A Case Study of Using an Object-Relational Paradigm in Building a Web Database Application

J. Wenny Rahayu [1] *David Taniar* [2] *Lee Nung Kion* [3] *Eric Pardede* [1]

[1] Department of Computer Science and Computer Engineering
La Trobe University
Bundoora, Victoria 3083, Australia
Email: wenny@cs.latrobe.edu.au
E.Pardede@latrobe.edu.au

[2] School of Business Systems
Monash University
PO Box 63B, Clayton, Victoria 3800, Australia
Email: David.Taniar@infotech.monash.edu.au

[3] Faculty of Cognitive Science and Human Development
University Malaysia Sarawak
Address
Email: lnkion@pl.jaring.my

Abstract

In this paper, we would like to share our experiences in building a web database application using an object-relational paradigm. The system we built is basically an online system for casual tutors to claim their work for payment. At the design stage, we use an object-oriented design. Since the database backend is a relational database management system (i.e. Oracle), which also supports some object-oriented features, we need to apply a transformation methodology to map our object-oriented design into relational tables incorporating some object features in Oracle. Once the necessary object types and tables have been created, at the programming stage we use a PHP scripting language accessing the Oracle database where the data is stored.

Keywords

Object-Oriented Design, Object-Relational Databases, Web Databases, and Online Database Application.

1 Introduction

In this section we provide the related work and the motivation of using object-relational paradigm for web database application.

1.1 Related Work

Online database application is gaining popularity among not only IT community but also general community because of a wide availability of Internet access. In web database applications, the database is accessed through the web [14]. Generally the database is stored in a Relational Database Management System (RDBMS) and web access is provided by a scripting language (e.g. Php) connected through an appropriate middleware [5, 15].

Like any other database applications (web and non-web), database design is an important initial step. After this conceptual design, implementation can then be carried out. The emergence of the Object-oriented (OO) methodology has shown its capabilities in modelling the real world better than the earlier relational methodology [7]. The Entity Relationship (ER) modeling [2] technique is the most widely use technique in the industries for conceptual web database design. But, it has several deficiencies, firstly it unable to capture the data modeler intents, especially for large complex applications. Secondly, it lacks of expressive power that represents the real world entities and finally, the transformation into relational (normalization) database takes many steps and results in many redundant small tables. The un-normalization step is necessary to avoid redundancy and create more effective relations. There are many extensions to the ER model [1], but so far none of them is widely accepted in the industries.

The OO modeling technique has becomes a very promising alternative to the ER modeling technique. With the introduction of Unified Modeling Language (UML) standard many web database has been developed using this OO model. The object oriented database management system (OODBMS) was introduced to directly encapsulate the OO model in the conceptual design step. Unfortunately, the OODBMS implementations are still lacking performance as compared to RDB in many aspects [4].

The shortcomings in relational modelling such as ER modelling and object-oriented modelling have given us motivation to use the object-relational modelling that will combine the good features of each previous data modelling.

1.2 Object-Relational Modeling

A more promising approach is to extend the matured features of RDB to support object features such as abstract data type and objects in Object Relational Database Management System (ORDBMS). Vendors such as Oracle, Informix, and Microsoft SQL server have extended their RDBMS to support objects features [3]. The RDBMS is still the most famous web database implementation platform due to few reasons, firstly, the relational database is supported by most vendors, secondly, it can be easily obtained in the market, event the free version such as MySql, lastly, most web programming language support relational database access and the resulting web database can be easily ported to other programming language. The ORDBMS is predicted to gains more popularity in the near future as more complex web database requirements and more vendors are directing this way.

In this case study we adopt an object-oriented design to design the database. The system we built is an online claim system used by casual tutors to claim their work for payment.

As a database backend, we choose Oracle DBMS. Since version 8, Oracle has provided some object functionalities, such as objects, ref types, collection types, nested objects, etc. However, Oracle is not by any means an Object-Oriented Database Management System (OODBMS).

Oracle is an RDBMS with some object features. This new era of DBMS is often known as "*Object-Relational DBMS*" [13]. Consequently, database design using an object-oriented modelling needs to be transformed into object-relational database schema for implementation [12].

In this paper we focus on our experience in designing a web database application using an object-oriented design and mapping it to an object-relational database. We will also show how the implementation looks like on the web.

The rest of this paper is organized as follows. Section 2 describes the tutor claim system, including the design using an object-oriented modelling. Section 3 explains an object-relational transformation whereby the object design in Section 2 is mapped to object-relational database schemas for implementation. Section 4 presents an implementation overview of the system. Section 5 discusses problems and limitations of object-relational features. Finally, Section 6 gives the conclusions and explains future extension.

2 A Tutor Claim System: The Case Study

In this section, we briefly describe the tutor claim system, and the design aspect of the system, using use case and object-oriented modelling.

2.1 Descriptions

The tutor claim system is an online system whereby casual tutors can claim their work for payment. This payment claim is done fortnightly as the payment in the Australian University system is carried out every fortnight. Casual tutors are teaching assistant that we hire for each subject in each semester. All of them are students, mostly postgraduate students. These tutors are hired by the department to assist the lecturers in delivering the subject. This assistance is normally in a form of tutorial and lab support, consultation, and marking (exam and assignments).

Prior to the online system, tutors have to lodge a claim form every fortnight to be approved by the associated lecturer in order to be processed in the financial system. With this online system, payment claim can be done through the web. Database maintenance is done by the general office personnel (administrator) who maintains tutor details, subject details, semester, budget, etc. Each lecturer normally has a certain budget, which can be used to hire casual tutors for the semester. Each lecturer still needs to approve/reject claims made by their tutors. Once a claim is approved, the budget is appropriately reduced, so that the lecturer knows how much budget he/she still has.

2.2 Use Case

In this system, three actors are identified. They are: *Staff* is person(s) who is handling administrative function in, *Supervisor* is a person who has authority to appoint a tutor/tutor to assist him/her in a subject he/she teaches in current semester and allocates the budget for the tutor/s. It is also known as lecture. *Tutor* is a supervisor's assistant. Fig. 1 shows the Use Case diagram.

The Use Case diagram shown in Fig. 1 also shows seven use case scenarios: from manage tutor to manage budget, and view report and claim processing. Each of these scenarios is associated with an actor. It also requires certain pre and post conditions, and performs certain tasks. Fig. 2 gives a table explaining these elements for each use case scenario.

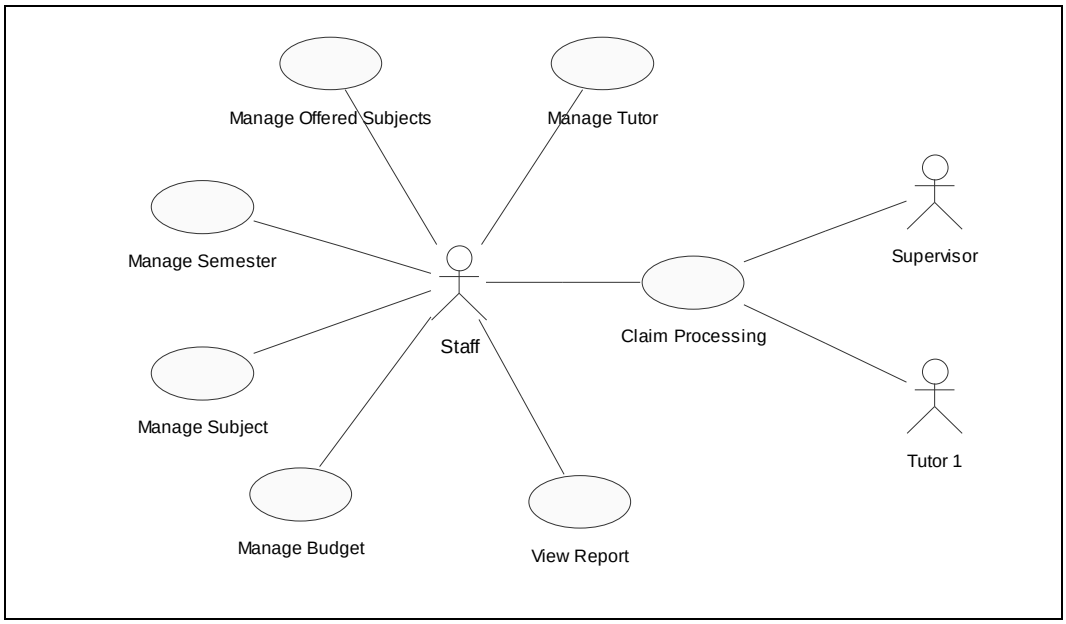


Figure 1. Tutor Claim System Use Case Diagram

Use Case Task	Actor	Preconditions	Postconditions	Basic Course
Manage Semester	Staff	None	Semester data is created / updated / deleted	Add / delete / edit semesters
Manage Subject	Staff	None	Subject is created / updated / deleted	Add / delete /edit subjects
Manage Offered Subject	Staff	Semester, Subject and Lecturer are already created	Offered subjects for current semester is created / updated / deleted	Add / delete / edit offered subjects
Manage Tutor	Staff	The system assumes a tutor is requested by a supervisor by submitting Budget form or by other means manually	Tutor is created / updated / deleted	Add / delete / edit tutors
Manage Budget	Staff	Supervisor submits a Budget Form. Offered Subject and Tutor are already created.	Budget is created / updated / deleted	Add / delete / edit budgets
View Report	Staff	None	Report is generated	None
Claim Processing	Tutor, Supervisor, Staff	Budget is already created	Claim is processed	Submit / approve / verify / pay claims

Figure 2. Use Case Scenarios

2.3 Object-Oriented Design

Fig. 3 shows an object-oriented diagram of the online tutor claim system. It consists of 12 classes. Class Users forms an *inheritance* hierarchy with classes Tutor, OfficeStaff, and Supervisor as its subclasses. Also notice that the inheritance is a *partition inheritance*, meaning that the instances of class Users must be one and only one of its subclasses [8].

In the diagram, there are two *aggregation* (whole-part) hierarchies. One aggregation is where class Claims consists of class ClaimLines, and the other aggregation is where class Budget consists of class ActivityType.

The association relationships are either uni-directional or bi-directional associations. Uni-directional associations are denoted by the directed arrows, whereas bi-directional associations do not use arrows. Either of them, the cardinality for each association can be 1-to-1 or 1-to-many. For example, a uni-directional 1-to-1 is between class OfferedSubject and Class SemesterInfo, and a bi-directional 1-to-many is between class Tutor and class Budget.

3 Object-Relational Transformation

As object-relational approach is developed using object-oriented model, before we discuss the transformation we need to know the aspect of object-oriented conceptual model (OOCM) that can be transformed. OOCM encapsulates the structural/static as well as behavioral/dynamic aspects of objects.

The static aspect involves the creation of the objects and classes that also includes decisions regarding their attributes. In addition, static aspect in OOCM also concerns on the relationship between objects, i.e. association, inheritance, and aggregation. Each of these relationships is associated with a set of constraints. The dynamic aspect of the OOCM involves the creation of the routines. Routines are specified as operations or methods, which are defined in the class that describe the object. The specified routines are the only operations that can be carried out on the attributes in the object.

In this paper, we only deal with the first aspect, which is the static aspect. Thus, there is a room for future research on dealing with the dynamic aspect transformation of object-relational approach for web database application.

For static aspect, we deal with different kind of relationships. There are three class relationships as noted in Fig. 3; they are *associations*, *inheritance*, and *aggregation*. In the following sections we provide mapping of each of these relationships. In the mapping process, the object diagram in Fig. 3 will be transformed into tables to be directly implemented into Oracle DBMS. The object-relational transformation rules that we use here are adopted from our previous work. Readers interested in object-relational transformation methodology can consult our existing papers [6, 7, 8, 9, 10, 11, 12]. In the following sections we summarize the results of the transformation process.

3.1 Mapping Associations

In this section, we describe transformation of 1-to-1, 1-to-many, and many-to-many association relationships.

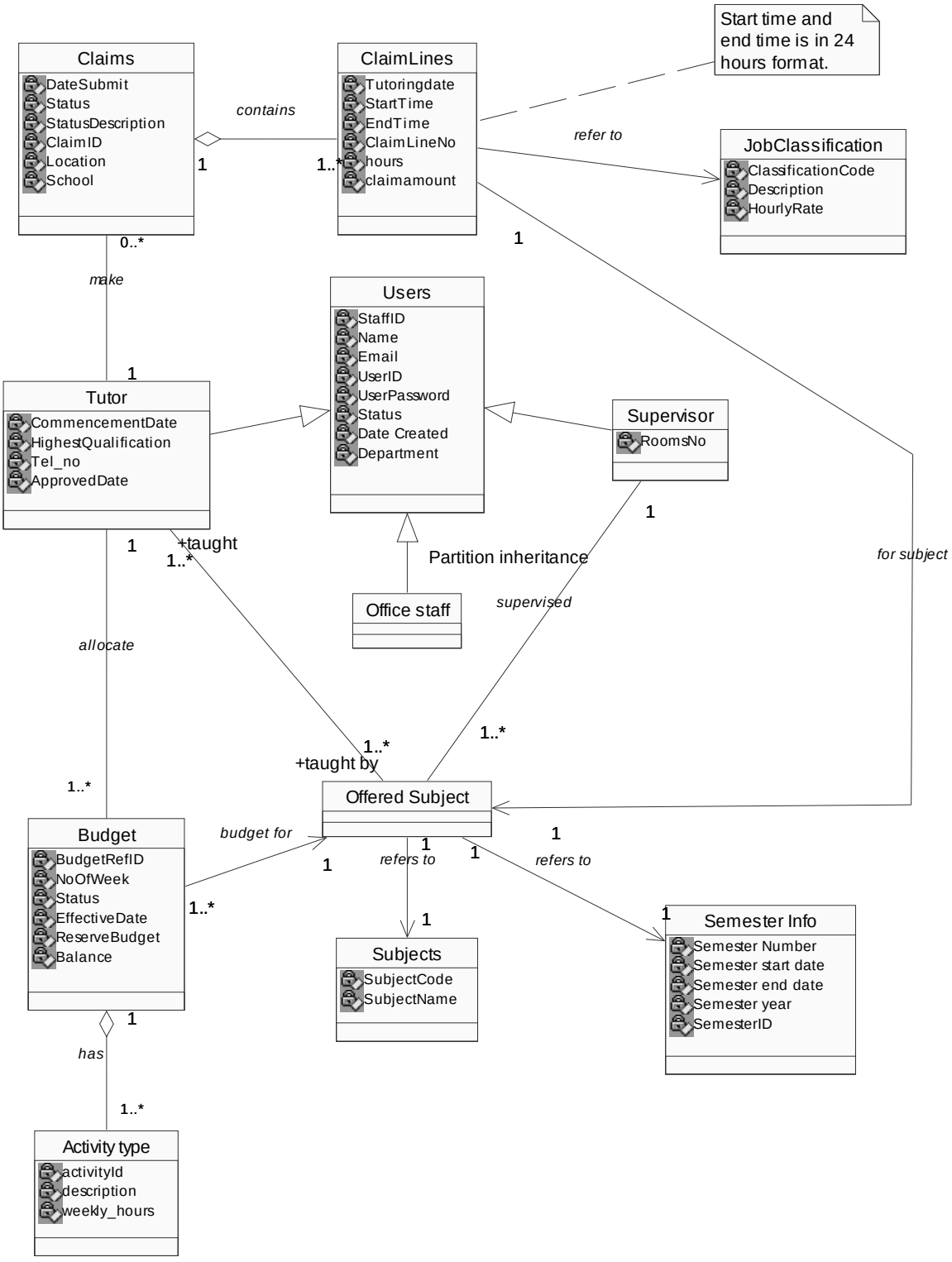


Figure 3. Object-Oriented Diagram

3.1.1 Transformation of 1-to-1 Association

There are a few 1-to-1 association relationships in the object diagram as shown previously in Fig. 3. To describe 1-to-1 association transformation, we use the association between OfferedSubject and Semester. The first step is to create an object type. This object type then becomes a table with additional information such as the primary key of the table. For example:

```
CREATE OR REPLACE TYPE Semester_typ AS OBJECT (
    SemesterID          NUMBER,
    Semester            NUMBER(1),
    Semester_year       NUMBER(5),
    Semester_start_date DATE,
    Semester_end_date   DATE,
    Current_sem         CHAR(1) );

CREATE TABLE Semester_tab OF Semester_typ
   (PRIMARY KEY SemesterID);
```

For the OfferedSubject class, we also need to create an object type. The OfferedSubject class needs to refer to the Semester table to know exactly which semester number and semester year a subject is offered. To do so, we create a reference from OfferedSubject class to Semester class. As OfferedSubject is also associated with Subjects, a proper reference is also linked with Subjects. In the OfferedSubject table, we need to use a SCOPE FOR clause to identify the domain scope of the references identified in the object type.

```
CREATE OR REPLACE TYPE Available_sub_typ AS OBJECT (
    subject_code_ref REF Subjects_typ,
    semester_ref REF Semester_typ);

CREATE TABLE Offered_Subject OF Available_sub_typ(
     SCOPE FOR (subject_code_ref) IS Subjects,
     SCOPE FOR (semester_ref) IS Semester_tab);
```

3.1.2 Transformation of 1-to-many Association

We use the association between Tutor and Budget to illustrate 1-to-many association transformation. In this case, one tutor may have many budgets, each budget object will corresponds to a subject it tutoring on. To transform this association, one REF type attribute is added to the Budget class. This attribute will reference to the ROW object table of Tutor type. The primary key for Tutor table will be the UserID.

```
CREATE OR REPLACE TYPE Tutor_typ AS OBJECT (
    Tel_no               VARCHAR2(10),
    HighestQualification VARCHAR2(50),
    ApprovedDate         DATE );

CREATE TABLE Tutor OF Tutor_typ;

CREATE OR REPLACE TYPE Budget_typ AS OBJECT(
    BudgetID             NUMBER,
    BudgetRef            VARCHAR2(15),
    EffectiveDate        DATE,
    ReserveBudget        FLOAT,
    Balance              FLOAT,
    Status               NUMBER(1),
```

```
    NoOfWeek            NUMBER,
    CommencementDate    DATE,
    Subject_code REF available_sub_typ,
    Budgetfor REF Tutor_typ);

CREATE TABLE Budget OF Budget_typ(
    PRIMARY KEY (BudgetID),
    SCOPE FOR (Budgetfor) IS Tutor));
```

3.1.3 Transformation of many-to-many Association

The only many-to-many association in the object diagram in Fig. 3 is between OfferedSubject and Tutor. One subject could have one or more tutor, and one tutor may be assigned tutoring for more than one subject. For this mapping, we create a new table call Taught_by to associate these two classes.

```
CREATE TABLE Taught_by (
   tutor_ref REF Tutor_typ SCOPE IS Tutor,
   subject_ref REF Available_Sub_typ SCOPE IS Offered_Subject);
```

3.2 Mapping Inheritance

The inheritance hierarchy in object diagram shown in Fig. 3 has class Users as a superclass and Supervisor, OfficeStaff, and Tutor as subclasses. A user can only be a Tutor, Supervisor or OfficeStaff and it must belong to one of these three types of user. To map this relationship, we create three base tables for Tutor, Supervisor, OfficeStaff and User. The Tutor, Supervisor, and OfficeStaff class will be related to its superclass class using UserID as foreign key. The User class will contain user_type attribute to keep track which type of user it belongs to. Beside that user_type attribute is also constraint to valid user type and not NULL conditions. The following is object types and tables for the superclass (Users) and one of its subclasses only (e.g. Supervisor).

```
CREATE OR REPLACE TYPE Users_typ AS OBJECT (
   FirstName     VARCHAR2(50),
   LastName      VARCHAR2(50),
   Email         VARCHAR2(80),
   UserID        VARCHAR2(15),
   UserPassword  VARCHAR2(20),
   Status        NUMBER(1),
   Datecreate    DATE,
   Department    VARCHAR(50),
   User_Type     VARCHAR2(15));

CREATE TABLE Users OF Users_typ (
   PRIMARY KEY   UserID,
   NOT NULL      User_Type,
   CHECK (User_Type IN ('tutor', 'supervisor','officestaff')));

CREATE OR REPLACE TYPE Supervisor_typ AS OBJECT (
   UserID       VARCHAR2(15),
   RoomsNo      VARCHAR2(10));

CREATE TABLE Supervisor OF Supervisor_typ
   (PRIMARY KEY (UserID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
```

```
    ON DELETE CASCADE);
```

3.3 Mapping Aggregation

In the aggregation, we use the aggregation between Claims and ClaimLines as an example. This relationship is created by using a nested table. The choice for nested table is because we must be able to retrieve all ClaimLine by only retrieving Claims object. Furthermore, this show more rigid relationship between ClaimLine is part of a Claim. We must not be allowed to manipulate ClaimLine alone in the application without retrieve through Claims class to where it belongs.

```
CREATE OR REPLACE TYPE ClaimLines_typ AS OBJECT (
    TutoringDate  DATE,
    ClaimLineNo   NUMBER,
    StartTime     DATE,
    EndTime       DATE,
    hours         NUMBER(2,1),
    claim_amount  NUMBER (5,3),
    jobclass_ref  REF JobClassification_typ,
    Location      Varchar2(20));

CREATE OR REPLACE TYPE ClaimLines_tab AS TABLE OF ClaimLines_typ;

CREATE OR REPLACE TYPE Claims_typ AS OBJECT (
    ClaimID      NUMBER,
    Place_by     REF Tutor_typ,
    DateSubmit   DATE,
    DateApprove  DATE,
    Status       NUMBER,
    SubjectCode  REF Available_sub_typ,
    has_claimline ClaimLines_tab);

CREATE TABLE Claims OF Claims_typ(
   PRIMARY KEY (ClaimID),
   SCOPE FOR (place_by) IS Tutor)
   NESTED TABLE has_claimline STORE AS Claimline_Store_tab;
```

3.4 Mapping Multiple Inheritance

To be more complete, we describe how to performance multi inheritance using object relational transformation method.

Fig. 4 shows the multiple inheritance relationship with four object classes, Customer, Commercial, Academic and Private Institution. The object class Customer is a super-class of the Commercial and the Academic class. The Private Institution class inherit from both Academic and Commercial classes.

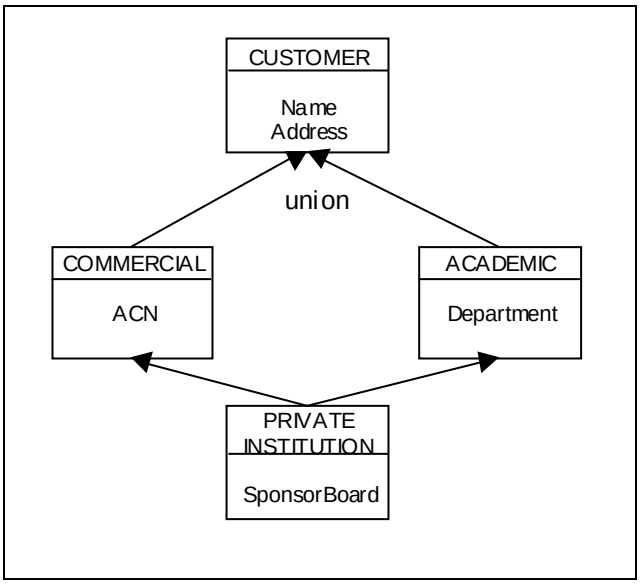


Figure 4. Multiple Inheritance

A *union inheritance* declares that the union of a group of subclasses constitutes the entire membership of the super-class. In the example above, a customer is either a commercial customer or an academic customer. On the other hand, a customer can also be both commercial and academic. To transform the multiple relationship above, we must first transform the union inheritance relationship. Three relational tables are created, that is the Customer table, the Commercial table and the Academic table. Below shows the SQL statements to create these tables.

```
CREATE TABLE CUSTOMER (
      ID      CHAR(10) not null,
      Name    CHAR(30) not null,
      Address CHAR(30), Primary key(ID));

CREATE TABLE COMMERCIAL (
      ID    CHAR(10) not null,
      ACN   CHAR(10),
      Primary key (ID),
      Foreign key (ID) reference CUSTOMER
      On delete cascade
      On update restrict);

CREATE TABLE ACADEMIC (
      ID CHAR(10) not null,
      Department   char(30),
      Primary key (ID),
      Foreign key (ID) reference CUSTOMER
      On delete cascade
      On update restrict);
```

The subclasses Academic and Commercial inherits it superclass Customer through the foreign key reference (i.e. ID). In the second stage we needs to transform the multiple inheritance relationship for Private Institution object. A new table PrivateEducationInstitution is created to represents this relationship.

```
CREATE TABLE PRIVATEEDUCATIONINSTITUTION(
      ID CHAR(10) not null,
      SponsorBoard  char(30),
      Primary key (ID),
      Foreign key (ID) reference CUSTOMER
      On delete cascade
      On update restrict);
```

As noted from the query above, the PrivateEducationInstitution table is related to its two super-classes through the indirect reference of ID in the Customer object. This ID is tightly related the inheritance relationship between private institution object to the Academic object and the Commercial object.

4 Implementation

The tutor claims system was implemented using PHP and Oracle Extended Object Relational features. The implementation combines both server-side scripting and database stored procedure to manipulate the data. For simple query, we use (e.g. insert/update/delete operation that involve one table) PHP script with OCI. However, some process needs to update more than one table. In this case, we create the stored procedure and execute it from PHP page. Handling this operation in a stored procedure makes us easier to maintain referential integrity between tables.

The system can be partition into three parts: (*i*) *Tutor access*, (*ii*) *Office staff access*, and (*iii*) *Supervisor access*. Below shows a use case of each of these three key person roles in our system.

4.1 PHP Scripts Implementation

PHP (Hypertext Preprocessor) (http://www.php.net) is an emerging language server-side and client side script to create web application. In our implementation, we use PHP version 4 server side embedded HTML scripting language to access the Oracle database. The PHP codes are embedded in HTML file through the open tag "<?" and the close tag "?>". The Apache 1.3 is use for the web pages web server, which supports the PHP scripting language by installing the proper dynamic link libraries. The Oracle database is accessed through the Oracle Call Interface (OCI) libraries, which provided along with the Oracle clients software package. Through this interface, the PHP is able to access stored procedure, tables and objects in the Oracle database. Furthermore, we can include SQL statements in web pages inside. These features allow dynamic access and update on the web database. Compared to other scripting languages such as JSP and ASP, PHP language is very similar to C or C++ language and provides many attractive programming language construct which is easy to learn and implements.

To access the Oracle web database, the PHP code must create a database handle. The database handle consists of a few individual oracle structures. These include firstly the server-handle, which holds the connection to the oracle-server, secondly, the session handle which carries the authenticated user/password and lastly service-context. Creating handle is the most expensive operation and care must be taken to determine the proper settings. The code to establish a database handle is as below.

```
<?
 $handle=OCILogon ("username","password","servername");
 if ($handle == false){
   echo OCIError($connection);
   die();
 }
?>
```

In the codes above, the **OCILogon()** function is called with three parameters, the username, the password and the Oracle server name to create a database handle. A persistent handle can also be created using **OCIPLogon()** function with the same set of parameters. The handle will be used as a parameter to execute the SQL statement or calling Oracle stored procedure. An example of executing a SQL statement using PHP script using the created database handle is as below.

```
<?
...
$SQLstmt="Select    c.classificationCode,    c.Description    from
JobClassification_tab c order by c.classificationcode";
$pstmt = OCIParse($handle,$SQLstmt);
OCIExecute($pstmt);
...
?>
```

The SQL statement (SQLstmt) is parsed by the Oracle syntax parser and returns a parsed SQL statement (`pstmt`). This parsed statement can be execute using **OCIExecute()** function. By using the PHP scripting language in our implementation, a rich and dynamic web page that access to the tutor claims web database can be achieved.

4.1 Tutor Access

When a tutor login in to the system, the main screen shown in Fig. 5 will be displayed. This screen shows all claims that has made by the tutor. The claim item is divided into two sections, the approved claims and submitted claims. The approved claims sections shows all claims that has been approved, verified and paid, whereas the other sections shows claims that awaiting approval and rejected claims.

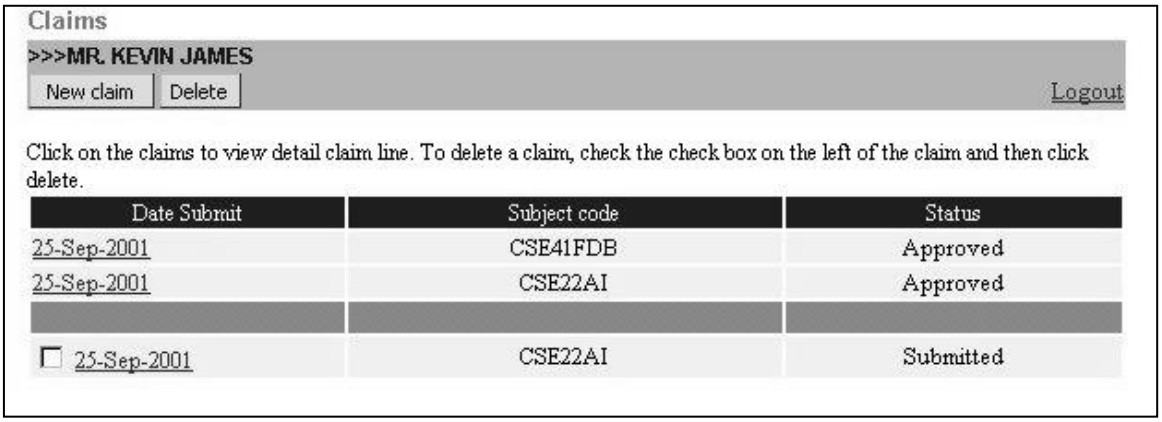


Figure 5. Tutor Claim Main Screen

To create a new claim, click on the **[New Claim]** button. The new claim screen will show as in Fig. 6. The claim subject is centred. That means each claim is for one subject only and submits to only one supervisor to approve it. The steps for adding a new claim line are as follows:

- To make a new claim line, fill in the complete data including, time of tutoring, location of tutoring, start time, end time of that tutoring, tutoring hours, and select Classification Code.
- After fill in the complete data, click **[Add]** to add the current claim line.

- To delete a claim line, check the check box on the left of that claim lines and click **[Delete]**. You can check more than one claim line to delete.
- To add another claim line, go to step 1 above.

After adding in all the claim lines, click **[Submit]** button to submit the claim. A confirmation screen will appear. Click **[Ok]** to go back to main screen.

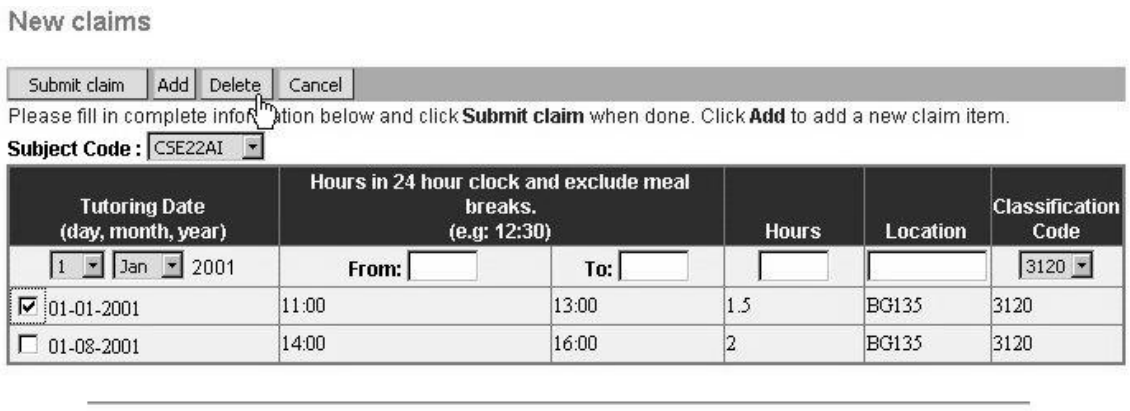


Figure 6. Added New Claim Line

4.2 General Office Staff Access

The navigation bar provides an easy navigation between different functionality for a staff. There are two categories of main task can be performed by a staff. They are the **Tasks** and **Reports** category. Each of the tasks and reports is explained in the following subsections.

4.2.1 Tutor

General office staffs manage the tutor data. They can check for the detail of the existing tutors or create new tutors. When click on the Tutor link on the navigator bar, a list of available tutors is listed. Fig. 7 below shows the tutor list. To check the detail of the tutor, the user need to click the user name, it will link to the selected **Tutor detail** page.

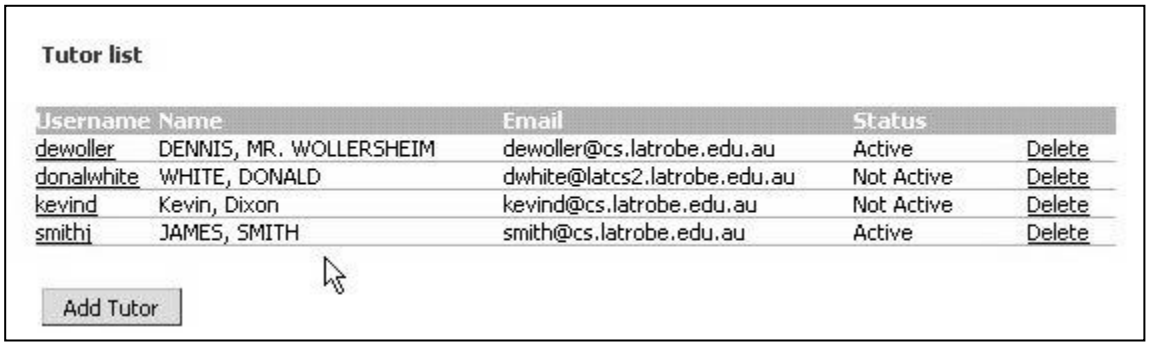


Figure 7. Available Tutor List

4.2.2 Subject

From the Subjects link on the navigation bar, a list of available subjects for the department will appear. This doesn't means the subject is offered for that current semester. The user needs to explicitly choose from these subjects to add in to the offered subject list. Fig. 8 below shows the subject list.

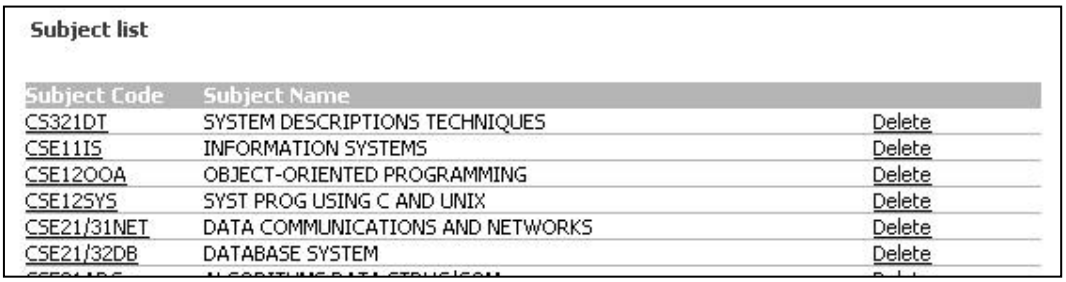


Figure 8. Department Subject List

4.2.3 Semester

The semester link on the navigation bar allows users to add a new semester (period) into the semester list. This information is very important to ensure correct operations of the whole claims system. The semester period is use for many data integrity checking such as valid claims data, checking for submitted, approved, verified, rejected claims and also for the available budget for that semester. There is only one semester can be make current (active) in the system. The *active semester* is the semester/period valid for current claims operation. Fig. 9 shows a list of added semester. The "Semester list" lists the Semester ID, start date, end date and the status of the semester. The *status* column shows the active or not active status of a semester.

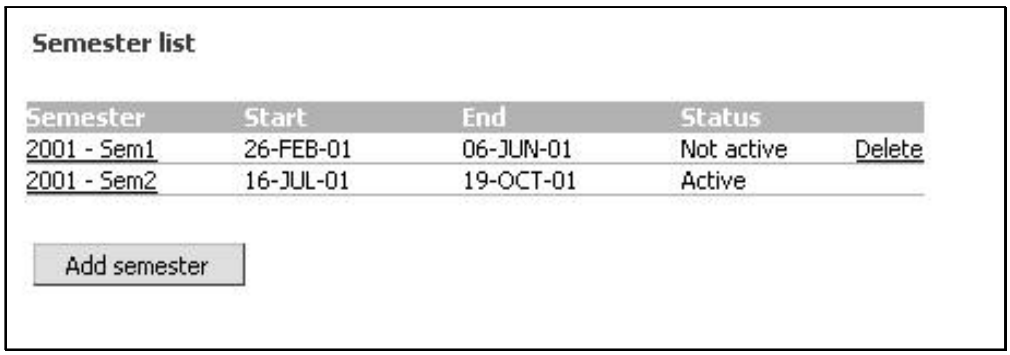


Figure 9. Semester List

4.2.4 Available Subjects

The available subject navigation bar allows users to maintain subject offered for the current semester. The added available subjects will always refer to the current semester activated in the step above. Care must be taken to ensure the current semester is correct before adding any new offered subjects. Fig. 10 shows the offered subject list.

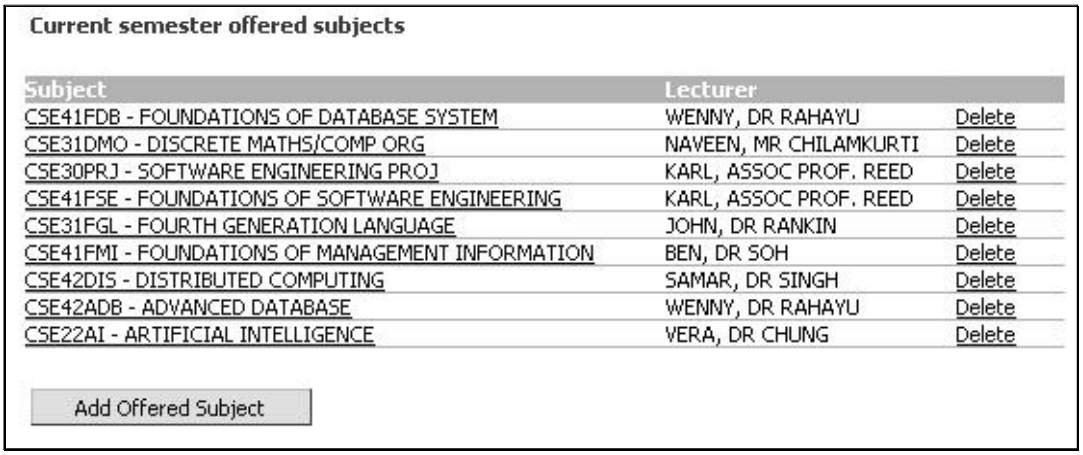


Figure 10. Subjects offered for the current semester

4.2.5 Budget

The Budget functionality allows staff to maintain budget for tutors. Note that, the budget is only valid for the current (active) semester. Before a tutor can be allocated a budget, the status must be activated first. Also the staff must ensure each activated tutor is assigned at least one budget. No more than one same subject should be assigned to a tutor. A list of budget for the current semester is listed in Fig. 11. Each budget has a reference no (e.g. document reference number) for identification.

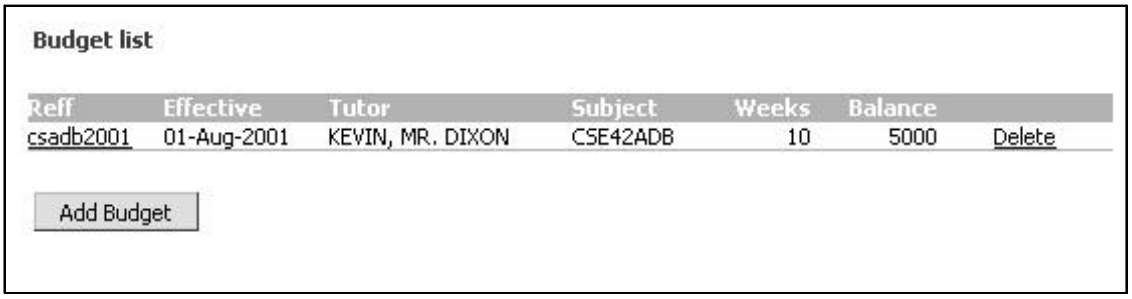


Figure 11. Budget List

To add a new budget to a tutor, click on the [Add Budget] button in Fig. 11. A budget form will show as in Fig. 12. Fill in the Budget reference no (ref), the commencement date, the allocated budget, select tutor for this budget, select a subject and number of weeks for this budget. This screen also allows staff to add activities such as in the budget request form.

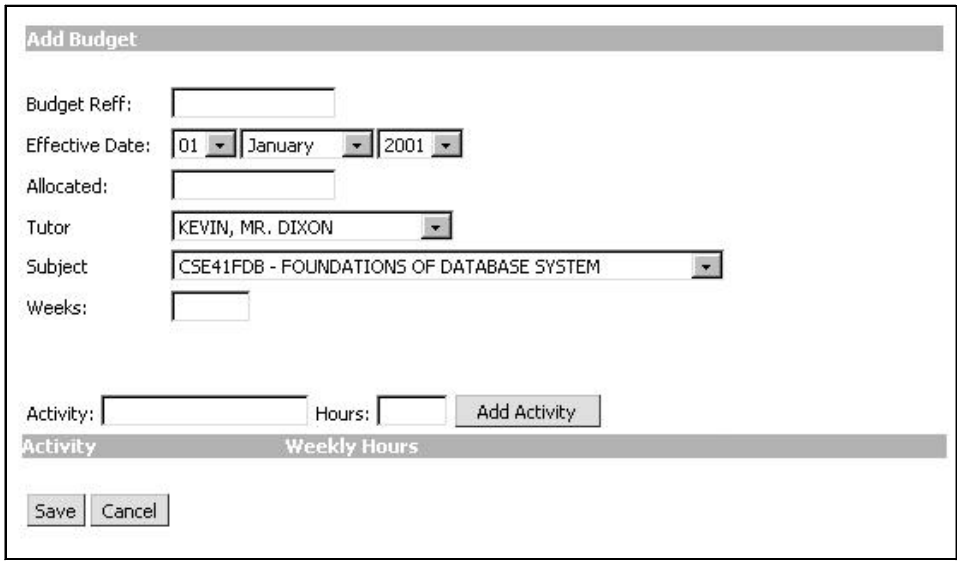


Figure 12. New Budget Form

4.2.6 Claim

The claim navigation link allows a staff to view approved, submitted, rejected and paid claims. The only authorities of a staff for claims are to verify and pay claim. When a claim has been approved by a supervisor, the claims should be verified by the staff and then after the tutor collects the payment, the staff should update the status to Paid/Claimed. Fig. 13 shows a claim detail of a particular tutor.

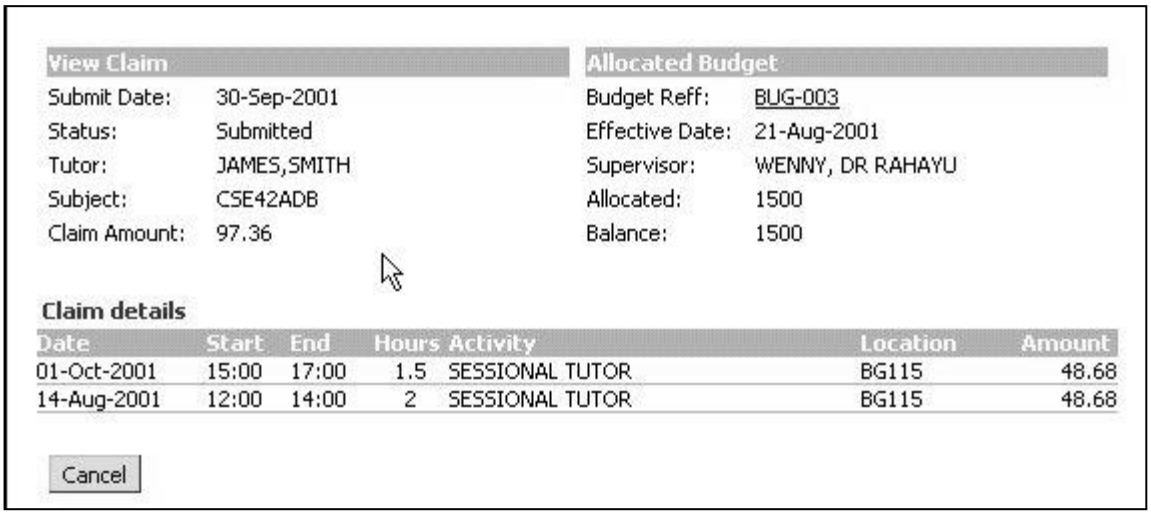


Figure 13. Claim Details

4.2.7 Reports

There are three reports a staff can view/print. The three reports are (*i*) *Tutor Claim Report*, (*ii*) *Balance Report*, and (*iii*) *Claim By Subject Report*. Fig. 14 shows an example of Tutor Balance Report.

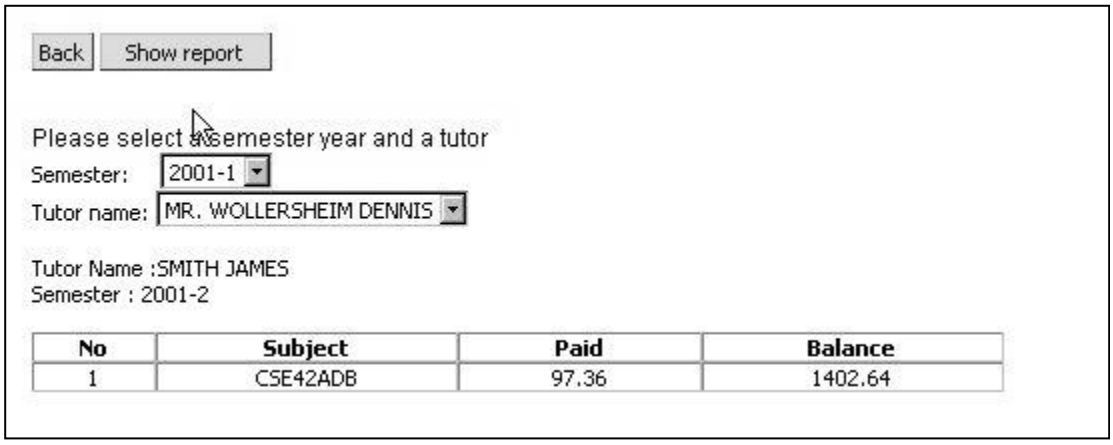


Figure 14. Tutor Balance Report

4.3 Lecturer/Supervisor Access

The lecturer is authorized to approve or reject a claim made by a tutor that taught a subject under his/her authority. The lecturer can also view the approved, rejected and submitted claims. A list of submitted claims is shown in Fig. 15.

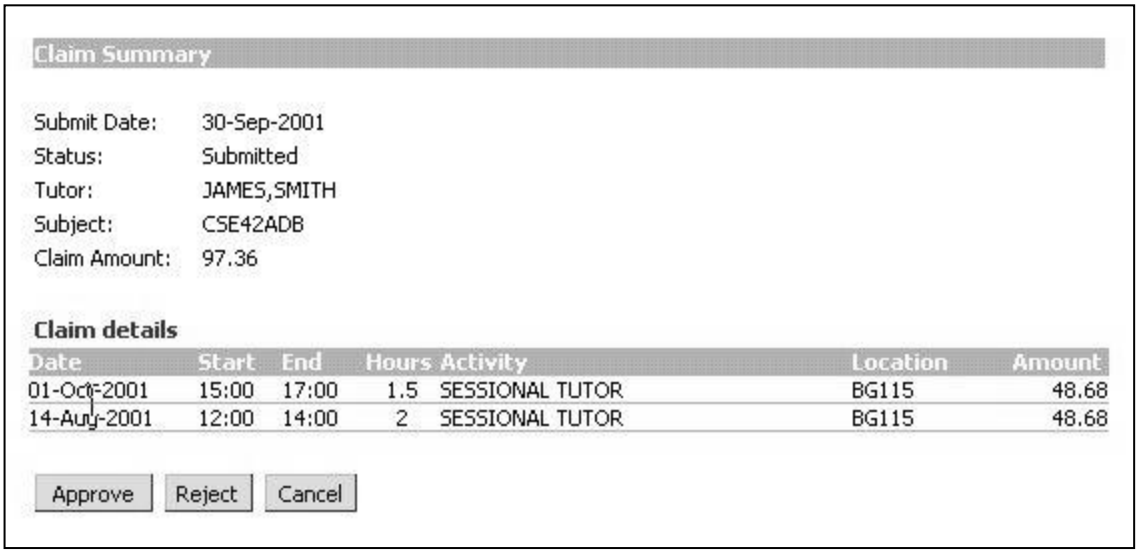


Figure 15. Approve/Reject Claim

5 Evaluation

In this section we describe an evaluation of object-relational features, and problems of object-relational paradigm in current ORDBMS.

5.1 Evaluation of Object-Relational Features

The online claim system uses *Oracle Extended Object Relational* (EOR) features to implement the online claim system. The EOR allow more direct mapping between the modeling and the implementation. Some of the Object Oriented features like inheritance are not directly supported by version of this Oracle. The ability to create custom data type make it easy for us to write stored procedures that directly used that type to define a data type. In RDBMS implementation, we need to define the data type explicitly to use in a stored procedure.

We found out that to make changes to the EOR transformation is much easier than in E-R. For example, we decided to changes the primary key of Tutor class from staffID to UserID (we already discard staffID field), in this case we don't need to change the association relationship between Budget class, Claims class and Taught_by table that reference to Tutor class. This is a result from the usage of ref data type with allow reference to object row id instead of UserID primary key. But in E-R transformation, we needs to change these three tables Foreign key field to reference to this new field.

In performing query through embedded SQL in PHP or through stored procedure, we found out the query is simpler and easier to modify. For example to query join the Claims class Offered Subject class, we only need to retrieve through path expression without a need to perform an explicit join. Below shows an example:

```
Select c.subjectcode.subject_ref.subjectname
From claims c;
```

But, if we only want to retrieve subjectcode from Claims class, using E-R is easier to retrieve and more understandable. The reference value stored in subjectcode reference field in claims class is not human readable, and difficult to debug whether the value stored is correct or valid. The reference value is stored as 16 bytes row object id.

The evaluation of the method used can be summarised in points below:

- Correctness. EOR encourages a strict and controlled way of dealing with errors. Since the system is decomposed into objects, and the interaction of these objects is organized in an orderly fashion, the trace-back mechanism of the errors is not hard to do.
- Completeness. EOR supports all the necessary aspects for web application database. Some of the examples are the ability to support dynamically changes of database schema, authorization, and other aspects that are not available if we are using OO web application, but still have the ability to express different kind of relationship of OO concept [4]
- Efficiency. EOR allows more direct mapping between the modelling and the implementation. Besides, to make changes to the EOR transformation is much easier than in an entity-relationship (ER) for example, because the usage of REF instead of traditional keys. In accordance to efficiency, by using object references, the database will give faster response to online users.
- Scalability. EOR has the ability to easily adapt to growing utilization requirements without the need for a major system redesign and implementation. As it still retains the Relational Database feature, this scheme unlike in pure OO Database allows users to tune system performance by providing a large number of parameters that can be set by administrator easily. The parameters might include the number of memory buffers, etc. [4].

5.2 Problems of Object-Relational Features

Oracle EOR has few limitations that we face in our implementations. We highlight it below.

5.2.1 Dangling Value

If the object to which a REF pointed to have deleted, the REF is left dangling (pointing to a nonexistent object). Oracle provides IS-DANGLING predicate to check whether a REF is dangling. In our implementation, this will result in difficulty to update rows that has references by other object. For example, the tutor has submitted a claim for a subject. If a staff accidentally deleted that subject from the offered subject table, and later realized that it is needed and add the same subject to this table, the subject REF by claims class will become dangling because the REF value points to the old row. This won't happen in RDBMS implementation because the key that referenced by Claims table record stills the same (if the Claims table doesn't put ON DELETE CASCADE constraint, or else that Claims record will be deleted). To solve this problem, there are three ways,

i. Explicitly check for dangling value in Claims table and set the ref value to NULL or delete that record.
ii. Secondly, doesn't allow deletion of an OfferedSubject rows if it is referenced by other Object.
iii. Using REFERENCES keyword to imposed referential integrity. That is to create a referential (like Relational implementation) from subjectcode attribute in Claims to OfferedSubject using Subjectcode REFERENCE OfferedSubject.

The three solutions above are two restricted and not practical. Firstly the solution is visible if we can recreate the link (and we have history of that link). The second solution is too inflexible, but in our opinion is the most useful method to impose integrity constraint. The third solution will violate OO concept, which doesn't allow using traditional ER referencing. It also creates a redundant SCOPE BY constraint.

5.2.2 Does not allow NULL value in nested table column

There are problems with this constraint, considering claim line that is a nested table for claim. If we provide a feature for a tutor to be able to store the claim line without submitting it first, then this constraint will restrict this implementation, because we must ensure all data must be completed before inserting claim line into a claim. What if the tutor forgot the exact hours of tutoring and left that field blank for future modification. The hour NULL value is not allowed in this case. The solution is to use any dummy value to replace the actual value in order to store the claim line into the nested table.

5.2.3 Type Dependency Problem

This is the most problematic problem we face in our development. We can't modify or alter an object that has other object dependent on it (although this problem also found in RDBMS implementation, it is not a trivial problem). If we force it to be altered, the dependant object will become invalid. It is good if that an object can be altered and Oracle is able to recreate the dependency again. In an object-oriented implementation, this is really problematic restriction because there are many dependencies in stored procedure, type or object table. It is also a good idea if oracle is able automatically drops the object dependency if we alter one object.

To solve this problem, we create a set of drop script to drop the whole database schemas and recreate again. The drop sequence must be correct, drop object that other object doesn't dependant on it. Then we must alter the table in SQL script files and recreate all the schemas again. This suggests that if Oracle could generate this type of dependency analysis and create a script for recreate and drop the whole schemas.

6 Conclusions and Future Extension

In this paper, we have described our experience in using an object-relational paradigm in the implementation of an online tutor claim system. The object-relational paradigm that we adopt consists of an object-oriented design and object-relational transformation. We have also evaluated this object-relational paradigm by discussing some problems in the current object features supported by an ORDBMS.

Future extension for this application is divided into two categories: functional and database. In the functional extension, we would like to enhance the system by providing more functions, such as providing a function to add staff/lecturer accounts to the system. Another possible functional extension is to have a direct submission to the finance department for payment. At the moment, the request for budget still needs to be done manually by the lecturer before the office staff can key in tutor information and assign a subject to a tutor. Security is another issue, as HTTP is a very weak protocol for data transmission.

From the database viewpoint, we would like to add more data integrity checking in the database server level. Complete integrity checks on the data should be perform before user submitting new data to store in the database. Our system put minimal effort on this and most of the integrity is performs at client side only. There is a few integrity checking that is important to implement. Firstly, the submission of tutor claims information, secondly the subject, semester and budget information. These data is crucial to ensure correct operating of the system.

References

[1] R.M. Blaha, W.J. Premerlani, and J.E. Rumbaugh., "Relational Database Design Using an Object-Oriented Methodology", *Communications of the ACM*, vol 31, no 4, 1988

[2] P.P. Chen., "The entity-relationship model: Toward a unified view of data". *ACM TODS I*, 1 (Mar. 1976).

[3] R.S. Devarakonda, "Object-Relational Database Systems – The Road Ahead", *ACM Crossroads Student Magazine*, http://www.acm.org/crossroads/xrds7-3/ordbms.html, May 2001.

[4] W. Kim, "*Modern Database Systems*", Addison-Wesley Publishing Company, 1995

[5] W. McCarty, *PHP4: A Beginner's Guide*, ISBN: 0-07-213371-6, McGraw-Hill/Osborne, 2001.

[6] J.W. Rahayu,, E. Chang, T.S. Dillon, and D. Taniar, "Relational Database Implementation of Generic Methods in an Inheritance Hierarchy", *International Journal of Computers and Their Applications*, 2002 (in press).

[7] J.W. Rahayu,, E. Chang, T.S. Dillon, and D. Taniar, "Performance Evaluation of the Object-Relational Transformation Methodology", *Data and Knowledge Engineering Journal*, 2001 (in press).

[8] J.W. Rahayu,, E. Chang, T.S. Dillon, and D. Taniar, "A Methodology for Transforming Inheritance Relationships in an Object-Oriented Conceptual Model to Relational Tables", *Information and Software Technology Journal*, vol 42, no 8, pp. 571-592, 2000.

[9] J.W. Rahayu,, E. Chang, and T.S. Dillon, "Composite Indices as a Mechanism for Transforming Multi-Level Composite Objects into Relational Databases", *The OBJECT Journal (Best of OOIS'98)*, Volume 5, No 1, Hermes Science Publications, 1999.

[10] J.W. Rahayu,, E. Chang, and T.S. Dillon, "Implementation of Object-Oriented Association Relationships in Relational Databases" *Proceedings of the International Database Engineering and Application Symposium*, UK, IEEE Computer Society Press, 1998.

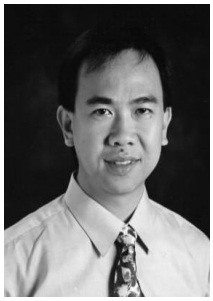
[11] J.W. Rahayu,, E. Chang, and T.S. Dillon, "A Methodology for the design of Relational Databases from Object-oriented Conceptual Models incorporating Collection Types", *Proceedings of the 18th International Conference on Technology of Object-oriented Languages and Systems*, Prentice-Hall, Melbourne, 1995.

[12] J.W. Rahayu and E. Chang, "A Methodology for Transforming an Object-oriented Data Model to a Relational Database", *Proceedings of the 12th International Conference on Technology of Object-oriented Languages and Systems*, Prentice-Hall, Melbourne, 1993.

[13] M. Stonebraker and D. Moore, *Object-Relational DBMSs The Next Great Wave*, Morgan Kaufmann Publisher, 1996.
[14] D. Taniar and J.W. Rahayu (eds.), *Web-Powered Databases*, Idea Group Publishing, 2003.
[15] S. Vesterli and B. Brown, *Oracle Web Applications 101*, ISBN 0-07-213221-3, McGraw-Hill/Osborne, 2001.

Authors Biography

Johanna Wenny Rahayu: received a PhD in Computer Science from La Trobe University, Australia, in 2000. Her thesis, supervised by Professors Elizabeth Chang and Tharam Dillon, was in the area of Object-Relational Database Design and Transformation Methodology. This thesis has been awarded the 2001 Computer Science Association Australia Best PhD Thesis Award. Dr Rahayu is currently a Senior Lecturer at La Trobe University. She has published two books and numerous research articles.



David Taniar received his PhD in Computer Science from Victoria University, Australia, in 1997 under the supervision of Professor Clement Leung. He is currently a Senior Lecturer at the School of Business Systems, Monash University, Australia. His research interests are in the areas of Design and Optimization of Databases, Data Warehousing, and Data Mining. He has published four computing books, and numerous research articles. He is also a Fellow of the Royal Society of Arts, Manufactures and Commerce.

Lee Nung Kion has just completed his Masters studies at La Trobe University, Australia in 2002. Currently he is a tutor at University Malaysia Sarawak. His research interests are in Neural Network Data Mining, Bioinformatics and Parallel Data Mining.

Eric Pardede has just received his Master of Information Technology at La Trobe University, Australia in 2002. At the same university he is now doing his PhD in Computer Science with research topic in Object-Relational Database Design, Transformation, Validation, and Optimization.