dbhelp.py: Python database helper module

John W. Shipman
2005-12-27 14:49

Table of Contents
1. Introduction ............................................................................................................................ 1
   1.1. How to get this publication ............................................................................................ 2
2. Using the dbhelp.py module .................................................................................................. 2
3. class Database: The database connection object ................................................................. 2
   3.1. Database.dbapi: Interface to the MySQLdb module ...................................................... 3
   3.2. Database.getTable(): Find a table ........................................................................... 3
   3.3. Database.execute(): Execute an SQL command ......................................................... 4
4. class Table: The table object .................................................................................................. 4
   4.1. Table.db: Database back pointer ................................................................................ 4
   4.2. Table.name: Table name ............................................................................................. 4
   4.3. Table.columnList: Column structure .......................................................................... 4
   4.4. Table.query(): Query a set of table rows .................................................................... 4
5. class Column: Description of one table column ....................................................................... 5
6. class Row: One row of a table ................................................................................................. 5
7. class QueryResult: One set of result rows ........................................................................... 6
8. class SortSpec: Sort key specifier .......................................................................................... 6

1. Introduction

The Python language has a well-defined database applications program interface (DBAPI). For documentation, see the Database topic guide [1].

However, it is still necessary for the user program to interact with this interface using SQL (Structured Query Language). Learning SQL is nontrivial; it is a subtle and complex language.

The dbhelp.py module is an object-oriented layer on top of the Python 2.0 DBAPI that insulates the user (to some extent) from the need to use SQL directly. For operations not provided by the classes of the dbhelp.py module, one can always send specific SQL to the interface.

This version will operate only with MySQL. An earlier version of this module was ported to Oracle and Sybase as well as MySQL, and this server independence should not be difficult to add later. Also, this (preliminary) specification does not support any operations that modify the database, as the immediate application (study of the xt3 database) requires only read access.

Before proceeding, you should know:

• The Python programming language is well-described at the Python website [2].

• A few things about SQL, primarily the general theory of relational databases and the field types supported by MySQL. See the MySQL AB manufacturer’s web site [3] for documentation.

The implementation of this module is described in a companion document, *dbhelp.py: Internal maintenance specification* [4].

1.1. How to get this publication

This publication is available in Web form [5] and also as a PDF document [6]. Please forward any comments to tcc-doc@nmt.edu.

2. Using the *dbhelp.py* module

To use the *dbhelp.py* module, you must first import it using one of these forms:

```python
import dbhelp
from dbhelp import *
```

If you use the first form, you can refer to any object `O` from the module as `"dbhelp.O"`. If you use the second form, you can refer to it simply as `"O"`.

Here are the classes defined in the *dbhelp.py* module:

**Database**

Represents a connection to a specific database. See Section 3, “class Database: The database connection object” (p. 2).

**Table**

Represents one table. See Section 4, “class Table: The table object” (p. 4).

**Column**

Describes one column of a table. See Section 5, “class Column: Description of one table column” (p. 5).

**Row**

Describes one row of a table. See Section 6, “class Row: One row of a table” (p. 5).

**QueryResult**

When you query the database, you get back a `QueryResult` instance. This instance is a Python generator, which generates the results of that query as a sequence of zero or more `Row` objects. See Section 7, “class QueryResult: One set of result rows” (p. 6).

**SortSpec**

When you want to return query results in some specific sorted order, you can use one or more `SortSpec` instances to describe the sort keys. See Section 8, “class SortSpec: Sort key specifier” (p. 6).

3. class Database: The database connection object

Connecting to the database has a certain cost in time and resources, so it is best to do it only once in each application.

Here is the calling sequence for the constructor:

```
```
Database ( options )

The options argument is a dictionary that supplies necessary values for the connection. It should have keys and values from this table:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>The hostname of the database server.</td>
</tr>
<tr>
<td>db</td>
<td>Name of the default database you will be using.</td>
</tr>
<tr>
<td>user</td>
<td>The username you want to use for the connection.</td>
</tr>
<tr>
<td>passwd</td>
<td>Password for this username.</td>
</tr>
</tbody>
</table>

Attributes and methods on instances of the Database object are described in the following sections.

### 3.1. Database.dbapi: Interface to the MySQLdb module

The .dbapi attribute of the Database instance is the MySQLdb module, which contains a number of useful functions and variables. For the full description of this interface, see Python Database API specification v2.0 [7]. Contents include:

- **Date(year, month, day)**
  Type constructor for date columns.

- **Time(hour, minute, second)**
  Type constructor for time columns.

- **Timestamp(year, month, day, hour, minute, second)**
  Type constructor for timestamp columns.

- **Binary(string)**
  Type constructor for a binary object.

- **STRING**
  Type object for string columns.

- **BINARY**
  Type object for binary columns.

- **NUMBER**
  Type object for numeric columns.

- **DATETIME**
  Type object for date and time columns.

- **ROWID**
  Type object for “row ID” columns.

### 3.2. Database.getTable(): Find a table

To retrieve the Table object representing a given table in a Database object D:

D.getTable ( tableName )

where tableName is the name of the desired table. If successful, this method returns an instance of the Table class representing that table. If there is no such table, KeyError is raised.

---

3.3. **Database.execute(): Execute an SQL command**

This is a low-level method that the `Database` class uses to execute an SQL command against the database. If `D` is an instance of the `Database` class, the calling sequence is:

```
D.execute ( sql, values )
```

Arguments:

- `sql`
  - An SQL command, as a string.

- `values`
  - If the SQL command has parameters (that is, it contains "'%'" sequences indicating where values are to be substituted), the values of those parameters are provided in this argument, which must be a tuple. If the SQL command has no parameters, omit this optional argument.

A cursor object is returned by this method. If the SQL command fails, an exception may be raised.

4. **class Table: The table object**

An instance of this class represents one table in the database. Do not directly instantiate this class: instead, see Section 3.2, “`Database.getTable(): Find a table`” (p. 3).

4.1. **Table.db**: Database back pointer

This attribute holds a back pointer to the containing `Database` object.

4.2. **Table.name**: Table name

Holds the name of the table.

4.3. **Table.columnList**: Column structure

This attribute holds a list of `Column` objects that describe the columns of this table. See Section 5, “class `Column`: Description of one table column” (p. 5).

4.4. **Table.query(): Query a set of table rows**

This method is used to query a table for a set of rows, possibly to be returned in a given order. For a `Table` object `T`, the calling sequence is:

```
T.query ( colPairs=None, sortList=None )
```

Arguments:

- `colPairs`
  - This argument selects which rows you want from the table. It is a sequence of two-element sequences 
    `(n, v)` where `n` is the name of a column in the table and `v` is the value you want to match.

  If you omit this argument, you get all the rows in the table.

- `sortList`
  - This optional argument specifies how you want the result set to be ordered. If supplied, it consists of a sequence of one or more `SortSpec` objects. The first gives the primary key, the second specifies the secondary key, and so forth.
If omitted, you get the rows in whatever order the database feels like giving them to you.

The result returned by this method is a generator that generates the rows of the query result as instances of the Row class. See Section 6, “class Row: One row of a table” (p. 5).

5. class Column: Description of one table column

An instance of the Column class describes one column of a table. Attributes include:

- `.db`
  The containing Database object.

- `.colName`
  The name of the column.

- `.colType`
  The SQL type name of the column.

- `.nullable`
  A Boolean value, true iff the column can contain nulls.

- `.key`
  Describes the key status of this column. If it is the primary key, it will contain the string "PRI". If part of a multiple key, it will contain "MUL". If the column is not a key, it will contain "" (the empty string).

- `.default`
  Gives the default value as a string. If there is no default value for this column, this attribute will contain None.

6. class Row: One row of a table

Results returned from queries are represented as instances of the Row class.

To get the value of one column named $N$ from a Row object $R$, simply use the expression $R[N]$. $N$ can also be an integer, if you know in advance the index of the column from which you want the result.

If the value of $N$ is none of the above, this operation will raise a KeyError exception.

Attributes and methods of a Row objects include:

- `.db`
  Containing Database object.

- `.keys()`
  Returns a list of the column names in self, in the same order they were returned by the database.

- `.show()`
  Returns a multiline string displaying all the columns of the row, in alphabetical order by column name, in this format:

  `colName: 'value'`
7. class QueryResult: One set of result rows

When you use the `.query()` method on a Table object, you get an instance of the QueryResult class. Such an instance is an iterator, and yields a sequence of zero or more Row objects.

Instances contain these attributes and methods:

- `.db`
  The containing Database object.

- `.typeCode(colName)`
  Returns the type code of the column whose name is `colName`. Type codes will match one of the type objects in the `.dbapi` attribute of the Database object. See Section 3.1, “Database.dbapi: Interface to the MySQLdb module” (p. 3).

- `.displaySize(colName)`
  Returns the display size (in characters) of the value whose column name is `colName`.

8. class SortSpec: Sort key specifier

To specify a key for ordering results into some sorted order, use an instance of the SortSpec class.

The constructor has this calling sequence:

```
s = SortSpec ( colName, desc=0 )
```

Arguments to this constructor:

- `colName`
  Name of the column to be used as a sort key.

- `desc`
  If true, results are returned in descending order by this key. If false, results are returned in ascending order.