Using CSV as an indexed persistent layer

Hervé Cauwelier

herve@itaapy.com

Introduction: the speaker

- Python developer for 4 years
- Former Zope developer
- Core developer of the itools library and the itools.cms framework
- Software architect at Itaapy, France

Summary

- The Python *csv* module
- Using the itools library
- Example
- Conclusion
- References

Out of the box: the *csv* module

Shipped with the standard library but:

- low-level
- only loads byte strings
- no support for Unicode
- basic API

The *csv* module is merely an interface to load and save byte strings.
It offers no way to typecast each column, as in an SQL table.

Using the itools library

- loading and saving high-level objects
- automatic type marshalling
- rows as intelligent objects, not just lists
- richer API
- easy to use and extend
- access row values by column name instead of number

Pushing the SQL comparison further:

- indexing columns
- query API
- stored procedures

Summary

- The Python *csv* module
- Using the itools library
- Example
- Conclusion
- References

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: our CSV file (1/2)

Column name	Data type
client_id	Integer
surname	Unicode
name	Unicode
certified	Boolean
email	Unicode
last_pay_date	DateTime
num_of_computers	Integer
register_date	Date
discount	Decimal

Example: our CSV file (2/2)

- We know the structure of our file: the **schema.**
- Each column has a known type: Integer, Unicode, Boolean, DateTime, Date, Decimal.
- The columns are ordered: #0 is always *client_id*

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: writing the handler (1/3)

Required imports:

The datatypes

```
from itools.datatypes import (Integer, Unicode,
                              Boolean, DateTime,
                              Date, Decimal)
```

The itools.CSV module

```
from itools.csv import CSV
```

Example: writing the handler (2/3)

```
class Clients(CSV):
    schema = {
        'client_id': Integer,
        'surname': Unicode,
        'name': Unicode,
        'certified': Boolean,
        'email': Unicode,
        'last_pay_date': DateTime,
        'num_of_computers': Integer,
        'register_date': Date,
        'discount': Decimal}
```

Example: writing the handler (3/3)

Base class (cont.):

```
class Clients(CSV):
    [...]

    columns = ['client_id', 'surname', 'name',
               'certified', 'email',
               'last_pay_date',
               'num_of_computers',
               'register_date', 'discount']
```

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: using the API (1/5)

Getting an instance of our clients handler:

```
>>> from itools.handlers import get_resource
>>> resource = get_resource('clients.csv')
>>> handler = Clients(resource)
```

Traversing rows:

```
>>> handler.get_rows()
<generator object at 0xb796afcc>
```

Traversing the ten first rows:

```
>>> handler.get_rows(range(10))
<generator object at 0xb7d99d8c>
```

Example: using the API (2/5)

Reading first row:

```
>>> handler.get_row(0)
[1, u"Macuk", u"Piotr", True, u"piotr@macuk.pl",
datetime.datetime(2006, 06, 03, 11, 59, 46), 3,
datetime.date(2005, 11, 29), Decimal("0.2")]
```

It looks like a list but...

```
>>> type(handler.get_row(0))
<class 'itools.csv.itools_csv.Row'>
```

Example: using the API (3/5)

Working with a copy not to change the handler state:

```
>>> row = handler.get_row(0).copy()
```

Reading attributes:

```
>>> row[0]
1
>>> row.register_date
datetime.date(2005, 11, 29)
```

Writing attributes:

```
>>> row.last_pay_date = datetime.now()
```

Example: using the API (4/5)

Some SQL-like queries:

- select count(*) from clients;

```
>>> handler.get_nrows()
235
```

- select avg(discount) from clients;

```
>>> all = [r.discount
...         for r in handler.get_rows()]
>>> sum(all) / handler.get_nrows()
Decimal("0.135")
```

Hint: write this as a method of the *Client* class.

Example: using the API (5/5)

Deleting rows:

```
>>> handler.del_row(2)
>>> handler.del_rows([41, 77, 214])
```

Adding a row:

```
>>> from itools.csv import Row
>>> new_row = Row([440, u"Doe", u"John",
u"Liverpool", u"john.doe@example.com", None,
4, datetime.now(), Decimal.encode("0.64")])
>>> handler.add_row(new_row)
```

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: indexing

Declaring the index types:

```
schema = {
    'client_id': Integer(index='keyword'),
    'surname': Unicode(index='text'),
    'name': Unicode(index='text'),
    'certified': Unicode(index='keyword'),
    'email': Unicode(index='keyword'),
    'last_pay_date': DateTime(index='keyword'),
    'num_of_computers': Integer(index='keyword'),
    'register_date': Date(index='keyword'),
    'discount': Decimal(index='keyword')}
```

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: searching (1/4)

- select * from clients where client_id=123;

```
>>> results = handler.search(client_id=123)
>>> results
[73]
>>> handler.get_row(results[0])
```

- select * from clients where name='john' and num_of_computers=4;

```
>>> for row_num in handler.search(name=u"john"
...         num_of_computers=4):
...     row = handler.get_row(row_num)
```

Example: searching (2/4)

Complex queries: using the *itools.catalog* package, and specifically the *itools.catalog.queries* module.

Language: Equal, Range, And, Or

Usage:

- Equal(name, value)
- Range(name, min, max)
- And(query1, query2, queryN...)
- Or(query1, query2, queryN...)

Example: searching (3/4)

From simple queries to complex queries:

```
>>> from itools.catalog.queries import Equal
>>> handler.search(query=Equal('client_id', 123))
```

Queries can be combined:

```
>>> from itools.catalog.queries import And
>>> handler.search(query=
...   And(Equal('name', u"john"),
...       Equal('num_of_computers', 4)))
```

Example: searching (4/4)

Combining can get you far:

```
>>> from itools.catalog.queries import Or
>>> registered_before_june_1st = Range(
...     'register_date', None, date(2006, 6, 1))
>>> discount_10_to_20 = Range('discount',
...     Decimal(".1"), Decimal(".2"))
>>> computers_10_or_more = Range(
...     'num_of_computers', 10, None)
>>> handler.search(query=
...     And(Or(registered_before_june_1st,
...            discount_10_to_90),
...         computers_10_more))
```

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: custom row class

Straightforward:

```
class Client(Row):
    [...]


class Clients(CSV):
    row_class = Client
```

Now all rows will be instances of *Client*

Benefits:

- change default behaviour
- extend the API with your use cases

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: writing a datatype

```
from itools.datatypes import DataType
from directory import Directory

class User(DataType):
    @staticmethod
    def decode(data):
        return Directory.get_user(data)

    @staticmethod
    def encode(value):
        return value.username
```

Example: summary

- Our CSV file
- Writing the handler
- Using the API
- Indexing
- Searching
- Custom row class
- Writing a datatype
- Serialising

Example: serialising

Dumping the handler state in CSV format into a string:

```
>>> handler.to_str()
"1","Macuk","Piotr","1","piotr@macuk.pl",
"2006-06-03 11:59:46","3","2005-11-29","0.2"
[...]
```

See how objects are represented as strings.

Synchronising the handler and its resource:

```
>>> handler.save_state()
```

Now *clients.csv* is written with all of our changes in memory.

Summary

- The Python *csv* module
- Using the itools library
- Example
- Conclusion
- References

Conclusion

CSV is worth investing on:

- easy to introspect
- easy to integrate
- import from Excel/Calc at no cost
- export back to Excel/Calc at no cost

itools.csv makes it easy:

- automatic loading and saving
- customisable object model
- easy indexation
- powerful search features

References

CSV on Wikipedia
http://en.wikipedia.org/wiki/Comma-separated_values

Request for Comment 4180 http://tools.ietf.org/html/4180

Python *csv* documentation
http://python.org/doc/2.4/lib/module-csv.html

itools http://ikaaro.org/itools

itools documentation
http://download.ikaaro.org/doc/itools/itools.html

Contact Hervé Cauwelier herve@itaapy.com