Module 4

PPT2

Metadata

- Metadata is data that describes other data. Meta is a prefix that -- in most information technology usages -- means "an underlying definition or description." Metadata summarizes basic information about data, which can make finding and working with particular instances of data easier.
- For example, *author*, *date created*, *date modified* and *file size* are examples of very basic document metadata. Having the ability to filter through that metadata makes it much easier for someone to locate a specific document.
- In addition to document files, metadata is used for:
- Images
- Videos
- Spreadsheets
- Web pages

- Meta tags are often evaluated by search engines to help decide a web page's relevance, and were used as the key factor in determining position in a search until the late 1990s. The increase in search engine optimization (SEO) towards the end of the 1990s led to many websites "keyword stuffing" their metadata to trick search engines, making their websites seem more relevant than others. Since then search engines have reduced their reliance on metatags, though they are still factored in when indexing pages. Many search engines also try to halt web pages' ability to thwart their system by regularly changing their criteria for rankings, with Google being notorious for frequently changing their highly-undisclosed ranking algorithms.
- Metadata can be created manually, or by automated information processing. Manual creation tends to be more accurate, allowing the user to input any information they feel is relevant or needed to help describe the file.
- Automated metadata creation can be much more elementary, usually only displaying information such as file size, file extension, when the file was created and who created the file.

- importlib.metadata is a library that provides for access to installed package metadata.
- Through this module, you can access information about installed packages in your Python installation.
- Together with its companion module, importlib.resources, importlib.metadata improves on the functionality of the older pkg_resources.
- What is pip? pip is the standard package manager for Python. It allows you to install and manage additional packages that are not part of the Python standard library.

```
from importlib import metadata
print(metadata.version("pip"))
pip_metadata = metadata.metadata("pip")
print( list(pip_metadata))
```

DataFS Data Management System

- DataFS is a package manager for data. It manages file versions, dependencies, and metadata for individual use or large organizations.
- Configure and connect to a metadata Manager and multiple data Services using a specification file and you'll be sharing, tracking, and using your data in seconds.
- Features
  - Explicit version and metadata management for teams
  - Unified read/write interface across file systems
  - Easily create out-of-the-box configuration files for users
  - Track data dependencies and usage logs
  - Use datafs from python or from the command line
  - Permissions handled by managers & services, giving you control over user access

Database Programming

- From a construction firm to a stock exchange, every organisation depends on large databases. These are essentially collections of tables, and' connected with each other through columns.
- These database systems support SQL, the Structured Query Language, which is used to create, access and manipulate the data. SQL is used to access data, and also to create and exploit the relationships between the stored data. Additionally, these databases support database normalisation rules for avoiding redundancy of data.
- The Python programming language has powerful features for database programming. Python supports various databases like MySQL, Oracle, Sybase, PostgreSQL, etc.
- Python also supports Data Definition Language (DDL), Data Manipulation Language (DML) and Data Query Statements.
- For database programming, the Python DB API is a widely used module that provides a database application programming interface.

Benefits of Python for database programming

- Programming in Python is arguably more efficient and faster compared to other languages.
- Python is famous for its portability.
- It is platform independent.
- Python supports SQL cursors.
- In many programming languages, the application developer needs to take care of the open and closed connections of the database, to avoid further exceptions and errors. In Python, these connections are taken care of.
- Python supports relational database systems.
- Python database APIs are compatible with various databases, so it is very easy to migrate and port database application interfaces.

You can choose the right database for your application. Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

DB-API (SQL-API) for Python

- Python DB-API is independent of any database engine, which enables you to write Python scripts to access any database engine.
- The Python DB API implementation for MySQL is MySQLdb. For PostgreSQL, it supports psycopg, PyGresQL and pyPgSQL modules.
- DB-API implementations for Oracle are dc_oracle2 and cx_oracle. Pydb2 is the DB-API implementation for DB2.
- Python's DB-API consists of connection objects, cursor objects, standard exceptions and some other module contents

- You must download a separate DB API module for each database you need to access. For example, if you need to access an Oracle database as well as a MySQL database, you must download both the Oracle and the MySQL database modules.
- The DB API provides a minimal standard for working with databases using Python structures and syntax wherever possible. This API includes the following −
  - Importing the API module.
  - Acquiring a connection with the database.
  - Issuing SQL statements and stored procedures.
  - Closing the connection

Connection objects

- Connection objects create a connection with the database and these are further used for different transactions. These connection objects are also used as representatives of the database session.
- A connection is created as follows:

```
>>>conn = MySQLdb.connect('library', user='suhas',
password='python')
```

- You can use a connection object for calling methods like commit(), rollback() and close() as shown below:

```
>>>cur = conn.cursor()  //creates new cursor object for executing SQL
statements
>>>conn.commit()  //Commits the transactions
>>>conn.rollback()  //Roll back the transactions
>>>conn.close()  //closes the connection
>>>conn.callproc(proc,param)  //call stored procedure for execution
>>>conn.getsource(proc)  //fetches stored procedure code
```

- What is MySQLdb?
- MySQLdb is an interface for connecting to a MySQL database server from Python.
- It implements the Python Database API v2.0 and is built on top of the MySQL C API.

Download and install "MySQL Connector":

```
C:\Users\Your Name\AppData\Local\Programs\Python\Python36-32\Scripts>python -m pip install mysql-connector-python
```

To test if the installation was successful, or if you already have "MySQL Connector" installed, create a Python page with the following content:

```
import mysql.connector
```

Create Connection

- Start by creating a connection to the database.
- Use the username and password from your MySQL database:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
print(mydb)
```

Assume −

- That we have created a database with name mydb.
- We have created a table EMPLOYEE with columns FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.
- The credentials we are using to connect with MySQL are username: ***root***, password: ***password***.
- You can establish a connection using the ***connect()*** constructor. This accepts username, password, host and, name of the database you need to connect with (optional) and, returns an object of the MySQLConnection class.

```
import mysql.connector #establishing the connection
conn = mysql.connector.connect(user='root',
password='password', host='127.0.0.1',
database='mydb')
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
#Executing an MYSQL function using the execute()
method
cursor.execute("SELECT DATABASE()")
# Fetch a single row using fetchone() method.
data = cursor.fetchone()
print("Connection established to: ",data)
#Closing the connection
conn.close()
```

- You can also establish connection to MySQL by passing credentials (user name, password, hostname, and database name) to ***connection.MySQLConnection()*** as shown below –

```
from mysql.connector import (connection)
#establishing the connection
conn =
connection.MySQLConnection(user='root',
password='password', host='127.0.0.1',
database='mydb')
#Closing the connection
conn.close()
```

Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

```
import mysql.connector
 mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE mydatabase")
```

Check if Database Exists

- You can check if a database exist by listing all databases in your system by using the "SHOW DATABASES" statement:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW DATABASES")
for x in mycursor:
  print(x)
```

```
import mysql.connector
#establishing the connection
mydb = mysql.connector.connect(  host="localhost",
  user="yourusername",  password="yourpassword")
#Creating a cursor object using the cursor() method
cursor = conn.cursor()
#Doping database MYDATABASE if already exists.
cursor.execute("DROP database IF EXISTS MyDatabase")
#Preparing query to create a database
sql = "CREATE database MYDATABASE";
#Creating a database
cursor.execute(sql)
#Retrieving the list of databases
print("List of databases: ")
cursor.execute("SHOW DATABASES")
print(cursor.fetchall())
#Closing the connection
conn.close()
```

Creating a Table

- To create a table in MySQL, use the "CREATE TABLE" statement.
- Make sure you define the name of the database when you create the connection
- Create a table named "customers":

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers (name
VARCHAR(255), address VARCHAR(255))")
```

Check if Table Exists

- You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="myusername",
  password="mypassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW TABLES")
for x in mycursor:
    print(x)
```

Insert Into Table

```
import mysql.connector
mydb = mysql.connector.connect(  host="localhost",
  user="myusername",   password="mypassword",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record inserted.")
```

Important!: Notice the statement: mydb.commit(). It is required to make the changes, otherwise no changes are made to the table.

Insert Multiple Rows

- To insert multiple rows into a table, use the executemany() method.
- The second parameter of the executemany() method is a list of tuples, containing the data you want to insert:

```
import mysql.connector
mydb = mysql.connector.connect(  host="localhost",   user="myusername",
password="mypassword",
  database="mydatabase")
mycursor = mydb.cursor()
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = [
 ('Peter', 'Lowstreet 4'),
 ('Amy', 'Apple st 652'),
 ('Hannah', 'Mountain 21'),
 ('Michael', 'Valley 345'),
 ('Sandy', 'Ocean blvd 2'),
 ('Betty', 'Green Grass 1'),
 ('Richard', 'Sky st 331'),
 ('Susan', 'One way 98'),
]
mycursor.executemany(sql, val)
mydb.commit()
print(mycursor.rowcount, "record was inserted.")
```

Select From a Table

- To select from a table in MySQL, use the "SELECT" statement:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="myusername",
  password="mypassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

Note: We use the fetchall() method, which fetches all rows from the last executed statement.

Selecting Columns

- To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="myusername",
  password="mypassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT name, address FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

Using the fetchone() Method

- If you are only interested in one row, you can use the fetchone() method.
- The fetchone() method will return the first row of the result:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="myusername",
  password="mypassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchone()
print(myresult)
```

Where

- WHERE is used to select data on some condition.
- SELECT column_name FROM table_name WHERE condition statement will be used to retrieve the data on some condition.

```
import mysql.connector
mydb = mysql.connector.connect(host="localhost", user="root",  password="salu@pari925",
  database="priyanka"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM students4 WHERE name = 'asd'")
print(mycursor.fetchall())
mydb.close()
```

Order By

- Use the ORDER BY to sort the result in ascending or descending order. It sorts the result in ascending order by default, to sort the result in descending order use the keyword DESC.
- SELECT column_names FROM table_name ORDER BY column_name statement will be used to sort the result in ascending order by a column.
- SELECT column_names FROM table_name ORDER BY column_name DESC statement will be used to sort the result in descending order by a column.

```
import mysql.connector
mydb = mysql.connector.connect(host="localhost",
user="root",  password="salu@pari925",
  database="priyanka"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM students4
ORDER BY name")
print(mycursor.fetchall())
mydb.close()
```

DESC

```
import mysql.connector
mydb = mysql.connector.connect(host="localhost", user="root",  password="salu@pari925",
  database="priyanka"
)
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM students4 ORDER BY name DESC")
print(mycursor.fetchall())
mydb.close()
```

Delete

- DELETE keyword is used to delete the records from the table.
- DELETE FROM table_name WHERE condition statement is used to delete records. If you don't specify the condition, then all of the records will be deleted.

```
import mysql.connector
mydb = mysql.connector.connect(host="localhost",  user="root",
password="salu@pari925",
  database="priyanka")
mycursor = mydb.cursor()
mycursor.execute("DELETE FROM students4 WHERE name = 'asd'")
mycursor.execute("SELECT * FROM students4")
print(mycursor.fetchall())
mydb.close()
```

Update

- UPDATE keyword is used to update the data of a record or records.
- UPDATE table_name SET column_name = new_value WHERE condition statement is used to update the value of a specific row.

```
import mysql.connector
mydb = mysql.connector.connect(host="localhost",  user="root", password="salu@pari925",
  database="priyanka")
mycursor = mydb.cursor()
mycursor.execute("UPDATE students4 SET name = 'Kareem' WHERE year =833")
mycursor.execute("SELECT * FROM students4")
print(mycursor.fetchall())
mydb.close()
```

Postgre SQL db

- https://www.enterprisedb.com/downloads/postgres-postgresql-downloads
- In Python, we have serval modules available to connect and work with PostgreSQL. the following are the list.
  - Psycopg2
  - pg8000
  - py-postgresql
  - PyGreSQL
  - ocpgdb
  - bpgsql
  - SQLAlchemy. SQLAlchemy needs any of the above to be installed separately.

- Psycopg2 is the most popular python driver for PostgreSQL.
- It is required for most Python and Postgres frameworks.
- Actively maintained and support the major version of python i.e. Python 3 and Python 2.
- It is thread-safe (threads can share the connections). It was designed for heavily multi-threaded applications.
- Using pip command, you can install Psycopg2 on any operating system including Windows, macOS, Linux, and Unix and Ubuntu. Use the following pip command to install Psycopg2.
- pip install psycopg2

Steps to connect PostgreSQL through python

- Use the connect() method of psycopg2 with required arguments to connect PostgreSQL.
- Create a cursor object using the connection object returned by the connect method to execute PostgreSQL queries from Python.
- Close the Cursor object and PostgreSQL database connection after your work completes.
- Catch Exception if any that may occur during this process.

```
import psycopg2

conn = psycopg2.connect(user = "postgres", password =
"salu@pari925", host = "127.0.0.1", port = "5432")

print("Opened database successfully")

conn.close()
```

Table creation

```
import psycopg2

conn = psycopg2.connect(database="postgres", user = "postgres", password = "salu@pari925", host = "127.0.0.1", port = "5432")

print("Opened database successfully")

c = conn.cursor()

c.execute('''CREATE TABLE COMPANY(ID INT, NAME TEXT, AGE INT NOT NULL, ADDRESS CHAR(50), SALARY REAL);''')

print("Table created successfully")

conn.commit()

conn.close()
```

Insert into table

```
import psycopg2

conn = psycopg2.connect(database="postgres", user = "postgres", password =
"salu@pari925", host = "127.0.0.1", port = "5432")

print("Opened database successfully")

c = conn.cursor()

c.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES
(1, 'Paul', 32, 'California', 20000.00 )");

c.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES
(2, 'Allen', 25, 'Texas', 15000.00 )");

c.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES
(3, 'Teddy', 23, 'Norway', 20000.00 )");

c.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) VALUES
(4, 'Mark', 25, 'Rich-Mond ', 65000.00 )");

conn.commit()

print("Records created successfully")

conn.close()
```