University of Sunderland

Technology Management for Organisations
Workshop 7
PonyORM Entity Mapping

Aims of the workshop

Last workshop we looked into SQL syntax and looked at the web-based administrative tool PhpMyAdmin to investigate our RDBMS. We executed some simple SQL queries, and created a basic Python script to connect to the database (insecurely), and perform some simple queries.

This workshop will build upon this, continuing with the Generator Expressions introduced previously, and finally using PonyORM for some Entity mapping. In this workshop you will have access to a specific user, with your own database. This will enable you to modify your own tables.

You are encouraged to read around the topics introduced in today's workshop, many of these include documentation pages for both SQL and PonyORM.

Make sure to fill in your ePortfolio for Teaching Week 6 / Workshop 6 during/after the workshop.

Feel free to discuss the work with peers, or with any member of the teaching staff.

Reminder

We encourage you to discuss the contents of the workshop with the delivery team, and any findings you gather from the session.

Workshops are not isolated, if you have questions from previous weeks, or lecture content, please come and talk to us.

Exercises herein represent an example of what to do; feel free to expand upon this.

Exercises

With the exercises for this workshop, please discuss answers with a peer as well as a member of the delivery team.

For the exercises today we are hosting a database with the following credentials:

host: **europa.ashley.work**
username: **student_ followed by your student ID**
password: **iE93F2@8EhM@1zhD&u9M@K**
database: **student_ followed by your student ID**

E.g student_bh12xy for the username, you also have a database under this name.

Warning: I would recommend **not** changing the password yet.

Useful Links:

https://docs.ponyorm.org/firststeps.html
https://docs.ponyorm.org/entities.html
https://docs.ponyorm.org/api_reference.html#entity-definition
https://docs.ponyorm.org/working_with_entity_instances.html

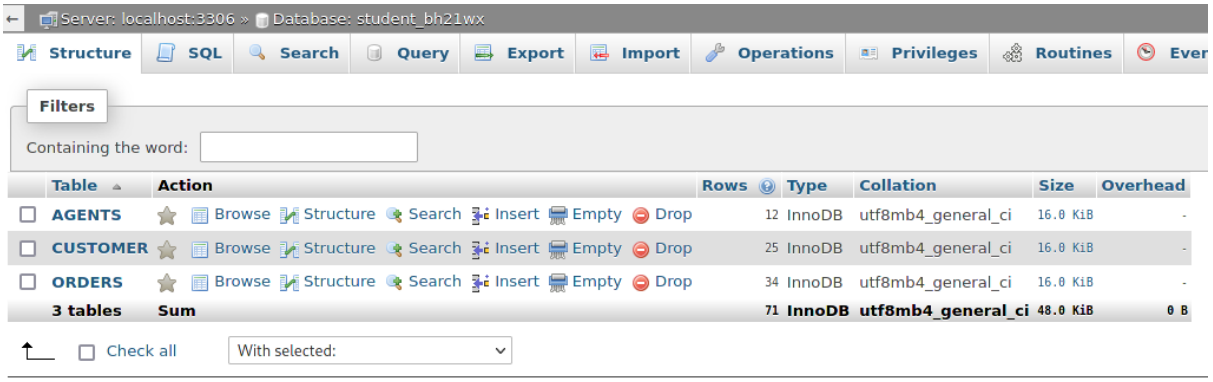
In general, you may find https://editor.ponyorm.com/ a useful tool from PonyORM which can design ER Diagrams, and map these directly into PonyORM.

Exercises

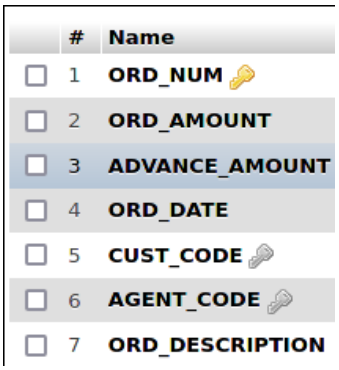
Exercise 1: Log in to the PhpMyAdmin interface using your new dedicated user. Have a look at the tables available to you. The following diagram represents the three tables presented.

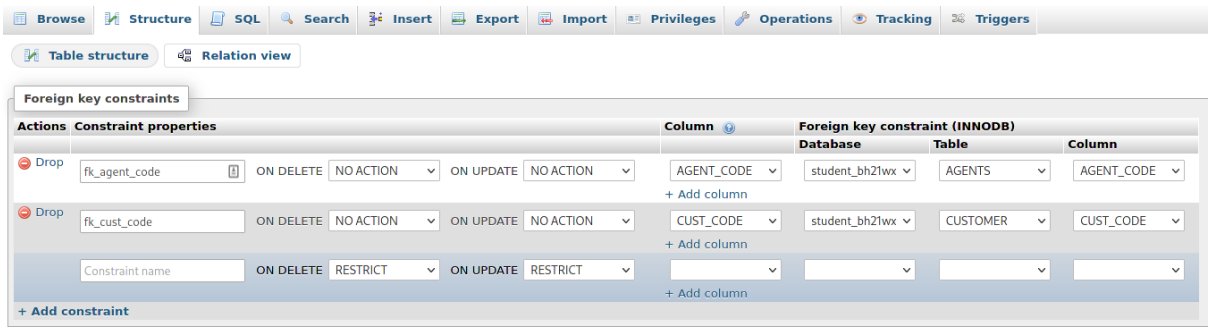
This is an example of a simple business database housing customers, orders, and agents. The lines connecting the tables show the relations between the tables. Here we can see that 'Orders' involves a customer and an agent and so this creates a **foreign key** reference (in red). However, each table here has an ID / Code as their primary key.

Explore the tables using the 'structure' view.

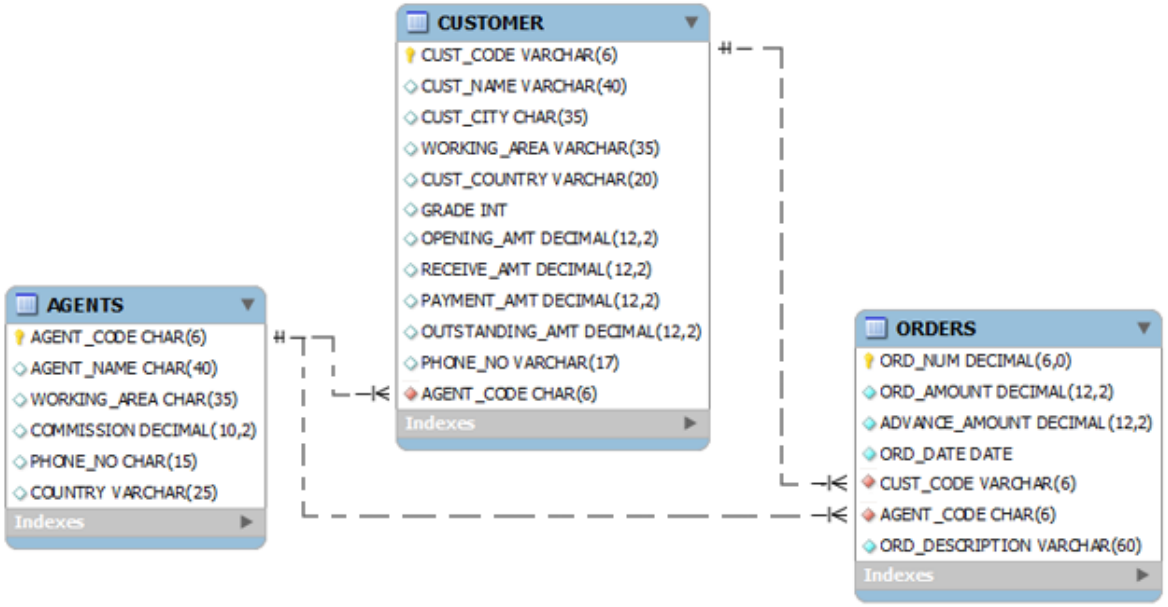


Consider the primary keys, foreign keys, and using 'relation view' you can see constraints on the table. For our implementation ON DELETE and ON UPDATE are set to NO ACTION to prevent additional issues. For now, this just formally denotes a field's connection to another table.





Note: This is an example of how such a database could be set up. There are other ways to design such relations, and their attributes.



Exercise 2: Go to the previous workshop, and use your new credentials in your PonyORM code (Remember username and database need changing!). Swap out your select queries, and substitute in some of the new tables you see.

Note: Replace db.select with db.execute

1. Select ALL Customers
2. Select ALL Orders, only returning the order amount. Once this has been returned from your query, use python to sum these.
3. Obtain the MAX commission of the agents in SQL alone.
4. Obtain the MODE of the working area of the agents
    a. Retrieve all records, returning only the working_area column via SQL query
    b. Using python, calculate the mode of those values.
5. Create a new Customer, be careful with the foreign keys. Remember: Each customer needs to be assigned a valid agent!

Feel free to try other queries from the last workshop.

Hint: MIN, MAX are functions available in SQL itself. MODE is not!

Exercise 3: Mapping our Database tables to/from Python requires us to define the schema within Python. Thankfully PonyORM provides this for us. You may wish to consult https://docs.ponyorm.org/entities.html from the documentation for defining entities.

In principle, we create a class which represents our 'Entities', which are usually our tables. These all have attributes of various types, lengths. Some of these may have additional constraints on them, such as the requirement of being entered, being unique, being a primary key, etc.

Let's create a simple one first. Add to your pony.orm import line: Entity, PrimaryKey, and Required.

```
from pony.orm import Database, db_session
from pony.orm import PrimaryKey, Required


db = Database()

class Cat(db.Entity):
      name = PrimaryKey(str)
      age = Required(int)

db.bind(...) # And so on.
db.generate_mapping(create_tables=True)
```

`generate_mapping()` here is the magical function call. This takes any of our custom created Entities with PonyORM and maps them to the equivalent SQL tables for our database. Here we specify a keyword argument create_tables=True as we want PonyORM to make the table if it doesn't exist. By default, this function will attempt to find a table in our database which matches the schema outlined in our Entity class (I.e it will look for Cat with name, and age matching).

Once executed, go to PhpMyAdmin, and check out the new table generated.

Exercise 4: At the moment, PonyORM seems to be doing magic. Let's try to see what's going on under the hood.

Add this with the rest of your imports:

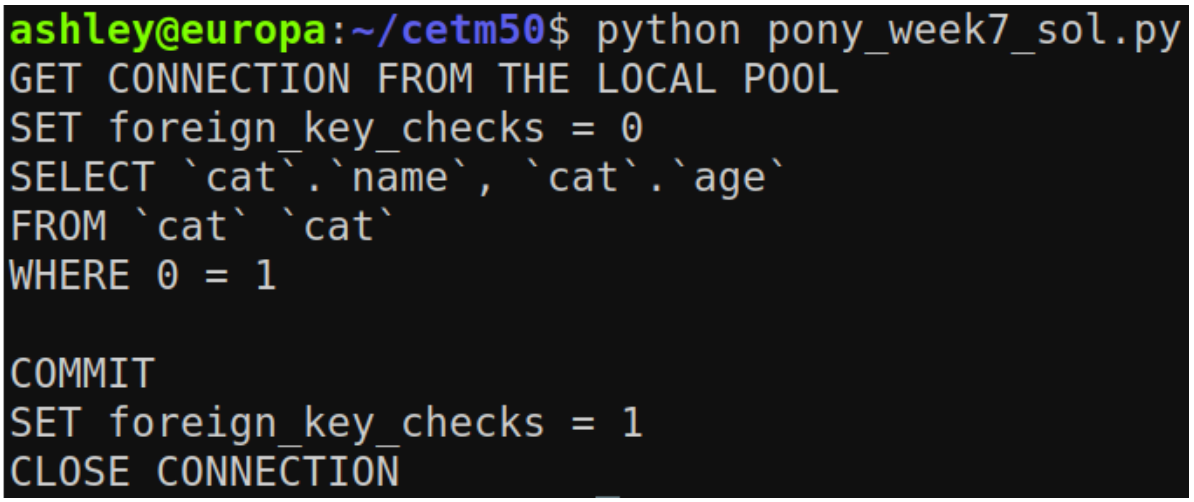
```
from pony.orm import set_sql_debug
```

Then before the `generate_mapping()` call, add:

```
set_sql_debug(True)
```

Re-run your solution, and you should now see SQL printing in the console showing what PonyORM is executing on your behalf.
You can see the establishing of a connection to the server, and the committing of any queries.

Note: This is a debug utility, which is designed to help you understand. Feel free to turn it off if you wish.



Exercise 5: We now have a class within Python, called Cat, which maps directly to an SQL Table in our RDBMS. PonyORM is handling all of the back and forth conversions for us. However, our table is empty and is useless unless we put some data in it.

Recall when we created objects from Classes, as PonyORM has mapped this to Python entities, this should be simple.

```
with db_session:
      my_cat = Cat(name="Kira", age=2)
```

Once executed we should see the following transaction SQL take place:

```
GET NEW CONNECTION
INSERT INTO `cat` (`name`, `age`) VALUES (%s, %s)
['Kira', 2]
```

Now when we check our PhpMyAdmin table view, we can see Kira has been entered into the database.

Exercise 6: At the moment, our Cat is defined very loosely. We simply have a name and an age. Both of these are required attributes.

Design a better Entity definition for Cat. Consider the attributes / fields we might hold for an animal. In general we have the following types available from PonyORM: Required, Optional, PrimaryKey, and Set. Let's re-imagine our Table for the purposes of a new veterinary clinic.

Once you have come up with a design, modify the class Cat to include these new entries. Remember: If you're using a new type (E.g Optional), you will need to import it just like PrimaryKey and Required at the top of your solution.

Hint: You will need to DROP the table from the Database if we fundamentally change the schema.

Exercise 7: Populate your new Cat table with many cats. Try to make a variety of entries. If you included Optional in your design for Ex 6, try leaving some of these blank.
For example, what happens if you forget to create your cat with the primary key? (Cat(age=2))

Exercise 8: We've put items into our database, but in order to be useful to our services and programs, we need to retrieve data from the database.

With SQL tables we generally index based on the primary key. So far we have created queries which can also select records based on other attributes. Let's look at how we can do that with PonyORM.

Reminder: Make sure our 'inserting' code isn't being executed here, otherwise you might get some errors.

As our Cat entity / Class has the primary key name, we can directly obtain cats by their name. (You may have fixed this in your design, but for the purposes of this example, we're still using my brief Cat class).

```
print( Cat["Kira"] )
```

If you have the SQL Debug on you can see it querying for Kira directly.

```
GET NEW CONNECTION
SELECT `name`, `age`, `owner`
FROM `cat`
WHERE `name` = %s
['Kira']

Cat['Kira']
```

Assign Cat["Kira"] to a variable **retrieved_cat.** We can now work with a Python entity directly which represents retrieved data from our database. As such, we can call attributes of the cat.

```
print( retireved_cat.age )
```

Exercise 9: We should now have some sufficiently complex cats in our Database, let's begin some more complex queries.

We can use the generator expressions covered in the last workshop to select all of our cats. Here we want all of our cats. I wrap the select() call in a print to see what we have.

```
from pony.orm import select
print( select( c for c in Cat ) )
```

We should get something like: **<pony.orm.core.Query object at 0x7f018889f250>** which tells us that we have a Query Object (https://docs.ponyorm.org/api_reference.html#Query)

Just like with List comprehensions and Dict comprehensions, these Generator Expressions are iterable. The only difference is that Generator Expressions are lazy and will only grab new results when absolutely necessary (saving a lot of memory!)

1. Assign a variable to the select generator expression
2. Directly iterate over this using a for loop.
    a. Each loop, print the result.

Note: We can also call `.count()` on the query object to find out how many results we have.

Exercise 10: Modify your select generator expression to return just the cat's age rather than the entire cat.

Exercise 11: Using your knowledge of filtering from List Comprehensions, modify your generator expression to only select the cats if their age is less than 4 (ensure you have some older cats in your database!).

Note: Make sure to check against the SQL debug output to check what's happening with your generator expression.

Exercise 12: Kira has just been to see the vets for a regular checkup, and they wish to modify the data on file for her. Currently there's no way to change the data once we enter it.

1. Retrieve data for Kira from the database. Either via Cat["Kira"] or a select generator expression. Assign this to a variable **current_cat**
2. Kira is now a year older. If we want to update a field, we can directly modify it similar to any other entity in Python.
   E.g `current_cat.age = current_cat.age + 1`

As we are within a db_session context, once we exit this block all transactions are committed if successful. Therefore, once this context closes Kira should automatically be updated within the Database. Check in PhpMyAdmin if this is the case.

Try modifying other and different attributes from your Cat Class, observing their changes in the database.

Exercise 13: We can create, we can retrieve, we can update, but so far we don't know how to delete. (The final part of CRUD).

There are two ways of accomplishing this:

1. Working with the results objects directly, we can call .delete() on them to execute an SQL DELETE query.
2. Using a delete generator expression. Similar to select (remember import delete similarly)

   E.g We can construct a query which matches records we wish to delete.

```
delete( c for c in Cat if c.name == "Steve" )
```

Try both techniques of deletion for some Cats in your database.

This covers all of the CRUD operations in brief. We will be covering some more aspects in later workshops; however, feel free to read documentation as this is a valuable skill as technology is forever changing. Reading and understanding these documents is key.

END OF EXERCISES