CIS 41B
Advanced Python Programming

Data Storage

De Anza College
Instructor: Clare Nguyen

Data Persistence

- Data persistence is the ability to store data to secondary storage when the application ends, and then read it back in at a later time when the application starts up again.
- A computer in hibernation mode, where all its state is saved on disk, so that it can wake up in its last state, is an example of persistence.
- Data storage can be a simple text file or it can be a database. And the storage can be on hard disk within one computer or across the network on a different server.
- So far we've discussed 2 ways to store data such that it can be conveniently read in at a later time.
  - Text file: If the data is mostly text or is made of one or more simple sequences of numeric values, then a text file is the simplest format.
  - CSV file: If the data is made up of multiple data records (such as a student record of name, id, gpa, courses taken), then using a CSV file is a simple way to store data such that many applications, including non-Python ones, can easily read in the data.
- Next we see 3 new ways to save application data: pickle, JSON, and database files.

Pickle to File

- Pickle is Python's way of serializing a Python object into binary bytes.
- A Python object is essentially a data structure that can contain other Python objects and nesting of other Python objects.
- To pickle an object is to 'flatten' it into a stream of binary bytes, typically to save it to a file. Then the file can be read in at another time or by another Python application.
- To serialize an object we use the pickle module:

```
import  pickle
```

- To pickle an object and save it to a file, we use the dump method:

```
pickle.dump( obj_name, open(filename, "wb") )
```

  The "wb" mode is to write binary data. The file that is produced is a binary file that contains data and information to reconstruct the obj_name object.

- To reconstruct a pickled object, we use the load method:

```
pickle.load( open(filename, "rb") )
```

  The "rb" mode is to read binary data. The binary data in the file is converted into the original object obj_name.

Pickle to an Object

- We can pickle and reconstruct an object without saving it to file:

```
b = pickle.dumps( obj_name)
```

```
newObj = pickle.loads( b )
```

  where b is a binary string that pickle produces.

- Some common uses of pickle:
  - Save a program's state (or current data) to disk so that it can continue from where it ended last time.
  - Send data to another Python process in a multi-core or distributed system (covered in the next module).
  - Easily save a Python object (with its data hierarchy) and restore the object.
  - Convert a Python object to a string so that it can be used as a dictionary key, or for caching and memoization.
  - A pickled file or object can only be restored by a Python program.
- What is picklable?
  - Atomic data types: bool, int, float, None.
  - Container data types: string, byte string, list, tuple, dictionary, set, all of which can contain any combination of picklable data types.
  - Functions and classes that are defined at the top module, and objects with components that are picklable.

JSON File

- In the previous module we've seen that it's possible to store large data objects with multiple nested structures in a JSON (Javascript Object Notation) file.
- A JSON file is easily converted into and back from a dictionary with nested structures.
- A JSON file:
  - Keeps the nested data structures intact
  - Can be read in by non-Python applications
  - Can be read by humans because the data is encoded in 'utf-8' format
  - Is considered safer to use than a pickle file because no functions (no code) can be saved in a JSON file.
- Disadvantage of JSON files: The module that handles JSON only works with certain Python data types. If we want to save an object of our own 'custom' class in JSON format, we will need to write code that turns the object into a dictionary first.
- Python data types that can be converted to JSON:

dict	int
list	float
tuple	bool
str	None

JSON Module

- To work with JSON we bring in the json module:
```
import  json
```
- To create a JSON string from a Python object:
```
jason_obj = json.dumps(python_obj)
```
- To create a Python object from a JSON string:
```
python_obj = json.loads(jason_obj)
```
- To create a JSON file format from Python data:
```
with open('data.json', 'w') as fh:
    json.dump(data, fh) )
```
To create a file with proper indentation for human readability:
```
with open('data.json', ‘w') as fh:
    json.dump(data, fh, indent=3)
```
- To read in a JSON file:
```
with open('data.json', 'r') as fh:
    data = json.load(fh)
```

Why Use a Database?

- When dealing with large scale data, it is necessary to use a database to store the data in a systematic way so we can efficiently retrieve the data.
- Unlike a data structure, a database can provide different ways to retrieve data so that we can access a portion of the data without having to modify how the data is store.
- For example, initially we store a set of data in a dictionary as key / value pairs because our analysis requires that we search for certain *keys*. However, if later analyses require that we search for certain *values*, then our initial dictionary would no longer be the most efficient way to store data.
- With a database, we can search for data in multiple ways. As long as we have a good idea of the type of data analysis we want to do, we can set up the database for flexible and fast data retrieval.
- Database concepts first appear in the 1960's when it was becoming possible to have relatively fast random data access. The database field grew and today we have optimized database systems that rely on serious math computations to store and organize data.

Types of Database

- The simplest “database” is a text file.
- The next level of storing data is a CSV file, in which data are organized in table format. Each row represents one data record, and the columns in a row are attributes of the data record.
- The next higher level of storing data are SQL and noSQL databases. SQL databases have existed longer than noSQL databases.
- noSQL databases are designed around key-values, graphs, or documents (or groupings of data). Some big-named noSQL databases are MongoDB and Cassandra.
- SQL databases are designed around tables that are associated with each other. Some big-named SQL databases are Oracle database and MySQL.
- Python has a built-in general purpose SQL database called SQLite. This database is packaged with the Python core, and it’s the one we’ll use.

SQL Database

- SQL (Structured Query Language) is a standard language that is used to query a relational database. This is why relational databases are called SQL databases.
- A relational database models the data by storing data in one or more tables of rows and columns, and then linking these tables to each other.
- Because the tables in a relational database have relationships or links to each other, it allows the data to be cross-referenced across the tables, and this allows for very flexible data retrieval.

Relational Database Terminology (1)

- The following list uses programming terminology to briefly explain database terminology:
  - A relational database contains one or more tables that are related to each other.
  - A table is made up of *tuples* and *attributes*.
  - A *tuple* is a *record* or a row in a table, it represents a data object such as a student or a bank account or a music CD.
  - An *attribute* is a column or field in a table.
- Example of a table with 4 tuples, each representing a student record:

2031	Franklin	Aretha
3902	Torvalds	Linus
2704	Nightingale	Florence
1923	Van Rossum	Guido

Each tuple has 3 attributes: student ID, last name, first name

Relational Database Terminology (2)

- The data records in the tables are linked through a key, which is an ID that uniquely identifies a row in a table.
- Each table has a primary key column. Any table that needs to link to that table will have a foreign key column whose value will match the first table's primary key.
- Example:

Student Table

2031	Franklin	Aretha	3
3902	Torvalds	Linus	2
2704	Nightingale	Florence	1
1923	Van Rossum	Guido	2

Primary key: Student ID

Foreign key: major

Major Table

1	Nursing
2	Computer Science
3	Music

Primary key: major

The foreign key "major" is the link from the Student table to the Major table.

RDBMS

- There are many **r**elational **d**ata**b**ase **m**anagement **s**ystems (RDBMS), which are the software tools that helps us create, maintain, and query data in a relational database.
- Some of the popular RDBMS that are used in enterprise applications are from Microsoft, Oracle, SAP, etc.
- These RDBMS are for large to very large scaled data, and they typically reside in their own servers. Any query from our code to these database systems will be go through the network that connects us to the servers.
- The database system that we use in this class is SQLite, which is a small scale RDBMS that's built into Python, and the entire database that we create is stored as a single .db file.
- SQLite can be accessed in 2 ways: through a browser and through Python code.
- The SQLite browser allows us to view and manipulate the database file directly through a GUI interface. It can be downloaded from: http://sqlitebrowser.org/
- For this class we will access SQLite through Python code, but the GUI tool can be useful for debugging.

SQL

- Our Python code accesses the database through SQL commands, so we first need to have an overview of basic SQL commands
- SQL commands are used to:
  - Create a table
  - Delete a table
  - Insert data
  - Modify data
  - Query data
  - Delete data

SQL command: CREATE TABLE (1)

- Format to create a table:

```
CREATE TABLE TableName (
   column1_name    data_type    optional_restraint,
      . . .
   columnN_name   data_type    optional_restraint
);
```

- SQL data type:

INTEGER	TEXT	REAL	NULL
int	str	float	None

Python data type:

- Optional restraint:

PRIMARY KEY	NOT NULL	UNIQUE
Used as primary key	Can't be null	Has to be unique

Meaning:

- Example:

```
CREATE TABLE StudentsDB (
   id   INTEGER NOT NULL PRIMARY KEY,
   firstName  TEXT,
   lastName   TEXT
);
```

SQL command: CREATE TABLE (2)

- The restraint PRIMARY KEY or UNIQUE at an attribute means that the attribute has to be unique in the table. If another data record has the same attribute, then an exception will be raised.
- The attribute that has PRIMARY KEY restraint doesn't have to be filled in if we don't have an ID field for the data record. The database will automatically fill in the attribute with counting numbers starting at 1 and counting up.
- Sometime there are attributes that we want to be UNIQUE in the table, but we also know that different data records can have the same attribute. In this case, we add the restraint: ON CONFLICT IGNORE
- Example:

```
CREATE TABLE Majors(
   id   INTEGER NOT NULL PRIMARY KEY,
   major  TEXT UNIQUE ON CONFLICT IGNORE
);
```

  - The values for the id attribute will be unique in the table, and a duplicate id being inserted will cause an exception.
  - The values for the major attribute will also be unique in the table. And if there is a duplicate major being inserted into the table, then it will be ignored and not stored in the table.

SQL command: CREATE TABLE (3)

- If the table has many columns of the same data type, it's cumbersome to list all the columns in the CREATE TABLE command.

```
CREATE TABLE TableName (
   column1_name    data_type,
     . . .
   columnN_name   data_type);     # where N is a large number
```

- Instead we can create the table and use a loop to ALTER the table and ADD COLUMN.
- Example: Given that the table TableName above has been created and we need to add column1_name to columnN_name to the table:

```
for i in range(N) :
   cur.execute('''ALTER  TABLE  TableName  ADD COLUMN {}   data_type'''   \
                                        .format('column'  +  str(i) ))
   # or, using f-string:
   cur.execute(f'''ALTER  TABLE  TableName
                    ADD COLUMN {'column'  +  str(i)}   data_type''')
```

SQL command: DROP TABLE

- Format to delete a table:

```
DROP TABLE   IF EXISTS   TableName;
```

- The clause IF EXISTS is useful so that the DROP command only runs if there is a table with a matching name. Otherwise, if the DROP command runs and the table doesn't exist, there will be an exception.
- If the table to be dropped is linked to another table by a FOREIGN KEY constraint, then there will be an exception and the table will not be deleted.

SQL command: INSERT INTO

- To insert one or more rows of data into a table:

```
INSERT INTO  TableName
(column1_name, column2_name)
VALUES
(value1, value2);
```

Insert 1 row

```
INSERT INTO  TableName
(column1_name, column2_name)
VALUES
(value01, value02),
(value11, value22);
```

Insert multiple rows

- The list of column names is optional.
  - If not used: the list of values must have a value for every column in the table.
  - If used: the number of column names and the number of values must be the same.
- Example: Both of these commands insert 3 values into the Students table.

```
INSERT INTO  StudentsDB
(id, firstName, lastName)
VALUES
(2031, “Franklin”, “Aretha”);
```

```
INSERT INTO  StudentsDB
VALUES
(2031, “Franklin”, “Aretha”);
```

SQL command: UPDATE

- Format to modify a row of data of a table:

```
UPDATE   TableName
SET column1_name = new_value1,
     column2_name = new_value2
WHERE
     search_condition;
```

- The SET clause sets the new value to each column that matches the column name
- The WHERE clause is optional. If it is used, then only the rows where the search_condition is true will be modified. If it is not used, then all rows will be modified.
- The search_condition of the WHERE clause is discussed in a later slide.
- Example:

```
UPDATE  StudentsDB
SET  firstName = “ARETHA”
WHERE id = 2031;
```

SQL command: SELECT (1)

- Format to fetch particular columns in a table:

```
SELECT  column1_name, column2_name
FROM  TableName;
```

Will return all of column 1 and column 2 in the table

- Format to fetch all columns in a table:

```
SELECT  *
FROM  TableName;
```

Will return all columns or the entire table

- Format to fetch columns in a table and sort data:

```
SELECT column1_name, column2_name
FROM TableName
ORDER BY column2_name  ASC,
                column1_name DESC;
```

Will return columns 1 and 2, with column 1 sorted by descending order, and then column 2 sorted by ascending order.

If no ASC or DESC, the default is ascending sort.

SQL command: SELECT (2)

- Format to limit the number of rows that are returned:

```
SELECT  column1_name
FROM  TableName
LIMIT num;
```

Will return the first num entries of column 1

The LIMIT clause is often used with the ORDER BY clause to get the top values or bottom values of a column.

- Format to fetch rows based on a condition

```
SELECT  column2_name
FROM  TableName
WHERE  search_condition;
```

Will return column 2 of the rows where the search_condition is true.

SQL command: DELETE

- Format to delete rows in a table:

```
DELETE  FROM  TableName
WHERE  search_condition;
```

- Example:

```
DELETE   FROM  StudentsDB   WHERE  id = 1009;
```

SQL command: WHERE clause (1)

- The search_condition of the WHERE clause can be a numeric compare:

=	Equal
<> or !=	Not equal
<	Less than
>	Greater than
<=	Less than equal to
>=	Greater than equal to

- Example:

```
SELECT lastname FROM StudentsDB WHERE id = 2031;
```

SQL command: WHERE clause (2)

- The search_condition of the WHERE clause can be a logical compare:

Operator	**Explanation**	**Example**
AND	True only if both expressions are true	WHERE cost > 100 AND count < 2
BETWEEN	True if the value is within a range	WHERE major BETWEEN 1 and 5 WHERE lastname BETWEEN ‘R’ and ‘Z’
IN, NOT IN	True if value is in / not in a list of values	WHERE major IN (2, 3) WHERE major NOT IN (2, 3)
LIKE	True if value matches a pattern	WHERE name LIKE ‘%Ross%’ WHERE name LIKE ‘Frankl_n’ % is wildcard for 0 or more characters _ is wildcard for 1 character
NOT	Reverses the boolean value	WHERE name NOT BETWEEN ‘R’ and ‘Z’
OR	True if either expression is true	WHERE major = 2 OR major = 3

Connect to a Database

- The Python SQLite module is sqlite3: `import sqlite3`
- Connect to the database: `conn = sqlite3.connect('sample.db')`

  The connect method returns an object (conn) which is the handle to the database. The database is a .db file and resides in the current directory.
- If the .db file exists, then the handle connects to the existing database. If the .db file doesn't exist, then the handle connects to a new database.
- Set up for SQL commands: `cur = conn.cursor()`

  The cursor method returns an object (cur) from which we can call SQL commands.
- It is possible to use try except to catch the sqlite3.DatabaseError exception from the connect method. However, a connect exception is rare and only happens if there is not enough memory for the .db file.

Run SQL Commands

- Use the cursor object's execute method to run SQL commands:

```
cur.execute("one line SQL commands")
```

```
cur.execute('''multi-line
                SQL commands''')
```

- Note that the SQL command(s) are string input arguments for the execute method.
  Use single or double quotes for a one-line command.
  Use triple quotes (single or double) for multi-line commands.

- When all SQL commands have run, commit the changes to the database by using the connect object (not the cursor object):

```
conn.commit()
```

  Without a commit, no change will take effect in the database.

- When done with the database, close the connection:

```
conn.close()
```

Prepare the SQL Commands

- An SQL command is useful in a program only if we can pass data to it. For example:

  cur.execute("SELECT name FROM Students WHERE id = 1234")

  is not very useful because the id value is always a literal 1234.
- To pass a variable to the SQL command:
  - Use ? as a place holder for the variable in the SQL command
  - Use a tuple of variable names as the 2nd argument to execute
- Example

```
cur.execute("SELECT name  FROM StudentsDB  WHERE id = ?", (stuID,) )
```

```
cur.execute("'SELECT name  FROM StudentsDB
                WHERE gpa > ?  AND  major = ?'",  (minGPA, inputMajor) )
```

- The number of ? in the SQL command and the length of the tuple must be the same. Each ? is associated with an element of the tuple, in order from left to right.

Retrieve the Query Result

- After a SELECT, we need to retrieve the query result.
- Each data record retrieved is a tuple, and individual data fields can be indexed from the tuple.
- If there is only one result, use fetchone():

```
cur.execute("SELECT name  FROM StudentsDB  WHERE id = ?",  (stuID,) )
result = cur.fetchone()       # result is a tuple containing the name
```

- If there are multiple results, use fetchall():

```
cur.execute(''SELECT name  FROM StudentsDB  WHERE gpa > ?'',  (minGPA,))
results = cur.fetchall()        # results is a list of tuples
```

- If there are multiple results, use the query as a generator:

```
for record in cur.execute(''SELECT name  FROM StudentsDB'') :
    print(record[0])          # record is a tuple of 1 element: the name
```

Data Modeling

- A database is often made up of multiple tables that are linked or related together.
- The tables and their relationships are used to model data and they become the design for the database.
- As the number of data records becomes large, a good data model leads to good database design, which leads to fast queries.
- Some rules for modeling data:
  - Each ‘real world’ object should appear in the database only once.
  - If the object belongs in 2 tables, then show it as a relationship between the 2 tables.
  - For each attribute of an object, determine if it’s specific only to the object or if it’s shared among other objects.
  - If an attribute is specific to the object, then it goes into the same table as the object.
  - An attribute that is shared among other objects goes into a different table than the object.

Data Modeling Example

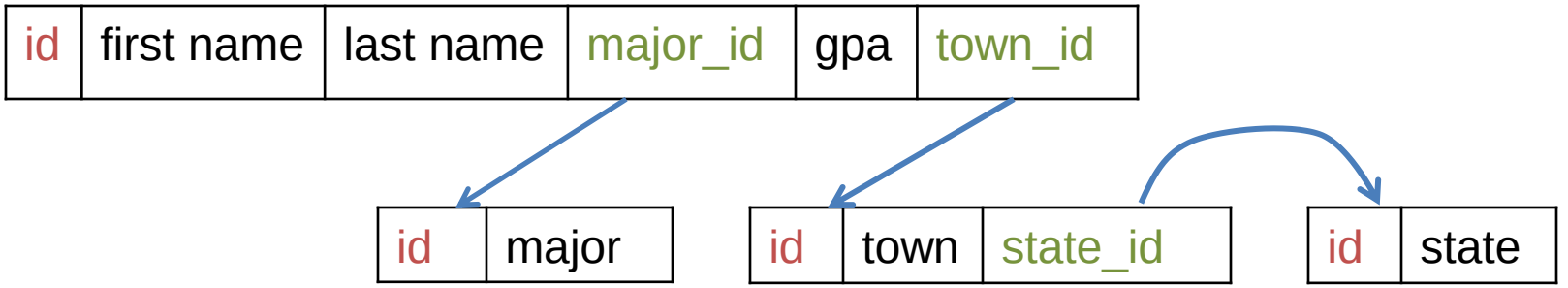
- Suppose we need to model student records for a college. Here is a sample of student records:

123	Grace	Hopper	CS	3.78	New York	NY
947	Michio	Kaku	Physics	3.82	San Jose	CA
235	Guido	VanRossum	CS	2.72	San Francisco	CA
439	Jennifer	Lopez	Music	3.28	Los Angeles	CA
673	Jerry	Rice	Business	2.45	San Francisco	CA
822	Andrea	Jung	Business	3.67	New York	NY

- The attributes that belong to a specific student are: student id, first name, last name, GPA.
  The attributes that can be shared among students are: major, hometown, and state.
- Based on the observations above, the data can be modeled using 4 tables: Student, Major, Hometown, State.
- The Student table is the main table and has links to the Major and Hometown tables. The Hometown table has a link to the State table.

Represent Data Models with Tables

- Each table contains rows of unique data that belong to the table.
  For example: in the Student table are rows of student records.
- Each table contains multiple columns: a required primary key column and other columns for the attributes of each data record.
- If an attribute column is a link to another table, then that column is the foreign key column.
- Example: One row of the 4 tables representing student data records



- The required primary keys in each table are in the red id field
- The foreign keys in each table are in the green id field, they show the link between the tables. Each foreign key is the start of the link (in blue), and the end of the link is a primary key of another table.

Example: Creating the Tables

- When the data model is done, use SQL to create the tables in the model.
- It is a good idea to create the table from the “outside in” order. This means create all tables that don’t have foreign keys first, then work our way to the tables that have foreign keys.
- Example of the table creation order for the student records database:
  1. States table with: state id (primary key), state name
  2. Towns table with: town id (primary key), town name, state id (foreign key)
  3. Majors table with: major id (primary key), major name
  4. Students table with: student id (primary key), student name, major id (foreign key), gpa, town id (foreign key)

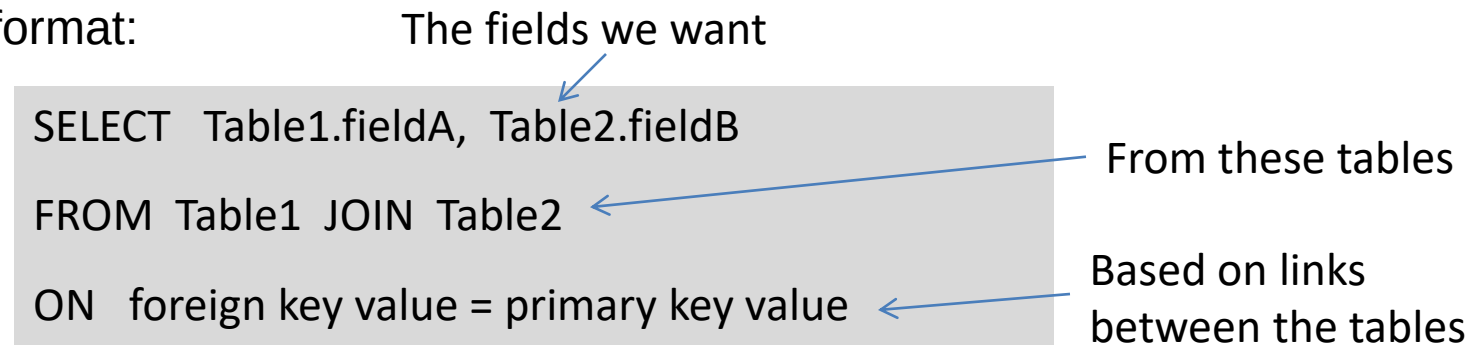
Example: Inserting Data

- Use INSERT INTO commands to insert each row of data records into all the tables.
- It is a good idea to insert data from the “outside in” order as well. This means insert data into tables that don’t have foreign keys first, then insert data into the more central tables with foreign keys.
- When inserting data into tables without foreign key, the primary key will be automatically filled in and incremented if the constraints are: INTEGER NOT NULL PRIMARY KEY
- This means we can fetch the primary key with each insert, and use it as the foreign key as we go to the tables that require foreign keys.
- Example:

```
# insert ‘CS’ into Majors table
cur.execute('INSERT INTO  Majors (major)  VALUES (“CS”)')
# fetch the id of ‘CS’ in the Majors table
cur.execute('SELECT id  FROM Majors  WHERE major = “CS”)
major_id = cur.fetchone()[0]
# use the primary key of Majors table as a foreign key for StudentsDB table
cur.execute('''‘INSERT  INTO   StudentsDB
                    VALUES ( 123, “Hopper, Grace”, major_id, 3.87)'''‘)
```

Example: Query for Data

- Use SELECT and JOIN commands to retrieve data from different tables.
- Because the data for one record are stored in different tables, when we query for data, often we need to use the SQL command JOIN in order to get data from multiple tables.
- SQL format:



- The JOIN command looks through all the tables that are joined together.
- The ON clause is used to select the correct link between the tables and to select the correct attribute.
- Example: Find names of students who major in CS

```
SELECT Students.name    FROM Students JOIN Majors
ON   Students.major_id = Majors.id   AND   Majors.major = “CS"
```

Going further…

- For data persistence, we can save our application data in a text file, CSV file, pickle file, JSON file, or database.
- We've discussed the basic concepts of working SQL databases to show how a database can provide flexibility in querying for data.
- There are many more features of SQL, such as the ability to link tables both ways. So far our tables have been set up in a one-to-many relationship, where there is one StudentsDB table that is linked to multiple other tables. But real world data modeling often involves a many-to-many relationship, such as a StudentsDB table and a classesDB table. One student can take many classes, and a class can have many students.
- In addition to SQLite, Python programs can also utilize other types of databases.

Up Next: System