**University**

**Problem Description**

City University (CU) keeps an extensive database for daily operational purposes. The database includes information pertaining to the campuses, faculties, buildings, personnel, degrees, and subjects offered, and other data derived from them. Information Technology Services (ITS), responsible for maintaining the database system within the university, decided to use an ORDB and Oracle™ for the database implementation.

CU has eight campuses around the state of Victoria. The Campus database is linked to the Building and Person databases. Although each campus offers different degree courses and has different faculties, at this stage, there is no direct link from these data to the Campus table. Figure 1 shows the sample data for this table.

CU has five faculties, each of which is an aggregation of a different department, school, and research centre. Each of them is implemented as a separate object and has derived object tables. As we do not need to access the data of the departments, schools, and research centres directly for this database system, the data is implemented using a nested table. Figure 2 shows the sample for the Faculty table and its nested tables. Note that the attributes school\_prof and dept\_prof are themselves objects. Thus, they have their own attributes including name, contact, and year of inauguration. An attribute unit in the Research\_Centre nested table will have more than one value and thus needs to be implemented using collection types.

Each campus has several buildings, each of which is an aggregation of different rooms such as offices, classrooms, and labs. The faculty can occupy many buildings. However, one building can only be allocated to one faculty. Note that there is an attribute bld\_location, which is the location of the building on the particular campus map.

As mentioned previously, a building can be divided into offices, classrooms, and labs, each with its own attributes. Figure 4 shows the sample for the Office, Classroom, and Lab tables. Note that the attribute lab\_equipment in Labs has to be implemented using collection types. For this aggregation, we are using the clustering technique instead of a nested table because there will be

association relationships needed between the part table Office and another table to show the staff who occupies the office.

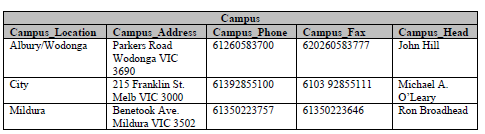


Figure : Campus table

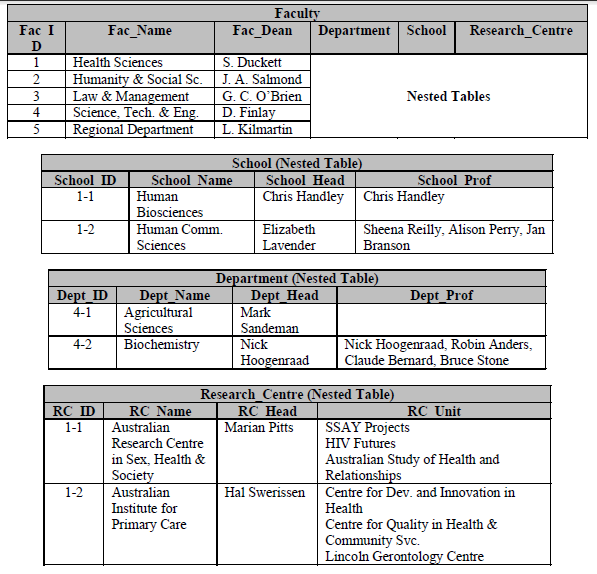


Figure : Faculty table and the nested tables

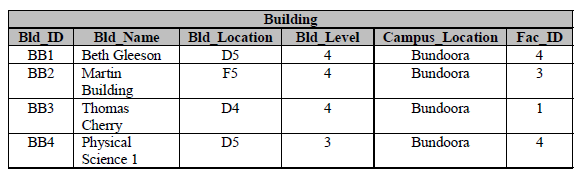


Figure : Building table

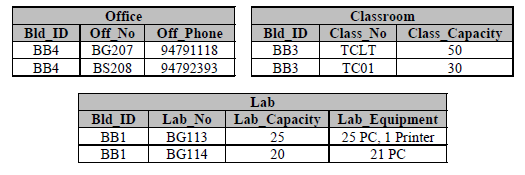


Figure : Office, Classroom, and Lab tables

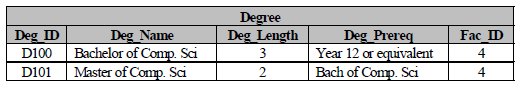


Figure : Degree table

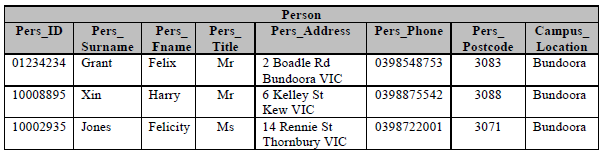


Figure : Person table

Every faculty offers students a number of degrees. The information about the degree is stored in the Degree table (see Figure 5). Obviously, one particular degree can be offered by only one faculty.

One substantial part of the database is the personnel data. The university personnel can be categorized into two major types: staff and student. A staff can be categorized in more detail into administrator, technician, lecturer, and tutor. A lecturer can further be categorized into senior lecturer and associate lecturer. A tutor, on the other hand, can also be a student and, thus, has to be implemented in a multiple inheritance relationship.

While Figure 6 shows the Person table, Figure 7 shows the tables for its subclasses. Empty fields show that the attribute can be null.

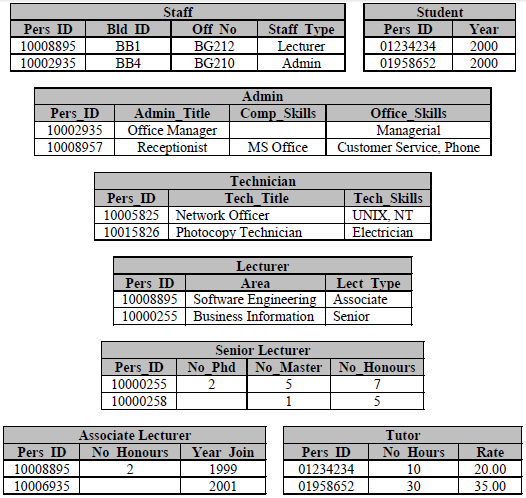


Figure : Person’s subclass tables

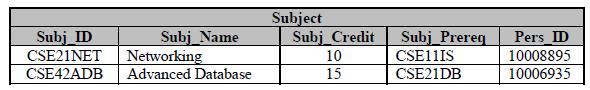


Figure : Subject table

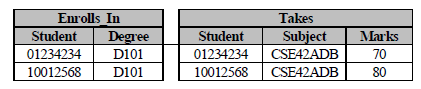


Figure : Enrolls\_In and Takes tables

The Student\_T class is linked to the Degree\_T class. One student can take more than one degree at a time. The Student\_T class is also linked to another class, Subject\_T. It contains the information about the subject ID, subject name, subject credit, subject prerequisite, and its description. On the other hand, the Subject\_T class is linked to the Lecturer\_T class, which obviously shows the lecturer in charge of the subject. Figure 8 shows the Subject table Figure 9 shows the tables associated with the Student table: respectively, the Enrolls\_In table that is formed by the association to the Degree table, and the Takes table that is formed by the association to the Subject table. Note that the tables do not exactly store only the ID, for example, student\_ID in the Enrolls\_In table. The whole object with the particular ID is being referenced because of the implementation of object references.

ITS implements the generic methods inside the classes, which will need a lot of updates. They include Subject\_T, Degree\_T, and all the classes derived from Person\_T. There are also generic stored procedures for insertion and deletion into tables that are not derived from objects, that is, table Enrolls\_In and table Takes.

Beside the generic methods, there are some user-defined queries that are frequently made for this database. These user-defined queries will be implemented as user-defined methods, listed below.

* Method to show the names and the heads of the schools, departments, and research centres of a faculty. This method is implemented in Faculty\_T.
* Method to insert the data of a building into a new table, namely, Building\_Details. This method will be implemented in Building\_T.
* Method to display the details of the offices and their occupants. This method will be implemented in the Office\_T class.
* Method to save into a new table, namely, Degree\_Records, which will store the degree details and the number of students enrolled in it. This method will be implemented in the Degree\_T class.
* Method to show the details of the lecturer that will be implemented in the Lecturer\_T class