Report

keith Null

April 16, 2018

1 Problem Description

In exercise one, the task is to create a database and tables according to the given data. And then, use Python to store the data into the database.

In exercise two, the task is to design SQL enquiries to get the data we need from the database.

2 Problem Analysis

To create the database and tables, it's better to operate directly in the MySQL shell rather than using Python. So we just need to run codes like "CREATE DATABASE xxx" and "CREATE TABLE xxx" in the shell.

To store data, evidently we should use Python to connect to the database and execute SQL enquiries like "INSERT INTO".

As for designing SQL enquiries to get data from the database, "SELECT xxx FROM xxx" and "JOIN" are practical ways.

3 Experiment Process

3.1 Preparpations

Although only MySQL is needed in this experiment, for the sake of simplicity, we install XAMPP (Version 7.2.2) and then start MySQL in the panel to get everything prepared properly.

3.2 Creating the database and tables

In the shell of MySQL, enter codes as follows:

```
Mysql -u root
```

Create the database and one thing to note, to avoid encoding problems, we need to set the default character set as utf8:

```sql
CREATE DATABASE AcademicDB DEFAULT CHARSET utf8;
```

Then use the database we have created and create tables in it. When creating tables, we need to set the data type in accordance with the data and also use utf8 to encode.

```sql
USE AcademicDB;
CREATE  TABLE  papers  (PaperID  char(8),  Title
↪  text,PaperPublishYear integer(4), ConferenceID char(8))
↪  DEFAULT charset utf8;
CREATE  TABLE  authors  (AuthorID  char(8),  AuthorName
↪  tinytext) DEFAULT charset utf8;
CREATE  TABLE   conferences   (ConferenceID
↪  char(8),ConferenceName tinytext)DEFAULT charset utf8;
CREATE  TABLE   affiliations   (AffliationID
↪  char(8),AffliationName tinytext)DEFAULT charset utf8;
CREATE TABLE paper_author_affiliation (PaperID char(8),AuthorID
↪  char(8),  AffliationID  char(8),AuthorSequence tinyint
↪  unsigned)DEFAULT charset utf8;
CREATE TABLE   paper_reference   (PaperID   char(8),ReferenceID
↪  char(8))DEFAULT charset utf8;
```

3.3 Using Python to save the data in the database

To simplify and beautify the code, we import the module PyMySQL and define a class DataInserter with methods like connect() and insert_from_file().

First, after creating an instance of this class, we need to connect it the database by providing host, port, user and password, etc.

```python
def connect(self, user, password, db,host="localhost",
↪  port=3306, charset="utf8"):
    try:
        self.connection = pymysql.connect(
        host=host, user=user, password=password,
        db=db, port=port, charset=charset)
        self.cursor = self.connection.cursor()
    except:
        print("Failed to connect to the database!")
    else:
        self.connected = True
```

When insert_from_file() is called, it checks whether the connection has been established successfully and if not, just return directly.

In addition, to keep the user informed of the process progress, it opens the file twice to count the lines first and then handle it actually.

Note that as some data contain digital types, we need to convert some string into integer. For instance, what we need is 2004 rather than “2004”. To implement this and avoid unnecessary code, this method takes to_digit (a tuple) as a parameter to mark which columns need to be converted.

```python
def insert_from_file(self, table,file_name,to_digit=()):
    if not self.connected:
        print("Haven't connected to the database yet!")
        return
    all_line = 0
    with open(file_name, encoding="utf8", mode="r") as file:
        for line in file:
            all_line += 1
    print("{0} has {1} line{2} to handle.".format(
       file_name, all_line, "s" if all_line >= 2 else ""))
    with open(file_name, encoding="utf8", mode="r") as file:
        current_line = 1
        for line in file:
            print("{0}/{1}".format(current_line, all_line), end="\r")
            data = line.strip().split("\t")
            for i in to_digit:
                data[i] = int(data[i])
            sql="INSERT INTO {0} VALUES {1};".format(table,tuple(data))
            try:
                self.cursor.execute(sql)
            except:
                print("Failed to insert data {0} into table {1}(Line {2} of {3})".format(data,table,current_line , file_name))
                self.connection.rollback()
                break
            current_line += 1
        self.connection.commit()
    print("Done successfully!")
```

For example, when saving papers.txt into papers table, we just need to call the method in this way:

```python
inserter.insert_from_file(table="papers", file_name="data/papers.txt", to_digit=(2,))
```

3.4 Designing SQL enquiries

Without too much difficulty, the SQL enquires satisfying the requirements are as follows:

```sql
SELECT title,paperpublishyear
FROM papers
WHERE paperid="58EA85EE";

SELECT authorid
FROM paper_author_affiliation
WHERE paperid="58EA85EE"
ORDER BY AuthorSequence ASC;

SELECT authors.authorname
FROM authors
INNER JOIN paper_author_affiliation ON authors.authorid
=paper_author_affiliation.authorid
WHERE paper_author_affiliation.paperid="58EA85EE"
ORDER BY paper_author_affiliation.AuthorSequence ASC;

SELECT count(*)
FROM paper_reference
WHERE referenceid="800F1DB6";
```

3.5 Optimizing the enquiry time with index

Running the above enquiries might not take a very long time, even the slowest one of them (to be exact, the third one) only takes a few seconds. But to pursue the best, we can speed the enquiries up using index.

Take the third enquiry which involves "JOIN" as example, without any optimization, it takes 2.54s. If we simply create an index in the table paper_author_affiliation, the same enquiry only takes 0.24s, faster more than 10 times.

```sql
CREATE INDEX index_paperid ON paper_author_affiliation(paperid);
```

So obviously, creating index properly can definitely optimize the enquiry time.

4 Conclusion and reflection

If the data involve non-ascii characters, remember to set the character set as utf8.

Use class and module to write Python code effectively and beautifully.

If some SQL enquiries are too slow, try to create index properly.