django-book-manager

Release 0.3.2

Caltech IMSS ADS

Jan 27, 2023

CONTENTS

Current version is 0.3.2.

Github Repository: https://github.com/caltechads/django-book-manager

This reusable Django application provides models suitable for managing a list of books with ratings, somewhat like a private Goodreads.

Its real purpose is to provide sample models, with sample data, for use in testing other Django libraries. Often, when authoring new Django libraries, we need a simple example application to use so that we can test out our code.

CHAPTER

ONE

GETTING IT

You can get `django-book-manager` by using pip:

```
pip install django-book-manager
```

If you want to install it from source, grab the git repository from GitHub and run `setup.py`:

```
git clone git://github.com/caltechads/django-book-manager.git
cd django-book-manager
python setup.py install
```

CHAPTER

TWO

INSTALLING IT

To enable `django-book-manager` in your project you need to add it to `INSTALLED_APPS` in your project's `settings.py` file:

```
INSTALLED_APPS = (
    ...
    'book_manager',
    ...
)
```

Then, apply the migrations to add the schema to your database:

```
./manage.py migrate
```

CHAPTER

THREE

USING IT

3.1 Models

`django-book-manager` provides these models:

- *Book*: a book with title, slug, publishing dates, number of pages, authors, etc.
- *Author*: an author. *Book* has a many to many relationship with this
- *BookAuthor*: this is a many to many through table between *Book* and *Author* that exists to record billing order of authors on a book (first author, second author, etc.)
- *Publisher*: a publisher. *Book* has a foreign key relationship with this
- *Binding*: a binding (hardcover, softcover, ebook, ...). *Book* has a foreign key relationship with this
- *Reading*: a reading record of a book by a reader. This is a many to many through table between *Book* and the `AUTH_USER_MODEL` that records a rating, review, notes, date read, etc. for a particular user.
- *Shelf*: a collection of *Reading* objects, used by readers to classify books

3.2 Management commands

`django-book-manager` also supplies a command that can be used to load a Goodreads user library export into Django, splitting it into all the above models as appropriate.

To generate an export from Goodreads, go to your Goodreads account and:

- Click "My Books"
- At the bottom of that page, click "Import and Export"
- At the top of that page, click "Export Library"

To load the CSV thus generated into Django, first create a user for yourself in Django, then:

```
./manage.py import_csv <csvfile> <username>
```

A sample Goodreads export is available in this repository as *sandbox/data/books.csv*.

CHAPTER

FOUR

FEATURES:

...

4.1 Management commands

4.1.1 import_csv

synposis
Imports a Goodreads CSV export into our database and associate the books listed therein with a Django user.

The `import_csv` command imports a Goodreads CSV export into our database, creating or updating `Book` objects (with their dependent `Binding`, `Publisher` and `Author` objects), and associates them with user by creating a `Reading` object for each one, and adding the `Reading` to a `Shelf` as appropriate.

Why?

Goodreads was the model for this package, and its export file matches our data structure. It was an easy to get set of rich data.

The export file should have the columns named in the class documentation for `GoodreadsImporter`.

Usage

To generate an export from Goodreads, go to your Goodreads account and:

- Click "My Books"
- At the bottom of that page, click "Import and Export"
- At the top of that page, click "Export Library"

To load that export into the database and associate it with a user with username `username`:

```
$ ./manage.py import_csv goodreads.csv username
```

To load the export and overwite any existing book data in the database with that in the file:

```
$ ./manage.py import_csv --overwrite goodreads.csv username
```

4.2 Developer Interface

4.2.1 Models

This part of the documentation covers all the models provided by `django-book-manager`.

Books

class book_manager.models.**Book**(*id*, *created*, *modified*, *title*, *slug*, *isbn*, *isbn13*, *num_pages*, *year_published*, *original_publication_year*, *binding*, *publisher*)

Database table: `book_manager_book`

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **title** (*CharField*) – Book Title. The title of the book
- **slug** (*AutoSlugField*) – Slug. Used in the URL for the book. Must be unique.
- **isbn** (*CharField*) – ISBN
- **isbn13** (*CharField*) – ISBN
- **num_pages** (*PositiveIntegerField*) – Num Pages
- **year_published** (*IntegerField*) – Year Published
- **original_publication_year** (*IntegerField*) – Original Publication Year

Relationship fields:

Parameters

- **binding** (ForeignKey to *Binding*) – Binding (related name: *books*)
- **publisher** (ForeignKey to *Publisher*) – Publisher (related name: *books*)
- **authors** (ManyToManyField to *Author*) – Authors (related name: *books*)
- **readers** (ManyToManyField to User) – Readers (related name: books)

Reverse relationships:

Parameters

- **bookauthor** (Reverse ForeignKey from *BookAuthor*) – All book authors of this book (related name of *book*)
- **readings** (Reverse ForeignKey from *Reading*) – All readings of this book (related name of *book*)

exception DoesNotExist

exception MultipleObjectsReturned

get_next_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=True*, ***kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=True*, ***kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

authors: ManyToManyField

Type: ManyToManyField to *Author*

Authors (related name: *books*)

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager

binding: ForeignKey

Type: ForeignKey to *Binding*

Binding (related name: *books*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

binding_id

Internal field, use *binding* instead.

bookauthor_set

Type: Reverse ForeignKey from *BookAuthor*

All book authors of this book (related name of *book*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

isbn: Field

Type: CharField

ISBN

A wrapper for a deferred-loading field. When the value is read from this

isbn13: Field

Type: CharField

ISBN

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

num_pages: Field

Type: PositiveIntegerField

Num Pages

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

original_publication_year: Field

Type: IntegerField

Original Publication Year

A wrapper for a deferred-loading field. When the value is read from this

property other_authors: QuerySet

Return all authors other than the top-billed author for this book. These are the authors with order>1 in our *BookAuthor* through table.

Returns

The queryset of *Author* objects for the non-primary author.

property primary_author: *Author*

Return the top-billed author for this book. This is the author with order=1 in our *BookAuthor* through table.

Returns

The *Author* object for the primary author

publisher: ForeignKey

Type: ForeignKey to *Publisher*

Publisher (related name: *books*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

publisher_id

Internal field, use *publisher* instead.

readers: ManyToManyField

Type: ManyToManyField to User

Readers (related name: books)

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

Pizza.toppings and Topping.pizzas are ManyToManyDescriptor instances.

Most of the implementation is delegated to a dynamically defined manager

readings

Type: Reverse ForeignKey from *Reading*

All readings of this book (related name of *book*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager

slug: Field

Type: AutoSlugField

Slug. Used in the URL for the book. Must be unique.

A wrapper for a deferred-loading field. When the value is read from this

title: Field

Type: CharField

Book Title. The title of the book

A wrapper for a deferred-loading field. When the value is read from this

year_published: Field

Type: IntegerField

Year Published

A wrapper for a deferred-loading field. When the value is read from this

class book_manager.models.**Author**(**args*, ***kwargs*)

Database table: book_manager_author

An author of a *Book*. Books can have multiple authors.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **first_name** (*CharField*) – First name
- **last_name** (*CharField*) – Last name
- **middle_name** (*CharField*) – Middle name
- **full_name** (*CharField*) – Full name

Reverse relationships:

Parameters

- **books** (Reverse ManyToManyField from *Book*) – All books of this author (related name of *authors*)
- **bookauthor** (Reverse ForeignKey from *BookAuthor*) – All book authors of this author (related name of *author*)

exception DoesNotExist

exception MultipleObjectsReturned

get_next_by_created(**, field=<django_extensions.db.fields.CreationDateTimeField: created>, is_next=True, **kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(**, field=<django_extensions.db.fields.ModificationDateTimeField: modified>, is_next=True, **kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(**, field=<django_extensions.db.fields.CreationDateTimeField: created>, is_next=False, **kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(**, field=<django_extensions.db.fields.ModificationDateTimeField: modified>, is_next=False, **kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

bookauthor_set

Type: Reverse ForeignKey from *BookAuthor*

All book authors of this author (related name of *author*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

books

Type: Reverse `ManyToManyField` from *Book*

All books of this author (related name of *authors*)

Accessor to the related objects manager on the forward and reverse sides of a many-to-many relation.

In the example:

```
class Pizza(Model):
    toppings = ManyToManyField(Topping, related_name='pizzas')
```

`Pizza.toppings` and `Topping.pizzas` are `ManyToManyDescriptor` instances.

Most of the implementation is delegated to a dynamically defined manager

created

Type: `CreationDateTimeField`

Created

A wrapper for a deferred-loading field. When the value is read from this

first_name: Field

Type: `CharField`

First name

A wrapper for a deferred-loading field. When the value is read from this

full_name: Field

Type: `CharField`

Full name

A wrapper for a deferred-loading field. When the value is read from this

id

Type: `AutoField`

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

last_name: Field

Type: `CharField`

Last name

A wrapper for a deferred-loading field. When the value is read from this

middle_name: *Field*

Type: CharField

Middle name

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

class book_manager.models.**BookAuthor**(**args*, ***kwargs*)

Database table: book_manager_bookauthor

This is a through table between *Book* and *Author* that allows us to keep our book authors in the correct order.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **order** (*PositiveIntegerField*) – Author order

Relationship fields:

Parameters

- **book** (ForeignKey to *Book*) – Book (related name: bookauthor)
- **author** (ForeignKey to *Author*) – Author (related name: bookauthor)

exception DoesNotExist

exception MultipleObjectsReturned

author

Type: ForeignKey to *Author*

Author (related name: bookauthor)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

author_id

Internal field, use *author* instead.

book

Type: ForeignKey to *Book*

Book (related name: bookauthor)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

book_id

Internal field, use *book* instead.

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

order

Type: PositiveIntegerField

Author order

A wrapper for a deferred-loading field. When the value is read from this

class book_manager.models.**Publisher**(**args*, ***kwargs*)

Database table: book_manager_publisher

A publisher of a *Book*. Books have zero or one publishers.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **name** (*CharField*) – Publisher name. Publisher name

Reverse relationships:

Parameters

books (Reverse ForeignKey from *Book*) – All books of this publisher (related name of *publisher*)

exception DoesNotExist

exception MultipleObjectsReturned

get_next_by_created(**, field=<django_extensions.db.fields.CreationDateTimeField: created>, is_next=True, **kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_modified(**, field=<django_extensions.db.fields.ModificationDateTimeField: modified>, is_next=True, **kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(**, field=<django_extensions.db.fields.CreationDateTimeField: created>, is_next=False, **kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

books

Type: Reverse ForeignKey from *Book*

All books of this publisher (related name of *publisher*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

name: Field

Type: CharField

Publisher name. Publisher name

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

class book_manager.models.**Binding**(**args*, ***kwargs*)

Database table: book_manager_binding

A binding of a *Book* ("ebook", "mass market paperback", "hardcover", etc.). Books have zero or one bindings.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **name** (*CharField*) – Binding type. Binding type

Reverse relationships:

Parameters

books (Reverse ForeignKey from *Book*) – All books of this binding (related name of *binding*)

exception DoesNotExist

exception MultipleObjectsReturned

books

Type: Reverse ForeignKey from *Book*

All books of this binding (related name of *binding*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

`Parent.children` is a `ReverseManyToOneDescriptor` instance.

Most of the implementation is delegated to a dynamically defined manager

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

name: Field

Type: CharField

Binding type. Binding type

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

Readings

A `Reading` is a single person's use of a *Book*. It records that person's notes, ratings, reading count, etc.

class book_manager.models.**Reading**(**args*, ***kwargs*)

Database table: book_manager_reading

This model holds user-specific data about a reading of a *Book*

Parameters

- **id** (*AutoField*) – Primary key: ID
- **created** (*CreationDateTimeField*) – Created
- **modified** (*ModificationDateTimeField*) – Modified
- **rating** (*PositiveIntegerField*) – Rating
- **private_notes** (*TextField*) – Private Notes. Private notes that only you can see
- **review** (*TextField*) – Review. Notes that anyone can see
- **read_count** (*PositiveIntegerField*) – Read count. How many times you've read this book
- **date_added** (*DateField*) – Date added. Date this book was added to your reading list

- **date_read** (*DateField*) – Date read. Date you first read this book

Relationship fields:

Parameters

- **reader** (ForeignKey to User) – Reader (related name: readings)
- **book** (ForeignKey to *Book*) – Book (related name: *readings*)
- **shelf** (ForeignKey to *Shelf*) – Shelf (related name: *readings*)

exception DoesNotExist

exception MultipleObjectsReturned

get_next_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=True*, ***kwargs*)

Finds next instance based on *created*. See get_next_by_FOO for more information.

get_next_by_date_added(*, *field=<django.db.models.DateField: date_added>*, *is_next=True*, ***kwargs*)

Finds next instance based on *date_added*. See get_next_by_FOO for more information.

get_next_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=True*, ***kwargs*)

Finds next instance based on *modified*. See get_next_by_FOO for more information.

get_previous_by_created(*, *field=<django_extensions.db.fields.CreationDateTimeField: created>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *created*. See get_previous_by_FOO for more information.

get_previous_by_date_added(*, *field=<django.db.models.DateField: date_added>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *date_added*. See get_previous_by_FOO for more information.

get_previous_by_modified(*, *field=<django_extensions.db.fields.ModificationDateTimeField: modified>*, *is_next=False*, ***kwargs*)

Finds previous instance based on *modified*. See get_previous_by_FOO for more information.

book

Type: ForeignKey to *Book*

Book (related name: *readings*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

book_id

Internal field, use *book* instead.

created

Type: CreationDateTimeField

Created

A wrapper for a deferred-loading field. When the value is read from this

date_added

Type: DateField

Date added. Date this book was added to your reading list

A wrapper for a deferred-loading field. When the value is read from this

date_read

Type: DateField

Date read. Date you first read this book

A wrapper for a deferred-loading field. When the value is read from this

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

modified

Type: ModificationDateTimeField

Modified

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

private_notes

Type: TextField

Private Notes. Private notes that only you can see

A wrapper for a deferred-loading field. When the value is read from this

rating

Type: PositiveIntegerField

Rating

A wrapper for a deferred-loading field. When the value is read from this

read_count

Type: PositiveIntegerField

Read count. How many times you've read this book

A wrapper for a deferred-loading field. When the value is read from this

reader

Type: ForeignKey to User

Reader (related name: readings)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

reader_id

Internal field, use *reader* instead.

review

Type: TextField

Review. Notes that anyone can see

A wrapper for a deferred-loading field. When the value is read from this

shelf

Type: ForeignKey to *Shelf*

Shelf (related name: *readings*)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

shelf_id

Internal field, use *shelf* instead.

class book_manager.models.**Shelf**(**args*, ***kwargs*)

Database table: book_manager_shelf

This model is used to organize *Reading* instances for a user into buckets ("read", "to-read", "abandoned"). Shelves are per-user.

Parameters

- **id** (*AutoField*) – Primary key: ID
- **name** (*CharField*) – Shelf name. Name of a shelf on which books can live

Relationship fields:

Parameters

reader (ForeignKey to User) – Reader (related name: shelves)

Reverse relationships:

Parameters

readings (Reverse ForeignKey from *Reading*) – All readings of this shelf (related name of *shelf*)

exception DoesNotExist

exception MultipleObjectsReturned

id

Type: AutoField

Primary key: ID

A wrapper for a deferred-loading field. When the value is read from this

name: Field

Type: CharField

Shelf name. Name of a shelf on which books can live

A wrapper for a deferred-loading field. When the value is read from this

objects = <django.db.models.Manager object>

reader

Type: ForeignKey to User

Reader (related name: shelves)

Accessor to the related object on the forward side of a many-to-one or one-to-one (via ForwardOneToOneDescriptor subclass) relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

reader_id

Internal field, use *reader* instead.

readings

Type: Reverse ForeignKey from *Reading*

All readings of this shelf (related name of *shelf*)

Accessor to the related objects manager on the reverse side of a many-to-one relation.

In the example:

```
class Child(Model):
    parent = ForeignKey(Parent, related_name='children')
```

Parent.children is a ReverseManyToOneDescriptor instance.

Most of the implementation is delegated to a dynamically defined manager

4.2.2 Widgets

This part of the documentation covers all the reusable django-wildewidgets widgets provided by django-book-manager.

4.2.3 Importers

class book_manager.importers.**GoodreadsImporter**

Usage: GoodreadsImporter().run(csv_filename, user)

Import data into our database from a Goodreeads CSV Export.

- Import any new *book_manager.models.Binding*, *book_manager.models.Publisher*, and *book_manager.models.Author* instances
- Import the book from each row as a *book_manager.models.Book*

- Import the user specific data from each row as a *book_manager.models.Reading* associated with the user `user`

A Goodreads CSV export has these columns:

Column name	Type	Notes
Book Id	int, unique	goodreads internal id
Title	str	
Author	str	First Last
Author l-f	str	Last, First
Additional Authors	str	First Last1, First Last2…
ISBN	str	value is "=" if empty
ISBN13	str	value is "=" if empty
My Rating	int	0, 1, 2, 3, 4, 5
Average Rating	float	2 decimals
Publisher	str	can be empty
Binding	str	can be empty
Number of Pages	int	can be empty
Year Published	int	can be empty
Original Publication Year	int	can be empty
Date read	date	YYYY/MM/DD
Date added	date	YYYY/MM/DD
Bookshelves	str	comma separated
Bookshelves with positions	str	NAME (#NUM), comma sep
Exclusive Shelf	str	NAME
My Review	text	can be empty
Spoiler	text	can be empty
Private Notes	text	can be empty
Read count	int	
Owned copies	int	

__init__() → None

import_book(*row: Dict[str, Any]*, *overwrite: bool = False*) → *Book*

Get or create a `Book` based on `row`, a row from our `csv.DictReader` reader of our Goodreads export.

Parameters
row – a row from our Goodreads export

Keyword Arguments
overwrite – if `True`, overwrite any existing book data for this book

Returns
A `Book` instance

import_reading(*book: Book*, *user: User*, *row: Dict[str, Any]*) → None

Import the data for the `Reading` record for `user`.

Parameters

- **book** – the book for which we're importing reading data
- **user** – the user whose reading data we're importing
- **row** – the row from the Goodreads CSV, as output by `csv.DictReader`

load_lookups(*filename: str*) → None

Find the unique bindings, publishers and authors in the Goodreads export CSV `filename` and create them in the database as necessary.

Parameters
filename – the filename of the CSV file to read

run(*filename: str*, *user: User*, *overwrite: bool = False*) → None

Load the books in the CSV identified by `filename` into the database, splitting each row into appropriate *book_manager.models.Book*, *book_manager.models.Author*, *book_manager.models.Publisher* and *book_manager.models.Binding* records, creating the foreign keys and many-to-many targets as needed.

`bookmanager.models.Reading` data will always be overwritten, and `bookmanager.models.Book` data will be preserved, unless `override` is `True`.

Parameters
filename – the filename of the Goodreads CSV export file

Keyword Arguments
overwrite – if `True`, overwrite any existing `Book` with data from the CSV

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

F

G

I

L

M

N

O

P

R

S

T

Y