

COMP3062 Advanced Databases

99-09 Assignment 2: ONTOS Programming

1 General Information

The second assignment is an ONTOS programming assignment which is due at 1700 (ie: 5pm) on Tuesday of week 13 (26th October). Use the on-line “mark” to submit your files. The name of the assignment is “ass2”. A `Makefile` must be submitted. When we run

`make`

it must do the following things.

1. compile two commands: namely, `init-db` and `unit-db`.
 - `init-db` which will be used to load data from files into an ONTOS database.
 - `unit-db` which will be used to enquiry the ONTOS database. The usage of the `unit-db` will be given later in this handout.
2. create a database – create a directory, copy the schema file into the newly created directory, and register area and database. Refer to `make db` in the `Makefile` mentioned below.
3. classify a database called `${USER}tutorialDB`. Refer to `make classify` in the `Makefile` mentioned below.

Also `make clean` must be provided to clean everything.

The `Makefile` given at
`/data2/dcs/comp3062/public/examples/ontos/introduction`
contains relevant examples.

You cannot use `g++` facilities in conjunction with ONTOS, so you must use `CC`.

You are not allowed to use `OSQL` for this assignment. When it is needed to access all objects belonging to a class, you have to use `OC_instanceIterator`. Such an example is given in the well-known directory of `introduction` — use “`make ass2`” to compile the command called `mainExtension`.

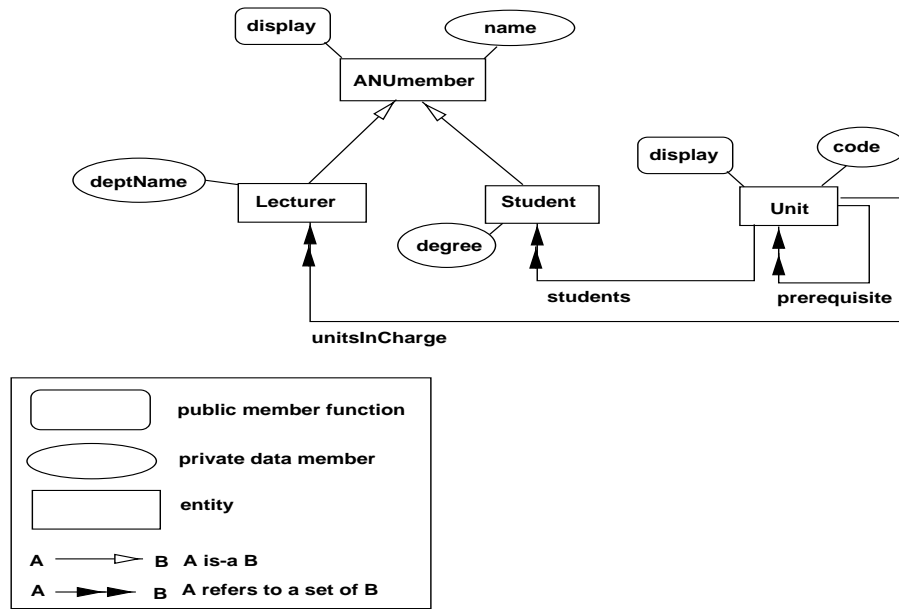


Figure 1: A Database Schema

2 Database Schema

A database schema is given in Figure 1 which is the same as we used in the first assignment.

- **Entities:** All entities given in Figure 1 have to be truly implemented with C++ classes. The is-a type hierarchy should remain unchanged. All classes must be persistent classes.
- **Data Members:** All data members must be private data members. As for a data member, called AB , associated with $A \rightarrow B$, you have to implement it as an **abstract reference** to an `OC_Set` class object which manages a set of B objects. Because ONTOS DB treats `char*` as a pointer to a non-persistent object, you have to use an array of chars in a persistent class to manage a string if needed.
- **Constructors and initial values:** You must provide constructor(s) for a class to initialise ALL data members defined in that class and its super classes.
For simplicity, the default `degree` of students is `BSEng`, and the default `deptName` of lecturers is `DCS`.
- **Member functions:** All the functions given in Figure 1 must be public member functions. The `display` function defined in `ANUmember` must be

a virtual public function which will be inherited to its subclasses. The `display` function must print all data members once and only once.

- On `ANUmember`: name only.
- On `Student`: name and degree.
- On `Lecturer`: name and `deptName`.
- On `Unit`: code, a set of student names, a set of lecturer names, and a set of (direct) prerequisite unit codes.

Note: You can add any number of member functions onto a class but you cannot add/remove data members.

- **Extensions and Sets:** Don't use the set classes and the iterator classes you have implemented in your first assignment. You must use the similar classes provided by ONTOS DB. Refer to the laboratory handouts and the reading material.

3 Coding Style

Each class must be implemented in two files. As for `ANUmember` class, you need `ANUmember.h` for `ANUmember` class. In header files, don't include any header files except the header files provided by ONTOS because the `classify` utility can only handle persistent classes. You need to have `ANUmember.cxx` to implement member functions for the `ANUmember` class. The `main` function and all non-member functions should be kept in different C++ source files.

4 Data

These sorted files are given at `/dept/dcs/comp3062/public/ass1/data`.

- The `student-unit` file contains a list of pairs. Each pair represents a relationship between a student name and a unit code the student takes.
- The `lecturer-unit` file contains a list of pairs. Each pair represents a relationship between an employee name and a unit code.
- The `prerequisite` file contains a list of pairs. Each pair represents a relationship between two units. The second is a prerequisite of the first.

An example is given showing how to read data from a file `/dept/dcs/comp3062/public/ass1/src/io`.

5 Construction of an ONTOS Database

You need to use `init-db` to load all data into an ONTOS database.

`init-db`

6 Queries

You need to support the following queries.

1. Given a string which can be either `ANU-Ext`, or `Student-Ext`, or `Lecturer-Ext`. Print the result of the `display` function for all objects in the corresponding extension.
2. Given a unit code (a string). Print the result of the `display` function for the corresponding unit object that has the same unit code in the `unitExt`.
3. Given the full name of a Lecturer. Print the list of units he/she is involved.
4. Given two full students names. Print the codes of the units the students have in common. Note: the student names given as arguments may be identical.
5. Given two unit codes. Find out if the second unit is a prerequisite of the first unit (directly or indirectly).
6. Given a unit code, list all units students have to do in order to do that unit (directly or indirectly).

The command `unit-db` shall take a file-name as its argument.

`unit-db file`

In the file, each line is a pair of the question number given above and the query. A question number can appear multiple times in the file. The number of lines in the file can vary. Examples are given below.

```
1 ANU-Ext
2 comp2031
3 Mr.Right
2 comp2030
4 Mr.A Mr.B
5 comp2030 comp2029
6 comp2033
0
```

where 0 indicates that there are no more queries, and that the program should exit normally.

7 Submission

Use the on-line `mark` tool to submit a `Makefile` and all necessary source files (`*.cxx`, `*.h`). Running the command

```
make
```

must produce executables called `init-db` and `unit-db` and create and classify a database as outlined in section 1.

All files must be submitted by 1700 (ie: 5pm) on Tuesday of week 13 (26th October).

7.1 Extensions and Late Penalties

Extensions will only be granted in exceptional circumstances. Requests should be made directly to Lex Weaver and supporting documentation will need to be provided.

Late submissions will be accepted if made within seven (7) days of the deadline. However, a penalty of -10% for each day late (or part thereof) will be applied to these submissions. Thus, for assignments being marked out of 30, the penalty would be -3 marks per day.

8 NOTE

ONTOS doesn't recover database area entries from its internal tables when areas are deleted. This is the case even when we run "`DBATool -e delete area $USERtutorialDB`", which is used in the example `Makefile`. So, the number of areas will eventually exceed the ONTOS limit. To cope with this, we have to have a policy.

- register area and database ONCE.
- run "make clean-db" only ONCE.
- * run "make fresh-db" when you want to copy a clean area and rerun classify. Don't use make clean-db unnecessarily!

The commands for "make fresh-db" can be found in the `Makefile` at `.../introduction/Makefile.fresh`

The ONTOS system tables will periodically (every week or two) be rebuilt in order to minimise the disruption that could be caused by this problem. This will delete all database areas defined at the time. However, it's better for this to happen preemptively than for ONTOS to seize up at 1700 on the Friday before the assignment is due!