables/Album/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Album/recs/[{AlbumId}]"
    },
    "InvoiceLine": {
      "get": "http://0.0.0.0:8080/api/database/tables/InvoiceLine",
      "add": "http://0.0.0.0:8080/api/database/tables/InvoiceLine/add",
      "record": "http://0.0.0.0:8080/api/database/tables/InvoiceLine/recs/[{InvoiceLineId}]"
    },
    "Genre": {
      "get": "http://0.0.0.0:8080/api/database/tables/Genre",
      "add": "http://0.0.0.0:8080/api/database/tables/Genre/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Genre/recs/[{GenreId}]"
    },
    "Playlist": {
      "get": "http://0.0.0.0:8080/api/database/tables/Playlist",
      "add": "http://0.0.0.0:8080/api/database/tables/Playlist/add",
      "record": "http://0.0.0.0:8080/api/database/tables/Playlist/recs/[{PlaylistId}]"
    }
  }
}
```


- GET */api/database/tables/{entity_name}* - return all records of entity type:


```
{
  "add": "http://0.0.0.0:8080/api/database/tables/Album/add",
  "data": [
    {
      "__links__": {"ForeignKey('Artist.ArtistId')": "http://0.0.0.0:8080/api/database/tables/Ar
      "__url__": "http://0.0.0.0:8080/api/database/tables/Album/recs/[1]",
      "ArtistId": 1,
      "Title": "For Those About To Rock We Salute You",
```

```
      "AlbumId": 1
    },
    {
      "__links__": {"ForeignKey('Artist.ArtistId')": "http://0.0.0.0:8080/api/database/tables/Ar
      "__url__": "http://0.0.0.0:8080/api/database/tables/Album/recs/[2]",
      "ArtistId": 2,
      "Title": "Balls to the Wall",
      "AlbumId": 2
    },
    ...
  ]
}
```

- GET *http://0.0.0.0:8080/api/database/tables/{EntityName}/recs/{EntityId}* - return entity:

```
{
  "__url__": "http://0.0.0.0:8080/api/database/tables/Artist/recs/[1]",
  "__links__": {},
  "ArtistId": 1,
  "Name": "AC/DC"
}
```

- GET *http://0.0.0.0:8080/api/database/mode* - return Map Of the Domain Entities (MODE):

```
{
  "entity01": {
    "id": "entity01", // (required) ID of the entity
    "name": "Entity 01", // (optional) Human readable name of the entity (i18n translatable?).
    "attributes": { /* set of attributes of the entity */
      "id": {
        "id": "id", // (required) Name of the attribute (column name)
        "name": "ID", // (optional) Human readable name of the attribute (i18n translatable?).
        "type": "integer" // (required) Type of the entity ([boolean, string, text, integer, num
      },
      "write_uid": {
        "id": "write_uid", // (required) Name of the attribute (column name)
        "name": "Write User ID", // (optional) Human readable name of the attribute (i18n transl
        "type": "integer" // (required) Type of the entity ([boolean, string, text, integer, num
      },
      ...
    },
    "relations": [ /* list of relations of the entity */
      {
        "own_attr": "write_uid", // (required) Code of the attribute of the 'entity01' that is u
        "rel_entity": "user", // (required) Code of the other entity from this relation.
        "rel_attr": "id", // (required) Code of the attribute of the 'other entity' from this re
        "type": "many2one" // (required) Type of the relation ([many2one, one2many])
      },
      ...
    ]
  },
  .
  .
  .
  "entityZZ": {
    ...
  }
}
```

- **POST *http://0.0.0.0:8080/api/database/uni-grid-request* - query UniGridRequest.** POST data:

```
{
  "entities": [
    { /* root entity with related entities and theirs attributes */
      "id":         "entity01", /* (required) entity name (table or view name) */
      "alias":      "entity01", /* (required) alias to use in the other rules (filtering, or
      "relation":   { /* (required) relation between parent entity and the current entity, e
      "attributes": [ /* (optional) list of the current entitie's attributes and related ent
        { /* attribute or related entity */
          "id":       "id",           /* (required) ID of the entitie's attributes */
          "alias":    "entity01_id",  /* (required) alias to use this attributes in the othe
          "selected": "true",         /* (required) 'true' - this attribute will be included
          "summaries": ["sum", "avg"] /* array of the summary types for attribute */
        },
        ...
      ]
    },
    ...
  ],
  "where":    { /* filtering */ },
  "order":    [ /* sorting */ ],
  "offset":   0, /* pagination */
  "limit":    100 /* pagination */
};
```

Example request:

```
{
  "unigrid": {
    "entities": [
      {
        "attributes": [
          {
            "entity": {
              "attributes": [
                {
                  "id": "Name",
                  "alias": "Album_1_Artist_ArtistId_Name",
                  "selected": true
                }
              ],
              "id": "Artist",
              "relation": {"attr_parent": "ArtistId"}
            }
          },
          {
            "id": "Title",
            "alias": "Album_1_Title",
            "selected": true
          }
        ],
        "id": "Album",
        "relation": null
      }
    ],
    "where": {
      "cond": {
        "with": "AND",
```

```
        "entries": [
          {
            "func": {
              "name": "ILIKE",
              "args": [
                {"alias": "Album_1_Title"},
                {"value": "rest"}
              ]
            }
          }
        ]
      }
    },
    "order": [],
    "offset": 0,
    "limit": 75
  }
}
```

Example result:

```
{
  "data": [
    [
      "Accept",
      "Restless and Wild"
    ]
  ],
  "cols": [
    "Album_1_Artist_ArtistId_Name",
    "Album_1_Title"
  ],
  "size": {
    "offset": 0,
    "total": 1,
    "frame": 1
  }
}
```

CHAPTER 6

Developing damvitool

6.1 Install damvitool for development

Clone damvitool from github and go to damvitool directory:

```
$ git clone https://github.com/praxigento/damvitool.git
$ cd damvitool
```

Run bootstrap.py to set up buildout:

```
$ python bootstrap.py
```

Run buildout, which downloads and installs various dependencies and tools.

6.2 Running the tests

You can run the backend tests using py.test. Buildout has installed it for you in the bin subdirectory of your project:

```
$ bin/py.test damvitool
```

To run frontend tests, you must initialize node.js environment:

```
$ npm install
```

After that, run tests:

```
$ npm test
```

CHAPTER 7

Roadmap

- Ability to save queries
- Extended authorisation support with fine grained control of access to queries/tables
- Editing of records
- Charting engine for data visualization

CHAPTER 8

Indices and tables

- *genindex*
- *modindex*
- *search*