Practical Session #13 - Persistence Layer

Tom Mahler
In This Recitation We’ll Cover

- What Is Persistence Layer
- Supporting Additional Features
- ORM - Object Relational Mapping
What Is Persistence Layer
What Is Persistence Layer

- Assume we have an application that is storing and accessing data to/from a DB, as seen in class

- If the application accesses the DB directly, each change in the DB structure, table format or application needs, will require changes in several parts of the code
What Is Persistence Layer

- Persistence layer is a design pattern for managing the storing and accessing of permanent data.

- It manages the communication between the application and the DB and creates a separation between the application logic and the DB access which allows for better code stability.
What Is Persistence Layer

- In order to demonstrate an implementation of the persistence layer design pattern we will review the implementation of the assignment tester as seen in class.

- Reminder, the assignment tester is a python application designed to grade students assignments.
What Is Persistence Layer

It uses a DB with the following tables:

- **Students**: which includes the id of the student and its name
- **Assignments**: which includes the number of the assignment and a string representing the expected output of the `run_assignment` method of the corresponding assignment
- **Grades**: which includes the grade of each student on a specific assignment
What Is Persistence Layer

- The implementation of a persistence later revolves around 3 types of objects:
  - DTO - Data Transfer Object
  - DAO - Data Access Object
  - Repository

- The full working code can be found in the class materials
DTO - Data Transfer Object

- A DTO is an object that represents a record in most cases from a single table
- Its variables represent the columns of the table
- DTOs are passed to and from the persistence layer
DTO - Data Transfer Object

- When passed from the persistence layer to the application logic, they contain the data retrieved from the database.

- When passed from the application logic to the persistence layer, they contain the data that should be written to the database.
The DTO naming convention is that a DTO named 'Abc' represents a table named ‘abcs’

We will use this convention in the future to map a DTO object to the table it represents
Examples

See code...
(See DTO.py)
DAO - Data Access Object

- These objects contain methods for retrieving and storing DTOs
- In most cases, each DAO is responsible for a single DTO
Examples

- See code... (See DAO.py)
In many cases, DTO and DAOs are not sufficient, since each DAO only knows how to handle a specific DTO, where should we put methods that aren't related to just a single DTO(table)?

For example:
- create_tables method?
- Queries that span multiple tables (using join)?
Repository

- which DAO should hold these methods?
  - The answer is in the repository

- The repository is similar to a DAO but it manages a group of related DTOs
Examples

- See code...
  (See Repository.py)
Using the objects described above, our application logic implementation can be something like this.
Examples

See code... (See spl13.py)
Supporting Additional Features
Looking at the method `print_grades` at the application logic implementation above, we can see that the method is inefficient.

In order to find the name of the student to whom belongs the grade, the method goes over all the students one by one.
Join Query

- This could have been prevented if we used a join query to match between the students grade and its name.

- But where will we place such a query?
  - The answer is obviously in the repository.
Examples

▶ See code...
  (See Repository_Join.py)
Join Query

- with that addition to the repository our print_grades function can simply be
Examples

- See code...
  (See Repository_Join.py)
Update

- Suppose we want our assignment tester to also support in appeals, we will need a method to update a student's assignment grade.

- The DAOs we have seen so far only supported 2 functions, insert and find.

- Let's try and add support for the appeals in the _Grades class:
Examples

- See code...
  (See DAO_Update.py)
Update

- Now we have support for appeals, but what if we want to support additional update functions such as update student name (in case of a marriage 😊)?

- In class we have seen the ORM and generic DAO classes designed to implement generic DAO methods.

- Let's first revise the ORM and then try and add an update method to the generic DAO class.
ORM - Object Relational Mapping
ORM - Object Relational Mapping

- The ORM is a method for mapping between a certain DTO object and its related table in a manner that suits any given DTO.

- Reminder, the DTO uses several assumptions in order to function properly:
  1. Each DTO class represents a single table.
Reminder, the DTO uses several assumptions in order to function properly:

2. The different DTO classes are obeying a common naming conventions:
   - A DTO class named Foo will represents a table named foos

3. DTO constructor parameters names == DTO fields names == table represented by the DTO column names
Examples

- See code...
  (See ORM.py)
Let's go over the code above line by line to make sure we understand it.

1. **Line 1:** using the inspect module, we get the arguments of the constructor of the `dto_type` received
   - `dto_type` is a class type, `__init__` is the constructor
2. **Line 2:** the first argument of the constructor (or any method) is 'self' so we remove it because it does not represent a column in the table
ORM - Object Relational Mapping

- Lets go over the code above line by line to make sure we understand it.

3. **Line 3:** gets the name of the columns from the table the cursor last executed on
   - Reminder, the ORM method will be called after a `select` has been executed

4. **Line 4:** creates an array that at each index \(i\), will hold \(j\) where \(j\) is the column number in the table of the \(i^{th}\) element in the constructor arguments
Let's go over the code above line by line to make sure we understand it.

5. **Line 5:** using the col_mapping created the previous line and the row_map method, it will create a DTO object for each record returned from the select command executed before the ORM method was called
Examples

- Using the ORM, we can construct a generic DAO

- See code...
  (See ORM.DAO.py)
Generic Delete

- Now after we have learned about the ORM and generic DAO, we can add more generic methods.

- Before we try to add a generic update, let's start with something simpler and add a generic delete.
Generic Delete

- Notice that the SQL delete command has a structure similar to that used in our find method.

- So adding delete will be just a minor modification to our find method that instead of returning the DTO objects using the ORM, it will simply execute the delete.
Examples

- See code...
  (See ORM_DAO_Delete.py)
Generic Update

- After we have seen how to add generic delete, let's finally try and add our generic update.

- In order to add generic update, we must allow for both a number of set values and a complex where condition.

- That will require us to use two different dictionaries, one containing the set values, and the other the condition.