

2001

Meeting the Challenge of Dynamic User Requirements Using Data-Driven Techniques on a 4GL-Database Environment

Christopher Bolan
Edith Cowan University

Follow this and additional works at: https://ro.ecu.edu.au/theses_hons



Part of the [Software Engineering Commons](#)

Recommended Citation

Bolan, C. (2001). *Meeting the Challenge of Dynamic User Requirements Using Data-Driven Techniques on a 4GL-Database Environment*. https://ro.ecu.edu.au/theses_hons/847

This Thesis is posted at Research Online.
https://ro.ecu.edu.au/theses_hons/847

Edith Cowan University

Copyright Warning

You may print or download ONE copy of this document for the purpose of your own research or study.

The University does not authorize you to copy, communicate or otherwise make available electronically to any other person any copyright material contained on this site.

You are reminded of the following:

- Copyright owners are entitled to take legal action against persons who infringe their copyright.
- A reproduction of material that is protected by copyright may be a copyright infringement. Where the reproduction of such material is done without attribution of authorship, with false attribution of authorship or the authorship is treated in a derogatory manner, this may be a breach of the author's moral rights contained in Part IX of the Copyright Act 1968 (Cth).
- Courts have the power to impose a wide range of civil and criminal sanctions for infringement of copyright, infringement of moral rights and other offences under the Copyright Act 1968 (Cth). Higher penalties may apply, and higher damages may be awarded, for offences and infringements involving the conversion of material into digital or electronic form.

USE OF THESIS

The Use of Thesis statement is not included in this version of the thesis.

**Meeting the Challenge of Dynamic User Requirements Using
Data - Driven Techniques on a 4GL - Database
Environment.**

By

Christopher Bolan BSc (Computer Science)

**A Thesis Submitted in Partial Fulfilment of the
Requirements for the Award of**

Bachelor of Science Honours (Software Engineering)

Faculty of Communications, Health and Science

Edith Cowan University

Date of submission – February 2001

Abstract

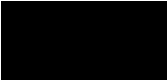
Accompanying the ever-growing reliance on computers within contemporary organisations, the task of software maintenance is, increasingly, becoming a resource burden. The author has identified that there is a need for proven techniques to allow the modelling of flexible/changing user requirements, to enable systems to cope with requirements creep without suffering major code change and associated down-time from rebuilds of the database.

This study ascertains the applicability of extension to current data modelling techniques that allows innate flexibility within the data model. The extension of the data model is analysed for potential benefits in the provision of such a dynamic/flexible base to realise 'maintenance friendly' systems and, in consequence, alleviate the cost of later, expensive maintenance.

Declaration

I certify that this thesis does not, to the best of my knowledge and belief:

- (i) incorporate without acknowledgement any material previously submitted for a degree or diploma in any institution of higher education;
- (ii) contain any material previously published or written by another person except where due reference is made in the text; or
- (iii) contain any defamatory material

Signed:  _____

Date: 19/3/2001 _____

Christopher Bolan

Acknowledgements

I would like to take this opportunity to acknowledge and thank all the people who have helped and assisted me in this work. However there are some people to whom I wish to express particular gratitude.

Thank you to my supervisor, Michael Collins, for all your time and effort, I couldn't have done this without you. "Unus, sed leo!" [One, but a lion!]- Aisopos (Fabulae 194). Also, thanks must go to Celia for letting me borrow him.

Thank you to my family: to Mum, for always believing in me and giving me so much love and support; to Dad, for all your guidance and for giving me a hunger for knowledge, without which I could have never become who I am today; and to my brother Michael just for being himself.

Thank you to all my friends who have been there when I needed support or to let off steam. Especially Johanna, Jeremy, Troy, Cameron, David, Katie, Dominique, Rachael, Sarah, Bonnie, Todd, Joanne, Nikki and Mr 'M' for letting me crash at his pad.

Thank you to all the teachers that have inspired me through the years: especially to Jean Hall, Penny Cookson, Julian Terry, and John Doyle.

And finally to the Authors whose books have inspired me and fuelled my imagination, as Albert Einstein once said: "Imagination is more important than knowledge".

Table of Contents

Abstract	i
Declaration	ii
Acknowledgements	iii
Table of Contents	iv
Table of Figures	vi
Chapter 1: Introduction	1
1.1 Background to the study.....	1
1.2 Significance of the Study	6
1.3 Purpose of the Study	9
1.4 Research Questions	11
1.5 Summary	11
Chapter 2: Review of the Literature.....	14
2.1 General Literature	14
2.2 Specific Studies Similar to the Current Study.....	18
2.3 Other Literature of Significance to this Study	23
2.3.1 Normalisation	23
2.3.2 Entity Relationship Modelling.....	25
2.3.3 Native Oracle Features used to facilitate flexibility	28
2.3.3.1 %TYPE	28
2.3.3.2 Setting Properties Programmatically	30
2.3.3.3 Dynamic Record Groups.....	31
2.3.3.4 List of Values	32
2.4 Summary	33
Chapter 3: Research Method.....	34
3.1 The Problem	34
3.2 E-R Analysis	36
3.3 Database Design.....	36
3.4 Application Development	37
3.5 Oracle Environment	38
3.6 Summary	39
Chapter 4: The Demonstration Application.....	40
4.1 The Database.....	40

4.1.1 Initial Design	40
4.1.2 Flexible Design.....	44
4.1.3 Creating the database.....	48
4.2 The Application.....	48
4.2.1 Flexible Setup.....	49
4.2.1.1 Category Maintenance	50
4.2.1.2 Subcategory Maintenance	51
4.2.1.3 Composition Rule Maintenance.....	57
4.2.2 Data Entry.....	59
4.2.2.1 Dynamic Data Entry	59
4.2.2.2 Static Data Entry	69
4.2.3 Reporting	71
4.3 Summary	76
Chapter 5: Findings.....	77
5.1 Findings on Research Question, part (a).....	77
5.2 Findings on Research Question, part (b).....	79
Chapter 6: Conclusion.....	81
Glossary.....	83
Bibliography.....	86
Appendix A - Data Collection Form	89
Appendix B - Collection Category Setup.....	92
Appendix C - Database Creation Scripts.....	98
CEEDG13.SQL.....	98
CEEDG13.TAB.....	98
CEEDG13.IND	106
CEEDG13.CON	109
CEEDG13.SQS	114
Appendix D - Update Layout Procedure	116
Appendix E - Static Data Entry Screens	120
Personal Details.....	120
Change Password	120
Discipline Memberships.....	121
Research Memberships	121
Appendix F - Sample Reports	122

Table of Figures

Figure 1. Distribution of Maintenance Effort (Lientz & Swanson, 1980)	3
Figure 2. Distribution of Maintenance Activities (Stacey, 1995, p.1)	3
Figure 3. Relative Cost of Change (Oracle, 2001).....	7
Figure 4. The dynamic search condition.....	17
Figure 5. SQL Description of Concept HR codes tables (Courtesy of Concept Systems).....	19
Figure 6. ER Diagram of 'Front End' setup tables	21
Figure 7. Application E-R Diagram (O'Connor, 1999, p.36)	22
Figure 8. A sample ER diagram containing three entities: Department, Employee and Tasks.....	26
Figure 9. The mapping of Chen's relationships (Hall, 1998, p.24)	27
Figure 10. The flex database table.....	29
Figure 11. Utilising %TYPE in PL/SQL	29
Figure 12. Calling a LOV using PL/SQL	32
Figure 13. Initial ER-Model of system	41
Figure 14. Description of the BOOK table	43
Figure 15. ER - model of category data.....	45
Figure 16. Storage requirements of category items	46
Figure 17. Dynamic ER-Model	47
Figure 18. System maintenance flow.....	49
Figure 19. Description of the CATEGORY table.....	50
Figure 20. Category maintenance form	50
Figure 21. Description of the SUB_CATEGORY table	52
Figure 22. Subcategory maintenance screen flow	52
Figure 23. Subcategory ownership form.....	53
Figure 24. Add subcategory form (format 1).....	54
Figure 25. Add subcategory form (format 2).....	55
Figure 26. View subcategory details form (format 1).....	56
Figure 27. View subcategory details form (format 2).....	56
Figure 28. Composition rules maintenance form.....	57
Figure 29. Column mapping table	58
Figure 30. Staff return entry flow.....	60
Figure 31. Description of the STAFF_RETURN	60
Figure 32. Choose Collection Period form.....	61
Figure 33. Description of RETURN_ITEMS	62
Figure 34. View Return Items - base level form.....	63
Figure 35. Update Layout pseudo code	64
Figure 36. Choose Category form.....	65
Figure 37. Choose Category form - Select a category	66
Figure 38. Choose Category Record Group.....	66
Figure 39. Choose Category form - Select a sub category.....	67
Figure 40. Choose Subcategory Record Group	67
Figure 41. View Return Items - possible configuration 1	68
Figure 42. View Return Items - possible configuration 2.....	68
Figure 43. Static data entry flow.....	70
Figure 44. SQL query used to define report structure.....	73
Figure 45. CF_DATA pseudo code	74
Figure 46. Oracle Reports data model	75

Chapter 1: Introduction

This chapter contains an introduction to the study, describing software maintenance, why it has become a significant focus of the software engineering discipline and, further, how this focus had led to an increased awareness of flexibility as an important design goal. The aims of the study are itemised, and the research questions are stated.

1.1 Background to the study

Contemporary organisations desiring to remain competitive must review the pace and manner in which they conduct day-to-day business. Increasingly, as Callon (1996, p.106) reports, they rely upon Information Technology (IT) to adapt to the rapidity of change at large and to maintain competitive advantage.

Furthermore, as found by Hall & Ligezinski, (1997a, p.1), the costs underlying provision of IT services are now coming under renewed scrutiny. This is especially true where contemporary development environments are faced with the dual problems of increasing IT resource costs and tighter fiscal management trends.

Maintenance of software brings an ever-increasing cost to a typical organisation's IT budget. Organisations are reluctant to release exact figures, but Pressman (1997 p.762), cites trends that show a steady increase in such costs from thirty five percent of the total project budget in the 1970s to sixty percent in the 1980s. Pressman's observed trends are acknowledged by McCracken (cited in Pressman, 1997, p.762),

whose studies indicate that if the trend of increasing maintenance cost continued, as identified in the 1980s, then organisations may become ‘maintenance bound’.

An organisation is described as ‘maintenance bound’ where it is so buried in maintenance that there are no available resources to assign to new projects (Pressman, 1997, p.762). The author suggests that with today’s phenomenon where entire IT organisations consist of employees principally occupied by maintenance of existing software systems, McCracken’s ‘future’ may have become the reality.

Pressman’s (1997) supposition is re-enforced by the findings of Hall and Ligezinski (1997a, p.2), who estimate the cost of software maintenance as being between sixty and seventy percent of an organisation’s total software development budget.

The cost of maintenance may be understood better through the decomposition of maintenance into three main categories as described by Swanson (cited in Pressman, 1997, p.763):

- **Corrective Maintenance:** the identification and correction of software errors, commonly known as ‘bug fixing’.
- **Adaptive Maintenance:** modification of existing software to conform to changing requirements.
- **Perfective Maintenance:** adds new capabilities, modifies existing functions.

The survey conducted by Lientz and Swanson (1980), who sampled 487 members of the Data Processing Managers Association, revealed a distribution of the above categories, illustrated in figure 1:

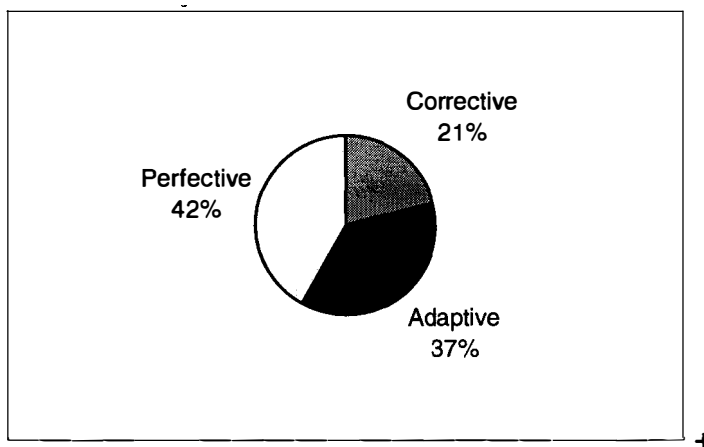


Figure 1. Distribution of Maintenance Effort (Lientz & Swanson, 1980)

Figure 1 illustrates that the largest combined portion of maintenance effort, and thus spending, is a result of adaptive and perfective maintenance. Inspection suggests that any effort in improving the maintainability of a system should focus on the perfective and adaptive areas.

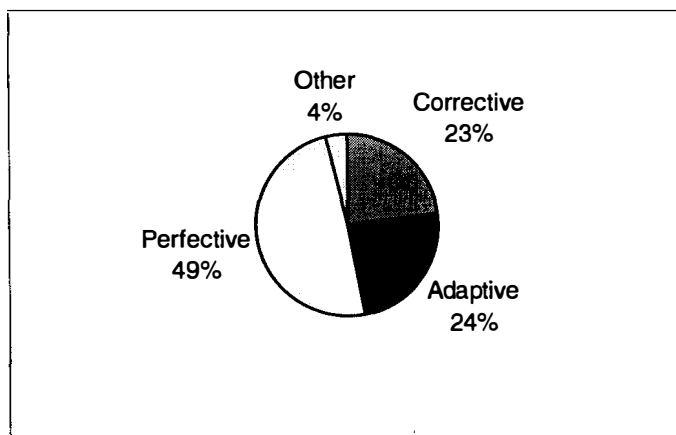


Figure 2. Distribution of Maintenance Activities (Stacey, 1995, p.1)

It may be seen that the trend has not altered significantly in recent times, with a study conducted by Stacey (1995, p.1) showing a similar breakdown, illustrated in Figure 2, in the focus of maintenance activities.

The above studies into the distribution of maintenance activities measure the relative effort of performing the maintenance activities and focus on the visible cost.

However, according to Stacey (cited in O'Connor, 1999, p.6) the "hidden costs of maintenance can be even greater" due to:

- Loss or postponement of development opportunities.
- Customer dissatisfaction at not having their needs met.
- Reduction in software quality due to maintenance introduced errors.

By example, Woolfolk, Ligezinski and Johnson (1996, p.482) cite the case of an unnamed American factory where management had decided to increase the efficiency of the organisation by removing middle management and altering the departmental structure. Within six to eight weeks most of the effected personnel had adapted to the changes in procedures and formed new communication lines and work practices. However, Woolfolk et al (1996, p.483) observed that, after a year the "computer systems were only 90 percent complete at a cost exceeding a quarter of a million dollars".

The literature offers two significant approaches, namely, the employment of flexible software, and that of end user development, to combat the trend of increasing maintenance costs. Each of these will now be introduced.

End user development permits end users to assume the role of developers, thereby taking full responsibility for the creation of their own applications (Mehandjiev & Bottaci, 1998, p.3). Not everyone, however, sees end user development as a panacea. Panko (1998, p.16) found wide criticism for it in the software community due to an increased likelihood of errors derived from informal methods wanting of the rigour known to be necessary in programming. Such rigour, however, may be achieved within professionally developed systems that are designed to incorporate flexibility during use.

Mehandjiev and Bottaci (1996, p.432) see flexible software as helping to bridge the gap between developers and advanced users by allowing the latter to access features that control and modify the behaviour of a system. Likewise, O'Connor (1999, p.7) suggests that "flexible software can be seen as the middle ground between professional IT staff maintained systems and end user development". Through the use of flexible software, Mehandjiev and Bottaci foresee an advantage, over conventional system development methods, for organisations to adapt applications rapidly to changing requirements.

Woolfolk, Ligezinski & Johnson (1996, p.486) propose that flexible systems fall into one of three categories of flexibility with respect to adaptive maintenance:

- Weak flexibility: requires modification to the underlying data structures (e.g. entities and/or tables).
- Medium flexibility: requires modification to both data values and procedural code.
- Strong flexibility: where data value modifications alone are required.

The author suggests that strong flexibility, appropriate to the needs of contemporary organisations, may be achieved by integrating flexible principles into the underlying database design of an application. Such flexibility may then help to facilitate a cost reduction in both time and effort of eventual maintenance, especially in the activities of perfective and adaptive maintenance. The savings to be gained from flexible development might, reasonably, permit funds to be allocated towards new developments and decrease the risk of an organisation becoming 'maintenance bound'.

1.2 Significance of the Study

The relative cost of implementing change in a non-flexible system, illustrated in figure 3, is greatly increased during each phase of the software development lifecycle. It is, therefore, important to focus on the provision of flexibility as part of the requirements capture/analysis phase of a project, in order to mitigate increased development costs resulting from a lack of such focus. The need for such early provision is reinforced by Weinberg (1990), whose study demonstrated the inclusion

of maintainability as a part of requirements capture led to an increased awareness of maintenance issues throughout an entire project lifecycle.

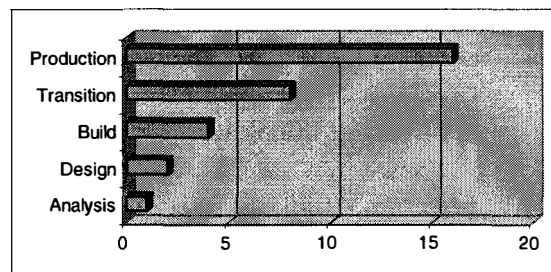


Figure 3. Relative Cost of Change (Oracle, 2001)

Requirements, as specified by Blum (1993a, p.43), fall into three categories as:

- **Closed:** defined and stable requirements.
- **Abstract:** requirements that have no 'concrete' representation, but which incorporate products necessary for development of the system. Primary examples of abstract requirements are 'security' and 'user friendliness'.
- **Open:** requirements where the problem domain is flexible. A flexible problem domain suggests that requirements may lack definition, or change, either during development or post-implementation.

It is the flexibility, or changeability, of 'open' requirements that flexible software techniques aim to address. However, as O'Connor (1999, p.9) notes, most applications are specified from requirements observed in all three, not from just one, of Blum's categories.

Many static systems, i.e. systems without in-built flexibility, are based on fixed requirements and use a traditional 'sign off' approach to validate the satisfaction of requirements. Frequently, though, difficulties in specification lie in that requirements, true at the time of the original specification change/evolve, exhibiting the phenomenon known as 'requirements creep'. To formalise the remedy, Behforooz & Hudson (1996, p.396) state that "maintainability should be specified and software should provide for the highest level of flexibility and ease of maintenance", giving the following reasons:

- Software maintenance is expensive.
- The advantages of including maintenance as a design goal far outweigh the costs.
- A system may spend between 65 and 80 percent of its life in maintenance.

Others endorse the difficulty of specifying requirements fully, as Woolfolk, Ligezinski & Johnson (1996, p.482) observed that such specifications are "imperfect since a significant part of such requirements lie in the future". Poor specification, in response to requirements creep may, in turn, "lead to systems that are judged unsatisfactory or unacceptable by the client and have high maintenance costs" (Hofmann, Pfeifer & Vinkhuyzen, 1993, p.43).

While studies, such as the Object Database Management Group's (ODMG) research into improved methods for requirements capture ("ODMG 2.0", 1998), software maintenance continues to be an area neglected in the systems development lifecycle. Such neglect is especially noticeable when maintenance phase support is compared to

that of other phases of the software lifecycle. As pointed out by Liu, Yang & Zedan (1998, p.1) “the approaches/tools of maintenance are rather weak when contrasted to those of development”. Liu, Yang & Zedan further suggest the underlying reason for the neglect is that software development is a mature process while maintenance is still viewed as being a difficult and expensive area.

Acknowledging the neglect in maintenance related studies, this study aims to demonstrate that existing static data requirement modelling techniques may be adapted to facilitate a reduction in maintenance effort though the inclusion of underlying flexibility. Outcomes of the study offer systems analysts and programmers a number of potential advantages:

- A flexible method of capturing data.
- A need to learn few (potentially zero) new symbols and/or notations.
- A minimisation of the effort involved in capture of flexible requirements.

1.3 Purpose of the Study

The previous sections indicated that there is credible support in the literature for flexibility as a design goal. However, as shall be shown, there is little clear statement in the literature on how this flexibility may be integrated into either the resultant application or, more importantly, the design model. The purpose of this study is to demonstrate how flexibility may be incorporated successfully into a ‘live’ system i.e. one that may exhibit Blum’s “open” or flexible requirements.

While flexible systems are not new, previous research (O’Connor, 1999; Layng, 1998) has focused on the adaptation of an application around a static data model. The

employment of a static data model has meant that while applications may exhibit features of a flexible system in the user interface (the so-called 'front end'), the type and volume of stored information (the so-called 'back end') remains static. The author proposes that development with a data model that exhibits a measure of intrinsic back end flexibility is an appropriate alternative.

The study focuses on the flexibility of data in the 'back end' of an application, making it appropriate to choose a contemporary data-oriented implementation language such as the Oracle 4GL. Additionally, selection of Oracle's 4GL allows comparison with studies of business-rule oriented front-end flexibility in that development environment, such as those conducted by Hall & Ligezinski (1997a), O'Connor (1999) and Layng (1998).

To summarise, the purpose of this study is to investigate the implementation of a dynamic system using the Oracle 4GL to provide flexibility in the 'front end' in conjunction with dynamic mapping to the underlying database 'back end'.

1.4 Research Questions

The main question:

"How may the challenge of dynamic/flexible user requirements be met using data driven techniques using a contemporary 4GL-database environment?"

The major components of the above question are: -

- a) *"How may one model dynamic user requirements through an extension of current data modelling techniques? "*
- b) *"How may one implement user requirements of flexibility using data driven techniques in a contemporary 4GL environment? "*

1.5 Summary

The rising cost of maintenance is an issue that has enjoyed little research effort when compared to other, better-established, phases of the software development lifecycle. While previous studies have indicated the need for corrective action, no notable progress has been achieved towards the realisation of improvements. This study demonstrates that, through an extension of existing techniques, a standard 4GL may be used to implement flexibility in both the 'front end' and 'back end' of an application.

Chapter two provides a review of the relevant literature. The review draws upon previous work, in textbooks, documented research papers and articles, to provide guidance and justification for the study's approach of adaptation/enhancement of existing modelling techniques. To further the study's implementation, existing

techniques that provide flexibility within an application are explored. Finally, documented features of the Oracle 4GL environment that facilitate system flexibility are identified.

Chapter three describes the research design. The methodology undertaken to answer the research questions proposed in chapter one is described, together with the Oracle environment employed to develop the demonstration application.

Chapter four outlines the flexible software techniques implemented using native Oracle features (to be introduced in chapter two), and details the usage of the demonstration application.

Chapter five presents the results and findings of the demonstration application. The findings are discussed in relation to the initial research criteria and compared to previous studies that were discussed in chapter two.

Chapter six concludes the study. A summary is given of the initial aims of the study and the manner in which they were addressed. Finally, the implications of the results and findings are discussed with respect to their benefits to current practice and to future research into flexible software.

This document concludes with a glossary of terms, followed by the end text references for documents used to support the study, and several appendices, which are

used to provide clarification and amplification of significant areas of the study: namely,

- Appendix A consists of the data collection form used by the manual submission system (the system is detailed in chapter three);
- Appendix B presents the category hierarchy with the required field configurations (the category hierarchy is detailed in chapter three);
- Appendix C lists the database creation SQL scripts (the database is introduced in chapter three and detailed in chapter four);
- Appendix D illustrates the PL/SQL code used in the Update Layout procedure (the Update Layout procedure is detailed in chapter four);
- Appendix E consists of screen captures of the demonstration application's static data entry screens (the static data entry screens are covered in chapter four); and
- Appendix F demonstrates reports generated by the application using the full sample data provided).

Chapter 2: Review of the Literature

2.1 General Literature

Early attempts at flexible systems, such as that proposed by Parnas (1979, p.128) involved designs based on components. Parnas (1979, p.128) suggested a methodology wherein a project commenced with the identification of minimal subsets, each of which might perform a useful service, and then progressed with the addition of minimal increments to the system. Parnas' chosen components conformed to the following general rules:

- Each component should perform a single function.
- A component should not be reliant on the format and output of data from another component.
- Components should not assume that any functionality exists already in the system.

The component-based methodology, such as proposed by Parnas, was gradually replaced in the late 1980s by an object-oriented (O-O) approach to systems development.

Booch (1994, p.37) suggests that flexibility may be achieved through the use of the O-O approach to define a system as a collection of objects rather than components. In the O-O approach a measure of flexibility is achieved through the use of the native object features such as inheritance, abstraction and polymorphism. These O-O features allow a programmer to import previously created classes from standard

libraries, and extend/adapt or overload elements of them without requiring any changes to be made to the original class. In order to obtain maximum synergy between the features of O-O, Booch (1994, p57) states: “modules should be cohesive (by grouping logically related abstractions) and loosely coupled (by minimising the dependencies among modules)”. While the O-O technique does permit inclusion of flexibility in its design, and provides an ability to approach a system definition in iterative and incremental fashion, it is, arguably, not a suitable modelling environment for poorly defined systems that suffer major requirements creep during the maintenance phase.

A fragment-based specification method is presented by Blum (1993b, p.728) to allow a conceptual model to be created for systems where requirements are inclined to be dynamic or poorly understood. Blum’s (1993b, p.730) ‘fragment-based’ method stores concepts, known about the application, as fragments in a database. In Blum’s method, specification fragments are assembled to form a definition of the software functions utilised in code generation for the application. However, whilst Blum (1993b, p.731) advocates the use of methods that allow the gathering of ‘open requirements’, he provides criticism in that “there are high risks in developing systems with open requirements: the resulting product may not be useful”.

Ensor and Stevenson (1997, p.503) propose the implementation of flexibility via user extensibility. User extensibility refers to systems that allow a user to modify a system from within the application, thereby maximising the utility of a system. Ensor and Stevenson (1997, p.504) divide their proposal into two types of flexibility: namely

'schema extensibility' and 'algorithmic extensibility', in order to classify better those systems with 'open requirements'. To elaborate, 'schema extensibility' refers to situations where new attributes and/or entities are required. 'Algorithmic extensibility', then, refers to the changing of the underlying business rules in order to supplement, update or replace the current rules. Ensor and Stevenson advise that where a rule is likely to change, the code used to implement the rule should be stored as a package in a database table, allowing for the easy manipulation of existing rules. The flexibility achieved through the use of these database packages may be extended through the storage of every atomic action (a fundamental, indivisible code element) as a procedure or function in the database, thereby maximising the amount of code reuse and, in-turn, decreasing maintenance costs.

Woolfolk, Ligezinski & Johnson (1996, p.3) put forth the idea of a 'dynamic search condition' as a method to implement 'algorithmic extensibility'. The 'dynamic search condition' is used to allow the association of business rules together with data, thus providing both user extensibility and system flexibility. By way of mechanism, such association is achieved through the use of two files and a search program, where the first file contains the data and the second file contains the key values needed to search the data. The search program, illustrated in figure 4, accepts predefined arguments and then searches the data file according to a pattern determined by a key that is built from the key values in the second file. The effect of the mechanism is to allow the program to change the access rules to a table (notably without any modification to the program), effectively providing a facility for changes in business rules applied to the data.

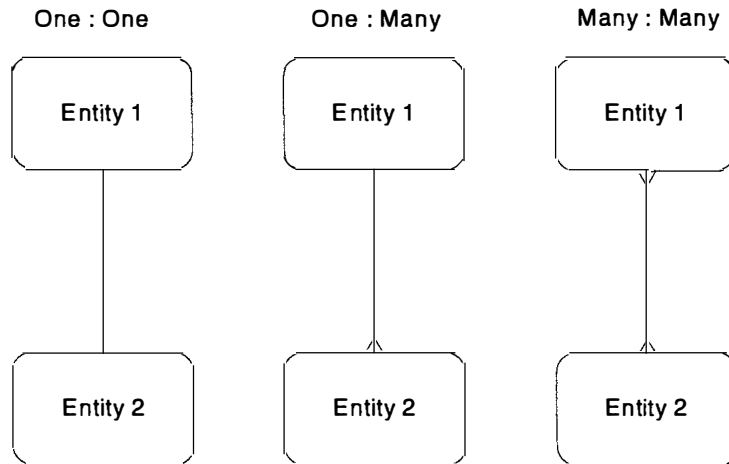


Figure 4. The dynamic search condition

Mehandjiev & Bottaci (1996, p.450) observe that a purely graphical language might provide a level of flexibility not achievable with standard ‘algorithmic extensibility’. Layng (1998, p.16) states: “if an application can be written in a language that is purely graphical rather than text based then there is a huge potential for users to modify the structure of the program as they wish”. However, as yet no ‘pure graphical’ language of any note has been developed that may be applied to the general domain, although the Object Management Group, through their development of Unified Modelling Language (UML), is making advances in this area (Pohl, 1997, p.142).

In order to provide a standard for extensible systems, graphical or otherwise, Hall & Ligezinski (1997b, p.6) suggest four main areas to be addressed by any system before it may be considered flexible:

- **Access security:** where each user needs to access only that functionality for which they have authorisation.
- **Dynamic report formulation:** where reports produced by the system must be able to adapt to changes in the system.
- **Data entry processes:** where data entry screens should not be fixed, as they define the way a user 'sees' and interacts with a system.
- **Business rules applying for specific conditions:** where flexible application of business rules is allowed under certain circumstances.

2.2 Specific Studies Similar to the Current Study

A method proposed by Hall & Ligezinski (1997b, p.7) to satisfy their four nominated areas of flexibility, is via the use of 'common code' tables that allow both specification and implementation of flexible software. They state: "When a new code value needs to be added, or modified, it can be accessed at runtime through database queries, dynamic record groups and lists of values as opposed to the usual predefined pop-lists". Common code tables, in which values, located on a one to one basis against identifiers known to the program, are read by a program at run time and, therefore, stored ready to be queried immediately prior to use. An example of such use may be to store values for display in screen menus, where the contents of the menus change according to context.

For example, a table called FRUITS might be used in a system to store the name of a fruit and its associated value, and this table may, in turn, be linked via a record group (explained in depth later) for display in a ‘front end’ object or selection list/menu. Thus if a fruit is added to the ‘common code’ table FRUITS the value would automatically be linked with its associated object, e.g. facilitating easy changes to menu contents by editing the FRUITS table contents.

The technique has been used in many recent large systems programmed in Oracle Forms. One such system is the CONCEPT Human Resource Management software, which uses such a ‘common code’ table nominated, appropriately, CODES (illustrated in figure 5), to store codes together with their designated values and descriptions for the entire system. By example, the CONCEPT system exemplifies a useful extension of flexibility to the standard ‘common code’ technique through the ‘kind’ column in the CODES table. Values in the CODES tables are grouped by what ‘kind’ they belong to, and, in turn, the kind determines a link, or indirection, to a specific object to be accessed at runtime. Through this extension CONCEPT is able to store the ‘common codes’ for the entire application in the one table.

Name	Null?	Type
-----	-----	----
KIND	NOT NULL	VARCHAR2 (15)
CODE	NOT NULL	VARCHAR2 (50)
DESCRIPTION		VARCHAR2 (50)
LENGTH		NUMBER

Figure 5. SQL Description of Concept HR codes tables (Courtesy of Concept Systems)

O'Connor suggests an alternative use for 'common tables' to provide a high level of flexibility. The 'front end' tables are used to store a large portion of the setup of the 'front end' of an application. O'Connor (1999, p.36) provides an example where the 'front end' screens of an application are stored in five tables (illustrated in figure 6):

- **ST_FORM:** Stores the names and characteristics of those forms that are present in the system, including such details as the width and height of the form that will be displayed at run-time.
- **ST_ITEM_TYPE:** Stores the names of all item types that may appear on any given form. These include buttons, text items, Lists of Values (LOVs) and radio-button groupings.
- **ST_ITEM:** Stores the names of all possible items that may be included on any form.
- **ST_FORM_ITEM:** Stores the items that will appear on a particular form and indicates how the various different run-time properties (e.g. position, width, and font) will be set when that form is run.
- **ST_FORM_REF:** Stores the names of items that should be displayed on a form based upon values stored in a different table, (e.g. where items are displayed on a Customer maintenance form according to a group to which they belong).

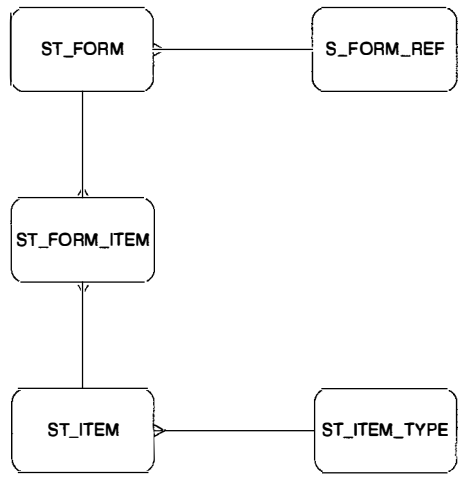


Figure 6. ER Diagram of 'Front End' setup tables

The 'setup' tables shown in figure 6 are mapped onto the overall application database according to the Entity-Relationship (E-R) diagram depicted in figure 7. Each form is then defined by altering the defined setup values in a maintenance form, so that when a window is first displayed a call is made to an appropriate function to display the required items. O'Connor (1999, p.56) notes about his design: "the dynamic screen concept appeared a good idea in theory, however it does have many limitations in practice". The limitations mentioned in O'Connor's study stem from the increased overhead of querying all five setup tables (shown in figure 6) prior to loading each screen during the application's runtime.

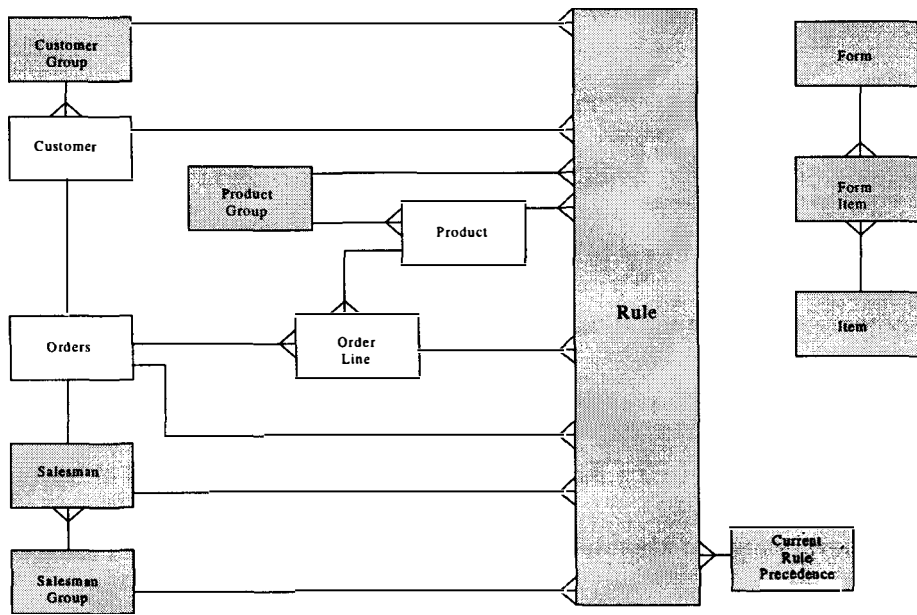


Figure 7. Application E-R Diagram (O'Connor, 1999, p.36)

While the above studies (O'Connor, 1999; Hall & Ligezinski, 1997b) demonstrate techniques utilised successfully to provide flexible systems, they do limit flexibility in the type of data stored. The 'common code' and 'front end' table techniques are limited as they achieve user extensible systems exclusively through the implementation of algorithmic extensibility, ignoring schema extensibility. The study proposes to address this limitation through a dynamic data model (explained in chapter four), providing a level of flexibility superior to the techniques previously described.

2.3 Other Literature of Significance to this Study

Recall that the aim of this study is to provide a system with schema extensibility through an extension of existing techniques. Inevitably, such extension includes the rules governing normalisation, modelling of data, and specific techniques for the implementation of native features that promote or facilitate flexibility in environments such as Oracle. A modelling method that complements normalisation, and that will be employed in the study, is that of Entity-Relationship (E-R) modelling, where business rules may be applied and where the completeness of data attributes is not necessarily known at inception of the model. In other words, the tables we use to provide flexibility must be constructed according to the rules of normalisation and E-R modelling that apply to any well-founded database. Both of these techniques will now be elaborated, together with other literature of significance to the study.

2.3.1 Normalisation

Codd (1970, p.378) formulated a mathematically based set of design principles for use in designing relational databases. These principles have become formalised in terms of progressive normalisation of data via, at least, three normal forms (known as: first, second, and third normal form). Other normal forms are now known to exist but will not be covered in this review. Normalisation is, in effect, a data drive, 'bottom-up' approach and its purpose is to remove what are known as the 'three side effects', which are, paraphrased from Beynon-Davies (1996, p.144):

- **Deletion side effect:** where the deletion of a specific piece of data results in the loss of related information that is still valid.
- **Update side effect:** where the operation of updating a single piece of data requires the modification of more than one database column.

- **Insertion side effect:** where data may not be entered as it is dependent on having at least one link to another part of the system.

The removal of the Beynon-Davies' side effects ensures database stability following volatility of its contained data.

Beynon-Davies (1996, p.145) describes normalisation as consisting of four steps:

1. Collect the data set into a list of un-normalised data.
2. Transform the resultant list into tables in the first normal form: defined by Beynon-Davies (1996, p.147) as:

“A relation is in first normal form if and only if every non-key attribute is functionally dependent upon the primary key.”

3. Transform first normal form tables to second normal form: defined by Beynon-Davies (1996, p.149) as:

“A relation is in second normal form if and only if it is in first normal form and every non-key attribute is fully functionally dependent on the primary keys.”

4. Transform second normal form tables to third normal form: defined by Beynon-Davies (1996, p.151) as:

“A relation is in third normal form if and only if it is in second normal form and every non-key attribute is non-transitively dependent on the primary key.”

Following the above steps produces a stable and efficient relational data model on which an application may be based with confidence. Accordingly, the practice of normalisation is adhered to in this study in order to yield such stability and efficiency of stored data in its example application. Normalisation is also used to analyse and incorporate data storage information into the data set to provide a schema extensible system. This will be explained in chapter four.

2.3.2 Entity Relationship Modelling

To complement the 'bottom-up' approach to database design, i.e. normalisation, a 'top down' analysis, known as Entity-Relationship (E-R) modelling, is generally followed by current practitioners. E-R modelling is employed particularly in situations where an analyst may not know all needed data before modelling commences. It provides a rapid approach for producing a model that may subsequently be refined using normalisation techniques and, as Beynon - Davies, (1996, p.162) reports: "In practice, database developers normally do more data modelling than they do normalisation."

The E-R modelling approach was devised by Chen (1976) to facilitate data modelling for problem areas for which a database solution is envisaged and is used frequently in association with the relational model.

The first step in conducting E-R modelling is the identification of all necessary entities. The exact definition of what constitutes an entity varies, but Beynon - Davies (1996, p.162) provides the following: "An entity may be defined as a thing which the entire enterprise recognises as being capable of an independent existence and which can be uniquely identified". This definition concurs with that of Hall (1998, p.7) who

defines an entity as “An individual object, concept or event about which the organisation chooses to collect and store data”. In E-R models, named rectangular objects depict entities, as shown in figure 8.

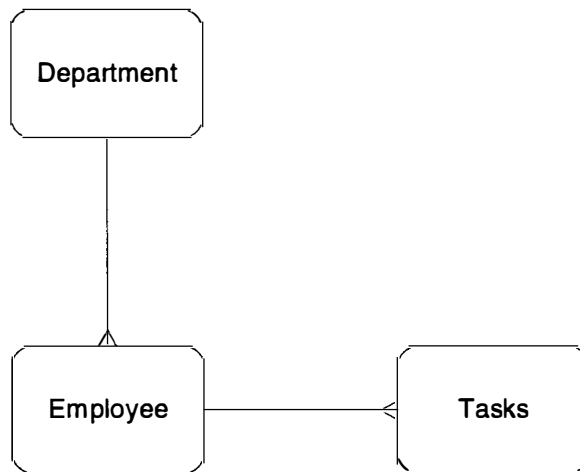


Figure 8. A sample ER diagram containing three entities: Department, Employee and Tasks.

To model the interaction/association between entities a relationship is used, depicted an example of which is shown in figure 9. Three relationships are known to exist in Chen’s model:

- **One to one relationships:** where an instance of one entity has a direct single link to another separate entity.
- **One to many relationships:** where an instance of one entity has multiple direct links to another separate entity.
- **Many to many relationships:** where many instances of the same entity have links to many instances of another separate entity

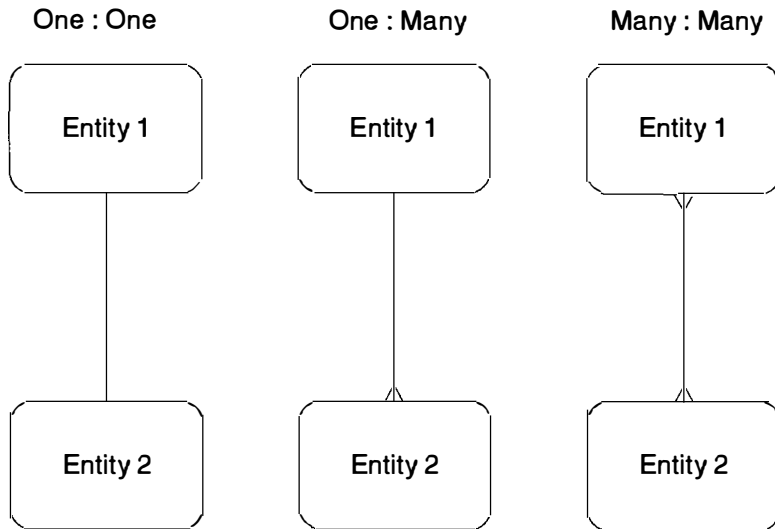


Figure 9. The mapping of Chen's relationships (Hall, 1998, p.24)

A completed E-R model, in conjunction with the normalised data, shows the structure and detail of an intended database, in which entities are implemented as tables and relationships achieved by linking key attributes common to related tables.

2.3.3 Native Oracle Features used to facilitate flexibility

The Oracle development environment is provided for the creation of the ‘front end’ of an Oracle application. The ‘front end’ facilitates, for the user, an interface to the underlying database ‘back end’. Inherent in the Oracle development environment are many features that may be used to enhance and create flexible applications.

O’Connor (1999, p.24) suggests that the most useful of these flexibility enhancing features are:

- %TYPE
- Setting Properties Programmatically
- Dynamic Record Groups
- List of Values

Each of O’Connor’s ‘flexibility features’ will now be elaborated in turn.

2.3.3.1 %TYPE

The identifier %TYPE declares a variable of:

- a previously declared variable
- a column in a table

The %TYPE identifier is utilised in the variable declaration section of Oracle procedures, functions and packages to ensure type consistency. Consider the database table depicted in figure 10 and the PL/SQL procedure block illustrated in figure 11:

<u>Table</u>	<u>Columns</u>		
Flex	Row_ID	NUMBER	NOT NULL
	Char_Col	VARCHAR2(10)	
	Num_Col	NUMBER	
	Date_Col	DATE	

Figure 10. The flex database table

```

DECLARE
    Flex_Value Flex.Char_Col%Type
BEGIN
    SELECT *
        INTO Flex_Value
        FROM Flex
        WHERE Row_ID = 1;
    RETURN Flex_Value;
END;

```

Figure 11. Utilising %TYPE in PL/SQL

In figure 11, the variable Flex_Value is declared from the same type as a value originating in the database column Char_Col in the table Flex shown in figure 10. The full value of using the %TYPE identifier in this manner becomes apparent when used in database side subprograms of procedures, packages and functions. In these database side subprograms, efficacy is achieved as they are automatically recompiled before every runtime. Such recompilation allows the %TYPE to reassign the variable type when required, such as following a change in the underlying type of a database item. Even when the %TYPE is used in client side functions, the type used by the application may be updated by performing a manual re-compile.

The %TYPE feature enhances the flexibility of database procedures and functions by allowing them to adapt to changes in data formats. Stacey (1995, p.3) held such changes accountable for the greatest portion of software maintenance activities.

Stacey's opinion agrees with that of Feuerstien (1996, p.25) who states that the most common cause of application failure is "the undying belief held by programmers that a particular value will never change and so can be hard coded into the program."

O'Connor (1999, p.27) cites a Ministry of Justice [of Western Australia] case management system that has implemented a change of underlying database types by using the %TYPE feature and subsequent recompilation to activate the changes.

2.3.3.2 Setting Properties Programmatically

A property is defined in Oracle (2001) as "an attribute of an item that may altered to modify the setup of that item." Manipulation of these attributes is achieved via property functions that allow, for example, a facility for manipulation of the physical properties of graphical objects, where the manipulation may result in an enhancement to the flexibility of a system. Development environments such as Oracle's Developer provide utilities of GET and SET property functions, examples of which include:

- **SET_ITEM_PROPERTY:** Modifies all instances of an item in a block by changing a specified item property.
- **GET_ITEM_PROPERTY:** Returns property values for the specified item.
- **SET_WINDOW_PROPERTY:** Sets a property for the selected window.
- **GET_WINDOW_PROPERTY:** Returns the current setting for the selected window property for a given window.
- **SET_BLOCK_PROPERT:** Sets the given block characteristic of a specified block.

- **GET_BLOCK_PROPERTY:** Returns information about a specified block.

Through these standard SET and GET functions it is possible to manipulate the ‘front end’ of an application, thereby allowing a quasi-visual form of flexibility that approximates Mehandjiev & Bottaci’s (1996, p.450) idea of providing flexibility by using a graphical method.

2.3.3.3 Dynamic Record Groups

Record groups may be used to provide both query and data flexibility in applications. They form sets/lists of information that may be created and populated through the use of SQL queries. Typically, record groups, are used to generate small subsets of data for those specialised operations where no information update is occurring, e.g. when checking if a value exists. Hall & Ligezinski (1997b, p.7) suggest that dynamic record groups are best employed in conjunction with the previously described common code tables.

A typical example of a record group would be a small SQL query that extracted a list of items, each of which has a common property, from a table e.g. a list of all the male employees in an organisation’s payroll system. The record group might then be used inside an application in preference to a non-record group situation where the same query would have to be run multiple times to achieve the same effect.

2.3.3.4 List of Values

LOVs are a native feature of Oracle and may be used to replace the traditional static lists, known also as ‘pop lists’ or combo-boxes, found in standard application front ends. They are associated directly with a record group, thereby “ensuring the information they contain is always current” (O’Connor, 1999, p.29).

Additionally, unlike ‘pop lists’, a LOV does not require a direct link to a text item, thereby allowing sufficient flexibility to attach a LOV to a button or a menu item. While a LOV may be associated directly with a text item in its property settings, a function such as that depicted, in figure 12, as `SHOW_LOV` may be used to handle the activation of the LOV programmatically. The `SHOW_LOV` function has a return type of Boolean, returning `TRUE` where a value is selected or `FALSE` if the user cancelled the LOV.

```
DECLARE
    LOV_Used  BOOLEAN;
BEGIN
    LOV_Used := SHOW_LOV('A_LOV');
END;
```

Figure 12. Calling a LOV using PL/SQL

2.4 Summary

Employing facts and documented experience, selected literature has been used to provide a basis for a discussion about identifiable, desirable features that may promote development of flexible systems. The concept of flexibility has been discussed with an emergent idea that to provide a strong level of flexibility a system should be alterable through data values that are held in tables, rather than by the use of code. In addition, those inherent features of the ORACLE database language that may be employed to promote and facilitate flexibility in a relational based system were reviewed.

Chapter 3: Research Method

This chapter outlines the methodology followed by the study to provide answers to the research question proposed in chapter one. It also describes the guidelines for the development of the demonstration application that is discussed in chapter four.

3.1 The Problem

In order to address the research question of “How may the challenge of dynamic/flexible user requirements be met using data driven techniques in a 4GL-database environment?” a demonstration application was designed and implemented using the Oracle 4GL environment. The resultant application implements a staff activity submission system and was chosen for the following reasons:

- The ‘real-life’ requirement for such a system by Edith Cowan University (ECU);
- The unpredictable nature and rapid changes in requirements manifested in the current manual system.

The requirements of the system were gathered in a series of interviews and information/data analysis sessions, the results of which were entered into a CASE tool. A summary of the problem statement is as follows:

On a quarterly basis, the faculty creates the staff newsletter. In order to collect the information for the newsletter, a form (see appendix A) is emailed to all staff members and, ideally, this form is then completed by

staff and emailed back to the newsletter editor. The editor then manually collates and corrects all the information in accordance with the data collection guidelines (see appendix B) and uses this information to create the newsletter.

The following problems were observed with the manual system:

- Difficulty of tracking whether a staff member has/hasn't submitted a return in a specific quarter.
- Manual collation of returns is very time consuming.
- It is hard to inform staff of changes to the form.
- Returns on the form are often in an inappropriate format.

ECU management, following requests by users, decided to commission a computer-based solution. The replacement system needed to address the following issues:

- Allowing staff to enter/maintain their own returns at any time.
- Allowing updates of categories and their associated return items.
- Providing reports to allow easy data extraction.
- Permitting two levels of access: i.e. standard user and administrator.

An initial analysis, described later in this chapter, of the problem area was utilised as a guide for the design and construction of the demonstration application. The

application's development was then separated into several phases, each of which is introduced in this chapter and, further, discussed in depth in chapter four.

3.2 E-R Analysis

The first phase in the design of the demonstration application was an analysis of the problem domain together with the client/user-supplied data. The analysis was used to construct an E-R model and normalised view of the system's data. Recalling that part (a) of the research question was stated: "*How may one model dynamic user requirements through an extension of contemporary data modelling techniques?*", E-R modelling was used to build a model to demonstrate such extension.

The extension was achieved through an investigative process; wherein the system was modelled initially using standard modelling techniques and normalisation. The standard model was then analysed for weakness, and the identified weaknesses were addressed to facilitate the creation of a dynamic model. This process has been detailed further in chapter four.

3.3 Database Design

Upon completion of the E-R analysis the resultant model and normalised data set were converted, using Oracle CASE tools, into a relational database. The successful generation of the database creation scripts from the CASE tool (Appendix C) with the supplied E-R model confirmed the model's validity and provided the 'back end' of the demonstration application.

3.4 Application Development

Development of the demonstration application commenced following the database design phase, in accordance with the requirements gathered therein. Oracle's Developer 6i tool-set was selected for development, for reasons of availability, maintainability and relevance to the task-in-hand, and consists of:

- **Oracle Forms Developer 6i:** A graphical based development language for constructing user interfaces.
- **Oracle Reports Designer 6i:** An SQL based reporting tool used to develop and publish data queries.
- **Oracle SQL 8i:** A structured database query language that contains enhancements over and above the standardised SQL.

Furthermore, the tools, all from the same vendor, ensured maximum coordination between the 'front end' under development and the 'back end' developed during the database design phase. The application was then integrated in an incremental fashion, where each individual form (screen) was developed as a standalone module before being merged into the overall application.

The development and implementation of the demonstration application, in conjunction with the flexible database design, was used to answer part (b) of the research question, stated previously as: *"How may one implement user requirements of flexibility using data driven techniques in a contemporary 4GL environment?"*

3.5 Oracle Environment

The development and presentation of the Staff Submission System was performed on two network and internet enabled IBM compatible personal computers. One machine acted as a development machine and comprised a Pentium III processor with a clock frequency of 700MHz, memory capacity of 128Mb, and a hard disk drive with 10 Gb of storage. Microsoft Windows NT 4.0 (Service Pack 5) was chosen as the operating system, due to its compatibility with the Oracle software used.

The second machine acted as a client, and was typical of that supplied to a user of the Staff Submission System. The client machine comprised a Pentium III processor with a clock frequency of 600MHz, memory capacity of 64Mb, and a hard disk drive with 8 Gb of storage. Typical, again, of a user's machine, client applications were hosted by the Microsoft Windows 98 operating system.

The following products were installed on the development machine:

- Personal Oracle Database version 8.1.6
- Oracle Developer 6i
- Oracle Designer 6i

The following products were installed on the client machine:

- Oracle 8i Client Run-time
- Adobe Acrobat Reader
- Internet Explorer 5.01

In short, the environment and development tools were chosen to reflect a standard industry configuration, based upon the technical advice given by several experienced and qualified practitioners and ECU technical support staff.

3.6 Summary

This chapter has introduced the problem statement that the sample application addressed, outlined the development phases of the study, and related them to the respective component of the research question that they were employed to address.

The environment used to implement the study was introduced and discussed. Chapter four elaborates on the phases introduced in this chapter (principally, those of analysis, design, and development) and details the demonstration application.

Chapter 4: The Demonstration Application

This chapter details the development of the demonstration application, nominated the Staff Submission System. The system was developed in accordance with the research method outlined in chapter three, incorporating, where appropriate, the techniques and features reviewed in chapter two.

4.1 The Database

The sample application's implementation commenced with database construction. In accordance with contemporary design techniques, an incremental process was followed, wherein individual database tables and constraints were modelled and reviewed, in both an initial and flexible design, before being integrated into the database. This allowed for progressive testing and validation of the design whereupon the creation scripts were generated using the CASE tool previously outlined in chapter three. The manner in which the model and resultant database were constructed will now be elaborated.

4.1.1 Initial Design

In order to prevent any neglect/loss of data requirements during the database design, the system was first subjected to straightforward E-R modelling to produce an initial standard model, depicted in figure 13, without consideration of any flexible requirements of the system. This was undertaken in order to identify, subsequently, any possible weaknesses exhibited by the standard model of the system, and to provide a base for development of the flexible model.

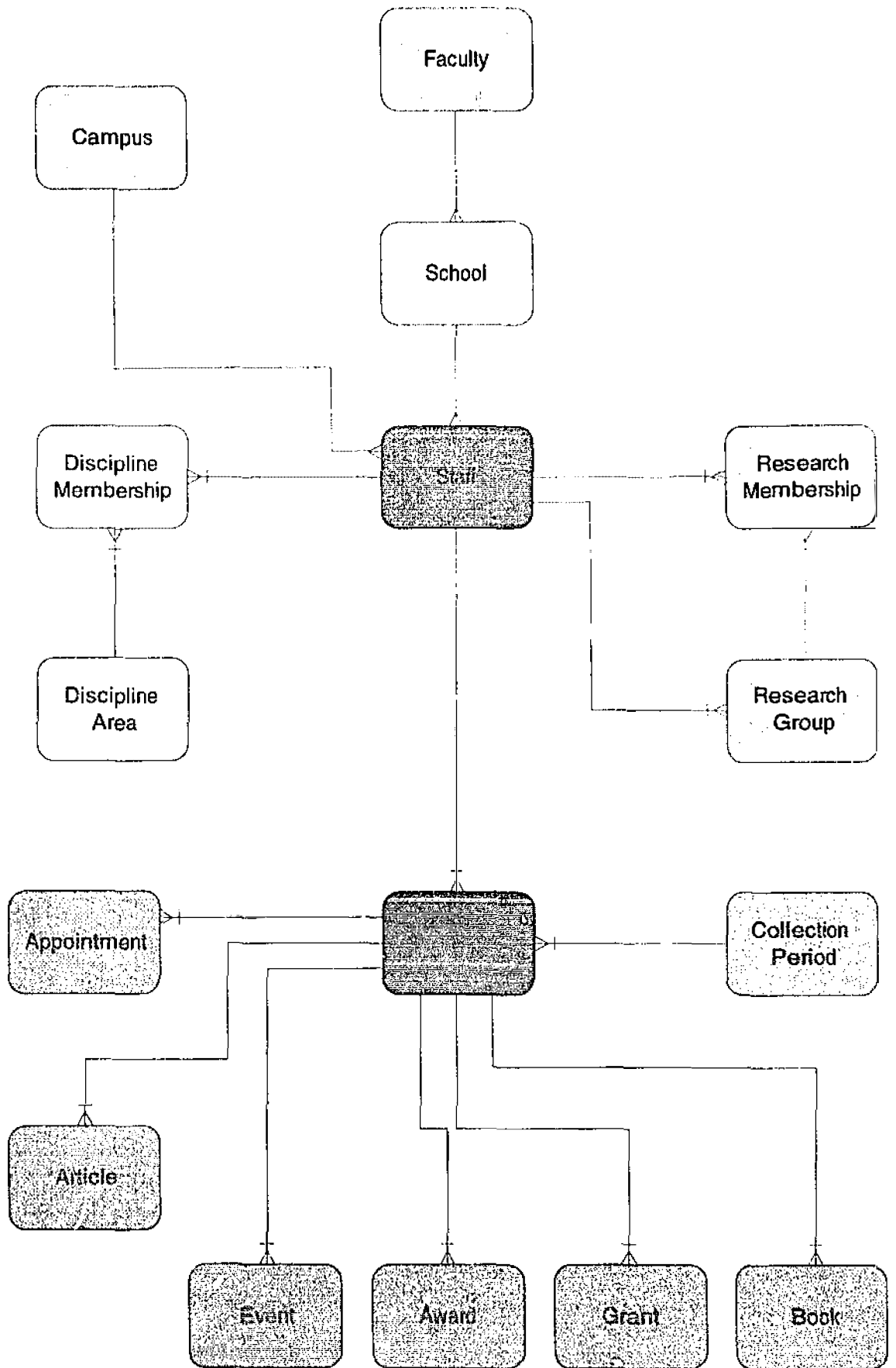


Figure 13. Initial ER-Model of system

Upon examination, two problems were identified as being inherent in the standard model illustrated in figure 13: namely,

- a lack of categorisation; and
- an inefficiency of storage of optional data.

Both of these problems are now explained, in turn.

The importance of the categorisation of elements, listed in appendix B, is ignored by the standard model, which relies upon category elements being 'hard coded' into the menu screens of the application. The menu screens would be required to store the correct category information in order to store data in all six of the item tables (i.e. Article, Award, Event, Appointment, Book, and Grant) illustrated in figure 13. The storage via such screens mandates the use of a single field in each item table to store data for the category of which the item is a member. It may be seen that the structure for such hard-coding of the category hierarchy is rigid and, further, that such rigidity may lead, eventually, to increased maintenance costs in the event of revision of the category hierarchy becoming necessary.

A second deficiency of the standard model, illustrated in figure 13, lies in the amount of overhead for optional data to be stored by all six of the item tables. For example, the 'Book' table described in figure 14 contains four columns that are not needed in every situation, but that would appear on related data entry screens.

Name	Null?	Type
-----	-----	----
RETURN_NO	NOT NULL	NUMBER
CATEGORY	NOT NULL	VARCHAR2 (150)
AUTHOR	NOT NULL	VARCHAR2 (100)
DATE	NOT NULL	DATE
BOOK_TITLE	NOT NULL	VARCHAR2 (150)
CHAPTER_TITLE		VARCHAR2 (150)
PAGES		VARCHAR2 (20)
PUBLISHER		VARCHAR2 (150)
VENUE		VARCHAR2 (150)

Figure 14. Description of the BOOK table

Both of the identified problems suggest that the standard model exhibits extreme limitations in its ability to allow for ‘schema extensibility’ (discussed in chapter two). By example, should the user wish to enter a new type of item, (e.g. videos) into the category hierarchy to allow a new type of input data, the application would require the following maintenance tasks:

- The addition of a new table to store the data, i.e. a new entity added to figure 13, linked to the ‘Return’ table.
- The creation of, at least, one new screen to allow data entry into the new table.
- The alteration of hard coded category settings to accommodate the new item type and its associated screen(s).

The nature of the maintenance tasks implies that the standard E-R model's limitations would have a noticeable cost associated with modifications to those hard-coded features required in response to requirement changes.

4.1.2 Flexible Design

To overcome the limitations of the initial, non-flexible, data model the problem area was reviewed in order to effect a reduction of possible future maintenance costs. The increased maintenance imposed by the standard model's six separate item tables (illustrated in figure 13) was mitigated by the introduction of a single table structure. The single table is capable of storing all of the category hierarchy information that would, in the non-flexible, standard application, be hard coded.

When modelling the category hierarchy the following points were considered:

- Categories may contain one or more subcategories.
- Subcategories consist of one or more data items, e.g. a subcategory storing book details may consist of the following data items: author names, publisher details, date published, and book title.
- A data item may exist in more than one subcategory e.g. one or more subcategories may include 'name' as a valid field.

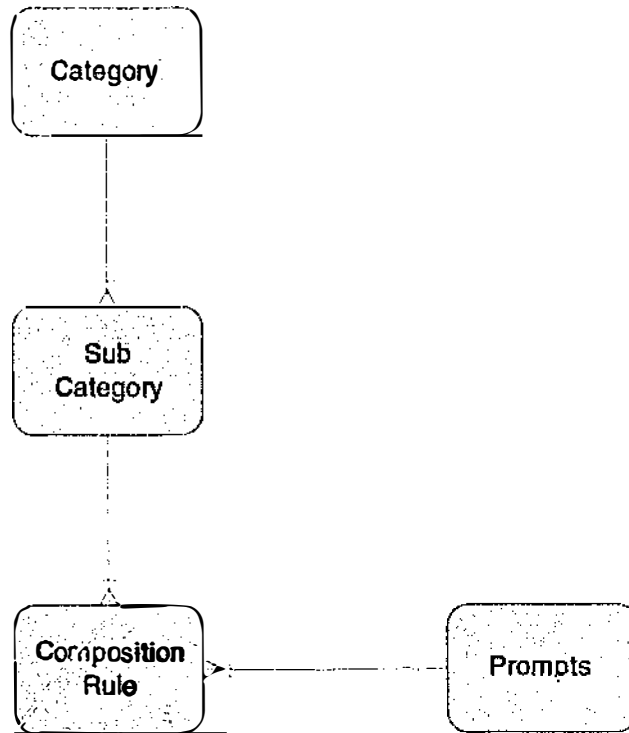


Figure 15. ER - model of category data

The resultant data model, illustrated in figure 15, permits storage of all the information contained within the category collection list described in appendix B. In addition the 'Composition Rule' entity permits storage of dynamic mapping data allowing application fields to be remapped automatically at runtime.

However, before this dynamic mapping might be added to the 'Composition Rule' entity, a dynamic storage entity was required to provide a suitable data storage area onto which to map the application fields. In order to achieve an optimal form for the dynamic entity, the storage requirements of each return item in appendix B were mapped onto the table shown in Figure 16.

Cat	Sub Cat	Sub Sub Cat	Varchar2	Number	Memo	Date
1	1		2	0	1	0
	2		3	1	1	0
	3		4	0	1	0
2	1		4	1	1	0
	2	1	4	0	0	1
		2	5	0	0	1
		3	5	0	0	1
		4	5	0	0	1
		5	5	0	0	1
		6	5	0	0	1
		7	4	0	0	1
		8	3	0	0	1
		9	3	0	0	1
		10	3	0	0	1
		11	4	0	0	0
	3		3	0	1	0
	4		1	0	1	0
	5		3	0	1	0
3	1		3	0	1	1
	2		3	0	0	1
	3		3	0	1	0
	4		2	0	1	0
	5		4	0	1	0
4	1		3	0	1	0
	2		4	0	1	0
	3		2	0	1	0
	4		2	0	1	0
5			2	0	1	0
		Max	5	1	1	1

Figure 16. Storage requirements of category items

The table depicted in figure 16 indicates that a dynamic entity, nominated 'Return Item', that contains columns of five character, one memo, one date and one number type may be used to store any category item. To this end, and to allow for future growth, a dynamic entity configured with six characters, one memo, two date and two number columns was implemented. The dynamic entity provided the required bridge between the data model in figure 13 and the category definition data model in figure 15. The resultant, complete, model is illustrated in Figure 17.

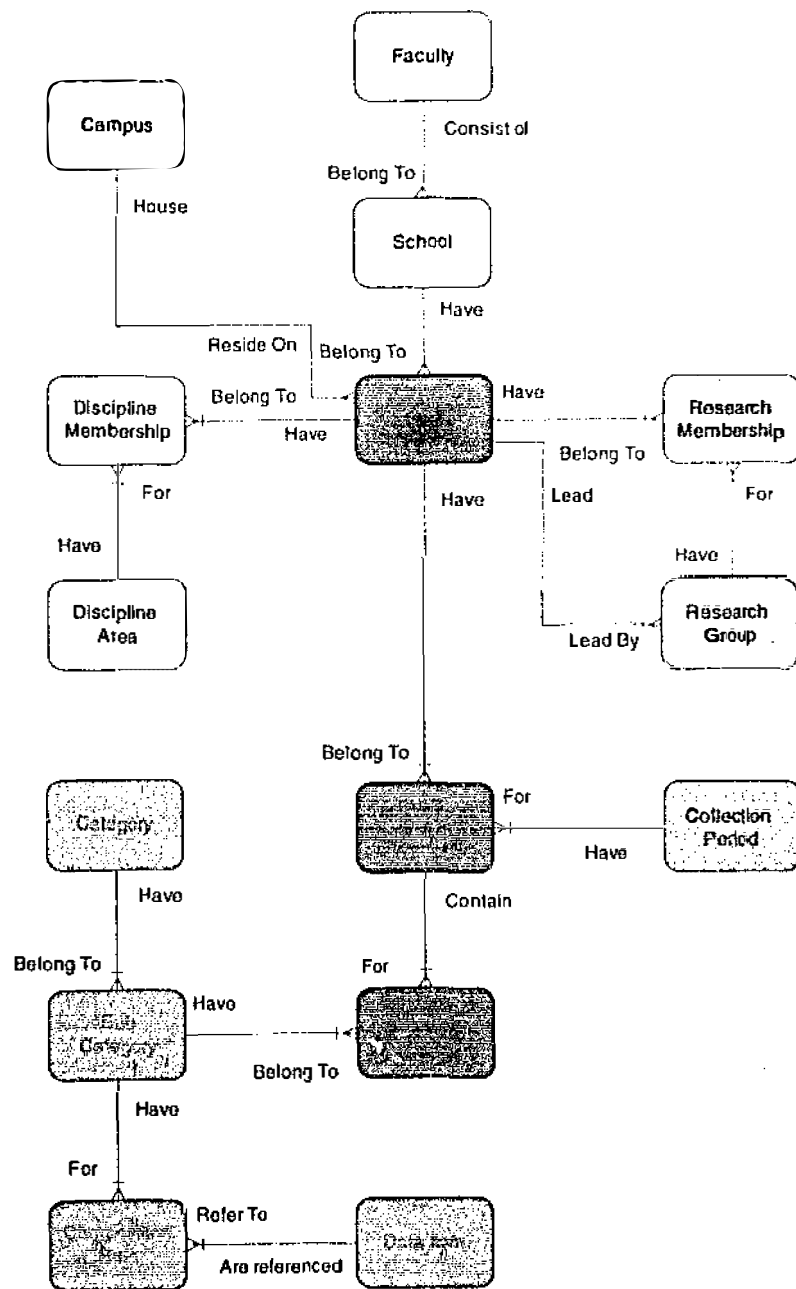


Figure 17. Dynamic ER-Model

The bridge works by defining the configuration of the 'Return Item' entity (in figure 17), for each combination of the category hierarchy. The configurations are stored in the composition rules entity and accessed via application scripts whenever data is required to be stored/retrieved from the dynamic entity. Thus each category combination may have its own unique way of viewing the data stored in the dynamic entity of 'Return Item'.

4.1.3 Creating the database

The resultant system model was entered into the Oracle CASE repository using the E-R modelling tool. The CASE tool was then used to produce the database creation scripts listed in appendix C, whereupon these were run on the database to form the 'back end' of the flexible application.

4.2 The Application

The flexible application, consisting of a graphically based 'front end', was constructed to sit atop the 'back end' database. To facilitate the flexibility, the sample application development was split into three logical areas:

1. The first area was concerned with the setup and maintenance of the flexible area of the application i.e. that of the category tables illustrated in figure 15.
2. The second area contained those standard functions of the system that utilise the setup data from the category tables illustrated in figure 15, to present flexible data entry screens.
3. The third area of the application focussed upon retrievability of the data entered into the flexible system.

Each of the above development areas will now be elaborated.

4.2.1 Flexible Setup

The nature of the flow and connectivity of the system's forms, or screens, is derived from the category setup data. Accordingly, a form flow diagram was constructed, illustrated in figure 18. Each 'box' shown in figure 18 translates directly to one or more functions/forms in the final application.

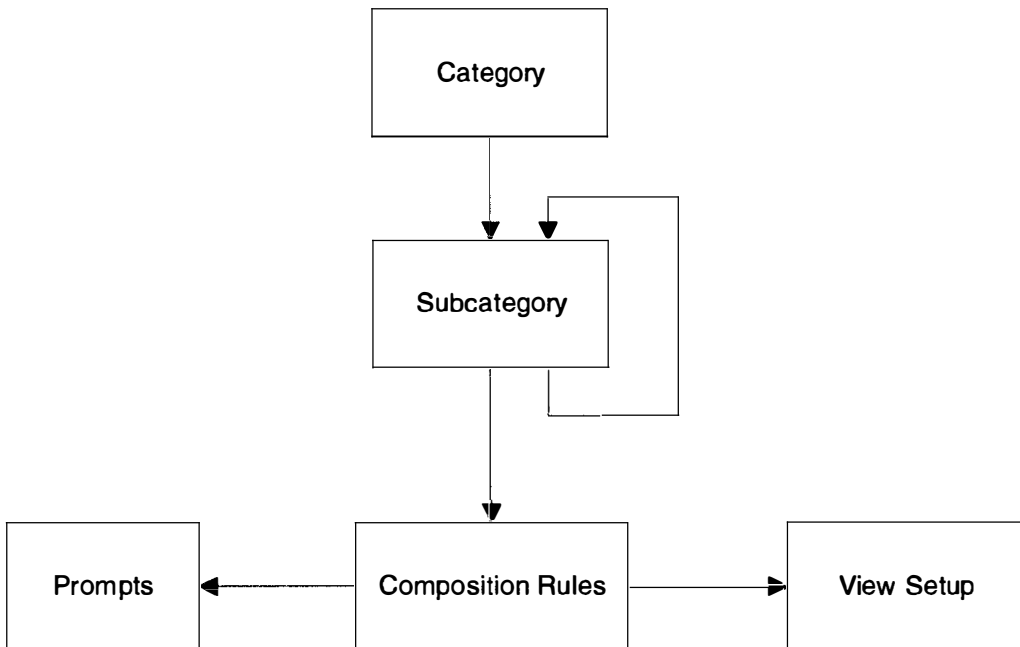


Figure 18. System maintenance flow

In order to maintain system integrity, this flow forces the application's users to adhere to both the logical and the physical relationship existent between the category maintenance tables (those illustrated in figure 15). The flow also ensures that the transition of the application forms/screens gives the user an implicit, even intuitive, understanding of the relationship between the setup information and how it is controlled by the system. In order to demonstrate this understanding each of the functions/forms illustrated in figure 18 will be explored in detail.

4.2.1.1 Category Maintenance

The user, through the category maintenance form, illustrated in figure 20, carries out category maintenance. The form provides a graphical interface between the user and the category database table they are altering, described in figure 19.

Name	Null?	Type
-----	-----	-----
CAT_NO	NOT NULL	NUMBER
CAT_NAME	NOT NULL	VARCHAR2 (50)
CAT_DESCRIPTION		LONG
CAT_DATE_CREATED	NOT NULL	DATE
CAT_DATE_REMOVED		DATE

Figure 19. Description of the CATEGORY table

The screenshot shows a graphical user interface window titled "CATEGORIES". The window contains the following elements:

- Category No:** A text input field containing the value "1".
- Name:** A text input field containing the value "Teaching and Learning".
- Description:** A large text area for entering a description.
- Date Created:** A date input field containing the value "01/01/2000".
- Date Removed:** An empty date input field.
- Buttons:** A set of five buttons arranged in two rows: "Remove" and "Drill Down" in the top row; "Previous", "Back", and "Next" in the bottom row.

Figure 20. Category maintenance form

Each category maintenance function is associated with one of the five buttons on the form, shown in figure 20. When a button is selected, one of the following operations is activated:

- **Previous:** To navigate to the preceding category information, and where no preceding category exists then this button appears disabled.
- **Back:** To close the category maintenance form, and return to the main menu.
- **Next:** To Navigate to the succeeding category information; and where no succeeding category exists then this button becomes disabled.
- **Remove:** To set the removal date to today's date, having the effect of removing this category from the category hierarchy. Furthermore, this button will raise an error message if the currently selected category contains any open subcategories.
- **Drill Down:** To open the subcategory maintenance form described in the next section.

These operations permit the user to maintain the data stored in the category table with minimal effort.

4.2.1.2 Subcategory Maintenance

In similar style to the category maintenance functions, subcategory maintenance functions focus on the maintenance of a specific table, in this case the subcategory table, nominated SUB_CATEGORY, described in figure 21. However, there is an extra level of complexity involved in the maintenance of a subcategory due to its

breakdown into further subcategories (in effect, sub subcategories). Three forms, illustrated in figures 23 to 26, were developed to achieve sub-category maintenance, the flow and hierarchy of which is depicted in figure 22.

Name	Null?	Type
CAT_NO	NOT NULL	NUMBER
SUB_CAT_NO	NOT NULL	NUMBER
SUB_SUB_CAT_NO	NOT NULL	NUMBER
SUB_CAT_NAME	NOT NULL	VARCHAR2 (50)
SUB_CAT_DESCRIPTION		LONG
SC_DATE_CREATED	NOT NULL	DATE
SC_DATE_REMOVED		DATE

Figure 21. Description of the SUB_CATEGORY table

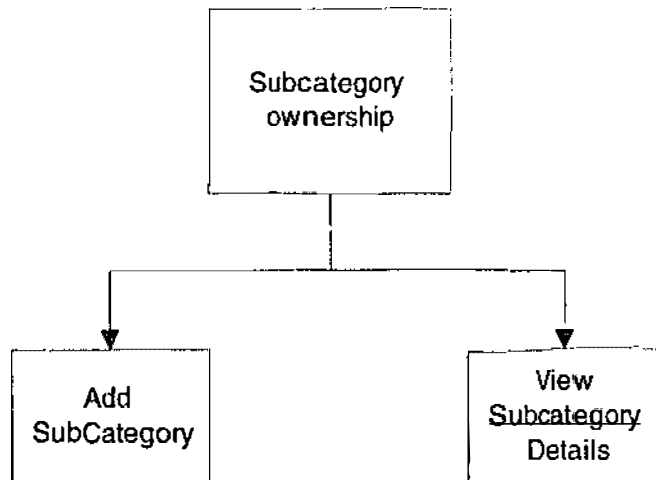


Figure 22. Subcategory maintenance screen flow

The entry point of the subcategory maintenance flow is via a 'Subcategory Ownership' form, illustrated in figure 23. This form, which is called from the previously discussed 'drill down' button on the category maintenance form (figure 20), displays a list of the subcategories relating to the category selected in the calling

form. The form's purpose is to allow the user to see an overview of the category-subcategory hierarchy.

The screenshot shows a window titled 'SUB_CATEGORIES'. At the top, there is a 'Category' field with the value '1' and a text box containing 'Teaching and Learning'. Below this is a section titled 'Sub Categories' containing a table with the following data:

Sub Cat No	Sub Sub Cat No	Sub Cat Name	Rules	Date Removed
1	0	Teaching initiatives	TRUE	
2	0	Teaching awards and grants	TRUE	
3	0	Achievements by undergraduate students	TRUE	

At the bottom of the form, there are four buttons: 'Add Sub Category', 'Add Sub Sub Category', 'View Details', and 'Back'.

Figure 23. Subcategory ownership form

The purpose of each of the four buttons on the 'Subcategory ownership' form, figure 23, is as follows:

- **Add Sub Category:** To Open the 'Add Subcategory' form (figure 24) for the input of a new subcategory.
- **Add Sub Sub Category:** To open the 'Add Subcategory' (figure 25) form for the input of a new sub subcategory.
- **View Details:** To open the 'View Subcategory Details' (figure 26) form and display the details for the selected subcategory.

- **Back:** To close the 'Subcategory ownership' form, and return to the category maintenance form.

Providing addition of subcategories, a variant of the 'Add Subcategory' form is called according to whether the 'Add subcategory' button or the 'Add sub sub category' button is pressed on the 'Subcategory ownership' form (figure 23). The two variants are shown in figures 24, for editing Subcategories, and figure 25, which has an additional field necessary for adding Sub Sub Categories.

The screenshot shows a window titled "ADD_SUB_CATEGORY". Inside the window, there is a "Sub Category" section. It contains two input fields: "Category" with the value "2" and "Research and Creative Works", and "Sub Category" with the value "3". Below these is a large "Description" text area. At the bottom of the form, there are two date fields: "Date Created" with the value "23/01/2001" and "Date Removed" which is empty. At the very bottom, there are two buttons: "Save" and "Cancel".

Figure 24. Add subcategory form (format 1)

Figure 25. Add subcategory form (format 2)

In order to edit either sub or sub-sub categories, thus completing the flow depicted in figure 22, an appropriate variant of the ‘View subcategory details’ form is provided. These are illustrated in figure 26 and 27. The buttons at base of these forms provide the following functionality:

- **Remove:** Sets the ‘date removed’ field to today’s date, which has the effect of removing the subcategory from active use.
- **View Rules:** Calls the ‘composition rules’ form, which, in turn displays the setup rules of the current subcategory.
- **Back:** Closes the current form and returns control to the previously mentioned ‘Subcategory ownership’ form.

VIEW_SUB_CATEGORY

Sub Category

Category 2 Research and Creative Works

Sub Category 1 Research Grants

Description

Date Created 01/01/2000 Date Removed

Remove View Rules Back

Figure 26. View subcategory details form (format 1)

VIEW_SUB_SUB_CATEGORY

Sub Category

Category 2 Research and Creative Works

Sub Category 2 Publications

Sub Sub Category 2 Book chapter

Description

Date Created 01/01/2000 Date Removed

Remove View Rules Back

Figure 27. View subcategory details form (format 2)

4.2.1.3 Composition Rule Maintenance

Composition rule maintenance activities represent the final stage in the category maintenance flow (figure 18). They provide a means by which the user may control dynamic properties of the system and are facilitated by using the composition rules maintenance form illustrated in figure 28.

Prompt No	Prompt Text	Column No	Type	Sequence No
13	Author(s)	1	VARCHAR2	1
14	Date	10	NUMBER	2
18	Title of chapter	2	VARCHAR2	3
15	Title	3	VARCHAR2	4
16	Publisher	4	VARCHAR2	5
17	Venue	5	VARCHAR2	6

Figure 28. Composition rules maintenance form

Recall the discussion in section 4.1.2, wherein a bridge was created to facilitate the mapping of configuration rules that determine how the dynamic table, 'Return Items', will be accessed. The rules are setup/maintained via the 'Composition rules maintenance' form (figure 28). In addition, the form is used for configuring the on-screen layout of the main data entry screen, which will be discussed in the next section.

The columns in the composition rules maintenance form each control an individual link of the dynamic mapping process via the following functionality:

- **Prompt:** The label used to describe the data stored in its associated column e.g. a prompt value of 'Name' would result in a field called Name being displayed on the dynamic data entry screen and in the flexible reports.
- **Column:** The number of the database column to be mapped to, in accordance with the column-mapping table illustrated in figure 29.

Column No	Database Field	Type
1	VCHAR_ONE	VARCHAR2
2	VCHAR_TWO	VARCHAR2
3	VCHAR_THREE	VARCHAR2
4	VCHAR_FOUR	VARCHAR2
5	VCHAR_FIVE	VARCHAR2
6	VCHAR_SIX	VARCHAR2
7	NUM_SEVEN	NUMBER
8	NUM_EIGHT	NUMBER
9	TEXT_NINE	LONG
10	DATE_TEN	DATE
11	DATE_ELEVEN	DATE

Figure 29. Column mapping table

- **Type:** This field is automatically updated, based upon the database column number in the previous field, as shown in figure 29.
- **Sequence:** This field determines the order in which the defined fields will be displayed in both dynamic screens and reports. Alteration of this value facilitates immediate change of all related screen layouts.

Thus, through the 'Composition rules maintenance' form the user may define the rules governing access to the dynamic table, 'Return Item' and, further, how an individual display item will be seen by other users of the system during data entry.

4.2.2 Data Entry

Recall, as discussed in section 4.2, the second phase in the development of the application is concerned with data entry. The data entry functions of the Staff Submission System are separated into two further sections:

1. Those functions relating to the dynamic features of the database; and
2. Those functions relating to the static features of the database.

Each of the above data entry areas will now be elaborated.

4.2.2.1 Dynamic Data Entry

The functions controlling dynamic data entry relate to the process by which a staff member (user) creates and edits return items. As with the category maintenance process (discussed in the previous section), development of the staff return entry process was initiated by creation of the form flow diagram illustrated in figure 30.

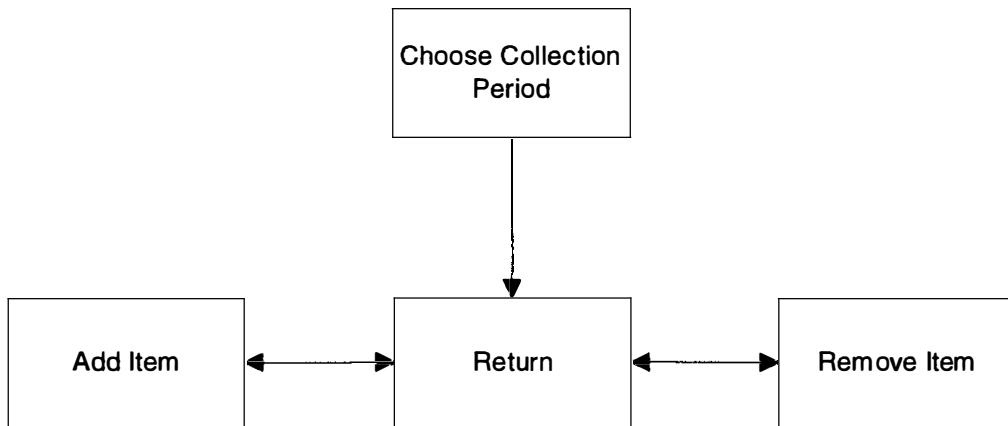


Figure 30. Staff return entry flow

Figure 30 models the logical flow for the entry/maintenance of return items, beginning with selection of a collection period deriving from the need (detailed in the problem statement stated in chapter three), for staff to submit a return every quarter. Selection of a collection period was facilitated in the data model by the definition and storage of the quarterly return dates inside the table nominated ‘Collection Period’ (figure 17). A link is provided between a staff member and a collection period, in the ‘Staff Return’ table described in figure 31, allowing the system to log the date of submission and the collection period of a return item.

Name	Null?	Type
-----	-----	-----
STAFF_NO	NOT NULL	VARCHAR2 (8)
COLLECT_NO	NOT NULL	NUMBER
RETURN_DATE	NOT NULL	DATE

Figure 31. Description of the STAFF_RETURN

With this functionality in mind, the 'Choose Collection Period' form, illustrated in figure 32, was developed, wherein a user may choose between viewing a previously submitted return and the submission of a new return

Staff Member		
Staff No	201877	
Surname	BOLAN	
First Names	Christopher	

Staff Returns		
View Return	Collection Period	Return Date
View Return	01/01/2000 - 31/03/2000	
View Return		
View Return		
View Return		
View Return		

New Return Back to Main Menu

Figure 32. Choose Collection Period form

The functionality of each of the buttons on the 'Choose Collection Period' form, figure 32, is as follows:

- **Back to Main Menu:** To close the 'Choose Collection Period' form, and return to the main menu.
- **New Return:** To open a LOV containing a list of all collection periods that a staff member hasn't submitted a return for, and, in consequence, to add the selected period to the 'Choose Collection Period' form data.

- **View Return:** To open an existing return related to the adjacent collection period in the 'View Return Items' form described below.

Following development of the 'Choose Collection Period' form, it became necessary to create a method of viewing submitted return items. Functionality for this was provided by a 'view return items' form, which facilitated the addition/maintenance of return items. Recall that the rules defining access to the 'Return Items' table are defined by the configuration rules maintained by the composition rule maintenance functions (discussed in Section 4.2.1.3). The 'View Return Items' form provided a focus for the translation the configuration rules into an aesthetically functional format.

Name	Null?	Type
-----	-----	----
STAFF_NO	NOT NULL	VARCHAR2 (8)
COLLECT_NO	NOT NULL	NUMBER
CAT_NO	NOT NULL	NUMBER
SUB_CAT_NO	NOT NULL	NUMBER
SUB_SUB_CAT_NO	NOT NULL	NUMBER
LINE_NO	NOT NULL	NUMBER
DISPLAY_IN_REPORTS		VARCHAR2 (1)
VCHAR_ONE		VARCHAR2 (240)
VCHAR_TWO		VARCHAR2 (240)
VCHAR_THREE		VARCHAR2 (240)
VCHAR_FOUR		VARCHAR2 (240)
VCHAR_FIVE		VARCHAR2 (240)
VCHAR_SIX		VARCHAR2 (240)
NUM_SEVEN		NUMBER
NUM_EIGHT		NUMBER
TEXT_NINE		LONG
DATE_TEN		DATE
DATE_ELEVEN		DATE

Figure 33. Description of RETURN_ITEMS

Staff Return Items		CAT_DESC		Staff No	STAFF_N
Cat No	CAT_NO	SUB_CAT_DESC		Line No	LINE_NO
Sub Cat No	SUB_CAT_N	SUB_SUB_CAT_DESC		Collect No	COLLECT_N
Sub Sub Cat No	SUB_SUB_C				
Vchar One	VCHAR_ONE				
Vchar Two	VCHAR_TWO				
Vchar Three	VCHAR_THREE				
Vchar Four	VCHAR_FOUR				
Vchar Five	VCHAR_FIVE				
Vchar Six	VCHAR_SIX				
Num Seven	NUM_SEVE				
Num Eight	NUM_EIGH				
Text Nine	TEXT_NINE				
Date Ten					
Date Eleven					
Add Item		Delete Item	Previous Item	Next Item	Back to Menu
		Save			

Figure 34. View Return Items - base level form

The placement of all columns of the ‘Return Item’ table, illustrated in figure 34, onto the ‘View Return Items’ form enabled the form to be altered physically at runtime through the use of ‘GET’ and ‘SET’ property functions. Recall that the general applicability of these functions was reviewed in section 2.3.3.2, whilst their applicability specific to this component of the study will now be elaborated.

The ‘GET’ and ‘SET’ property functions were utilised in a procedure named ‘update layout’, presented in pseudo code in figure 35 and listed in appendix D. The ‘update layout’ procedure uses the category details of the current item to access the user-defined rules stored in the ‘Composition Rules’ table, described in section 4.2.1.3. The configuration rules are used to restrict the fields that may be displayed, to establish the labels next to each of those fields, and to determine the order in which the fields appear.

```

BEGIN
  GET CURRENT CATEGORY VALUES
  IF USER IS ADMINISTRATOR
    SHOW DISPLAY_IN_REPORTS FIELD
  ELSE
    HIDE DISPLAY_IN_REPORTS FIELD
  END IF

  FOR EACH COLUMN IN THE RETURN_ITEMS TABLE
    SET ITEM_NAME TO CURRENT ITEM
    CHECK COMPOSITION_RULE DATA ON ITEM
    IF COLUMN IS NOT NEEDED
      HIDE THE ITEM
    ELSE
      SHOW THE ITEM
      DISPLAY RELATED ITEM PROMPT
      POSITION ITEM ON SCREEN
    END IF
  END LOOP
END

```

Figure 35. Update Layout pseudo code

As the format of the display may have changed since its last invocation, selection of any of the 'Next', 'Previous', 'Add', and 'Delete' buttons triggers the procedure 'update layout' following standard Oracle navigation features, so that the contents of the 'View Return Items' screen are consistent with the form's underlying configuration data. In addition, selection of the 'Add' button will invoke a 'Choose Category' form, depicted in figure 36, to allow a user to select a return type from a list of categories, sub categories and if necessary, sub sub categories.

The 'Choose Category' form, illustrated in figure 36, consists of three text boxes and their associated selection buttons. Each selection button is related through its internal properties to a LOV that enables the display a list of available categories, sub categories or sub sub categories. The list displayed is determined by the point of entry within the hierarchy implied in the data model shown in figure 15, i.e. the LOV showing sub categories is limited based on the previously selected category value demonstrated in figures 37 and 39.

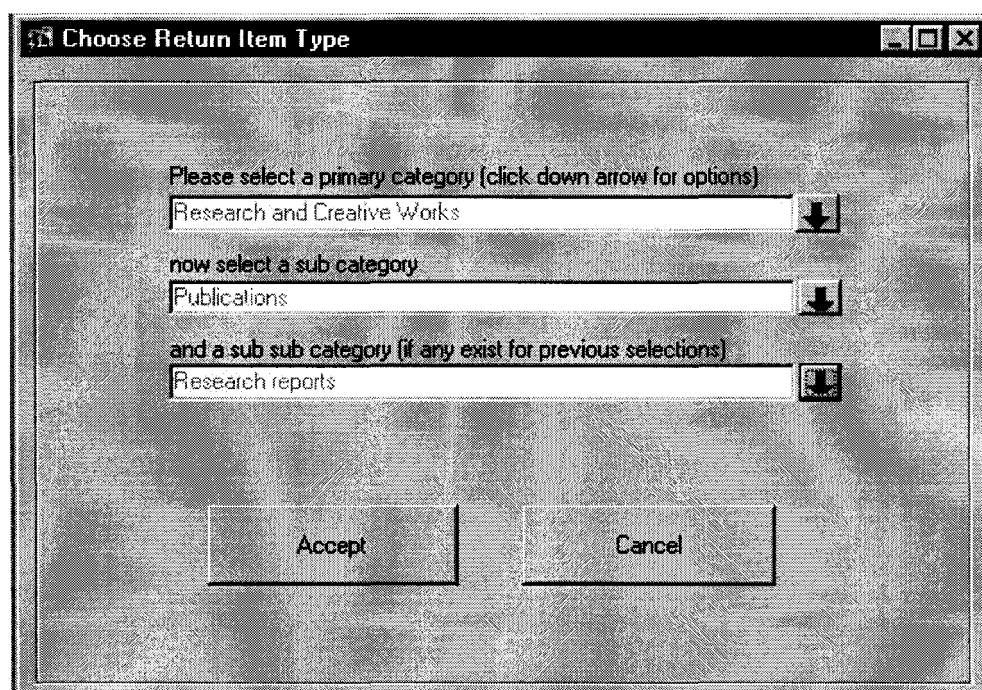


Figure 36. Choose Category form

Each of the LOVs are based upon dynamic record groups that are re-queried each time the LOV is called, ensuring the list provided is always up to date as discussed in section 2.3.3.3. The queries behind the category and subcategory record groups are illustrated in figures 38 and 40. Figures 38 and 40 are positioned adjacent to their respective LOVs in figures 37 and 39.

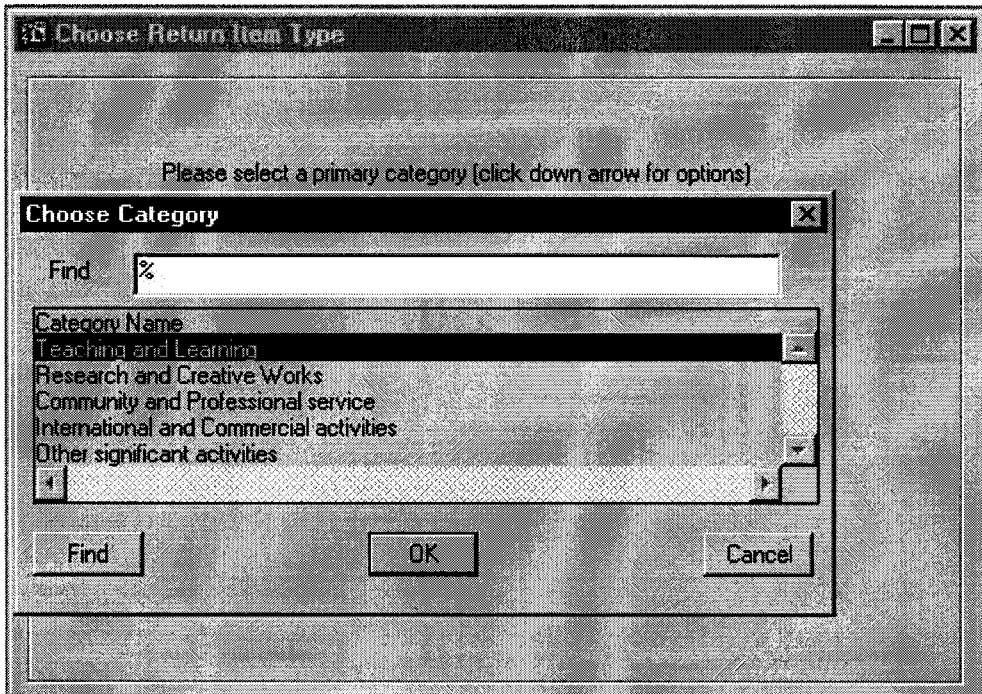


Figure 37. Choose Category form - Select a category

```
SELECT ALL CATEGORY.CAT_NO, CATEGORY.CAT_NAME
FROM CATEGORY
WHERE ((CATEGORY.CAT_DATE_REMOVED IS NULL)
OR CATEGORY.CAT_DATE_REMOVED <= SYSDATE
```

Figure 38. Choose Category Record Group

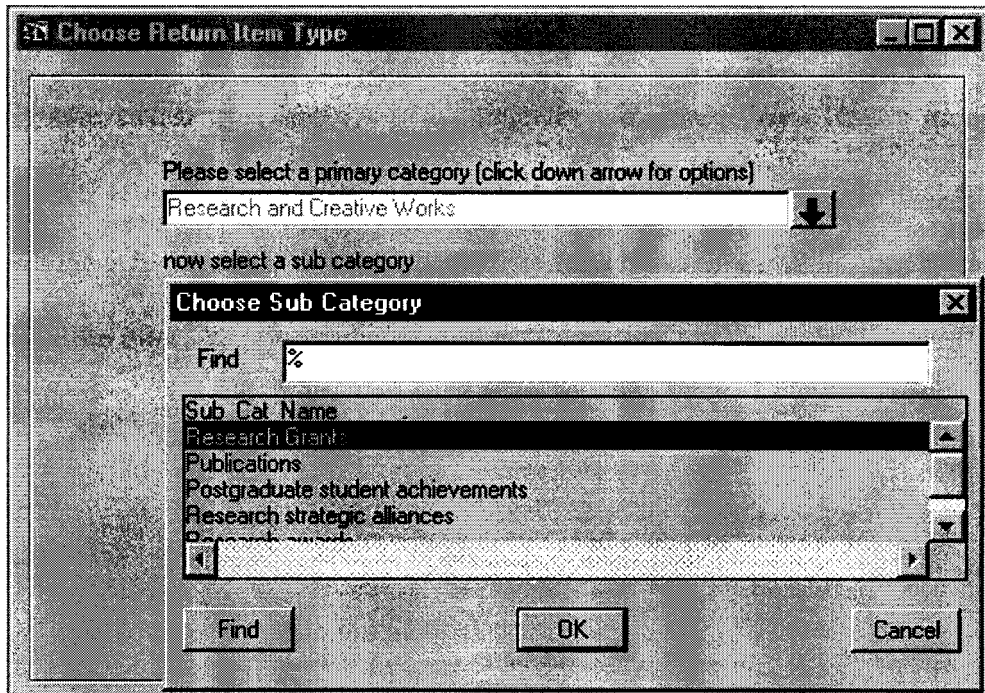


Figure 39. Choose Category form - Select a sub category

```

SELECT ALL SUB_CATEGORIES.SUB_CAT_NO,
          SUB_CATEGORIES.SUB_CAT_NAME
FROM SUB_CATEGORIES
WHERE SUB_CATEGORIES.CAT_NO = :CHOOSE_ITEM_TYPE.CAT_NO
AND SUB_CATEGORIES.SUB_SUB_CAT_NO = 0
AND ((SUB_CATEGORIES.SC_DATE_REMOVED IS NULL) OR
      (SUB_CATEGORIES.SC_DATE_REMOVED >= SYSDATE))
AND (SUB_CATEGORIES.SC_DATE_CREATED <= (SYSDATE))

```

Figure 40. Choose Subcategory Record Group

After a valid category/sub category combination has been selected on the 'Choose Category' form the 'Accept' button may be pressed, whereupon the selected category values are passed to the 'View Return Items' form in readiness to display the new data item. In addition, the 'View Return Items' form calls the 'update layout' procedure (whose pseudo-code is outlined in figure 35), thereby reconfiguring its layout to the format dictated by the data contained in the 'Composition Rules' table.

By example, two sample displays of dynamically configured 'View Return Items' forms are illustrated in figures 41 and 42. It is important to acknowledge that the setup of these two forms is entirely dependent upon the configuration rules entered via forms within the category maintenance flow (figure 18).

The screenshot shows a web form titled 'Staff Return' with the following fields and controls:

- Staff No: 201877
- Staff Name: Christopher BQLAN
- Section: **Community and Professional service**
- Sub-section: **Appointments to Boards or External Committees**
- Form fields:
 - Position
 - Name of Board/Committee
 - Nominating group
 - Date of tenure
- Control: Display In Report
- Navigation buttons: Add Item, Delete Item, Previous Item, Next Item, Back to Menu, Save

Figure 41. View Return Items - possible configuration 1

The screenshot shows a web form titled 'Staff Return' with the following fields and controls:

- Staff No: 201877
- Staff Name: Christopher BQLAN
- Section: **Research and Creative Works**
- Sub-section: **Research Grants**
- Form fields:
 - Recipient(s)
 - Funding Body
 - Title of Grant
 - Description
 - Amount granted
 - Collaborating Organisations
- Control: Display In Report
- Navigation buttons: Add Item, Delete Item, Previous Item, Next Item, Back to Menu, Save

Figure 42. View Return Items - possible configuration 2

The flow between the 'Choose Collection Period', 'View Return Items', and 'Choose Category' flexible data entry form enables the user to view or alter any type of item previously defined in the category maintenance screens. Notably, these flexible data entry forms interface with the system's static data entry forms seamlessly, hiding their underlying complexity from users of the system. The static data entry forms are outlined in the following section.

4.2.2.2 Static Data Entry

The remaining data entry forms are focused on the static part of the database illustrated in figure 17. The flow of static forms is shown in figure 43. The forms have no flexible content; accessing directly the current values of underlying database fields associated with displayed fields and, further, were developed with standard Oracle features. In short, they do not contribute to the enhancements demonstrated in the study. Accordingly, for the purpose of clarity only, they will be outlined briefly below.

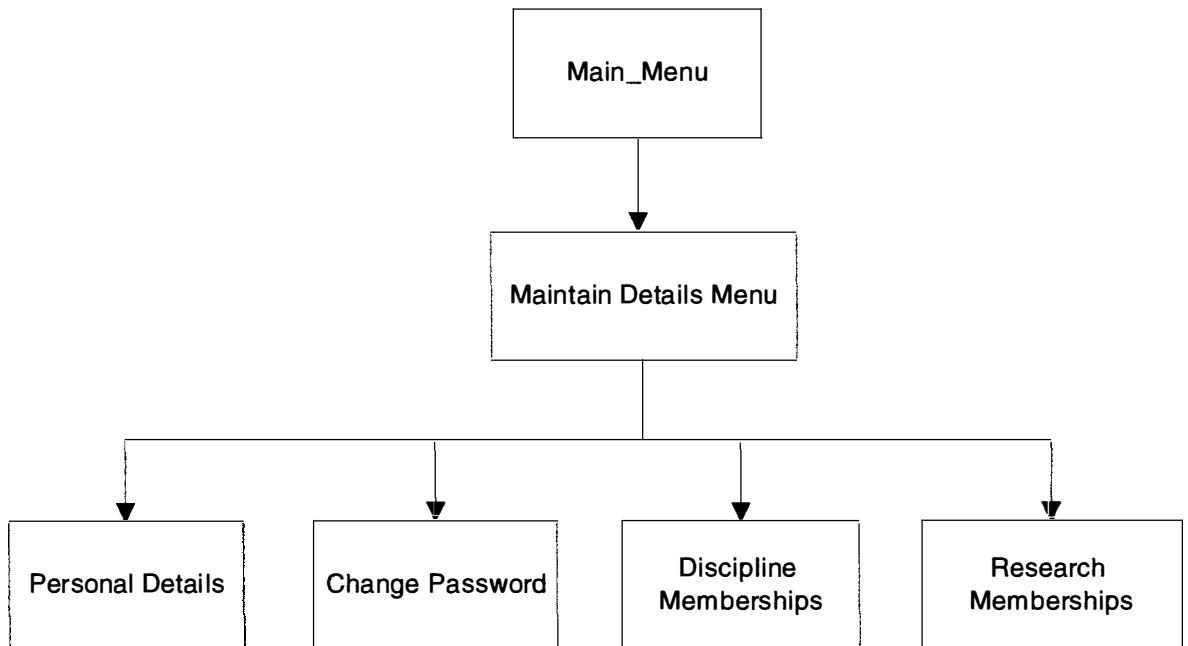


Figure 43. Static data entry flow

The system’s static data entry forms display data derived, in standard format, from databases external the Staff Submission System, and facilitate the following functionality, the flow of which is illustrated in figure 43:

- **Personal Details:** To add/alter data in the ‘STAFF’ table.
- **Change Password:** To change a user’s password.
- **Discipline Memberships:** To add/alter the Discipline Areas of which a user is a member.
- **Research Memberships:** To add/alter the Research Groups of which a user is a member.

Examples of the appearance of each of these forms are contained within appendix E. They were removed to an appendix, as they do not form part of the study’s focus and to maintain brevity within the main document and, for the benefit of the reader.

4.2.3 Reporting

The third area of the application, identified in section 4.2, is occupied with querying and reporting of data in the database. The reporting functions use the same configuration rules as the 'View Return Items' form (outlined in section 4.2.2.1), thereby fulfilling the requirement that they: "*allow easy data extraction*" identified in the statement of the problem in section 3.1.

The reporting requirement was satisfied by provision of nine reports, each developed to provide a wide range of data extraction options. In brief, the purpose of each of the reports is as follows:

- **Individual Return:** To report a complete listing of all return items submitted by an individual within a specific collection period.
- **Period Report:** To report all return items submitted within a specific collection period.
- **Specific Category:** To report all return items of a specific category, submitted within a specific collection period.
- **Rules:** To report the current configuration rules contained in the category maintenance tables (figure 15).
- **No Return:** To report names and contact details of those staff who did submit a return within a specific collection period.
- **Discipline Membership:** To report a list of staff members who belong to a specific discipline.

- **Research Membership:** To report a list of staff members belonging to a specific research group.
- **Individual Membership:** To report the disciplines and research groups to which an individual staff member belongs.
- **Staff in System:** To report a list of details for staff as stored in the database.

The flexible nature of the database necessitated the development of a SQL query akin to the ‘update layout’ procedure (represented in figure 35) so that standard Oracle Reports software might present the correct layout for each data item. The resultant SQL query, described in figure 44, followed the indirection defined by the contents of composition rules table, replicating the flexibility employed to configure the ‘View Return Items’ form (outlined in section 4.2.2.1). In other words, as will be shown in following paragraphs, the same configuration rules established for flexible user screens are employed to provide flexible system reports, thereby achieving a doubling of the rules’ efficacy.

```

SELECT
  STAFF_RETURN_ITEMS.STAFF_NO,
  STAFF_RETURN_ITEMS.line_NO,
  STAFF_RETURN_ITEMS.COLLECT_NO,
  STAFF_RETURN_ITEMS.CAT_NO,
  STAFF_RETURN_ITEMS.SUB_CAT_NO,
  STAFF_RETURN_ITEMS.SUB_SUB_CAT_NO,
  COMPOSITION_RULES.SEQUENCE_NO,
  COMPOSITION_RULES.PROMPT_NO,
  COMPOSITION_RULES.COLUMN_NO,
  PROMPTS.FIELD_PROMPT
FROM
  STAFF_RETURN_ITEMS,
  COMPOSITION_RULES,
  PROMPTS
WHERE COMPOSITION_RULES.CAT_NO = STAFF_RETURN_ITEMS.CAT_NO
  AND COMPOSITION_RULES.SUB_CAT_NO = STAFF_RETURN_ITEMS.SUB_CAT_NO
  AND COMPOSITION_RULES.SUB_SUB_CAT_NO = STAFF_RETURN_ITEMS.SUB_SUB_CAT_NO
  AND PROMPTS.PROMPT_NO = COMPOSITION_RULES.PROMPT_NO
  AND STAFF_RETURN_ITEMS.CAT_NO = :P_CAT_NO
  AND STAFF_RETURN_ITEMS.SUB_CAT_NO = :P_SUB_CAT_NO
  AND STAFF_RETURN_ITEMS.SUB_SUB_CAT_NO = :P_SUB_SUB_CAT_NO
  AND STAFF_RETURN_ITEMS.DISPLAY_IN_REPORTS = 'T'
ORDER BY SEQUENCE_NO

```

Figure 44. SQL query used to define report structure

The query listed in figure 44 returns the structure of a value in the 'Return Item' table together with its category details, providing a level of indirection to the data that will appear in the report. The method of indirection is via a function that reads the structure according to the configuration rules (described in section 4.1.2.3) before resolving and returning the data pointed to by the rules. This function was named CF_DATA and, for simplicity, is presented as pseudo code in figure 45.

```
GET THE CURRENT COLUMN NUMBER
CASE COLUMN NUMBER OF :
  1 TO 6   : RETURN VARCHAR2
  7 TO 8   : RETURN NUMBER
  9        : RETURN LONG
  10 TO 11: RETURN DATE
END CASE
```

Figure 45. CF_DATA pseudo code

The SQL query and CF_DATA functions were then merged using the Oracle Report product's graphical data model interface tools. The emergent model, illustrated in figure 46, separates the query into three groups as follows (relating to the tables shown in figure 17):

- **G_STAFF_NO**: Data related to the 'Staff Return' table.
- **G_CAT_NO**: Data related to the Category tables of 'Category' and 'Sub Category'.
- **G_FIELD_PROMPT**: Data relating to the 'Prompt' and 'Return Item' tables.

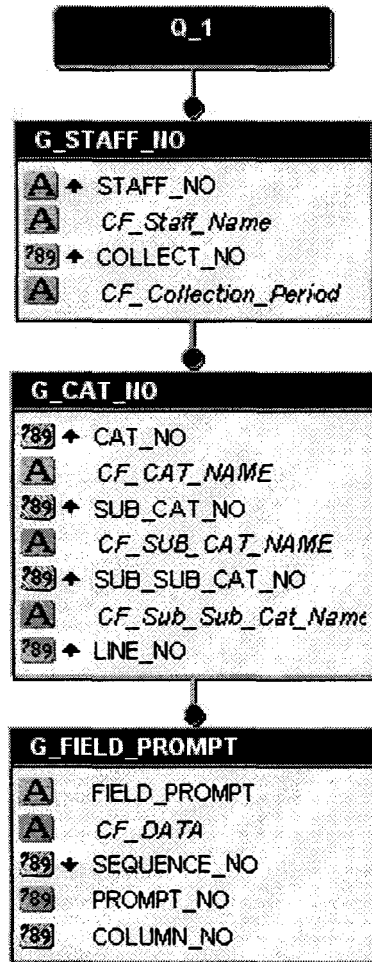


Figure 46. Oracle Reports data model

Notably, the mechanism described in this section uses the flexible data model to provide reports governed by the same configuration rules used to configure dynamically the system's user screens. To help demonstrate the flexible nature of the reports a selection of the sample reports has been included in appendix F. They were removed to an appendix to maintain brevity within the main document and, for the benefit of the reader.

4.3 Summary

This chapter has detailed the phases introduced in chapter three, namely those of analysis, design, and development of the sample application, and detailed and discussed those techniques and procedures that were used to implement them.

Specifically detailed were the construction of the staff submission system in terms of flexible 'front end' and 'back end' development to satisfy a the problem statement and associated requirements stated in chapter three.

Chapter 5: Findings

This chapter discusses the study's findings, in relation to the research question stated in chapter one as: *"How may the challenge of dynamic/flexible user requirements be met using data driven techniques in a 4GL-database environment?"* Recalling, further, that the main research question consists of two components, both are now restated and each is addressed in turn.

5.1 Findings on Research Question, part (a).

"How may one model dynamic user requirements through an extension of current data modelling techniques?"

Part (a) of the research question was responded to by developing a sample application that investigated the viability of using an extension of existing data modelling techniques to capture and store a problem that is dynamic in nature. The author, in chapter two, has reviewed the exiguous literature occupied with the capture of dynamic requirements and, further, observed that few of the techniques described were practical enough for real application. An alternative approach of extending an existing data modelling technique was proffered as being superior to the creation of new modelling techniques as the tools and language facilities are already widely available.

To address part (a), the study extended standard modelling techniques, discussed in chapter two, defined in chapter three, and demonstrated in chapter four, for capture of the dynamic data requirements of a replacement Staff Submission System for ECU.

The enhanced E-R model reflects the requirement that a systems administrator may, post-implementation, whilst the system is live, wish to alter dynamically the category hierarchy and configuration rules of the system.

The resultant E-R model, discussed in chapter four, was extended to incorporate extra data thereby allowing the storage of dynamic mapping. This extension required neither new symbols nor relationships to be added to the currently accepted E-R modelling tool set. The extension's success was validated by the ability of the Oracle CASE tool to convert the dynamic E-R model into database creation scripts, as the CASE tool strictly adheres to the standard E-R modelling precepts.

The author proposes that the successful extension of the E-R modelling technique yields two conclusions:

1. The capture of dynamic requirements using E-R modelling has been overlooked due to a lack of publications on the subject; and
2. Current, static E-R models may be extended through the study's techniques to implement preventative maintenance and to provide extensibility.

5.2 Findings on Research Question, part (b).

“How may one implement user requirements of flexibility using data driven techniques in a 4GL environment?”

In response to research question, part (b), the study examined the feasibility of implementing those dynamic requirements, captured in response to research question part (a), in a 4GL environment. As stated in chapter three, the author chose the Oracle 4GL due to its ready availability and the use of the environment by previous studies into those parallel areas reviewed in chapter two.

The user requirements, as stated in chapter three, necessitated that the application be both flexible, to prove the point of the study, and functional, to demonstrate applicability of the study to satisfy real-world problems. Thus, the development of the application extended to demonstrate flexibility that mitigated costs of potential maintenance of the application’s data entry and reporting functions.

The study resulted in the creation of a Staff Submission System that allows an advanced user to alter the appearance and functionality of specific areas of the application. The flexibility was based on a database, generated from a dynamic E-R model. Further, a 4GL Oracle forms ‘front end’ was designed to ensure the application provided a pleasant and practical means to manage the flexible aspects of the system.

The completed application has undergone extensive acceptance testing by the author and the study's clients, i.e. ECU staff, to verify system properties and, further, is undergoing commissioning and is expected to go "live" in July 2001.

Chapter 6: Conclusion

To introduce this final chapter, let us first examine the motivation of this study. The author, a programmer/analyst, is regularly contracted by organisations requiring changes to existing software systems. The author's experiences and observations, plus those gathered from colleagues and from the literature, suggest that much remedial effort involved in system alterations during maintenance stems from a lack of foresight during initial analysis and design. In short, as implied in chapter one, this study was motivated by a desire to reduce that effort involved in perfective and adaptive maintenance, thereby enhancing the efficiency of programmers involved in maintenance activities.

In chapter two, literature, concerned with existing techniques for specifying and developing dynamic/flexible systems, was reviewed. The review concluded that while limited research pointed out the lack of suitable techniques, it failed to present published methods for practical solutions to the problem. Furthermore, published research that addressed ways to minimise maintenance effort through flexibility focussed on that of the 'business rule' or 'front end', and overlooked the fundamental importance of data flexibility.

In order to demonstrate a possible solution to the emergent problem of designing and developing a system with innate data flexibility, a 'real life' Staff Submission System, required by ECU, was selected. Accordingly, the Staff Submission System was

developed with a focus on the reduction of eventual maintenance that may be achieved through flexibility.

During the application's development, flexibility was built-in into all possible aspects of the system. The resultant database incorporated dynamic mapping data that allowed runtime mapping from those database attributes, which required flexibility, to the user forms/reports where their requirements were not fully definable.

The successful creation and implementation in the study produces several implications for application designers and developers:

- A potential reduction in maintenance costs.
- The retention of use of established modelling techniques to accommodate dynamic requirements/abilities.
- The ability to develop application 'front ends' that, subsequently, may adapt to requirements changes with neither coding changes nor substantial recompilations and installations.
- An attendant increase in system life and flexibility.

Finally, the techniques established in this study are not limited to Oracle based applications and may be applied to database management systems in general.

Through their use, future maintenance efforts might be mitigated to a point of major cost reductions in software maintenance, thereby releasing funds for the development of new or enhanced products.

Glossary

ECU: Edith Cowan University, 100 Joondalup Drive, Joondalup 6027, Western Australia.

Form: A form is a collection of objects with which a user may interact in order to view and modify database tables. Forms may consist of windows, canvases, text items, buttons and other windows dialog mechanisms. Typically a form will contain a number of different, but related, blocks.

LOV: A list of values (LOV) is a modal pick list and visual presentation of data contained in a record group. From such a list, users may select a single valid value, which is normally used to populate an item.

ODMG: “the Object Database Management Group undertakes continuing work on standards for object database management systems (ODBMSs)” (“ODMG 2.0”, 1998).

Object Oriented (Approach): Comprises Object Oriented Analysis, Design and Programming, each of which, for clarity, is defined separately as follows, as taken from Booch (1994, p.38-39):

“**Object-Oriented Analysis:** a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain.

Object-Oriented Design: a method of design encompassing the process of OO decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system under design.

Object-Oriented Programming: a method of implementation in which programs are organised as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes via inherited relationships.”

PL/SQL: Procedural Language/Structured Query Language. PL/SQL is a procedural language developed by the Oracle Corporation for use in its products. It is functionally similar to many 3GLs and has a strong similarity to, and relationship with the language ADA. PL/SQL is used to provide a flexible means to enhance SQL code.

Record Groups: Structured sets of data used to facilitate interaction between the database and an application, most commonly using LOVs. Record groups are often perceived as virtual tables.

SQL: Structured Query Language is the standard language used in conjunction with relational database management systems. The American National Standards Institute (ANSI) and the International Standards Organisation (ISO) approved the standard jointly in 1992.

Triggers: A trigger is a section of code that is used to extend the functionality of an application. Each trigger contains one or more PL/SQL statements. A trigger may be associated with an event, such as when a button is selected, whereupon the trigger executes.

UML: “The Unified Modelling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artefacts of software systems, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is the successor to the modelling languages found in the Booch, OOSE/Jacobson, OMT and other methods. Many companies are incorporating the UML as a standard into their development processes and products, which cover disciplines such as business modelling, requirements management, analysis & design, programming and testing.” (“UML”, 2001).

Bibliography

- Behforooz, A. & Hudson, F, J., (1996). Software Engineering Fundamentals. Oxford University Press: Oxford, UK.
- Beynon-Davies (1996). Database Systems. Wiltshire, Great Britain: Anthony Rowe Ltd.
- Blum, B.I. (1993a). On the Engineering of Open Software Systems. International Symposium on Engineered Software Systems, (pp. 43-57). Malvern.
- Blum, B.I. (1993b). Representing Open Requirements with a Fragment-Based Specification. IEEE Transactions on Systems, Man, and Cybernetics. 23 (3), 724-736.
- Booch, G. (1994). Object-Oriented Analysis and Design with Applications. Benjamin/Cummings Publishing: California.
- Callon, J.D. (1996). Competitive advantage through Information technology. McGraw-Hill Companies: Sydney.
- Chen, P.P. (1976). The Entity-Relationship Model - Toward a Unified View of Data. ACM Transactions on Database Systems. 1 (1), 9-36.
- Codd, E.F. (1970). A Relational Model for Large Shared Data Banks. Communications of the ACM. 13 (1), 377 - 387.
- Ensor, D., & Stevenson, I. (1997). Oracle design. O'Reilly Associates: USA.
- Feuerstein, S. (1996). Advanced Oracle PL/SQL. O'Reilly Associates: USA.
- Hall, M.J. (1998). System and Database Design: Lecture 3.
- Hall, M.J. & Ligezinski.P. (1997a). Designing flexible software to accommodate dynamic user requirements: An alternative solution to a continuing IS problem. In Proceedings of World Conference on Systemics, Cybernetics and Informatics ISAS '97 Vol. 1. Caracas, Venezuela.

- Hall, M.J. & Ligezinski.P. (1997b). Developing flexible software with Oracle tools. Proceedings of Oracle Openworld 1997 Conference – Step Into the Future Today. Melbourne, Vic: ANZORA.
- Hofmann, H.F., Pfeifer, R. & Vinkhuyzen, E. (1993). Situated Software Design. Proceedings of the Fifth International Conference on Software Engineering and Knowledge Engineering. San Francisco: USA.
- Layng, M. (1998). Software Flexibility in a Web Environment. Honours dissertation, Edith Cowan University, Perth, Western Australia.
- Lientz, B. & Swanson E. (1980). Software Maintenance Management. Addison Wesley
- Liu, X., Yang, H. & Zedan, H. (1998). Improving Maintenance Through Development Experiences. Unpublished paper presented at the Fifth Workshop on Empirical Studies of Software Maintenance.
- Mehandjiev, N., & Bottaci, L. (1996). User Enhanceability for Organisational Information Systems through Visual Programming. Advanced Information Systems Engineering: 8th International Conference, CAiSE'96 (pp. 432-456) Springer-Verlag, 1996.
- Mehandjiev, N., & Bottaci, L. (1998). The Place of User Enhanceability in User - Oriented Software Development. Journal of End User Computing. 10 (2) 4-14.
- O'Connor, S. (1999). Implementing Flexible Software Techniques in a 4GL Environment. Honours dissertation, Edith Cowan University, Perth, Western Australia.
- ODMG 2.0 (1998, July). ODMG 2.0: If You Want Better Answers... [online]. Available WWW: <http://www.omdg.org> [17/09/2000].
- Oracle. (January 2001). Oracle Technet [online]. Available WWW: <http://technet.oracle.com> [26/01/2001]
- Panko, R.R., (1998). What we know about Spreadsheet errors. Journal of End User Computing 10 (2) 15-21.

- Parnas, D.L. (1979). Designing Software for Ease of Extension and Contraction. IEEE Transactions on Software Engineering. SE-5, (2) 128-137.
- Pohl, I. (1997). Object-Oriented Programming using C++ (2nd Ed.). Addison-Wesley: Sydney.
- Pressman, R.S. (1997). Software Engineering a Practitioners Approach. (4th Ed.). McGraw Hill.
- Stacey, M. (1995) Distorting Design: Unevenness as a Cognitive Dimension of Design Tools [online] Available WWW: <http://www.mk.dmu.ac.uk/~mstacey/documents/uneven-a.htm> [15/08/2000]
- UML. (2001, January). UML [online]. Available WWW: <http://www.omg.org/uml/> [10/02/2001]
- Weinberg, J. (1990). Jerry's Site [online]. Available WWW: <http://www.geraldmweinberg.com/> [02/02/2001]
- Woolfolk, W.W., Ligezinski, P. & Johnson, B. (1996). The Problem of the Dynamic Organisations and the Static System: Principles and Techniques for Achieving Flexibility. Proceedings of the 29th Annual Hawaii International Conference on Systems Science (p. 482-491).

Appendix A - Data Collection Form used in the Manual

Submission System

DATA COLLECTION FOR THE MONTHS OF: _____

ENTER INFORMATION RELATING TO COLLECTION PERIOD ONLY

PLEASE DO NOT USE ABBREVIATIONS

NAME: _____

TEACHING AND LEARNING

e.g. innovation and flexibility, internationalisation of curriculum, links with industry and professions, strategic partnerships and pathways, teaching grants and awards, achievement by undergraduate students.

RESEARCH GRANTS & ACTIVITIES

For research grants (please include amount granted, funding body, title of research project, co-researcher/s, collaborating organisations), research profile, research management, postgraduate research awards, achievements by postgraduate research students.

PUBLICATIONS

(Please give details of refereed publications in the data collection period)

Book: Authored - research:

Book Chapter (in A1 type Books):

Full written paper - refereed proceedings:

CONFERENCE PRESENTATIONS

Please give details of conferences attended, papers presented, significant role in the conference or organisation of the conference (please provide title of conference, paper presented, venue and dates).

AWARDS/RECOGNITION

Please give details of any significant staff achievements or awards (include title of award, awarding body and reason for award)

TV/RADIO/MEDIA PARTICIPATION

Please provide programme/article title, radio/TV station/newspaper, date, topic, reason for interview/article.

INTERNATIONALISATION OR INTERNATIONAL ACTIVITIES

e.g. strategic partnerships, international students, international visitors (please list name, duration and purpose of visit, activities undertaken, area of expertise, institution and country).

COMMERCIAL ACTIVITIES

Please give details of any commercial activities.

SERVICES AND SUPPORT

e.g. students (access, enrolments, support), equity and EEO, staffing (staff development, staff profile), IT, physical environment, safety.

MANAGEMENT

e.g. implementation of Strategic Plan, quality issues, general higher education issues.

STUDENT ACTIVITIES/ACHIEVEMENTS

(for inclusion in the Faculty Student Newsletter)

OTHER SIGNIFICANT ACTIVITIES (which are not covered by other categories)

Appendix B - Collection Category Setup

1. Teaching and Learning

1.1 Teaching initiatives

Name(s)

School

Description of initiative

1.2 Teaching awards and grants

Recipient(s)

Awarding Body

Title of award/grant

Description

Amount Granted

1.3 Achievements by undergraduate students

Name

School

Course

Description

Staff involved

2. Research and Creative Works

2.1 Research Grants

Recipient(s)

Funding Body

Title of research grant

Amount granted

Collaborating organisations

2.2 Publications

Book:

Author

Date

Title
Publisher
Venue

Book chapter:

Author
Date
Chapter Title
Book Title
Publisher
Venue

Article in refereed journal:

Author
Date
Title of article
Title of journal
Volume
Page numbers

Article in non-refereed journal:

Author
Date
Title of article
Title of journal
Volume
Page numbers

Article in refereed conference proceedings:

Author
Date
Title of article
Title of proceedings
Volume
Page numbers

Article in non-refereed conference proceedings:

Author
Date
Title of article
Title of proceedings
Volume
Page numbers

Paper presented at conference:

Author
Date
Title of presentation
Conference
Venue

Research reports:

Author
Date
Title
Publisher

Audio-visual recordings:

Author
Date
Title
Publisher

CD-ROM/computer software:

Author
Date
Title
Publisher

Creative works:

Presenter
Name of work
Event
Venue

2.3 Postgraduate student achievements

Name of student
School
Description of achievement
Staff involved

2.4 Research strategic alliances

Institution/organisation
Description of strategic alliance

2.5 Research Awards

Recipient
Awarding body
Title of award
Description of award

3. Community and professional service

3.1 TV/radio/media participation

Name
Programme/article title
Radio/TV station or publication
Description
Date

3.2 Appointments to Boards or External Committees

Position
Name of Board or Committee
Nominating group
Date of tenure

3.3 Awards/recognition - significant achievements or awards

Recipient

Achievement/award

Awarding body

Reason for the award

3.4 Service/Initiatives

Name

School

Description

3.5 Visitors

Name

Institution

School/Centre

Purpose

Duration

4. International and Commercial activities

4.1 Strategic alliances/partnerships

Name

School

Institution

Description

4.2 International visitors

Name

Institution

School

Purpose of visit

Duration

4.3 International students

Name

School

Description

4.4 Commercial activities

Name

School

Description

5. Other significant activities

Name

School

Description

Appendix C - Database Creation Scripts

CEEDG13.SQL

```
--Generated By: Chris Bolan
--Purpose:  Calls the necessary creation scripts needed to completely
--          create the database
SPOOL ceedg13.lst
@@ ceedg13.tab
@@ ceedg13.ind
@@ ceedg13.con
@@ ceedg13.sqs
SPOOL OFF
```

CEEDG13.TAB

```
--Generated By: Chris Bolan
--Purpose:  Creates the database tables
PROMPT Creating Table 'STAFF_RETURN'
CREATE TABLE STAFF_RETURN
  (STAFF_NO VARCHAR2(8) NOT NULL
  ,COLLECT_NO NUMBER NOT NULL
  ,RETURN_DATE DATE NOT NULL
  )
/

COMMENT ON COLUMN STAFF_RETURN.STAFF_NO IS 'Staff members staff
number, unique ID, assigned by the university'
/

COMMENT ON COLUMN STAFF_RETURN.COLLECT_NO IS 'Automatically generated
index value'
/

COMMENT ON COLUMN STAFF_RETURN.RETURN_DATE IS 'Date of the return'
/

PROMPT Creating Table 'SUB_CATEGORIES'
CREATE TABLE SUB_CATEGORIES
```

```

(CAT_NO NUMBER NOT NULL
,SUB_CAT_NO NUMBER NOT NULL
,SUB_SUB_CAT_NO NUMBER DEFAULT 0 NOT NULL
,SUB_CAT_NAME VARCHAR2(50) NOT NULL
,SUB_CAT_DESCRIPTION LONG
,NO_SUB_SUB_CATS NUMBER NOT NULL
,SC_DATE_CREATED DATE NOT NULL
,SC_DATE_REMOVED DATE
)
/

COMMENT ON COLUMN SUB_CATEGORIES.CAT_NO IS 'Auto generated index
value'
/

COMMENT ON COLUMN SUB_CATEGORIES.SUB_CAT_DESCRIPTION IS 'Description
of the sub category'
/

PROMPT Creating Table 'SCHOOLS'
CREATE TABLE SCHOOLS
(SCL_CODE VARCHAR2(6) NOT NULL
,FAC_CODE VARCHAR2(6) NOT NULL
,SCL_NAME VARCHAR2(70) NOT NULL
,SCL_DESCRIPTION LONG
)
/

COMMENT ON COLUMN SCHOOLS.SCL_CODE IS 'Automatically generated index
value'
/

COMMENT ON COLUMN SCHOOLS.FAC_CODE IS 'Automatically generated index
value'
/

COMMENT ON COLUMN SCHOOLS.SCL_NAME IS 'Name of the school'
/

COMMENT ON COLUMN SCHOOLS.SCL_DESCRIPTION IS 'Description and extra
infomation about the school'
/

```

PROMPT Creating Table 'DISCIPLINE_AREAS'

CREATE TABLE DISCIPLINE_AREAS

(DIS_CODE VARCHAR2(10) NOT NULL

,DIS_NAME VARCHAR2(50) NOT NULL

,DIS_DESCRIPTION LONG

)

/

COMMENT ON COLUMN DISCIPLINE_AREAS.DIS_CODE IS 'code'

/

COMMENT ON COLUMN DISCIPLINE_AREAS.DIS_NAME IS 'Name of discipline'

/

COMMENT ON COLUMN DISCIPLINE_AREAS.DIS_DESCRIPTION IS 'Description
and extra information about discipline'

/

PROMPT Creating Table 'PROMPTS'

CREATE TABLE PROMPTS

(PROMPT_NO NUMBER NOT NULL

,FIELD_PROMPT VARCHAR2(50) NOT NULL

,PROMPT_DESCRIPTION LONG

)

/

COMMENT ON COLUMN PROMPTS.PROMPT_NO IS 'Auto generated index value'

/

COMMENT ON COLUMN PROMPTS.FIELD_PROMPT IS 'Name of the item (Label)'

/

COMMENT ON COLUMN PROMPTS.PROMPT_DESCRIPTION IS 'Description and
information about the item'

/

PROMPT Creating Table 'RESEARCH_MEMBERSHIPS'

CREATE TABLE RESEARCH_MEMBERSHIPS

(R_GROUP_NAME VARCHAR2(60) NOT NULL

,STAFF_NO VARCHAR2(8) NOT NULL

```
,RG_START_DATE DATE DEFAULT '01-JAN-2000' NOT NULL
,RG_TERM_DATE DATE
)
/
```

```
COMMENT ON COLUMN RESEARCH_MEMBERSHIPS.R_GROUP_NAME IS 'Name of the
group'
/
```

```
COMMENT ON COLUMN RESEARCH_MEMBERSHIPS.STAFF_NO IS 'Staff members
staff number, unique ID, assigned by the university'
/
```

```
COMMENT ON COLUMN RESEARCH_MEMBERSHIPS.RG_START_DATE IS 'Start date
of membership'
/
```

```
COMMENT ON COLUMN RESEARCH_MEMBERSHIPS.RG_TERM_DATE IS 'Date
membership terminated'
/
```

```
PROMPT Creating Table 'COLLECTION_PERIODS'
```

```
CREATE TABLE COLLECTION_PERIODS
(COLLECT_NO NUMBER NOT NULL
,CP_START_DATE DATE NOT NULL
,CP_END_DATE DATE NOT NULL
)
/
```

```
COMMENT ON COLUMN COLLECTION_PERIODS.COLLECT_NO IS 'Automatically
generated index value'
/
```

```
COMMENT ON COLUMN COLLECTION_PERIODS.CP_START_DATE IS 'Date
collection starts'
/
```

```
COMMENT ON COLUMN COLLECTION_PERIODS.CP_END_DATE IS 'Date collection
ends'
/
```

```
PROMPT Creating Table 'FACULTIES'
```

```
CREATE TABLE FACULTIES
```

```
(FAC_CODE VARCHAR2(6) NOT NULL
,FAC_SHORT_CODE VARCHAR2(5) NOT NULL
,FAC_NAME VARCHAR2(70) NOT NULL
,FAC_DESCRIPTION LONG
)
/
```

```
COMMENT ON COLUMN FACULTIES.FAC_CODE IS 'Automatically generated
index value'
```

```
/
```

```
COMMENT ON COLUMN FACULTIES.FAC_NAME IS 'Name of the faculty'
```

```
/
```

```
COMMENT ON COLUMN FACULTIES.FAC_DESCRIPTION IS 'Description and extra
infomation about the faculty'
```

```
/
```

```
PROMPT Creating Table 'CATEGORY'
```

```
CREATE TABLE CATEGORY
```

```
(CAT_NO NUMBER NOT NULL
,CAT_NAME VARCHAR2(50) NOT NULL
,CAT_DESCRIPTION LONG
,CAT_DATE_CREATED DATE NOT NULL
,CAT_DATE_REMOVED DATE
)
/
```

```
COMMENT ON COLUMN CATEGORY.CAT_NO IS 'Auto generated index value'
```

```
/
```

```
COMMENT ON COLUMN CATEGORY.CAT_NAME IS 'Name of the category'
```

```
/
```

```
COMMENT ON COLUMN CATEGORY.CAT_DESCRIPTION IS 'Description of the
category'
```

```
/
```

```
PROMPT Creating Table 'STAFF_RETURN_ITEMS'
```

```
CREATE TABLE STAFF_RETURN_ITEMS
```

```
(STAFF_NO VARCHAR2(8) NOT NULL
```



```

, COLLECT_NO NUMBER NOT NULL
, CAT_NO NUMBER NOT NULL
, SUB_CAT_NO NUMBER NOT NULL
, SUB_SUB_CAT_NO NUMBER NOT NULL
, LINE_NO NUMBER NOT NULL
, DISPLAY_IN_REPORTS VARCHAR2(1) DEFAULT 'T' NOT NULL
, VCHAR_ONE VARCHAR2(240)
, VCHAR_TWO VARCHAR2(240)
, VCHAR_THREE VARCHAR2(240)
, VCHAR_FOUR VARCHAR2(240)
, VCHAR_FIVE VARCHAR2(240)
, VCHAR_SIX VARCHAR2(240)
, NUM_SEVEN NUMBER
, NUM_EIGHT NUMBER
, TEXT_NINE LONG
, DATE_TEN DATE
, DATE_ELEVEN DATE
)
/

```

```

COMMENT ON COLUMN STAFF_RETURN_ITEMS.CAT_NO IS 'Auto generated index
value'
/

```

PROMPT Creating Table 'STAFF'

```

CREATE TABLE STAFF
( STAFF_NO VARCHAR2(8) NOT NULL
, SCL_CODE VARCHAR2(6) NOT NULL
, CMP_CODE VARCHAR2(2) NOT NULL
, SURNAME VARCHAR2(50) NOT NULL
, FIRST_NAMES VARCHAR2(50) NOT NULL
, LOGIN_PIN VARCHAR2(20) NOT NULL
, SECURITY_ROLE VARCHAR2(1) NOT NULL
, DATE_OF_BIRTH DATE
, USED_NAME VARCHAR2(50)
, USER_NAME VARCHAR2(8)
, TELEPHONE_NO VARCHAR2(12)
, OFFICE VARCHAR2(8)
, EMAIL VARCHAR2(40)
, POS_CATEGORY VARCHAR2(15)

```

```

)
/

COMMENT ON COLUMN STAFF.STAFF_NO IS 'Staff members staff number,
unique ID, assigned by the university'
/

COMMENT ON COLUMN STAFF.SCL_CODE IS 'Automatically generated index
value'
/

COMMENT ON COLUMN STAFF.SURNAME IS 'Staff members surname'
/

COMMENT ON COLUMN STAFF.FIRST_NAMES IS 'Staff members first name'
/

COMMENT ON COLUMN STAFF.USED_NAME IS 'Name staff member is known by'
/

COMMENT ON COLUMN STAFF.TELEPHONE_NO IS 'Staff members contact
telephone number'
/

COMMENT ON COLUMN STAFF.OFFICE IS 'Staff members office number'
/

COMMENT ON COLUMN STAFF.EMAIL IS 'Staff members email address'
/

PROMPT Creating Table 'COMPOSITION_RULES'
CREATE TABLE COMPOSITION_RULES
(CAT_NO NUMBER NOT NULL
, SUB_CAT_NO NUMBER NOT NULL
, SUB_SUB_CAT_NO NUMBER NOT NULL
, COLUMN_NO NUMBER NOT NULL
, PROMPT_NO NUMBER NOT NULL
, DATA_TYPE VARCHAR2(10) NOT NULL
, SEQUENCE_NO NUMBER(2) NOT NULL
)
/

```

```
COMMENT ON COLUMN COMPOSITION_RULES.CAT_NO IS 'Auto generated index value'
```

```
/
```

```
COMMENT ON COLUMN COMPOSITION_RULES.COLUMN_NO IS 'Number of the column the item is to be stored in'
```

```
/
```

```
COMMENT ON COLUMN COMPOSITION_RULES.PROMPT_NO IS 'Auto generated index value'
```

```
/
```

```
PROMPT Creating Table 'RESEARCH_GROUPS'
```

```
CREATE TABLE RESEARCH_GROUPS
```

```
(R_GROUP_NAME VARCHAR2(60) NOT NULL
```

```
,GRP_LEADER_NO VARCHAR2(8)
```

```
,GRP_ADDRESS VARCHAR2(50)
```

```
,GRP_INFO LONG
```

```
)
```

```
/
```

```
COMMENT ON COLUMN RESEARCH_GROUPS.R_GROUP_NAME IS 'Name of the group'
```

```
/
```

```
COMMENT ON COLUMN RESEARCH_GROUPS.GRP_LEADER_NO IS 'Staff members staff number, unique ID, assigned by the university'
```

```
/
```

```
COMMENT ON COLUMN RESEARCH_GROUPS.GRP_ADDRESS IS 'Mailing address of the group'
```

```
/
```

```
COMMENT ON COLUMN RESEARCH_GROUPS.GRP_INFO IS 'Information about the group'
```

```
/
```

```
PROMPT Creating Table 'DISCIPLINE_MEMBERSHIPS'
```

```
CREATE TABLE DISCIPLINE_MEMBERSHIPS
```

```
(STAFF_NO VARCHAR2(8) NOT NULL
```

```
,DIS_CODE VARCHAR2(10) NOT NULL
```

```
)
```

/

COMMENT ON COLUMN DISCIPLINE_MEMBERSHIPS.STAFF_NO IS 'Staff members
staff number, unique ID, assigned by the university'

/

COMMENT ON COLUMN DISCIPLINE_MEMBERSHIPS.DIS_CODE IS 'code'

/

PROMPT Creating Table 'CAMPUS'

CREATE TABLE CAMPUS

(CMP_CODE VARCHAR2(2) NOT NULL
,CMP_NAME VARCHAR2(100) NOT NULL
,CMP_ADDRESS VARCHAR2(100)
,CMP_COUNTRY VARCHAR2(100)
)

/

PROMPT Creating Table 'BLANK_COMMIT'

CREATE TABLE BLANK_COMMIT

(BC_DUMMY VARCHAR2(1))

/

CEEDG13.IND

--Generated By: Chris Bolan

--Purpose: Creates indexes on table columns to facilitate faster
-- searching

PROMPT Creating Index 'RET_STF_FK_I'

CREATE INDEX RET_STF_FK_I ON STAFF_RETURN
(STAFF_NO)

/

PROMPT Creating Index 'RET_COL_PRD_FK_I'

CREATE INDEX RET_COL_PRD_FK_I ON STAFF_RETURN
(COLLECT_NO)

/

PROMPT Creating Index 'SUB_CAT_CAT_FK_I'

CREATE INDEX SUB_CAT_CAT_FK_I ON SUB_CATEGORIES
(CAT_NO)

/

PROMPT Creating Index 'SCH_FAC_FK_I'

```
CREATE INDEX SCH_FAC_FK_I ON SCHOOLS
  (FAC_CODE)
```

/

PROMPT Creating Index 'RSH_MBRSHF_STF_FK_I'

```
CREATE INDEX RSH_MBRSHF_STF_FK_I ON RESEARCH_MEMBERSHIPS
  (STAFF_NO)
```

/

PROMPT Creating Index 'RSH_MBRSHF_RSH_GRP_FK_I'

```
CREATE INDEX RSH_MBRSHF_RSH_GRP_FK_I ON RESEARCH_MEMBERSHIPS
  (R_GROUP_NAME)
```

/

PROMPT Creating Index 'RET_ITM_RET_FK_I'

```
CREATE INDEX RET_ITM_RET_FK_I ON STAFF_RETURN_ITEMS
  (STAFF_NO
  , COLLECT_NO)
```

/

PROMPT Creating Index 'RTN_ITM_SUB_CAT_FK_I'

```
CREATE INDEX RTN_ITM_SUB_CAT_FK_I ON STAFF_RETURN_ITEMS
  (CAT_NO
  , SUB_CAT_NO
  , SUB_SUB_CAT_NO)
```

/

PROMPT Creating Index 'STF_SCH_FK_I'

```
CREATE INDEX STF_SCH_FK_I ON STAFF
  (SCL_CODE)
```

/

PROMPT Creating Index 'STF_CMP_FK_I'

```
CREATE INDEX STF_CMP_FK_I ON STAFF
  (CMP_CODE)
```

/

```
PROMPT Creating Index 'COM_RUL_COL_NO'
CREATE UNIQUE INDEX COM_RUL_COL_NO ON COMPOSITION_RULES
  (CAT_NO
  ,SUB_CAT_NO
  ,SUB_SUB_CAT_NO
  ,COLUMN_NO)
/
```

```
PROMPT Creating Index 'COM_RUL_SUB_CAT_FK_I'
CREATE INDEX COM_RUL_SUB_CAT_FK_I ON COMPOSITION_RULES
  (CAT_NO
  ,SUB_CAT_NO
  ,SUB_SUB_CAT_NO)
/
```

```
PROMPT Creating Index 'COM_RUL_DAT_ITM_FK_I'
CREATE INDEX COM_RUL_DAT_ITM_FK_I ON COMPOSITION_RULES
  (PROMPT_NO)
/
```

```
PROMPT Creating Index 'COM_RUL_SEQ_NO'
CREATE UNIQUE INDEX COM_RUL_SEQ_NO ON COMPOSITION_RULES
  (CAT_NO
  ,SUB_CAT_NO
  ,SUB_SUB_CAT_NO
  ,SEQUENCE_NO)
/
```

```
PROMPT Creating Index 'RSH_GRP_STF_FK_I'
CREATE INDEX RSH_GRP_STF_FK_I ON RESEARCH_GROUPS
  (GRP_LEADER_NO)
/
```

```
PROMPT Creating Index 'DISCP_MBR_STF_FK_I'
CREATE INDEX DISCP_MBR_STF_FK_I ON DISCIPLINE_MEMBERSHIPS
  (STAFF_NO)
/
```

```
PROMPT Creating Index 'DISCP_MBR_DISP_AREA_FK_I'
CREATE INDEX DISCP_MBR_DISP_AREA_FK_I ON DISCIPLINE_MEMBERSHIPS
```

(DIS_CODE)

/

CEEDG13.CON

--Generated By: Chris Bolan

--Purpose: Enforces inter-table constraints and keys

PROMPT Creating Primary Key on 'STAFF_RETURN'

ALTER TABLE STAFF_RETURN

ADD CONSTRAINT RET_PK PRIMARY KEY

(STAFF_NO

,COLLECT_NO)

/

PROMPT Creating Primary Key on 'SUB_CATEGORIES'

ALTER TABLE SUB_CATEGORIES

ADD CONSTRAINT SUB_CAT_PK PRIMARY KEY

(CAT_NO

,SUB_CAT_NO

,SUB_SUB_CAT_NO)

/

PROMPT Creating Primary Key on 'SCHOOLS'

ALTER TABLE SCHOOLS

ADD CONSTRAINT SCH_PK PRIMARY KEY

(SCL_CODE)

/

PROMPT Creating Primary Key on 'DISCIPLINE_AREAS'

ALTER TABLE DISCIPLINE_AREAS

ADD CONSTRAINT DISP_AREA_PK PRIMARY KEY

(DIS_CODE)

/

PROMPT Creating Primary Key on 'PROMPTS'

ALTER TABLE PROMPTS

ADD CONSTRAINT DAT_ITM_PK PRIMARY KEY

(PROMPT_NO)

/

PROMPT Creating Primary Key on 'RESEARCH_MEMBERSHIPS'

```
ALTER TABLE RESEARCH_MEMBERSHIPS
ADD CONSTRAINT RSH_MBRSHPK PRIMARY KEY
(STAFF_NO
,R_GROUP_NAME
, RG_START_DATE)
/
```

PROMPT Creating Primary Key on 'COLLECTION_PERIODS'

```
ALTER TABLE COLLECTION_PERIODS
ADD CONSTRAINT COL_PRD_PK PRIMARY KEY
(COLLECT_NO)
/
```

PROMPT Creating Primary Key on 'FACULTIES'

```
ALTER TABLE FACULTIES
ADD CONSTRAINT FAC_PK PRIMARY KEY
(FAC_CODE)
/
```

PROMPT Creating Primary Key on 'CATEGORY'

```
ALTER TABLE CATEGORY
ADD CONSTRAINT CAT_PK PRIMARY KEY
(CAT_NO)
/
```

PROMPT Creating Primary Key on 'STAFF_RETURN_ITEMS'

```
ALTER TABLE STAFF_RETURN_ITEMS
ADD CONSTRAINT RTN_ITM_PK PRIMARY KEY
(STAFF_NO
, COLLECT_NO
, CAT_NO
, SUB_CAT_NO
, SUB_SUB_CAT_NO
, LINE_NO)
/
```

PROMPT Creating Primary Key on 'STAFF'

```
ALTER TABLE STAFF
ADD CONSTRAINT STF_PK PRIMARY KEY
```



```

(STAFF_NO)
/

PROMPT Creating Primary Key on 'COMPOSITION_RULES'
ALTER TABLE COMPOSITION_RULES
ADD CONSTRAINT COM_RUL_PK PRIMARY KEY
(CAT_NO
, SUB_CAT_NO
, SUB_SUB_CAT_NO
, COLUMN_NO)
/

PROMPT Creating Primary Key on 'RESEARCH_GROUPS'
ALTER TABLE RESEARCH_GROUPS
ADD CONSTRAINT RSH_GRP_PK PRIMARY KEY
(R_GROUP_NAME)
/

PROMPT Creating Primary Key on 'DISCIPLINE_MEMBERSHIPS'
ALTER TABLE DISCIPLINE_MEMBERSHIPS
ADD CONSTRAINT DISCP_MBR_PK PRIMARY KEY
(STAFF_NO
, DIS_CODE)
/

PROMPT Creating Primary Key on 'CAMPUS'
ALTER TABLE CAMPUS
ADD CONSTRAINT CMP_PK PRIMARY KEY
(CMP_CODE)
/

PROMPT Creating Check Constraints on 'SCHOOLS'
ALTER TABLE SCHOOLS
ADD CONSTRAINT SCHOOLS_CK CHECK ((SCL_CODE= UPPER(SCL_CODE)))
/

PROMPT Creating Check Constraints on 'FACULTIES'
ALTER TABLE FACULTIES
ADD CONSTRAINT FACULTIES_CK CHECK ((FAC_CODE = UPPER(FAC_CODE)))
/

```

PROMPT Creating Check Constraints on 'STAFF'

ALTER TABLE STAFF

ADD CONSTRAINT STAFF_SURNAME_UPPER CHECK (SURNAME = UPPER(SURNAME))

/

PROMPT Creating Check Constraints on 'COMPOSITION_RULES'

ALTER TABLE COMPOSITION_RULES

ADD CONSTRAINT AVCON_3421_COLUM_000 CHECK (COLUMN_NO BETWEEN 1 AND 11)

ADD CONSTRAINT AVCON_3421_DATA__000 CHECK (DATA_TYPE IN ('MEMO', 'NUMBER', 'VARCHAR2', 'DATE'))

ADD CONSTRAINT AVCON_3421_SEQUE_000 CHECK (SEQUENCE_NO BETWEEN 1 AND 11)

/

PROMPT Creating Check Constraints on 'CAMPUS'

ALTER TABLE CAMPUS

ADD CONSTRAINT CAMPUS_CK CHECK ((CMP_CODE = UPPER(CMP_CODE)))

/

PROMPT Creating Foreign Keys on 'STAFF_RETURN'

ALTER TABLE STAFF_RETURN ADD CONSTRAINT

RET_STF_FK FOREIGN KEY

(STAFF_NO) REFERENCES STAFF

(STAFF_NO) ADD CONSTRAINT

RET_COL_PRD_FK FOREIGN KEY

(COLLECT_NO) REFERENCES COLLECTION_PERIODS

(COLLECT_NO)

/

PROMPT Creating Foreign Keys on 'SUB_CATEGORIES'

ALTER TABLE SUB_CATEGORIES ADD CONSTRAINT

CAT_SUBCAT_FK FOREIGN KEY

(CAT_NO) REFERENCES CATEGORY

(CAT_NO)

/

PROMPT Creating Foreign Keys on 'SCHOOLS'

ALTER TABLE SCHOOLS ADD CONSTRAINT

SCH_FAC_FK FOREIGN KEY

```

        (FAC_CODE) REFERENCES FACULTIES
        (FAC_CODE)
/

PROMPT Creating Foreign Keys on 'RESEARCH_MEMBERSHIPS'
ALTER TABLE RESEARCH_MEMBERSHIPS ADD CONSTRAINT
RSH_MBRSHP_STF_FK FOREIGN KEY
    (STAFF_NO) REFERENCES STAFF
    (STAFF_NO) ADD CONSTRAINT
RSH_MBRSHP_GRP_FK FOREIGN KEY
    (R_GROUP_NAME) REFERENCES RESEARCH_GROUPS
    (R_GROUP_NAME)
/

PROMPT Creating Foreign Keys on 'STAFF_RETURN_ITEMS'
ALTER TABLE STAFF_RETURN_ITEMS ADD CONSTRAINT
RTI_SUB_CAT_FK FOREIGN KEY
    (SUB_SUB_CAT_NO
    ,CAT_NO
    ,SUB_CAT_NO) REFERENCES SUB_CATEGORIES
    (SUB_SUB_CAT_NO
    ,CAT_NO
    ,SUB_CAT_NO) ADD CONSTRAINT
RTN_ITM_RET_FK FOREIGN KEY
    (STAFF_NO
    ,COLLECT_NO) REFERENCES STAFF_RETURN
    (STAFF_NO
    ,COLLECT_NO)
/

PROMPT Creating Foreign Keys on 'STAFF'
ALTER TABLE STAFF ADD CONSTRAINT
STF_SCH_FK FOREIGN KEY
    (SCL_CODE) REFERENCES SCHOOLS
    (SCL_CODE) ADD CONSTRAINT
STF_CMP_FK FOREIGN KEY
    (CMP_CODE) REFERENCES CAMPUS
    (CMP_CODE)
/

```

PROMPT Creating Foreign Keys on 'COMPOSITION_RULES'

ALTER TABLE COMPOSITION_RULES ADD CONSTRAINT

CAT_COM_RUL_FK FOREIGN KEY
(SUB_CAT_NO
,SUB_SUB_CAT_NO
,CAT_NO) REFERENCES SUB_CATEGORIES

(SUB_CAT_NO
,SUB_SUB_CAT_NO
,CAT_NO) ADD CONSTRAINT

COM_RUL_PRMP_T_FK FOREIGN KEY

(PROMPT_NO) REFERENCES PROMPTS
(PROMPT_NO)

/

PROMPT Creating Foreign Keys on 'RESEARCH_GROUPS'

ALTER TABLE RESEARCH_GROUPS ADD CONSTRAINT

RSH_GRP_LDR_FK FOREIGN KEY
(GRP_LEADER_NO) REFERENCES STAFF
(STAFF_NO)

/

PROMPT Creating Foreign Keys on 'DISCIPLINE_MEMBERSHIPS'

ALTER TABLE DISCIPLINE_MEMBERSHIPS ADD CONSTRAINT

DISCP_MBR_STF_FK FOREIGN KEY
(STAFF_NO) REFERENCES STAFF

(STAFF_NO) ADD CONSTRAINT

DISCP_AREA_MBR_FK FOREIGN KEY

(DIS_CODE) REFERENCES DISCIPLINE_AREAS
(DIS_CODE)

/

CEEDG13.SQS

--Generated By: Chris Bolan

--Purpose: Defines sequences

PROMPT Creating Sequence 'PROMPT_SEQ'

CREATE SEQUENCE PROMPT_SEQ

NOMAXVALUE

NOMINVALUE

NOCYCLE

NOCACHE

/

PROMPT Creating Sequence 'LINE_SEQ'

CREATE SEQUENCE LINE_SEQ

NOMAXVALUE

NOMINVALUE

NOCYCLE

NOCACHE

/

PROMPT Creating Sequence 'COLLECT_SEQ'

CREATE SEQUENCE COLLECT_SEQ

NOMAXVALUE

NOMINVALUE

NOCYCLE

NOCACHE

/

Appendix D - Update Layout Procedure

PROCEDURE Update_Layout IS

Item_Id Item; --Stores Item ID Numbers

Item_Counter Number; --Loop counter 1..11

Column_Present Number; --Stores column number of a column to be shown

Category Number; --Stores current category #

Sub_Category Number; --Stores current sub category #

Sub_Sub_Category Number; --Stores current sub sub category #

Item_Prompt_No Number; --Stores the Prompt_No to locate correct text_prompt

Position_No Number; --Stores the sequence number of a field

Seq_Locator Number; --Stores the sequence number of the text field

End_Or_Null Number; --Stores the number of items in a return

item_name Varchar2(12); --Stores the item name of a field

New_Prompt_Text Varchar2(50); --Stores the Data_Item Text prompt for a field

Staff_Member Varchar2(8); -- The current staff member

BEGIN

--Stores current value of Cat values. These might need to be inputs

Category := Name_IN('Staff_Return_Items.Cat_No');

Sub_Category := Name_IN('Staff_Return_Items.Sub_Cat_No');

Sub_Sub_Category := Name_IN('Staff_Return_Items.Sub_Sub_Cat_No');

End_Or_NULL := 0;

IF :Global.Access_Level = 'A' OR :Global.Access_Level = 'D' THEN

Staff_Member := :Global.Shadow_ID;

ELSE

Staff_Member := :Global.Login_ID;

END IF;

--Allows Display in report field to be used by advanced users

Item_id := Find_Item('Staff_Return_Items.Display_In_Reports'); --Find the unique item id

IF :GLOBAL.Access_Level = 'A' OR :GLOBAL.Access_Level = 'A' THEN

Set_Item_Property(item_id,Visible,PROPERTY_TRUE);

Set_Item_Property(item_id,Enabled,PROPERTY_TRUE);

Set_Item_Property(item_id,Update_Allowed,PROPERTY_TRUE);

Set_Item_Property(item_id,Required,PROPERTY_FALSE);

Set_Item_Property(item_id,Queryable,PROPERTY_TRUE);

```

ELSE
    Set_Item_Property(item_id,Visible,PROPERTY_FALSE);
END IF;

--Checks to See if a Text field is included and marks is position
BEGIN
    SELECT Sequence_No
        INTO Seq_Locator
        FROM Composition_Rules C
        WHERE C.Cat_No = Category
            AND C.Sub_Cat_No = Sub_Category
            AND C.Sub_Sub_Cat_No = Sub_Sub_Category
            AND C.Column_No = 9;
    EXCEPTION
        WHEN NO_DATA_FOUND THEN --No Text Field
            Seq_Locator := 0;
    END;

FOR item_counter IN 1..11 --For Each Column
LOOP
    --Assigns which item will be tested in this iteration
    IF Item_Counter = 1 THEN
        item_name := 'VCHAR_ONE';
    ELSIF Item_Counter = 2 THEN
        item_name := 'VCHAR_TWO';
    ELSIF Item_Counter = 3 THEN
        item_name := 'VCHAR_THREE';
    ELSIF Item_Counter = 4 THEN
        item_name := 'VCHAR_FOUR';
    ELSIF Item_Counter = 5 THEN
        item_name := 'VCHAR_FIVE';
    ELSIF Item_Counter = 6 THEN
        item_name := 'VCHAR_SIX';
    ELSIF Item_Counter = 7 THEN
        item_name := 'NUM_SEVEN';
    ELSIF Item_Counter = 8 THEN
        item_name := 'NUM_EIGHT';
    ELSIF Item_Counter = 9 THEN
        item_name := 'TEXT_NINE';
    ELSIF Item_Counter = 10 THEN

```

```

    item_name := 'DATE_TEN';
ELSIF Item_Counter = 11 THEN
    item_name := 'DATE_ELEVEN';
END IF;

--See if the column is needed (0 = No)
BEGIN
    SELECT Column_No, Prompt_No, Sequence_No
        INTO Column_Present, Item_Prompt_No, Position_No
        FROM Composition_Rules C
        WHERE C.Cat_No = Category
            AND C.Sub_Cat_No = Sub_Category
            AND C.Sub_Sub_Cat_No = Sub_Sub_Category
            AND C.Column_No = Item_Counter;
EXCEPTION
    WHEN NO_DATA_FOUND THEN --Column isn't needed
        Column_Present := 0;
        Item_Prompt_No := 0;
        Position_No := 0;
END;

Item_id := Find_Item('Staff_Return_Items.'||item_name); --Find the unique item id

IF Column_Present = 0 THEN --Column isn't needed and is made invisible
    Set_Item_Property(item_id,Visible,PROPERTY_FALSE);
ELSE --Column needs to be shown
    Set_Item_Property(item_id,Visible,PROPERTY_TRUE);
    Set_Item_Property(item_id,Enabled,PROPERTY_TRUE);
    Set_Item_Property(item_id,Update_Allowed,PROPERTY_TRUE);
    Set_Item_Property(item_id,Required,PROPERTY_FALSE);
    Set_Item_Property(item_id,Queryable,PROPERTY_TRUE);
    End_Or_NULL := End_Or_NULL + 1;

--Changes the prompt
SELECT Field_Prompt
    INTO New_Prompt_Text
    FROM PROMPTS
    WHERE Prompt_NO = Item_Prompt_No;

```



```

Set_Item_Property(item_id,Prompt_Text,New_Prompt_Text);

--Arranges Fields in correct order
IF Seq_Locator = 0 THEN --No text field
    Set_Item_Property(item_id,Position,140,(140 + ((Position_No - 1) * 14)));
ELSIF Position_No <= Seq_Locator THEN --Text field but field occurs before it
    Set_Item_Property(item_id,Position,140,(140 + ((Position_No - 1) * 14)));
ELSE --Field occurs after text field
    Set_Item_Property(item_id,Position,140,(140 + ((Position_No - 1) * 14) + 42));
END IF;
END IF; --End of Column checking IF
END LOOP; --End of Item_counter Loop

IF ((End_Or_NULL = 0) AND (:GLOBAL.Force_Choice = 'TRUE')
AND (:Global.Reject_Force = 'FALSE')) THEN
    Launch_Choose_Screen;
    :GLOBAL.Force_Choice := 'FALSE';
END IF;
END; --End of Trigger

```

Appendix E - Static Data Entry Screens

Personal Details

Staff Details

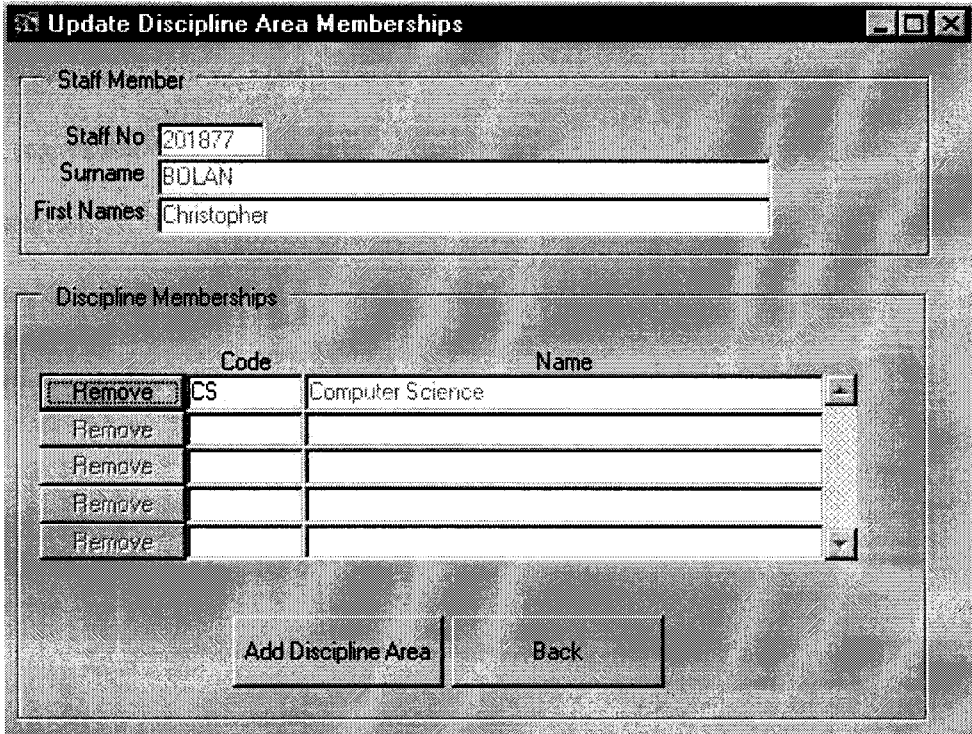
Staff No	<input type="text" value="201877"/>
Surname	<input type="text" value="BOLAN"/>
First Names	<input type="text" value="Christopher"/>
Used Name	<input type="text"/>
User Name	<input type="text"/>
Date Of Birth	<input type="text" value="01-JAN-1900"/>
Faculty	<input type="text" value="Communications, Health and Science"/>
School	<input type="text" value="Computer and Information Science"/>
Campus	<input type="text" value="Churchlands"/>
Pos Category	<input type="text" value="ACADEMIC"/>
Security Role	<input type="text" value="A"/>
Telephone No	<input type="text" value="5581"/>
Office	<input type="text" value="5.102"/>
Email	<input type="text"/>

Change Password

LOGIN

Current Password	<input type="text"/>
New Password	<input type="text"/>
Re-type New Password	<input type="text"/>

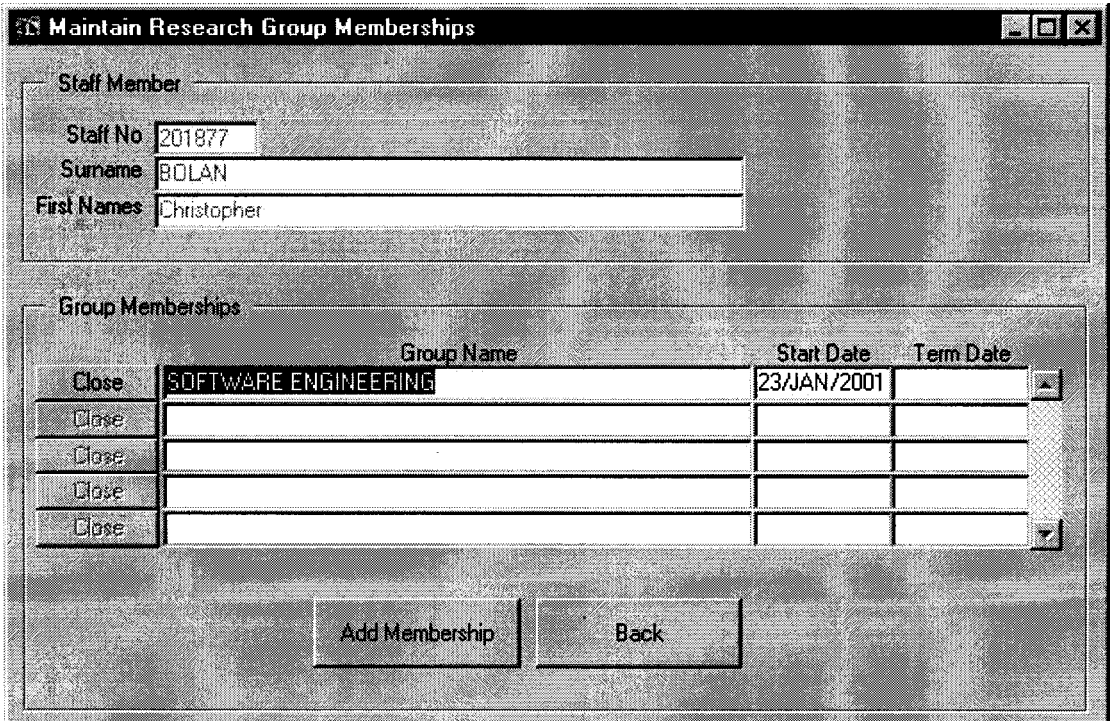
Discipline Memberships



The screenshot shows a window titled "Update Discipline Area Memberships". It contains a "Staff Member" section with three text boxes: "Staff No" (201877), "Surname" (BDLAN), and "First Names" (Christopher). Below this is a "Discipline Memberships" section with a table. The table has columns for "Code" and "Name". The first row contains "CS" and "Computer Science". Each row has a "Remove" button to its left. At the bottom of the window are two buttons: "Add Discipline Area" and "Back".

	Code	Name
Remove	CS	Computer Science
Remove		
Remove		
Remove		

Research Memberships



The screenshot shows a window titled "Maintain Research Group Memberships". It contains a "Staff Member" section with three text boxes: "Staff No" (201877), "Surname" (BDLAN), and "First Names" (Christopher). Below this is a "Group Memberships" section with a table. The table has columns for "Group Name", "Start Date", and "Term Date". The first row contains "SOFTWARE ENGINEERING", "23/JAN/2001", and an empty field. Each row has a "Close" button to its left. At the bottom of the window are two buttons: "Add Membership" and "Back".

	Group Name	Start Date	Term Date
Close	SOFTWARE ENGINEERING	23/JAN/2001	
Close			
Close			
Close			

Appendix F - Sample Reports

This appendix contains reports generated by the sample application, in a pdf (Adobe Acrobat) format. Each report is preceded by a title page, describing its use.

Composition Rule Report

This report details the configuration rules currently applied to the database and forms/reports.

Staff Return Items Composition Rule Report 13/02/2001

Category 1 Teaching and Learning

Sub category 1 Teaching initiatives

Column No Prompt No Field Prompt

1 1 Name(s)

2 2 School

9 3 Description

Category 1 Teaching and Learning

Sub category 2 Teaching awards and grants

Column No Prompt No Field Prompt

1 4 Recipient(s)

2 5 Awarding Body

3 6 Title of Award

4 3 Description

7 7 Amount granted

Category 1 Teaching and Learning

Sub category 3 Achievements by undergraduate students

Column No Prompt No Field Prompt

1 1 Name(s)

2 2 School

3 8 Course

9 3 Description

4 9 Staff Involved

Category 2 Research and Creative Works

Sub category 1 Research Grants

Column No Prompt No Field Prompt

1 4 Recipient(s)

2 10 Funding Body

3 11 Title of Grant

9 3 Description

7 7 Amount granted

4 12 Collaborating Organisations

Staff Return Items Composition Rule Report 13/02/2001

Category 2 Research and Creative Works
Sub category 3 Postgraduate student achievements

Column No	Prompt No	Field Prompt
1	1	Name(s)
2	2	School
9	3	Description
3	9	Staff Involved

Category 2 Research and Creative Works
Sub category 4 Research strategic alliances

Column No	Prompt No	Field Prompt
1	27	Institution/Organisation
9	3	Description

Category 2 Research and Creative Works
Sub category 5 Research awards

Column No	Prompt No	Field Prompt
1	4	Recipient(s)
2	5	Awarding Body
9	3	Description
3	6	Title of Award

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 1 Book

Column No	Prompt No	Field Prompt
1	13	Author(s)
10	14	Date
2	15	Title
3	16	Publisher
4	17	Venue

Staff Return Items Composition Rule Report 13/02/2001

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 2 Book chapter

Column No	Prompt No	Field Prompt
1	13	Author(s)
10	14	Date
2	18	Title of chapter
3	15	Title
4	16	Publisher
5	17	Venue

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 3 Article refereed journal

Column No	Prompt No	Field Prompt
1	13	Author(s)
10	49	Year
2	19	Title of article
3	20	Title of journal
4	21	Volume
5	22	Page numbers

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 4 Article in non-refereed journal

Column No	Prompt No	Field Prompt
1	13	Author(s)
10	14	Date
2	19	Title of article
3	20	Title of journal
4	21	Volume
5	22	Page numbers

Staff Return Items Composition Rule Report 13/02/2001

Category	2	Research and Creative Works
Sub category	2	Publications
Sub Sub Category	4	Article in non-refereed journal

Category	2	Research and Creative Works
Sub category	2	Publications
Sub Sub Category	5	Article in refereed conference proceedings

Column No	Prompt No	Field Prompt
1	13	Author(s)
10	14	Date
2	19	Title of article
3	23	Title of proceedings
5	22	Page numbers

Category	2	Research and Creative Works
Sub category	2	Publications
Sub Sub Category	6	Article in non-refereed conference proceedings

Column No	Prompt No	Field Prompt
1	13	Author(s)
10	14	Date
2	19	Title of article
3	23	Title of proceedings
4	21	Volume
5	22	Page numbers

Category	2	Research and Creative Works
Sub category	2	Publications
Sub Sub Category	7	Paper presented at a conference

Staff Return Items Composition Rule Report 13/02/2001

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 7 Paper presented at a conference

Column No Prompt No Field Prompt
1 13 Author(s)
10 14 Date
2 15 Title
3 24 Conference
4 17 Venue

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 8 Research reports

Column No Prompt No Field Prompt
1 13 Author(s)
10 14 Date
2 15 Title
3 16 Publisher

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 9 Audio-visual recordings

Column No Prompt No Field Prompt
1 13 Author(s)
10 14 Date
2 15 Title
3 16 Publisher

2 Research and Creative Works

Staff Return Items Composition Rule Report 13/02/2001

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 10 CD-ROM/computer software

Column No	Prompt No	Field Prompt
1	13	Author(s)
10	14	Date
2	15	Title
3	16	Publisher

Category 2 Research and Creative Works
Sub category 2 Publications
Sub Sub Category 11 Creative works

Column No	Prompt No	Field Prompt
1	25	Presenter(s)
2	15	Title
3	26	Event
4	17	Venue

Category 3 Community and Professional service
Sub category 1 TV/radio/media participation

Column No	Prompt No	Field Prompt
1	28	Name
2	29	Programme/article title
3	30	Radio/TV Station or Publication
9	31	Topic
10	14	Date

Category 3 Community and Professional service
Sub category 2 Appointments to Boards or External Committees

Column No	Prompt No	Field Prompt
-----------	-----------	--------------

Staff Return Items Composition Rule Report 13/02/2001

Category	3	Community and Professional service
Sub category	2	Appointments to Boards or External Committees
Column No	Prompt No	Field Prompt
1	32	Position
2	33	Name of Board/Committee
3	34	Nominating group
10	35	Date of tenure

Category	3	Community and Professional service
Sub category	3	Awards/recognition
Column No	Prompt No	Field Prompt
1	4	Recipient(s)
2	36	Achievement/Award
3	5	Awarding Body
9	37	Reason for award

Category	3	Community and Professional service
Sub category	4	Service -(several)
Column No	Prompt No	Field Prompt
1	1	Name(s)
2	2	School
9	3	Description

Category	3	Community and Professional service
Sub category	5	Visitors
Column No	Prompt No	Field Prompt
1	28	Name
2	38	Institution
3	39	School/Centre
9	40	Purpose
4	41	Duration

Staff Return Items Composition Rule Report 13/02/2001

Category 4 International and Commercial activities

Sub category 1 Strategic alliances/partnerships

Column No Prompt No Field Prompt

1	28	Name
2	2	School
3	27	Institution/Organisation
9	3	Description

Category 4 International and Commercial activities

Sub category 2 International visitors

Column No Prompt No Field Prompt

1	28	Name
2	38	Institution
3	2	School
8	40	Purpose
4	41	Duration

Category 4 International and Commercial activities

Sub category 3 International students

Column No Prompt No Field Prompt

1	28	Name
2	2	School
9	3	Description

Category 4 International and Commercial activities

Sub category 4 Commercial activities

Column No Prompt No Field Prompt

1	28	Name
2	2	School
9	3	Description

Category 5 Other significant activities

Staff Return Items Composition Rule Report 13/02/2001

Category 5 Other significant activities

Sub category 1 Other

Column No Prompt No Field Prompt

1 1 Name(s)

9 3 Description

Period Report

This report contains a list of all items submitted via the dynamic data entry screens for a specific collection period. It is configured exclusively by the flexible SQL query detailed in section 4.2.3.

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 001121

01/OCT/2000 - 31/DEC/2000

HUNT, Lynne

Category Teaching and Learning

Sub Category Teaching initiatives

Name(s) Assoc Professor Lynne Hunt
School School of Nursing and Public Health
Description Written text for two web sites:
1. Work-based University Learning:
<http://www.edu.edu.au/ssa/worklinks/>
2. Race Around ECU:
<http://www.ecu.edu.au/pa/raecu/>

Category Teaching and Learning

Sub Category Teaching initiatives

Name(s) Assoc Professor Lynne Hunt
School School of Nursing and Public Health
Description Developed and presented a workshop
for ECU's Professional Development
Unit entitled: University Work-
based Learning: New Ideas and
Strategies

Category Research and Creative Works

Sub Category Publications

Sub Sub Category Book

Author(s) Dr Kaosar Afsana, PhD student in the
School of Nursing and Public Health
and Sabina Faiz Rashid

Date

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 001121

Category

Sub Category

Sub Sub Category

Title

Discoursing Birthing Care:
Experiences from Bangladesh

Publisher

University Press Ltd:

Venue

Dahaka

Category

Research and Creative Works

Sub Category

Publications

Sub Sub Category

Article refereed journal

Author(s)

Berne, L.A., Patton, W., Milton, J.,
Wright, S., Hunt, L., Peppart, J.
and Dodd, J.

Year

Title of article

A qualitative assessment of
Australian parents' perceptions of
sexuality education and
communication

Title of journal

Journal of Sex Education and Therapy

Volume

Page numbers

Category

Community and Professional service

Sub Category

TV/radio/media participation

Name

Assoc Professor Lynne Huhnt and J
Trotman

Programme/article title

Claremont Cameos: Dorothy Hewitt

Period Report for : 01-OCT-2000 - 31-DEC-2000

Staff No 001121

Category

Sub Category

Radio/TV Station or
Publication

ABC Radio

Topic

Social History Unit Broadcast

Date

Category

Community and Professional service

Sub Category

Awards/recognition

Recipient(s)

Assoc Professor Lynne Hunt

Achievement/Award

Sybe Jongeling Prize for Outstanding
Dedication to Research

Awarding Body

ECU Postgraduate and Honours Student
Association

Reason for award

Staff No 002982

01/OCT/2000 - 31/DEC/2000

BITTLES, Alan

Category

Other significant activities

Sub Category

Other

Name(s)

Professor Alan Bittles

Description

Public Lecture on Medical Ethics?
So who let them clone Christopher
Skase? at the Alexander Library on
20 October.

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 004229

01/OCT/2000 - 31/DEC/2000

LAVERY, Paul

Category Research and Creative Works

Sub Category Research Grants

Recipient(s) Dr Paul Lavery and Dr C Oldham

Funding Body Wenner-Gren Foundation

Title of Grant Visiting Scientist Award

Description To support visiting scientists to Sweden

Amount granted 5500

Collaborating Organisations

Category Research and Creative Works

Sub Category Research Grants

Recipient(s) Dr Paul Lavery

Funding Body Water and Rivers Commission

Title of Grant Assessment of the Environmental Impacts of Algal Harvesting

Description

Amount granted 2000

Collaborating Organisations

Category Research and Creative Works

Sub Category Research Grants

Recipient(s) Dr Paul Lavery

Funding Body DEP

Title of Grant Assessment of Seagrass Health in the

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 004229

Category

Sub Category

Perth Metropolitan Waters

Description

Amount granted 11000

Collaborating Organisations

Category

Research and Creative Works

Sub Category

Research Grants

Recipient(s)

Dr Paul Lavery

Funding Body

Cockburn Cement Ltd

Title of Grant

The Ecological significance of
seagrass ecosystems

Description

Amount granted 2035

Collaborating Organisations

Category

Research and Creative Works

Sub Category

Research Grants

Recipient(s)

Dr Paul Lavery

Funding Body

ARC Small Grant and ECU

Title of Grant

An alternative method to trace
sewage pollution in well mixed
coastal waters

Description

Period Report for : 01-OCT-2000 - 31-DEC-2000

Staff No 004229

Category

Sub Category

Amount granted 10000

Collaborating Organisations

Category Research and Creative Works

Sub Category Research Grants

Recipient(s) Dr Paul Lavery and Dr Glenn Hyndes

Funding Body ARC Small Grant

Title of Grant The role of transported macrophyte material for fish production in unvegetated marine habitats

Description

Amount granted 9600

Collaborating Organisations

Category Research and Creative Works

Sub Category Publications

Sub Sub Category Article refereed journal

Author(s) Vanderklift, M.A. and Lavery, P.S.

Year

Title of article Patchiness in assemblages of epiphytic macroalgae on Posidonia coriacea at a hierarchy of spatial scales

Title of journal Marine Ecology Progress Series

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 004229

Category

Sub Category

Sub Sub Category

Volume 192

Page numbers 127-135

Category Research and Creative Works

Sub Category Publications

Sub Sub Category Article refereed journal

Author(s) Wood, N. and Lavery, P.

Year

Title of article Monitoring seagrass ecosystem health
- the role of perception in defining
health and indicators

Title of journal Ecosystem Health

Volume 6

Page numbers 134-148

Category Research and Creative Works

Sub Category Publications

Sub Sub Category Article refereed journal

Author(s) Lavery, P. and Vanderklift, M.A.

Year

Title of article Comparison of spatial patterns in
seagrass epiphyte assemblages using
species and functional group-level
data

Title of journal Soc. Ital. Di Biol. Mar.

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 004229

Category

Sub Category

Sub Sub Category

Volume

7

Page numbers

251-254

Category

Research and Creative Works

Sub Category

Publications

Sub Sub Category

Article refereed journal

Author(s)

Vanderkluft, M.A. and Lavery, P.

Year

Title of article

Small-scale spatial patterns in
epiphyte assemblages of *Posidonia*
coriacea and *Amphibolis griffithii*
Soc. Ital. Di Biol. Mar.

Title of journal

Volume

7

Page numbers

294-297

Category

Research and Creative Works

Sub Category

Publications

Sub Sub Category

Article in refereed conference proceedings

Author(s)

Dr Paul Lavery

Date

Title of article

Comparison of spatial patterns in
seagrass epiphyte assemblages using
species and functional group data

Title of proceedings

Proceedings of the 4th International
Seagrass Biology Workshop

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 004229

Category

Sub Category

Sub Sub Category

Page numbers

Category

Research and Creative Works

Sub Category

Publications

Sub Sub Category

Article in refereed conference proceedings

Author(s)

Dr Paul Lavery

Date

Title of article

Differences in spatial patterns in assemblages of epiphytic macroalgae between seagrass hosts

Title of proceedings

Proceedings of the 4th International Seagrass Biology Workshop

Page numbers

Category

Research and Creative Works

Sub Category

Publications

Sub Sub Category

Article in refereed conference proceedings

Author(s)

Dr Paul Lavery

Date

Title of article

Can d15N of different macroalgae be used to map temporal patterns in sewage pollution

Title of proceedings

Proceedings of the Swedish Society for Marine Research Biennial Conference

Page numbers

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 004229

Category Research and Creative Works
Sub Category Publications
Sub Sub Category Article in refereed conference proceedings
Author(s) Dr Paul Lavery
Date
Title of article Monitoring Seagrass Ecosystem
Health. The role of perception.
Title of proceedings Australian Marine Sciences
Association (WA) Conference
Page numbers
Staff No 007657 01/OCT/2000 - 31/DEC/2000

BLADES, Andrew

Category Research and Creative Works
Sub Category Publications
Sub Sub Category Book chapter
Author(s) Blades, A.J.
Date
Title of chapter The 4 Phases of Risk Realisation
Title in Doughty, K (Ed) Business
Continuity Planning: Protecting Your
Organisation's Life
Publisher Auerbach:New York
Venue

Category Research and Creative Works
Sub Category Publications
Sub Sub Category Book chapter

Period Report for : 01-OCT-2000 - 31-DEC-2000

Staff No 007657

Category

Sub Category

Sub Sub Category

Author(s)

Blades, A.J.

Date

Title of chapter

Learning from a Crisis

Title

in Doughty, K (Ed) Business
Continuity Planning: Protecting
Your Organisation's Life

Publisher

Auerbach: New York

Venue

Staff No 007744

01/OCT/2000 - 31/DEC/2000

LESLIE, Gavin

Category

Research and Creative Works

Sub Category

Publications

Sub Sub Category

Paper presented at a conference

Author(s)

Assoc Professor Gavin Leslie

Date

Title

Roster Satisfaction in the intensive
care unit (ICU) - a review of the
introduction of 12 hour shifts

Conference

25th Australian and New Zealand
Scientific Meeting on Intensive Care

Venue

Canberra

Category

Research and Creative Works

Sub Category

Publications

Period Report for : 01-OCT-2000 - 31-DEC-2000

Staff No 007744

Category
Sub Category
Sub Sub Category Paper presented at a conference
Author(s) Assoc Professor Gavin Leslie
Date
Title Twelve hour rostering in critical care - lessons from the emergency department and intensive care unit
Conference 6th Nursing Practice Conference:
Nursing: Charting a new course
Venue Adelaide

Category Community and Professional service
Sub Category Awards/recognition

Recipient(s) Assoc Professor Gavin Leslie
Achievement/Award Editor publications
Awarding Body Australian College of Critical Care Nurses (ACCCN)
Reason for award Responsibilities include Australian Critical Care (refereed journal), Critical Times (quarterly national newspaper) and website (www.ACCCN.com.au)

Staff No 200005

01/OCT/2000 - 31/DEC/2000

GROOM, Philip

Category Research and Creative Works
Sub Category Publications

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 200005

Category

Sub Category

Sub Sub Category Article refereed journal

Author(s) Groom, P.K., Froend, R.H., Mattiske, M. and Koch, B.L.

Year

Title of article Myrtaceous shrub species respond to long-term groundwater levels on the Gngangara Groundwater Mound, Northern Swan Coastal Plain

Title of journal Journal of the Royal Society of Western Australia

Volume 83

Page numbers 75-82

Staff No 205484

01/OCT/2000 - 31/DEC/2000

BOUSSAID, Farid

Category Research and Creative Works

Sub Category Research Grants

Recipient(s) Dr Farid Boussaid

Funding Body FCHS

Title of Grant Seed Grant

Description Toward Advanced CMOS Imaging Technology

Amount granted 5092

Collaborating Organisations

Category Research and Creative Works

Period Report for :01-OCT-2000 - 31-DEC-2000

Staff No 205484

Category

Sub Category

Publications

Sub Sub Category

Paper presented at a conference

Author(s)

Dr Amine Bermak, Dr Farid Boussaid
and Assoc Professor Salim Bouzerdoun

Date

Title

A digitally programmable current
mode analog shunting inhibition
cellular neural network

Conference

7th IEEE International Conference on
Electronics, Circuits and Systems

Venue

Kaslik, Lebanon

Staff Without Submissions

This report provides a listing of all staff members who have not submitted a return for a given collection period.

STAFF MEMBERS WITHOUT RETURNS FOR:

01-OCT-2000 - 31-DEC-2000

Staff No	Surname	First Names	Telephone No	Email
057656	AHERN	Kathy	8611	
006359	ALDER	Jackie	5459	
058181	ALLISON	Denise	6156	
044097	ANDERSON	Karen	6296	
006889	ANDREWS	Julie	6508	
009631	ANGELICHEVA	Dora	5714	
008411	ARMSTRONG	Colin	6030	
203306	ARMSTRONG	Helen	6856	
048039	BAKER	Eileen	5539	
016109	BALLANTINE	Kevin	6213	
009012	BANASIEWICZ	Nathan	6115	
043431	BANNISTER	Mark	6336	
002367	BARNES	Jeff	5424	
007172	BARTON	Lynn	5289	
204978	BAUMANIS	Andrew	6383	
005307	BELL	Catherine	5482	
000207	BENNETT	Ian	6350	
203024	BERMAK	Amine	5877	
203563	BHATTARAI	Nirja	6638	
201221	BIGLARI-ABHARI	Morteza	5785	
021848	BLOOM	Lyn	5883	
201877	BOLAN	Christopher	5581	
200791	BORDAS	Nardia	8585	
201400	BOUZERDOUM	Abdesselam	5059	
001216	BOYCE	Mary	6328	
050737	BRIGHTWELL	Richard	8564	
020853	BROCK	Lorna	8562	
203048	BRODALKA	Joseph	6353	
046623	BROGAN	Mark	6300	
204983	BURNETT	Angus	5860	
040045	BURT	Lorraine	8612	
054973	BYRNE	David	8591	
203621	BYRNE	Eoin	6699	
202133	CADMAN	Robert	5876	
204645	CHAI	Douglas	5874	
003954	CHANDLER	David	5716	
030824	CHIRATHAMJAREE	Chai	6356	
044695	CHOW	Shirley	8574	
016563	CLAYDEN	Judy	6298	
098124	COLEMAN	Marion	8566	
002123	COLLINS	Michael	6363	
045866	COLLINS	Simon	6335	
016635	COMBER	Geoff	6361	
094094	CORNELIUS	Mary	5553	
205149	CRAMER	Jennifer	8623	
036089	CRAWFORD	Anne	8024	

STAFF MEMBERS WITHOUT RETURNS FOR:

01-OCT-2000 - 31-DEC-2000

Staff No	Surname	First Names	Telephone No	Email
004954	CRAWFORD	Ian	6334	
053807	CROSS	Bob	6374	
016723-1	CROSS	Jim	5881	
016723-2	CROSS	Jim	5881	
200748	CROSTHWAITE	Marilyn	5692	
054050	DANAHER	Maurice	6541	
999998	DAVIS	Paul		
108002	DEVLIN	Anna	6052	
092662	DHALIWAL	Harbhajan	5019	
099522	DIXON	Carol	6569	
006102	DOWNES	Karen	5467	
090667	DOWNIE	Margaret	8183	
005038	DOYLE	John	5031	
005799	DOYLE	Steve	6054	
003180	DROUET	Elizabeth	5448	
050366	DRURY	John	8618	
026913	DUFF	John (Edward)	6231	
019801	ELAM	Anne	5505	
016918	EMBREY	Lynn	5655	
004934	ESHRAGHIAN	Kamran	5839	
025401	FRENCH	Sandie	6299	
092179	FREW	Katherine	5583	
007241	FROEND	Ray	5563	
048207	GALBRAITH	Alan	8563	
017030	GAMBLE	Ross	5450	
017056	GARNETT	Patrick	5665	
201001	GARNETT-LAW	Bryan	6115	
008959	GIBLETT	Rodney	6051	
003051	GIBSON	Barry	5037	
200371	GIBSON	Marlene	8192	
000889	GODFREY	Paul	6713	
005399	GOODE	Elizabeth	6351	
076486	GOSLING	Joanne	8581	
006993	GRAY	Jason	6460	
044652	GRAY	Lorraine	8605	
048231	GREEN	Lelia	6204	
091555	GURURAJAN	Raj	6017	
003477	HABIBI	Daryoush	5787	
046113	HALL	Jean	6427	
200745	HANNAN	Christine	8561	
002201	HARRIS-WALKER	Jody	5557	
055108	HAUCK	Yvonne	8570	
200720	HERLIHY	Bianca	6719	
006373	HERRINGTON	Jan	6190	
001962	HINCKLEY	Stephen	5710	
200785	HOGAN	Vanessa		

STAFF MEMBERS WITHOUT RETURNS FOR:

01-OCT-2000 - 31-DEC-2000

Staff No	Surname	First Names	Telephone No	Email
006367	HOPE	Peter	5653	
000429	HORWITZ :	Pierre	5558	
204794	HYNDES	Glenn	5798	
022779	JACKSON	Glenda	5451	
007455	JAFFAR	Taib	6330	
200168	JAMIESON	Sharon	8534	
003190	JAUNZEMS	Linda	5847	
090544	JENNINGS	Kaye	5847	
205245	JIVOTOVSKY	Lev	5467	
203335	JOHN	Maria (Liz)	5121	
017435	JOHNSON	Angela	5651	
009275	JOHNSON	Julie	6570	
205064	JOLLEY	David (William)	6877	
040512	JONES	Bronwyn	8598	
009873	JONES	Sue	6333	
002094	JOSEPHI	Beate	6691	
003903	KALAYDJIEVA	Luba	5456	
002122	KARPATHAKIS	George	6321	
027713	KINNEAR	Adrienne	6499	
204375	KONGRAS	Tiffany	6353	
007066	KOTHAPALLI	Ganesh	5792	
200034-1	KRISTJANSON	Linda	8617	
200034-2	KRISTJANSON	Linda	8617	
035406	KUCZBORSKI	Wojciech	6013	
001757	LACHOWICZ	Stefan	5580	
001368	LAIDMAN	William	6514	
002034	LANCE	Hugh	5556	
044724	LANGRIDGE	Miriam	8558	
033443	LEDWITH	Colleen	5884	
201055	LEE	Julie	5448	
074940	LEGGETT	Monica	6476	
099514	LEHMANN	Pauline	8585	
202016	LESLIE	Mark	6507	
017582	LESLIE	Norman	6214	
006454	LI	Dongguang	6358	
017603	LINSTEN	Joram	5578	
005921	LOURENS	Geoff	6367	
093964	LOY	Poh-Kin	5831	
006428	LUCA	Joe	6412	
004363	LUFF	Jonathon	5557	
001756	LUND	Mark	5644	
034745-1	LUU	Kim	6101	
034745-2	LUU	Kim	6101	
092291	MACKIE	Doreen	5661	
001854	MAJ	S Paul	6277	
204795	MANN	Graham	6863	

STAFF MEMBERS WITHOUT RETURNS FOR:

01-OCT-2000 - 31-DEC-2000

Staff No	Surname	First Names	Telephone No	Email
075416	MARRIOTT	Rhonda	8610	
002586	MATA.	Gina	8589	
005456	MAYRHOFER	Debra	6014	
204522	MCCRUM	Janet	8581	
017742	MCDOUGALL	David	5439	
007348	MCGLUE	William (Bill)	6230	
006095	MCKEE	Alan	6859	
201456	MCLEOD	Nicole	5852	
007281	MCPAHON	Mark	6434	
057710	MCPHEE	Irene	8569	
017849	MEREDITH	Chris	5562	
201264	METTAM	Brad	5557	
036433	MILLAR	Clay	6525	
037188	MILLAR	Jim	6547	
006104	MILLER	Russell	6555	
001694	MONDELLO	Helen	6455	
204590	MONTEROSSO	Leanne	8621	
203951	MORRIS	Fiona	5012	
004710	MUELLER	Ute	5272	
006264	MUSSETT	Janis	5590	
002600	NEDVED	Milos	5672	
050307	NEEDHAM	Alan	6667	
046228	NEWMAN	Coral	5880	
044636	NEWNHAM	Helen	8613	
006306	NG	Christine	8624	
073605	NIKOLETTI	Suzanne	8182	
001614	NOBLE	Kay	5612	
088241	O'NEILL	Tom	6431	
001340	O'SHAUGHNESSY	Michael	6212	
200790	O'SHEA	Mairead	5189	
203047	OLAKA	Francis	5782	
037233	OLIVER	Ron	6372	
003340	OMARI	Arshad	6459	
053401	PAM	Maxwell	6218	
003494	PATAK	Annette	6658	
005815	PATAK	Paul	6647	
006355	PEARSON	Deborah	6214	
001902	PEDLER	Pender	5082	
205014	PERKINS	Timothy	5459	
040740	PERRY	Shirley	8437	
050585	PHILLIPS	Megan	8584	
014091	PHILLIPS	Vincent	6650	
001953	PIKE	Graham	5625	
022630	PLATEL	Karl	6217	
009002	POLAND	John	6016	
088807	POULLAY	Sam	8620	

STAFF MEMBERS WITHOUT RETURNS FOR:

01-OCT-2000 - 31-DEC-2000

Staff No	Surname	First Names	Telephone No	Email
200925	POWER	Marion	5507	
201853	PUMPHREY	Melissa		
094916	PURCELL	Magdalen	5514	
037962	QUIN	Robyn	6221	
021426	QUINN	Delia	6345	
096330	RAPSEY	John	6518	
007166	RECHER	Harry	5758	
003985	REDMOND	Janice	5655	
002101	RING	Geoffrey	6369	
009347-1	RING	Jan	6362	
0093472	RING	Jan	6362	
008575	RIVETT	Donelle	5476	
095169	ROBBINS	Graeme	6872	
044716	ROBERTS	David	8609	
068638	ROBERTS	Peter	5455	
044927	ROCHE	Valerie	5025	
098044	RODGER	Martin	8554	
029276	ROSE	Elizabeth	6803	
003429	RUMMEY	Jackie	6325	
055685	RYDER	David	5452	
006945	SACCO	Paul	5642	
104731	SADIQUE	Deborah	6684	
001434	SALMON	Alison	5466	
050711	SELLAPPAH	Su	8578	
085981	SERRELL	Maxine	8573	
009010	SHANLEY	Eamon	8631	
202032	SHI	Beilin	5443	
018622	SHOESMITH	Brian	6219	
002216	SINCLAIR	Kelvin	6542	
000645	SKINNER	Chris	5453	
007135	SMITH	Barbara	8533	
018657	SMITH	Clifton		
005411	SMITH	Ingrid	5221	
005466	SMITH	Keith	6516	
007347	SMITH	Kevin	5846	
023448	SNADER	Sharron	6220	
050745	SPICKETT	Evadne	6347	
087960	STEVENSON	Anne	8593	
054236	STEWART	Angus	5697	
003485	SWAN	Geoff	6425	
204224	TAN	Dennis	6680	
002962-1	TERRY	Julian	5734	
002962-2	TERRY	Julian	5734	
002807	THOMSON	Neil	5053	
007094	TONKIN	Colleen	8597	
205155	TUBBS	Mileva	5848	

STAFF MEMBERS WITHOUT RETURNS FOR:

01-OCT-2000 - 31-DEC-2000

Staff No	Surname	First Names	Telephone No	Email
201263	TURNER	Eric	5477	
009161	VAN ETTEN	Eddie	5566	
002015	WAGNER	Gulten	6072	
009100	WANG	Wei	5714	
200272-1	WARNOCK	Kathryn	8595	
200272-2	WARNOCK	Kathryn	8595	
002488	WATSON	Anthony	6470	
002496	WHITE	Douglas	5564	
000171	WILSON	Vicky	6301	
003708	WOOD	Dennis	6107	
042294	WOODROFF	Susan	8583	
204636	WORSLEY	Penelope	5716	
205160	YANG	Danian		
002129	YEO	Malcolm	6577	
050534	YIP	Vincent	8576	
999997	YU	Zhi Huan	5034	
200867	ZHAO	Xiaoli	5782	

Total: 247

Staff Membership Report

This report details the discipline areas and research groups to which an individual staff member belongs.

Staff Membership Report

Staff No001340

Surname O'SHAUGHNESSY

First Names Michael

Discipline Areas:

Film and Video

Media Studies

Research Groups :

Group Name

BIOMEDICAL AND SPORTS SCIENCE

Start Date Term Date

07-NOV-00