file:dict_of_sets_with_default.py

Link

> sqlalchemy/examples/association/dict_of_sets_with_default.py at 8503dc2e948908199cd8ba4e6b1d1ddcf92f4020 · sqlalchemy/sqlalchemy · GitHub

---

▼ Code

```
import operator

from sqlalchemy import Column
from sqlalchemy import create_engine
from sqlalchemy import ForeignKey
from sqlalchemy import Integer
from sqlalchemy import String
from sqlalchemy.ext.associationproxy import association_proxy
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import relationship
from sqlalchemy.orm import Session
from sqlalchemy.orm.collections import KeyFuncDict


class Base:
    id = Column(Integer, primary_key=True)


Base = declarative_base(cls=Base)


class GenDefaultCollection(KeyFuncDict):
    def __missing__(self, key):
        self[key] = b = B(key)
        return b


class A(Base):
    __tablename__ = "a"
    associations = relationship(
        "B",
        collection_class=lambda: GenDefaultCollection(
            operator.attrgetter("key")
        ),
    )

    collections = association_proxy("associations", "values")
    """Bridge the association from 'associations' over to the
    association proxy of B.
    """


class B(Base):
    __tablename__ = "b"
    a_id = Column(Integer, ForeignKey("a.id"), nullable=False
    elements = relationship("C", collection_class=set)
    key = Column(String)

    values = association_proxy("elements", "value")
    """Bridge the association from 'elements' over to the
    'value' element of C."""

    def __init__(self, key, values=None):
        self.key = key
        if values:
            self.values = values
```

```
class C(Base):
    __tablename__ = "c"
    b_id = Column(Integer, ForeignKey("b.id"), nullable=False
    value = Column(Integer)

    def __init__(self, value):
        self.value = value


if __name__ == "__main__":
    engine = create_engine("sqlite://", echo=True)
    Base.metadata.create_all(engine)
    session = Session(engine)

    # only "A" is referenced explicitly.  Using "collections"
    # we deal with a dict of key/sets of integers directly.

    session.add_all([A(collections={"1": {1, 2, 3}})])
    session.commit()

    a1 = session.query(A).first()
    print(a1.collections["1"])
    a1.collections["1"].add(4)
    session.commit()

    a1.collections["2"].update([7, 8, 9])
    session.commit()

    print(a1.collections["2"])
```

▼ Documentation

```
"""An advanced association proxy example which
illustrates nesting of association proxies to produce multi-l
collections, in this case a dictionary with string keys and s
```

```
as values, which conceal the underlying mapped classes.

This is a three table model which represents a parent table r
dictionary of string keys and sets as values, where each set 
collection of integers. The association proxy extension is us
details of this persistence. The dictionary also generates ne
upon access of a non-existent key, in the same manner as Pyth
"collections.defaultdict" object.

"""
```

Prompt

> As a code documentation assistant, you are responsible for documenting at the file/script level. When provided a file level code, your approach involves generating a header comment to be added at the top of the file. This comment should be the documentation for the code and include all relevant information needed to understand or use the script. The code is provided below.

Responses

ChatGPT

```
"""
File-level Documentation

This script defines three SQLAlchemy models: A, B, and C, and de
```

```
Classes:
- Base: The base class for all models, with a primary key 'id'.
- A: Represents a model 'a' with a relationship to model 'B'.
- B: Represents a model 'b' with a relationship to model 'C'.
- C: Represents a model 'c' with an 'Integer' value.

Dependencies:
- SQLAlchemy: The script depends on SQLAlchemy for defining and

Usage:
1. The script creates an SQLite database using 'create_engine' a
2. It defines three models 'A', 'B', and 'C', and establishes re
3. The script demonstrates how to add, query, and modify data in

Note: This script serves as an example and may need modification

Author: [Your Name]
Date: [Date]
"""
```

GPT4

```
"""
File: sqlalchemy_association_proxy_example.py

This script demonstrates the use of SQLAlchemy for ORM (Object-R
It showcases the creation and manipulation of database tables us
base class, association proxy, and relationship handling.

Key Features:
- Declarative Base: Defines the base class for declarative class
- Association Proxy: Utilizes the `association_proxy` feature fo
- Custom Collection Class: Implements `GenDefaultCollection` to
- Relationship Mapping: Establishes relationships between differ
- Session Handling: Demonstrates the creation of a session and

Models:
- A: Represents a table 'a' with a relationship to 'B' and uses
- B: Represents a table 'b' with a foreign key relationship to
- C: Represents a table 'c' with a foreign key relationship to

Usage:
The script is executed as a standalone program. It creates an i
adding and retrieving data using SQLAlchemy's session managemen
with SQLAlchemy and should be adapted for practical application

Dependencies:
- SQLAlchemy: A Python SQL toolkit and Object-Relational Mappin
"""
```

Bard

```
"""
This code demonstrates how to use association proxies to create

## Setting up the Database

The code starts by importing the necessary modules from SQLAlche

The `create_engine()` function is then used to create a new SQL

The `Base.metadata.create_all(engine)` method is called to crea

## Defining the Classes

The code defines three classes: `A`, `B`, and `C`. These classe

The `A` class has a many-to-many relationship with the `B` clas
The `B` class has a many-to-many relationship with the `C` clas

The `key` column in the `B` table is used to create a unique ke

## Working with the Classes

The code creates a new instance of the `A` class and adds it to

The code then queries the session for the `A` instance and prin

Finally, the code updates the set for the key `2` with the valu

## Association Proxies

Association proxies are a way to create a virtual relationship 

## Conclusion

This code demonstrates how to use association proxies to create
"""
```

LLama2

```
# File: sqla_example.py
# Description: Example usage of SQLAlchemy ORM and declarative 
# Author: Your Name

# Import necessary libraries
from sqlalchemy import (
    Column,
    create_engine,
    ForeignKey,
    Integer,
    String,
    association_proxy,
    declarative_base,
    relationship,
    Session,
    KeyFuncDict,
)

# Define base class with integer ID column and primary key
class Base:
    id = Column(Integer, primary_key=True)

# Create declarative base using Base class
Base = declarative_base(cls=Base)

# Define dictionary-like collection class with missing item han
class GenDefaultCollection(KeyFuncDict):
    def __missing__(self, key):
        self[key] = b = B(key)
        return b

# Define class A with tablename "a" and relationships to classe
class A(Base):
    __tablename__ = "a"
    associations = relationship(
        "B",
        collection_class=lambda: GenDefaultCollection(
            operator.attrgetter("key")
        ),
    )
    collections = association_proxy("associations", "values")
    """Bridge the association from 'associations' over to the '
    association proxy of B."""

# Define class B with tablename "b" and relationships to class
class B(Base):
    __tablename__ = "b"
    a_id = Column(Integer, ForeignKey("a.id"), nullable=False)
    elements = relationship("C", collection_class=set)
    key = Column(String)
    values = association_proxy("elements", "value")
    """Bridge the association from 'elements' over to the
    'value' element of C."""

    def __init__(self, key, values=None):
        self.key = key
        if values:
            self.values = values

# Define class C with tablename "c" and relationship to class B
class C(Base):
    __tablename__ = "c"
    b_id = Column(Integer, ForeignKey("b.id"), nullable=False)
    value = Column(Integer)

    def __init__(self, value):
        self.value = value

# If this file is run as main, create database connection and a
if __name__ == "__main__":
    engine = create_engine("sqlite://", echo=True)
    Base.metadata.create_all(engine)
    session = Session(engine)

    # Add data to table A using collections attribute
    session.add_all([A(collections={"1": {1, 2, 3}})])
    session.commit()

    # Retrieve data from table A and print associated collectio
    a1 = session.query(A).first()
    print(a1.collections["1"])

    # Modify associated collection for a1
    a1.collections["1"].add(4)
    session.commit()

    # Update associated collection for a1
    a1.collections["2"].update([7, 8, 9])
    session.commit()

    # Print updated associated collection for a1
    print(a1.collections["2"])
```

StarChat

```
"""
This module demonstrates how to implement custom collections in
It also shows off some other features like Association Proxies w
The specific implementation details may not apply to most projec
```