INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

Conference Management System
Implemented by Oralce Developer 2000

Wen Yang

A Major Report
In
The Department
Of
Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March 2000

Abstract

Conference Management System

Implemented by Oracle Developer/2000

Wen Yang

Oracle is an extremely powerful and flexible relational database system (RDBMS), it provides various tools for application development and for performing administrative functions, such as SQL*Plus, PL/SQL, Net8, Developer/2000, Designer/2000 and so on. This project mainly covers a brief introduction to Oracle Form 45 and Report 25, which are parts of the Developer/2000 suite of development tools. These products can all be used individually or as an integrated application development platform. Although each has a separate purpose, there are many similarities that enable one to create consistent and powerful applications. Oracle Forms is a powerful application-development tool for building client/Server database applicaions, that are portable to a variety of GUI and character mode platform. The another Developer/2000 tool called Oracle Reports allows the developer to create sophisticated reports in a variety of layouts and contains many customization features. Oracle Developer/2000 tools are optimized to take compile advantage of powerful features in the oracle server.

The another part of this project is an example application named Conference Management System implemented using Oracle Database Management System. The interfaces were built with Oracle Forms and Reports. It designed to be a complete solution for a conference management and easy to use. It demonstrates a prototype for an application coded using Oralce Developer/2000 and PL/SQL. It runs on Oracle7 server release 7.3.2.3.0, SQL*Plus release 3.3.2.0.0 and PL/SQL 2.3.2.3.0.

Acknowledgements

I would like to express my sincere gratitude to Dr. Desai for his guidance and patient.

I also would like to take this opportiunity to thank my friend L. Chen for her help in providing me many useful material and all other friends and colleagues for their generous support, encoragement and assitance in this project.

Contents

1 Overview **1**

2 Starting the Oracle Forms **2**

2.1 Introduction to Oracle Forms 2

2.2 Starting the Oracle Forms Designer 3

2.2.1 Start the Oracle Forms Designer 3

2.2.2 Connect to ORACLE from the Designer 3

2.3 Introduction to Object Navigator 5

2.3.1 Expanding or Collapsing a Entry 5

2.3.2 Changing the Name of an Object 5

2.3.3 Creating and Deleting Objects 6

2.3.4 Moving, Copying and Cutting Objects 6

3 Creating Oracle Forms and Writing Code **8**

3.1 Creating a Form with Single Block 8

3.1.1 Create a Form Module 8

3.1.2 Create a Base Table Block 9

3.1.3 The Layout Editor 14

3.1.4 Set Object Properties in Properties Window 15
3.1.5 Save, Generate and Run the Form 15
3.2 Creating a Master-Detail Form 20
3.2.1 Create a Master Block 21
3.2.2 Create a Detail Block 21
3.2.3 Creating Complex Master-Detail Relationship 24
3.3 Using PL/SQL to Write Event Trigger 26
3.3.1 Create triggers 26
3.3.2 Write Trigger Code 28
3.4 Writing Program Unit 31
3.4.1 Create a Program Unit 31
3.4.2 Edit a Program Unit 35
3.4.3 Delete a Program Unit 35

4 Creating Menus, Alerts and LOVs **36**

4.1 Creating a Menu 36
4.1.1 Create a Menu Module 36
4.1.2 Create Menus and Menu Items 37
4.1.3 Assign Commands to Menu Items 39
4.1.4 Save and Generate a Menu Module 40
4.1.5 Attach a Menu Module to a Form 40
4.2 Creating Alerts 41
4.2.1 Create an Alert 41
4.2.2 Display an Alert 42
4.3 Creating List of Values (LOVs) 43

4.3.1 Create a Static Record Group 43
4.3.2 Create a Query Record Group 45
4.3.3 Create a LOV 45
4.3.4 Attach an LOV to an Item 47

5 Starting the Oracle Reports **49**
5.1 Introduction to Reports 49
5.2 Starting the Oracle Report Designer 50
5.2.1 Start Report Designer 50
5.2.2 Connect to Database 50
5.3 Introduction to Object Navigator 51

6 Creating and Modifing Reports **52**
6.1 Creating a Single Table Report 52
6.1.1 Specify the Data Model 53
6.1.2 Specify the Default Layout 54
6.1.3 Save, Generate and Run the Report 57
6.1.4 Create Computation to a Report 58
6.1.5 Create Page Number Stamps 60
6.2 Creating a Master-Detail Report 61
6.2.1 Specify the Master-Detail Data Model 62
6.2.2 Link the Queries 63
6.2.3 Create Summaries 64
6.2.4 Specify the Master-Detail Layout 65
6.2.5 Save, Generate and Run the Master-Detail Report 65

6.3 Modify the Report in Layout Editor 66
6.3.1 Edit, Format Text 66
6.3.2 Create a Page Break 67

7 Conference Management System **68**
7.1 Introduction 68
7.2 System Analysis 69
7.2.1 Developing an ER model for CMS 71
7.2.2 Converting the ER Diagram into a Relational Database Schema 72
7.2.3 Normalization 73
7.3 Table Creation 73
7.4 Functions Implemented and The User Interfaces 83
7.5 Source Code 106

8 Summary and Future Work **117**
8.1 Summary 117
8.2 Future Work 118

A Bibliography **120**

Chapter 1

Overview

This project is designed to be both a user's guide for Oracle Developer 2000 Forms and Reports and an example application developed by Oracle Forms and Reports. Chapter 1 is an introducation for this report.

Chapter 2 introduces Oracle Forms' Navigator and how to start it.

Chapter 3 discusses how to create a Form and describes some of the major features of PL/SQL.

Chapter 4 covers menus, alerts and LOVs.

Chapter 5 introduces Oracle Reports' Navigator and how to start it.

Chapter 6 describes how to create Reports and modify Reports.

Chapter 7 contains an example application named Conference Management System.

Chapter 8 contains summary of this report and future work.

Chapter 2

Starting the Oracle Forms

2.1 Introduction to Oracle Forms

Oracle Form is a powerful application-development tool for building client-server database applications. Oracle Forms is part of Developer 2000. When you build applications with Oracle Forms, three components will be involved:

1. Oracle Forms Designer is an application development enviroment which includes a set of visual tools that allow you to create objects, set their properties, and write code for your applications.

2. Oracle Forms Generator is used to generate application files to create executable runfiles for runtime deployment. Generating a form module compiles all of its code objects and create an .FMX runfile.

3. Oracle Forms Runform is the runtime engine that form operators use to run a finished Oracle Forms application.

2.2 Starting the Oracle Forms Designer

Note: This section assumes that the ORACLE database and Oracle Forms have been installed on your system.

2.2.1 Start the Oracle Forms Designer

Entering following command at unix prompt, Oracle Forms opens a new form module for you automatically.

```
%f45desm &
```

The & is the ampersand character. The Object Navigator for Forms will appear with default module named MODULE1(See figure. 1).

To open an existing module by selecting **File→Open**, choose the file name you want open.

2.2.2 Connect to ORACLE from the Designer

1. Select **File → Connect...**, Datebase connect window opens(See figure.2).

2. Enter a valid username, password. Leave the database field to blank (press the tab key to move between the fields).

3. Choose **Connect**.

When you connect to the database successfully, the status line in the bottom of Object Navigator(See figure. 1) displays the connect lamp <con>.

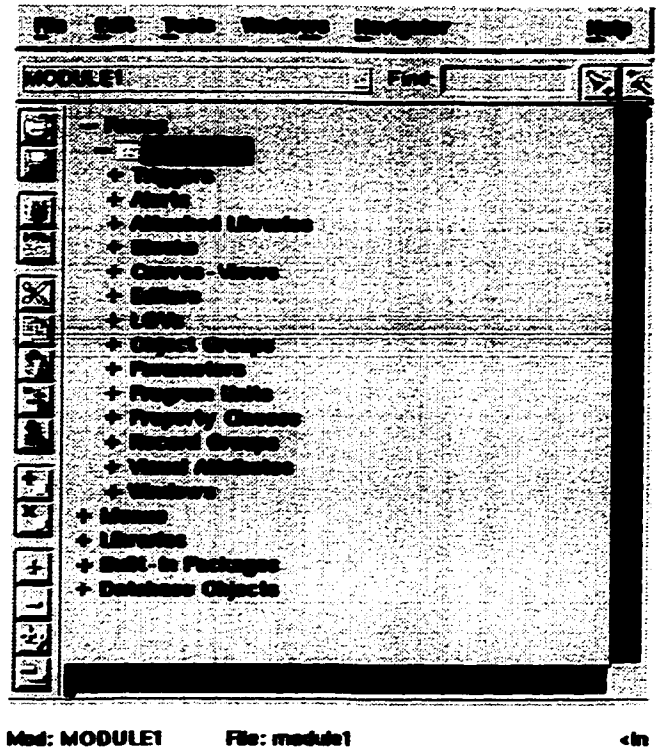


Figure 1: Object Navigator

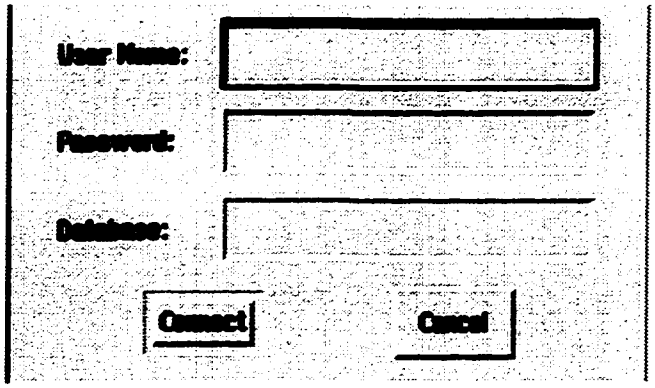


Figure 2: Connect Window

2.3 Introduction to Object Navigator

The Object Navigator provides a hierarchical display of the objects in all open modules. The top-level nodes include Forms, Menus, Libraries, Built-in Packages and DataBase Object. Objects are grouped under the appropriate node. For example, all of the windows defined in a form module appear under the windows node. The window node, in turn, appears under the appropriate form module object, see figure 1. All other nodes, and the objects they contain, are indented to indicate that they belong to these higher-level nodes.

Objects and nodes in the Navigator are displayed with a + or - symbol to indicate whether they are currently expanded or collapsed. A gray + is displayed for a node that does not yet contain lower-level objects.

Each object in the Navigator is displayed with an icon that indicates its type. For some objects, double-clicking the icon invokes an editor appropriate to that type of object.

2.3.1 Expanding or Collapsing a Entry

You can expand or collapse a Navigator entry by clicking the + or - symbol in front of it. Click the + or - symbol to expand or collapse objects at the level immediately below the level of the current object. Shift+click to Expand or Collapse All, meaning that they expand or collapse entries at every level that is below the current entry.

2.3.2 Changing the Name of an Object

Clicking on the default name, MODULE1, the cursor changes to an I-beam, type the name specified, then press **Return**.

The another way is double-click the module object icon in the Object Navigator to display the properties window. Type the new name in the Name Property, and press

Return to accept the name change.

2.3.3 Creating and Deleting Objects

To create an object in the Navigator, select either a node or an object of the type you want to create and choose **Navigator→Create**, or click on **Create** button (see figure.3) on the tool palette on the left hand side of Object Navigator:

Figure 3: Create Button

When a node does not yet contain any objects and so is displayed with a grayed +, you can create an object of that type by double-click the +.

To delete an object in the Navigator, select it and choose **Navigator→Delete**, or, click on **Delete** button (see figure.4) on the tool palette:

Figure 4: Delete Button

Note: When the current selection is a node, rather than an object, deleting remove all objects under that node. When the current selection is a module, deleting closes that module.

2.3.4 Moving, Copying and Cutting Objects

- To move an object, click and hold on the icon beside it, drag it to the desired destination and then release it. Or, choose **Edit→Cut**, move the cursor to the desired destination, then choose **Edit→Paste**.

- To copy the current selection, hold Control button and drag the selection to the desired location. Or, choose **Edit→Copy**, move the cursor to the desired destination, then choose **Edit→Paste**.
- To cut the current selection, choose **Edit→Cut** or click the **Delete** button on the tool palette
- To cut or copy all objects within a node, position the cursor on the node, then choose **Edit→Cut** or **Edit→Copy**.

Note Restrictions on Copy/Move Destinations:

1. Objects can be copied from one module to another, but cannot be moved from one module to another.

2. Objects must be dragged or pasted to the appropriate location in the module hierarchy.

3. Only objects of the same type can be moved as part of a multiple selection.

Chapter 3

Creating Oracle Forms and Writing Code

3.1 Creating a Form with Single Block

At first, start the Oracle Forms Designer and connect to the database. A form module must be open. We assume two tables named SUPPLIER and ORDERS with the attributes given below had been created:

SUPPLIER (Sid, Sname, Saddress, Sphone)

ORDERS (Order#, Sid, Odate, Ostatus, Oprice, Opayment)

3.1.1 Create a Form Module

When you start the Forms designer, the Object Navigator displayed and automatically creates a default Form called MODULE1, see figure.1.

You can build an application by modifying MODULE1, or create a new form module in the Object Navigator, then change the name of Form to a new name, (e.g.

ORDER).

3.1.2 Create a Base Table Block

In general, a *block* on a form corresponds to a *table* in the database. For this example, only one block will be created for the ORDERS table.

To invoke the New Block window, select **Tools→New Block...** from menu to display the New Block window. Or, put the cursor on the block item, then click the **create** button (figure.3) in the tool palette. A New Block window with four tabs appears(See figure.5):

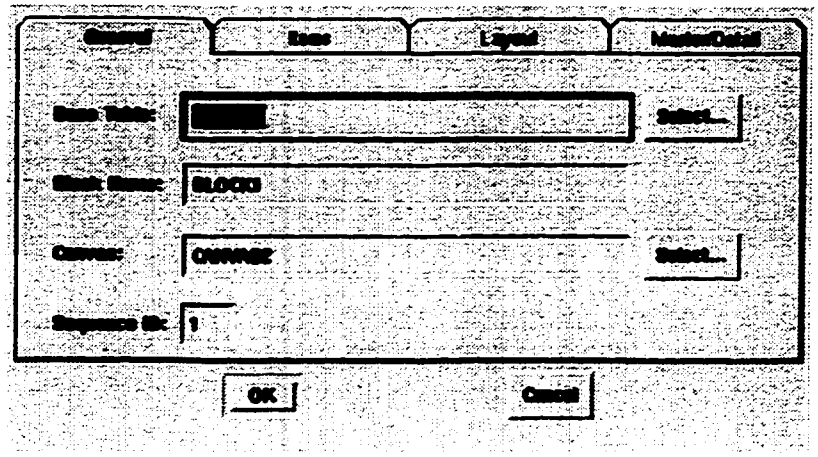


Figure 5: New Block Window

For this case, you will build an order-entry application and would create a base table block to correspond to ORDERS table. When you create the base table block, Oracle Forms also creates default text items for each column in the base table.

The General Tab in the New Block Window

In the Base Table field, enter the name of the table, ORDERS, or click on the **Select** button to the right of the Base Table field, the dialog window(See figure.6) will pop

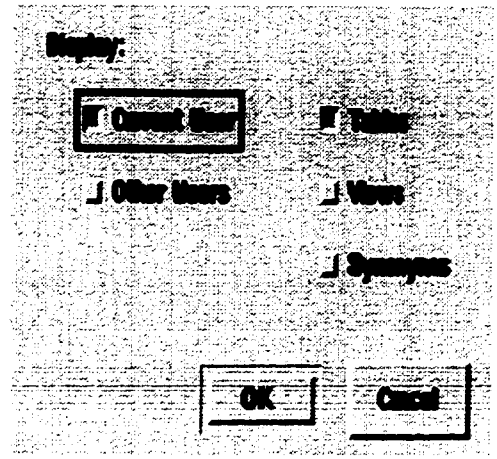


Figure 6: Dialog Window

up. Make sure the **Current User** and **Tables** buttons are selected and click on the OK button. A list of names of tables in database will appear(See figure.7), double-click the table name, **ORDERS**.

In the **Block Name** field, enter name for the block, **ORD_BLK**, or accept the default name. The block name is an internal descriptor that is not displayed at runtime. In the **Canvas** field, type the name of the canvas-view, **CAN_ORD**, or accept default name, on which you want to place the items that will be created in the block. The general tab as figure 8.

The Items Tab in the New Block Window

Clicking on the **Items** tab on the New Block window. Choose **Select Columns...** to display the names of all columns of ORDERS.

If you want Oracle Forms to create an item for a column, that column must be included. Included columns are displayed with a + (plus), excluded columns with - (minus). To include or exclude a column, select the column in the list, then set the **Include** option **On** or **Off**. By double-clicking the column names to include or exclude them.

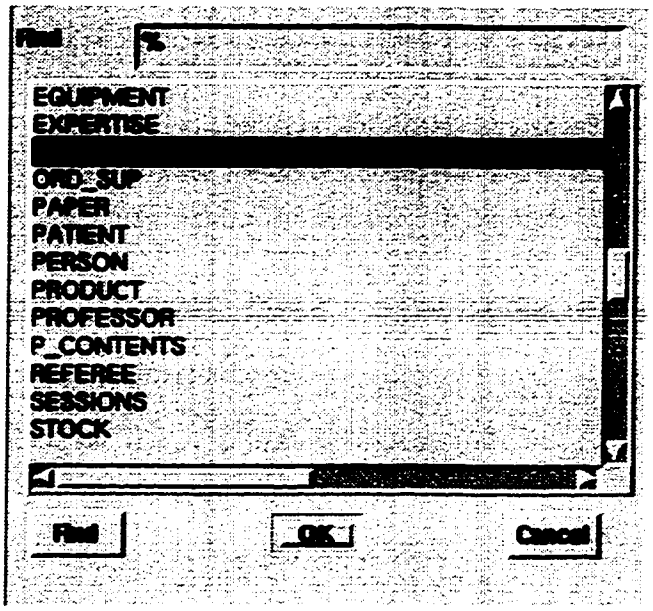


Figure 7: List Table Window

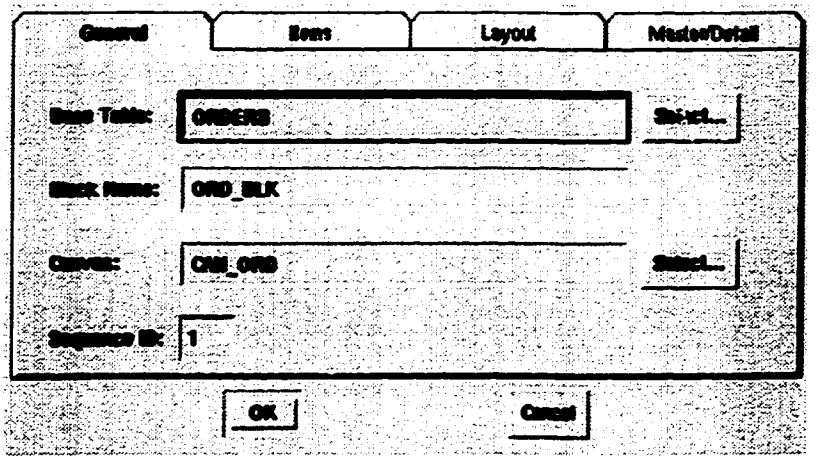


Figure 8: The General tab in New Block Window

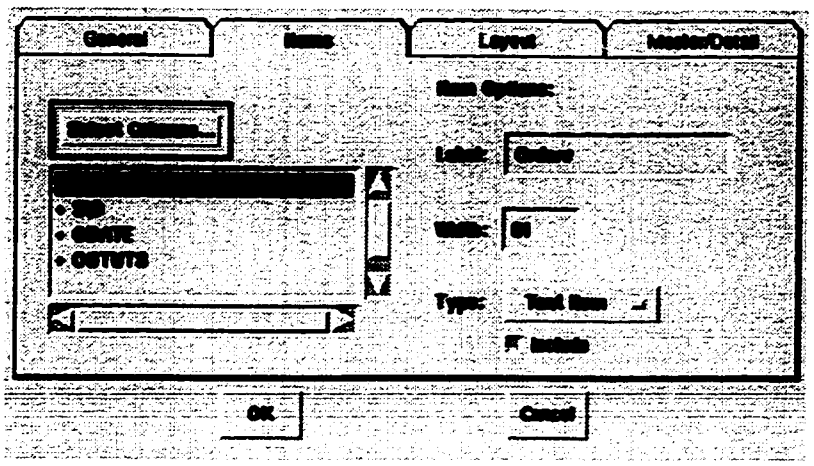


Figure 9: The Item Tab in the New Block Window

(Optional) For each column, select an appropriate column label and width for the form field. For this case leave each field as **Type:** **`Text Item`**.

The Layout Tab in the New Block Window

Clicking on the **Layout** tab on the New Block window. The Layout window allows you to specify the default layout and other options for a base table block.

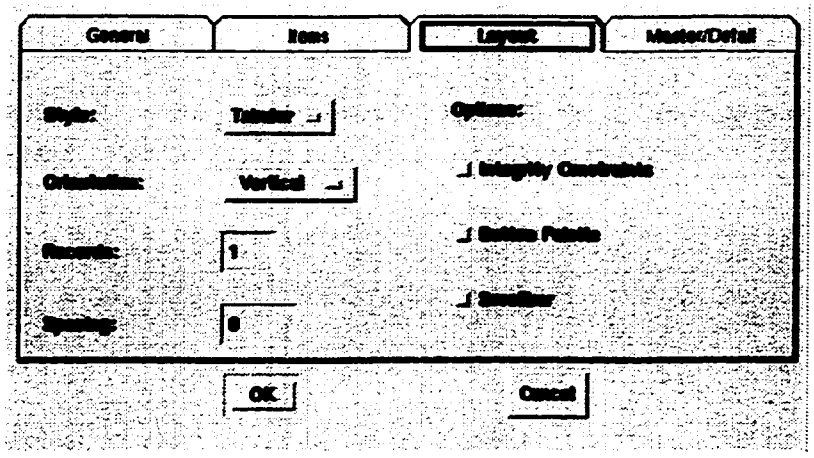


Figure 10: The Layout Tab in the New Block Window

Style

Tabular – When Orientation is set to `Vertical`, Oracle Forms places all items next to each other across a single row, with boilerplate labels above each item. When Orientation is set to `Horizontal`, Oracle Forms places items in a single column, one above the other, with a boilerpate label to the left of each item.

Form – Oracle Forms attempts to place the items in a two-column format, with boilerplate text labels positioned to the left of each item. In general, using Tabular for multi-record blocks, Form for single-record blocks.

Orientation Specify how multiple records should be displayed.

Vertical – Arrangement places consecutive records in a downward direction. Typically, the fields are next to one another in columns.

Horizontal – Arrangement places consective records off to the right hand side.

Record Indicates the number of records to display at one time. For tabular style, more than one record can be displayed.

Spacing The distance between consecutive records. 0 spacing places consecutive records next to each other.

Intergrity Constraints Enforces integrity constraints declared in the database. An example of this will be given in the section on the Master/Detail Forms.

It is also possible to include a `Button Palette` and/or a `Scrollbar` by selecting those options.

Finishing Up the New Block

After specifing all options for the new block, click on the `OK` button to create the block.

3.1.3 The Layout Editor

To view the actual form and its blocks and fields, double-click the object icon for the Canvas-Views, **CAN_ORD**, in the Object Navigator. Or press the right mouse button in the Object Navigator to display a pop up menu, then choose **Layout Editor**. You also select **Layout Editor** from **Tools** in menu bar. Figure 11 is Layout Editor with ORDERS block.

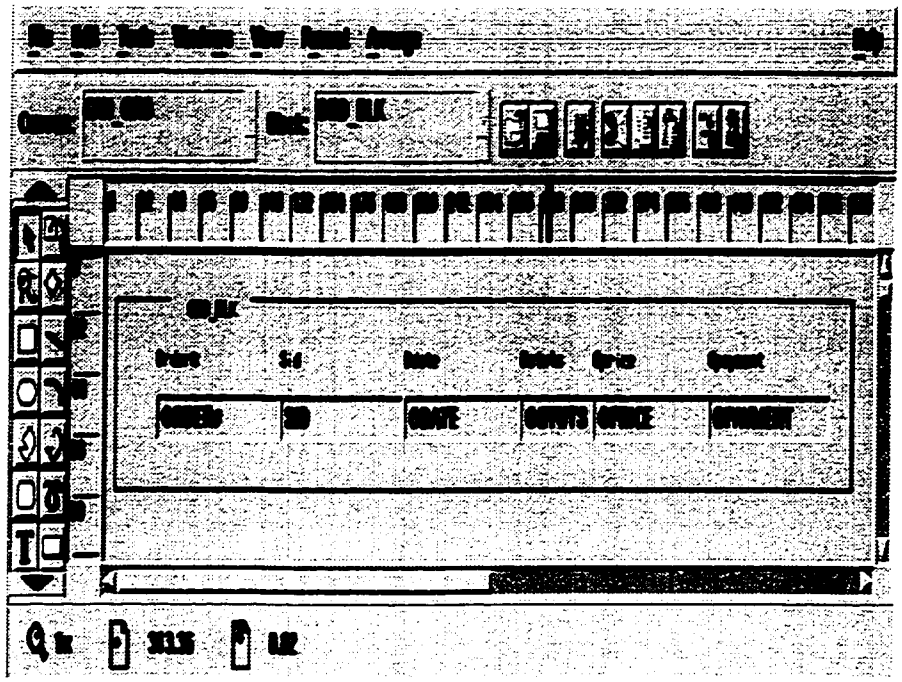


Figure 11: The Layout Editor with ORDERS Block

The Layout Editor is a graphical design facility for creating and arranging inteface items and boilerplate text and graphics. In the Layout Editor, fields and labels can be moved around by clicking and dragging the item with left button. Other text, lines and boxes etc., can be added using the tool palette on the left side of the window.

- To change the text of a label choose the **text** button from the tool palette and then click on a label. To stop editing the label, click anywhere outside of the label text.

- To add text to the block choose the **text** button on the tool palette, click on a open area, type the new text. To change the font, highlight the text, click the **Font** item from **Format** menu and select size.
- To change the text color of a label, use the pointer tool to highlight a label and then click on the **Text Color** button to choose the color.
- To change the width of a label of a field by clicking on the field, drag one of the handles(small black boxes around the field) to re-size the field.

3.1.4 Set Object Properties in Properties Window

The properties window displays the properties of the object you select. You can use the properties window to review and change its properties.

To invoke the properties window, double-click on an object in the Layout Editor to display its properties, or click on an object with the right mouse button, choose **Properties** option from pop-up menu. Figure 12 shows the properties for the ORDER# item.

3.1.5 Save, Generate and Run the Form

Forms can be saved in files in a directory of your choice. The 'source code' for a form is stored in file with a **.fmb** extension. Compiled forms have a **.fmx** extension.

Save the Form

In the Object Navigator, click on ORDER to make it the current object, and then choose **File→Save**. In the **Save** Dialog Box(See figure.13), enter the name **order.fmb** in **Save** field, then click **OK** to save the file.

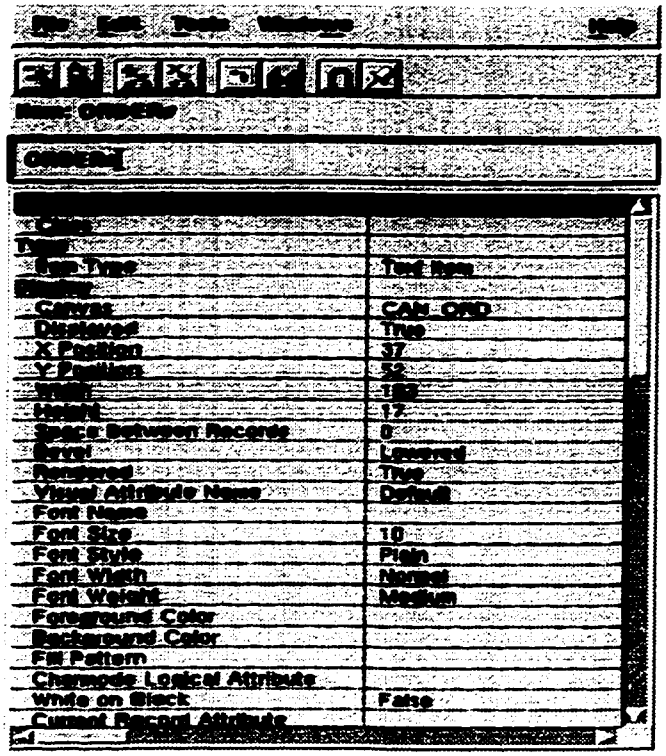


Figure 12: Properties for Order# item

Compile/Generate a Form

Before a form can be *executed*, it must be *Compiled* and *Generated*. Compiling runs a PL/SQL compiler on all of the procedures and triggers in the form. Generating create the .fmx file that is then used to run the form.

To compile a form, first make sure the form is saved. Select the **`Compile...`** option from the **`File`** menu.

To Generate the .fmx file for a form, select **`File`**→**`Administration`**, choose **`Generate`** from the pull-down menu. If the form generation is successful, *Module generated successfully* will appear on the status bar at the bottom of the screen.

When you generate a form, Oracle Form compiles any uncompiled PL/SQL code it contains. Any compilation errors cause generation to fail, a record of the compilation

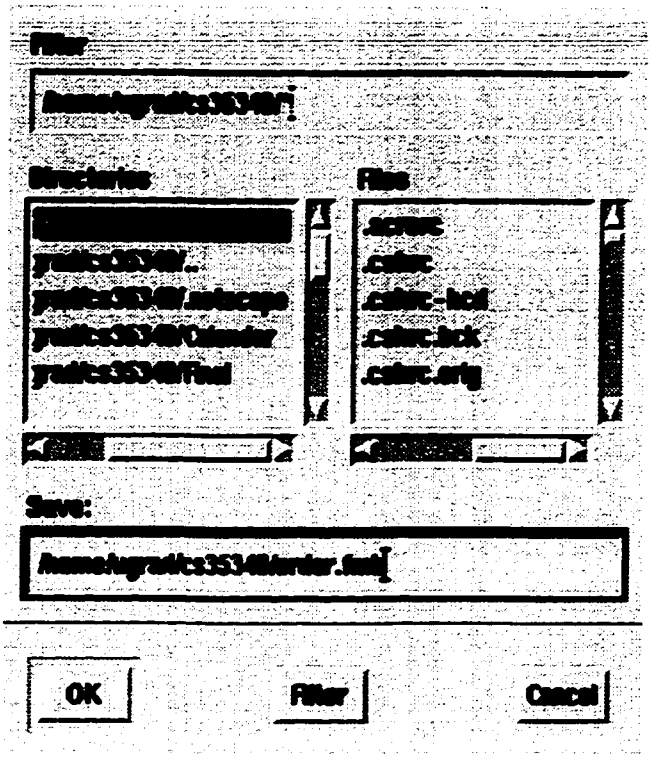


Figure 13: Save Dialog Box

process, including error messages, is kept in a file with a `.err` extension. (For example, if the form is named `order` then the record of the compilation will be stored in `order.err`). When this happens, the generation errors alert is displayed, showing appropriate error messages.

Run a Form

After a form has been saved and compiled, it can be executed. Choose **Run** option from `File` menu, the form will display in a new window(See figure.14).

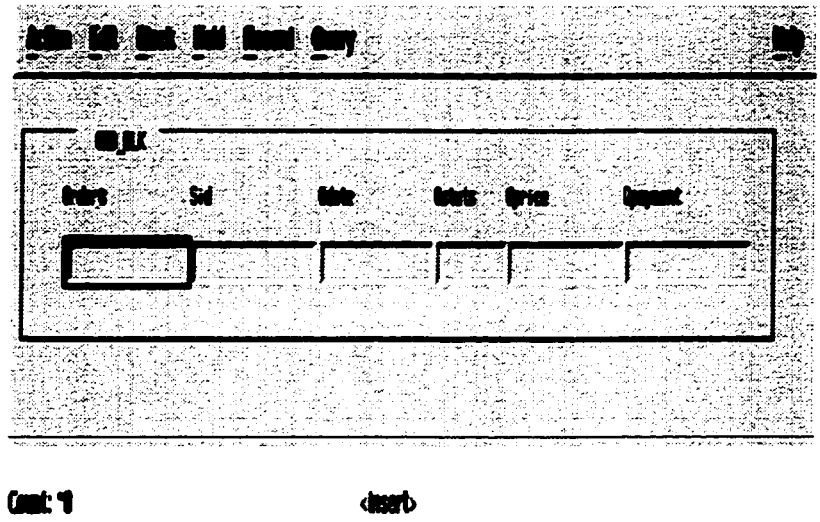


Figure 14: ORD_BLK Form After Running

If you are not already connected to the database, Forms displays an alert, asking if you want to log on to the database. If you are connected successfully, the status line displays the connect lamp <con>.

Using the various menus and items in the form one can query the database, enter new data, change data and save them.

Queries a form using:

1. To retrieve all records

Choose **Query→Execute**, the records are displayed as showing figure 15. Press the Down or Up arrow key, or select **Next, Scroll down or Previous, Scroll Up** from **record** menu to view other records.

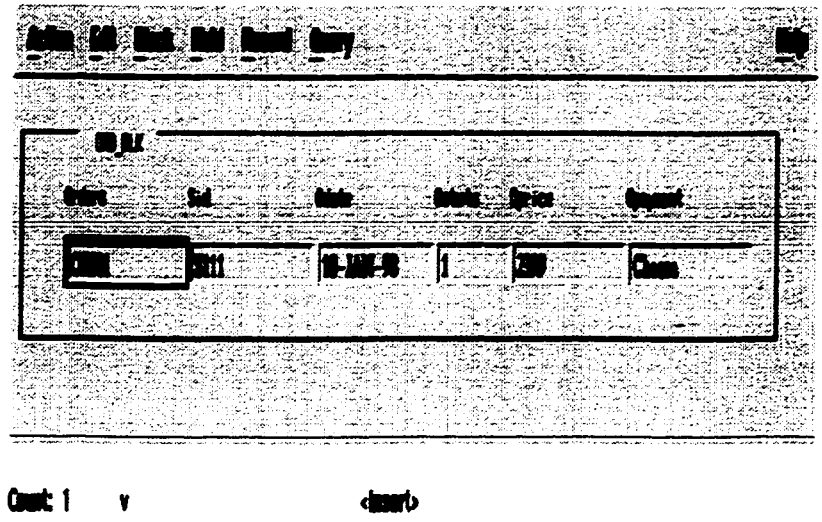


Figure 15: Retrieve All Records.

2. To retrieve a specifc record

Choose **Query→Enter** to specify an item. For example, in the **Order#** item, type an order number, **OR003**, you want to view, then choose **Query→Execute**, the record relating **OR003** will appear.

3. To retrieve a group of records

Choose **Query→Enter**, Specify a item by entering partial value using pattern matching characters:

- _ represents any single character;
- % represents any conbination of characters, including no characters.

For example:

Jon_s matches Jones, Jonas, Jonus

Enter% matches Enter, Enters, Enterprise

_in%s matches Bins, Fines, Winners

To change data in a form, you can press TAB key to the field of interest, and type over the existing data.

To enter new data into the form, you can scroll to the last record and then down once more to move to a blank record. Data can be typed into the fields and TAB key can be pressed to move between fields on the form.

To save both changed and new records on a form, choose **Save** from **Action** menu.

To exit from a running form, pull down the **Action** menu and select the **Exit** menu item.

3.2 Creating a Master-Detail Form

The Master-Detail relationship is a common relationship between entities. In an Entity-Relationship diagram, these are shown as "One to Many" relationship. In a physical database design, a single *Master* record references one or more *details* records in another table. A record in the detail table will relate to exactly one master record in the master table. Another name for this relationship is called parent-child.

A Master-Detail form has two blocks arranged in a master-detail relationship. The first block corresponds to the **master** table and the second block corresponds to the **detail** table. There are two major functions in a Master-Detail form:

1. Oracle Forms coordinates values between the two blocks through a series of form and block level triggers.

2. Oracle forms guarantees that the **detail** block will display only records that are associated with the current record in the **master** block.

3.2.1 Create a Master Block

Using the steps given in the previous section on Creating a Form with a Single Block, create a new block named SUPPLIER that contains all of the columns in the SUPPLIER table.

1. From the **Tools** menu choose **New Block...** option.
2. On the General tab, click on the **Select** button to select the **SUPPLIER** table, change the Block Name to **SUPPLIER_BLK** and Canvas to **CAN_SUPPLIER**.
3. Click on the Items tab and click on **Select Columns...** button to include all of the columns of SUPPLIER.
4. Click on the Layout tab and choose style as **Tabular** and Orientation as **Vertical**. Allow only 1 record to be displayed with spacing of 0.
5. When done, click on the OK button to create the block.
6. Save the form as **sup_ord.fmb** and then **compile** and **run** it to make sure it is working property.

3.2.2 Create a Detail Block

Now that we have the master block SUPPLIER created, we can create the detail block ORDERS and associate it with the master block. Perform the following steps:

1. Choose **New Block...** from the **Tools** menu.
2. On the General tab, click on the **Select...** button to select the **ORDERS** table. Making the Block Name aiso changes to **ORDER_BLK** and the Canvas Name changes to **CAN_SUPPLIER**.

3. Click on the Items tab and click on **Select Columns...** button to include all of the columns.

4. Click on the Layout tab and choose style as **Tabular** and Orientation as **Vertical**. Allow 3 record to be displayed with spacing of 0. Also click on the options for **Integrity Constraints** and for a **Scrollbar**.

5. Click on the **Master/Detail** tab.

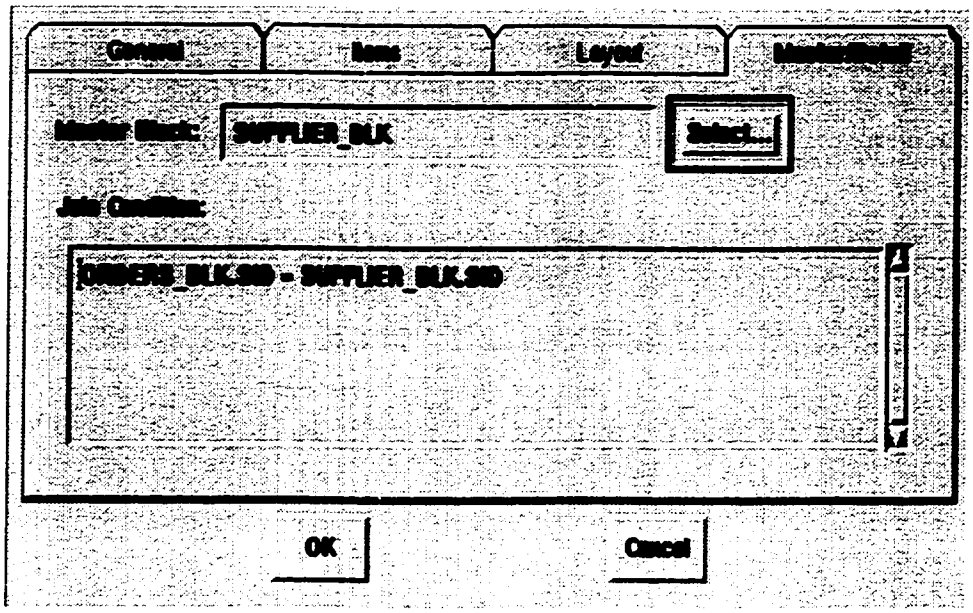


Figure 16: Master-Detail tab in New Block window

The Master/Detail tab in the New Block window (see figure. 16) allows you to define a master-detail relationship between the base table block you are creating (the detail block) and a master block.

In Master Block field, type the name of the master block. The block specified must already exist in the current form. In this example, type in the name of the Master Block, **SUPPLIER_BLK**, or choose it from **Select...** button. Join Condition specifies the join condition that links a detail record to a master record. Type in the Join Condition field:

```
ORDERS_BLK.Sid = SUPPLIER_BLK.Sid
```

When done, click on the **OK** button to create the block.

Save the form (it should have the name **sup_ord.fmb**) and then *compile* and *run* it.

Figure 17 shows the master-detail form running.

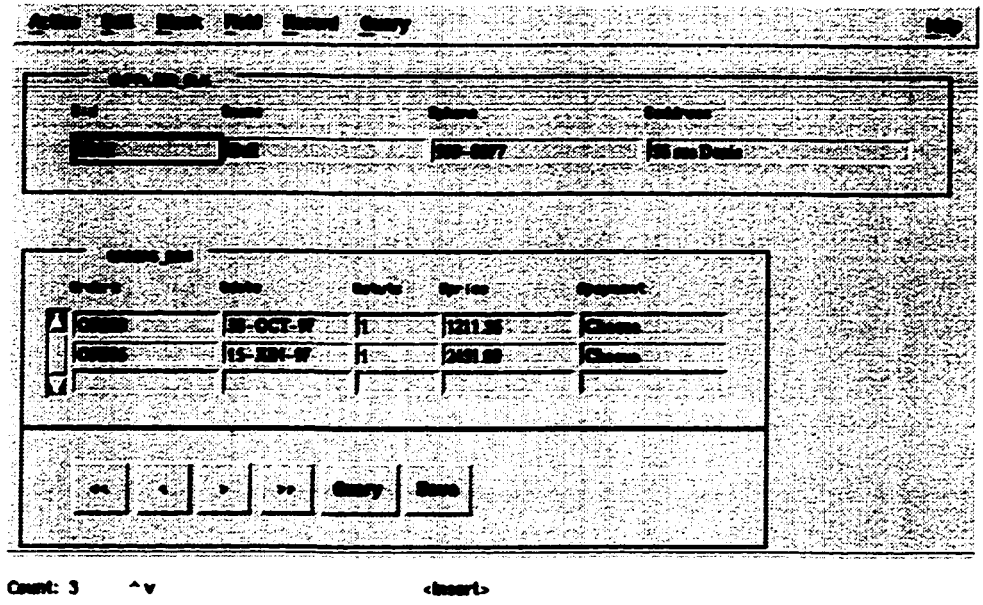


Figure 17: Layout Editor with Maste/Detail

Notice that by scrolling the master block **SUPPLIER** to other supplier numbers (using the up and down arrow keys), the records of **Orders** for that supplier are automatically queried and displayed.

The another way to create a relation is as follows:

In the Object Navigator(figure. 1), insert a relation object under the relations node for the master block (SUPPLIER_BLK). Position the cursor on the Relations node under the master block, then click **Create** button in the tool palette. The relation dialog appears(See figure. 18):

- In the Relation Name field, enter a name of this relation or accept default name.

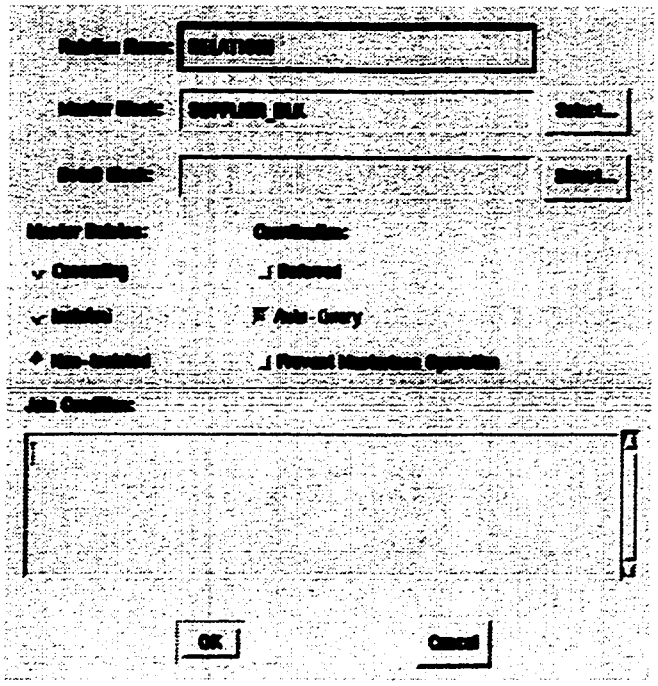


Figure 18: Relationship Dialog window

- In the Master Block field, the name of master block (SUPPLIER_BLK) is entered for you automatically.
- In the Detail Block field, type the name of the detail block, ORDERS_BLK, or choose it from **Select...** button.
- In the Join Condition field, type

```
ORDERS_BLK.Sid = SUPPLIER_BLK.Sid
```

- Choose **OK** to create a master-detail form.

3.2.3 Creating Complex Master-Detail Relationship

Many applications require complex master-detail relationships that involve more than two blocks. To create such relationships, simply defines as many individual relations

as needed. Any block can be the master or detail in more than one relation, and a block that is the master in one relation can be the detail in another.

Consider an application for purchase order, we had created three tables and their relations in the database, **Orders, Supplier, Product**.

One supplier has one or more orders and one or more product. One order belongs to one supplier and has one or more product. One product belongs to one or more orders and suppliers.

We can create them in the form using three ways:

1. Master with Dependent Details:

A master with dependent details relatioship includes a master block and n levels of detail blocks, such that the first detail block is itself a master for its own detail block.

In this way, we make the **Supplier** as master block, **Orders** as detail block for **Supplier** block but as master block for **Product** block, **Product** block as detail block.

2. Master with Independent Details

A master with independent details relationship involves two or more detail blocks, each of which has the same master block.

We can choose one of the three blocks as master block, the other two blocks as detail block. In this example, the master block **Supplier** along with the detail block **Orders** and **Product**.

3. Detail with Two Masters

A detail with two masters relationship involves a single detail block that has two master blocks. Oracle Forms displays the appropriate detail records for whichever master block is the current block in the form.

In the sample application referred to earlier, the **Product** block could be a detail block having two master blocks, **Orders** and **Supplier**.

3.3 Using PL/SQL to Write Event Trigger

PL/SQL Editor Context

Type - Sets editor context to a specific type of code object. When the current module is a form, **Type** can be set to *Trigger or Program Unit.* When the current module is a menu, **Type** can be set to *Menu Startup Code, Menu Item Code, or Program Unit.* When the current module is a library, **Type** is always *Program Unit.*

Object - Sets editor context to a specific object scope.

Name - The Name option lists all of the code objects defined in the current context. Selecting a code object from the Name poplist makes that object the current object in the editor.

The primary method of adding code to a form is through a trigger. Every trigger has a name, and contains one or more PL/SQL statements, it encapsulates PL/SQL code so that it can be associated with an event and executed and maintained as a distinct object.

A trigger must be attached to a specific object in the form, either an item, a block, or the form itself. Trigger fires (i.e. runs) when its event happens. For example, in a form with three buttons, each button has a When-Button-Pressed trigger attached to it, and each trigger contains different PL/SQL statements. When the Button Pressed event occurs, Oracle Forms responds by firing only the When-Button-Pressed trigger attached to the selected button.

Triggers are usually added as the last stage in the form development process.

3.3.1 Create triggers

In the Object Navigator, triggers appear under the **Triggers** node for the object to which they are attached.

- Start the Object Navigator(See figure. 1).

- Expand the object (i.e. SUPPLIER_BLK) under the item by clicking on its + sign, the trigger node appears.
- Select **triggers**, click the **Create** button, new triggers list window opens(See figure. 19).

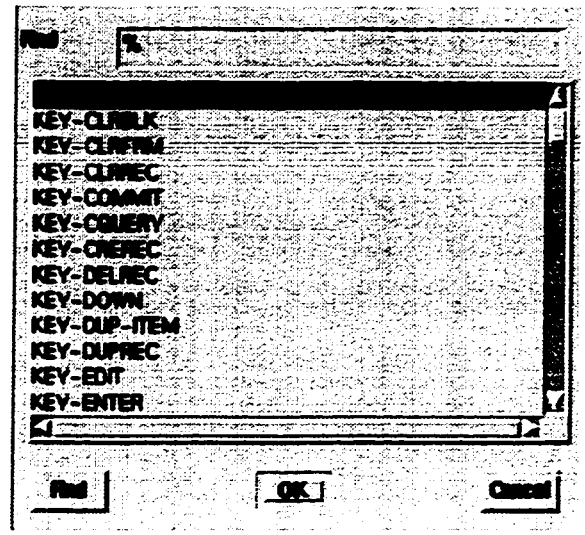


Figure 19: New Trigger List Window

- Enter a search string in find field then press **Find** button to find the trigger name.(i.e. WHEN-BUTTON-DOWN)
- Choose the trigger name WHEN-BUTTON-DOWN from the new trigger list window, click **OK**, PL/SQL editor opens(See figure. 20).
- Write the trigger code in the trigger code window, press the **Compile** button to compile the trigger.
- When done, click **Close** button to dismiss the editor.

You can create a trigger for an item in the Layout Editor, select the item, press the right button of mouse, from the popup menu, select **Trigger** to invoke the PL/SQL

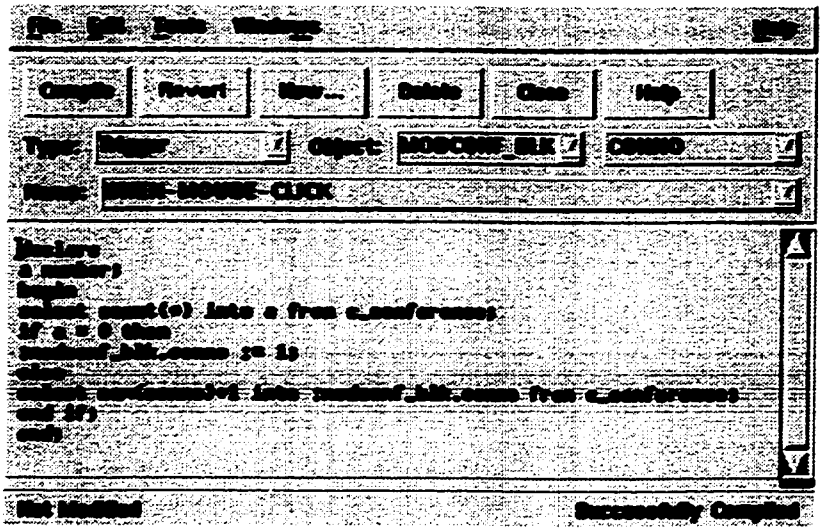


Figure 20: PL/SQL Editor

Editor, choose **New** to invoke the new trigger window, the remainder is same as above way.

3.3.2 Write Trigger Code

The code in Oracle Form triggers is written in oracle's PL/SQL language which is an extension to the SQL database language.

The text of an Oracle Forms trigger is an anonymous PL/SQL block. A block can consist of three section:

1). A declaration section for variables, constants, cursors, and exceptions, which is optional,

2). A section of executable statements, which is required.

3). A section of exception handlers, which is optional.

PL/SQL structure

The syntax for delimiting the sections of a PL/SQL block looks like this:

```
DECLARE
       -- declarative statements (optional)
BEGIN
       -- executable statements (required)
EXCEPTION
       -- exception handlers (optional)
END;
```

In a trigger, only the executable section is required. When you write a trigger that does not have a DECLARE section, you do not need to include the BEGIN and END Keywords, as they are added implicitly. The following example shows such a trigger:

```
/* Key-CLRREC Trigger:     */
IF :System.Record_Status = 'CHANGED' OR
   :System.Record_Status = 'INSERT' THEN
    Commit_Form;
END IF;
Clear_Record;
```

If, however, your trigger has a DECLARE section, you must include the BEGIN and END keywords (see following code) so the compiler can detect the start of the executable section.

```
/*   When-Button-Pressed Trigger:    */
DECLARE
      lov_return  boolean;
BEGIN
      lov_return := show_lov('Order#_Lov');
END;
```

Calling Built-in Subprogram in Triggers

Oracle Forms includes many built-in subprograms that you can call from triggers. A built-in subprogram is a package procedure or function. Built-in subprograms are variable to handle a variety of application functions. In the Object Navigator, you can broswe a list of Oracle Forms built-ins under the Built-in Package nodes.

1. To move the input focus from one item to another, you may call:

NEXT_ITEM, NEXT_BLOCK, NEXT_RECORD, GO_ITEM, GO_BLOCK;

2. To perform operations on data in the form or in the database:

CLEAR_RECORD, DELETE_RECORD, ENTER_QUERY, EXECUTE_QUERY, COMMIT_FORM;

3. To control the display of interface objects at runtime:

MOVE_WINDOW, HIDE_WINDOW, SHOW_EDITOR, SHOW_LOV, SHOW_ALERT.

For example:

/*When-Button-Pressed trigger for Order button*/

Go_Block('ORD_BLK');

When this subprogram is called, the **Orders block** is displayed.

Writing SQL Statement in Triggers

The SQL statements can be used in trigger to perform some operations. These all write in a trigger editor (Figure. 20). To refer to an item in the form use the following notation: block_name.item_name

```
select sum(Oprice)
into :ORD_BLK.totalprice
from orders;
```

To reference the value of an item, preface the item name with a block name and a colon:

table_name.column_name = :block_name.item_name

Compile Triggers

Oracle Forms compiles any uncompiled triggers in a form when you generate the form to create a runtime file, and compiling individual triggers is not required. However, compiling locally allows you to find and correct errors in your trigger code at the time you write the trigger, rather than when you generate the form.

3.4 Writing Program Unit

The program unit is a PL/SQL subprogram that you write and re-use in the form. This section explains how to create this subprogram that can be called from any trigger in that module, it can be a procedure or a function.

3.4.1 Create a Program Unit

In the Object Navigator, select the desired Program Units node and then click **Create** button. The New Program Unit window appears(See figure. 21).

Specify the **Name** of the program unit and then specify the program unit **Type** (either Procedure, Function, Package Spec, or Package Body). Click OK, the PL/SQL editor window opens(see figure. 20). In the PL/SQL editor, write, edit and compile the program unit code. If there are errors displayed in the compilation message pane, click on an error to navigate to its location in the source code and correct it.

If no error appears, click **Close** to dismiss the PL/SQL editor.

Procedure example:

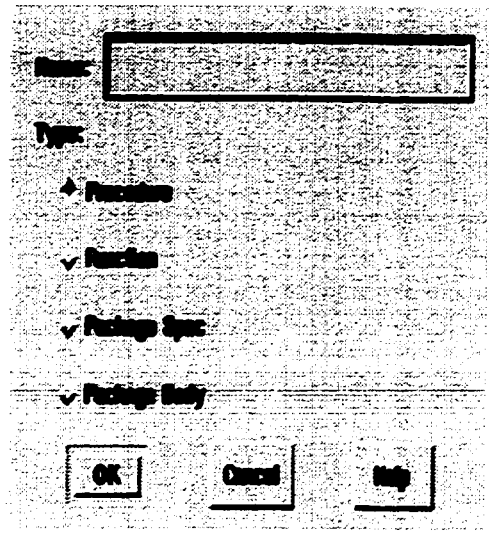


Figure 21: New Program Unit window

The procedure specification begins with the keyword PROCEDURE and ends with the procedure name or with the list of formal parameters. The procedure body begins with the keyword IS and ends with the keyword END.

Syntax:

```
PROCEDURE procedure_name [argumet list] IS
   [local variable declaration]
BEGIN
   statements
   [EXCEPTION exception_handlers]
END;
```

The following procedure processes an order by decrementing the invertory available for a particular product. To call this procedure, you pass in the product ID, the number of units being ordered, and the warehouse that should fill the order:

```
PROCEDURE do_order (units_ordered IN NUMBER, prod_id IN NUMBER,
                    warehouse IN NUMBER) IS
```

```
units_in_stock    NUMBER;
BEGIN
   SELECT amount_in_stock INTO units_in_stock FROM inventory
      WHERE product_id = prod_id AND warehouse_id = warehouse;
   IF units_in_stock >= units_ordered THEN
      UPDATE inventory
      SET amount_in_stock = units_in_stock - units_ordered
          WHERE product_id = prod_id;
   ELSE
      Message('Insufficient stock on hand.')
      Raise Form_Trigger_Failure;
   END IF;
END;
```

You should use this procedure in a trigger as following:

```
do_order (:order_blk.units, :order_blk.prod_id, :warehouse_blk.id);
```

Function example:

Every function must execute a RETURN statement.

Syntax:

```
FUNCTION function_name [argument list] RETURN [type] IS
   [local variable declaration]
BEGIN
   statements
   RETURN (result);
   [EXCEPTION exception_handlers]
   RETURN (result);
END;
```

The following function returns the number of product units in stock at a particular warehouse. When you call this function, you pass in a product ID and warehouse ID:

```
FUNCTION get_invertory (product NUMBER, warehouse NUMBER)
         RETURN NUMBER IS
         amount NUMBER;
BEGIN
   SELECT amount_in_stock INTO amount FROM inventory
   WHERE product_id = product and warehouse_id = warehouse;
   RETURN amount;

EXCEPTION
   WHEN OTHERS THEN
      Message('Invalid product name or warehouse ID.');
      RETURN (-1);
END;
```

You could use this function in a trigger as follows:

```
DECLARE
   inv                    NUMBER;
   amount_required        NUMBER := :order_block.order_item;
BEGIN
   inv := Get_invertory(:product.id, :warehouse.id);
   IF inv < 0 THEN
      ...      -- handle the error here
   ELSE IF inv >= amount_required THEN
      ...      -- process the order here
   END IF;
END;
```

If no error appears, click **Close** to dismiss the PL/SQL editor.

3.4.2 Edit a Program Unit

- In the Object Navigator, double-click the program unit icon you wish to edit.
- In the PL/SQL editor, edit the program unit.
- Click **Compile** to recompile the program unit when you are finished editing.
- Click **Close** to close the PL/SQL editor when you are finished.

3.4.3 Delete a Program Unit

- In the Object Navigator, click the program unit node.
- Click the **Delete** button (Figure. 4) to delete the program unit.
- Click **Yes** to confirm the message box that appears. If the program unit you delete is referenced in other program units, you must remove those references to reflect the deletion.

Chapter 4

Creating Menus, Alerts and LOVs

4.1 Creating a Menu

Every form runs with a menu, either the default menu provided by Oracle Forms, or a custom menu you create using the Menu Editor.

The Menu Editor provides a visual and intuitive way to quickly create a menu skeleton. Once the skeleton is created, you add commands to each menu item, and attach the menu module to a form.

Assume that you want to create a custom menu as in figure. 22.

4.1.1 Create a Menu Module

The first step in designing a custom menu is to create a menu module. In the Object Navigator, double-click on the Menus node, or put the cursor in the menu item, then click on the **Create** button, and change the default name, such as **main_menu**.

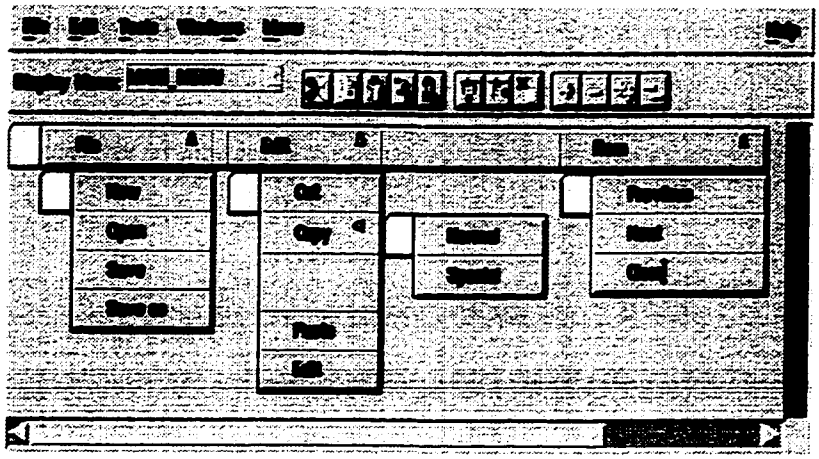


Figure 22: Menu Editor

4.1.2 Create Menus and Menu Items

Create a Main Menu

1. Select **main_menu** node, then choose **Tools→Menus** editor. The Menu Editor(See figure. 23) opens, showing a new menu with one item selected.

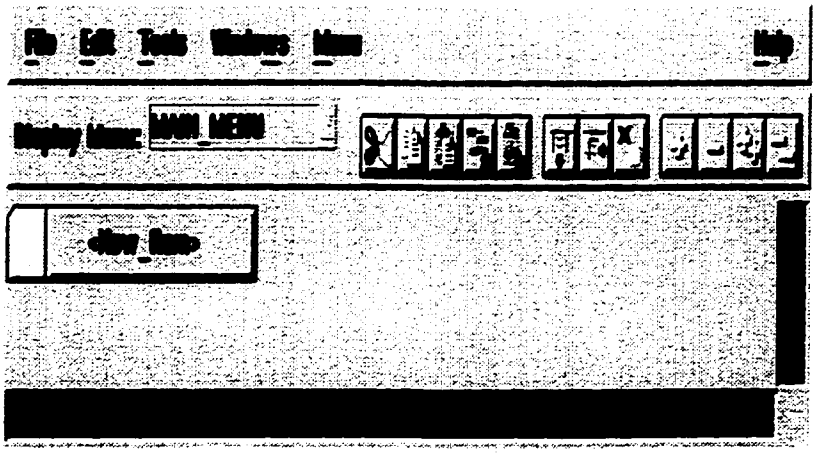


Figure 23: Menu Editor with a New Menu

2. Type **File** to replace the default name.

3. Select **Menu→Create Right** or select the item and press **Create Right** button(Figure. 24) from toolbar, a new item is added to the main menu with the default name **<New Item>**.

Figure 24: Create Right button

4. Type **Edit** to replace the default name.

5. Repeat step 3 and 4 to add the **Item**.

Create Menu Items

Now we'll create menu item under each main menu title.

1. In the menu, click in the **File** menu item.

2. Select **Menu→Create Down** or select the item and click **Create Down** button(Figure. 25),

Figure 25: Create Down Button

   Oracle Form adds a new item below **File**, with the default name **<New Item>**.

3. Type **New** to replace the default name.

4. Use Create Down button to add the **Open, Save and Save as** menu items.

5. Click in **Edit** to select the Edit menu.

6. Use **Create Down** button to add the Cut, Copy, Paste and Edit menu items.

7. Select the **Item** menu, add Previous, Next and Clear menu items by repeating above steps.

Create Submenu Items

Select the parent item on the individual menu, such as **Copy**. Click the **Create Right** button, or press Control-Right arrow to add the first item on a new submenu. Type **Normal** to replace the default label, and then click the **Create Down** button to add the another submenu items, replace the default label with **Special**.

Delete items from a menu or submenu

Select the item you want to delete, choose **Menu→Delete** (or click on the Delete button). Oracle Forms displays an alert asking if you are sure you want to delete, and click **Yes**.

4.1.3 Assign Commands to Menu Items

Every menu item must have a valid **Command Type** property: Null, Menu, PL/SQL etc. Most menu items execute PL/SQL commands, so their Command Type is **PL/SQL**. If a menu item has submenus, its Command Type must be changed to **Menu**. For example, File, Edit and Item must have Command Type property set to **Menu**. Null is used to separator items.

Assigning a command to a menu item, you may select the desired menu item in the Menu Editor(e.g. Save) and choose the **properties...** option from the pop up meun by clicking the right mouse button.

In the Properties window, check that Save's Command Type is set to PL/SQL.

Double-click on the **Command Text** property (or click on the **More...** button) to display the PL/SQL Editor. Type the code in the Editor.

```
DO_KEY('commit_form');
```

Click **Compile** button to compile the code, and finally click **Close** to dismiss the window.

4.1.4 Save and Generate a Menu Module

Once all of the menu structure has been created and commands for each menu item have been specified, the menu module must be saved to a file with a **.mmb** extension and generated.

To save the menu module, select File→Save(or Save As) to save the menu module you just created. Oracle Forms displays the **Save As** dialog, with the default file name **module.mmb**. Replace the default name with **mymenu.mmb** and click OK. Choose File→Exit to close the Designer.

To generate the menu module, pull down the **File** menu from toolbar, click on Administration menu and choose Generate from the submenu. Generating a menu module results in a file with a **.mmx** extension. In this example, the generated menu module is **mymenu.mmx**

4.1.5 Attach a Menu Module to a Form

Before attaching a menu to a form, you must generate the menu module, set the form's Menu Module property, and generate the form.

For example, you want attach **mymenu** module to a form:

At first, make sure **main_menu** is saved and generated. Open that form's property window, set the Menu Module property to the **main_menu**, once a menu is attached

to a form, Oracle Forms automatically loads the **main_menu.mmx** menu file when the form is running.

4.2 Creating Alerts

An alert is modal window that displays a message notifying the operator of some application condition.There are three styles of alerts: **Stop, Caution and Note**. Each style denotes a different level of message severity. Message severity is represented visually by a unique icon that displays in the alert window.

4.2.1 Create an Alert

- In the Navigator, select the Alerts node and then click the **Create** button. Double-click the alert icon to display its property window. Change the **Name** in properties window to a meaningful name, e.g. *no_price_alert.*
- Give the alert window a **Title**, e.g. *Please take note.*
- Set the **Alert Style** to appear in the alert, e.g. *Note.*
- Give a default Message (the actual message will be given later in the trigger).
- Define one or more buttons for the alert by entering a text label in the Button 1, Button 2 and Button 3 fields. (The default text labels are **OK** for Button 1 and **Cancel** for Button 2).

Note: At least one button must have a label. Buttons that do not have label are not displayed. On most window managers, the default button has a distinctive appearence.

4.2.2 Display an Alert

To diaplay an alert, your application must execute the SHOW_ALERT built-in subprogram from a trigger.

SHOW_ALERT is a function that returns a numeric constant.

```
Show\_Alert(alert\_name)
Return NUMBER;
```

- Create a trigger by double-clicking the triggers icon in the object navigator.
- In the PL/SQL editor for the new trigger. Write the PL/SQL code for the trigger.

```
DECLARE
  -- return variable from the function Show_Alert
  alert_button MUMBER;
BEGIN
  IF :item.price IS NOT NULL THEN
    -- call the user-named calculation subprogram
    calculate_totals;
  ELSE
    -- display the alert
    alert_button := Show_Alert('no_price_alert');
     IF alert_button = ALERT_BUTTON1 THEN
       -- invoke the price list window
       Go_Block('product_prices');
     END IF; -- do nothing if operator selects button 2
  END IF;
END;
```

4.3 Creating List of Values (LOVs)

An LOV is a scrollable popup window which shows the acceptable values for the associated text item. For example, when you enter the Sid item in ORDER form (see the example of 4.2.1), it might be difficult to remember all of the sid of suppliers. One solution to this problem is to make a list of appropriate values available when you navigates to Sid item.

In the following section, we will illustrate how to create a LOV and how to attach it to an item.

A LOV is based on a Record Group which is a query that returns a collection of data.

There are two kinds of record group values:

1). **Static record group** – a record group whose values are set at design time. The values are stored in the form, not in the database. It is useful when the values are known in advance and will not to change.

2). **Query record group** – a record group whose values are stored in the database. The values are read from the database at runtime. It is useful when the values are not known in advance and are likely to change.

4.3.1 Create a Static Record Group

Select the **Record Groups** node in Object Navigator, click on the **create** button on the tool palette to popup New Record Group window(See figure. 26).

Select **Static Values** then press **OK**, Columns Specification window opens. Give a column name, specify the data type and length, type the values for that column in the **Column Values** field, eg. Cash, Cheque and Master Card. Click **OK** to create the static record group(See figure. 27).

Open the properties window for the record group to change the name to a meaningful

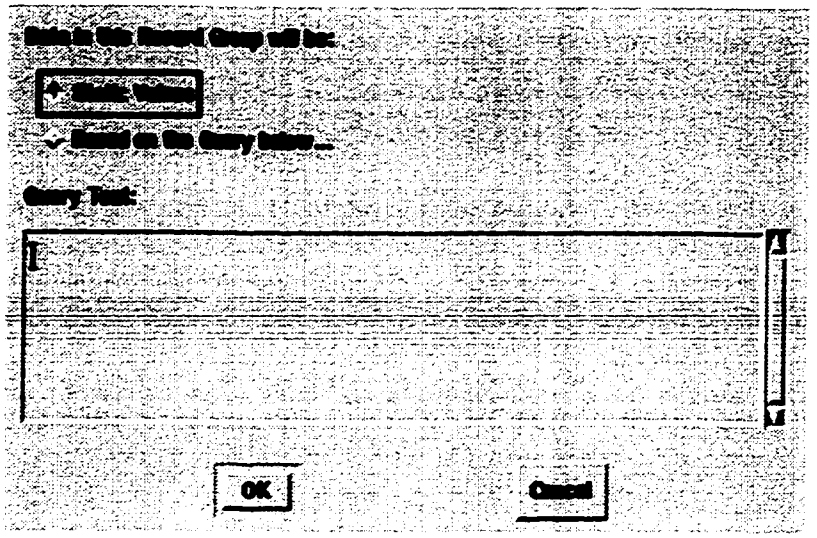


Figure 26: New Group Window

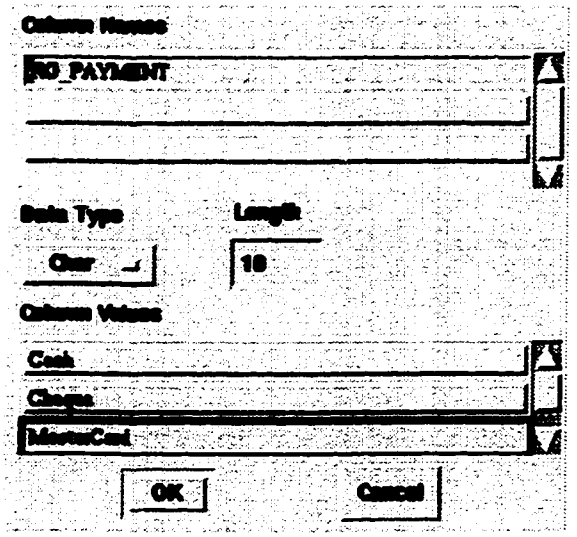


Figure 27: Column Specification Window

name.

4.3.2 Create a Query Record Group

Select the Record Group node in Object Navigator, click on the **create** button to open New Group window(See figure.28), select **Based on the Query below...**, type a statement query to retrieve the data from the database in Query Text field, example as shown in figure 28.

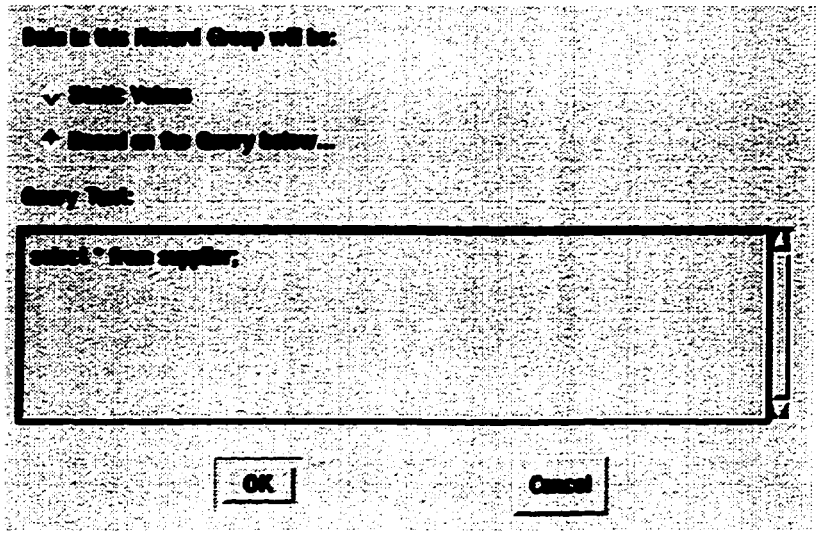


Figure 28: New Record Group window with a query

Click OK to create the query record group. Change its name in the properties window, e.g. RG_SID.

4.3.3 Create a LOV

In the Object Navigator, double click **LOVs** node to open a New LOV window. Select **Existing Record Group** and enter its name or press **Select...** button to choose

one from a list of existing record groups. For example: choose RG_SID, see figure. 29.

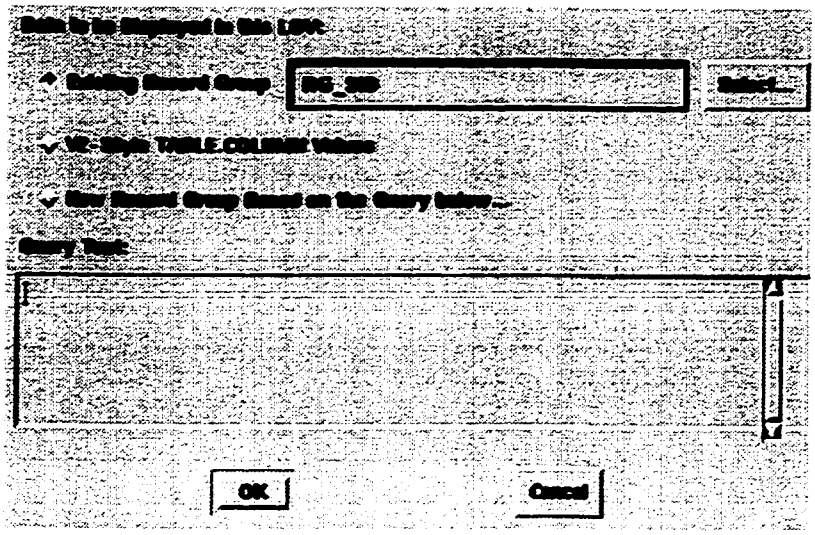


Figure 29: New LOV window with a Exist Record Group

Press **OK** to create a New LOV.

Open properties window, replace its default name with a meaningful name, e.g. LOV_SID, double click on the **Column Mapping** property to open the LOV Column Mapping window(See figure. 30).

In the **Column Names** field, select the columns from the record group to include them in the LOV. In the **Return Item** field, type the form's target item name, e.g. ORD_BLK.SID. Enter the **Column Title** as it should appear at runtime. e.g. Supplier ID. Click **OK** to create the new LOV. If you do not want a column to be returned, leave the **Return Item** blank.

You can also created a LOV which doesn't base on an exist record group, in New LOV window select **New Record Group Based on the Query below...** and type a query in **Query Text** field.

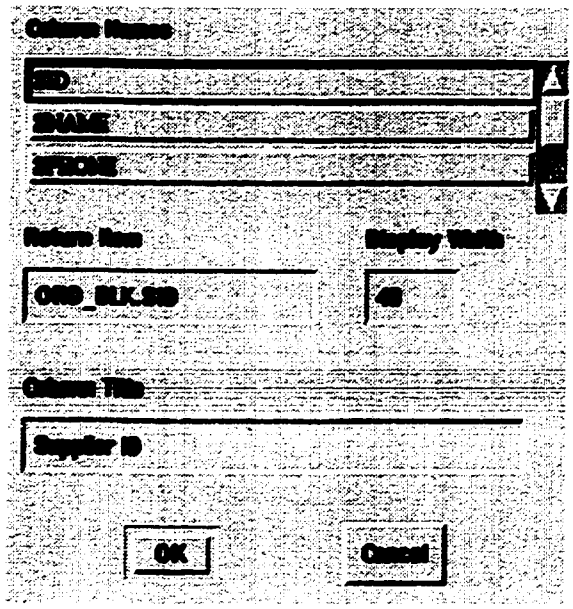


Figure 30: LOV Column Mapping window

4.3.4 Attach an LOV to an Item

After creating a LOV, you attach the LOV to an item. In our example, we attach the LOV_SID to the Sid item of the ORD_BLK. In the Object Navigator, open the properties window for Sid item of the ORD_BLK. Set the **LOV** property by entering the name of the LOV to **LOV_SID**.

(optional) Set the **LOV for validation** property to *True*, so that values from the LOV can be entered in the field.

(optional) Open the properties window for LOV, set the **Auto Display** propety to *True* to display the LOV automatically at runtime.

After attaching the **LOV** to an item, at runtime, Oracle Form displays the <List> lamp on the status line whenever an LOV is available. Once invoked, an **LOV** appears in a small window, allowing you to select a single value. For instance, we attached the **LOV** to Sid item of ORD_BLK previously, in the runtime we can get a window when we invoke the Sid item(See figure. 31).

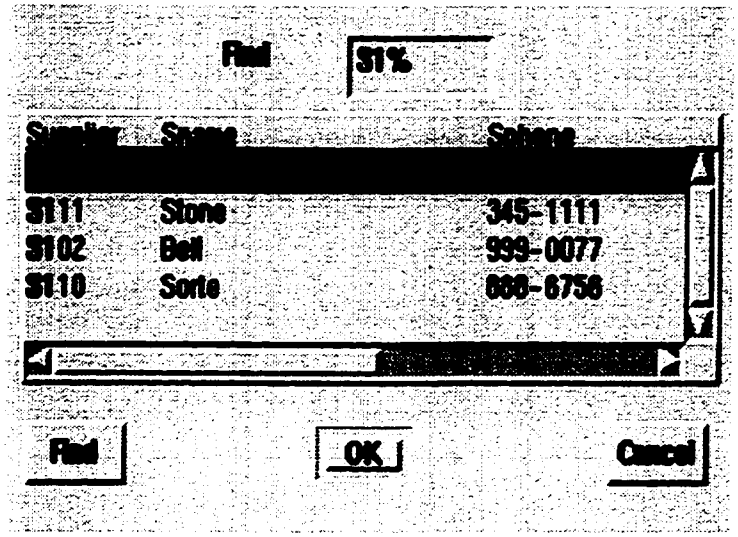


Figure 31: LOV List window

Chapter 5

Starting the Oracle Reports

5.1 Introduction to Reports

Oracle Reports is a tool for developing, displaying, and printing production-quality reports. It is designed for application developers who are familiar with SQL and PL/SQL.

There are four steps to building a report with Oracle Reports:

Create a new report definition. Define the data model (choose the data, data relationships, and calculations you will use to produce the report output). Specify a layout. You can use a default, customizing it if desired, or create your own layout. Run the report, previewing the output in the Previewer.

Report has many kind of style: Tabular, Break, Master/detail, Matrix(crosstab), Nested matrix, Matrix break, Form letter, Mailing label, World Wide Web, Multimedia and OLE2.

5.2 Starting the Oracle Report Designer

5.2.1 Start Report Designer

Enter the following command at the unix prompt:

```
% r25desm &
```

The & is the ampersand character.

The Object Navigator for reports will appear (See figure. 32).

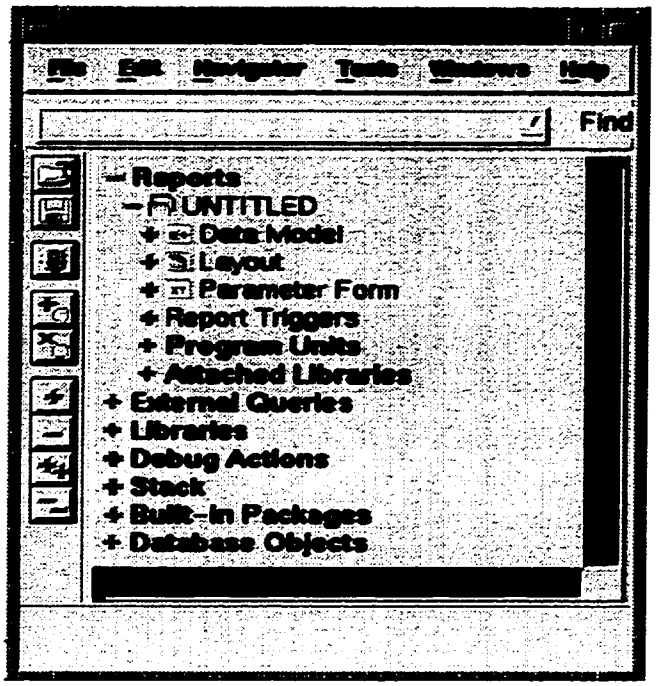


Figure 32: Object Navigator

5.2.2 Connect to Database

As with Oracle Forms, the first step in using Oracle Reports is to Connect to an Oracle server. Choose the `Connect...` from the `File` menu. In the connect dialog

(See figure. 2), fill your Username and Oracle Password, then click OK. If connected successfully, <Con> will be displayed at the bottom of the Object Navigator screen.

5.3 Introduction to Object Navigator

Object Navigator shows all objects that Oracle Reports has created for you, as part of the report definition.

Data Model - in which you define the data for the report.

Layout - in which you create the report layout.

Parameter Form - in which you customize the appearance of the Runtime Parameter Form, a window that optionally appears at runtime and enables you to enter parameter values that affect report execution.

Chapter 6

Creating and Modifing Reports

6.1 Creating a Single Table Report

There are four steps to build a report with Oracle Reports:

1. Specify a data model (choose the data, data relationships, and calculations you will use to produce the report output).
2. Specify a layout. Oracle Reports provides six default layout styles; you can choose one and customize it if desired, or create your own layout.
3. Create or Customize the Parameter Form. All reports have a default parameter form.
4. (optional) Create any triggers or program units that will be executed with the report.

In this section, we will create a report that views the contents of a single table.

6.1.1 Specify the Data Model

The first step is to specify the data model: in the Object Navigator, double-click the Date Model node icon, or highlight the Data Model, click on it with the right mouse button and choose **Editor** from pop-up menu. The Data Model Editor will appear(See figure. 33).

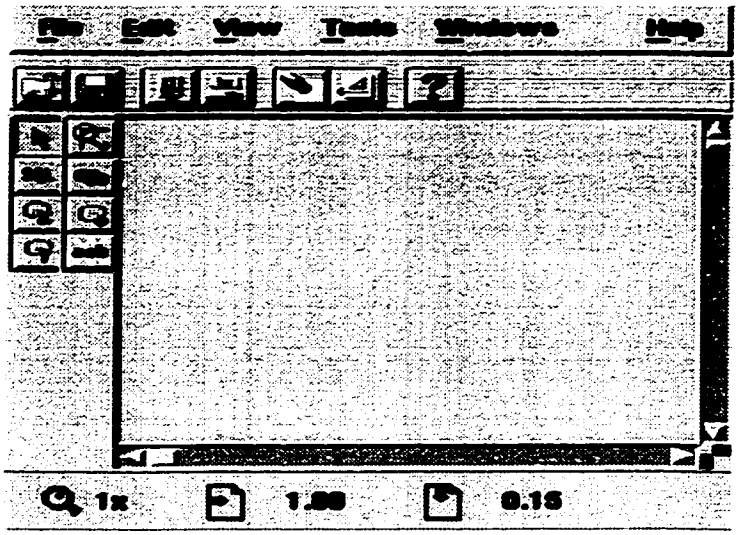


Figure 33: Data Model Editor

In the Data Model Editor, single-click the Query button from tool palette to create a query object(See figure. 34),

Figure 34: Query Button

click in the main area (canvas region) of the window, then property window appears.

Choose **General** tab of the property window, in **Name text** box, for example, replace the default name to **Q_orders** or accept default name. In **SELECT Statement text** box, type a SELECT statement (e.g., **select order#, sid, odate, oprice, opayment from orders order by order#;**).

Note: Select the columns in the order you want them to appear in the report output(See figure. 35).

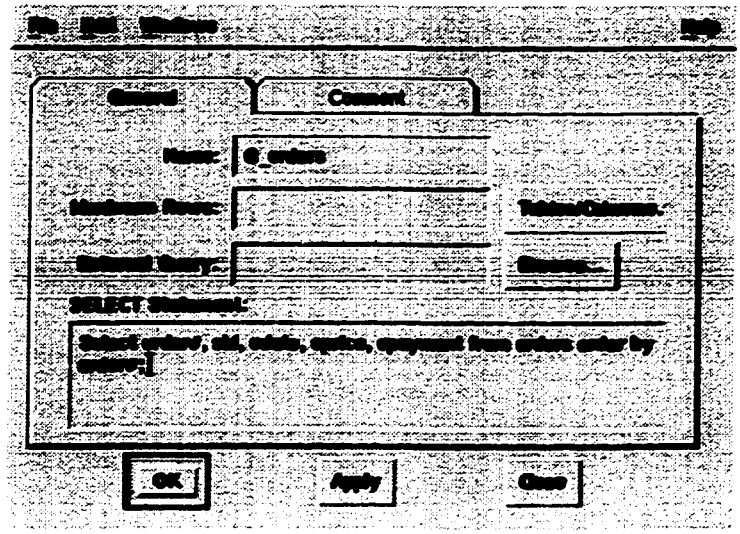


Figure 35: Property Sheet for Q_orders

Click `OK` to display the **`Q_orders`** query and its associated columns below it in a group named **`G_orders`** in the Data Model Editor(See figure. 36).

At any time, the properties for the query, the group and each of the columns will be displayed by clicking on the query or the group with the right mouse button, it produce a pop-up menu with the properties menu item.

Re-sort the order of the columns in a group by clicking and draging the column name to its new position.

6.1.2 Specify the Default Layout

Once the data model has been completed, the layout of the report must be specified.

- Click the default Layout button (See figure. 37)

  in the toolbar at upper of data model window, the default Layout screen will appear(See figure. 38).

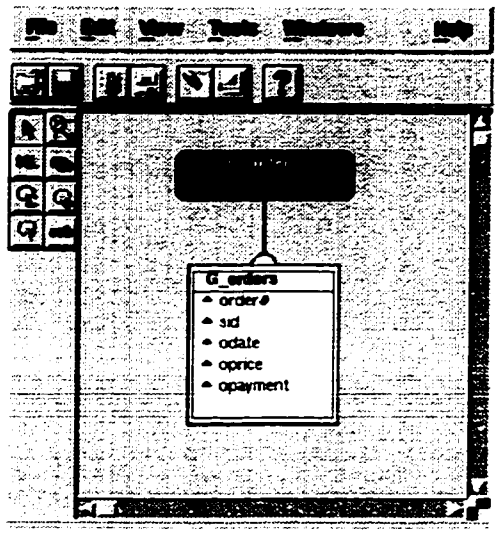


Figure 36: Data Model Editor with Q_orders

Figure 37: Default Layout Button

The six layout choices include:

Tabular – Simple table with column headings at the top and data records in consecutive rows below.

Master/detail – Multiple tabular reports broken up by related collections of data.

Form – Column headings on the left hand side with data values next to them on the right.

Form Letter – Arbitrary placement of data items within a text body.

Mailing Label – No column heading and records grouped into repeating sections sized to print directly to a sheet of mailing labels.

Matrix(crosstab) – Column labels on both the left and the top with the data values in the middle.

- (Optional) In the Data/Selection tab, customize the report output:

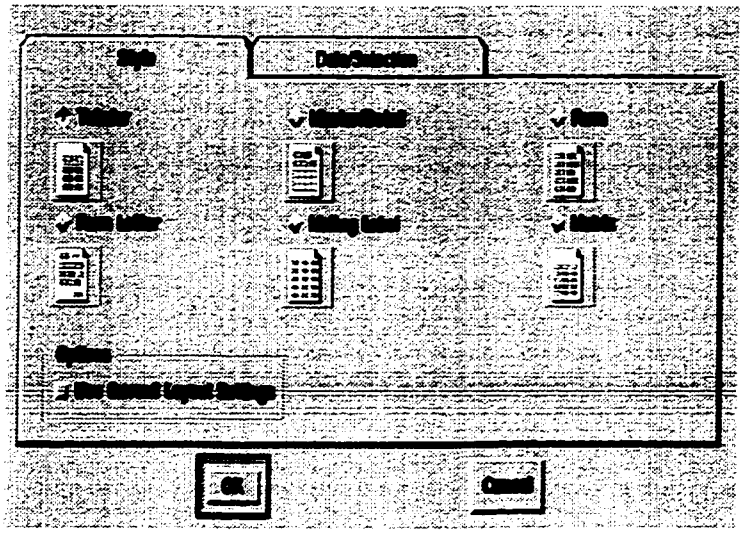


Figure 38: Default Layout Screen

To remove a column from the report, click to deselect it.

To rename a column heading, revise its label.

To adjust field length, change the width.

- In this example, choose the **Tabular**. Click **OK** to display the Layout Editor(See figure. 39).

Each of the nested Frames indicates either the entire report(outer most box), the current group or the actual columns themselves (inner most). Columns can be resized, headings can be moved and formatted(change font and size, etc.)boxes, lines and text can be added, etc. There are a number of drawing and editing tools on the palette on left side of the screen.

Now we have created the data model and a default layout.

Next, the report will be generated and run.

A default parameter form will be created automatically.

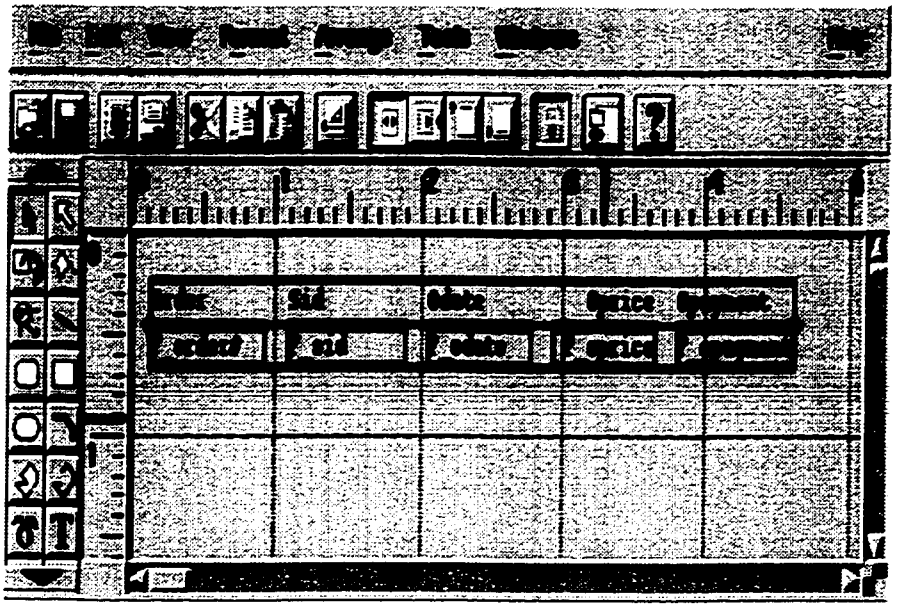


Figure 39: Layout Editor with Q_orders

6.1.3 Save, Generate and Run the Report

To save a report, choose the **Save** option from **File** menu. The source code for Oracle Reports are saved in files with an **.RDF** file name extension.

Compiled and generated reports are saved with a **.rep** extension.

For this example, save this report as **ORDER.RDF**

Once the report is saved, choose the **Run...** option from the **File** menu. The Runtime Parameter Form opens(See figure. 40).

The Runtime Parameter Form has two fields. One is a list of possible **Destination Type** for the report includeing: **Screen, Printer, E-Mail, File** and **Preview**. The **Destination Name** field will change depending on the destination type.

Keeping the Runtime Parameter Form set at Destination Type as **Screen** and click on the **Run Report** button.

As the report runs, an activity screen will appear giving an indication of the processing that is currently underway(See figure. 41).

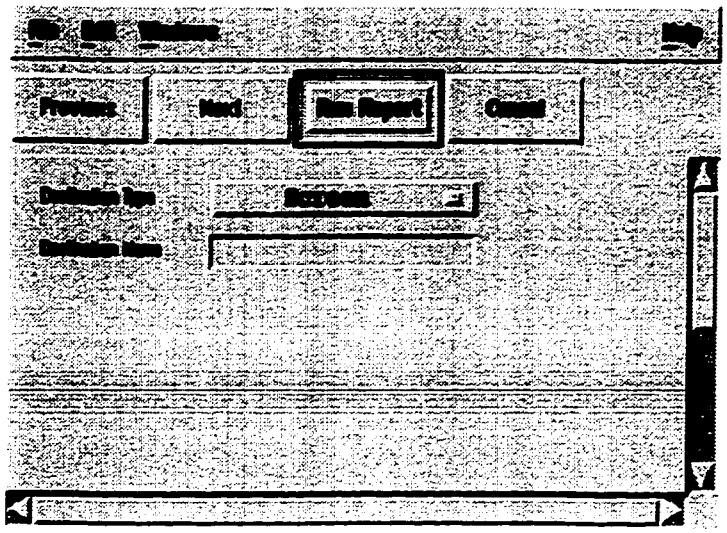


Figure 40: Runtime Parameter Form

6.1.4 Create Computation to a Report

Create a Break Report

To remove the duplicate values, create a break report based on the ORDERS identification number. This will group all information related to a particular supplier under the same **sid** in the report output. To add the summaries, create your own computational columns.

In the Data Model editor, clicking and hold the G_orders group object and drag it down and then drag the break column (e.g. **sid**) above the group to create a new group. Double-click the new group object to display the property sheet to replace the default name to **G_sid** for the group and clicking **OK**, the Data Model Editor is shown in figure. 42.

Whenever you change the format in the data model, you'll need to updata the report layout. The next step is to update the layout to incorporate your changes.

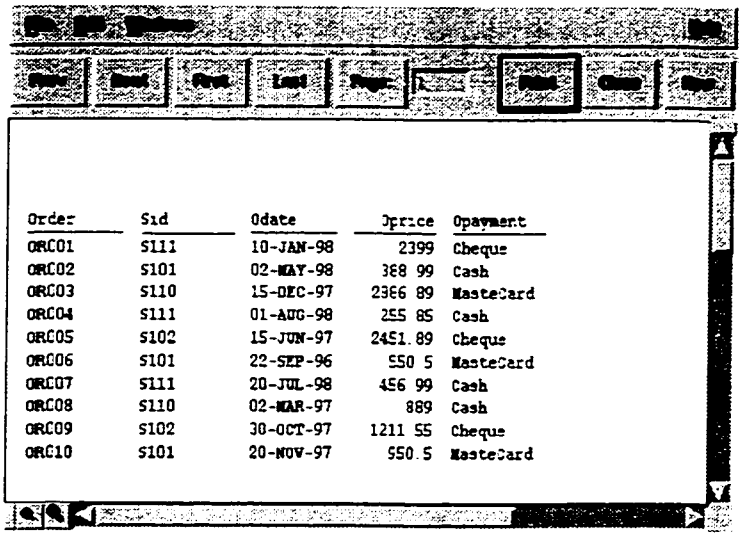


Figure 41: Activity Window

Select **Default Layout** from the **Tools**, if the selection of the Default Layout isn't changed, then choose **OK**. Otherwise, in confirmation box, clicking **Yse** button to replace the existing layout.

Running the report to display the report output as figure. 43.

Create a Summary Column

In the Data Model Editor, single-click the **Summary Column** button (See figure. 44) in the tool pallette.

To create a column with a group, click in the group at the position you want the column placed. To create a report-level column, click in outside of group area. For example, In the property sheet, rename the column to Totalprice and set the **Function** to **Sum** and the **Source** option to **oprice**, and then click **OK**.

After running, the report previewer screen is as following (See figure. 45):

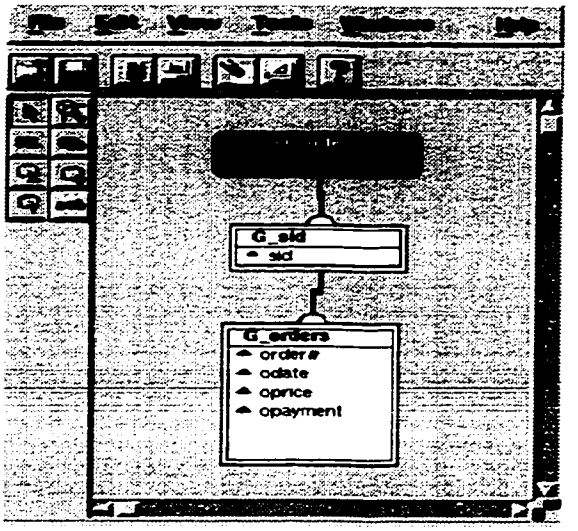


Figure 42: Q_orders with a Break Column sid

Create a Formula Column

Clicking once on the **Formula Column** button (See figure. 46) in the tool palette,

then clicking area within the G_orders. A new column, initially named CF_1, is created. Replace the name CF_1 in the new column's property sheet. Specify the other options.

Enter a formula

Click the **Edit...**, a Program Unit Editor will appear, add the formula using the PL/SQL code in the Program Unit Editor, then click **Compile**. When the code is compiled successfully, click **Close**.

6.1.5 Create Page Number Stamps

If you want to create a page number in the top or bottom margin of the report, In the layout editor, click the **Margin** button (See figure. 47) to display the margin area of the report,

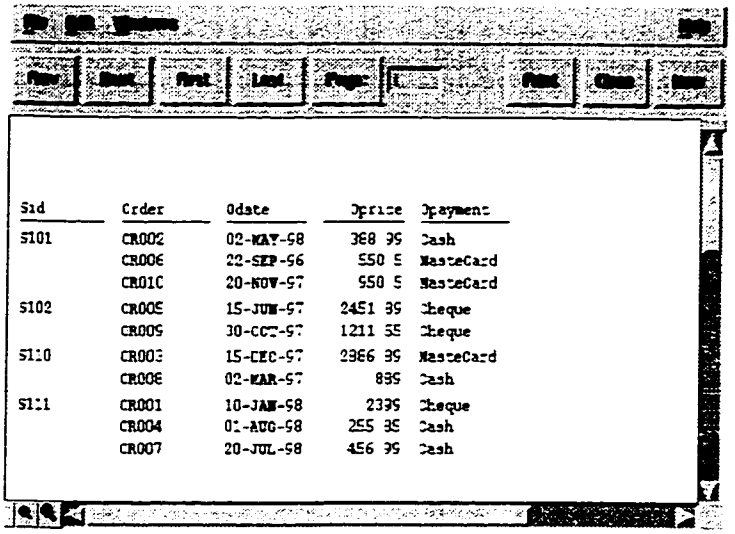


Figure 43: Report Previewer Window with a Break Column sid

Figure 44: Summary Column Button

click the **Field** icon (See figure. 48) from the tool palette,

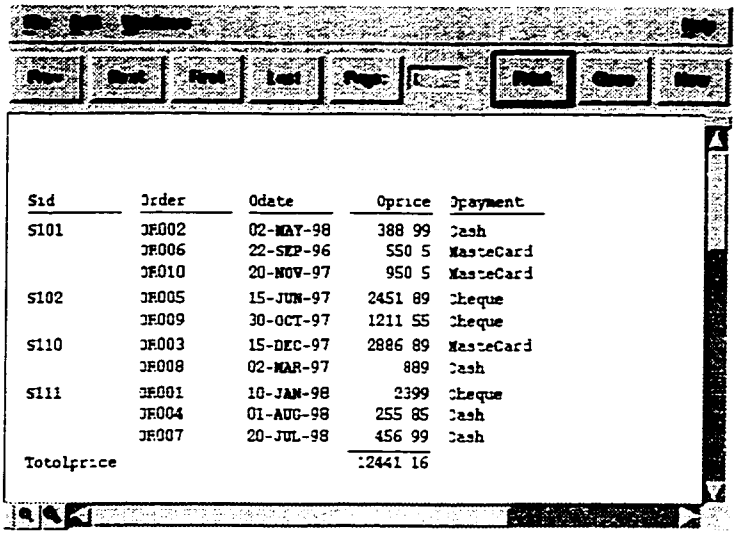
click and drag a rectangle in the bottom margin area. Double-click this field object to display the property sheet.

In the From text box of property sheet, select **Physical Page Number** from the drop-down list.

(Option) clicking the Page Numbering button to change defaults in the dialog box. Finally click OK.

6.2 Creating a Master-Detail Report

A master/detail report contains at least two groups of data. For every value of the master group, the related values in the detail group are retrieved. By comparsion, a break report has a similar data model, but has only one query. In this section, we



Sid	Order	Odate	Oprice	Opayment
S101	OR002	02-MAY-98	388 99	Cash
	OR006	22-SEP-96	550 5	MasteCard
	OR010	20-NOV-97	950 5	MasteCard
S102	OR005	15-JUN-97	2451 89	Cheque
	OR009	30-OCT-97	1211 55	Cheque
S110	OR003	15-DEC-97	2886 89	MasteCard
	OR008	02-MAR-97	889	Cash
S111	OR001	10-JAN-98	2399	Cheque
	OR004	01-AUG-98	255 85	Cash
	OR007	20-JUL-98	456 99	Cash
Totolprice			12441 16	

Figure 45: Report Previewer Window with Totalprice Row

Figure 46: Formula Column Button

will go throgh the steps for creating a master/detail report. To avoid any confusion, save and close any existing reports before proceeding.

6.2.1 Specify the Master-Detail Data Model

As in the single-table report, we begin by specifying the data model for the Master-Detail report. To create a Master/Detail report, we will use the SUPPLIER and ORDERS tables.

Begin in the Object Navigator and invoke the Data Model Editor. In the Data Model

Figure 47: Margin Button

Figure 48: Field Icon

Editor, add two queries.

In query Q_1, display the property sheet, go to General:Name, and rename the query to Q_supplier, specify the SELECT Statement as:

```
select * from supplier;
```

Query Q_supplier will act as the Master query for the report.

In query Q_2, rename the query Q_orders, specify the SELECT Statement as:

```
select * from orders;
```

Query Q_orders will act as the Detail query in this Master-Detail report. The Data Model Editor should look like figure. 49.

6.2.2 Link the Queries

The data link relates the two queries via the primary and foreign keys of the two tables. In this case, using the column sid of ORDERS and sid of SUPPLIER to create a link to establish the relationship between the queries.

To link the master query (Q_orders) with the detail query(Q_supplier):

- Select the **Data Link** button in the tool palette (See figure. 50).
- Click on the SID column of group Q_SUPPLIER and drag the mouse over to the SID1 column of group Q_ORDERS. The property sheet appears.

  For this example, close the link property sheet without changing any settings.

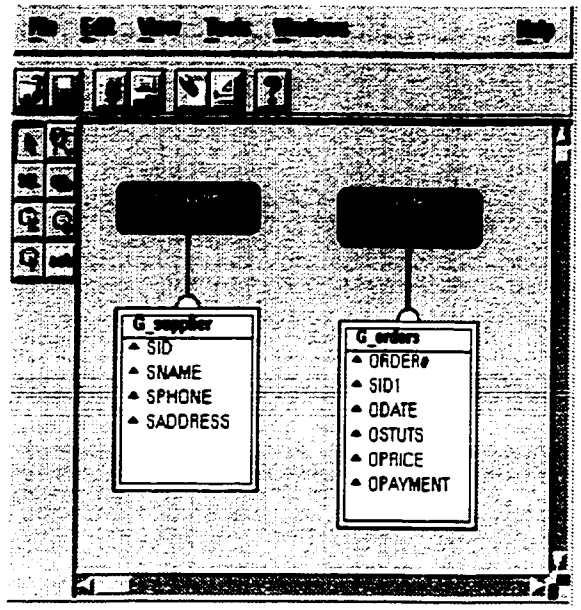


Figure 49: Data Model with two Queries

Figure 50: Data Link Button

A Link (solid black line) will appear between group Q_SUPPLIER and query Q_ORDER (See figure. 51).

Note: The direction of the arrow shows the flow of the data from the parent group to child query.

Suppose we want the report to display total price for each supplier(or master record), and total price of all suppliers (report total). we can do this in the data model using summaries.

6.2.3 Create Summaries

Create the total price for each supplier by selecting the **summary column** button and click in the master group. Double-clicking on the summary column displays

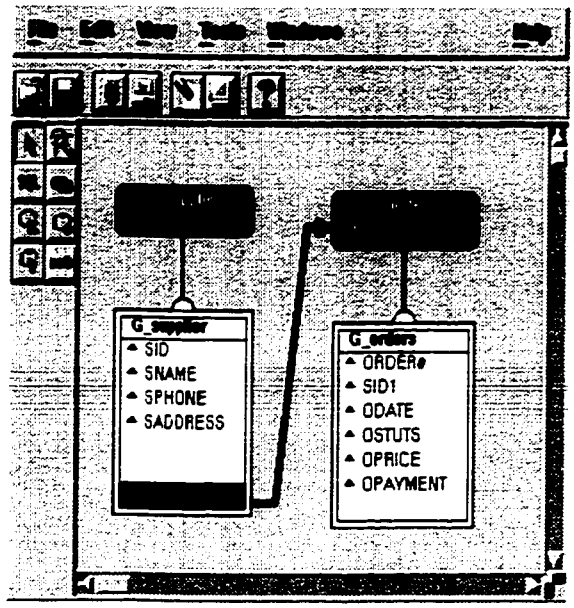


Figure 51: Data Model Editor with a Link Between Two Queries

its property sheet, where you can specify the type of computation to perform. For second summary, total price for all suppliers, selecting the summary column button and clicking outside of the groups.

At this point, the Data Model is completed.

6.2.4 Specify the Master-Detail Layout

In the Data Model, click the default layout button. When the default layout screen appears, choose Master/Detail and click on the OK button. The previewer with Master-Detail report will appear (See figure. 52).

6.2.5 Save, Generate and Run the Master-Detail Report

For now, the report can be saved, generated and Run. For this example, save the Master-Detail report as SUP_ORD.RDF, the Runtime Parameter Form is created by

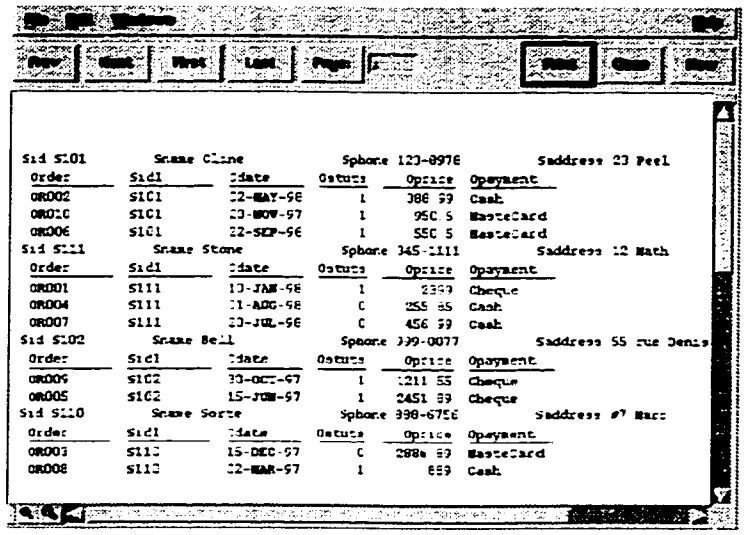


Figure 52: Report Previewer with a Maste/Detail Report

default and includes options to output the report to the **Screen** etc.

6.3 Modify the Report in Layout Editor

6.3.1 Edit, Format Text

To edit text, click the text tool in the tool palette on the left hand

Figure 53: Text Tool Button

click inside the text object, insert or modify the text, then click outside the text object.

To format text, click the text object, click the **format** option from menu to change the **font, size, style** and so on of the text.

To cut, copy or paste text, click the text tool, and then click the text object, click and drag to mark the text you want to cut or copy, choose **Edit→Cut** or Copy. To

paste text, choose **Edit→Paste**

6.3.2 Create a Page Break

In the Layout editor, double-click the object at which you want to insert a page break.

In the property sheet, click the **General Layout: Pagination setting you require**, then click **OK**.

Note: To set a page break after each repeating frame, select an Object: Maxmun Records per Page setting of 1.

Chapter 7

Conference Management System

7.1 Introduction

In this project, a database application of Oracle Developer/2000, Conference Management System(CMS), was designed and implemented. The user interfaces were built by Form4.5, the reports were created by Report2.5. The purpose of the project is explore a flexible enough way to provide a management for various conference types and an easy enough interface for the user without any coding, all functions are handled through the user interface.

The database in CMS stores all the information pertinent to the conference as follows: 1. An organizing committee that looks after the logistic, including accommodations and social program; 2. Convocations of a conference; 3. An program committee which oversees the technical program including invited talks, tutorials and special presentation; 4. The people presenting the paper during a convocation; 5. The papers submitted for the convocation with authors' information; 6. Referees who review papers submitted for the convocation; 7. A number of parallel sessions to be held on each day of the convocation; 8. Publications that contain the accepted papers and the related publisher; 9. The accounts for attendees; 10. The activities including the social program.

In addition, the project also provides the functions for several special queries such as: 1. Find the referees who were late in returning the review, 2. Find referees who have not returned their review by due day; 3. Find the referees who accept or reject more than n% of the papers they review; and 4. Find papers whose authors haven't returned the final vesion of their paper.

Furthermore, the project may display the reports in the information of the conference based on the database as follows:

1. Conference list; 2. Convocation list; 3. Session list; 4. Paper list; 5. Account list; 6. Attendee list; 7. Program committee list; 8. Referee list; 9. Social events; 10. Accepted authors list; 11. All authors list.

The database design of the project consist of five main steps:

1. System Analysis
2. Table Creation
3. Function Implementing and Graphic User Interface Building
4. Report Creation
5. Data Test

7.2 System Analysis

Database design is based on the information about the organization that is to be served by the database. The information should include the main applications and user groups; the documentation used or generated by these applications or user groups; and a prioritized list of requirements of each application or user group.

The following is the Software Requirements Specifications for this database:

1. Conference has title, theme and is identified by ConferenceNo.

2. Convocation which is a conference holding has title, conno, start date, end date and lieu and identified by an CONVONO.

3. Each conference has an orgnizing committee, membership of various committees may change over time. Each member is identified by PERSONID has firstname, lastname, position, address, email and phone attributes.

4. Each time a conference is held(convocation), it has a program committee headed by a program chair or program co-chairs, it's attributes are similar to orgnizing committee.

5. The program committee organizes the conference to be held on a number of days with a number of parallel sessions on each date. Each session has a theme, room number, time slot, a chair, and is assigned a number of accepted papers;

6. A number of papers are submitted to a conference. Each paper is identified by PAPERNO, and it has paper's title, coverage area, submit date and final due date. Each such paper, for each convocation of a conference, is given a number starting from 101 and all reference to the paper would be using this number.

7. Each paper is authored by one or more authors each of who has an address including an email.

8. Each paper is sent by the program co-chair to at least three referees. A paper must not be self-reviewed. The reviews are to be returned within a predetermided number of days, along with their comments and decision of acceptace or rejection.

9. Each person attending the conference, has a name, address including an email and phone and account which records the amounts of fees paid, amounts outstanding and list of payment. Fees may be for the conference or one of its attached activities such as tutorial, special excursions and so on.

10. For a convocation of a conference, the organizing committee organizes seriese of activities including excursions, visits to local attractions and so on, which identified by ENO, it has title, place, date, time slot and fee attributtes.

11. The accepted papers are published by a publisher that is identified by name, address, contactperson, etc. The publication has an ISBN, number of pages and contents with ordering of the accepted paper which were received in the final form before the deadline. The page number of the paper are also accessible from the paper entry.

According to system analysis, the relation schemes will be completed in three steps:

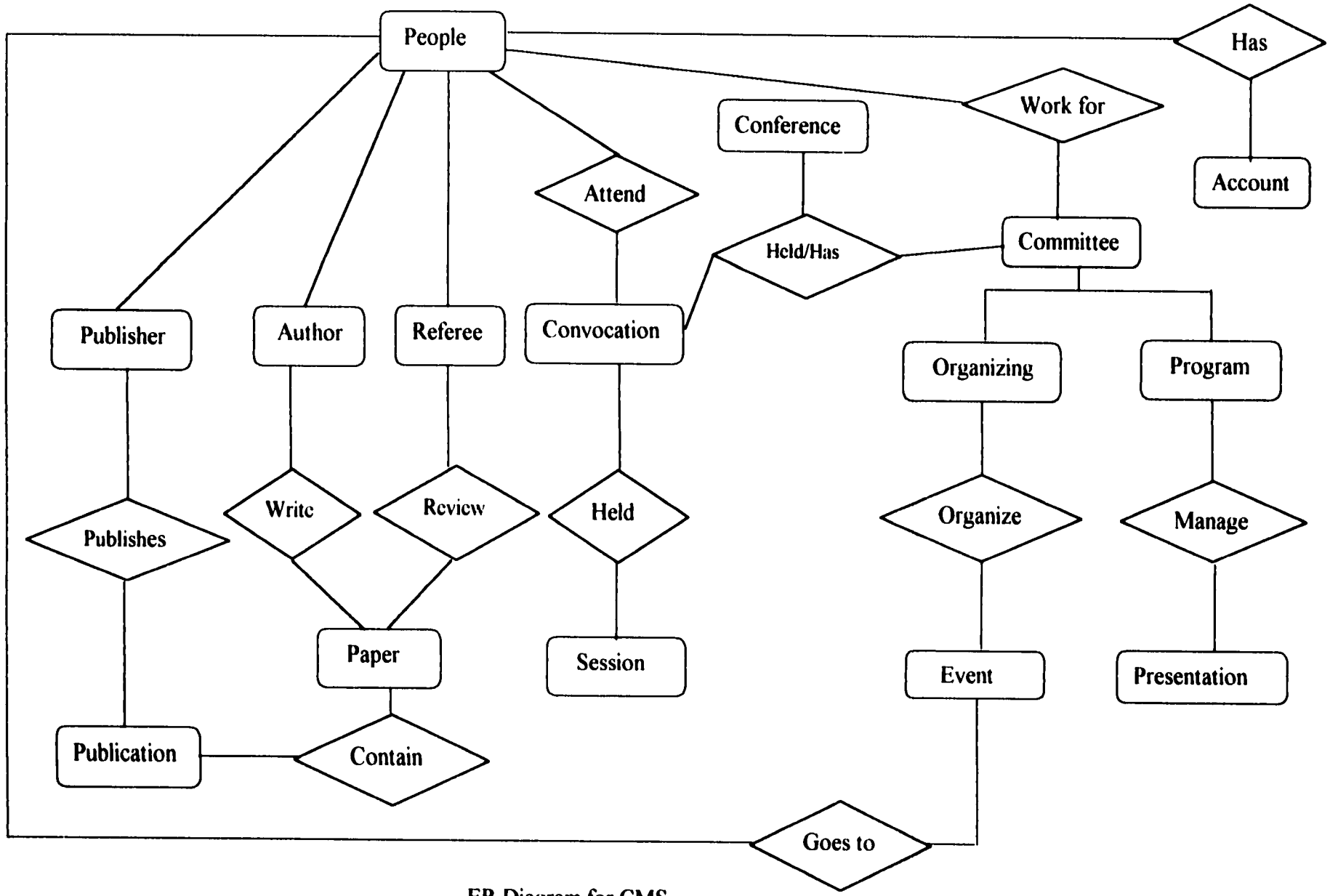
1. Developing an Entity-Relationship model;
2. Converting the ER Diagram into a relational database schema;
3. Normalization.

7.2.1 Developing an ER model for CMS

The Entity-Relationship(ER) model is high-level conceptual data model which is a set of concepts that describe the structure of a database and the associated retrieval and update transactions on the database. The main purpose for developing a high-level data model is to support a user's perception of data, and to conceal the more technical aspects associated with database design.

The Entity-Relationship(ER) model uses ER diagrams to represent the conceptual schema that is independent of the DBMS.

The ER diagram for CMS is based on above system analysis(see the ER diagram).



ER Diagram for CMS

7.2.2 Converting the ER Diagram into a Relational Database Schema

Based on the ER diagram, the following 17 relation schemas are obtained, making refinements to scheme if need. In each relation, the primary key and foreign key(s) are bold and italicized respectively.

Conference (**conno**, title, theme)

Convocaton (**convono**, *conno*, title, csdate, cedate, lieu)

Session (**sno**, *convono*, chairman, title, room, ssdate, sedate, equipment)

Paper (**paperno**, *sno*, title, coveragearea, submitdate, finaldue)

Person (**personid**, firstname, lastname, position, address, email, phone)

Organizing (*conno, personid*)

Program (*convono, personid*)

Attend (*convono, personid*)

Author (**name, *paperno***, address, email)

Activity (**eno**, *convono*, title, place, edate, estime, eetime, fee)

Goactivity (*eno, personid*)

Referee (**referee, *paperno***, email, assigndate, submitdate, rcomment, result)

Publisher (**name**, address, contactname, email, phone)

Publication (**ISBN**, title, numofpage, *publisher*)

Account (**account#**, *personid*, accofee, eventfee)

Payment (*account#*, paydate, payamount)

Contain (*paperno, ISBN*, pagenum)

7.2.3 Normalization

Normalization is a formal technique for analysing relations based on their primary key (or candidate keys in the case of BCNF) and functional dependencies. The technique involves a series of rules that can be tested against individual relations so that a database can be normalized to any degree. When a requirement is not met, the relation violating the requirement must be decomposed into relations that individually meet the requirements of normalization.

The process of normalization takes a relation through the various normal forms. At each stage the process remove undesirable characteristics from the relation.

For each relation in the schema, identify the functional dependencies that exist among the various attributtes, find out all candidate keys of relation, check whether the relation is in BCNF/3NF. If not, decompose the relation scheme into BCNF/3NF schemes.

7.3 Table Creation

The datebase tables were created according to above relation scheme using SQL*Plus tool.

SQL*Plus is one of the most powerful tools used in developing applications for the Oracle database. It provides direct access to Oracle. Although it might seem that SQL*Plus is not user-friendly like graphical query tools, it provides a great deal of flexibility and runs on all platforms supporting Oracle. You can use SQL*Plus to manipulate data and create database objects, it also is a powerful prototyping tool that you can use to develop and test SQL statements before integrating them into your application.

At first, Run SQL*Plus and connect to database, then execute the SQL*Plus command **Start**. There are seventeen tables in this project database. The following is the file to create table and insert sample data for this project.

```
****************************************
*              Drop.sql                *
****************************************

DROP TABLE c_conference  cascade constraints;
DROP TABLE c_convocation cascade constraints;
DROP TABLE c_session cascade constraints;
DROP TABLE c_person cascade constraints;
DROP TABLE c_organizing cascade constraints;
DROP TABLE c_program cascade constraints;
DROP TABLE c_paper cascade constraints;
DROP TABLE c_attend cascade constraints;
DROP TABLE c_author cascade constraints;
DROP TABLE c_activity cascade constraints;
DROP TABLE c_goactivity cascade constraints;
DROP TABLE c_referee cascade constraints;
DROP TABLE c_publisher cascade constraints;
DROP TABLE c_publication cascade constraints;
DROP TABLE c_contain cascade constraints;
DROP TABLE c_account cascade constraints;
DROP TABLE c_payment cascade constraints;

****************************************
*            Creat_table.sql           *
****************************************

/***************c_conference***********/
create table c_conference
(
ConNo                   NUMBER(8)           NOT NULL,
Title                   VARCHAR2(20)        NOT NULL,
Theme                   VARCHAR2(20),
    PRIMARY KEY (ConNo)
);


/**************c_convocation*************/
create table c_convocation
(
ConvoNo                    NUMBER(8)         NOT NULL,
```

```
ConNo                    NUMBER(8)          NOT NULL,
Title                    VARCHAR2(20)       NOT NULL,
Csdate                   DATE,
Cedate                   DATE,
Lieu                     VARCHAR2(20),
    PRIMARY KEY (ConvoNo),
    FOREIGN KEY (ConNo) references c_conference
);


/******************c_session**************/
create table c_session
(
SNo                      NUMBER(8)          NOT NULL,
ConvoNo                  NUMBER(8)          NOT NULL,
Chairman                 VARCHAR2(10),
Title                    VARCHAR2(30)       NOT NULL,
Room                     VARCHAR2(10),
Ssdate                   DATE,
Sedate                   DATE,
Equipment                VARCHAR2(30),
    PRIMARY KEY (SNo),
    FOREIGN KEY (ConvoNo) references c_convocation
);


/******************c_person**************/
create table c_person
(
PersonID                 NUMBER(8)          NOT NULL,
FirstName                VARCHAR2(30)       NOT NULL,
LastName                 VARCHAR2(30)       NOT NULL,
Position                 VARCHAR2(30),
Address                  VARCHAR2(20),
Email                    VARCHAR2(20),
Phone                    VARCHAR2(20),
     PRIMARY KEY (PersonID)
);
```

```
/***************c_organizing**************/
create table c_organizing
(
ConNo                       NUMBER(8)        NOT NULL,
Personid                    NUMBER(8)        NOT NULL,
      PRIMARY KEY (ConNo, Personid),
      FOREIGN KEY (Personid) references c_person,
      FOREIGN KEY (ConNo) references c_conference
);


/****************c_program*****************/
create table c_program
(
ConvoNo                     NUMBER(8)         NOT NULL,
Personid                    NUMBER(8)         NOT NULL,
      PRIMARY KEY (ConvoNo, Personid),
      FOREIGN KEY (Personid) references c_person,
      FOREIGN KEY (ConvoNo) references c_convocation
);


/****************c_paper*********************/
create table c_paper
(
PaperNo                     NUMBER(8)            NOT NULL,
SNo                         NUMBER(8)            NOT NULL,
Title                       VARCHAR2(20)         NOT NULL,
CoverageArea                VARCHAR2(40),
SubmitDate                  Date,
FinalDue                    Date,
     PRIMARY KEY (PaperNo),
     FOREIGN KEY (SNo) references c_session
);


/****************c_attend*********************/
create table c_attend
(
Convono                NUMBER(8)          NOT NULL,
```

```
Personid                NUMBER(8)          NOT NULL,
    PRIMARY KEY (Convono, Personid),
    FOREIGN KEY (ConvoNo)  references c_convocation,
    FOREIGN KEY (Personid) references c_person
);


/****************c_author********************/
create table c_author
(
PaperNo                 NUMBER(8)          NOT NULL,
Name                    VARCHAR2(20)       NOT NULL,
Address                 VARCHAR2(20),
Email                   VARCHAR2(20),
     PRIMARY KEY (PaperNo, Name),
     FOREIGN  KEY (PaperNo)  references c_paper
);


/****************c_activity*********************/
create table c_activity
(
ENo                     NUMBER(8)          NOT NULL,
ConvoNo                 NUMBER(8)          NOT NULL,
Title                   VARCHAR2(30)       NOT NULL,
Place                   VARCHAR2(30),
EDate                   DATE,
Estime                  VARCHAR2(6),
Eetime                  VARCHAR2(6),
Fee                     NUMBER(8, 2),
    PRIMARY KEY (ENo),
    FOREIGN KEY (ConvoNo)  references c_convocation
);


/*******************c_goactivity***************/
create table c_goactivity
(
ENo                             NUMBER(8)   NULL,
PersonID                        NUMBER(8)  NULL,
```

```
    PRIMARY KEY (ENo, PersonID),
    FOREIGN KEY (ENo)  references c_event,
    FOREIGN KEY (PersonID)  references c_person
);


/******************c_referee**********************/
create table c_referee
(
Referee                    VARCHAR2(15)    NOT NULL,
PaperNo                    NUMBER(8)       NOT NULL,
Email                      VARCHAR2(30),
AssignDate                 DATE,
SubmitDate                 DATE,
Rcomment                   VARCHAR2(30),
Result                     VARCHAR2(30),
    PRIMARY KEY (Referee, PaperNo),
    FOREIGN KEY (PaperNo)  references c_paper
);


/*****************c_publisher********************/
create table c_publisher
(
Name                     VARCHAR2(20)      NOT NULL,
Address                  VARCHAR2(30)      NOT NULL,
ContactName              VARCHAR2(30),
Email                    VARCHAR2(40),
Phone                    VARCHAR2(20),
   PRIMARY KEY (NAME)
);


/******************c_publication*****************/
create table c_publication
(
ISBN                   VARCHAR2(20)      NOT NULL,
Title                  VARCHAR2(40)      NOT NULL,
NumOfPage              NUMBER(8),
Publisher              VARCHAR2(20)      NOT NULL,
```

```
    PRIMARY KEY (ISBN),
    FOREIGN KEY (Publisher) references c_publisher (name)
);


/*******************c_account*********************/
create table c_account
(
Account#                NUMBER(8)       NOT NULL,
PersonID                NUMBER(8)       NOT NULL,
AccoFee                 NUMBER(8, 2),
EventFee                NUMBER(8, 2),
      PRIMARY KEY (Account#),
      FOREIGN KEY (PersonID)  references c_person
);


/*******************c_payment*********************/
create table c_payment
(
Account#            NUMBER(8)       NOT NULL,
PayDate             DATE,
PayAmount           NUMBER(8, 2),
     PRIMARY KEY (Account#, PayDate, PayAmount),
     FOREIGN KEY (Account#) references c_account
);


/*******************c_contain*********************/
create table c_contain
(
PaperNo                 NUMBER(8)         NOT NULL,
ISBN                    VARCHAR2(20)      NOT NULL,
PageNum                 NUMBER(8),
    PRIMARY KEY (PaperNo, ISBN),
    FOREIGN KEY (PaperNo)  references c_paper,
    FOREIGN KEY (ISBN)  references
    c_publication
);
```

```
**********************************
*         Insert.sql             *
**********************************

insert into c_conference
values(1, 'conference 1', 'Theme 1');
insert into c_conference
values(2, 'conference 2', 'Theme 2');


insert into c_convocation
values(1, 1, 'convocation 1-1', '1-jan-98', '20-jan-98', 'Hell Building');
insert into c_convocation
values(2, 1, 'convocation 1-2', '10-feb-99', '20-feb-99', 'Building A');


insert into c_session
values(1, 1, 'John', 'Session 1-1', 'H333', '2-jan-98',
'6-jan-98', 'Table, TV');
insert into c_session
values(2, 1, 'Jerry', 'Session 1-2', 'H666', '7-jan-98',
'20-jan-98', 'Video, TV');


insert into c_person
values(1, 'Jerry', 'Blone', 'member ', '#656 durocher', 'jerry@email.com',
'514-222-3333');
insert into c_person
values(2, 'Jack', 'Chen', 'member ', '#123 St. Denis', 'jack@cs.ca',
'514-111-3322');


insert into c_organizing
values(1, 1);
insert into c_organizing
values(2, 2);
```

```
insert into c_program
values(1, 3);
insert into c_program
values(2, 5);


insert into c_paper
values(1, 1, 'Paper 1-1', 'ConverageArea 1', '1-jan-98', '20-jan-98');
insert into c_paper
values(2, 1, 'Paper 1-2', 'ConverageArea 2', '10-jan-98', '30-jan-98');


insert into c_attend
values(1, 1);
insert into c_attend
values(2, 3);


insert into c_author
values(1, 'Wisle', '#234, Massional', 'wisle@xx.com');
insert into c_author
values(1, 'Lao', '#33, guy', 'lao@yy.com');


insert into c_activity
values(1, 1, 'movies', 'Empire', '26-jan-98', '19:30', '22:00', 25.00);
insert into c_event
values(2, 1, 'visiting', 'Old Montreal', '28-jan-98', '10:30', '12:00', 35.00


insert into c_goactivity
values(1, 1);
insert into c_goactivity
values(1, 2);


insert into c_referee
values('John', 1, 'john@cs.ca', '2-jan-98', '20-jan-98',
'comment1', 'Accept');
insert into c_referee
```

```
values('Martin', 1, 'martin@css.ca', '2-jan-98', '20-jan-98',
'comment2', 'Reject');


insert into c_publisher
values('Smae', '#7877 St. Drovo', 'Cinthic', 'cinthic@smae.com',
'541-777-0909');
insert into c_publisher
values('Demem', '#1111 St. decarie', 'wenly', 'wenly@demem.com',
'541-222-2222');


insert into c_publication
values('PID 000123-98', 'Computer maginzine', 120, 'Smae');
insert into c_publication
values('SDR 900234-78', 'Database Group', 80, 'Smae');


insert into c_account
values(10000001, 1, '300.00', '55.00');
insert into c_account
values(10000002, 2, '400.00', '75.00');


insert into c_payment
values(10000001, '25-jan-98', 200.00);
insert into c_payment
values(10000001, '27-jan-98', 100.00);


insert into c_contain
values(1, 'SDR 900234-78', 20);
insert into c_contain
values(2, 'IUI 790111-23', 63);
```

7.4 Functions and The User Interfaces Implementing

In order to complete required functions, user interfaces were designed as following:

1. **Main Window**

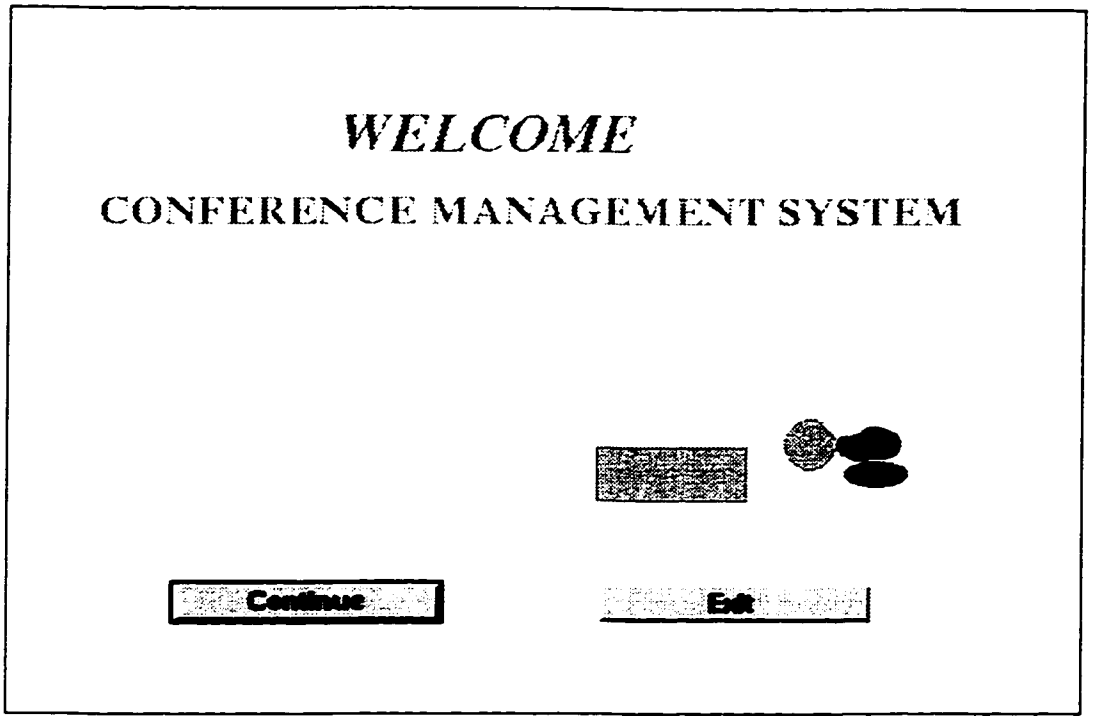


Figure 55: Main Window

The main window is first window of CMS (Figure. 55).

2. **Menu Window**

In the window (Figure. 56), user can access functional windows by selecting options from menu bar and listed buttons.

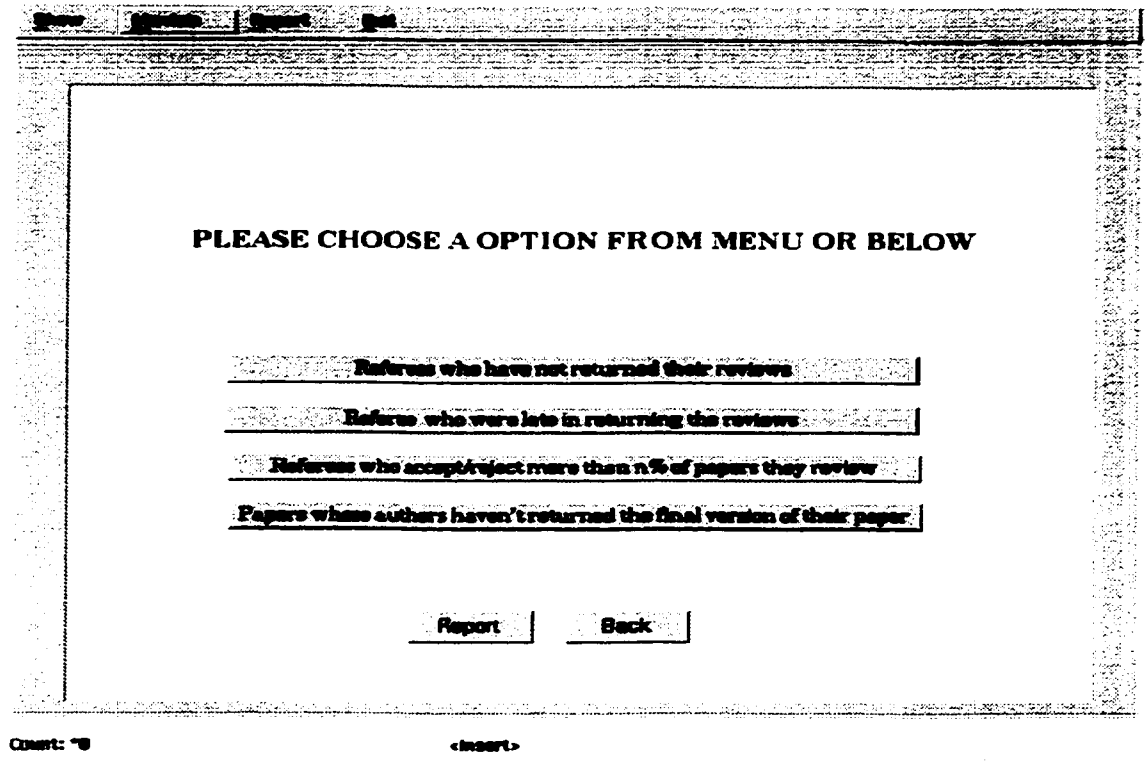


Figure 56: Menu Window

3. **Add a new conference and its organizing committee and other associated detail.**

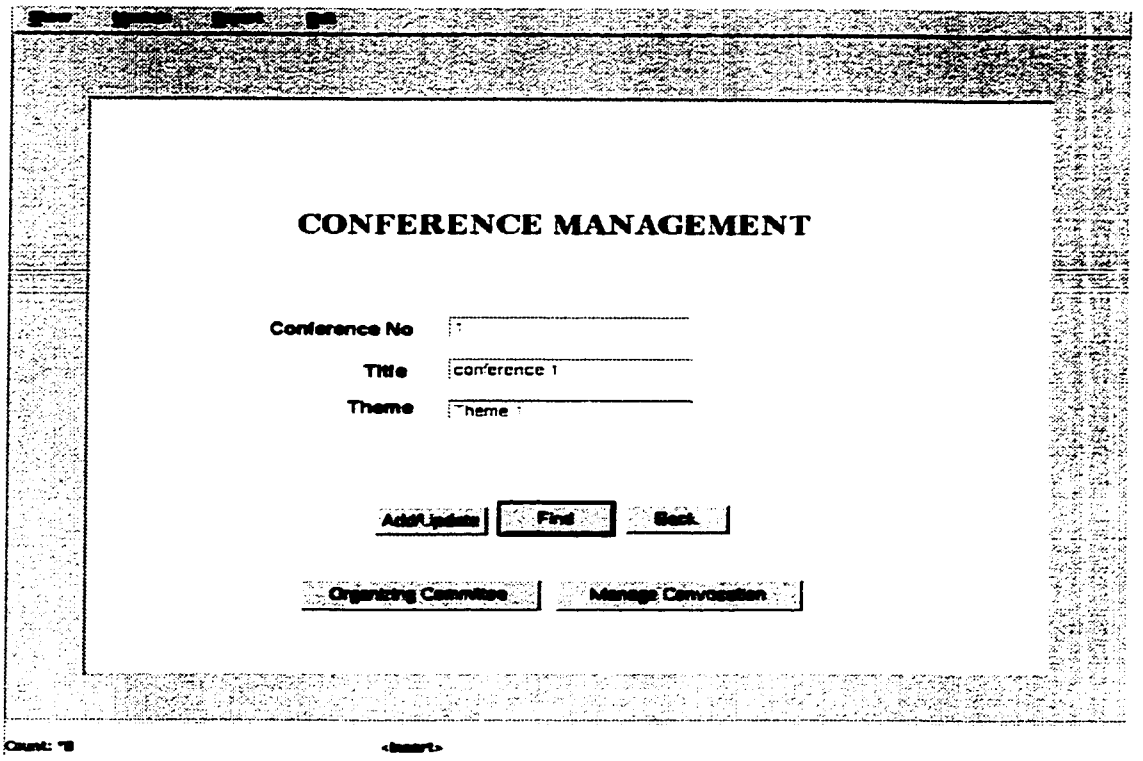


Figure 57: Conference Management

In the **Conference Management** window (Figure. 57), user can add a new conference, update an exist conference, browse a list of conferences which exist in database and can switch to **menu** window, **organizing committee** window and **convocation management** window.

In the **Organizing Committee** window (Figure. 58), user can add a new organizing committee member, update an existing member and browse a list of members who exist in the database under a specific conference.

4. **For an existing conference, add a convocation for it including the details of its date, lieu, program committee etc. Associate papers, authors, attendees with this convocation.**

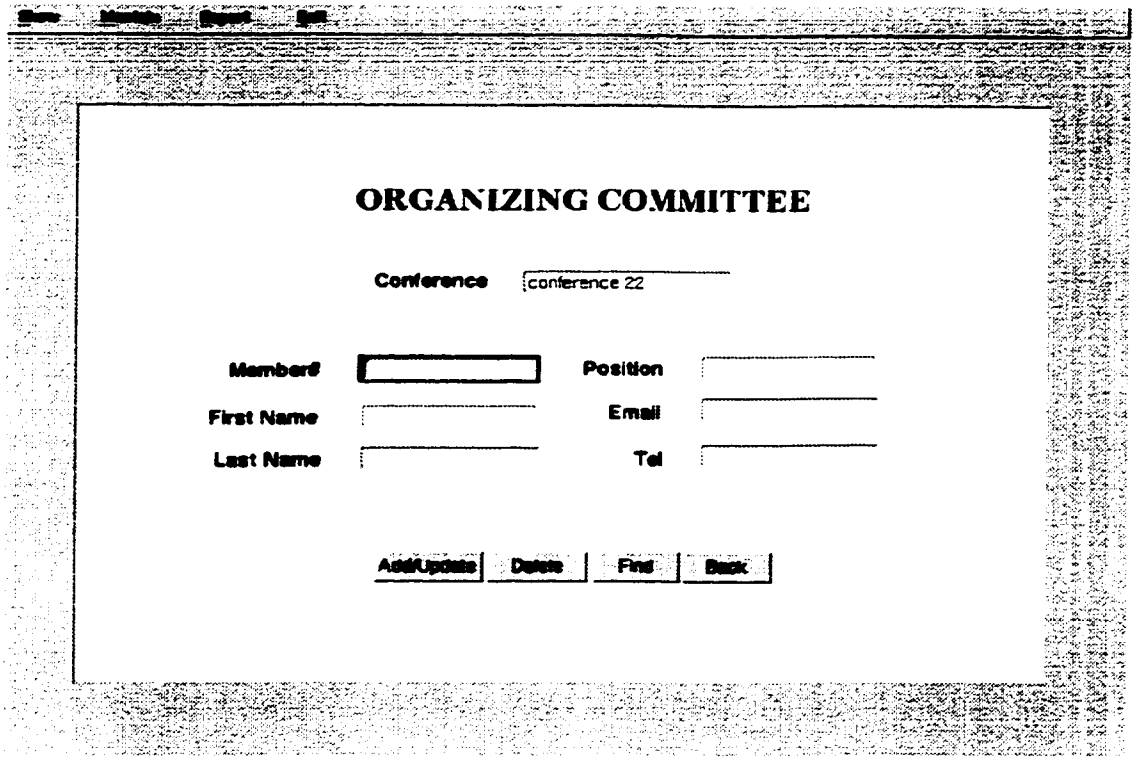


Figure 58: Organization Management

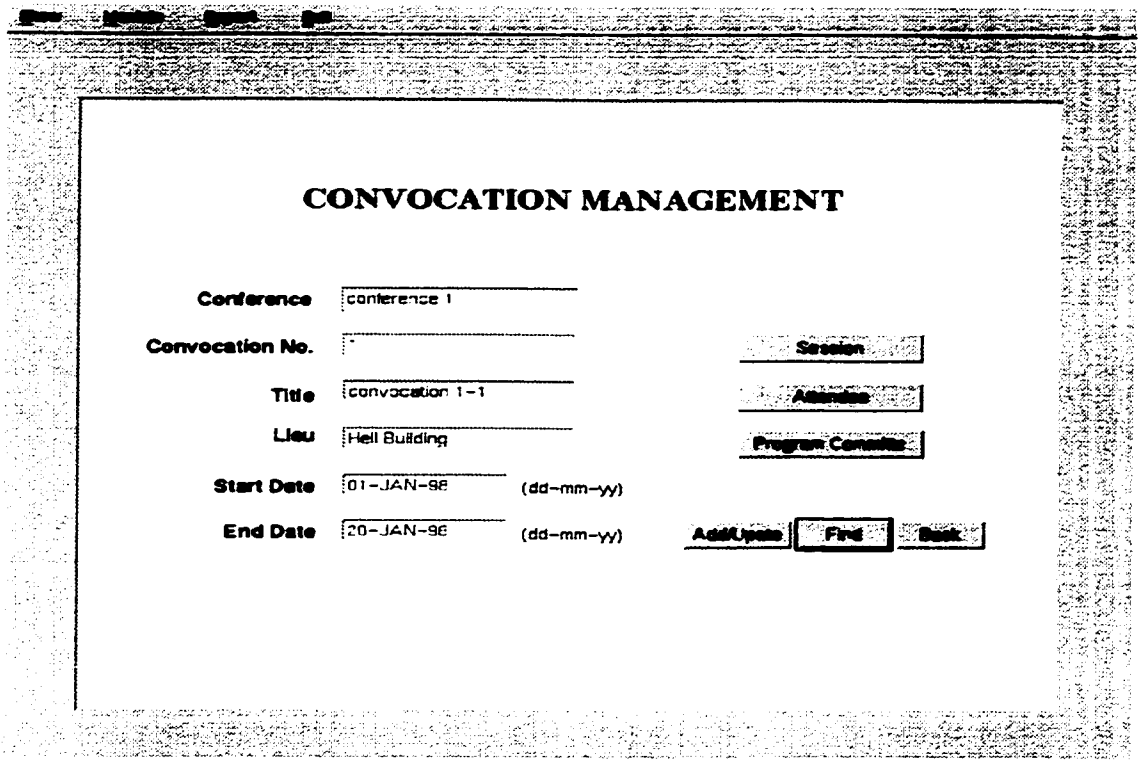


Figure 59: Convocation Management

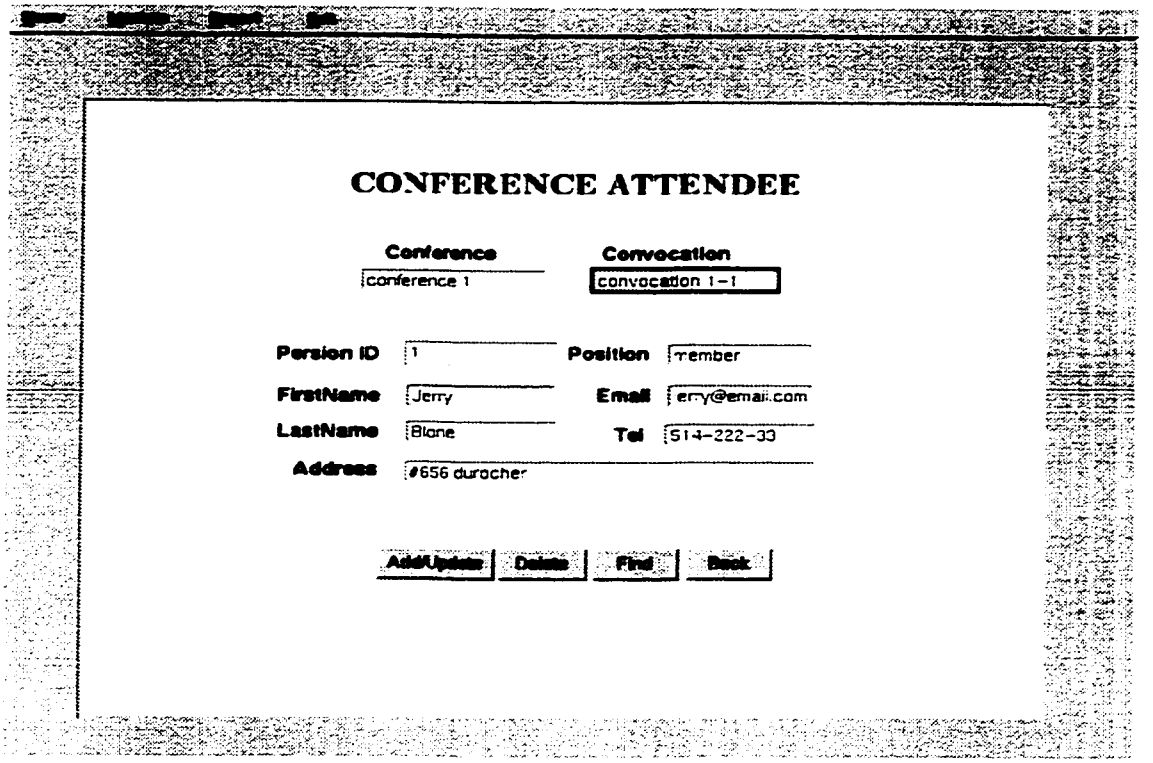


Figure 60: Attendee Management

In the **Convocation Management** window (Figure. 59), user can add a new convocation, update an existing convocation, browse a list of convocations which exist in the database and can swich to **Session Management** window, **Attendee Management** window and **Program Committee** window.

In the **Attendee Management** window (Figure. 60), user can add a new attendee to a specific convocation, update an existing attendee, browse a list of attendees who exist in the database.

In the **Paper Management** window (Figure. 61), user can add a new paper with its author information, update an existing paper, browse a list of papers which exist in the database and can swich to **referee Management** window.

5. **For a convocation of a conference, add activities along with their detail information, such as date, place, time slot, title, fee.**

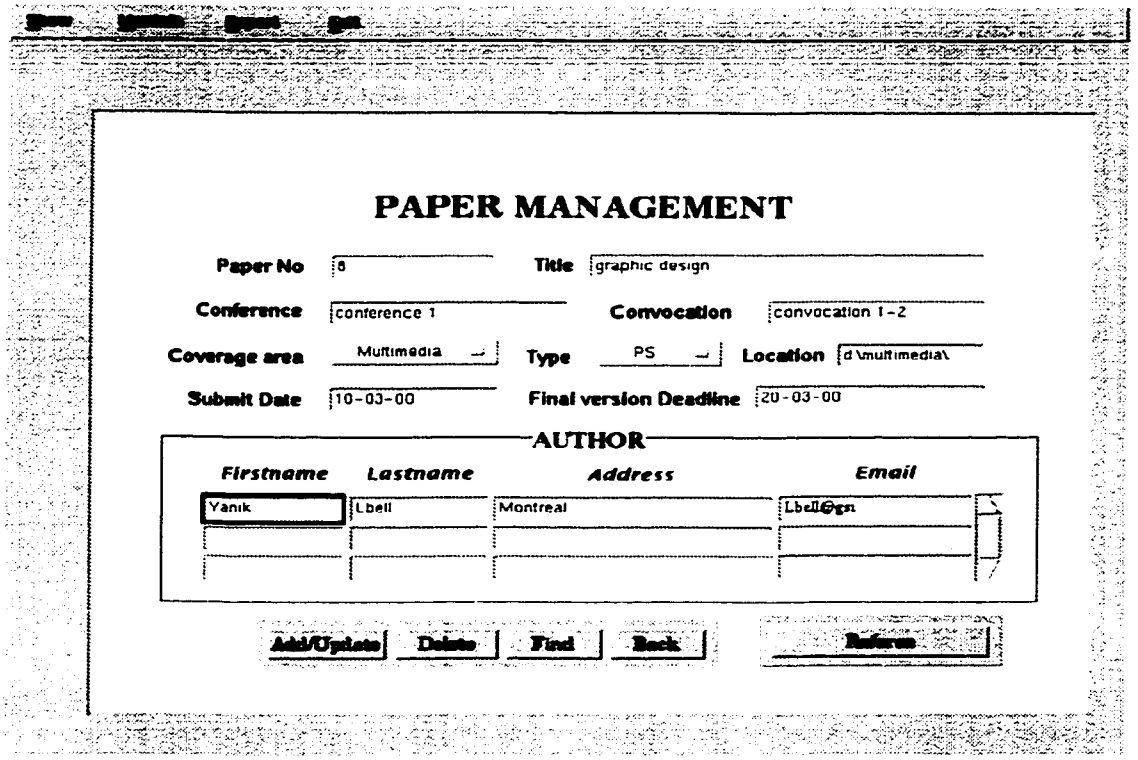


Figure 61: Paper Management

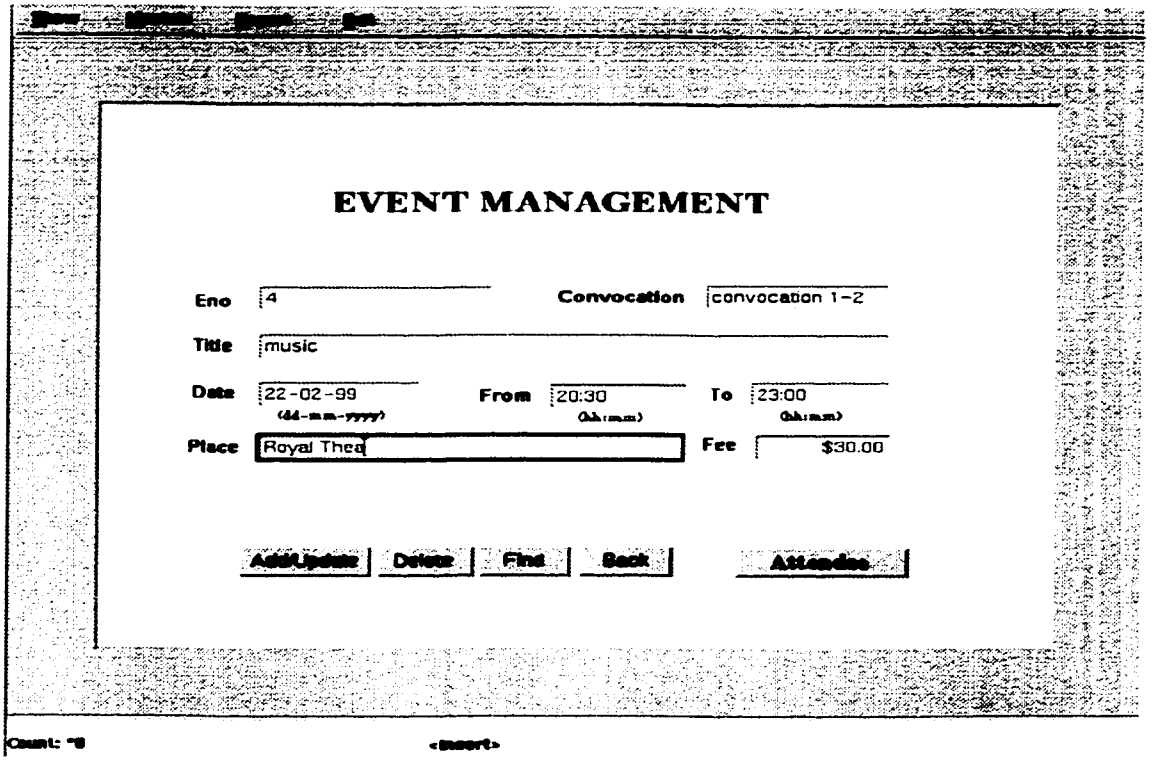


Figure 62: Event Management

In the **Event Management** window (Figuer. 62), user can add a new new, update an exist event, browse a list of events which exist in the database and can swich to **Event attendee Management** window for specific convocation.

6. **For a convocation, add sessions, and for sessions, assign chairs, papers, rooms, special equipment required.**

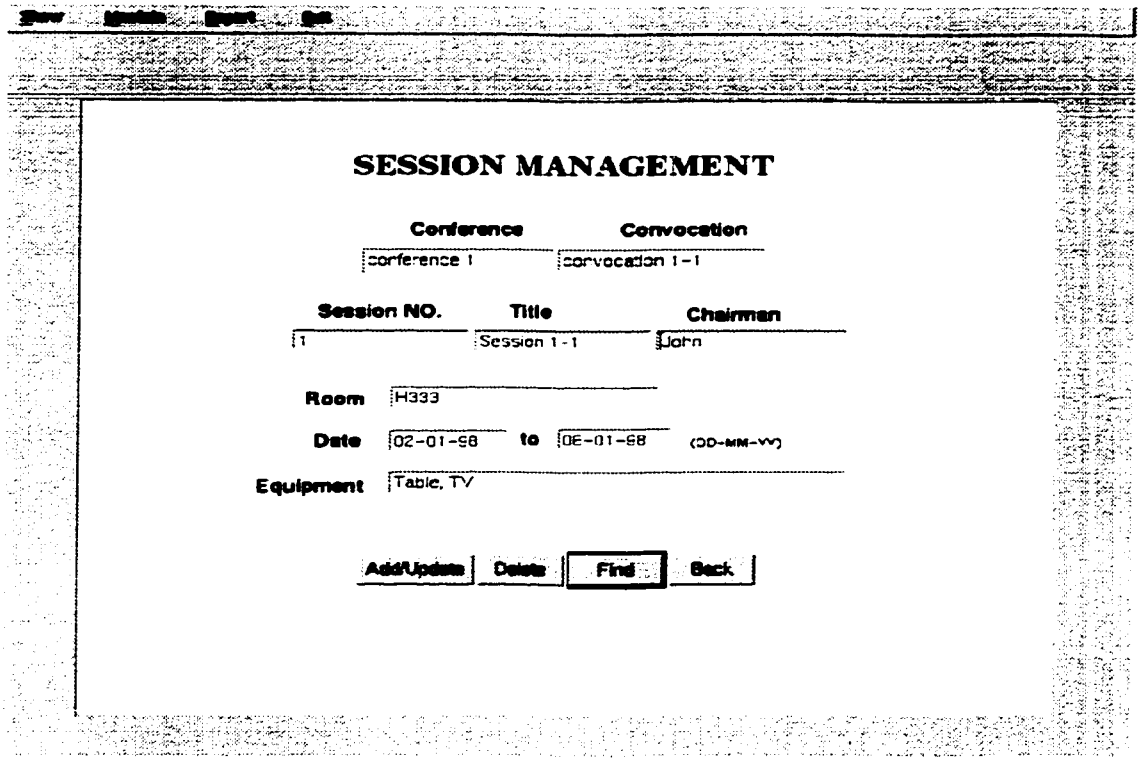


Figure 63: Session Management

In the **Session Management** window (Figure. 63), user can add a new session, update an existing session, browse list of sessions which exist in the database to specific convocation.

7. **Find which referees were late in returning the reviews.**

In the above window (Figure. 64), user can browse the referees' information who have not returned their reviews by due date.

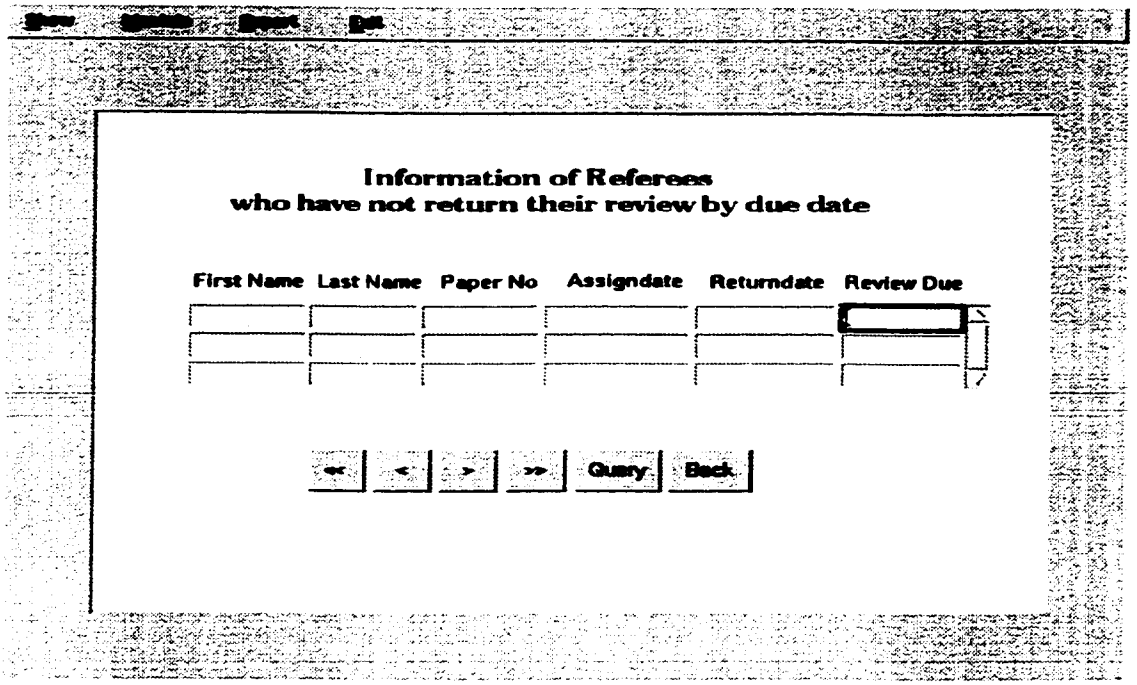


Figure 64: Referees were late in returning the reviews

8. Find referees who have not returned their review by due date.

In the above window (Figure. 65), user can browse a the referees' information who have not returned their reviews by the due day.

9. Find referees who accept/reject more than n% of papers they review.

In the above window (Figure. 66), user may choose accept or reject option and enter the number of percent, then click **Query** button to browse list information of referees who accept or reject more than n% of papers they review.

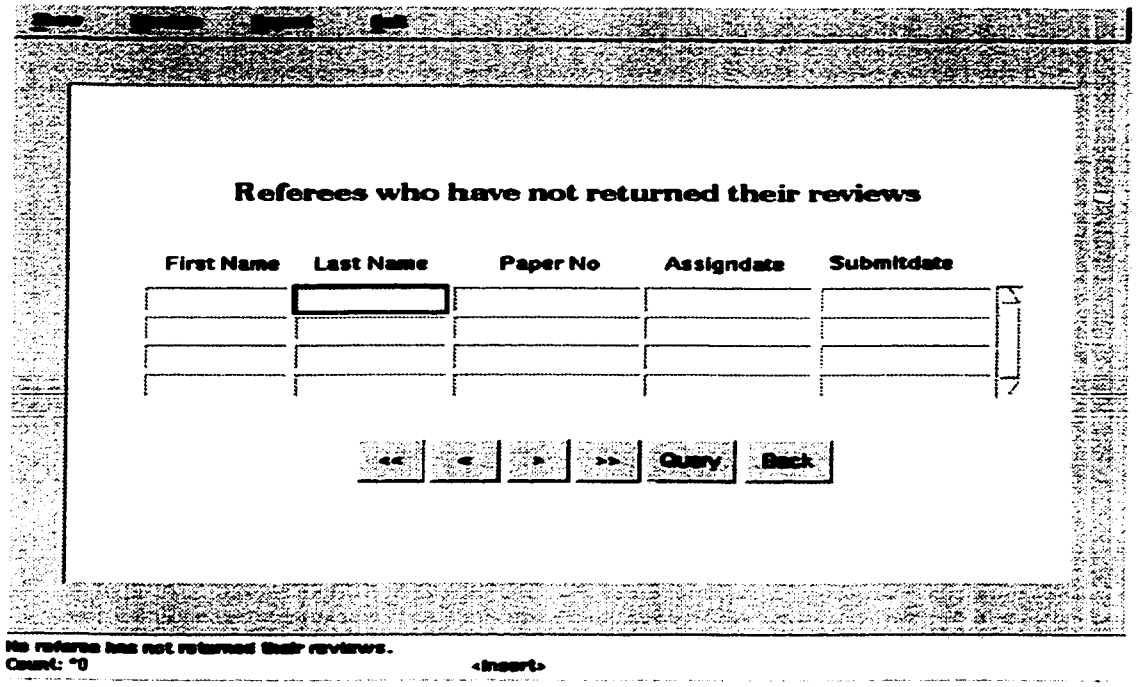


Figure 65: Referees who have not returned their review by due date

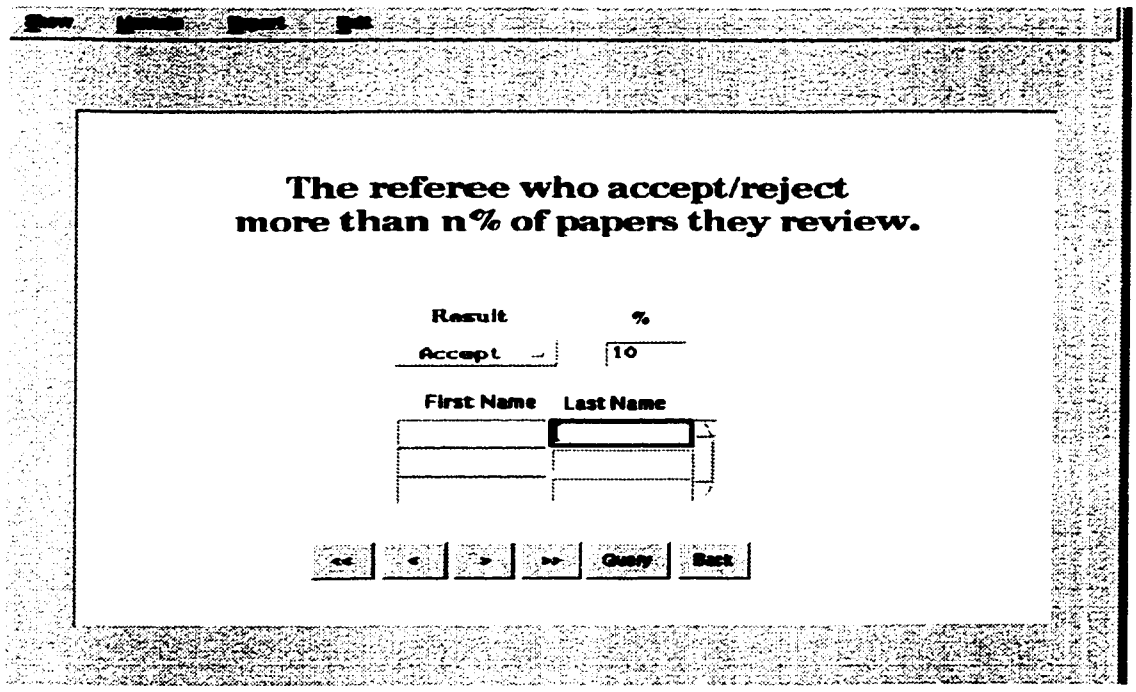


Figure 66: Referees who accept/reject more than n% of papers

10. **For a given convocation of a conference, find papers whose authors haven't returned the final version of their paper.**

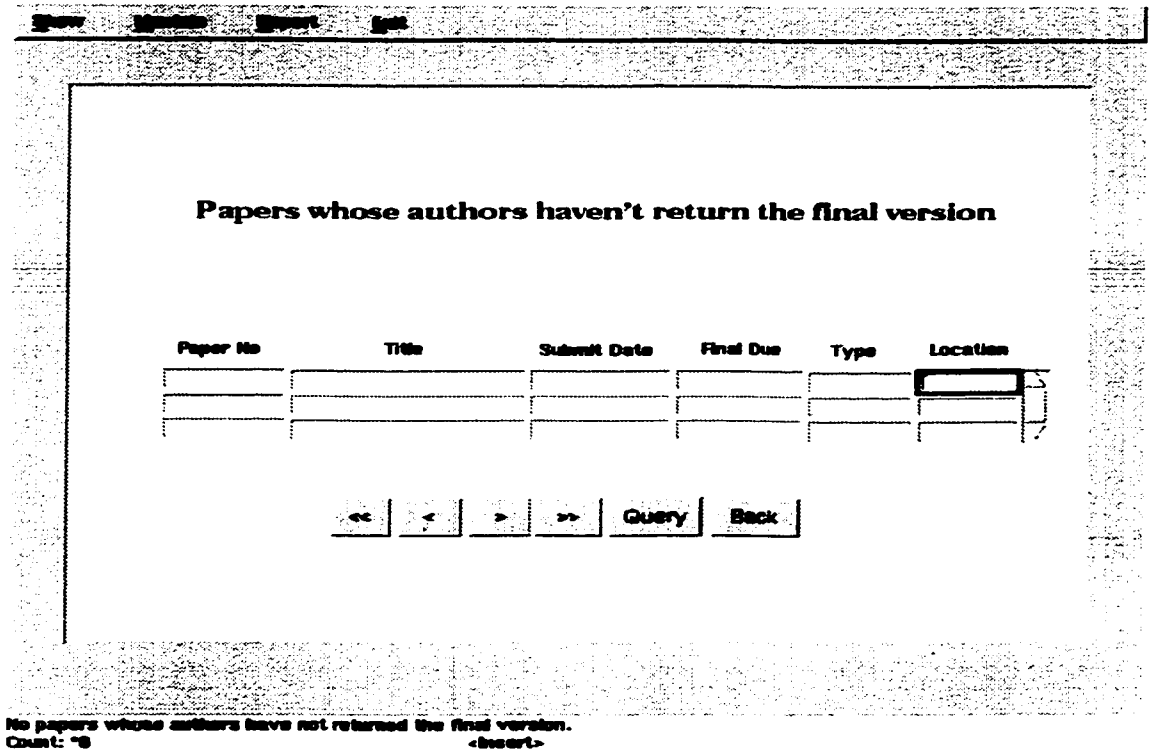


Figure 67: Papers whose authors have not returned the final version of paper

In the above window (Figure.67), user may browse a list of information of paper whose authors haven't returned the final version of their paper to a specific convocation.

11. **Manage papers submitted, referees assigned, reviews received, late reviews and reminders required, acceptance and rejection decision, final version of acceptance papers and authors attending.**

In the **Referee Management** window (Figure.68), user add information of referee and his/her comments and result to a given paper.

12. **Publication details.**

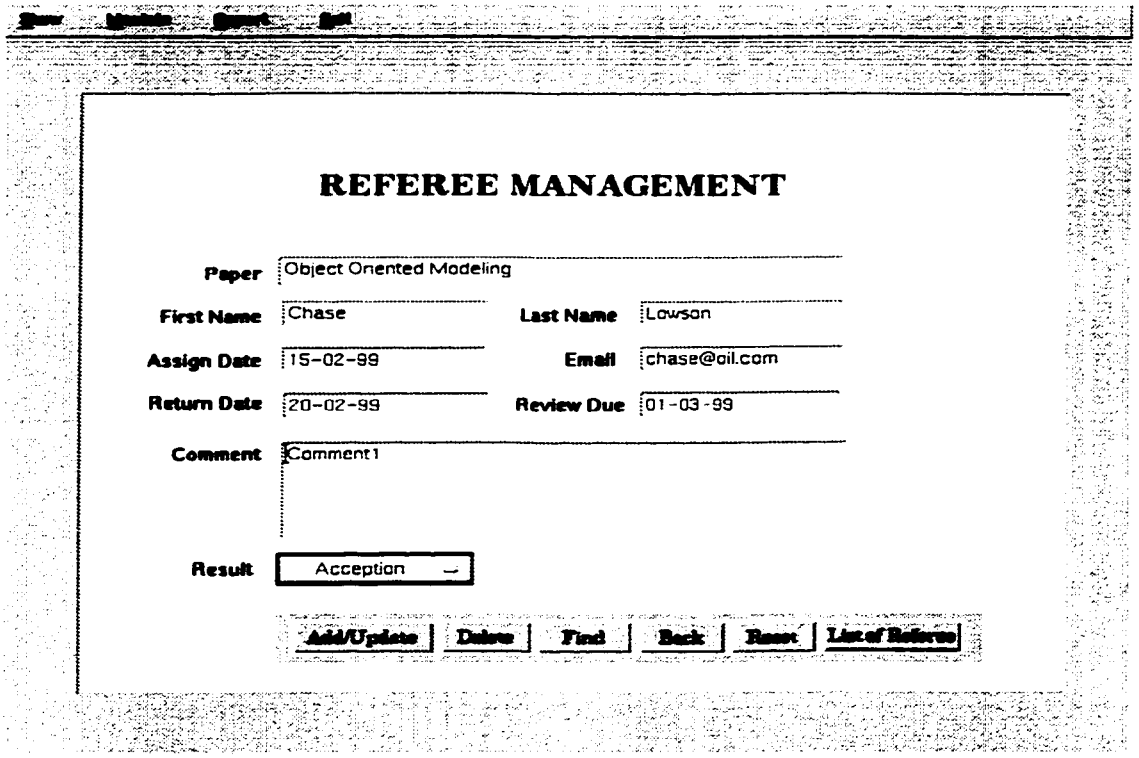


Figure 68: Referees Management

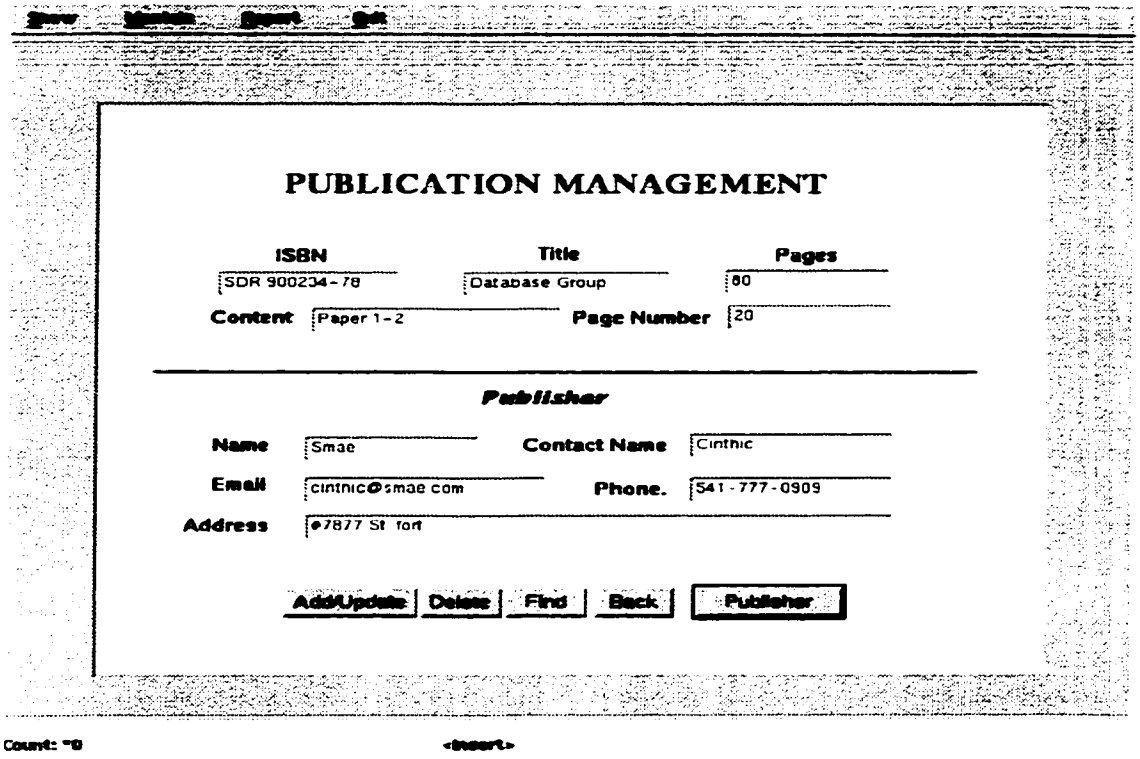


Figure 69: Publication Management

In the **Publication Management** window (Figure.69), user may add a new publication with papers contained in this publication, and information of publisher of this publication, update the information of a publication which exist in database and browse a list of publications.

13. Details of accounts and list of payment for attendees.

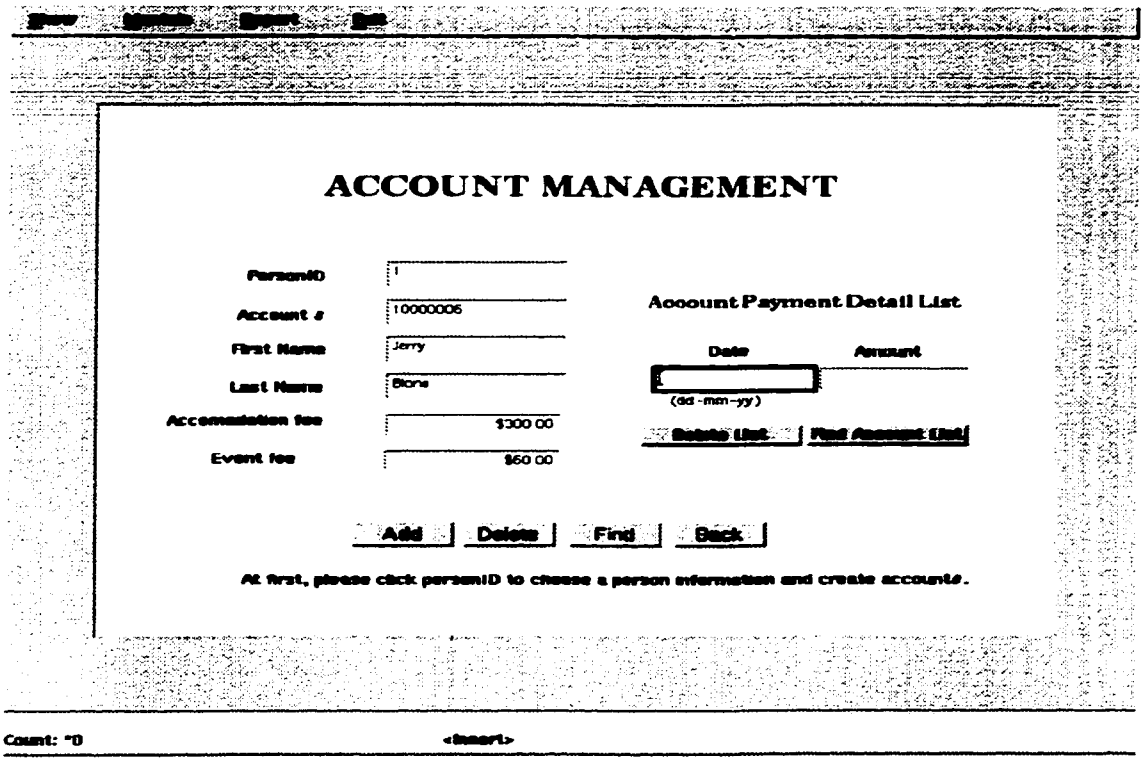


Figure 70: Account Management

In the **Account Management** window (Figure.70), user may add information of account for a attendee who already exist in the database with detail payment list.

Ad hoc queries, such as the following, is also to be supported:

14. Find the details for a given conference.

In the **Conference Information** window (Figure.71), user may browse all the

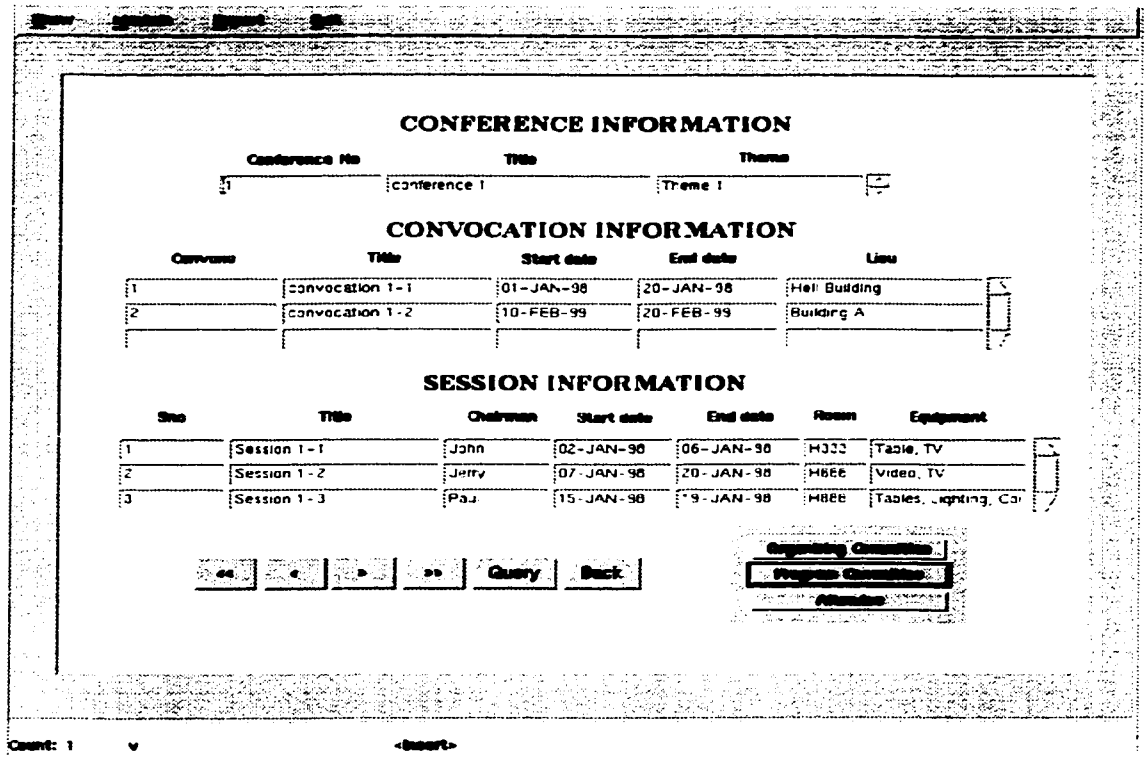


Figure 71: Conference Information

details for a given conference, including information of its convocations, sessions and may switch to **Organzing Committee** window, **Program Committee** window and **Attendee** window.

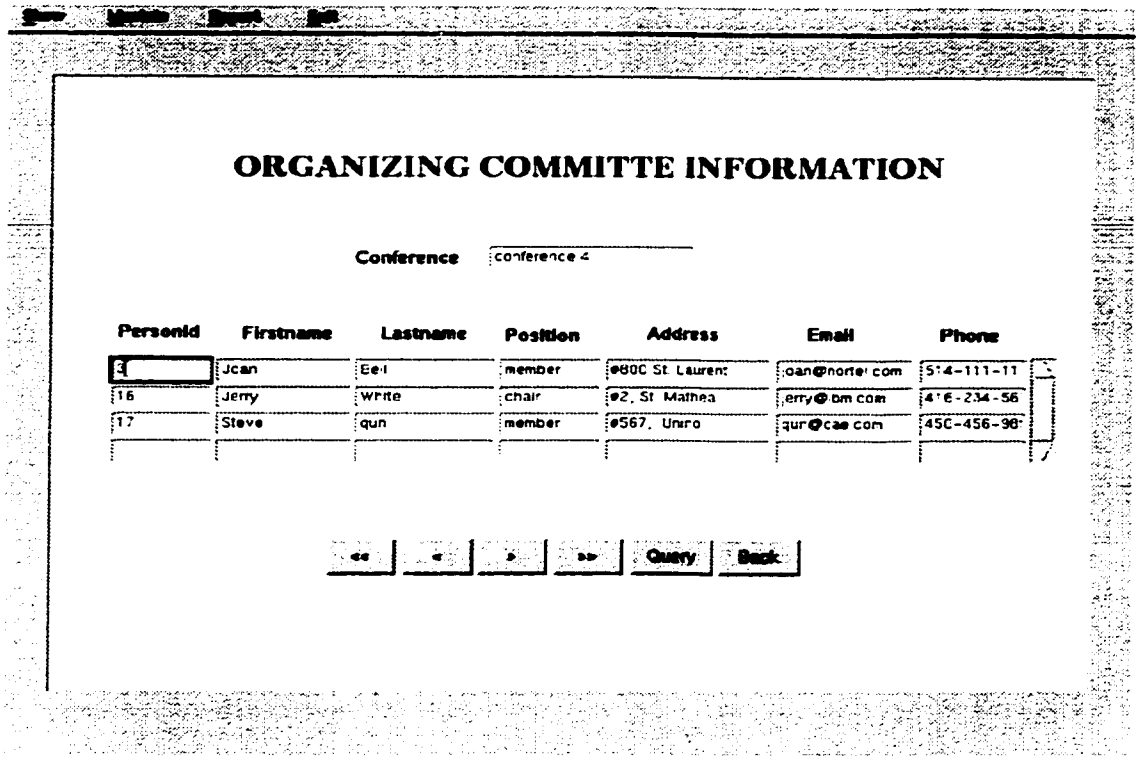


Figure 72: Organizing Committee Information

In the **Organizing Committee Information** window, user may browse the information of this committee members for a given conference.

In the **Program Committee Information** window (Figure.73), user may browse the information of this committee members for a given convocation.

In the **Attendee Information** window(Figure.74), user may browse the information of attendees for a given convocation.

When user browse the information of attendees without giving a convocation, a warn message is displayed to remaind user to choose a convocation title (Figure 75).

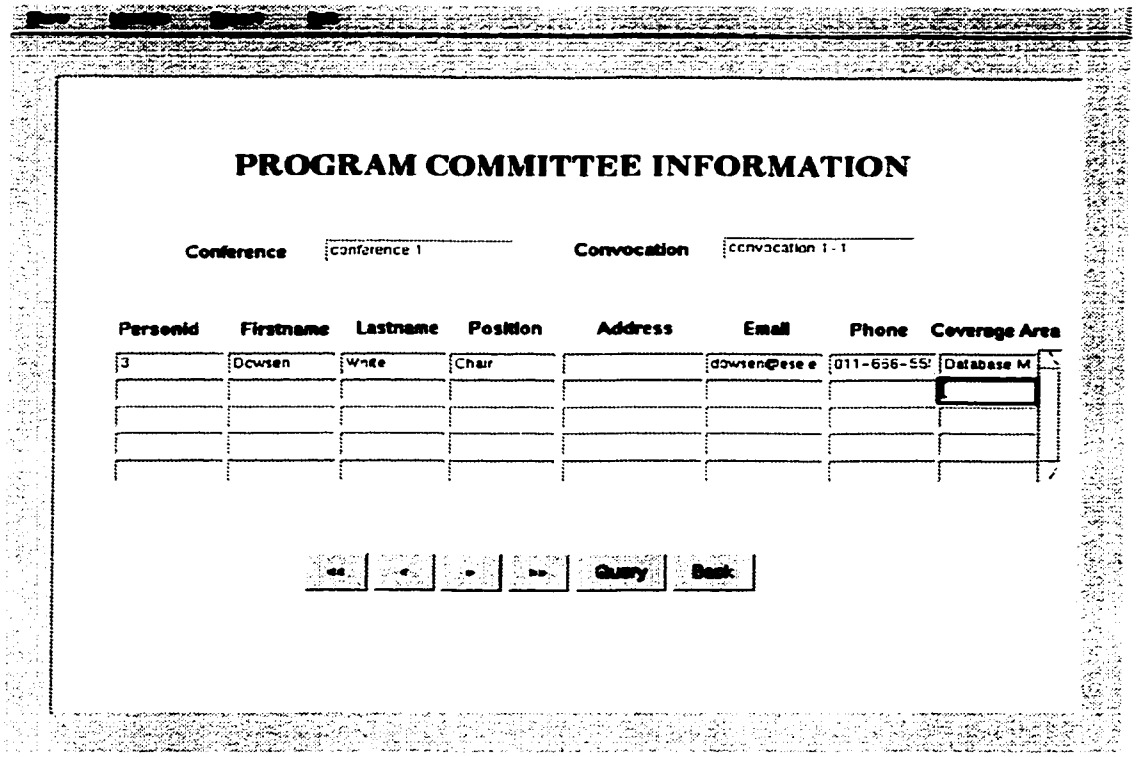


Figure 73: Program Committee Information

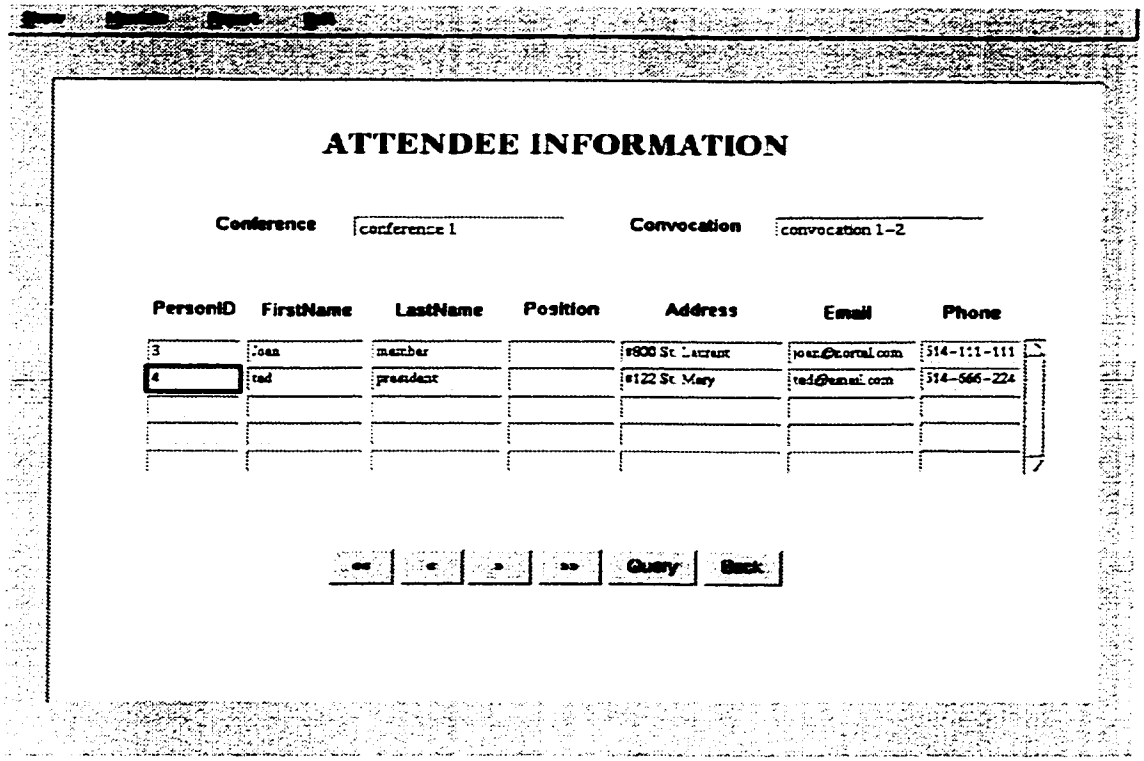


Figure 74: Attendee Information

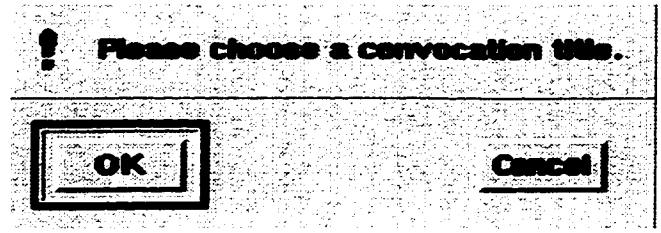


Figure 75: Warning Message

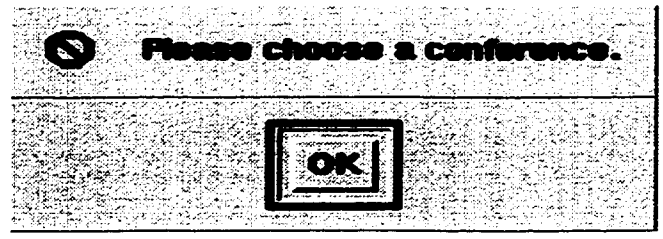


Figure 76: Warning Message

When user want choose a conference title without giving a conference, a warn message will be displayed to remained user to choose a conference title first (Figure 76).

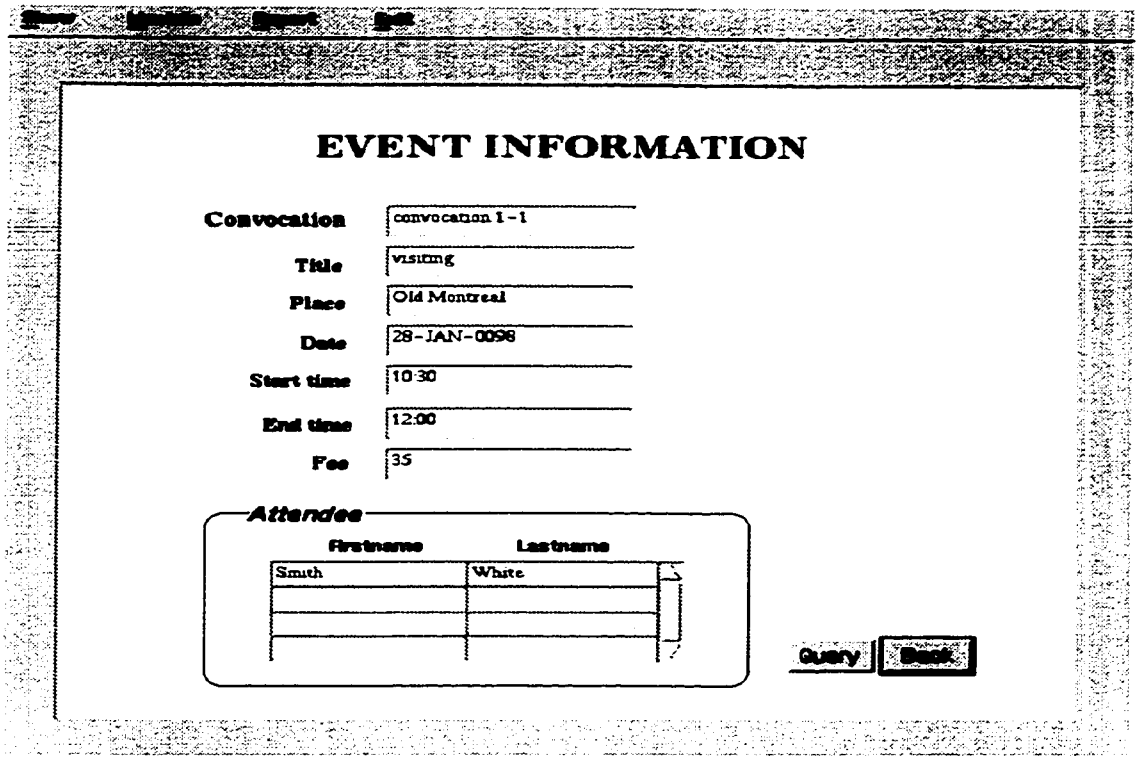


Figure 77: Event Information

In the **Event Information** window(Figure. 77), user may browse the information of events with its attendees for a given convocation.

15. Find the details of paper and publication.

In the **Paper Information** window(Figure. 78), user may browse the information of events with their attendees for a given convocation and may switch to window(Figure. 61).

In the **Publication Information** window(Figure. 79), user may browse the information of publications, their containing papers and their publisher.

16. Find details of account of attendees.

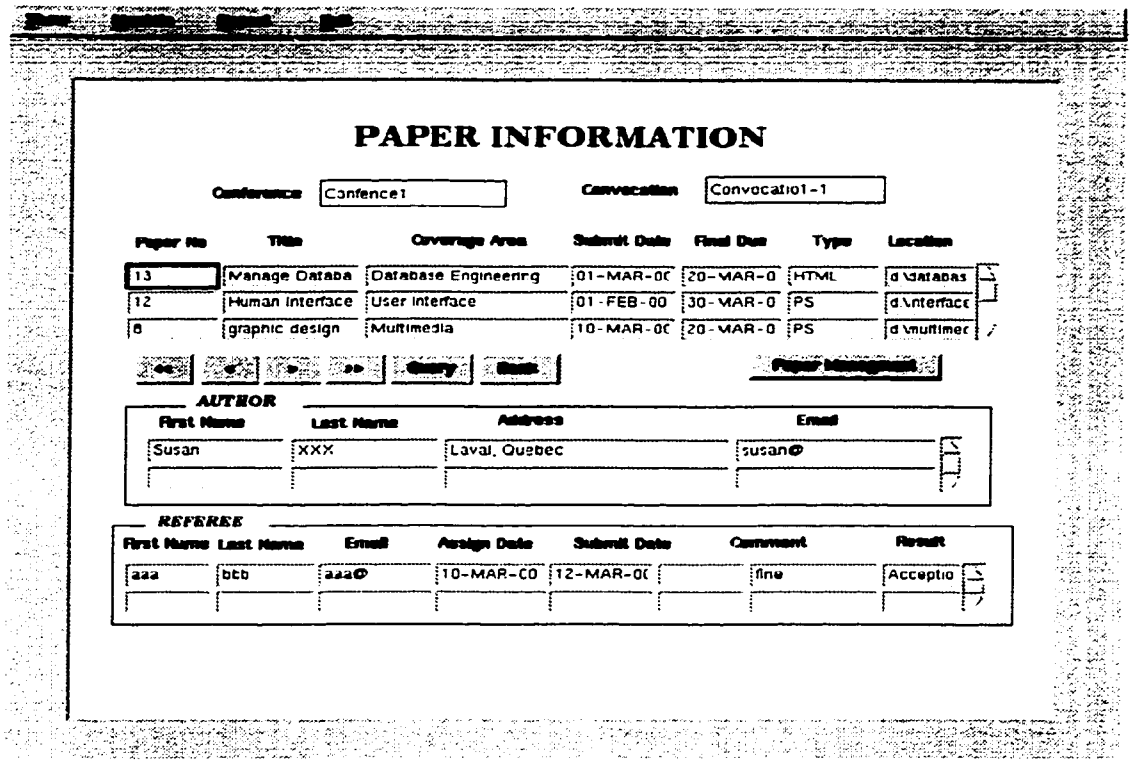


Figure 78: Paper Information

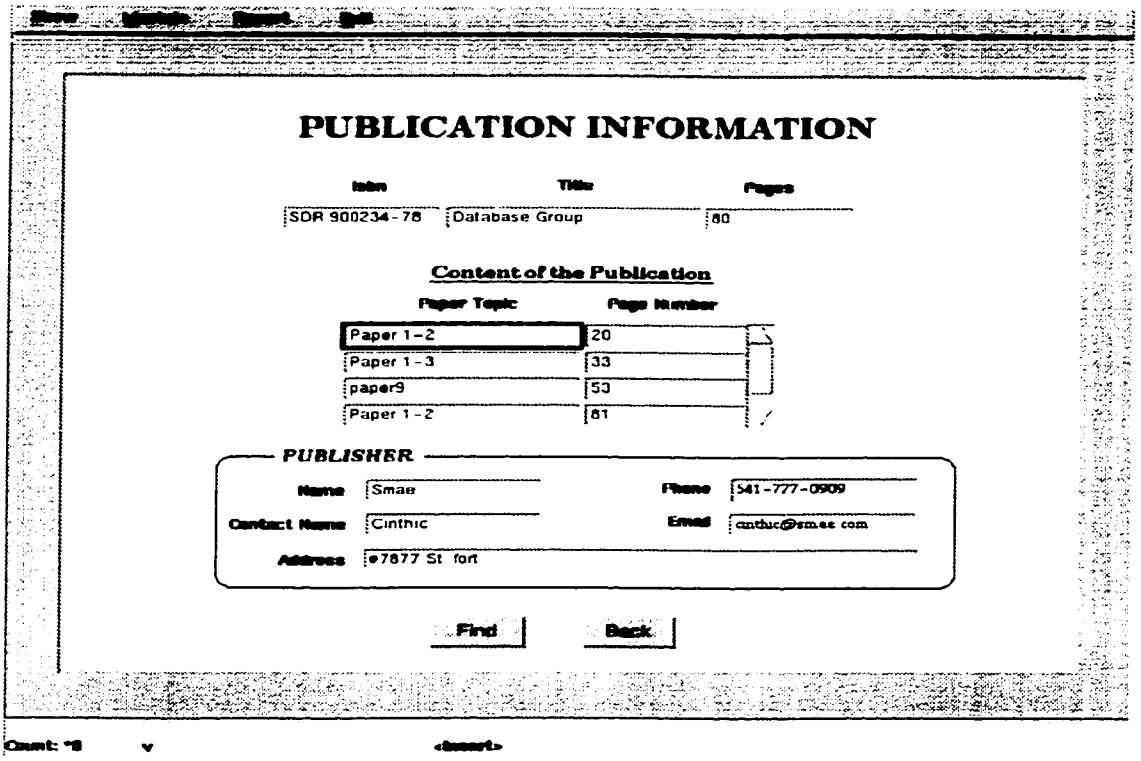


Figure 79: Publication Information

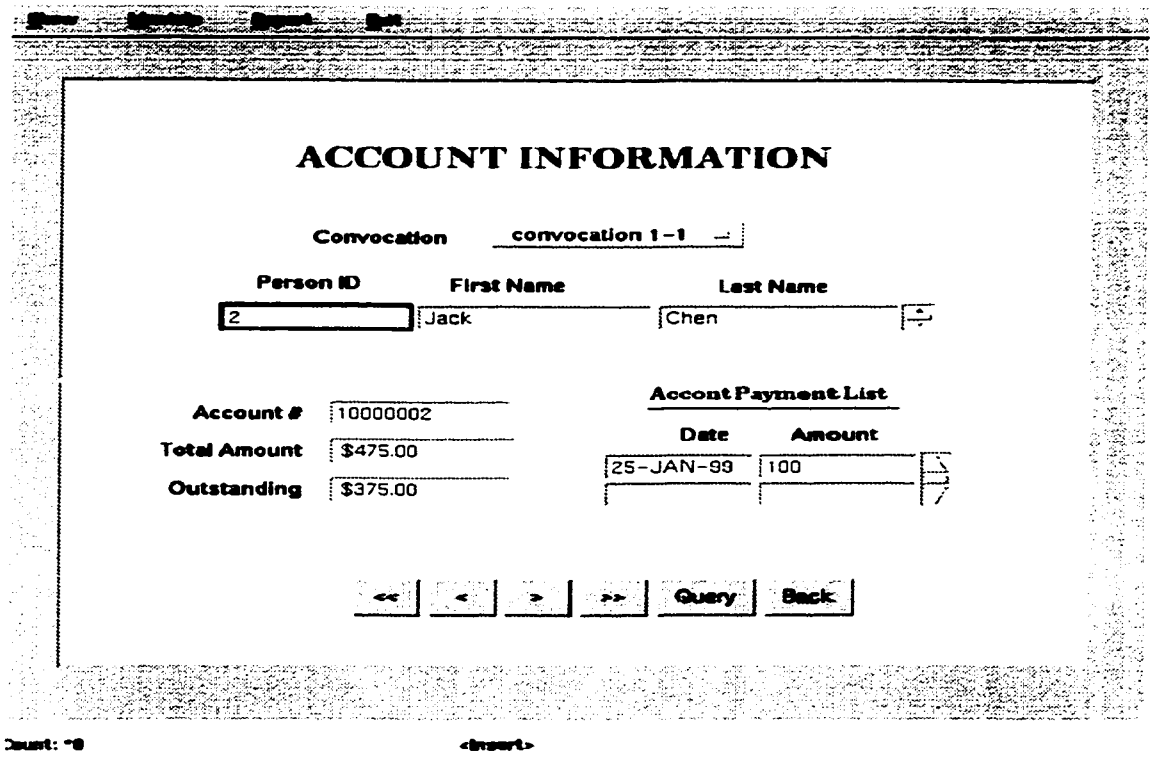


Figure 80: Account Information

In the **Account Information** window(Figure. 80), user may browse the information of attendees' account and their detail payment lists for a given conference.

7.5 Source Code

Following is parts of source code of this project.

```
*****************************
*       Mod_Paper.fmb       *
*****************************

-------Add/Update_Button--------
declare
  v_count number;
  v_sno c_session.sno%type;
  v_author number;
  ret number;
begin
  if :modpaper_blk.paperid is null or
     :modpaper_blk.title is null or
     :modpaper_blk.submit is null
  then
     ret := show_alert('alert1');
  else
  select count(*) into v_count
  from c_paper
  where paperno = :modpaper_blk.paperid;

  select sno into v_sno
  from c_session
  where title = :modpaper_blk.session;

  select count(*) into v_author
  from c_author
  where paperno = :modpaper_blk.paperid;
```

```
if v_count = 0 then
  if :modpaper_blk.session is null then
     message('Please choose a session');
  else
  --***insert into c_paper table
    insert into c_paper
    values(:modpaper_blk.paperid, v_sno,
           :modpaper_blk.title, :modpaper_blk.coverage,
           :modpaper_blk.submit, :modpaper_blk.final);
    message('One paper record added!');

   commit;
   end if;

  --***insert into c_author table
  go_block('modauthor_blk');

  if :modauthor_blk.name is not null then
  first_record;
 loop
  insert into c_author
  values(:modpaper_blk.paperid, :modauthor_blk.name,
         :modauthor_blk.address, :modauthor_blk.email);
  commit;
  next_record;

  exit when :modauthor_blk.name is null;
  end loop;
  previous_record;
  else
       ret := show_alert('alert2');
  end if;
  message('One record added');

  else
  update c_paper
  set title = :modpaper_blk.title, coveragearea = :modpaper_blk.coverage,
      submitdate = :modpaper_blk.submit, finaldue = :modpaper_blk.final
  where paperno = :modpaper_blk.paperid;
```

```
     commit;
     message('One record updated!');

    if v_author <> 0 then
      delete c_author
      where paperno = :modpaper_blk.paperid;
    end if;

    go_block('modauthor_blk');
    if :modauthor_blk.name is not null then
     first_record;
    loop
     insert into c_author
     values(:modpaper_blk.paperid, :modauthor_blk.name,
            :modauthor_blk.address, :modauthor_blk.email);
     commit;
     next_record;
     exit when :modauthor_blk.name is null;
     end loop;
     previous_record;
     end if;
   end if;
  end if;
end;



----------Find_Button------------

declare
  lov boolean;
begin
  lov := show_lov('paper');
  if :modpaper_blk.paperid is not null then
  findauthor;
  select title into :modpaper_blk.session from c_session
  where sno in (select sno from c_paper
                where paperno = :modpaper_blk.paperid);
  end if;
end;
```

```
----------Session_Button---------

declare
 lov_return boolean;
begin
 lov_return := show_lov ('session');

end;

********************************
*        Result.fmb            *
********************************

----------Query_Button---------
PROCEDURE pro_referee IS
BEGIN
  declare
   v_referee  c_referee.referee%type;
   v_count_reject number;
   v_count_accept number;
   ret number;
   ret1 number;
   ret3 number;
   v_count  number;
   v_n  real := 0.0;
   v_number number := 0;
   cursor refcursor is
   select distinct referee from c_referee group by referee;
  begin

   if :refresult_blk.presult is null or :refresult_blk.n is null then
   ret := show_alert('alert1');
   else
   go_block('result_blk');
   open refcursor;
  LOOP
   fetch refcursor into v_referee;
   exit when refcursor%NOTFOUND;
```

```
    select count(*) into v_count
    from c_referee
    where referee = v_referee;

    if :refresult_blk.presult = 'Reject' then
    select count(*) into v_count_reject
    from c_referee
    where referee = v_referee and result = 'Rejection';

    v_n := (v_count_reject/v_count) * 100;
    end if;

    if :refresult_blk.presult = 'Accept' then
    select count(*) into v_count_accept
    from c_referee
    where referee = v_referee and result = 'Acception';

    v_n := (v_count_accept/v_count) * 100;
    end if;

    if v_n > To_number(:refresult_blk.n) then
     :result_blk.referee := v_referee;
     next_record;
     v_number := v_number + 1;
    end if;

    END LOOP;
    close refcursor;
    previous_record;

    if v_number = 0 then
    ret1 := show_alert('alert2');
    end if;
    end if;
    end;
END;

*******************************
*   Mod_Conference.fmb        *
*******************************
```

```
****Convocation Management****

-------Add/Update_Button------
declare
 v_count number;
 v_conno c_conference.conno%type;
 ret number;
begin
 select count(*) into v_count from c_convocation
 where convono = :modconvo_blk.convono;

 select conno into v_conno from c_conference
 where title = :modconvo_blk.conf;

 if v_count = 0 then
  if :modconvo_blk.conf is null or
     :modconvo_blk.convono is null or
     :modconvo_blk.convotitle is null
  then
    ret := show_alert('alert2');
  else
    insert into c_convocation
    values (:modconvo_blk.convono, v_conno, :modconvo_blk.convotitle,
            :modconvo_blk.csdate, :modconvo_blk.cedate,
            :modconvo_blk.convolieu);
    message('One record added');
  end if;
 else
  update c_convocation
  set title = :modconvo_blk.convotitle,
      csdate = :modconvo_blk.csdate,
      cedate = :modconvo_blk.cedate,
      lieu = :modconvo_blk.convolieu
  where convono = :modconvo_blk.convono;
  message('One record updated');

 end if;
commit_form;
end;
```

```
---------Find_Button--------
declare
 lov_return boolean;
 v_conno c_conference.conno%type;
 v_count number;
 ret number;
 ret1 number;
begin
 if :modconvo_blk.conf is null then
  ret1 := show_alert('alert4'); --choose conference
 else
 lov_return := show_lov ('convocation');

 select title into :modconvo_blk.conf from c_conference
 where conno = :modconvo_blk.conno;
end if;
end;

******Program Mangement******

------Add/Update_Button------
declare
 v_count number;
 v_convono number;
 ret number;
begin
 select count(*) into v_count
 from c_person
 where personid = :modpro_blk.memberid;

if v_count = 0 then
 if :modpro_blk.memberid is null or
    :modpro_blk.fname is null or
    :modpro_blk.lname is null or
    :modpro_blk.memposition is null
 then
    ret := show_alert('alert1');
 else
```

```
   insert into c_person
   values(:modpro_blk.memberid, :modpro_blk.fname, :modpro_blk.lname,
          :modpro_blk.memposition, null,
          :modpro_blk.mememail, :modpro_blk.memtel);
   commit;
   message('One person added');

   select convono into v_convono
   from c_convocation
   where title = :modpro_blk.convo;

   insert into c_program
   values (v_convono, :modpro_blk.memberid);
   commit;
 end if;
 else
 update c_person
 set firstname = :modpro_blk.fname,
     lastname = :modpro_blk.lname,
     position = :modpro_blk.memposition,
     email = :modpro_blk.mememail,
     phone = :modpro_blk.memtel
     where personid = :modpro_blk.memberid;
 commit_form;
 end if;
end;

*******************************
*        Mod_Account.fmb      *
*******************************

----------Add_Button----------

declare
  pid   number;
  v_count number;
  v_count1 number;
  v_account c_account.account#%type;
begin
   if :modacc_blk.account# is null then
```

```
    message('Please choose account information.');
    else
    select count(*) into v_count
    from c_account
    where personid = :modacc_blk.personid;

    if v_count <> 0 then
    update c_account
    set accofee = :modacc_blk.accofee
    where account# = :modacc_blk.account#;
    else
    insert into c_account
    values (:modacc_blk.account#, :modacc_blk.personid,
            :modacc_blk.accofee,  :modacc_blk.eventfee);
    end if;
    commit;

    if :modacc_blk.pdate is not null and
       :modacc_blk.pamount is not null then

    select count(*) into v_count1
    from c_payment
    where account# = :modacc_blk.account# and
          paydate = :modacc_blk.pdate and
          payamount = :modacc_blk.pamount;

    if v_count1 = 0 then
    insert into c_payment
    values(:modacc_blk.account#, :modacc_blk.pdate, :modacc_blk.pamount);
    else
     message('This record already exist.');
    end if;
    end if;
    commit;
    end if;
end;

--------Delete_Button-------
delete c_payment
where account# =  :modacc_blk.account#;
```

```
delete c_account
where account# = :modacc_blk.account#;

commit;
clear_form;

---------Find_Button---------
declare
  lov_return boolean;
begin
  lov_return := show_lov('account');-- Account list

  if :modacc_blk.personid is not null then
   select firstname, lastname
   into :modacc_blk.firstname, :modacc_blk.lastname
   from c_person
   where personid = :modacc_blk.personid;

   :modacc_blk.pdate := '';
   :modacc_blk.pamount := '';
  end if;
end;

------Delete_List_Button------
delete c_payment
where account# = :modacc_blk.account# and
      paydate = :modacc_blk.pdate and
      payamount = :modacc_blk.pamount;
commit_form;

:modacc_blk.pdate := '';
:modacc_blk.pamount := '';


------Find_Account_List-------
declare
 lov_return boolean;
begin
 if :modacc_blk.account# is not null then
```

```
 lov_return := show_lov('list');  -- Account List
 end if;
 if lov_return is null then
 message('The person' || :modacc_blk.firstname || :modacc_blk.lastname||'is n
 end if;
end;


*****************************
*     Laterreferee.fmb      *
*****************************

---------Query_Button--------
declare
  v_referee      c_referee.referee%type;
  v_paper        c_paper.title%type;
  v_assigndate   c_referee.assigndate%type;
  v_submitdate   c_referee.submitdate%type;
  cursor mycursor is
  select  r.referee, p.title, r.assigndate, r.submitdate
  from c_referee r, c_paper p
  where r.submitdate > r.assigndate + 7 and p.paperno = r.paperno;
begin
  go_block('lateref_blk');
  clear_block;
  open mycursor;
  loop
  fetch mycursor into v_referee, v_paper, v_assigndate, v_submitdate;
  exit when mycursor%NOTFOUND;
  :lateref_blk.referee := v_referee;
  :lateref_blk.paper := v_paper;
  :lateref_blk.assigndate := v_assigndate;
  :lateref_blk.submitdate := v_submitdate;

  next_record;
  end loop;
  close mycursor;
  previous_record;
end;
```

Chapter 8

Summary and Future Work

8.1 Summary

Oracle Developer/2000 features a point-and-click interface so it is easy to use; in addition, it has intelligent defaulting, user extensibility, portablility, integration and the capability to access non-Oracle database. These features are unique among the tools available that access Oracle databases. Although some of these features might exist in other products, no other products feature all these attributes combined.

However, there are also some shortcoming:

1. Oracle Developer Form 4.5 is not very convenient. Visual Basic or Java is better choice for designing graphical user interface.

2. There is slight problem of color saturation when interface designer wish to get desire effect of the color.

This project covers a brief introduction of Oracle Developer 2000, it mainly describes Form 4.5 and Report 2.5 tools and how to use them to develop a database application, like Object Navigator, Layout editor, Lov, Various 3D buttons that can be used to perform functions. Through the use of menus, module and other functionality, Developer/2000 is a complete database management tool. Application of Conferece Management System has been developed by using Form 4.5 and Report 2.5, it demostrates how the Oracle Developer/2000 tools can be used to generate and compile

modules of forms, interfaces, triggers, procedure units and reports.

Many times, users need reports that summarize information available on screen, the project also provides ability to display reports based on the date stored on the database.

8.2 Future Work

1. Menu security features let user specify which operators will be allowed to use which menus. User can grant operators access at two levels:

   - access to all of the menu items in a module;
   - access to specified menu items only.

   When the current operator does not have access to certain menu items, they can be hidden completely or displayed as disabled.

   By using menu security features, user can avoid having to create multiple applications that perform same functions. For example, two departments in a company might both use the same application, but employees in the first department can be given access to parts of the application that can not be accessed by employees in the second department.

2. Another feature of Developer/2000 is that all of its components have been designed to be integrated. Oracle Graphics 2.5 is an important part of Developer 2000, which enables Oracle developers to present graphical representations of Oracle data. Graphical research provides an impact that cannot be achieved with mere textual views of numerical data.

   This project has not addressed this issue. Although Oracle Graphics will produce these charts in a standalone mode, the true power of this tools is as an OLE 2 server application that can be incorporated into OLE container applications such as Visual Basic, Excel, and, of course, Oracle Forms and Oracle

Reports. Furthermore, CMS can use graphic to show various information of conference management.

Appendix A

Bibliography

[Ora97] Scott Urman, " Oracle8 PL/SQL Programming", *Oracle Press*, 1997

[Data96] Thomas M.Connolly, Carolyn E. Begg, Anne D. Strachan, "Database Systems, A Practical Approach to Design, Implementation and Management",*Addison-Wesley*, 1996.

[OnTug] Oracle Tutorial, ugweb.cs.ualberta.ca/ oracle/

[OnDor] Online document for Oracle, www.oracle.com.sg/products/tools/dev2k/

[OraF95] Oracle Corporation, "Oracle Developer 2000 Form 4.5 Guide Manual", 1995

[OraR95] Oracle Corporation, "Oracle Developer 2000 Form 4.5 Reference Manual", 1995

[OraS95] Oracle Corporation, "Oracle 7 Server Application", 1996

[Data94] Toby J. Teorey, "Database Modeling & Design", 1994

[OraU96] Rachel Bechker, Orace[tm] Unleashed, *SAMS Publishing* 1996