ERP Financial Data Archiving System

Francesco Ferretti (s020940)

Master Thesis

July, 2004

Supervised by Jens Thyge Kristensen

Department of Computer Science and Engineering
Institute of Informatics and Mathematical Modelling

- Technical University of Denmark -

IMM-THESIS-2004-40

Preface

This thesis is authored by Francesco Ferretti and is the resulting work of the thesis project carried out by the author at the Technical University of Denmark. The project, titled *"ERP Financial Data Archiving System"*, has been carried out for a telecommunication company, during the period between January 19th, 2004 and July 16th, 2004.
The thesis has been completed under the supervision of Associate Professor Jens Thyge Kristensen, within the department of Informatics and Mathematical Modelling.
The project has been accomplished according to the requirements for the attainment of a Master of Science in Engineering at Technical University of Denmark.

Kgs. Lyngby, July 16th, 2004

Francesco Ferretti, s020940

Acknowledgements

I wish to acknowledge the special supervision of Associate Professor Jens Thyge Kristensen and thank him for the dedication and time he took to offer me his precious advice and to stimulate many rewarding discussions.

My sincere gratitude to Mr. Simon Broadhurst, Oraplus' president, for his professional supervision and for giving me the opportunity of carrying out this project on one of his clients.

I also thank all my colleagues at Oraplus and at the client site for their support and helpful advice.

Finally, I am forever indebted to my parents and Valeria for their total confidence in me and the exceptional understanding, patience and encouragement during the entire period it took me to research and write this thesis.

In loving memory of my mother

Abstract

The thesis project described in this document, focuses on a problem faced by a telecommunication company[1].
The problem was connected to the use of "Enterprise Resource Planning" (ERP) systems, which many large organizations have now adopted. These ERP systems are application platforms that assist companies in running their business by covering areas such as finance, payroll, procurement, and so on. In many cases such systems are based on a so-called *"global single instance"* (GSI), a centralised architecture at world level that implies collecting data coming from each branch within the group. In large organizations, gathering all these data for a long period of time in a single repository has recently started affecting the overall performance of their ERP system. In the company considered here, in fact, system response times, manageability and efficiency were starting to suffer and to impact most of the day-to-day business activities, hence, the need to find a solution to this problem that is starting to concern increasingly more companies.

Unfortunately, backing-up all historical data and then removing them from the production environment is not a viable solution for several reasons. Firstly, data warehousing activities are carried out on such data, and this, entails data to be readily accessible and stored with their logical organization and relationships. Secondly, management normally inspects historical data for verification and auditing purposes. Such inspection activities that are often carried out on a non-regular basis require data to be (again) readily accessible and typically from well-known user interfaces. Furthermore, fiscal authorities require organisations to reproduce fiscal reports upon request. This implies that for a number of years historical data must be kept in the same original format and available in those system locations from which all programs (used for generating fiscal reports) pick up source data. Acquiring a new production server or replacing some of its components with newer and higher performance components is also not viable, since nowadays the budgets of IT departments are increasingly tighter and generally do not

[1] For confidentially reasons, the company name is withheld and referred to as *"The Company"* in the sequel.

allow huge expenses. Consequently, for these and other related reasons, a different approach had to be taken.

The solution found consists of a *"financial archiving system"* that enables all the historical data to be removed from the production environment, making it possible to allocate more resources to non-historical data. All data removed is at the same time fully accessible thanks to the implementation of a kind of *"data distribution"* policy that allows a separate distribution of logical and physical data in the system. Historical data is in fact "spread" on a number of hard disks residing on machines other than the one for production. Such disks and machines correspond, for example, to components that for several reasons (e.g. inadequate processing power, speed or storage capacity, etc.) are no longer used for production environments and are left unused or dedicated to less demanding purposes.

Technically, the financial archiving system entails the creation of an "archive" database schema into the ERP database. Such database schema contains the same tables and related objects (initially empty) as the standard database schema in which both historical and non-historical data are normally stored. An *archiving program* integrated in the ERP system, then takes care of selecting, from the standard schema, all eligible data for archiving, copying them into the archive schema and finally deleting them from the original location (i.e. the standard schema).

All this is achieved without compromising the ERP producer support (potentially affected by customised non-authorised deletion operations).

Moreover, historical data is made available through the same ERP user interface thus allowing users to view their data in the usual way (through the same forms) and making it possible to apply all ERP functions to historical data too. This in turn, enables carrying out all needed data inspections, all data warehousing activities and reproducing fiscal reports upon the authorities' request.

Contents

Chapter 1

Introduction

During the last decade, business economy has been marked by increasingly profound and rapid changes. Many companies have developed and expanded at an impressive rate throughout the world. This is part of a system called (not always with positive connotations) the "global economy".
Besides touching important issues regarding the social aspects of such tendency, this rapid growth has significantly affected the way companies run their business and, behind the scenes, has remarkably changed their internal organization.

One of the most evident changes is the increased need of information inside the organizations. Nowadays, large enterprises that want to be successful, must necessarily make the most out of their business information flows.
Such changes have led to a technology-oriented business trend characterized by the use of *"Enterprise Resource Planning"* (ERP) systems.
ERP systems are a sort of application platform integrating all facets of the business and covering, for example, areas such as finance, logistics, manufacturing, marketing, human resources and so on.
The way the information coming from different areas are combined together enables the organization to improve many of its activities. Typically, this can be seen in activities such as corporate planning, forecasting and scheduling.

In other words, ERP systems make it possible to exploit business information flows, assisting the organization to run their business more effectively and efficiently.

1.1 Motivation

In some cases ERP systems are based on a *"global single instance"* (GSI) at world level. This is especially true for large enterprises with branches in many countries throughout the globe. In such cases, the management is provided with a great amount of information that in turn enables it to have the highest achievable perspective regarding the organization, and thus to further improve effectiveness and efficiency.

Adopting such single application platform throughout the whole organization has proved very successful. Nevertheless, such approach has recently posed some problems for large worldwide organizations dealing with vast amounts of data.
In such context, the ERP system in use in fact deals with data coming from each branch within the group. In most cases, this entails a considerable amount of data that cannot be deleted because it must be stored for a certain number of years in order to comply with the legal requirements of various tax authorities (i.e. the authorities in the countries the data refers to) before it can be deleted. Moreover, data warehousing activities – which is one of the fundamental aspects of ERP systems – make significant use of historical data and this, too, entails data retention. For these and other related reasons, the amount of data kept increases quickly and, most of the time, unsustainably.

Vast amounts of data poses problems with respect to storage equipment. When dealing with storage devices that are already considerably advanced and capable, further enhancements usually require heavy investments, i.e. after a certain level, the economic effort is extremely heavier (it is no longer increasing linearly). Therefore the need for better equipment can easily be eliminated from budget plans (typically connected to the estimated life of the current ERP system).
The problem however is not just related to storage capacity. There are crucial problems related to latency and response times which affect almost every activity in the system. In fact, even if more capable data storage equipment can be obtained, system enquires will involve data scansions that can only be marginally improved since they directly depend on the amount of data to be scanned and on the processing power. Most likely, this would require changing not

just the CPUs but the whole server(s) too. As the case above, after a certain level, the speed-up factors required may not be achievable within the IT department budgets.
Lastly, but likewise important, is the fact that keeping the underlying central data repository smaller prevents critical circumstances from occurring and also considerably facilitates all the ordinary administration work.

As mentioned above, only big enterprises dealing with large volumes of data have recently started to deal with such issues.
Such an enterprise is the one considered in this document, a telecommunication company operating across most of the European countries and in a number of other countries in the rest of the world.

This company has recently adopted an ERP system based on a global single instance and is already starting to deal with the aforementioned problem. The amount of data stored is getting to concerning levels and the ERP response times are thereby suffering.

Consequently, The Company and Oraplus, one of its IT services provider, started a project regarding this topic and the author was offered to work on it and to propose a solution that could help in keeping all the relevant historical data available while improving the performance of the ERP system.

1.2 Objectives

The project calls for devising and putting in place a comprehensive solution for archiving and purging financial data across all business units in The Company group, as further specified in the subsequent chapters. Specifically, this implies developing and implementing a ***financial archiving system*** in order to minimize the aforementioned negative effects posed by the use of the ERP system and connected with the huge amount of data handled. Such system should be integrated into the ERP system in use, permitting reasonable response times and carrying out all the needed data warehousing activities while complying with country-specific legal requirements.

1.3 The Company and the author

Francesco Ferretti, the author of this document, had the opportunity of working for The Company and Oraplus for 4 months during 2003. He was involved in implementing the ERP system currently in use in The Company, and in resettling several European branches of the group.
After the work experience, the author was informed of the problem explained beforehand and was offered to take on the project for the *financial archiving system*.

1.4 Document organization

This document, therefore describes the solution proposed to The Company in order to solve the aforementioned problem. The document has been structured in such a way as to highlight the key stages of the project. More precisely, the document has been organised as follows:

Chapters 1, 2 and 3	This part of the thesis provides introductory and background information aimed at describing the project scenario and the application domain.
Chapters 4 and 5	These chapters tackle aspects related to the adopted software process models and requirements engineering process. When relevant, the approach used in this part tends to consider in broader terms the background knowledge that had to be acquired and then reconsiders such knowledge in close relation to the project.
Chapters 6, 7 and 8	These chapters cover the design and implementation stages of the project along with integration aspects. The approach used here, tends to highlight all the analysis and underlying reasoning that led to the designed solution.

Chapters 9	This part focuses on the testing stage carried out on the system devised.
Conclusions	This ending chapter contains all final considerations summarising key characteristics of the solution devised and further developments.
Appendices	In these sections is enclosed all relevant material, such as analysis results, testing sessions outputs, source code, and so on.

Chapter 2

Glossary

This chapter provides the definitions of the terms and acronyms used herein that may be unknown and that need to be understood in order to fully comprehend the work.

Accounting calendar
The calendar that defines all the accounting periods and fiscal years to which a business transaction can be related. See Table 5.2 and section 5.4 for further details.

Accounting period
A period to which a business transaction relates. It is part of the accounting calendar and defines the fiscal period (e.g. fiscal month) used to report financial results.

Archiving program
One of the two components of the financial archiving system. It consists of a program that implements the business logic and carries out the archiving. See chapter 7 for a more detailed description.

Archive schema
The other main component of the financial archiving system. It reproduces the structure of the database objects needed to store the data eligible for archiving (also referred to as "parallel schema"). See chapter 7 for more a detailed description.

Closed transaction

A transaction that represents a logically completed “chain” of accounting documents. None of the blocks of the chain is in a “pending” status. Once a transaction is closed, no further changes can be carried out. Also see: *Transaction*.

Data file

A physical operating system file created by Oracle and located on a disk. It contains data structures such as tables and indexes (Cyran, 2002).

Database object

A logical entity defined and stored in a database. Examples of database objects are: tables, views, indexes, synonyms and stored procedures.

ERP

ERP stands for *Enterprise Resource Planning* and is a management system typically used by large organizations. ERPs consist of a multi-module platform that enable integrating all the facets of the business in order to best exploit all business information flows.

Financial archiving system

It is the system devised for archiving ERP financial data as described in the present document.

Fiscal period

The fiscal period is the units into which a fiscal year is divided (e.g. fiscal month). It used to report financial results to governments and legal authorities.

Fiscal year

It is the interval of the same duration as the calendar year which is used by a company for its accounting. It may or may not coincide with the calendar year. A company may consider it convenient to end its accounting year at a time when, for example, inventory stocks are down. As an example, in several countries the fiscal year runs from 1 April to 31 March.

Form

It is a group of related fields and graphical components that appears on a single screen. In Oracle E-Business Suite R11i, forms can be thought of as paper "documents" used to run a business. Data are thus entered by typing information into the form.

Functional currency

The currency used in business transactions recorded for a given set of books (e.g. Euro). Also see: *Set of books*.

Historical data

All financial data related to a certain number of past fiscal years. The specific number of years after which financial data is considered as "historical", is variable and depends on users needs.

Online data

All financial data related to the current fiscal year and to a small number of past fiscal years (typically one or two). Online data forms the set of data that is normally used during the day-to-day business activities.

Open transaction

An open transaction is a logically uncompleted "chain" of accounting documents. An open transaction has at least one of the underlying accounting documents in "pending" status. For example, an issued invoice whose payment has not yet been received. Also see: *Transaction*.

PL/SQL

A procedural extension of Oracle SQL that provides programming constructs.

Responsibility

It is a collection of functions within the Oracle E-Business Suite R11i needed to grant access to users within specific business areas. Each user is assigned one or more responsibilities depending on his role in the organization. Thus, for example, there are responsibilities for accountants (e.g. General Ledger user), for accountants managers (e.g. General Ledger Super-user), for buyers (e.g. Purchasing manager), and so on.

Schema
A collection of database objects, including logical structures such as tables, views, sequences, synonyms, indexes, stored procedures, clusters, and database links. Its name corresponds to the name of the user who controls it (Cyran, 2002).

Set of books
Every company is legally obliged to keep a systematic record of its business transactions. This must be done in order to comply with financial reporting requirements imposed by legal authorities. In the past, this was done by writing down information related to business transactions on several paper registers. Thus, each company had a set of registers (also called "set of books") for such purpose.
In order to record the business transactions, each set of books must specify a *chart of accounts*, a *functional currency* and an *accounting calendar* (see glossary terms for further details). These three elements are the mandatory elements for defining a valid set of books.
Nowadays, the accounting in large companies is carried out by specific software that supports in performing all business-related activities and especially the recording of business transactions. In such software applications, the above mentioned set of books is implemented with database tables and other related objects. Its definition is typically contained in a database table that stores the three above mentioned elements (in addition to other "profile" information).
Thus, from a legal point of view, a set of books is a financial reporting entity that uses a particular chart of accounts, functional currency and accounting calendar. Typically, a set of books is defined for each business location (although, in same cases, more sets of books can be defined for each business location, thereby creating sub-business locations).

SQL
SQL stands for Structured Query Language and is a standard language used to access data in relational databases.

Standing data
Standing data is one of the two classes of data stored in financial databases (the other class is "transaction data"). Standing data refers to financial *entities* such as customers, suppliers,

employees, and so on. Such data provides information on entities defined in the data model. It does not provide information on events occurring among entities (such as those events related to issuing an invoice, a purchase order, etc.). The characteristic of standing data is that it does not change greatly over time. Also see: *Transaction data*.

Synonym

A synonym is an alias for database objects (such as tables, views, sequences, or program units) that masks the real name and owner of the object. It can be used to provide public access to objects, and to simplify SQL statements.

Table

A table is the primary storage unit in a relational database. A table expresses entities and relationships, and consists of one or more units of information (rows), each of which in turn, consists of the same kind of values (columns).

Tablespace

A tablespace is a database storage unit that groups related logical structures together (Cyran, 2002).

Transaction data

In an ERP system a transaction is a logical "chain" of related information stored in a database that resembles a financial *event* occurring among a number of entities. For example, an invoice issued to a customer or a purchase order matched with an invoice are referred to as *"transaction data"*. In other terms, transaction data refers to financial operations (unlike "standing data", the other main class of financial data that represents financial entities). The characteristic of transaction data is that it generates a number of records in the database that change considerably over time. Also see: *Standing data*.

View

A view is a presentation of the data in one or more given tables according to specific informational needs.

Chapter 3

Background

This chapter provides an overview of the ERP system in use by The Company and for which the financial archiving system is developed.

The ERP system in use by The Company is provided by *Oracle Corporation*, a well known player in the ERP industry[2] and producer of one of the most popular database engines. Precisely, The Company adopts Oracle's *E-Business Suite R11i*, an ERP system that supports a great number of management activities for large enterprises.

The next two sections describe the functional and technical aspects of such ERP system.

3.1 ERP Functional Architecture

Oracle E-Business Suite R11i is a multi-module platform that serves areas such as human resources, procurement, sales, manufacturing, marketing, and so on. One of the core strengths of this applications is that they support *global business*. For example, they provide multi-national support for languages and character sets. They support and allow accounting operations to be carried out in many different currencies. They also enable inter-company accounting, i.e. accounting of all companies belonging to the group[3].

[2] Besides Oracle, the other major ERP suppliers are PeopleSoft, SAP, BAAN and JD Edwards, also collectively known as "JBOPS".

[3] Such feature is often highly desired by large enterprises because it allows a better control over the branches in the group.

In this context, attention is focused on the financial modules, i.e. modules that include the following:

General Ledger	This is the core module for carrying out accounting operations. It is a sort of *collection point* for all financial transactions. Within this module, all statutory financial reports are produced.
Receivables	This is an *accounts receivable* system that enables customer management and all related transactions (e.g. invoices, payments, etc.).
Purchasing	This supports the activities related to the supply chain cycle. It handles all acquisition activity within the company. For instance, it enables restocking inventories, and satisfying customer demand.
Payables	This takes care of all activities related to payments the company sustains when purchasing materials, services and so on.
Inventory	This supports company inventory management and all related activity (e.g. item cataloguing, on-hand quantities). For instance, when an item is purchased (or made), Inventory increases on-hand balance for the corresponding item.
Projects	This is a module that assists in monitoring projects. It enables estimating and monitoring project costs against the budget, to identify the project margin of earnings.

These modules can be considered as components interrelated with one another. Figure 3.1 gives an idea of such relationship. As can be seen, General Ledger module is not located like the other modules. This is to highlight the fact that General Ledger collects all the financial transactions coming from all the other modules (also referred to as “subledgers”).

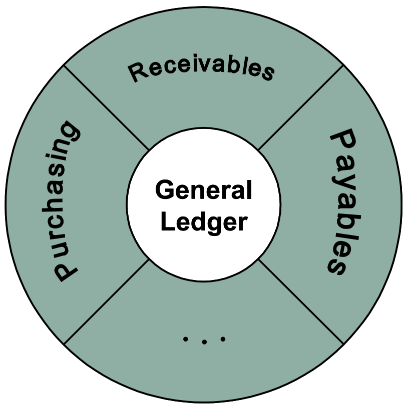
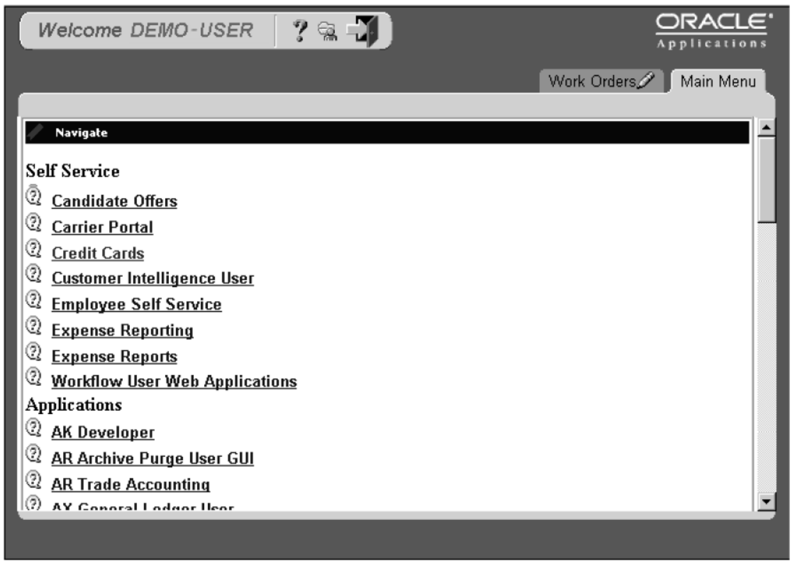


Figure 3.1 Oracle ERP modules organization

Responsibility-based access

Another aspect worth noting is related to the *“responsibility”* mechanism used to provide users access to the ERP system. Oracle E-Business Suite R11i has been designed in order to provide each user *custom-tailored* access. This is achieved by defining a number of user “profiles” each of which describes the enabled and disabled ERP functions. Such user profiles, called “responsibilities”, can be considered as employee roles (e.g. Accountant, accountant manager, accountant supervisor, and so on). This way, a user first logs in to the ERP system by connecting to a specific web page and authenticating with the proper user name and password,. He/she then has to choose one of his/her available responsibilities. Figure 3.2 shows an example of the responsibilities available to a “demo” user (Studdard, 2001).


Welcome DEMO-USER

Work Orders | Main Menu

Navigate

Self Service

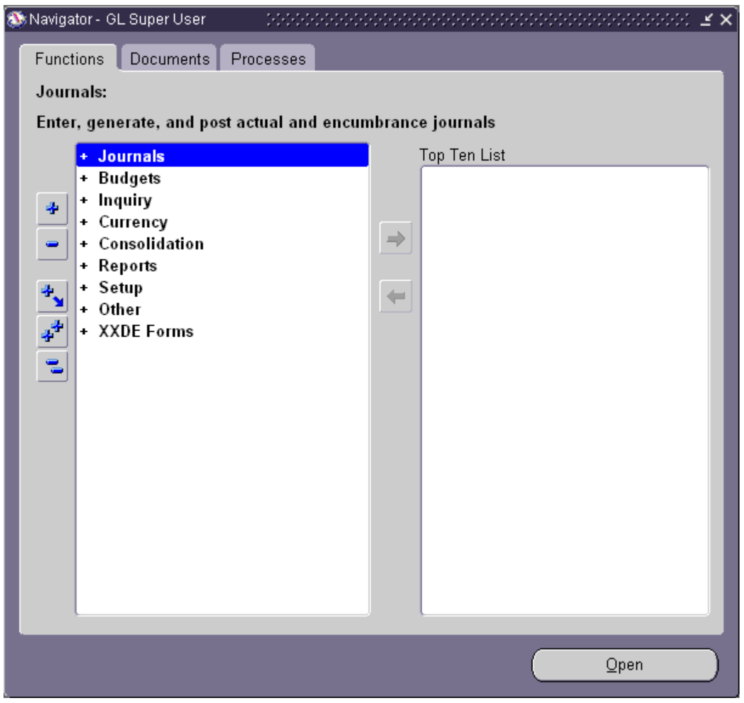
- Candidate Offers
- Carrier Portal
- Credit Cards
- Customer Intelligence User
- Employee Self Service
- Expense Reporting
- Expense Reports
- Workflow User Web Applications

Applications

- AK Developer
- AR Archive Purge User GUI
- AR Trade Accounting
- AX General Ledger User

Figure 3.2 User access responsibilities

Figure 3.3 instead, shows the so-called navigator appearing when choosing a responsibility (in this case a General Ledger responsibility).


Navigator - GL Super User

Functions | Documents | Processes

Journals:

Enter, generate, and post actual and encumbrance journals

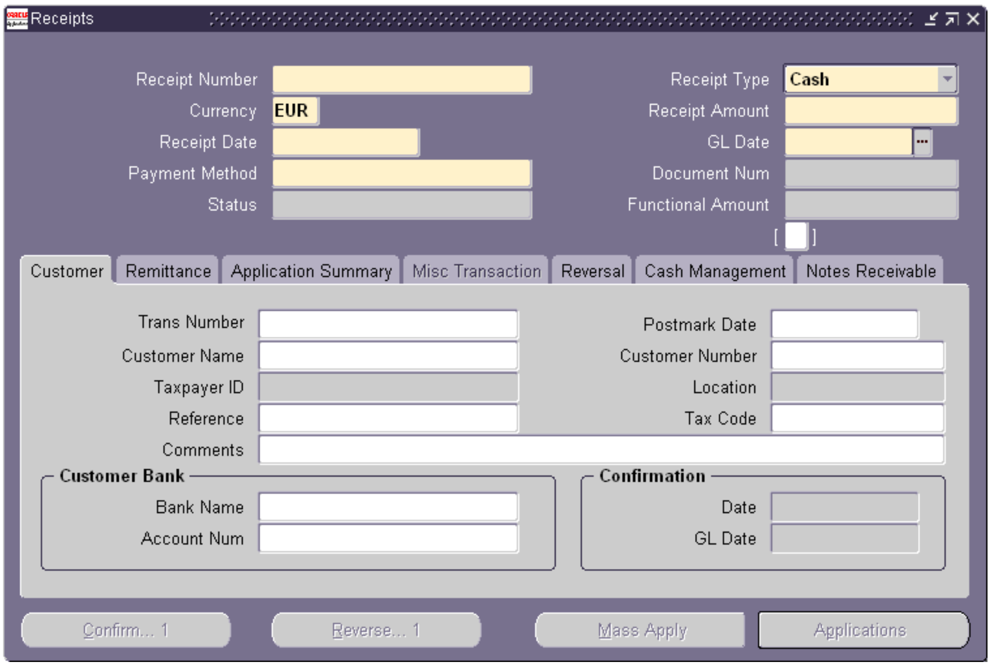
- + Journals
- + Budgets
- + Inquiry
- + Currency
- + Consolidation
- + Reports
- + Setup
- + Other
- + XXDE Forms

Top Ten List

Open

Figure 3.3 Navigator (General Ledger)

Figure 3.4 then illustrates one of the typical forms through which a user carries out his day-to-day work.


Receipts

Receipt Number		Receipt Type	Cash
Currency	EUR	Receipt Amount	
Receipt Date		GL Date	
Payment Method		Document Num	
Status		Functional Amount	

Trans Number		Postmark Date	
Customer Name		Customer Number	
Taxpayer ID		Location	
Reference		Tax Code	
Comments			

Customer Bank

Bank Name	
Account Num	

Confirmation

Date	
GL Date	

Figure 3.4 A typical form (AR Receipts)

3.2 ERP Technical Architecture

Large enterprises use such software systems in order to exploit business information flows. This, in turn, is obviously done to provide efficiency and to reduce costs (especially administrative and IT costs). However, it must be noted that these goals entail adequate underlying technological architectures. Oracle E-Business Suite R11i is built on integrated technology "stack", in which for example, *networking* and *distributed computing* aspects play a relevant role.

The Oracle ERP system architecture is in fact based on a distributed three-tier model consisting of a *client* tier, an *application* tier and a *database* tier as shown in Figure 3.5 (*Oracle Applications Concepts Release 11i*, 2002).

A Java-enabled web browser is located on the client tier (together with a plug-in called JInitiator). This browser manages a number of Java applets through which users can issue their requests.
The middle tier contains a number of components such as Forms Server, Reports Server, Concurrent Managers, and so on.

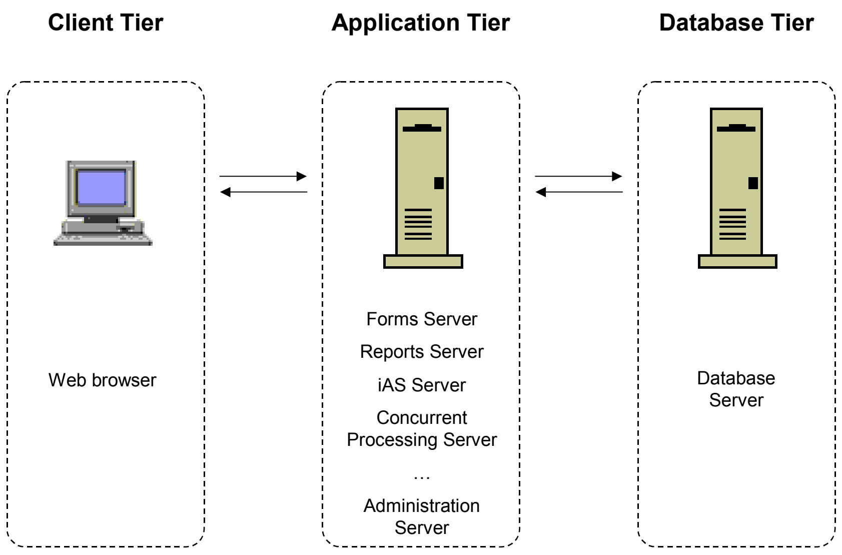


Figure 3.5 Oracle ERP Computing Architecture

As an example, the Forms Client, which is downloaded as a Java applet on the client tier, displays applications screen by cooperating with its counterpart, the Forms Server on the middle tier. In turn, the Forms Server assists client's requests by communicating with the last tier, in which a RDBMS stores and manages all data.
The architecture in this perspective can be seen in Figure 3.6 (*Oracle Applications Concepts Release 11i*, 2002).

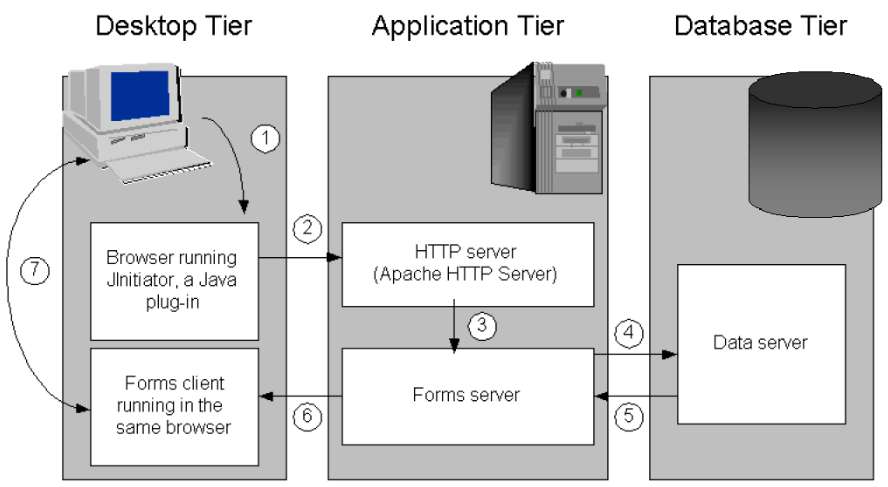

Desktop Tier
Application Tier
Database Tier
1
2
Browser running JInitiator, a Java plug-in
HTTP server (Apache HTTP Server)
3
4
Data server
7
Forms client running in the same browser
6
Forms server
5

Figure 3.6 Forms Architecture

Chapter 4

Software Process Models

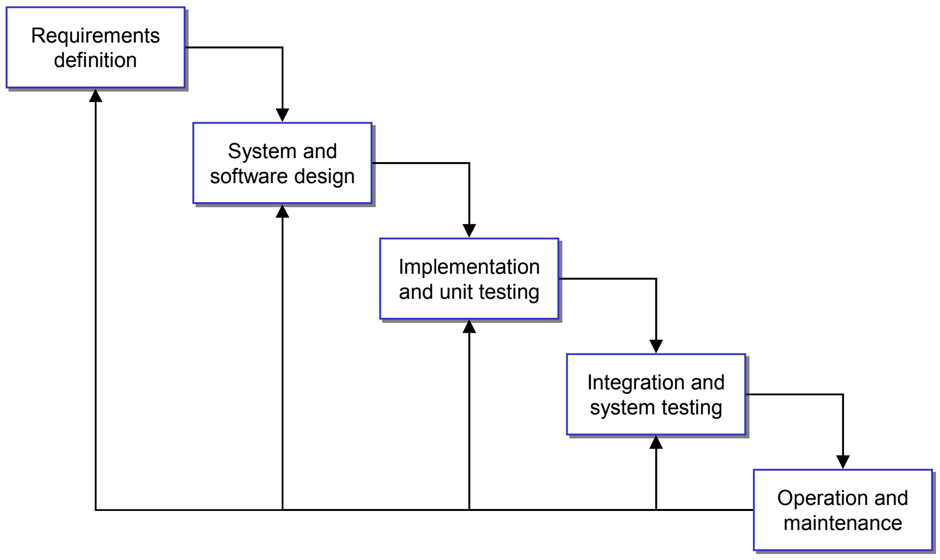
This chapter discusses aspects related to the software process models adopted in the present project.

Generally speaking, when carrying out a software project, all the activities carried out and their organization identify the so-called *software process*.

When dealing with complex software projects a necessary – yet insufficient – condition in order to succeed in the development concerns the degree of structure of the adopted software process. More precisely, unexpected results are less likely to occur when all activities that must be carried out during the project are organised and planned correctly.
The importance of such aspects is rather obvious and in fact nowadays a number of software process models have been proposed and used to support software projects.
Two well-known software process models have been involved in this project. In the next sections they are discussed, and the reasons that led to their adoption in the project are explained.

4.1 Waterfall model

This paradigm considers all fundamental project activities as separate project stages to be carried out in sequence. Thus, for instance, once requirements have been specified and agreed, development goes on to the software design stage, then to the implementation and so on. Figure 4.1 illustrates such software process model (Sommerville, 2001).



Requirements definition
System and software design
Implementation and unit testing
Integration and system testing
Operation and maintenance

Figure 4.1 The waterfall model

With regards to The Company's context, it must be noted that such approach is the one initially considered. That is, at the very beginning of the project the relatively small amount of information available led to adopting such process model as a viable way to start the project. Such process model in fact essentially represents the general software life cycle, and can to some extent be regarded as one of the most natural ways of becoming acquainted with the new work environment and indeed enable the project to start.

4.2 Evolutionary development

The other paradigm taken into account has been the so-called *"evolutionary development"*. According to this process model, an initial implementation is developed and submitted to users for comments. Then, according to users' feedback, refinements to the implementation are integrated. This is done in an iterative way until a satisfactory resulting system has been developed. Figure 4.2 shows a general representation of the evolutionary development (Sommerville, 2001).

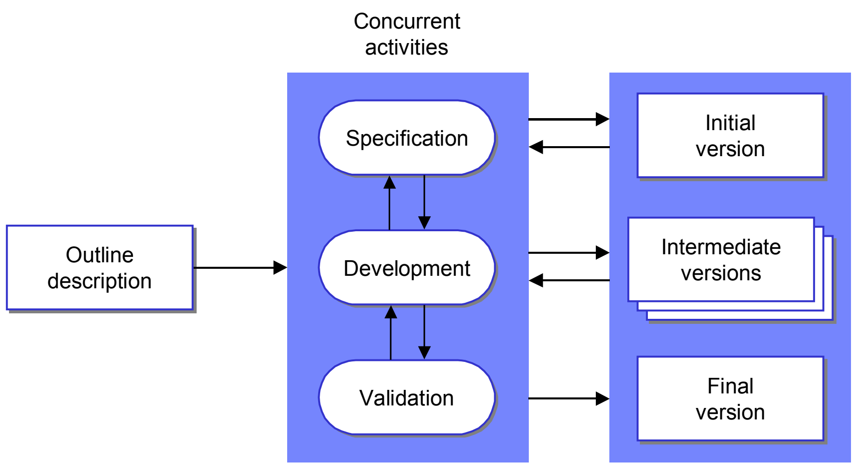


Figure 4.2 Evolutionary development

The characteristic of this software process model is that all main activities such as requirements specification, development, testing and so on, are performed concurrently and a frequent feedback among them is produced.

The way this process model is structured leads to another interesting aspect. It basically allows postponing some of the requirements and design decisions. Conversely, the waterfall model previously discussed does not allow such "flexibility", in that it requires the specification of requirements before proceeding with the design stage and the choice of a specific design approach before starting the implementation. In other terms, each project stage is considerably constrained by the accomplishment of the previous one.

In this project, the evolutionary development was taken into account just after the beginning of the project, as soon as users revealed uncertainties and misunderstandings with regards to the problem. Thus, an initial implementation of the archiving system was developed by means of the high-level requirements initially available. Afterwards, as users developed a better understanding of the problem, a number of details and changes were added to progressively refine the implementation.

4.3 Considerations

As noted, the waterfall model essentially represents the general software life cycle. Although it can be thought of as one of the most natural ways to view the project development it has some limitations, mostly concerning the way each project stage strictly depends on the accomplishment of the previous one.

In contexts such as the one of the present project, merely adopting such software process model may pose several risks. Users would see the financial archiving system only after the main coding activities have been completed. This may lead to a result that does not meet users' needs, and in that case, re-design and re-implementation could have a non-negligible impact.

The drawbacks of this process model were overcome by taking into consideration the above described "evolutionary development" approach. Early implementations were submitted to users in a short time in order to ensure a common understanding of needs, priorities, and so on. On the other hand, continuous checking of users' expectations and satisfaction has entailed a number of changes, and almost always each of them required an increasing effort for integration. Although there was not a great number of changes, the impression (quite foreseeable) is that the integration of many changes could lead to conflicts in the system and reduce robustness of the overall structure. Therefore, system structure and modularity should always be kept in mind, and the changes needed should all be within a well-defined scope (e.g. into one module).

This potential side effect of the evolutionary development approach has been kept under control by the combined adoption of the waterfall model. One of the advantages of the waterfall model derives from the fact that the main project stages are clearly separated and defined. With such approach systems can in fact be more structured, provide better reliability and minor impact for integrating changes.

Concretely, the waterfall model was adopted mostly in the initial stages of the project, when gathering information and specifying requirements. Its adoption was aimed at providing a high-level "guideline" for the whole project so as keep in mind the overall development path and the main system structure. To a certain extent such guideline can be seen in the organization of the present document (chapter 5 deals with requirements specification, chapter 6 and 7 focus on design aspects and all related considerations that emerged after specifying main requirements, chapter 8 provides information regarding the implementation, chapter 9 discusses testing activities).

The evolutionary development approach instead was concretely used for producing all intermediate versions of the financial archiving system. In particular, it was adopted when proposing the initial implementation that consisted of an empty “skeleton” and the interface by means of which the designated user could run the archiving program. It was then used when integrating each financial module. For instance, when dealing with the financial module “General Ledger”, a newer version of the system was provided and submitted to users. In cases like this all activities were carried out by working with users in an iterative scheme until a proper result was reached. Finally, such activities - when feasible - were carried out in parallel with different users, on different parts of the system - for example on different financial modules (further details are provided in chapter 9).

Chapter 5

Requirements engineering

Requirements engineering is the process aimed at specifying what must be obtained by the system being developed and how the system is expected to behave. This process is regarded as one of the key stages in software systems development.
This chapter thus focuses on the requirements engineering stage of the project. It describes the activities through which it was possible to obtain the software specification needed for the financial archiving system. A typical structure of the requirements engineering is the one shown in Figure 5.1 below (Sommerville, 2001), where an outline of the requirements engineering process is given by highlighting the activities involved and their relationships.

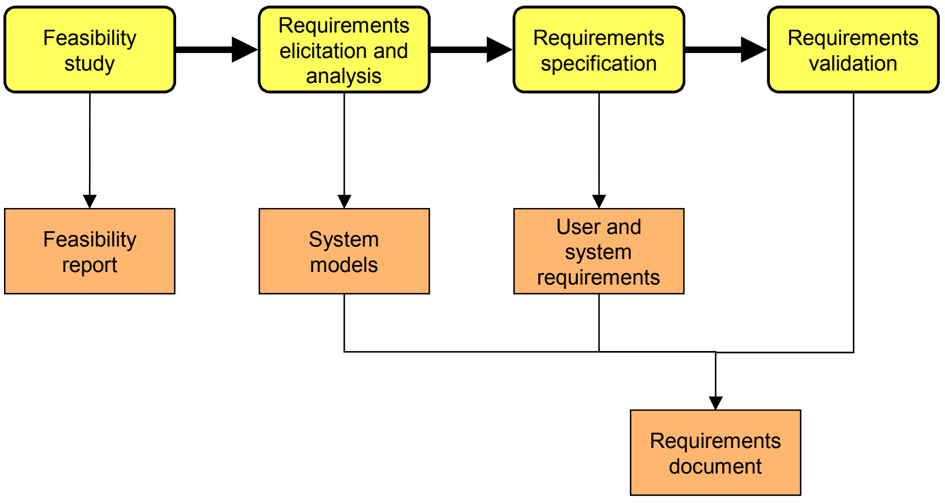


Figure 5.1 Requirements engineering process

The diagram shown here represents a *general* structure of the requirements engineering process. Typically, it serves as a “guideline” that has to be adapted on the specific context. Most of the time, the requirements engineering process cannot be considered as the process of applying a *structured method*. Requirements are almost always subject to changes. This may happen for several reasons: users may suddenly change their needs; political decisions at management level may influence project priority and commitment; other ongoing projects may affect the environment, and so on. Therefore, the requirements engineering process activities cannot be considered as completely separated tasks and executed in sequence. Sometimes, specific needs do not arise until after the system is built. That is, some of the requirements may only come up at later stages, as people involved develop a better understanding of the problem. Thus, these activities are basically executed repeatedly for a certain number of times. To get an idea of such aspect, the diagram depicted in Figure 5.1 can be modified as shown in Figure 5.2.

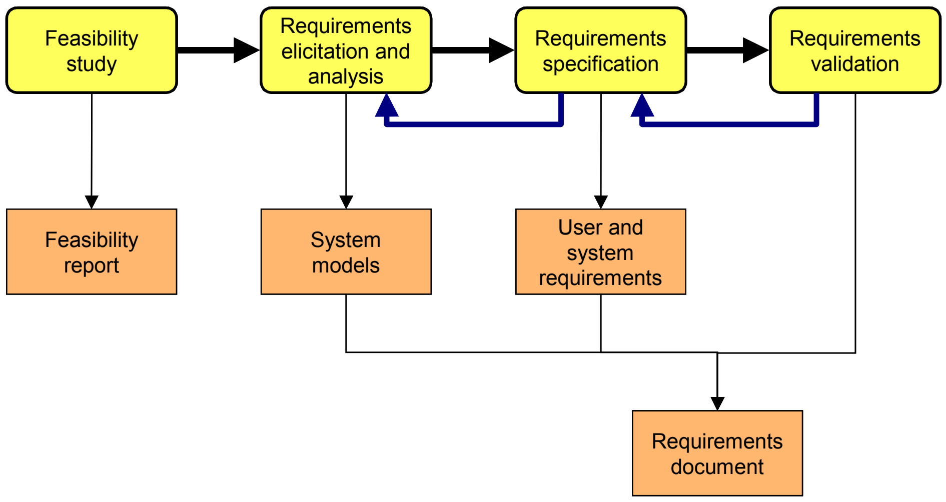


Figure 5.2 Requirements engineering process with feedback connections

It is worth noting that the above mentioned aspects related to changes in the requirements, involve also another specific requirements engineering activity expressly dedicated to requirements *management*. This activity should then be performed in *conjunction* with the other requirements engineering activities (see Figure 5.3).

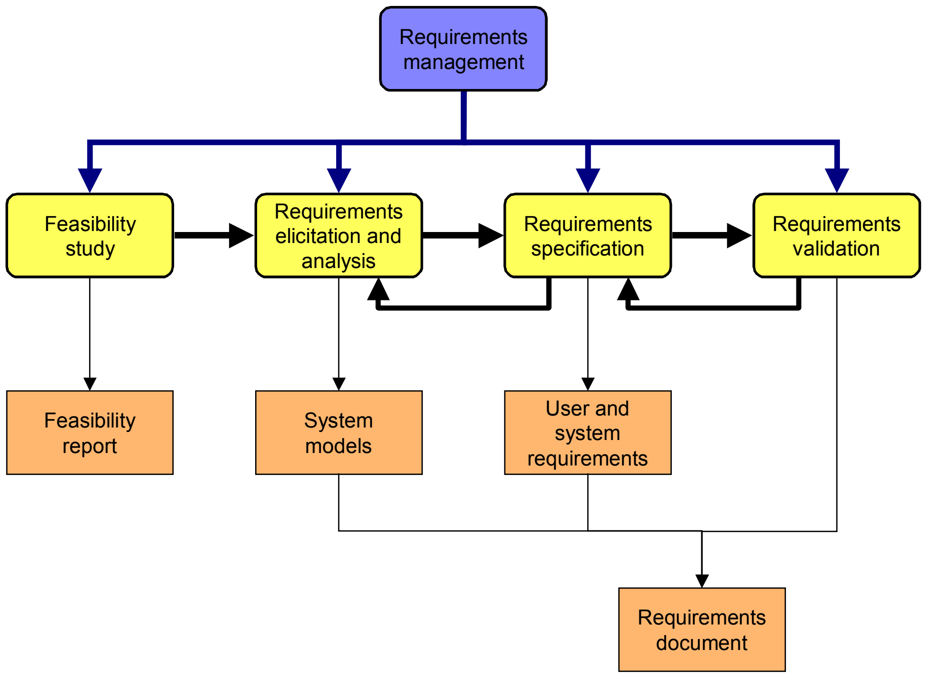


Figure 5.3 Requirements engineering process with management activity

The following part of the chapter describes the key points of the requirements engineering process carried out for The Company, based on the concepts outlined herein.

5.1 Feasibility study

When developing a *new* software system, a feasibility study is the initial step in the requirements engineering process. Such study is normally based on a sketch out of the system and aims at comprehending what extent the system contributes to business objectives. For example, it focuses on aspects related to cost and schedule constraints. It tries to understand whether or not the system requires the use of new technology or what kind of integration with other existing systems is required, and so on.

Analysing these and other related aspects can be problematic due to the influence of some external factors. Worth mentioning, for instance, are business politics or organization strategies. Similar factors play a relevant role in carrying out a project and often prevail over technical factors.

Conducting a feasibility study requires information *assessing* and *gathering*. With regards to the implementation of the financial archiving system in The Company such activities were conducted by participating in meetings as well as arranging and promoting them. Individual interviews were another effective means for these purposes. Meetings and interviews were held with managers of the departments in which the system was destined for use; with the intended end-users; with managers of systems that might be affected by the system being developed. In some cases, the same enquiry was posed to several stakeholders in order to understand how all the people involved conceived the system, and to verify if there was a common degree of awareness regarding the project.
More details on stakeholders involved in the project will be provided in the next subsection.

It must be noted that a brief feasibility study had already been carried out for this project that resulted in a clear intention of putting in place a software system in order to solve the problem described. That is why The Company and Oraplus decided to formulate a project proposal on this topic and looked for resources to allocate to it. In other terms, when Francesco Ferretti accepted to take on the project, it had already been decided that having such a system in place would actually be beneficial for the organization.
Although a clear decision had already been made, Francesco Ferretti considered it appropriate to integrate The Company's feasibility study with some additional information in order to make some ambiguous aspects clearer, and to develop a better understanding of the users' needs from the very beginning of the project.
Thus, by reviewing The Company's feasibility study and integrating additional information, a "starting picture" was drawn, as described by the following key points:

- A software system that allows archiving the historical data of each branch of the group needs to be devised in order to allocate more processing power to online data and thus overcome management issues on the central data repository.

- Most of the archived data should be kept in its original form in order to comply with legal requirements.
- Archived data will be used for data warehousing activities and, when needed, it should be made available for producing relevant reports for auditors.
- The financial archiving system should be integrated in the ERP system currently used by The Company.
- Employees from the accountancy department will be the users of the system.
- Permission to submit the archiving process will be granted only to a designated *"key user"*.
- When submitting the archiving process, the key user will indicate the financial modules on which archiving must be performed (e.g. General Ledger, Accounts Receivables, Purchasing, etc.) and will provide a time range to be considered.
- The archiving process should be run as needed. However, it is expected to be run every few months.
- Only technology currently in use in The Company's IT department will be allowed for developing the financial archiving system.
- The system should be developed according to the main development recommendations of The Company and Oraplus.

This review is aimed at clarifying some high-level aspects, such as business objectives and motivation, commitment and management attitudes, degree of freedom under a technical point of view, the people involved (stakeholders identified are described in the next subsection), and so on.
By addressing such aspects, this review intended to ensure a common initial understanding in order to enable the project to start in the proper way.

5.1.1 Stakeholders

System *stakeholders* were identified while reviewing and integrating The Company's feasibility study. Such term is used to identify all entities that in some way influence the project. For The Company, these include:

- The ERP system currently in use.
- Accountants, acting as system end-users.
- Accounting managers, also acting as system end-users but with approval authority.
- Internal and external auditors (e.g. fiscal authorities).
- The author and his manager.

5.2 Requirements elicitation and analysis

Once a feasibility study has been conducted and the project has been considered worth implementing it is necessary to proceed with the *requirements elicitation and analysis*. Figure 5.4 highlights the collocation of the requirements elicitation and analysis, recalling the requirements engineering process structure shown at the beginning of the chapter.

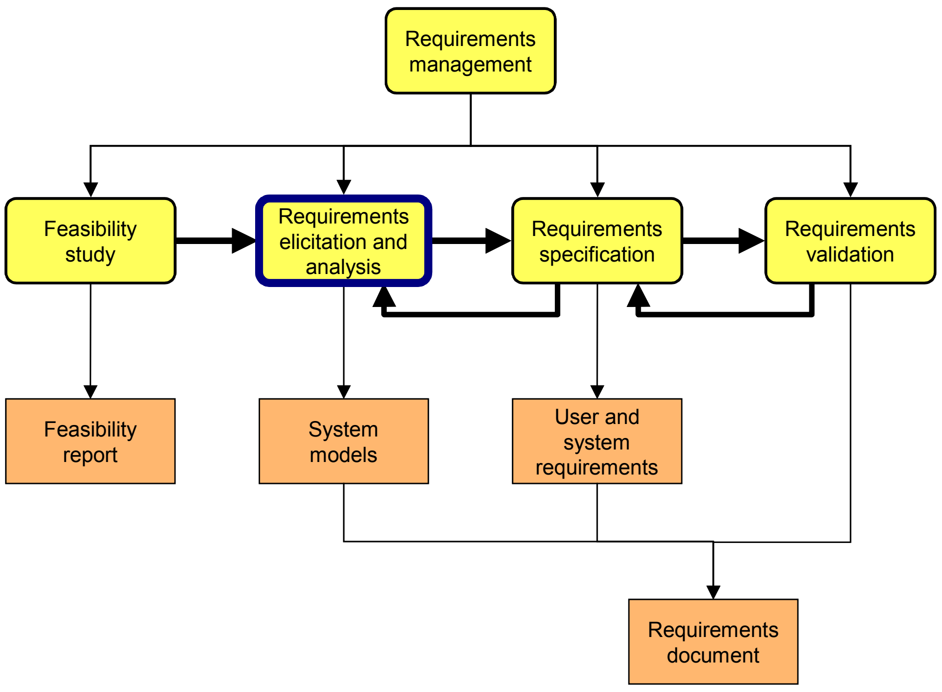


Figure 5.4 Requirements elicitation and analysis in the requirements engineering process

The aim of the requirements elicitation and analysis process is to identify the application domain, the set of services that the system should provide, the constraints posed to the software system, and so on.

This stage may involve diverse people in the organization and in turn this could generate some difficulties. In many cases users may find it complicated to explain what they want from the system or they may know it in general terms only. In other cases, users may pose unreasonable requests because they are unaware of the implications concerned. Furthermore, users tend to express their requests using a terminology pertaining to their own work (e.g. accounting terms).
As in feasibility studies, politics and strategic factors may play a relevant role, in this case influencing requirements. Changes in the business environment also represent an aspect that may influence requirements considerably.

The requirements elicitation and analysis process involves a number of activities. These depend on the specific context, but typically a set of general activities can be identified:

- ***Domain understanding*** An adequate knowledge and understanding of the application domain must be developed by the analysts. With regards to The Company's application domain, this activity has been particularly time consuming because of the need to understand the business processes involved (e.g. accountancy procedures).
- ***Requirements collection*** Stakeholders' requirements must be gathered through interviews, meetings and other similar means of interaction.
- ***Classification*** The whole set of requirements gathered must be rearranged into logical groups.
- ***Conflict resolution*** When dealing with a large variety of stakeholders, requirements may conflict. It is then necessary to locate such conflicts and sort them out.
- ***Prioritisation*** Each requirement is to be clearly described also in terms of "priority". That is, not all the requirements identified have the same importance.
- ***Requirements checking*** Requirements must be consistent and must reflect the stakeholders' needs. Thus, proper checks are needed on requirements.

– ***Requirements approval*** Although it may seem obvious, in large organizations, the requirements identified have to be formally approved by relevant managers before proceeding with next stages of the project.

At the beginning of this chapter it was mentioned that requirements engineering process activities cannot be considered completely separated tasks and executed in sequence, but have to be performed repeatedly for a certain number of times. The same applies to the requirements elicitation and analysis process. Given their nature, the above described activities are in fact executed as an iterative process with feedback presence between each activity. Such idea can be illustrated in Figure 5.5 (Sommerville, 2001). Given the specific context, the original diagram was adapted by integrating the block related to the requirements approval.

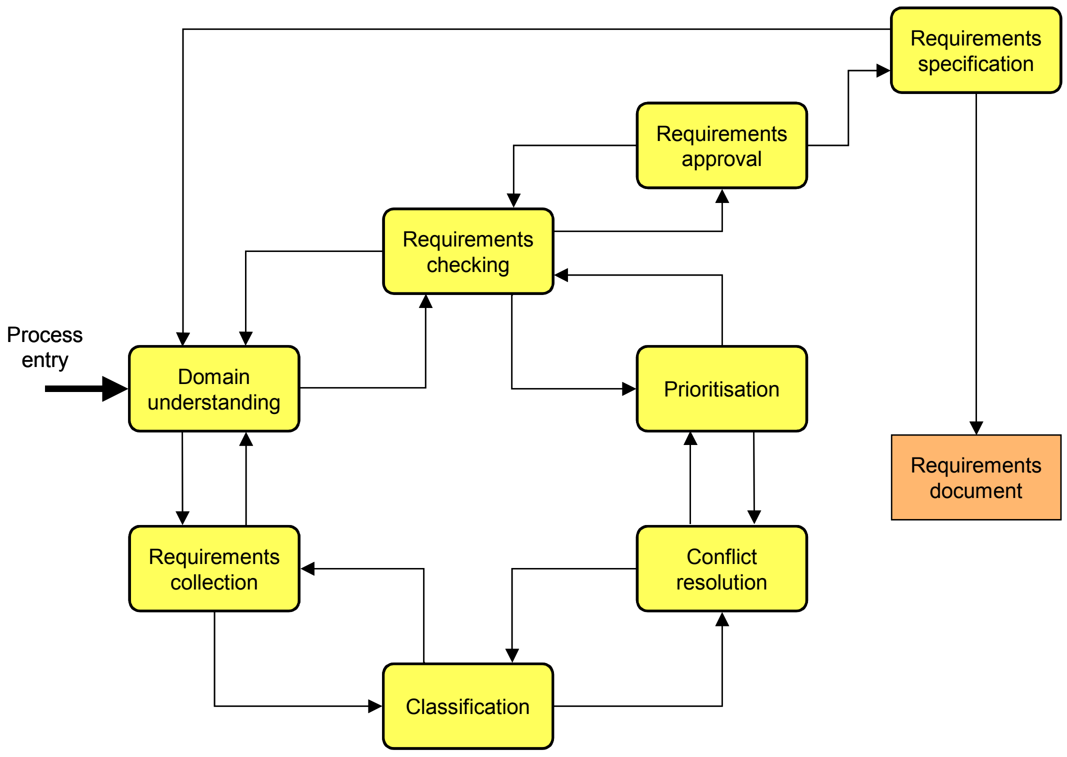


Figure 5.5 Requirements elicitation and analysis process

Requirements elicitation and analysis can be carried out with several techniques. Of the several techniques evaluated to carry out this project certain were considered particularly interesting. A description of their key points is listed below. The approach used, along with the underlying reasoning, is presented further ahead.

Viewpoint-oriented elicitation Several "viewpoints" can be identified when dealing with systems involving many different kinds of users. For example, the interest of end-users is usually different to that of managers or other stakeholders. This basically means that a problem may be seen in different ways. Thus, a viewpoint-oriented approach takes into account the existence of various perspectives, thereby allowing one to discover *conflicts* in the requirements proposed by stakeholders.

Scenarios This technique is based on the use of concrete examples of interaction with the system being developed. Instead of using abstract descriptions, requirements engineers may find it more profitable to discuss with stakeholders on *sample interaction sessions* (referred to as *"scenarios"*). Typically, an initial scenario of a specific interaction with the system is outlined. Then, during the process, further refinements and details are added to achieve a complete description of the given scenario. Thus, a key strength of scenario-based elicitation is that it allows capturing detailed information on the expected system behaviour, facilitating in this way the specification of requirements description.

Ethnography Ethnography-based techniques emphasize aspects related to the social and organizational context in which software systems are placed. In fact, when developing a software system one must keep in mind that such system will not work in isolation but will be used in a social and organizational environment, and system requirements may be deduced or constrained by that environment. Ethnography is useful in discovering implicit process details which reflect the gap between the assumed and the actual processes in which people are involved. However, ethnography is not appropriate for eliciting organizational or domain requirements since it mainly focuses on end-users. A better approach suggests the use of such technique in combination with other methods.

Structured analysis methods Such methods involve an analysis of the system that generates a set of system models. These models provide a graphical representation of the problem and the system that is supposed to provide a solution. They usually define their own process to derive the set of system models. Although structured methods play a considerable role in requirements engineering they have several weaknesses, especially in the early stages of the requirements engineering process. They do not support functional requirements modelling effectively. Often, they generate a lot of documentation which tends to hide the core part of the system requirements. In fact, in such documentation the models identified are very detailed and users often are not able to understand them, and thus verify the suitability of these models.

Prototyping Prototyping aims at exploiting users' involvement for supporting requirements elicitation, analysis and validation. This technique is based on the adoption of software system prototypes. A software system prototype represents a preliminary version of the software system to be adopted by the organization. By means of software prototypes users can get a concrete feedback on their previous requests and support in verifying correctness of system requirements. This way, analysts and development staff can get more precise information on users requirements, test and exemplify concepts, try out design alternatives, and so on.
Prototyping requires rapid development in order to keep costs under control and let users start experimenting with the prototype in the early stages of the project.

5.2.1 The approach used

Although the above described techniques represent relatively general approaches, each of them has different strengths and weaknesses. Some of them may carry out a specific task more effectively than the others, and vice versa. As practice suggests, best results can be achieved by combining together several techniques so as to take advantage of the strength of each. Concerning The Company's scenario, it was considered suitable to use an approach based on aspects derived from *scenarios*, *ethnography* and *prototyping* techniques.

The rationale for such decision is described by the following considerations resulting from the analysis of The Company's context:

- The problem posed has a remarkable functional component, in that main topics are related to accounting work. At the same time, the feasibility study highlights the presence of a small number of interacting sessions. These considerations suggest that at least part of the requirements could be specified using a few sample interacting sessions during which users describe their needs and also explain accounting topics.

- The problem to be solved involves social and organizational aspects related to several countries (e.g. the legal requirements regarding data retention imposed by each local tax authority). That is, the system being developed will be placed in a fairly large environment. Thus, system requirements may be deduced or constrained considerably by that environment. One then needs to immerse oneself in observing work procedures pertaining, for example, to each of the countries, to each of the accountancy areas, and so on.
  Such considerations denote an "international" nature of the project framework and suggest the need of observational techniques such as ethnography.

- The system to be developed will affect the production environment (that is the ERP system in use) by in some way archiving production historical data. In such cases where main production systems are affected by the introduction of another system, a specific path must be followed before integrating the new system. This is needed in order to assure that the new system will not pose any risks for the production environment. This path requires that tests and demonstration activities be carried out on test environments. Such tests and demonstrations typically aim at verifying specific parts of the system. Therefore, the system being developed is a sort of prototype that evolves with every test. Furthermore, the use of prototypes increases when dealing with a variety of different kinds of functional data, as is The Company's case (e.g. general ledger data, accounts receivable, accounts payables, etc.). Among other, these considerations lay down the motivations that suggest adopting an evolutionary/prototyping development as software process model.

– The system being developed will be used by a number of employees in The Company. These are basically a few accountants and a few managers, in both cases acting as end-users, as well as the author and his manager who carry out the project. According to viewpoint-oriented elicitations techniques, this means that more than one "viewpoint" can be found in the project. At the same time, such techniques entail the presence of a relevant number of different viewpoints. In other terms, the use of viewpoint-oriented elicitation techniques is justified when dealing with systems involving *many* different kinds of users, which in turn leads to a number of different viewpoints. Only in such cases can the strengths of these techniques be concretely exploited.

– Adopting structured analysis methods presumes the presence of a certain type of stakeholders, meaning that people involved must have the knowledge required to understand the system models generated (that are typically very detailed). Stakeholders' availability, in terms of time, must also be taken into account. Structured method analysis may require stakeholders to dedicate a significant amount of time to analysts and development staff. Such time availability is often not conceded due to external constraints. Although there is strong commitment in the project, other ongoing critical projects are keeping the stakeholders busy (or with unpredictable time schedules). It must also be noted that structured analysis methods do not support functional requirements modelling effectively, and in The Company's context such component is relevant.
  These considerations do not suggest the use of strong structured analysis methods for requirements elicitation and analysis. On the other hand, such methods may be used in other stages of the project, such as the design stage, which usually demands less time commitment from users, and gives more autonomy to analysts and the development staff.

Therefore, in summarizing the above considerations it can be observed that:
the nature of the project, together with its corresponding context, has suggested carrying out the requirements engineering process (and to some extent the more general software process too) interacting closely with end-users, interviewing them and immersing in their environment,

discussing with end-users on sample interacting sessions, and letting them try out system prototypes at increasing levels of refinement.

The next subsection describes the requirements specification resulting from the elicitation and analysis activities.

5.3 Requirements specification

The requirements engineering activities described in the previous part of this chapter led to the specification of a set of requirements, which in turn were used for the subsequent development of the financial archiving system.

The next two subsections highlight how the requirements have been *classified* and *specified*.

5.3.1 Classification

The requirements identified have been classified so as to structure them and provide a logical organization. Requirements then fall into one of the following "class" of requirements:

> **Functional Requirements** This class groups all the requirements that aim at describing the functionality that must be provided by the system. Requirements belonging to this class can be seen as those that focus on the *key services* to be implemented in the system.
>
> **Non-Functional Requirements** Here are collected all the requirements not directly related to individual functions provided by the system. These are essentially product requirements that refer to the system as a *whole*. They mainly act as constraints on functions provided by the system or on processes carried out during the project (e.g. standards or other recommendations to be applied to the development process).

Domain Requirements To this group belong all the requirements that originate from the application domain of the system being developed. They are not derived from specific user needs but tend to reflect basic characteristics of the application domain (e.g. legal requirements imposed on data retention).

These classes have been assigned a two-letter code as follows:

FR	Functional Requirements
NF	Non-Functional Requirements
DR	Domain Requirements

Such code was then used as part of the requirement identifier (see next subsection).

5.3.2 Specification language

The requirements shown here are meant to be used as “input” for the subsequent design and development stages. It must be noted that they are also needed by end-users in order to let them understand (and verify) how the system being developed will be characterised, and most importantly, by managers for the required approvals.

Typically, end-users do not have much technical knowledge and thus requirements must be written in such a way that allows them to be completely understood by readers. This means that requirements have to be written using, for instance, natural language statements, intuitive diagrams and so on. On the other hand, such approach may be prone to a number of problems. Writing requirements descriptions in a simple and intuitive way in order to keep them short and easy to read may result in ambiguities and inaccuracies. Just as an example, several concepts and requirements may easily be merged together in a single description.

For these reasons, specification of the following requirements was carried out with a template. Figure 5.6 shows the template invented expressly for the project.

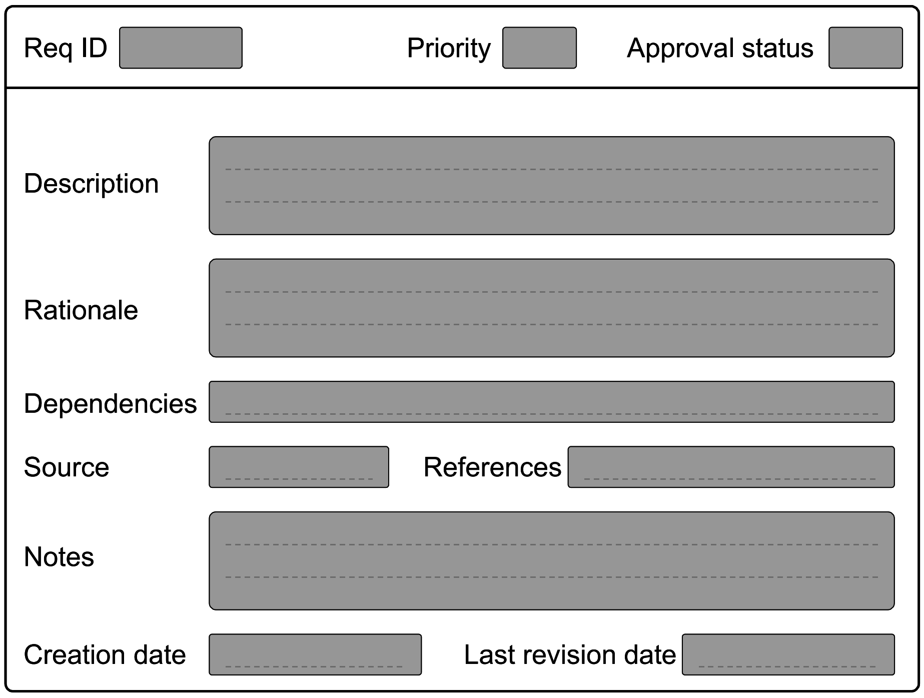


Figure 5.6 Template for expressing requirements

During the requirements engineering process printed copies of this template were used for supporting the activities. Their use, in occasions such as meetings and interviews, helped in minimising the above mentioned problems. By structuring the information assessment, it was possible to identify omissions and conflicts more easily and to recall, review, and manage the whole set of requirements more quickly.
As can be seen, key information such as the requirement description and the corresponding rationale, are located in a "central" position in order to be clearly identified. Other fields are available for information, such as the name of the person that raised the requirement (field "source") or reference, to relevant material (field "references"). The upper section is dedicated to the information that was used frequently when reviewing and managing the set of requirements. In these fields are noted down the requirement identifier, the priority associated, and the corresponding approval status. This information was located in the upper part as a separate section in order to facilitate the examination of the whole set of requirements. Most of the time all the requirements (especially those with high priority) had to be checked to make

sure they had actually been approved. Having this information in a clearly visible area on the top makes it easier to go through the whole set of requirements.
Moreover, in order to minimise errors when specifying requirements care was taken to use the language with consistency avoiding, for example, computer jargon or ambiguous terminology. Also, as practice suggests, *mandatory* requirements were expressed using the term "shall" and *desirable* requirements using the term "should".
At the same time, as per user request, details have been left out, whenever possible, so as to keep a certain degree of freedom and innovation in the process.

The next three subsections illustrate the requirements identified. For the sake of clarity, not all the information shown in the above mentioned template has been reported in the present document. Only the information that was considered more relevant for the reader has been described. More precisely, for each requirement it is stated:

Identifier	A unique identifier of the form "Req/YY/XX" where YY is a two-letter abbreviation corresponding to the requirement class, and XX stands for a sequential number.
Description	A statement describing the requirement.
Rationale	A statement explaining the reasons underlying the requirement.

5.3.3 Functional requirements

Below are reported the functional requirements that, as mentioned, aim at describing the functionality that must be provided by the financial archiving system.

Req/FR/01	A software system shall be put in place for archiving *historical financial data*.
Rationale:	It is required to devise a software system that allows to archive historical data of each business units of the company in order to allocate more processing

power to online data and thus overcome management issues on the central data repository. See Req/FR/05 for a complete list of all business units.

Req/FR/02 Archived data shall be made available to accountancy users.

Rationale: Data warehousing activities will be carried out on archived data. Thus, relevant users shall be able to access the archived data. That is, data shall not be placed, say, on tapes (on a shelf), otherwise they won't be always accessible (extraction from the tape would be required).

Req/FR/03 The archive program will require the following parameters:

- List of modules to be archived (e.g. by means of flags).
- Time range.

Rationale: When submitting the archiving program, the key user will indicate the financial modules on which archiving is to be performed and will provide a time range to be considered.

Req/FR/04 It should also be possible to schedule the archive program to run automatically with an arbitrary frequency.

Rationale: This is needed to minimize manual intervention, as additional feature.

Req/FR/05 Business units involved in the archiving process are: Belgium, Denmark, France, Germany, Ireland, Netherlands, Spain, Sweden and UK.

Rationale: Archiving has to be performed on most of the business units. At the moment of writing, these are the ones specified in the above list.

Req/FR/06 A given recorded transaction can be archived when the following conditions are satisfied:

- It belongs to one of the modules specified during the archive program submission.
- It refers to a date that falls into the time range specified during the archive program submission.
- It represents a "closed" transaction (see glossary for "closed transaction").

Rationale: In practical terms, a transaction can be considered eligible for archiving only if it is not needed by other transactions in the future. That is, no further actions will be taken against that transaction.

Req/FR/07 Ideally the archiving system shall perform data archiving on *all* financial modules. However, it shall include at least the following modules:

- General Ledger
- Accounts Receivables

Rationale: The financial archiving system being developed is meant to perform data archiving on *all* financial modules. However, the highest priority is given to those module (listed above) that involve higher volume of data.

5.3.4 Non-Functional requirements

In this section are described those requirements that refer to the system as a whole rather than to individual functions.

Req/NF/01 The system shall be *integrated* in the ERP system in use.

Rationale: The financial archiving system has to become a kind of "additional feature" of the current ERP system.

Req/NF/02 The archive program shall be submitted through the standard ERP submission window.

Rationale: As part of the Req/NF/01, the archive program is considered as a standard ERP program.

Req/NF/03 Only software applications and software technology currently in use in The Company's IT department are allowed to be used for developing the financial archiving system.

Rationale:	As part of the Req/NF/01, this is needed in order to provide integration and homogeneity with the ERP system.
Req/NF/04	Only available hardware in The Company's IT department is allowed to be used for developing the financial archiving system.
Rationale:	At this stage, IT budgets will not approve purchasing of additional hardware.
Req/NF/05	Whenever applicable, development standards and recommendations of The Company and Oraplus shall be adopted.
Rationale:	As part of the Req/NF/01, this is needed in order to provide integration and homogeneity with the ERP system and to fulfil IT policies.
Req/NF/06	The system shall provide facilities for verifying the correct execution of the archiving program.
Rationale:	Once the archiving program has been submitted, it is necessary to verify that the normal execution completion has been reached.
Req/NF/07	The archiving program shall provide logging facilities for tracing phases performed during its execution.
Rationale:	Once the archiving program has terminated its execution, it shall be possible to inspect a log in which all the key phases carried out are listed.
Req/NF/08	The archive program shall first verify that all prerequisites are satisfied, before performing any archive operation.
Rationale:	A number of checks have to be performed in order to ensure the correctness of the process.
Req/NF/09	A specific user shall be designated for the submission of the archiving program.
Rationale:	This is needed to ensure that the archiving program will be executed only by a proper *"key user"*.
Req/NF/10	Appropriate user documentation has to be provided.

Rationale: The designated *key user* has to be provided with an adequate user guide that supports him by explaining how to run the archiving program.

Req/NF/11 Training has to be provided to key user.

Rationale: The designated *key user* has to be trained in order to give him autonomy in the usage of the archiving program.

5.3.5 Domain requirements

In this section are taken into account those requirements that originate from the application domain of the system being developed.

Req/DR/01 Data archiving shall involve all data belonging to *financial* modules.

Rationale: Data belonging to modules such as General Ledger, Accounts Receivables, and so on, are considered as ***financial*** data and thus as data involved in the archiving process.

Req/DR/02 The time range specified while submitting the archiving program has to comply with the *accounting calendar* defined in the ERP system.

Rationale: This parameter has to fit with the accounting calendar in use. Such calendar is different from the solar calendar (it usually consists of 13/14 periods).

Req/DR/03 Historical data are those that refer to past fiscal years.

Rationale: The archiving process takes into account only ***historical*** data.

Req/DR/04 Archived data shall be kept in its ***original*** form.

Rationale: This is needed in order to comply with legal requirements and to give the ability to reproduce fiscal reports upon request.

Req/DR/05 Access to archived data shall be granted in ***read-only*** mode.

Rationale: This is needed in order to ensure data integrity. No one is allowed to change data that is supposed to be only for consultation.

Req/DR/06 For each business unit, the archiving process has to be performed on *every set of books*.

Rationale: A business unit consists of one or more financial reporting entity (so-called *set of books*). All set of books belonging to a business unit (subject to archiving) have to be archived.

Req/DR/07 The following data retention requirements have to be fulfilled:

Country	Data retention requirement
Belgium	Accounting records must be kept for a period of **10** years as of January 1st of the year following the closed accounting year.
Denmark	Accounting records must be kept for a period of **10** years.
France	Accounting records must be kept for a period of **10** years.
Germany	Documents are retained for **6** years, all other accounting records for **10** years.
Ireland	Accounting records must be kept for a period of **6** years after completion of transactions to which the accounting records relate.
Netherlands	Accounting records must be kept for a period of **10** years.
Spain	Accounting records must be kept for a period of **4** years from the last recorded entry.
Sweden	Accounting records must be kept for a period of **10** years.
UK	Accounting records must be kept for a period of **6** years plus current year.

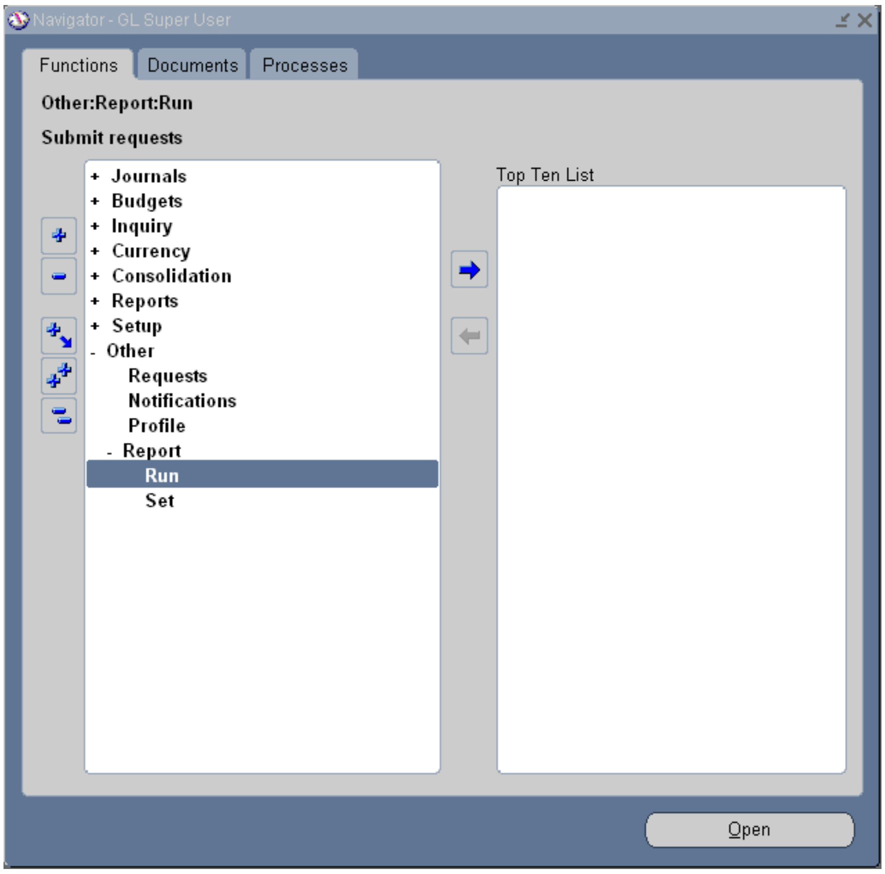
Table 5.1 Legal data retention requirements

Rationale: In each country, local authorities specify a period of time during which it is required to keep historical data. Therefore, historical data can be deleted only if older than the corresponding country-specific data retention requirement.

5.4 Sample scenarios

In addition to the requirements specified in the previous section, key sample scenarios have been identified to further describe users' needs.
The sample scenarios here described relate to the interaction with the archiving program, that is the component through which the designated key user will be able to archive data.

Archiving program execution Execution of the archiving program is performed through the standard requests submission window in the ERP system. The key user will connect to the ERP system and navigate to the standard requests submission window by choosing the menu items *"Main navigator → Other → Report → Run"* (also see Figure 5.7):



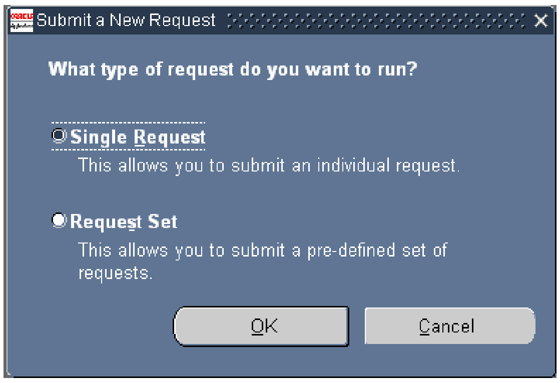
Other:Report:Run

Submit requests

```
+ Journals
+ Budgets
+ Inquiry
+ Currency
+ Consolidation
+ Reports
+ Setup
- Other
    Requests
    Notifications
    Profile
  - Report
      Run
      Set
```

Figure 5.7 Archiving program – menu path selection

Then, choose "Single request" in the dialog box appearing (also see Figure 5.8).


Submit a New Request

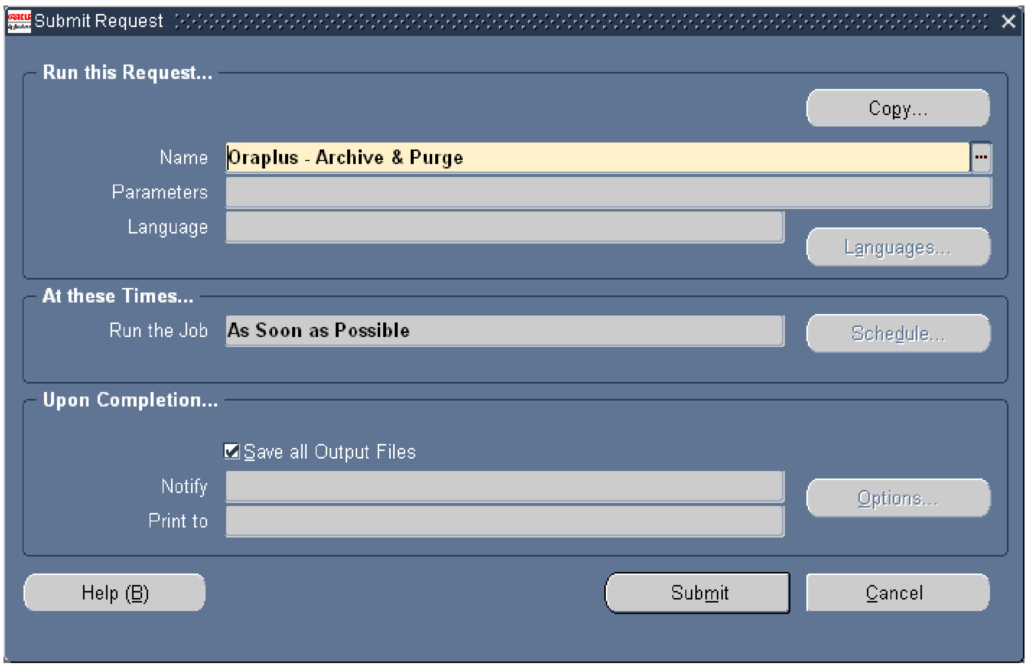
What type of request do you want to run?

● Single Request
This allows you to submit an individual request.

○ Request Set
This allows you to submit a pre-defined set of requests.

Figure 5.8 Archiving program – Single request selection

The window shown in Figure 5.9 will thus appear and the archiving program will be invoked by typing its name in the first field.


Submit Request

Run this Request...

Name: Oraplus - Archive & Purge
Parameters:
Language:

At these Times...

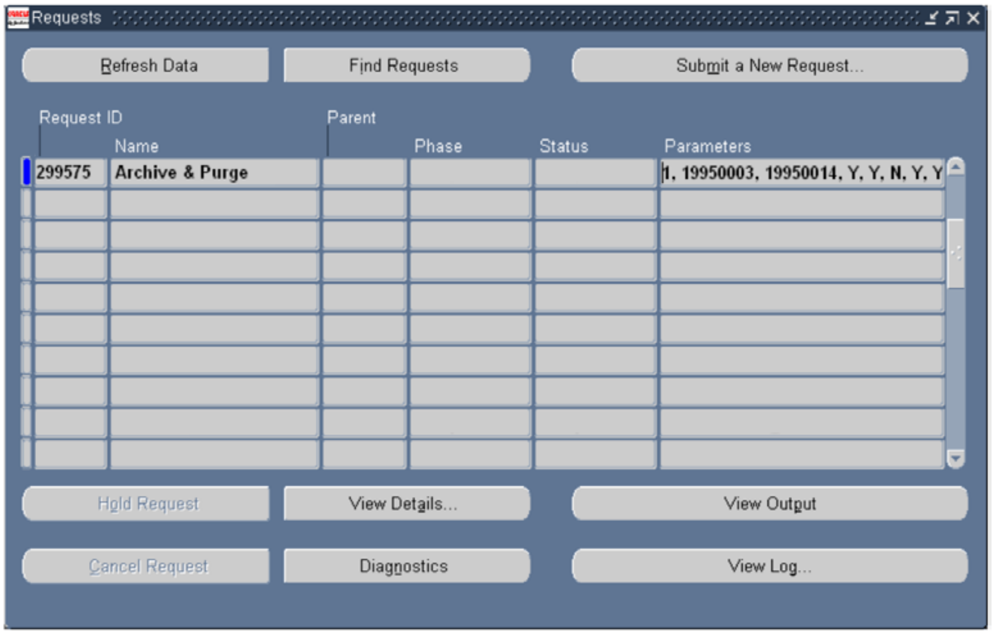
Run the Job: As Soon as Possible

Upon Completion...

☑ Save all Output Files
Notify:
Print to:

Figure 5.9 Archiving program - invocation

Once the program has been submitted, a new window will appear showing the current program status (see Figure 5.10). When the execution terminates (i.e. field "Phase" indicates "Completed"), the key user has to verify the successful program completion (i.e. field "Status" indicates "Normal") and check log and output files (by pressing the two buttons "View Log…" and "View Output").

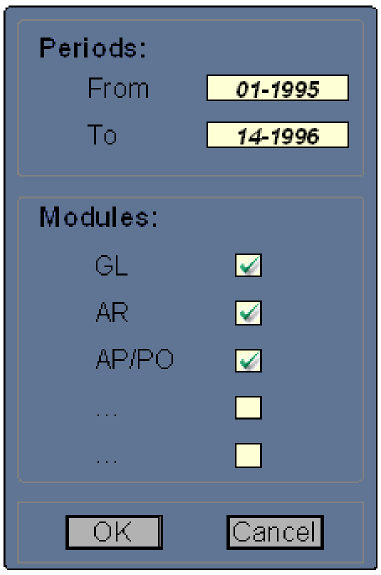


Request ID	Name	Parent	Phase	Status	Parameters
299575	Archive & Purge				1, 19950003, 19950014, Y, Y, N, Y, Y

Figure 5.10 Archiving program - status window

Parameters When executing the archiving program, parameters specification should look like as shown in Figure 5.11.

The upper area named "Periods", requires two values to be entered in order to define the time range into which financial data has to fall into. The value specified in each of these two fields is based on the accounting calendar adopted by The Company and is expressed as a six-digit number "XX-YYYY". The first two digits indicate the *accounting period number*, while the last four digits the *accounting year*.


Periods:

From 01-1995

To 14-1996

Modules:

GL ✓

AR ✓

AP/PO ✓

... ☐

... ☐

Figure 5.11 Parameters specification in the archiving program (sample window)

Thus, in the above example, the value *"01-1995"* stands for the first accounting period of the accounting year 1995. Likewise, the value *"14-1996"* corresponds to the fourteenth accounting period of the accounting year 1996. Relating these values to the solar calendar, the above example indicates the interval between "01-APR-1995" and "31-MAR-1997". That is, the accounting calendar starts from April and ends with March (next year).

In this context the accounting calendar consists of 14 periods, where two of them (13th and 14th) represent "adjustment" periods and span only over the last day of the accounting year. These represent "adjustment" periods that are normally used to allow final accountancy corrections before going into the next year.

Values in the field *"From"* are to be considered as starting from the *first* day of the specified accounting period. Similarly, values in the field *"To"* are to be considered as including the *last* day of the specified accounting period.

Table 5.2 illustrates the correspondence between the accounting and solar calendars for year "1995".

Accounting Calendar	Solar Calendar			
01-1995	From	01-APR-1995	to	30-APR-1995
02-1995	From	01-MAY-1995	to	31-MAY-1995
03-1995	From	01-JUN-1995	to	30-JUN-1995
04-1995	From	01-JUL-1995	to	31-JUL-1995
05-1995	From	01-AUG-1995	to	31-AUG-1995
06-1995	From	01-SEP-1995	to	30-SEP-1995
07-1995	From	01-OCT-1995	to	31-OCT-1995
08-1995	From	01-NOV-1995	to	30-NOV-1995
09-1995	From	01-DEC-1995	to	31-DEC-1995
10-1995	From	01-JAN-1996	to	31-JAN-1996
11-1995	From	01-FEB-1996	to	28-FEB-1996
12-1995	From	01-MAR-1996	to	31-MAR-1996
13-1995	From	01-MAR-1996	to	31-MAR-1996
14-1995	From	01-MAR-1996	to	31-MAR-1996

Table 5.2 Accounting and solar calendars

Chapter 6

Analysis

This chapter discusses key considerations and reasoning that have led to the solution provided to The Company. It starts by addressing a number of initial considerations (first section) and then focuses on three important aspects on which the financial archiving system were later based (second section).

6.1 Considerations

The review of all the identified requirements resulted in a number of initial considerations.

First of all, there is a clear need of improving performance and manageability of the ERP system and its server machine. Such objective is to be pursued by archiving financial historical data.

From a technical perspective, in this context the term "archiving" means the financial historical data that must be copied "somewhere" and then deleted from the ERP database. This way, the ERP database will manage only data belonging to the current fiscal year and, if needed, one or two past fiscal years. In other terms, "online" data will be represented by a considerably small fraction of the whole current volume of data. Benefits are also guaranteed by several investigations carried out on test environments before starting the project and especially by comparing current, degraded ERP performance with that at the very beginning of the centralization project (global single instance) when the ERP system was put in place (and historical data coming from all branches of the group were not yet in).

In theory, historical data could be stored on some kind of storage media such as magneto-optical medias or similar. However, this option would require adequate hardware for recording and making data available. This, besides posing a non-negligible economic impact, fails to comply with those requirements specifying that no additional hardware or new technology will be approved for purchasing by IT budget plans.
In any case, even if some kind of hardware with such capabilities should be acquired, its average access time, combined with the considerable work-load generated by data warehousing activities, could easily make this option not fully satisfactory.
Moreover, the amount of data to be archived tend to increase quickly and even if a *fixed* number of past fiscal years should be considered for storage, the amount of data to be archived would grow unpredictably due to other factors (for example because other branches will soon join in the process). Thus, any hypothetic additional storage equipment might soon require expansions (with further costs).
It must also be noted that in such case proper procedures should be put in place and resources used in order to maintain the additional storage system (this too with an economic impact).

A more critical issue is posed by those requirements that entail the ability to *reproduce* financial reports upon request (e.g. when auditors or tax authorities need them). This means that data must be put back into the ERP system in which the programs and all the setup needed to generate the requested financial reports are stored. This is not achievable due to constraints on the database. Such constraints prevent inserting data that do not satisfy a number of coded business rules. For instance, when trying to reinsert an invoice that is older than those already present in the system, database triggers would rise an error and not proceed with the insertion. In such cases, reinsertion should be done by importing a copy of the whole table (previously saved) and integrating it with the data added in the original table after the copy.
At the same time, it is not even possible to move programs that generate the financial reports somewhere else or to make them point to other sources of data (that is, the archived data). This is because not all of the source code is available for public scrutiny and also because changes to such programs are not allowed.

Thus, such option would not satisfy requirements implying the ability to reproduce financial reports. Of course, this is not acceptable.

These considerations highlight a sort of conflicting interest. On one hand, there is the need of improving performance and manageability of the ERP system and its server machine by *"getting rid"* of historical data. On the other hand, there is the need of *keeping* these historical data for data warehousing activities and fiscal reporting purposes.

The situation gets further complicated when dealing with deletion of data. As mentioned, in this context *archiving* data means that financial historical data are firstly *copied* in a suitable destination, and then, *deleted* from the original location. Since the deletion operation occurs in the production environment, special attention must be paid. Failing to carry out accurate deletion operations may cause severe data loss and make the ERP modules work improperly.

6.2 Adopted solution approach

As seen in the previous section, the process of devising a suitable solution is rather constrained by a number of requirements that impose using only the resources available. No new software technologies can be adopted, no special hardware can be acquired, and so on.

These restrictions imply that a viable solution can be found only if careful analysis are carried out on the actual information system and its available resources. In other terms, investigations are needed in order to become fully aware of system characteristics, and to define the features that could be exploited for the given objectives.

In particular, by studying the system in use it was possible to identify the key features on which the financial archiving system was based at a later stage.
Before focusing on the system architecture envisaged, it is essential to introduce these key aspects.

6.2.1 Tablespaces (Read-Only and Transportable features)

Tablespaces represent logical structures that divide an Oracle database into one or more *logical storage units* by grouping together related database objects. In turn, tablespaces are divided into one or more *datafiles* that allow physical storage. Figure 6.1 illustrates this relationship (Cyran, 2002).

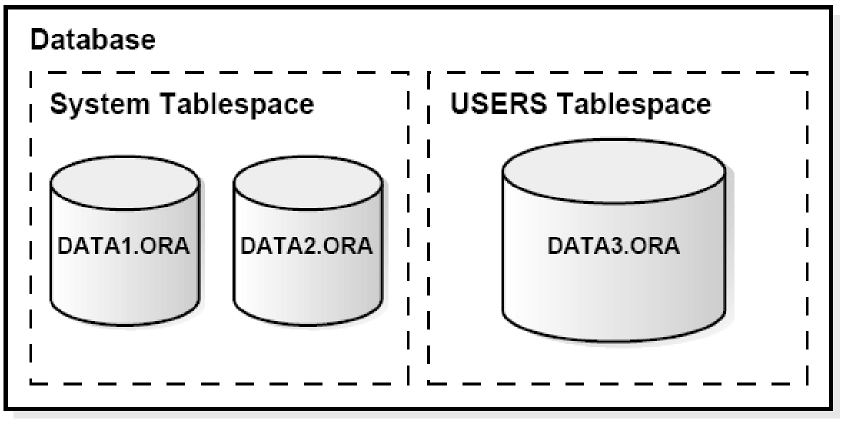


Figure 6.1 Datafiles and tablespaces

Figure 6.2 (Cyran, 2002) further highlights the fact that tablespaces may be stored on more than one datafiles (while a datafile can be associated with one tablespace only) and shows that database objects may span several datafiles.

At the moment of writing, the database version on which The Company's ERP system is based is Oracle's latest version on the market[4] and introduces a significant revision on tablespaces, solving known issues and providing important enhancements on this topic.
One of the interesting features of Oracle tablespaces is the possibility of making them *"read-only"*. When a tablespace is turned into a "read-only tablespace", data contained in it cannot be modified.

[4] To be precise, a newer version (Oracle 10g) has recently been announced by Oracle corporation but it is still in the process of entering the market.

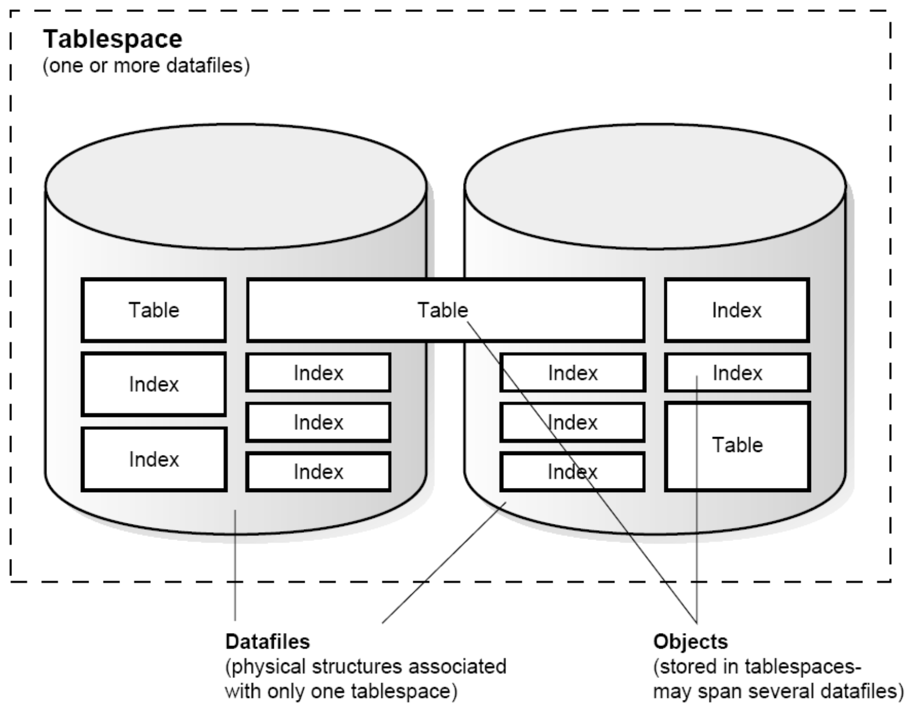


Figure 6.2 Datafiles, tablespaces and database objects

For the financial archiving system, having historical data in read-only tablespaces would prevent changes on data, regardless of user's update privileges. Thus, the requirement stating that historical data has to be kept in its original form, could be enforced by enabling such feature.

A more important benefit derived by the adoption of read-only tablespaces is related to *backup* activities. Typically, backup and recovery represent critical and important activities in the management of a database system. The impact of such tasks is usually closely related to the volume of the database.

By placing historical data into read-only tablespaces, the datafiles they consist of only need to be backed-up once, and thus they can be removed from regular maintenance activities. This means that, if it is known in advance that a large part of data will never change (as is the case here), the gain in terms of manageability is remarkable. Regular backup activities will only need to be done on a small fraction of the data (that is, on the online data).

Another interesting feature of Oracle tablespaces is that they can be made *"transportable"*. This allows dropping a tablespace from its database and importing it to another one (and if needed, importing into the original database again). This way, all tables and their associated indexes residing on a transportable tablespace could easily be moved into another database. Such feature would greatly help in building a historical database and avoid using the typical procedures commonly considered intricate and time-consuming (e.g. by using *export* and *import* files).

6.2.2 Partitioning

Manageability and performance can be further improved by means of the *partition mechanism*. Partitioning is meant to be used when dealing with very large database objects. It allows to break data down into smaller, more manageable parts called partitions (the same applies to indexes).

With the recent database versions, table's data can be partitioned according to several criterions. In this project such possibility could be used to structure archived data according to their *logical* separation. For instance, archived data could be partitioned, say, by business unit. This way, when querying a table, the internal database optimiser will not consider those partitions that do not actually need to be accessed for the given query, and thus improve execution times significantly. Figure 6.3 gives a graphical representation of the partitioning concept. On the left, is represented a non-partitioned table storing data belonging to all business units. On the right, the same table has been partitioned so as to keep data related to a specific business unit in its own partition. From a user perspective, that table is still seen as whole.

Another interesting aspect is represented by the fact that each partition is managed separately and can function independently of the other partitions.

Moreover, if combined with tablespaces, the use of partitions, can provide further improvements in terms of performance and flexibility. In fact, having each partition assigned to a separate tablespace allows the various partitions to be distributed across multiple disks. This way, a better use of storage resources can be achieved (e.g. frequently accessed partitions could be stored on faster disks).

Finally, if parallel execution option is enabled on the database (as is the case in The Company), partitioning allows further parallelization. In fact, partitioned tables and indexes can be accessed in parallel by assigning each of them to different parallel execution managers.

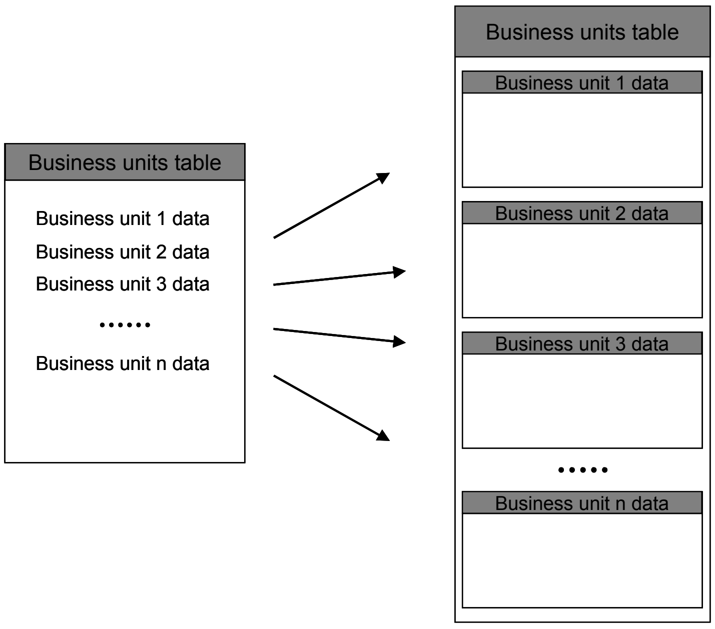


Figure 6.3 Table partitioning

6.2.3 Oracle's standard routines

As part of risk assessment activities, a number of critical aspects have been identified. They essentially stem from the deletion operations that have to be carried out when performing the archive process. As already pointed out, deletion operations carried out on production environments are inherently critical (e.g. due to difficulties in reversing the operation). Erroneous deletion operations may easily cause malfunctioning of the ERP system. However, there are more severe aspects related to deletion operations.

The first critical aspect that has been identified refers to the risk of compromising ERP *vendor's support*. In fact, Oracle recommends that data removal from their ERP system is carried out only by means of its standard ERP features, using the "Purge" feature provided with each financial module. Deleting data in any other way may prejudice Oracle support on the system. Oracle actually provides a "Purge" feature combined with an "Archive" feature for the main financial modules. At first sight, it could seem that these standard features are well suited for the purpose of the project. Unfortunately, as presently described, this is not true.

Recently Oracle designed a number of routines with the intention of assisting large enterprises in keeping their databases at a reasonable size and thus helping them to achieve better performance and manageability. To some extent, this may represent the goal of the present project. However, a closer look at these Oracle features reveals that they do not solve some of the important issues of this project. Basically, these Oracle routines are designed to remove data by exporting them into files. This way, a database actually has part of its data removed, allowing some of the objectives of this project to be achieved. One of the drawback is that the data removed only end up in files in the operating system. Such files are not accessible unless brought back into an identical environment. This means that the ability to reproduce fiscal reports cannot be readily satisfied. The same applies to data warehousing activities. The process of extracting data from the above files is somewhat time-consuming and not automated. In fact, the reverse process is meant to be performed only in particular situations and not on a regular basis.

In any case, a more critical problem prevents the use of Oracle's standard archive and purge. Tests and analyses on these features in fact revealed that for some of the ERP modules there is a *data aggregation* when generating the export files (see Appendix C for further details). This means that there is a sort of data loss, and this would prevent accessing archived data even if the reverse process were performed in an identical environment. This is not the result of anomalous behaviour. These features rather tend to archive only *key* information.

In The Company's context, such behaviour has been considered unsuitable. Failing to make exactly each part of the historical data available would not permit correct reproduction of fiscal reports and would not allow carrying out all the desired data warehousing activities.

Although Oracle routines do not fully fit The Company's requirements, it would make sense to investigate the possibility of integrating part of these routines into the financial archiving

system being developed. An eligible candidate is Oracle's "Purge" element. Investigations should firstly be addressed to such feature because with its use no custom deletion operations would be performed and thus Oracle's support (which plays a significant role in companies such as the one considered here) would not be compromised.
Some thought should also be given to maintainability of the system. In fact, coding a specific deletion policy in the financial archiving system would require manual intervention whenever a change in the ERP data model occurs[5]. Therefore, basing the financial archiving system on the standard "Purge" element means that the underlying programs incorporate Oracle's own deletion algorithms which, most importantly, are kept in line with ERP patching and upgrades.

The considerations described in this chapter lays the basis for the design stage. Design aspects along with further considerations are discussed in the next chapter where the architecture of the financial archiving system is presented.

[5] Coding a logic that in an automated way would cope with all possible future changes in the data model do not represent a realistic option.

Chapter 7

Design

This chapter focuses on the design part of the financial archiving system. It starts by giving an outline of the overall architecture, then four sections covering the following topics are provided:

- Concepts on Oracle's standard archive and purge routines.
- Integration of the above routines in the archiving program.
- Description of the archiving program.
- Description of the archive schema.

7.1 Overview

As specified by requirements (see requirements Req/NF/01 - Req/NF/05 in section 5.3.4), the financial archiving system must be structured in such a way to provide a tight integration with the ERP system currently in use. It has to represent a kind of "supplementary feature" of the ERP system. In other terms, the financial archiving system has to "reside" within the scope of the ERP system and enhance its capabilities (see Figure 7.1).

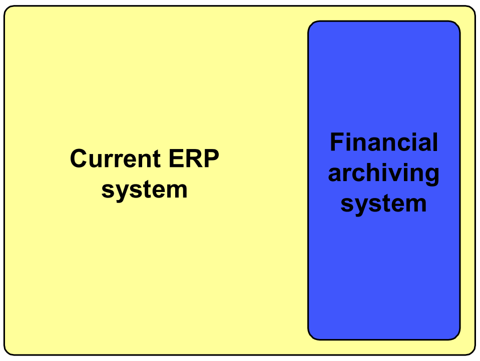


Figure 7.1 The financial archiving system as part of the current ERP system

The architecture of the financial archiving system consists of two main entities (see Figure 7.2):

- One that relates to the data model level and reproduces the structure of database objects needed to store data eligible for archiving (in the sequel “archive schema”).
- The other, that takes care of the activities needed to move historical data in the above archive schema. It is represented by a program that implements the business logic and performs the whole archiving process.

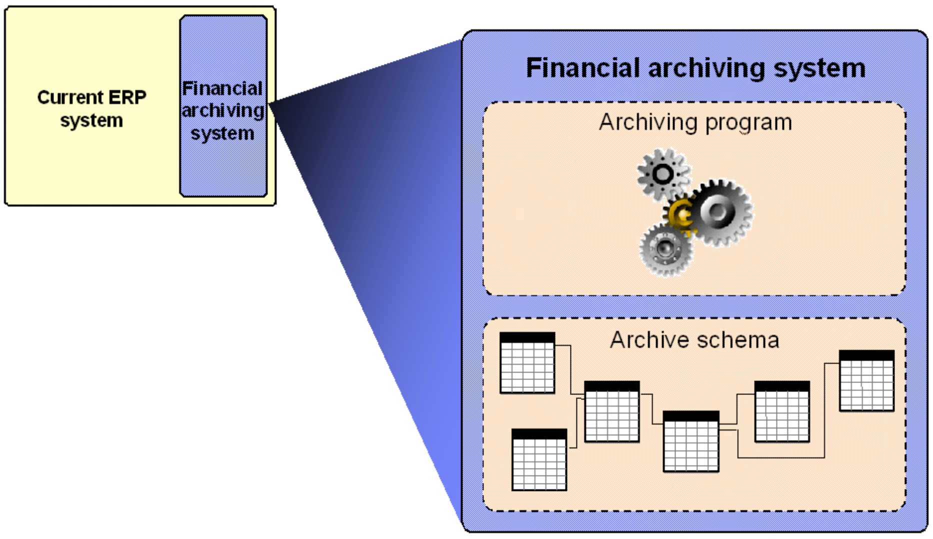


Figure 7.2 Structure of the financial archiving system

7.2 Oracle's standard routines

At the end of the previous chapter it has been mentioned that the ERP system currently in use by The Company, already provides built-in routines for archiving data. These routines (different for each ERP financial module) are meant to accomplish each a specific task in the archiving process. More precisely, Oracle's archiving procedures involve copying data into temporary tables, then moving them into data files in the operating system and finally delete them from the original location.
Unfortunately, such routines cannot be used by themselves to solve the problem addressed here. As explained in section 6.2.3, they fit only partially The Company's requirements.
However, investigations conducted on the characteristics of these routines revealed that several benefits could be achieved by their integration into the financial archiving program.
Results of such investigations are summarised below where it is provided a description of the routines for copying data (referred to as *"archive"* routines) and those for removing data from its original location (referred to as *"purge"* routines)[6]. Other considerations on the investigation conducted are then provided in section 7.3 where it is discussed the integration of the above routines in the archiving program.

7.2.1 Archive routines

Oracle's standard *archive* routines are intended to provide a copy of data that satisfies a number of archiving criterions. Such copy is normally placed in a set of temporary tables.
These routines are discrete by module (though, in few cases are shared, e.g. for Accounts Payables and Purchasing). Thus, they operate independently, and indeed are launched by an ERP application responsibility which has access only to data related to that module (and to a specific business unit within the module).

Oracle's standard archive routines operate by performing the following two steps:

[6] Despite such archive/purge distinction, the whole process (including purging) is usually referred to as *archive* process.

Data selection Financial data are selected according to a number of criterions. Part of these criterions are specified by the user (by means of input parameters). For example, the user that requests the execution of one of these routines, can specify the accounting timeframe into which a transaction has to fall in. Other criterions represent fundamental business rules that a given transaction has to satisfy as a minimum requirement in order to be removed. For example, for a given record, the grounding business rules firstly require that the "accountancy" status (for that record) is "closed", then that all possible relations with other records will not be compromised by the deletion and finally that the criterions specified by means of input parameters are all satisfied.

Data copying Data selected by means of the previous step is then copied into ad-hoc temporary tables. As an example, the archive routines provided in the *General Ledger* (GL) module, perform a copy as illustrated in Figure 7.3. It essentially involves the four GL tables in which are stored account balances, journal batches, journal entry headers and lines[7].

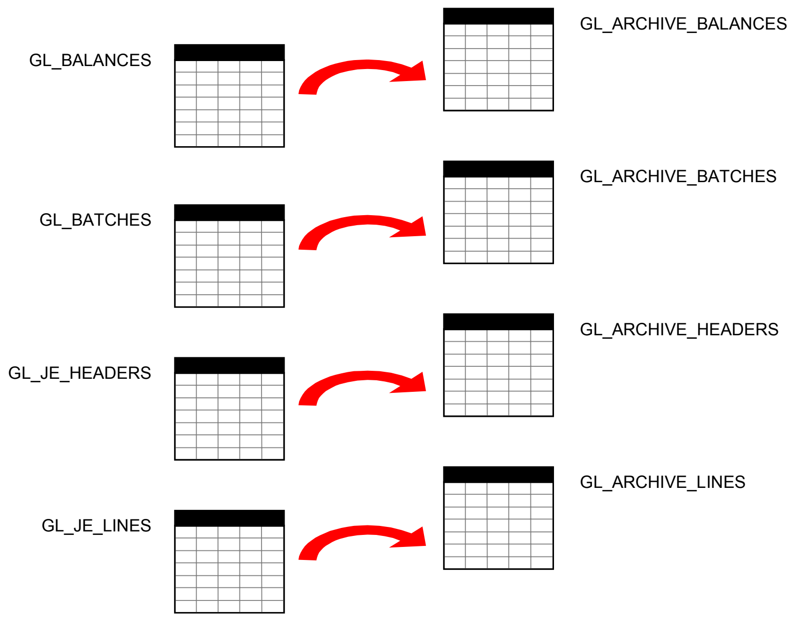


Figure 7.3 Data eligible for archiving is copied into temporary tables

[7] More precisely, some other tables are also involved in the process. However, they do not store financial data. They are used for control purposes (e.g. they keep track of executions and other related information) and are not relevant in this specific discussion.

It is to be noted that in the above example, the process of copying data into the archive tables occurs in a one-to-one fashion. In other modules a many-to-one copy fashion is adopted. In other terms, there is a sort of data aggregation. For example, the Accounts Receivables module is provided with archiving routines that aggregate into three tables data coming from twenty-one of them (see Figure 7.4).

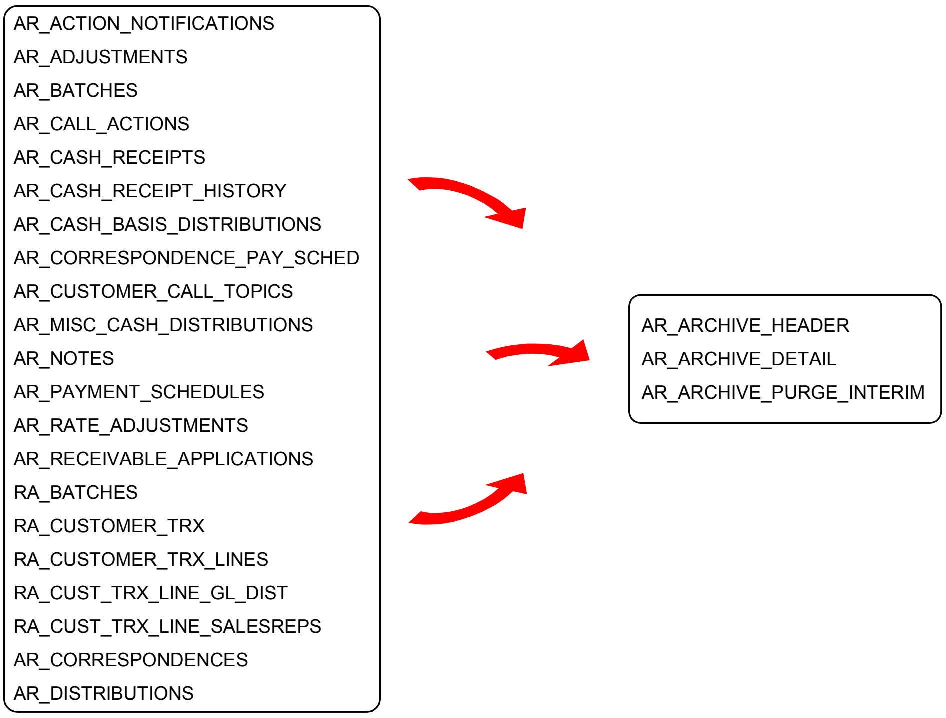


Figure 7.4 Data aggregation in Accounts Receivables

In such cases not all of the information of which a record consists of are actually copied into the archive tables (but only information that has been considered fundamental by the ERP producer). As mentioned in section 6.2.3, this is inherently "lossy" and represents an issue with respect to The Company's requirements. This aspect will be recalled shortly in section 7.3 when discussing the integration of these routines in the archiving program.

7.2.2 Purge routines

Oracle’s *purge* routines are designed to remove data from the ERP standard tables. Data affected by such deletion operations are those previously selected by the corresponding archive routines.
Oracle’s standard purge programs are discrete by module the same way standard archive routines are.

As previously noted, by adopting such routines no custom deletion operations would be performed and thus Oracle’s support on the ERP system would not be compromised. Also, this way maintainability of the financial archiving system would benefit since these purge routines would be kept in line with ERP data model changes, patching and upgrades (by Oracle).

7.3 Integration of Oracle’s routines

The design structure on which the archiving program has been based, entails firstly the adoption of Oracle’s standard *archive* routines.
With regards to these archive routines, it is worth to recall that they essentially copy data eligible for archiving in a set of temporary tables which, in turn, are placed in a data file at a later stage (by means of other specific routines). It is also worth to call attention to the data-loss issue caused by the data aggregation performed while copying data into temporary tables. This does not allow Oracle’s standard archive routines to be used by themselves for the purposes of the present project. Though, they can provide a considerable help.
In fact, archive routines encapsulate the business logic for selecting financial transactions eligible for archiving. Such business logic fits The Company’s requirements, in that it covers all kind of financial data produced by The Company’s accountancy department and provides a higher “granularity” of selection parameters if compared to The Company users’ needs. Thus, the adoption of these archive routines would considerably support in having The Company’s business logic encapsulated in the archiving system. Moreover, since there is a particular interest in adopting Oracle’s standard *purge* routines for carrying out deletion operations, the

combined adoption of Oracle's standard *archive* routines would result valuable in that it would guarantee in-sync *data selection* and *data deletion*. That is, the set of records selected by archive routines would be guaranteed to match exactly the set of records deleted by purge routines, and thus, related potential side effects would be avoided.
Also in this case, maintenance would significantly benefit since archive routines would be kept up-to-date with all relevant changes in the ERP system.

The issue to be solved is related to the *data aggregation* performed when copying data into temporary tables. Such data aggregation causes a data-loss that is not acceptable with respect to The Company's requirements[8]. Since data to be archived is meant to be moved in an "archive" schema, such data aggregation do not allow to readily copy data (as it is) from temporary tables into tables in the archive schema (which represents a copy of the standard ERP schema). That is, for a given record in one of the temporary tables, it is necessary to get to the exact source record(s) from which the given record in the temporary table has been generated. Then source record(s) has to be copied into the corresponding table(s) (in the archive schema) as it is.
The idea adopted for solving this issue takes its origin from analysis conducted on the behaviour of Oracle's purge routines. Investigations revealed that purge routines identify records to be deleted by first accessing the temporary tables. In fact, it has been found that each record in the temporary tables contains univocal IDs that allow to go back to the original "source" records. Purge routines then indeed access these temporary tables in order to go back to all records that have to be deleted.
Therefore, in those cases in which a data aggregation is performed, the direct copy (from temporary tables to archive ones) has been replaced with a mechanism that resembles the purge routines behaviour. That is, "insert" statements have been derived from the "delete" statements included in purge routines. A basic example of such transformation is shown in the following two figures, where a simple delete statement (Figure 7.5) and its transformation into an insert statement (Figure 7.6) are illustrated.

[8] In Appendix C are enclosed details on such data-loss.

As can be seen in Figure 7.5, rows are deleted from the standard database view *"ar_receivable_applications"*[9] of the financial module *"Accounts Receivables"*. Here, a record is removed if its *"cash_receipt_id"* is present in the table *"ar_archive_purge_interim"*, which in turn represents one of the temporary tables used by archive routines for storing data eligible for archiving. This way, purge routines delete records from standard tables only if they have been previously considered eligible for archiving (and placed into temporary tables) by archiving routines.

```
DELETE FROM ar_receivable_applications
WHERE cash_receipt_id IN (
                          SELECT related_id
                            FROM ar_archive_purge_interim
                           WHERE ...
                         )
```

Figure 7.5 Sample delete statement used
in the purge routines

Observing now Figure 7.6, the core part of the above delete statement can be found in the insert statement too(compare highlighted rows in the two statements). Such statement can be commented as follows:

- Records are selected from the standard table *"ar_receivable_applications_all"*.
- Records selected are then inserted in an identical table but residing in the archive schema (see prefix *"op_arch"* in row 1). Such table is meant to contain only archived data.
- A record is selected (and then inserted) if its "cash_receipt_id" is present in the temporary table *"ar_archive_purge_interim"* (with the same meaning described for the delete statement).
- Records selection actually consists of another intermediate "layer" represented by the inner select statement (see rows 5 to 14). This instruction is executed on the database

[9] Such view is based on the table *"ar_receivable_applications_all"*. Data selected are those belonging to a specific business unit. Therefore, the underlying table contains data for all business units. This is part of the "multi organization" architecture of the ERP system.

view *"ar_receivable_applications"*, which, as noted, contains only data belonging to a specific business unit[10]. Such select statement makes also use of set operators in order to prevent duplicate insertion of data into the archive schema (see rows 12 to 14).

```
 1   INSERT INTO op_arch.ar_receivable_applications_all
 2   SELECT *
 3     FROM ar_receivable_applications_all
 4    WHERE rowid IN (
 5                    SELECT rowid
 6                      FROM ar_receivable_applications
 7                     WHERE cash_receipt_id IN (
 8                                                SELECT related_id
 9                                                  FROM ar_archive_purge_interim
10                                                 WHERE ...
11                                               )
12                     MINUS
13                    SELECT orig_rowid
14                      FROM op_arch.ar_receivable_applications_all
15                   )
```

Figure 7.6 Sample insert statement derived from delete statements

The above example has been intentionally simplified in order to facilitate understanding of the statement transformation. For example the insert statement do not specify *source* and *destination* columns (row 1 and 2 in Figure 7.6). This is an intended simplification for the purposes of the above description. The actual code contains a specific function which dynamically retrieves an ordered list of all columns of the needed table[11]. Specification of source and destination fields in other general ways (e.g. by means of the asterisk) is not fully reliable and is not considered a good practice (also see note on SQL statements in section 8.1). This way, besides adhering recommended development standards, reliability is enhanced. This function has been written also for maintenance reasons. In fact, with no hard coded column

[10] All this code is placed in a loop cycle which is executed according to all business units involved. Thus, in the described statement, the specific business unit considered changes at every iteration of the loop cycle.
[11] This is done by accessing data dictionary tables which reflect the current tables definition.

names, changes to tables definition would not pose any issue. Another reason for such function derives from SQL tuning activity carried out for performance enhancements. As can be observed in the actual source code, whenever appropriate SQL statements are expressed adopting *dynamic SQL*. This can be seen in those instructions beginning with the reserved words *"execute immediate…"*.

The actual code can be examined in the Appendix A.

7.4 Archiving program

In the previous sections, archive and purge routines have been discussed along with their relationship with the archiving program.
In light of these discussions, the initial diagram presented in Figure 7.1 can be revised as depicted in Figure 7.7, where invocation of Oracle's standard archive and purge routines is shown.

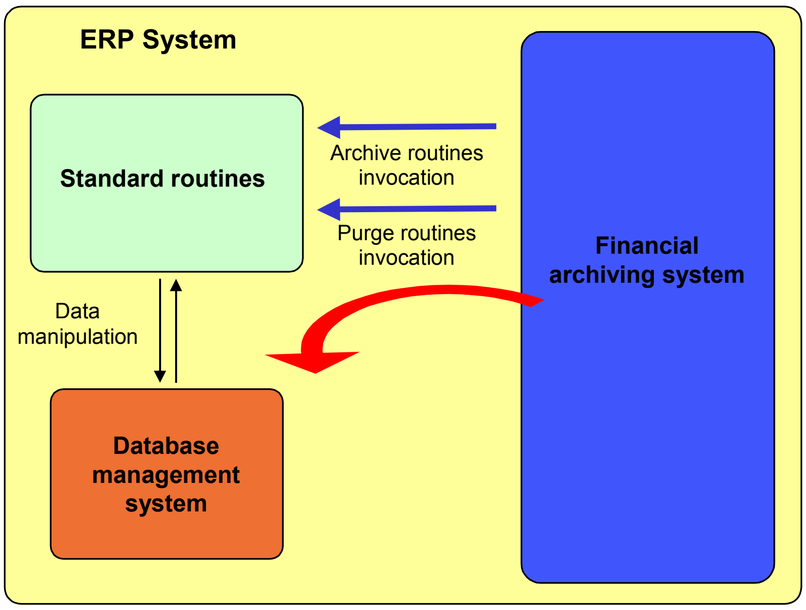

ERP System
Standard routines
Archive routines invocation
Purge routines invocation
Financial archiving system
Data manipulation
Database management system

Figure 7.7 Overall structure - revisited

It has also been discussed the issue of data aggregation posed by standard archive routines. As described, one way of tackling this difficulty is to adopt a mechanism similar to the one used in purge routines for identifying records to be deleted (in this case to be copied). Refinements to Figure 7.7 can be added in order to highlight the process of copying data eligible for archiving into an ad-hoc archive schema. This is shown in Figure 7.8 where it is also indicated the sequence in which routines invocation takes place (see circled numbers).

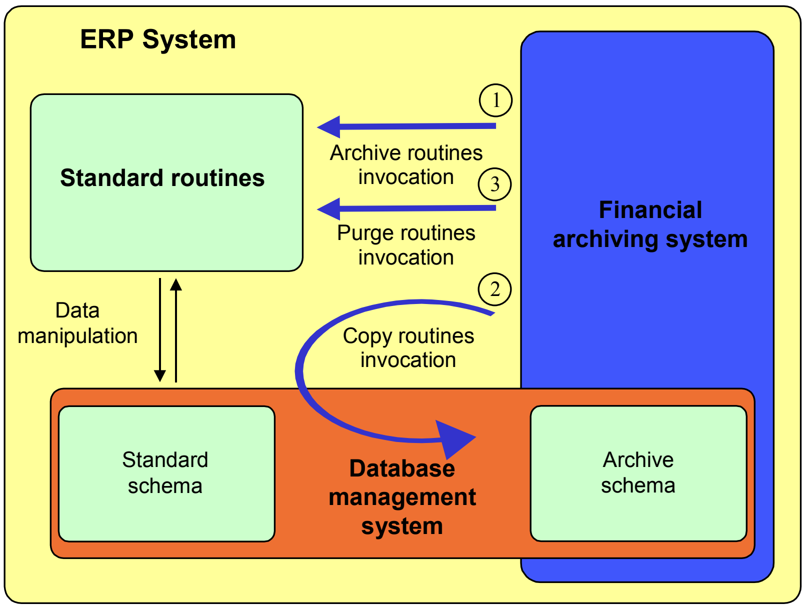

ERP System
Standard routines
1
Archive routines invocation
3
Purge routines invocation
Financial archiving system
Data manipulation
2
Copy routines invocation
Standard schema
Database management system
Archive schema

Figure 7.8 Copy process in the overall structure

Now, a closer look to the algorithmic aspects of the archiving program can be taken. In particular, the program behaviour can be illustrated in Figure 7.9 by means of pseudo-code. As can be observed, the whole logic is encapsulated in the main procedure named *"ArchiveAndPurge"*. Execution starts by performing a number of initialisation steps followed by several checks. Then two main nested cycles are executed:

– The outer one involves an iteration per each business unit. More precisely, if a business unit is divided down into sub-entities, then additional iterations are required (one for each of sub-entity).

– The inner one requires an iteration per each accounting period in the range specified by the user. For example, if the range specified is *"from 01-1996 to 06-1996"*, then six iterations will be required. Here, for each iteration, period-specific checks are first performed, then data eligible for archiving is identified by executing the archive routine (relevant for the financial module being processed). After that, a copy of all data identified is performed so as to make them available in the archive schema. Finally, the same set of data is deleted from tables in the standard schema (by invoking the relevant purge routine) and temporary tables are cleaned up.

```
procedure ArchiveAndPurge(…){

   Initialisations;
   Checks;

   for each business_unit{
      module-specific checks;
      for each period_in_range{
         period-specific checks;
         identify eligible data (by means of standard archive routine)
         copy data into archive tables
         delete data (by means of standard purge routine)
         clean up temporary tables
      }
   }

}// End ArchiveAndPurge
```

Figure 7.9 Archiving program - code structure

All procedures developed for the financial archiving system have been included in a package, which in turn has been installed into the database. The above main procedure, has then been included into the standard programs/reports list used in the ERP system and made available for execution to the designed key user. This task is illustrated in Appendix A (section A.1).

7.5 Archive schema

The archiving program described in the previous sections works essentially by "moving" data from one "location" to another. Technically, these locations represent the two following database schemas:

Standard schema	Acting as data *"source"* (also referred to as "apps" schema). It contains all ERP data and represents indeed the standard schema provided with Oracle's ERP system.
Archive schema	Acting as data *"recipient"* for historical data (also referred to as "parallel" schema).

The archive schema corresponds to the other main component of the financial archiving system. Its aim is to store data that has been considered as "historical" and thus not necessary for ordinary day-to-day accounting activities.
More in detail, it is a database schema containing a copy of all database objects (initially empty), residing in the standard ERP schema, that are used in the financial modules covered by the implementation (see Figure 7.10).

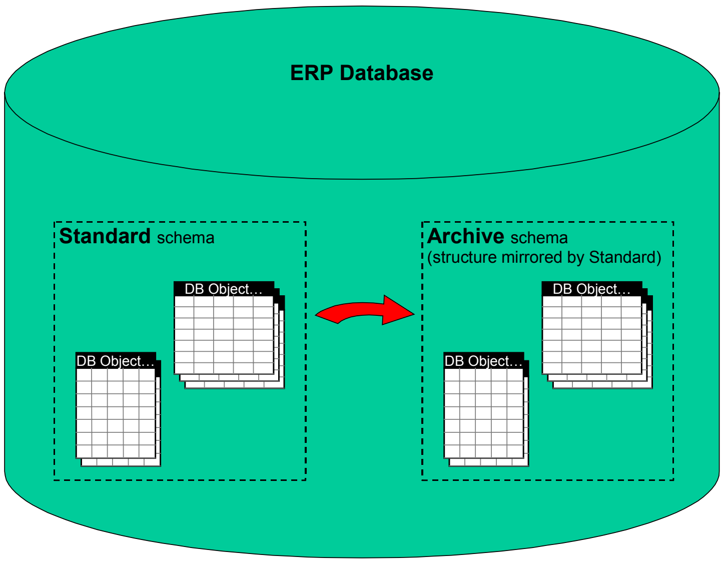


Figure 7.10 Archive schema

Such archive schema is created into the ERP database but its design allows data to be actually stored in other locations. In fact, as mentioned when describing the *"transportable"* and *"partitioning"* features (section 6.2), the adopted design approach makes it possible to store data into other remote databases. Most importantly, it also allows to enforce a "data distribution" policy at hardware disks level. That is to say, data can be distributed across multiple disks in order to provide a better utilization of storage resources. Cheaper disks, already available and mostly unused, can be utilised allowing, in this way, to free up expensive disk storage for production data.
The reasons why the archive schema was created into the ERP database are connected to the need of having archived data transparently available to the ERP application (which only "sees" its underlying database). As discussed, there must be for example the ability to reproduce fiscal reports upon request and this can only be done through the standard ERP application components and forms.

The set up of the archive schema required carrying out several tasks. The result is available in Appendix A (sectionA.2) where four installation scripts are shown in detail.
The following subsections are then aimed at describing each of these set up activities.

7.5.1 Archive database user

The first task required the creation of the "archive" database user ("OP_ARCH"). Such database user had to be granted a number of permissions (e.g. the permissions needed for creating tables, views, and so on). More precisely, the archive database user had to be granted all permissions granted to the database user "APPS" which is the user belonging to the standard schema. In fact, having the archive schema to be a kind of parallel copy (in terms of structure) of the standard schema, the archive user has to be able to access all objects accessed by the user "APPS". Further details are provided in Appendix A (section A.2.1).

7.5.2 Configuration and control tables

The second task required the creation of a set of configuration and control tables needed for the archiving program. More in detail, the following tables had to be created[12]:

Configuration table This table stores information related to the set of tables involved in the archiving process (e.g. tables belonging to the financial module *General Ledger*). This table is used by the archiving program in order to process all needed tables.

Archive_history table This "history" table was created for storing information regarding each execution of the archiving program. It is used in the archiving program by the main procedure in order to avoid re-archiving a period.

Configuration_partitions table This table stores information regarding partitioning. According to the specific users' needs, it is possible to specify how to partition tables. Thus, in this table is specified the name of the table to be partitioned, the column and its corresponding condition to be evaluated in order to group among the different partitions, and so on.

Configuration_responsibilities table This table stores information related to the business units involved in the archiving process. Business units (or sub-business units) are specified in this table by a corresponding responsibility profile. Thus, this table contains a list of responsibilities that is examined by the archiving program in order to perform archiving for each business unit.

Archive_exceptions table This table stores information regarding data that cannot be archived successfully. This is mainly used by the financial module Accounts Receivables in order to identify accounting transactions with errors. Thus, for example, the function "AR_Preview" uses this table for such purpose and do not allow

[12] The exact table definitions can be found in the first installation scrip in appendix Appendix A (section A.2.1).

execution of archiving (for the business unit and period being processed) if errors have been found. These errors essentially corresponds to "accounting conditions" not satisfied for specific accounting transactions. Thus, in such table are stored the information on the data causing the error (table and row ID) and a description of the error.

The instructions by means of which the archive database user and all the above tables were created are collected up in the first installation script (of four) enclosed in Appendix A (section A.2).

7.5.3 Archive tables

The next stage of the archive schema set up, involved the creation of all archive tables into which historical data had to be moved. Each of these tables had to be a kind of copy (initially empty) of its corresponding standard table residing in the standard schema. For example, a copy of the standard table "GL_BALANCES" (owned by the database user "APPS") had to be done by creating a corresponding table "GL_BALANCES" in the archive schema (and thus owned by the archive database user "OP_ARCH").
However, it must be noted that the two tables differ from each other in a column. Precisely, all archive tables contain an additional column used for storing the original row ID of the record initially stored in the standard schema. This was done in order to avoid duplications of data in the archive schema and in order to allow a complete traceability of data.
During this stage, all tables, indexes and so on, were created referencing the configuration table above described. For further details see the second installation script in section A.2.2.

7.5.4 Union views

An important aspect worth noting regards the archive tables discussed above. Typically, data stored in an ERP database can be classified into the following two classes:

- *Transaction data*
- *Standing data*

Transaction data refers to *events* representing financial operations such as a purchase or trade, while standing data represents somewhat "static" data that describes *entities* such as a vendors list, a banks list, and so on. The former, being subject to considerable changes, generates a number of records in the database that change over time. Figure 7.11 gives an idea of such aspect by illustrating a typical accounting process involving a number of events, each recorded in the database (James D. *et al*, 2002).

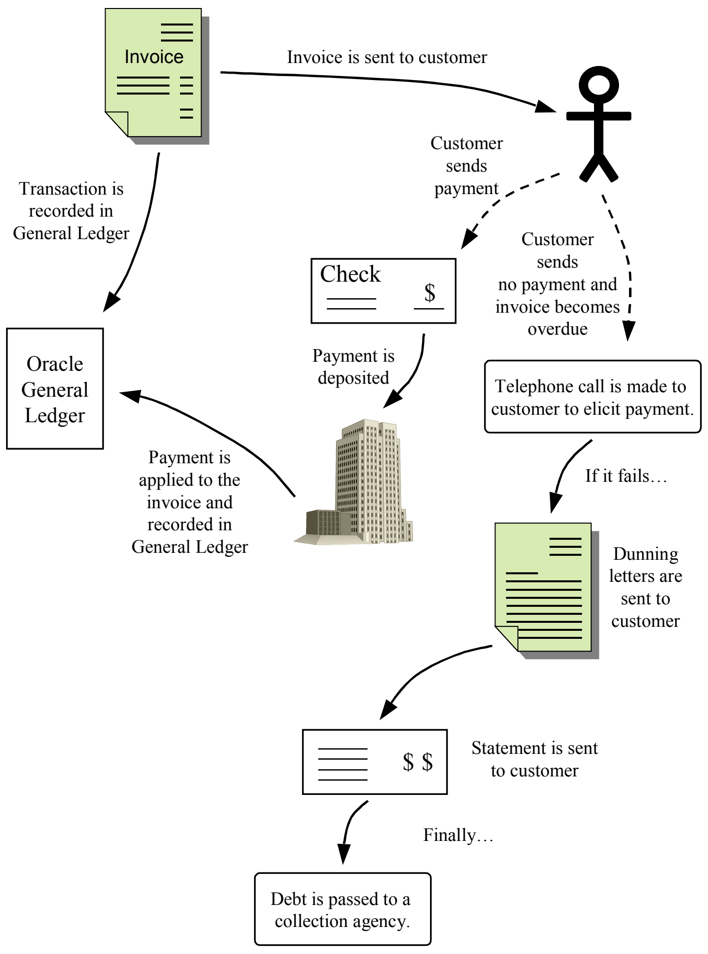

Invoice
Invoice is sent to customer
Customer sends payment
Transaction is recorded in General Ledger
Customer sends no payment and invoice becomes overdue
Check
$
Oracle General Ledger
Payment is deposited
Telephone call is made to customer to elicit payment.
Payment is applied to the invoice and recorded in General Ledger
If it fails…
Dunning letters are sent to customer
$ $
Statement is sent to customer
Finally…
Debt is passed to a collection agency.

Figure 7.11 Overview of a typical Oracle Receivables process

Conversely, standing data does not heavily change over time and is stored (for each group) in a relatively simple data model (also see glossary terms in chapter 2 for further details on *"standing data"* and *"transaction data"*).

Archiving *all* historical data is of course necessary. However, it is not sufficient. A critical problem emerges when carrying out data warehousing activities or reproducing (fiscal) reports. In fact, when printing a report based on historical data, online data (stored in the standard schema) may be needed as well. Just as an example, one can think of an old purchase invoice (which is part of the transaction data). As expected, such invoice refers to a vendor (which is part of the standing data) and when viewing or printing such archived invoice, the information presented might easily be incomplete because vendor information are not stored in the archive schema – they are stored in the standard schema instead, because the given vendor is still "active" and thus not eligible for archiving.

The same issue may arise when viewing online data, say an invoice again, and the referenced vendor has been disable and archived (i.e. it has been moved in the archive schema).

For those tables containing standing data, one way of tackling this issue was to (a) create database views that merged the content of standard tables with the content of the corresponding archive tables; (b) modify the database synonym used to access the standard table in order to make it point to the newly created database view.

For these reasons, in the above configuration table, a column was defined specifically for this purpose, indicating whether or not a table had to be part of a database view. Such information was later used in the second installation script for creating such views and changing the synonym indeed (see section A.2.2).

7.5.5 Application components

Once the main "archive" database objects had been created, it was necessary to create also a set of components aimed at manipulating the historical data. In fact, the standard database user "APPS", uses a number of procedures, functions and other components in order to operate with financial data. In other terms, all these components are basically needed to the upper application layer, in order to enable all functionalities related to each end-user application

responsibility. Since end-users of the financial archiving system were meant to access historical data through the same application interface, the same set of components had to be created also for the archive database user.
Such task was then performed by using a function, already available, that had been previously developed for maintenance purposes on a custom application installed into the ERP system. Such function had to be used after any kind of changes in the structure of the custom schema (e.g. recompiling invalid objects). In this context, this function was used specifying as custom schema the archive schema of the financial archiving system.
Once such operations had been carried out, the archive schema was completely created.
In Appendix A (section A.2.3) the script used for carrying out this task is shown.

7.5.6 Archive responsibility

As already noted, end-users were meant to access archived data through the standard application interface normally used for accessing online data. In order to do that all involved responsibility were "cloned" and associated to the archive schema. Each cloned responsibility had the name of its corresponding standard responsibility, prefaced by "ARCH". To take the financial module General Ledger as an example, the standard responsibility "GL Super User" became "ARCH GL Super User".
This way, when access to historical data is needed, users simply have to connect to the ERP system by choosing the responsibility (of the relevant financial module) prefaced by "ARCH". From this location, all components needed to carry out the usual tasks (printing reports, performing calculations, and so on) "point" to historical data.
Further details are provided in the Appendix A (section A.2.4) where the script for creating such archive responsibilities is enclosed.

7.6 Maintenance

Maintenance is undoubtedly one of the most important aspects of software engineering. A software system with good maintainability characteristics allows future (inevitable) changes, enhancements and error corrections to be integrated with a reasonable effort. In this context, the way the financial archiving system was designed took into account such maintenance aspects. For example, the mechanism through which deletion operations have been "delegated" to standard routines, reduces noticeably all maintenance tasks needed to keep the deletion algorithm in sync with the data model. The same applies to the mechanism adopted for selecting data eligible for archiving.

This way, maintenance mainly involves a smaller, "well-defined" part of the system, that is the part that essentially takes care of copying data into the archive schema. However, maintenance on this part should not be a major concern since all key operations are implemented *dynamically*, on the base of the actual data model information (i.e. by building statements at run time with information coming from data dictionaries tables).

Moreover, the *modular* structure of the system, allows other business units and financial modules to be integrated without requiring changes to the system architecture. At the moment of writing, in fact, several other modules are being included. Such modularity can be observed by recalling the program structure shown in Figure 7.9 and the related information provided in the previous sections of this chapter. As can be inferred, introducing, say, a new financial module, essentially requires:

- the invocation of selection/deletion routines (for the given module)
- the definition of the routine that copies eligible data into the archive schema
- the specification of the needed archive tables

Introducing a new business unit is even easier – it simply requires the specification of the business unit ID codes in the configuration tables.

Another point worth mentioning refers to those maintenance activities that have to be carried out after each execution of the archiving program.

As described, this program manipulates data stored into database tables. For example it deletes considerable amounts of data, and since memory space is not released automatically, it is necessary to consolidate table storage manually. In other terms, some database administration tasks are required (e.g. rebuilding indexes, moving tablespaces, setting up database links, and so on). However, such activities correspond to the *ordinary* database administration carried out regularly in every company using large IT systems (as is the case of the company considered here). Therefore, no impact in terms of maintenance is caused by the usage of the financial archiving system.

Therefore, summarising, the financial archiving system was developed taking into account the context into which it was meant to be used, and especially bearing in mind the fact that organizations evolve over time, and with them, users' needs as well.

Chapter 8

Implementation notes

This chapter discusses several aspects related to the implementation stage of the project. The following section gives some information on the development standards and recommendations involved in the implementation. Then, the other two subsequent sections describe the programming languages and tools used for developing the financial archiving system.

8.1 Development standards and recommendations

An number of recommendations and development standards were available both at The Company site and at Oraplus. The actual documents cannot be enclosed for confidential reasons. However, a brief description can be provided on those non-confidential advice that were more closely related to the development activities of this project. Such description, together with a comment on the student's personal approach can be provided by means of the following key points:

Code location

– When developing a customised application within a platform such as the ERP system discussed in this document, all custom code has to reside in a single location in the file system specifically dedicated to custom programs.

Naming conventions

- A naming convention has to be adopted in the source code. For example the scope of a variable can be included in its name by using an ad-hoc prefix. In the archiving program, global variables were prefaced with *"G_"*, while module parameters with *"P_"* and local variables with *"L_"*.
- For this project, it was recommended to use the following extensions for program files:

a.	Unix scripts	(.script)
b.	Package specification	(.pks)
c.	Package body	(.pkb)
d.	SQL*Plus scripts	(.sql)
e.	SQL*Loader scripts	(.ctl)
f.	Table creation scripts	(.tab)
g.	Synonym creation scripts	(.syn)
h.	Index creation scripts	(.ind)
i.	Test scripts	(.tst)

Functions

- When defining a function, a return statement should always be placed as the last executable statement.
- Multiple return points should be avoided.

Packages

- All related data structure and functionality should be grouped together in a single package.
- Package specification and package body should be defined in separate files. The specification part of a package do not change frequently over time and is used to resolve all references to other elements. When a package specification is recompiled, all dependent objects will be invalidated. By defining the package body in its own file, it is possible to change and recompile the main part of program (body) without affecting the status of any other object.

SQL statements

- Insert statements should define the column list. In fact, if it is omitted, the insertion operation will depend on the order of the columns in the table, which may be different in the various environments or subject to changes in successive product releases.

Loop constructs

- RETURN and EXIT statements should not be used inside a *loop* or a *for* construct in order to avoid unstructured termination. Boundary conditions should be used, instead.
- Loop constructs should not contain code statements that do not change over the iterations. Such statements should be place before, outside the loop construct.

8.1.1 Error handling

As noted, the above list do not consider general guidelines such as indentation, code comments and so on. It mentions only those advice that in some way were closely related to the scope of this project and to its technical field (i.e. Oracle databases). However, another topic that is worth mentioning is related to *error handling*. The personal impression was that such aspect could have been covered in more detail by the development recommendations. Providing a suitable error handling mechanism allows the definition of all admissible "ending status" of the program (that is, the ordinary one and the "exceptional" one). This way, a program terminates always in a known status and unexpected results are less likely to occur. Besides handling all "exceptional" program behaviours, a proper error handling may provide a significant support during the development and maintenance stages. For this project, in fact, the definition of the error handling mechanism, was driven by such aspects as well. The underlying idea was that *error location* had to be emphasized. In other terms, since the dimension of the program was not negligible, printing out a mere Oracle error message with no information on the error location, would have been a poor strategy. In the program language used, PL/SQL, error handlers are introduced in a PL/SQL block (e.g. function or procedure) as shown in Figure 8.1.

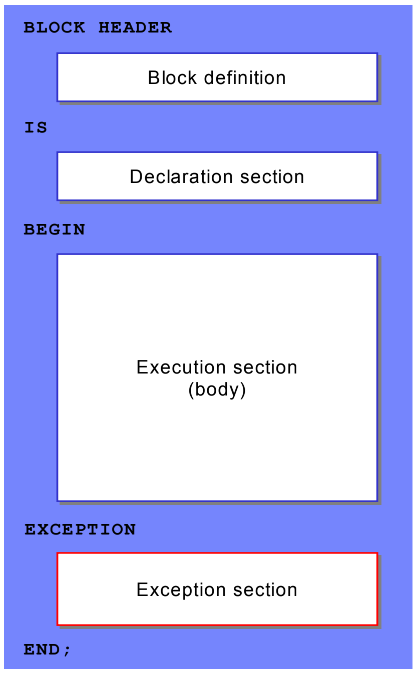


Figure 8.1 Block structure in PL/SQL

When an exception occurs in such a block, the execution terminates and control is passed to the exception section. As in many programming languages, exceptions can be trapped by a specific error handler. That is, a number of error handlers can be defined in the exception section in order to take different actions for different kind of exceptions (see Figure 8.2). Of course, in order to trap exceptions properly, handlers for more specific exceptions should come first. Also, if one wants to be sure of trapping *all* exceptions, a sort of *catch-all* handler should be defined as last handler.

Another important aspect of error handling that was considered in the archiving program, is the exception *propagation* mechanism. In PL/SQL, an exception raised in a block can propagate in an outer block. This happens either when the exception is *not handled* or when it is explicitly *re-raised* in the exception section. Such propagation continues until the exception is handled or it reaches the outermost block, back to the calling program.

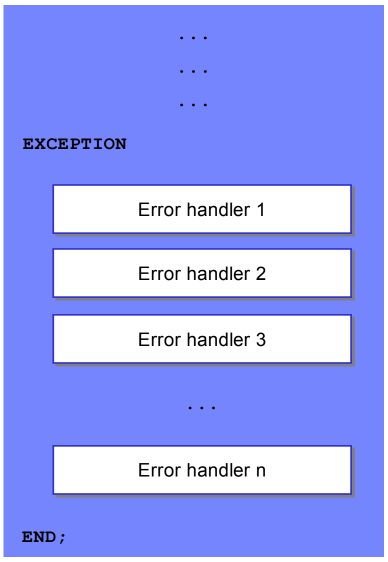


Figure 8.2 Error handlers in an exception section

For this project, the error handling part was implemented by defining error handlers for each execution block. The idea was to define:

- a global variable (character string) for storing error information (mainly the error location).
- a catch-all error handler (in every block) by means of which all exceptions could have been trapped.

When an exception was raised, say in a function, control was passed to the exception section where the global variable was set to its initial value plus a message indicating the function name (and if needed other relevant information). Then the exception was re-raised explicitly and trapped by the outer block. This outer block, in turn, performed the same operations. When the outermost block was reached, the global variable was printed together with the specific Oracle error message related to the initial exception and program execution stopped.
This way, it was possible to identify both the specific Oracle error message and the exact error location (e.g. the path from which the exception originated). Figure 8.3 gives some details on the key statements used in the implementation.

```
PROCEDURE VerifyArchiveTable

     ...
     ...

EXCEPTION

  WHEN OTHERS THEN
    G_err_location := 'procedure ''VerifyArchiveTable'''||ERR_SEP||G_err_location;
    RAISE;

END VerifyArchiveTable;
```

Figure 8.3 Tracing error location and re-raising the exception

Besides the above considerations, such error handling mechanism was supported by other characteristics of the archiving program. For example, the need of catch-all error handlers was due to the fact that many different types of exceptions could have been raised in this specific program. Adopting in a function a large set of error handlers increasingly more generalised would not have been profitable in this context because for all types of exceptions raised no specific "remedy" had to be taken – most of the time, the only feasible action to be performed was the program termination (with detailed log information).

The source code enclosed in Appendix A shows all details of such error handling implementation.

8.2 Programming languages

Development of the financial archiving system was carried out by adopting Oracle's programming languages in order to provide a tight integration with the overall existing ERP system. The reader not familiar with Oracle's programming languages may find a description on such topic in the following subsections.

8.2.1 Oracle SQL

When dealing with databases, information are typically managed by means of the Structured Query Language (SQL). From a user perspective, SQL consists of a set of statements through which users (and their programs) can access data stored in a database. Naturally, in this context, the language used was Oracle SQL, which corresponds to a superset of the SQL92 standard at entry level conformance specification[13], as defined by the American National Standards Institute (ANSI) and the International Standards Organization (ISO). This version of Oracle SQL is also mostly compatible with the minimal conformance level[14] of the latest standard specification SQL99.

8.2.2 PL/SQL

The programming language used for implementing the financial archiving system was Oracle PL/SQL. It has been designed as procedural extension of Oracle SQL. It provides programming constructs that, for example, make possible to establish data encapsulation, overloading, information hiding and the error handling discussed in the first section of this chapter.

A PL/SQL program consists of a number of "blocks" as shown in Figure 8.1. These are, for example, functions, procedures (or the so-called anonymous blocks). In other words, such blocks are the basic units of a PL/SQL program. The important characteristic of such blocks is that they can contain any number of sub-blocks. By means of this structure, each block usually corresponds to a problem to be solved, or part of it (sub-problem). PL/SQL therefore allows the adoption of problem solving techniques based on the divide-and-conquer approach.

[13] SQL92 is formally known as ANSI X3.135-1992. It defines three SQL conformance levels: *entry SQL*, *intermediate SQL* (a superset of entry SQL) and *full SQL* (a superset of intermediate SQL).

[14] Also known as "SQL-99 Core".

8.3 *Tools*

In the following two subsections are described the two main tools used to interact with Oracle databases.

8.3.1 SQL*Plus

SQL*Plus is the Oracle software tool typically used to interact with Oracle databases. By means of SQL*Plus, it is possible to run SQL commands and PL/SQL blocks against an Oracle database instance. In this project, SQL*Plus was essentially used for inspecting database objects and for running SQL and PL/SQL scripts. Such scripts were mainly needed for creating the database structure into which historical data had to be stored.

8.3.2 TOAD

TOAD is a software application that provides a number of features aimed at facilitating all the typical tasks carried out when dealing with databases. Such software tool is developed by Quest Software Inc. and is based on an advanced GUI-based editor.

In this project, TOAD was used, for example, to test functions and procedures, to analyse data models, tables, indexes and all other related database objects. During such development, TOAD provided a valid support on most of the activities carried out.

Chapter 9

Testing

During the development of the financial archiving system, a number of testing activities were carried out. Such activities can be grouped in two main classes: one that relates to the *functional testing* and the other that relates to the *technical testing*.
The two following sections are aimed at describing these two classes of test activities carried out during the project. Then, a third section is provided for a more detailed description of the test sessions performed and the corresponding results.

9.1 Functional testing

Functional testing activities played a relevant role during the development of the financial archiving system. These activities required a considerable cooperation with end users due to the need of their sign-off and to the necessity of an in depth knowledge of accountancy for an accurate interpretation of archiving results. These tests were performed in an "incremental" fashion, as soon as new parts of the system were made available.
In order to perform such testing activities a set of ad-hoc environments had been adopted. More precisely, the four test environments shown in Figure 9.1 had been used. Such set of test environments can be thought of as a set of "sand-box" environments aimed at supporting sessions of "user acceptance test" (UAT).

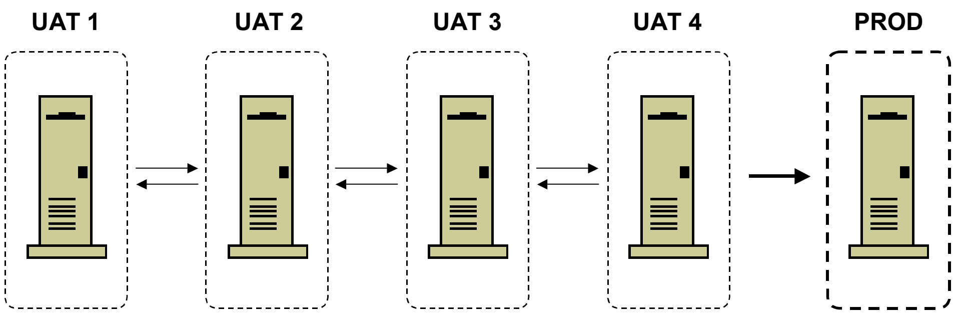


Figure 9.1 Code details for testing

The above structure was adopted in order to allow more accurate functional testing activities. Increasingly difficult tests had been performed into different environments. That is, more complete and complex tests were carried out in environments "closer" to the production environment. Moreover, by adopting such structure, a higher degree of "parallelism" in the testing activities had been possible.
Section 9.3 discusses in more detail the functional tests carried out.

9.2 *Technical testing*

With regards to the technical testing, most of the activities were aimed at validating correctness of the program code.
It is to be noted that the volume of the archiving system, as well as the time constraints and the repeated changes in users' needs made, to some extent, unpractical the implementation of a fully automated and exhaustive testing process of the program.
However, where no automated checks were feasible or justified, manual tests were performed. For example, most of the functions and procedures were tested individually by verifying that all branches in the code were actually executed when expected.
In critical stages of the development, ad-hoc variables were used to enable/disable a specific "debugging" mode execution.

Also, all reasonably foreseeable safety aspects were taken into account. For instance protection of standard production tables was ensured (e.g. by verifying the existence of insertion protection triggers).
Moreover, error handling mechanisms were put in place so as to cope with unexpected behaviours and to make the program terminate always in a specific way.
Error handling mechanisms were tested by forcing the generation of specific exceptions and verifying that the corresponding handler was actually executed.

9.3 Testing sessions

In this section is given a more detailed description of the testing activities mentioned above. The following subsections describe each group of the testing sessions carried out.

9.3.1 Path testing

Once each function had been tested, a different kind of test was performed on the whole program in order to verify the proper execution and integration of each module in the system. Such testing session can be thought of as a "path testing". It was performed by introducing a set of numeric variables used as "counters", in order to trace the execution flow. More in detail, the following steps were followed in setting up the testing session:

- Definition of a boolean variable (G_TestMode) used to enable/disable the *testing mode* (see source code in appendix A.1).
- Definition of a pair of numeric variables used to trace *entrance* and *exit* events in the main procedure. See variables G_TestCntArchAndPurge_Entering and G_TestCntArchAndPurge_Exiting, respectively at the beginning and at the end of main procedure ArchiveAndPurge (test IDs: ArchiveAndPurge.1 and ArchiveAndPurge.34).
- Definition of a numeric variable placed as first statement in the loop construct that processes all *business units*. See variable G_TestCntArchAndPurge_BUs at the beginning of loop CurResponsibility (test ID: ArchiveAndPurge.2).

– Definition of a numeric variable placed as first statement in the loop construct that processes all *periods* in the time range specified. See variable G_TestCntArchAndPurge_Periods at the beginning of loop CurPeriod (test ID: ArchiveAndPurge.3).
– Defintion of five numeric variables for tracing each of the main archiving phases. See variables G_TestCntArchAndPurge_Phase_1 to G_TestCntArchAndPurge_Phase_5 (test IDs: ArchiveAndPurge.4, ArchiveAndPurge.10, ArchiveAndPurge.16, ArchiveAndPurge.22 and ArchiveAndPurge.28).
– Definition of twenty five numeric variables, for tracing module selection (one for each financial module and archiving phase). See variables G_TestCntGL_Phase_1 to G_TestCntGL_Phase_5, G_TestCntAR_Phase_1 to G_TestCntAR_Phase_5, etc. (test IDs:

ArchiveAndPurge.5,	ArchiveAndPurge.6,	ArchiveAndPurge.7,
ArchiveAndPurge.8,	ArchiveAndPurge.9,	ArchiveAndPurge.11,
ArchiveAndPurge.12,	ArchiveAndPurge.13,	ArchiveAndPurge.14,
ArchiveAndPurge.15,	ArchiveAndPurge.18,	ArchiveAndPurge.19,
ArchiveAndPurge.20,	ArchiveAndPurge.21,	ArchiveAndPurge.23,
ArchiveAndPurge.24,	ArchiveAndPurge.25,	ArchiveAndPurge.26,
ArchiveAndPurge.27,	ArchiveAndPurge.29,	ArchiveAndPurge.30,
ArchiveAndPurge.31,	ArchiveAndPurge.32,	ArchiveAndPurge.33).

– Definition of a procedure for inspecting the value of test counters (see procedure PrintTestCounters). This procedure is called at the beginning of the main procedure ArchiveAndPurge, in order to show all initial values, and at the end in order to inspect the final status of the counters (test IDs: ArchiveAndPurge.1 and ArchiveAndPurge.34).

These variables and the printing procedure were used in the program source code as illustrated by the statement in Figure 9.2.

```
------------------------------------ TESTING PURPOSES ------------------------------------
-- Test ID: ArchiveAndPurge.1

-- If test mode is enabled then...
if G_TestMode then

  -- Prints out the initial value of all test counters (all zeros are expected)
  PrintTestCounters;
  -- Increases the 'entering' counter
  G_TestCntArchAndPurge_Entering:=G_TestCntArchAndPurge_Entering+1;
  -- Prints out log text and numeric log information
  Write('Entering main function ArchiveAndPurge');
  Write('Counter G_TestCntArchAndPurge_Entering is'||BLANK||G_TestCntArchAndPurge_Entering);

 end if;

------------------------------------------------------------------------------------------
```

Figure 9.2 Code details for testing purposes

For a given input, the above set of variables was valued so as to indicate the execution path and all the iterations performed.

The input sets were defined by considering the input parameters available to the key user of the archiving program. Precisely the key user has to provide the following input data:

Time range	Which is done by choosing a starting period and an ending period from a list of values. Such list of values is a standard component of the ERP application and contains all valid periods and all relevant consistency checks (e.g. the starting period must be prior or equal to the ending period).
Modules	Which is done by selecting all needed financial modules (by means of flags). At the moment of writing, the two main financial modules, GL and AR, have been fully implemented. Integration of shared modules AP and PO is in progress and two other modules, AX and FA, have already been predisposed for next integration activities.

As an example, the user can select, say, the whole year 1995 and all financial modules. This corresponds to the following input parameters:

From:	01-1995
To:	14-1995
GL:	Yes
AR:	Yes
AP_PO:	Yes
AX:	Yes
FA:	Yes

From such considerations, a number of test cases were devised and tried on the program. The most significant of them are collected in Table B.1 and Table B.2.

9.3.2 Archiving volumes

Another testing session aimed at verifying consistency of the archiving program by considering the number of records archived.
For each business unit and financial module, such testing session was structured as indicated by the following steps:

1. For each standard (source) table involved in the archiving process, its total number of records was calculated.
2. The same calculation was performed on all archive (destination) tables.
3. Then, the archiving program was executed (for a given financial module and time range).
4. After the archiving program completed successfully, steps 1 and 2 were re-executed.

Then, the four sets of figures obtained were compared as follows:

$$\begin{gathered}\textit{source\_table\_totals\_before\_arch} - \textit{source\_table\_totals\_after\_arch} \\ = \\ \textit{destination\_table\_totals\_after\_arch} - \textit{destination\_table\_totals\_before\_arch}\end{gathered}$$

where the above items refers to:

source_table_totals_before_arch	→	Totals computed in the above step 1.
source_table_totals_after_arch	→	Totals computed in the above step 2.
destination_table_totals_after_arch	→	Totals computed in the above step 4.
destination_table_totals_before_arch	→	Totals computed in the above step 4.

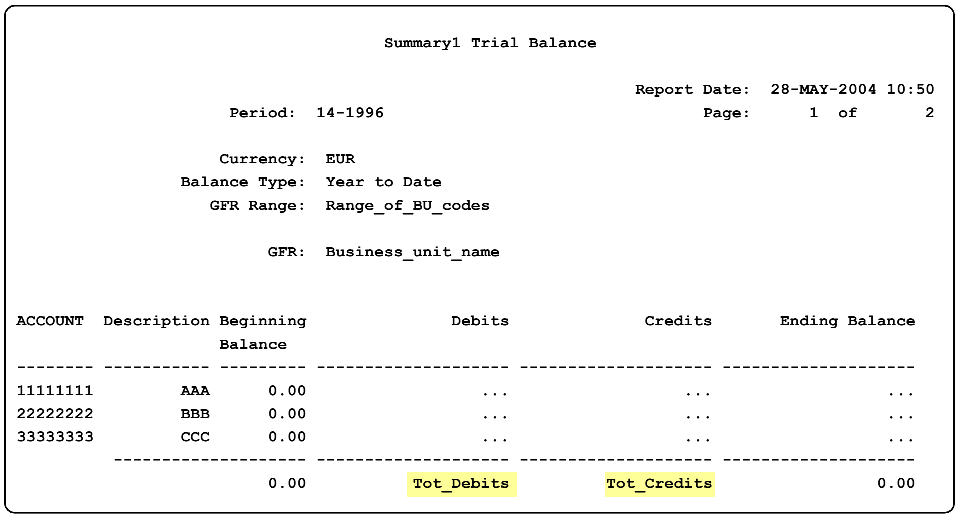
The above equation was expected to be satisfied. In fact, the number of records selected for archiving (first part in the equation) had to equal the number of record archived (second part in the equation). Appendix B (section B.2) describes such test on financial module *General Ledger*.

9.3.3 Economic amounts

Performing tests on the volume of records archived highlighted the process flow fairly at low level and, from a technical perspective, this is of course helpful. However, given the non-technical background of users, and since demonstrations had to be carried out on each financial module, another set of tests had to be arranged at a higher level in order to provide comprehensible information to users.

The idea was to try out some of the standard accounting reports on both the standard and the archive schema. For example, the *"Summary Trial Balance"* (version 1) report was one of the most useful accounting reports for functional testing purposes on financial module *General Ledger*. This report shows, in fact, *debits* and *credits* belonging to each account for a given business unit and a given time range. Figure 9.3 illustrates the information contained in this report[15].

[15] For confidentially reasons, amounts are withheld.



```
                                        Summary1 Trial Balance

                                                                 Report Date:  28-MAY-2004 10:50
                      Period:  14-1996                                  Page:      1  of       2

                     Currency:  EUR
                 Balance Type:  Year to Date
                    GFR Range:  Range_of_BU_codes

                          GFR:  Business_unit_name


ACCOUNT  Description Beginning              Debits              Credits       Ending Balance
                     Balance
-------- ----------- --------- -------------------- -------------------- --------------------
11111111         AAA      0.00                  ...                  ...                  ...
22222222         BBB      0.00                  ...                  ...                  ...
33333333         CCC      0.00                  ...                  ...                  ...
          -------------------- -------------------- -------------------- --------------------
                          0.00           Tot_Debits          Tot_Credits                 0.00
```

Figure 9.3 Summary trial balance

The test plan arranged consisted of the following steps:

1. A business unit and a period range were chosen and used as parameters for both the *Summary Trial Balance* report and the archiving program.
2. The *Summary Trial Balance* report was run on the standard schema before executing the archiving process.
3. The *Summary Trial Balance* report was also run on the archive schema before executing the archiving process.
4. The archiving process was then executed.
5. The *Summary Trial Balance* report was run once again on the standard schema.
6. The *Summary Trial Balance* report was run once again on the archive schema.

Once the above procedure was completed, an analysis on the reports obtained was performed. Of particular relevance were the to amounts highlighted in Figure 9.3 and the successful conditions had to be:

$$\textit{source\_Tot\_Credits\_before\_arch} - \textit{source\_Tot\_Credits\_after\_arch}$$
$$=$$
$$\textit{destination\_Tot\_Credits\_after\_arch} - \textit{destination\_Tot\_Credits\_before\_arch}$$

and

$$\textit{source\_Tot\_Debits\_before\_arch} - \textit{source\_Tot\_Debits\_after\_arch}$$
$$=$$
$$\textit{destination\_Tot\_Debits\_after\_arch} - \textit{destination\_Tot\_Debits\_before\_arch}$$

where the above items refers to:

source_Tot_Credits_before_arch	→	Totals computed in step 2.
source_Tot_Debits_before_arch	→	Totals computed in step 2.
destination_Tot_Credits_before_arch	→	Totals computed in step 3.
destination_Tot_Debits_before_arch	→	Totals computed in step 3.
source_Tot_Credits_after_arch	→	Totals computed in step 5.
source_Tot_Debits_after_arch	→	Totals computed in step 5.
destination_Tot_Credits_after_arch	→	Totals computed in step 6.
destination_Tot_Debits_after_arch	→	Totals computed in step 6.

As an example, if in the source environment credits *diminishes* for 1000 €, this implies that credits in the destination environment must *increase* for the same amount.

More precisely, even just one of the two above equations (indifferently the first or the second) is sufficient for the testing purposes here described. This is due to the nature of the recordings performed with the *double-entry* accountancy system. According to such accountancy system, in fact, the amount of debits must always equal the amount of credits.

Thus, such approach was used to perform functional tests and to show users how the underlying processing was actually moving historical data into the archive schema without affecting the corresponding functional “meaning” of data.
For this kind of tests, no material is enclosed in the appendix due to confidentiality reasons.

Conclusions

This last chapter deals with the conclusions that can be drawn from the implementation of the financial archiving system.

The system devised

With the new system it is possible to remove historical data from the ERP system in use, permitting the allocation of more resources to online data (i.e. non-historical data) that are used during day-to-day business activities.
A considerably reduced amount of data in the production environment leads to an overall improvement of system performance.
For example, the impact of database cloning and back-up activities is closely dependent on the database volume, and since such activities now involve a considerably smaller set of data, ordinary database administration and maintenance can be performed more easily and effectively both in terms of cost and time.
Besides improving database manageability, database efficiency is also increased. For the same reasons, in fact, data scansions on the main financial modules are now performed with improved response times.
The other important aspect of such system is that all historical data can *still* be accessed. It is in fact worth noting that historical data is not "packaged" in some way, placed in some kind of storage media and brought back when needed. Historical data is, instead, "spread" on a number of hard disks (other than those for production) installed on various available machines (usually smaller than those for production). Such disks and machines correspond to hardware components that for several reasons (e.g. inadequate processing power, speed or storage capacity, etc.) are no longer used for production environments anymore and are left unused or

dedicated to less demanding purposes. Historical data is then stored in these hard disks and kept in the same original and readily available format.
In other terms, a sort of "data distribution" policy has been implemented, enabling "hardware components re-usage" that in turn has allowed freeing up expensive resources for production data.
The system moreover has been designed in such a way as to allow access to historical data through the same ERP user interface. In other words, users access historical data the same way they access non-historical data. For example, an archived invoice can be viewed exactly in the same forms used when viewing an online invoice. This way, the impact on users' work procedures is minimal and almost no user training is required in order to enable using the archiving system properly.
However, what is most important is that such integration with the ERP system allows all ERP functions to be applied to historical data as well, as if they were non-historical data. For example, common financial reports normally generated on online data can be generated from historical data. This is particularly important for all management and supervision activities that often involve unexpected and urgent inspections on financial data. This becomes even more important when considering all fiscal requirements that entail the ability of reproducing fiscal reports upon the legal authorities' request.

Some thought should also be given to the architecture of the financial archiving system. Its main structure has been designed in order to privilege *modularity*. The main structure in fact allows integrating in the system all required financial modules as gradually as needed. Furthermore, such modularity, combined with software components re-usage, also facilitate maintenance, which is in fact limited to a well defined set of points.

Some notes on the current status of the system. At the moment of writing, the system developed was completed for all the required business units and regards the two main modules: *General Ledger* and *Accounts Receivables*. Users have completed acceptance tests and switch to production will take place shortly, presumably after completing the implementation of shared modules *Accounts Payables* and *Purchasing* that is currently in process.

Future developments

At the moment of writing, I consider that the financial archiving system will have to be further developed since other business units within the group will soon be included in the archiving process. No critical impact is however expected from such development. Given the system structure, essentially all that will be required is identifying the relevant business units ID codes and entering them in the archiving system configuration.

Something else that probably has to be added into the archiving system concerns legislative requirements regarding data retention for each branch within the company group (see Table 5.1). Although such requirements have been fulfilled by retaining all historical data, the deletion of historical data that is beyond the years specified in the legal data retention requisites is left to users and this was done owing to several changes in the user requirements which did not allow including this feature within the given timeframe of the thesis project. Consequently, a future work will most likely involve introducing a routine that basically deletes historical data from the archive schema when it goes beyond the number of years prescribed by country-specific legislative requirements.

Also worth mentioning is an additional feature developed during the project, by means of which, it is possible to notify users of the completion of specific events by forwarding them short telephone text messages and/or e-mails. This feature was initially developed because of the relatively long execution times required by the archiving program. Then after it was written as a separate PL/SQL package giving all ERP users carrying out time-consuming jobs (or a complex set of jobs) the possibility of being readily informed when completion comes about. A user either places the corresponding routine invocation in the relevant location in his/her program source code (e.g. as last statement) or runs the utility program through the standard ERP programs submission window as part of a set of requests (this case would consist of all user requests plus utility program at the end).
Given the good user feedback and appreciation shown, this utility will probably be installed into the production environment and made available to all users.

Another aspect that I consider important refers to the development recommendations made available by The Company for the implementation of the financial archiving system. A personal impression on such recommendations is that error handling aspects should have been covered more thoroughly. It is important to provide robust error handling in each program and it is also advisable to define a common set of guidelines to be followed by all teams for development activities. An idea could be to design a kind of ad-hoc “parser” into which a number of checks could be encoded. Such parser could hypothetically over time keep on encapsulating the contribution of all teams (in terms of best practices and advice) and then be used to analyse the source code of programs. This way, the quality of the programs would certainly soon improve (despite external factors such as employee turn-over, etc.) and there would be fewer unexpected behaviour risks.

Personal considerations

Lastly, a few considerations from a more personal perspective.
Carrying out a project in a factual work environment has definitely been a singular experience. The great commitment in the project by the users, together with the concrete challenge posed, further stimulated this experience and strongly motivated me during all the months that it took to finish my thesis.
Such experience however revealed several critical aspects that were not clearly foreseeable nor easily assessable at the beginning of the project. Firstly, the program had to be written for accounting purposes and this called for studies and investigations in this field in order to become acquainted and be familiar with it. Secondly, several times users suddenly changed their needs while carrying out the work, and this consequently changed my schedule. Furthermore, the nature of the project involved several financial areas and thus several users, which posed some difficulties concerning testing activities. Arranging the testing activities was at times complicated by other external factors such as temporary unavailability of test environments, urgent tasks carried out on the production environment, and so on.

In other terms, conducting a study in a real work environment implies facing several unavoidable “disruption” factors, but then again learning to deal with these factors appears to be just as important as the technical aspects.
Even though these factors seem obvious, they can have a non-negligible impact if considered in the context of this thesis. Trying to conciliate work issues with academic issues, as was the case of my thesis, sometimes called for additional efforts, but I must say that such a “lesson” was beneficial. And extremely beneficial were the findings and the technical expertise I acquired. Both these aspects greatly helped me to profitably combine methodical approaches learned at university with practical, hands-on work experience.
Another source of gratification is that The Company appears to appreciate my work and Oraplus intends to propose that I keep on working for the company.

APPENDICES

Appendix A

This appendix contains all source code needed for the financial archiving system. It consists of two main sections dedicated to the archiving program and to the underlying database structure.

Note: The source code reported below has been reformatted due to layout reasons of the document. This, in some cases has affected the indentation and in turn readability.

A.1 Archiving program

In this copy of the thesis, source code is withheld for confidentially reasons.

A.1.1 Package specification

In this copy of the thesis, source code is withheld for confidentially reasons.

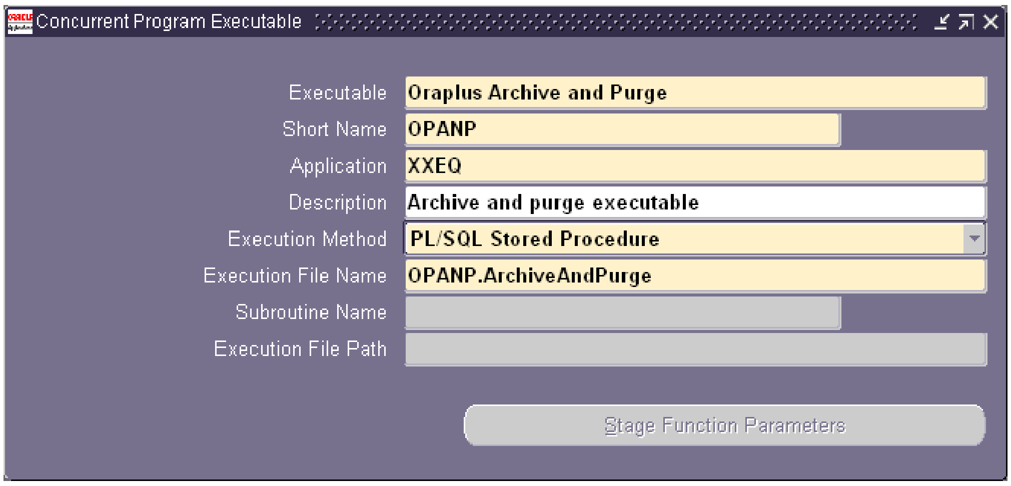
A.1.2 Package body

In this copy of the thesis, source code is withheld for confidentially reasons.

A.1.3 Program registration

Here is described the sequence of steps that were taken for registering the archiving program in the ERP application. This task is needed in order to make the archiving program available to the designated key user.

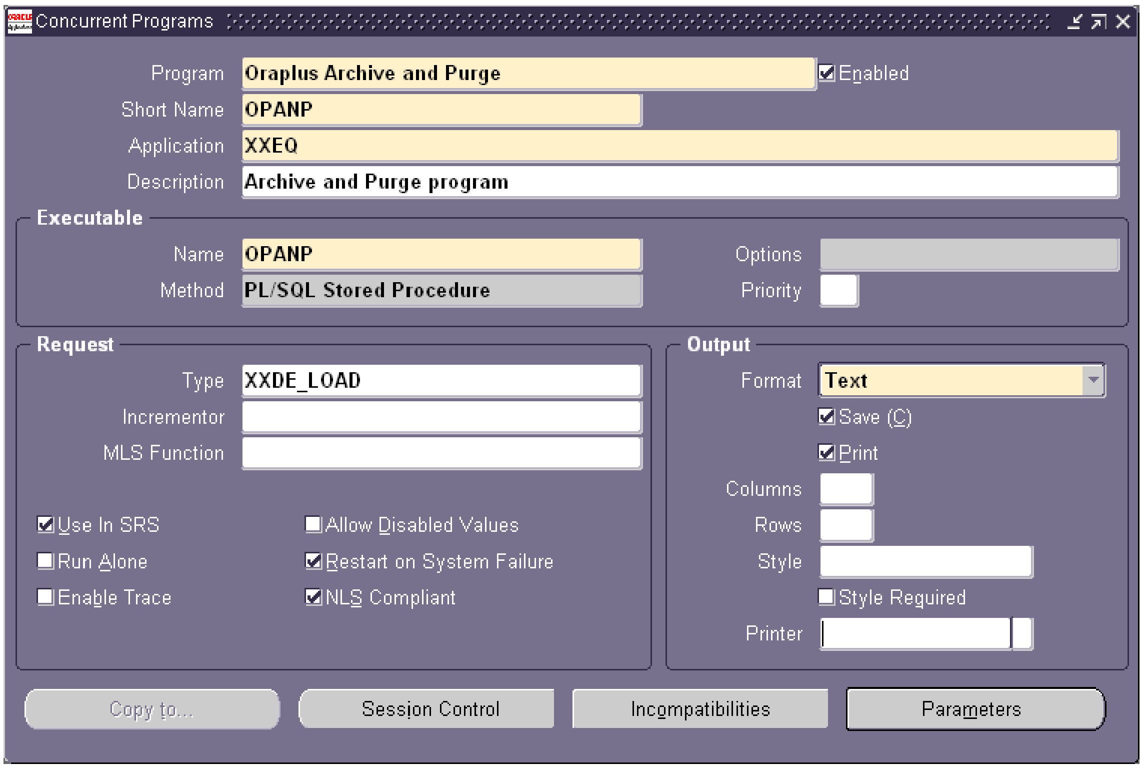
The first step consists of registering a so-called "executable", that is the PL/SQL main procedure "ArchiveAndPurge" of the package "OPANP" (see Figure A.1). Menu selection: *Concurrent* → *Program* → *Executable*.



Executable	Oraplus Archive and Purge
Short Name	OPANP
Application	XXEQ
Description	Archive and purge executable
Execution Method	PL/SQL Stored Procedure
Execution File Name	OPANP.ArchiveAndPurge
Subroutine Name	
Execution File Path	

Figure A.1 Executable definition

The second operation that must be carried out is the definition of a concurrent program as shown in Figure A.2. Menu selection: *Concurrent → Program → Define.*


Concurrent Programs

Program	Oraplus Archive and Purge	☑ Enabled	
Short Name	OPANP		
Application	XXEQ		
Description	Archive and Purge program		

Executable

Name	OPANP	Options	
Method	PL/SQL Stored Procedure	Priority	

Request

Type	XXDE_LOAD
Incrementor	
MLS Function	

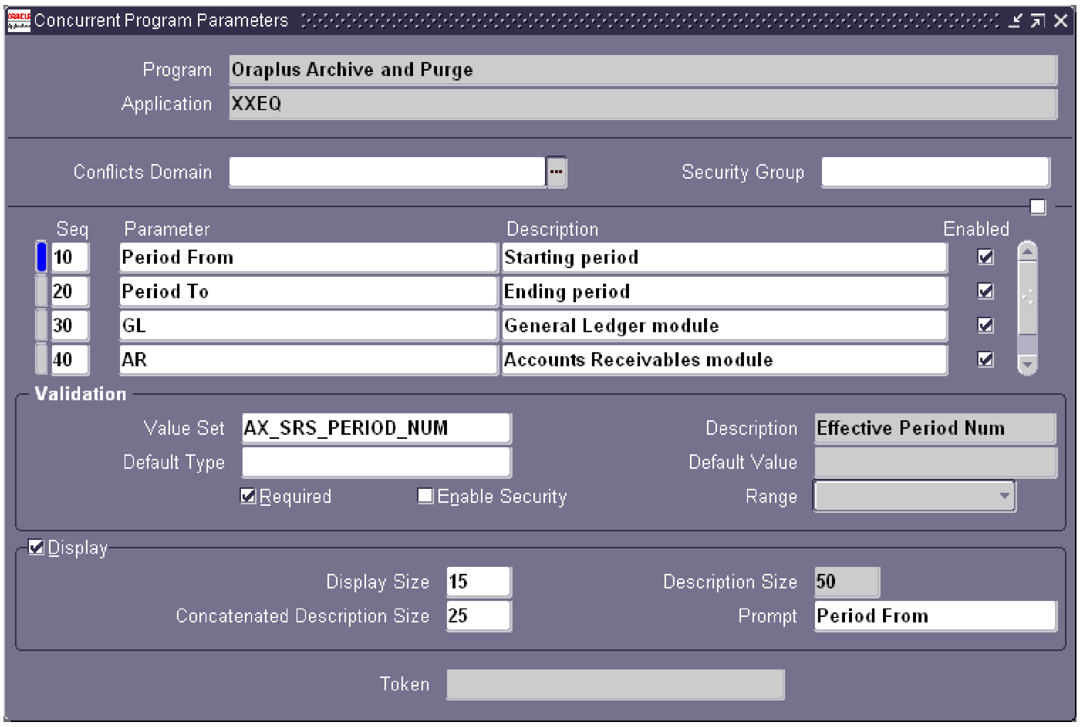
☑ Use In SRS ☐ Allow Disabled Values
☐ Run Alone ☑ Restart on System Failure
☐ Enable Trace ☑ NLS Compliant

Output

Format	Text
	☑ Save (C)
	☑ Print
Columns	
Rows	
Style	
	☐ Style Required
Printer	

Figure A.2 Concurrent program definition

Once the mandatory information have been entered, it is necessary to specify the parameters that will be passed to the underlying executable (that is to the main procedure “ArchiveAndPurge”). In Figure A.3 can be seen some of the parameters defined. Menu selection: *Choose button “Parameters” in the Concurrent Programs window.*


Concurrent Program Parameters

Program	Oraplus Archive and Purge
Application	XXEQ

Conflicts Domain		Security Group	

Seq	Parameter	Description	Enabled
10	Period From	Starting period	☑
20	Period To	Ending period	☑
30	GL	General Ledger module	☑
40	AR	Accounts Receivables module	☑

Validation

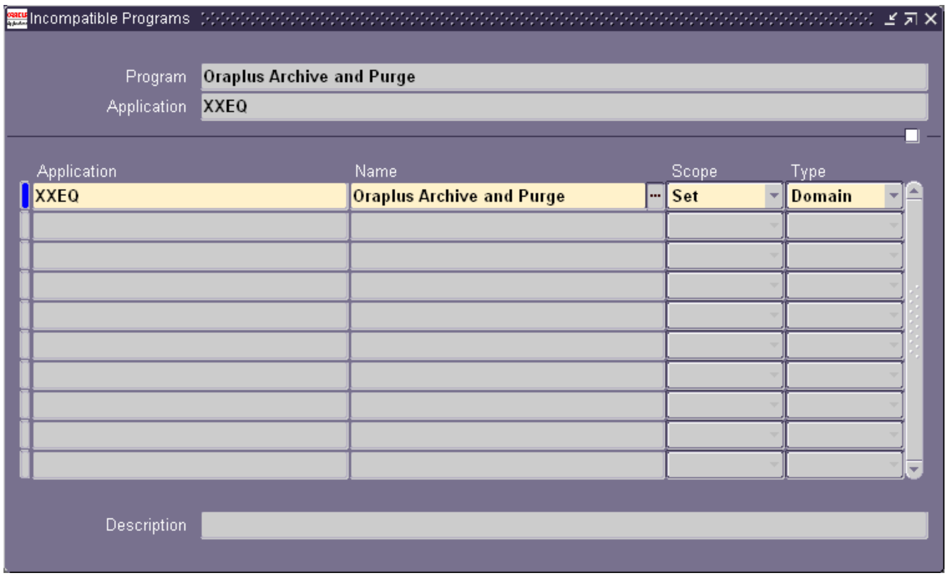
Value Set	AX_SRS_PERIOD_NUM	Description	Effective Period Num
Default Type		Default Value	
☑ Required	☐ Enable Security	Range	

☑ Display

Display Size	15	Description Size	50
Concatenated Description Size	25	Prompt	Period From
Token			

Figure A.3 Parameters definition

The next step is represented by the specification of programs incompatible with the archiving program. Since the archiving program can be influenced by itself only (administration tasks do not need to be specified here), the incompatible program to be specified is the one shown in Figure A.4. Menu selection: *Choose button "Incompatibilities" in the Concurrent Programs window.*



Program	Oraplus Archive and Purge
Application	XXEQ

Application	Name	Scope	Type
XXEQ	Oraplus Archive and Purge	Set	Domain

Description

Figure A.4 Incompatible programs

Finally, it is required to associate the registered program to the request group belonging to the designated key user. Menu selection: *Concurrent → Program → Request group.*

A.2 Database schema

This section contains the four main scripts used for creating the archive schema into which all historical data are contained.

A.2.1 Initial configuration

In this copy of the thesis, source code is withheld for confidentially reasons.

A.2.2 Archive table

In this copy of the thesis, source code is withheld for confidentially reasons.

A.2.3 Create archive schema

In this copy of the thesis, source code is withheld for confidentially reasons.

A.2.4 Create archive responsibilities

In this copy of the thesis, source code is withheld for confidentially reasons.

Appendix B

This appendix contains results of the main testing activities.

B.1 Path testing results

The following table summarises the main input sets defined in order to carry out path testing activities on the financial archiving system. For a given row, the first column shows the name of the input set defined; the second column indicates the input details; the third column gives a brief description of the input set.

Input set	Input Values	Input Description

A	Period range:
	
	From = '01-1995'
	To = '14-1995'
	
	Modules:
	
	GL = 'N'
	AR = 'N'
	AP_PO = 'N'
	AX = 'N'
	FA = 'N'
		Here, is selected a whole year and none of the modules. With no modules selected, this first test should show that no module-specific code is executed.

B	Period range:
	
	From = '01-1995'
	To = '14-1995'
	
	Modules:
	
	GL = 'Y'
	AR = 'N'
	AP_PO = 'N'
	AX = 'N'
	FA = 'N'
		In this input set, the time range is still a whole year but a single module is selected. Thus, execution of source code related to the module GL is expected.

C	Period range:
	
	From = '01-1995'
	To = '14-1995'
	
	Modules:
	
	GL = 'N'
	AR = 'Y'
	AP_PO = 'N'
	AX = 'N'
	FA = 'N'
		As input set B but with a different module selected.

Input set	Input Values	Input Description

D	Period range:
	
	From = ‘01-1995’
	To = ‘14-1995’
	
	Modules:
	
	GL = ’N’
	AR = ’N’
	AP_PO = ’Y’
	AX = ’N’
	FA = ’N’
		As input set B but with a different module selected.

E	Period range:
	
	From = ‘01-1995’
	To = ‘14-1995’
	
	Modules:
	
	GL = ’N’
	AR = ’N’
	AP_PO = ’N’
	AX = ’Y’
	FA = ’N’
		As input set B but with a different module selected.

F	Period range:
	
	From = ‘01-1995’
	To = ‘14-1995’
	
	Modules:
	
	GL = ’N’
	AR = ’N’
	AP_PO = ’N’
	AX = ’N’
	FA = ’Y’
		As input set B but with a different module selected.

G	Period range:
	
	From = ‘01-1995’
	To = ‘14-1995’
	
	Modules:
	
	GL = ’Y’
	AR = ’Y’
	AP_PO = ’Y’
	AX = ’Y’
	FA = ’Y’
		The time range is the same as above but all modules are now selected.

H	Period range:
	
	From = ‘01-1995’
	To = ‘01-1995’
	
	Modules:
	
	GL = ’Y’
	AR = ’Y’
	AP_PO = ’Y’
	AX = ’Y’
	FA = ’Y’
		In this input set, the time range consists of just one period. All modules are selected.

Input set	Input Values	Input Description

I	Period range:
	
	From = '01-1997'
	To = '02-1998'
	
	Modules:
	
	GL = 'Y'
	AR = 'N'
	AP_PO = 'N'
	AX = 'N'
	FA = 'N'
		In this input set, the time range consists of more than a fiscal year. Just one module is selected.

L	Period range:
	
	From = '01-1997'
	To = '02-1998'
	
	Modules:
	
	GL = 'Y'
	AR = 'N'
	AP_PO = 'Y'
	AX = 'Y'
	FA = 'N'
		In this input set, the time range consists still of than a fiscal year. Several modules are selected.

Table B.1 Input set definition

For each of the above described input sets, the expected values of ad-hoc counters were calculated and then compared with the corresponding values shown after the test case completion. These results are summarised in Table B.2.

Input set	Input values	Expected output	Observed output	Status

A	Period range:
	
	From = ‘01-1995’
	
	To = ‘14-1995’
	
	Modules:
	
	GL = ’N’
	AR = ’N’
	AP_PO = ’N’
	AX = ’N’
	FA = ’N’
		All counters printed out by the first invocation of procedure “PrintTestCounters” shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK

		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_1 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_2 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_3 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_4 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_5 = 14*9	126	OK
		G_TestCntGL_Phase_1 = 0	0	OK
		G_TestCntGL_Phase_2 = 0	0	OK
		G_TestCntGL_Phase_3 = 0	0	OK
		G_TestCntGL_Phase_4 = 0	0	OK
		G_TestCntGL_Phase_5 = 0	0	OK
		G_TestCntAR_Phase_1 = 0	0	OK
		G_TestCntAR_Phase_2 = 0	0	OK
		G_TestCntAR_Phase_3 = 0	0	OK
		G_TestCntAR_Phase_4 = 0	0	OK
		G_TestCntAR_Phase_5 = 0	0	OK
		G_TestCntAP_Phase_1 = 0	0	OK
		G_TestCntAP_Phase_2 = 0	0	OK
		G_TestCntAP_Phase_3 = 0	0	OK
		G_TestCntAP_Phase_4 = 0	0	OK
		G_TestCntAP_Phase_5 = 0	0	OK
		G_TestCntAX_Phase_1 = 0	0	OK
		G_TestCntAX_Phase_2 = 0	0	OK
		G_TestCntAX_Phase_3 = 0	0	OK
		G_TestCntAX_Phase_4 = 0	0	OK
		G_TestCntAX_Phase_5 = 0	0	OK
		G_TestCntFA_Phase_1 = 0	0	OK
		G_TestCntFA_Phase_2 = 0	0	OK
		G_TestCntFA_Phase_3 = 0	0	OK
		G_TestCntFA_Phase_4 = 0	0	OK
		G_TestCntFA_Phase_5 = 0	0	OK

Input set	Input values	Expected output	Observed output	Status
		All counters printed out by the first invocation of procedure “PrintTestCounters” shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_1 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_2 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_3 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_4 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_5 = 14*9	126	OK
	Period range:			
		G_TestCntGL_Phase_1 = 14*9	126	OK
	From = ‘01-1995’	G_TestCntGL_Phase_2 = 14*9	126	OK
	To = ‘14-1995’	G_TestCntGL_Phase_3 = 14*9	126	OK
		G_TestCntGL_Phase_4 = 14*9	126	OK
		G_TestCntGL_Phase_5 = 14*9	126	OK
B	Modules:	G_TestCntAR_Phase_1 = 0	0	OK
		G_TestCntAR_Phase_2 = 0	0	OK
	GL = ’Y’	G_TestCntAR_Phase_3 = 0	0	OK
	AR = ’N’	G_TestCntAR_Phase_4 = 0	0	OK
	AP_PO = ’N’	G_TestCntAR_Phase_5 = 0	0	OK
	AX = ’N’			
	FA = ’N’	G_TestCntAP_Phase_1 = 0	0	OK
		G_TestCntAP_Phase_2 = 0	0	OK
		G_TestCntAP_Phase_3 = 0	0	OK
		G_TestCntAP_Phase_4 = 0	0	OK
		G_TestCntAP_Phase_5 = 0	0	OK
		G_TestCntAX_Phase_1 = 0	0	OK
		G_TestCntAX_Phase_2 = 0	0	OK
		G_TestCntAX_Phase_3 = 0	0	OK
		G_TestCntAX_Phase_4 = 0	0	OK
		G_TestCntAX_Phase_5 = 0	0	OK
		G_TestCntFA_Phase_1 = 0	0	OK
		G_TestCntFA_Phase_2 = 0	0	OK
		G_TestCntFA_Phase_3 = 0	0	OK
		G_TestCntFA_Phase_4 = 0	0	OK
		G_TestCntFA_Phase_5 = 0	0	OK

Input set	Input values	Expected output	Observed output	Status
C		All counters printed out by the first invocation of procedure "PrintTestCounters" shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_1 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_2 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_3 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_4 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_5 = 14*9	126	OK
	Period range:			
		G_TestCntGL_Phase_1 = 0	0	OK
	From = '01-1995'	G_TestCntGL_Phase_2 = 0	0	OK
	To = '14-1995'	G_TestCntGL_Phase_3 = 0	0	OK
		G_TestCntGL_Phase_4 = 0	0	OK
		G_TestCntGL_Phase_5 = 0	0	OK
	Modules:	G_TestCntAR_Phase_1 = 14*9	126	OK
		G_TestCntAR_Phase_2 = 14*9	126	OK
	GL = 'N'	G_TestCntAR_Phase_3 = 14*9	126	OK
	AR = 'Y'	G_TestCntAR_Phase_4 = 14*9	126	OK
	AP_PO = 'N'	G_TestCntAR_Phase_5 = 14*9	126	OK
	AX = 'N'			
	FA = 'N'	G_TestCntAP_Phase_1 = 0	0	OK
		G_TestCntAP_Phase_2 = 0	0	OK
		G_TestCntAP_Phase_3 = 0	0	OK
		G_TestCntAP_Phase_4 = 0	0	OK
		G_TestCntAP_Phase_5 = 0	0	OK
		G_TestCntAX_Phase_1 = 0	0	OK
		G_TestCntAX_Phase_2 = 0	0	OK
		G_TestCntAX_Phase_3 = 0	0	OK
		G_TestCntAX_Phase_4 = 0	0	OK
		G_TestCntAX_Phase_5 = 0	0	OK
		G_TestCntFA_Phase_1 = 0	0	OK
		G_TestCntFA_Phase_2 = 0	0	OK
		G_TestCntFA_Phase_3 = 0	0	OK
		G_TestCntFA_Phase_4 = 0	0	OK
		G_TestCntFA_Phase_5 = 0	0	OK

Input set	Input values	Expected output	Observed output	Status
		All counters printed out by the first invocation of procedure "PrintTestCounters" shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_1 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_2 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_3 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_4 = 14*9	126	OK
	Period range:	G_TestCntArchAndPurge_Phase_5 = 14*9	126	OK
		G_TestCntGL_Phase_1 = 0	0	OK
	From = '01-1995'	G_TestCntGL_Phase_2 = 0	0	OK
	To = '14-1995'	G_TestCntGL_Phase_3 = 0	0	OK
		G_TestCntGL_Phase_4 = 0	0	OK
		G_TestCntGL_Phase_5 = 0	0	OK
D	Modules:	G_TestCntAR_Phase_1 = 0	0	OK
		G_TestCntAR_Phase_2 = 0	0	OK
	GL = 'N'	G_TestCntAR_Phase_3 = 0	0	OK
	AR = 'N'	G_TestCntAR_Phase_4 = 0	0	OK
	AP_PO = 'Y'	G_TestCntAR_Phase_5 = 0	0	OK
	AX = 'N'			
	FA = 'N'	G_TestCntAP_Phase_1 = 14*9	126	OK
		G_TestCntAP_Phase_2 = 14*9	126	OK
		G_TestCntAP_Phase_3 = 14*9	126	OK
		G_TestCntAP_Phase_4 = 14*9	126	OK
		G_TestCntAP_Phase_5 = 14*9	126	OK
		G_TestCntAX_Phase_1 = 0	0	OK
		G_TestCntAX_Phase_2 = 0	0	OK
		G_TestCntAX_Phase_3 = 0	0	OK
		G_TestCntAX_Phase_4 = 0	0	OK
		G_TestCntAX_Phase_5 = 0	0	OK
		G_TestCntFA_Phase_1 = 0	0	OK
		G_TestCntFA_Phase_2 = 0	0	OK
		G_TestCntFA_Phase_3 = 0	0	OK
		G_TestCntFA_Phase_4 = 0	0	OK
		G_TestCntFA_Phase_5 = 0	0	OK

Input set	Input values	Expected output	Observed output	Status

E	Period range:
	
	From = ‘01-1995’
	To = ‘14-1995’
	
	Modules:
	
	GL = ’N’
	AR = ’N’
	AP_PO = ’N’
	AX = ’Y’
	FA = ’N’
		All counters printed out by the first invocation of procedure “PrintTestCounters” shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK

		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_1 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_2 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_3 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_4 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_5 = 14*9	126	OK
		G_TestCntGL_Phase_1 = 0	0	OK
		G_TestCntGL_Phase_2 = 0	0	OK
		G_TestCntGL_Phase_3 = 0	0	OK
		G_TestCntGL_Phase_4 = 0	0	OK
		G_TestCntGL_Phase_5 = 0	0	OK
		G_TestCntAR_Phase_1 = 0	0	OK
		G_TestCntAR_Phase_2 = 0	0	OK
		G_TestCntAR_Phase_3 = 0	0	OK
		G_TestCntAR_Phase_4 = 0	0	OK
		G_TestCntAR_Phase_5 = 0	0	OK
		G_TestCntAP_Phase_1 = 0	0	OK
		G_TestCntAP_Phase_2 = 0	0	OK
		G_TestCntAP_Phase_3 = 0	0	OK
		G_TestCntAP_Phase_4 = 0	0	OK
		G_TestCntAP_Phase_5 = 0	0	OK
		G_TestCntAX_Phase_1 = 14*9	126	OK
		G_TestCntAX_Phase_2 = 14*9	126	OK
		G_TestCntAX_Phase_3 = 14*9	126	OK
		G_TestCntAX_Phase_4 = 14*9	126	OK
		G_TestCntAX_Phase_5 = 14*9	126	OK
		G_TestCntFA_Phase_1 = 0	0	OK
		G_TestCntFA_Phase_2 = 0	0	OK
		G_TestCntFA_Phase_3 = 0	0	OK
		G_TestCntFA_Phase_4 = 0	0	OK
		G_TestCntFA_Phase_5 = 0	0	OK

Input set	Input values	Expected output	Observed output	Status
		All counters printed out by the first invocation of procedure “PrintTestCounters” shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_1 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_2 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_3 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_4 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_5 = 14*9	126	OK
	Period range:			
		G_TestCntGL_Phase_1 = 0	0	OK
	From = ‘01-1995’	G_TestCntGL_Phase_2 = 0	0	OK
	To = ‘14-1995’	G_TestCntGL_Phase_3 = 0	0	OK
		G_TestCntGL_Phase_4 = 0	0	OK
		G_TestCntGL_Phase_5 = 0	0	OK
F	Modules:	G_TestCntAR_Phase_1 = 0	0	OK
		G_TestCntAR_Phase_2 = 0	0	OK
	GL = ’N’	G_TestCntAR_Phase_3 = 0	0	OK
	AR = ’N’	G_TestCntAR_Phase_4 = 0	0	OK
	AP_PO = ’N’	G_TestCntAR_Phase_5 = 0	0	OK
	AX = ’N’			
	FA = ’Y’	G_TestCntAP_Phase_1 = 0	0	OK
		G_TestCntAP_Phase_2 = 0	0	OK
		G_TestCntAP_Phase_3 = 0	0	OK
		G_TestCntAP_Phase_4 = 0	0	OK
		G_TestCntAP_Phase_5 = 0	0	OK
		G_TestCntAX_Phase_1 = 0	0	OK
		G_TestCntAX_Phase_2 = 0	0	OK
		G_TestCntAX_Phase_3 = 0	0	OK
		G_TestCntAX_Phase_4 = 0	0	OK
		G_TestCntAX_Phase_5 = 0	0	OK
		G_TestCntFA_Phase_1 = 14*9	126	OK
		G_TestCntFA_Phase_2 = 14*9	126	OK
		G_TestCntFA_Phase_3 = 14*9	126	OK
		G_TestCntFA_Phase_4 = 14*9	126	OK
		G_TestCntFA_Phase_5 = 14*9	126	OK

Input set	Input values	Expected output	Observed output	Status
		All counters printed out by the first invocation of procedure "PrintTestCounters" shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_1 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_2 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_3 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_4 = 14*9	126	OK
		G_TestCntArchAndPurge_Phase_5 = 14*9	126	OK
	Period range:			
		G_TestCntGL_Phase_1 = 14*9	126	OK
	From = '01-1995'	G_TestCntGL_Phase_2 = 14*9	126	OK
	To = '14-1995'	G_TestCntGL_Phase_3 = 14*9	126	OK
		G_TestCntGL_Phase_4 = 14*9	126	OK
		G_TestCntGL_Phase_5 = 14*9	126	OK
G	Modules:	G_TestCntAR_Phase_1 = 14*9	126	OK
		G_TestCntAR_Phase_2 = 14*9	126	OK
	GL = 'Y'	G_TestCntAR_Phase_3 = 14*9	126	OK
	AR = 'Y'	G_TestCntAR_Phase_4 = 14*9	126	OK
	AP_PO = 'Y'	G_TestCntAR_Phase_5 = 14*9	126	OK
	AX = 'Y'			
	FA = 'Y'	G_TestCntAP_Phase_1 = 14*9	126	OK
		G_TestCntAP_Phase_2 = 14*9	126	OK
		G_TestCntAP_Phase_3 = 14*9	126	OK
		G_TestCntAP_Phase_4 = 14*9	126	OK
		G_TestCntAP_Phase_5 = 14*9	126	OK
		G_TestCntAX_Phase_1 = 14*9	126	OK
		G_TestCntAX_Phase_2 = 14*9	126	OK
		G_TestCntAX_Phase_3 = 14*9	126	OK
		G_TestCntAX_Phase_4 = 14*9	126	OK
		G_TestCntAX_Phase_5 = 14*9	126	OK
		G_TestCntFA_Phase_1 = 14*9	126	OK
		G_TestCntFA_Phase_2 = 14*9	126	OK
		G_TestCntFA_Phase_3 = 14*9	126	OK
		G_TestCntFA_Phase_4 = 14*9	126	OK
		G_TestCntFA_Phase_5 = 14*9	126	OK

Input set	Input values	Expected output	Observed output	Status
		All counters printed out by the first invocation of procedure “PrintTestCounters” shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 9	9	OK
		G_TestCntArchAndPurge_Phase_1 = 9	9	OK
		G_TestCntArchAndPurge_Phase_2 = 9	9	OK
		G_TestCntArchAndPurge_Phase_3 = 9	9	OK
		G_TestCntArchAndPurge_Phase_4 = 9	9	OK
		G_TestCntArchAndPurge_Phase_5 = 9	9	OK
	Period range:			
		G_TestCntGL_Phase_1 = 9	9	OK
	From = ‘01-1995’	G_TestCntGL_Phase_2 = 9	9	OK
	To = ‘01-1995’	G_TestCntGL_Phase_3 = 9	9	OK
		G_TestCntGL_Phase_4 = 9	9	OK
		G_TestCntGL_Phase_5 = 9	9	OK
H	Modules:	G_TestCntAR_Phase_1 = 9	9	OK
		G_TestCntAR_Phase_2 = 9	9	OK
	GL = ’Y’	G_TestCntAR_Phase_3 = 9	9	OK
	AR = ’Y’	G_TestCntAR_Phase_4 = 9	9	OK
	AP_PO = ’Y’	G_TestCntAR_Phase_5 = 9	9	OK
	AX = ’Y’			
	FA = ’Y’	G_TestCntAP_Phase_1 = 9	9	OK
		G_TestCntAP_Phase_2 = 9	9	OK
		G_TestCntAP_Phase_3 = 9	9	OK
		G_TestCntAP_Phase_4 = 9	9	OK
		G_TestCntAP_Phase_5 = 9	9	OK
		G_TestCntAX_Phase_1 = 9	9	OK
		G_TestCntAX_Phase_2 = 9	9	OK
		G_TestCntAX_Phase_3 = 9	9	OK
		G_TestCntAX_Phase_4 = 9	9	OK
		G_TestCntAX_Phase_5 = 9	9	OK
		G_TestCntFA_Phase_1 = 9	9	OK
		G_TestCntFA_Phase_2 = 9	9	OK
		G_TestCntFA_Phase_3 = 9	9	OK
		G_TestCntFA_Phase_4 = 9	9	OK
		G_TestCntFA_Phase_5 = 9	9	OK

Input set	Input values	Expected output	Observed output	Status
I	Period range: From = ‘01-1997’ To = ‘02-1998’ Modules: GL = ’Y’ AR = ’N’ AP_PO = ’N’ AX = ’N’ FA = ’N’	All counters printed out by the first invocation of procedure “PrintTestCounters” shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_1 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_2 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_3 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_4 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_5 = 16*9	144	OK
		G_TestCntGL_Phase_1 = 16*9	144	OK
		G_TestCntGL_Phase_2 = 16*9	144	OK
		G_TestCntGL_Phase_3 = 16*9	144	OK
		G_TestCntGL_Phase_4 = 16*9	144	OK
		G_TestCntGL_Phase_5 = 16*9	144	OK
		G_TestCntAR_Phase_1 = 0	0	OK
		G_TestCntAR_Phase_2 = 0	0	OK
		G_TestCntAR_Phase_3 = 0	0	OK
		G_TestCntAR_Phase_4 = 0	0	OK
		G_TestCntAR_Phase_5 = 0	0	OK
		G_TestCntAP_Phase_1 = 0	0	OK
		G_TestCntAP_Phase_2 = 0	0	OK
		G_TestCntAP_Phase_3 = 0	0	OK
		G_TestCntAP_Phase_4 = 0	0	OK
		G_TestCntAP_Phase_5 = 0	0	OK
		G_TestCntAX_Phase_1 = 0	0	OK
		G_TestCntAX_Phase_2 = 0	0	OK
		G_TestCntAX_Phase_3 = 0	0	OK
		G_TestCntAX_Phase_4 = 0	0	OK
		G_TestCntAX_Phase_5 = 0	0	OK
		G_TestCntFA_Phase_1 = 0	0	OK
		G_TestCntFA_Phase_2 = 0	0	OK
		G_TestCntFA_Phase_3 = 0	0	OK
		G_TestCntFA_Phase_4 = 0	0	OK
		G_TestCntFA_Phase_5 = 0	0	OK

Input set	Input values	Expected output	Observed output	Status
		All counters printed out by the first invocation of procedure “PrintTestCounters” shall be zero. The second invocation, instead, has to show the following results:	All zeros	OK
		G_TestCntArchAndPurge_Entering = 1	1	OK
		G_TestCntArchAndPurge_Exiting = 1	1	OK
		G_TestCntArchAndPurge_BUs = 9	9	OK
		G_TestCntArchAndPurge_Periods = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_1 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_2 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_3 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_4 = 16*9	144	OK
		G_TestCntArchAndPurge_Phase_5 = 16*9	144	OK
	Period range:			
		G_TestCntGL_Phase_1 = 16*9	144	OK
	From = ‘01-1997’	G_TestCntGL_Phase_2 = 16*9	144	OK
	To = ‘02-1998’	G_TestCntGL_Phase_3 = 16*9	144	OK
		G_TestCntGL_Phase_4 = 16*9	144	OK
		G_TestCntGL_Phase_5 = 16*9	144	OK
L	Modules:	G_TestCntAR_Phase_1 = 0	0	OK
		G_TestCntAR_Phase_2 = 0	0	OK
	GL = ’Y’	G_TestCntAR_Phase_3 = 0	0	OK
	AR = ’N’	G_TestCntAR_Phase_4 = 0	0	OK
	AP_PO = ’Y’	G_TestCntAR_Phase_5 = 0	0	OK
	AX = ’Y’			
	FA = ’N’	G_TestCntAP_Phase_1 = 16*9	144	OK
		G_TestCntAP_Phase_2 = 16*9	144	OK
		G_TestCntAP_Phase_3 = 16*9	144	OK
		G_TestCntAP_Phase_4 = 16*9	144	OK
		G_TestCntAP_Phase_5 = 16*9	144	OK
		G_TestCntAX_Phase_1 = 16*9	144	OK
		G_TestCntAX_Phase_2 = 16*9	144	OK
		G_TestCntAX_Phase_3 = 16*9	144	OK
		G_TestCntAX_Phase_4 = 16*9	144	OK
		G_TestCntAX_Phase_5 = 16*9	144	OK
		G_TestCntFA_Phase_1 = 0	0	OK
		G_TestCntFA_Phase_2 = 0	0	OK
		G_TestCntFA_Phase_3 = 0	0	OK
		G_TestCntFA_Phase_4 = 0	0	OK
		G_TestCntFA_Phase_5 = 0	0	OK

Table B.2 Input sets and outputs comparison

B.2 Volume of records

This section illustrates the testing sessions performed on volumes of records. The results shown here regarded the financial module *General Ledger*[16]. Values were taken from one of the test environments containing a part of all historical data (approximately four years).

Table B.3 indicates the number of records removed from the standard tables of the financial module *General Ledger* for the business unit *Belgium*. The observed values refer to the execution of the archiving program for the time range *"01-1995 → 14-1995"*. The first column indicates the tables involved in the archiving process. For each of them, the table owner "APPS" is specified as prefix. The second column indicates the total number of records observed *before* performing the archiving process. Then, the third column shows the total number of records observed *after* the archiving process. Then, the last column shows the *differences* (absolute value) of the second column from the third one.

Source table	**Totals**		
	Before archiving	After archiving	Difference
APPS.GL_BALANCES	3654711	2733646	921065
APPS.GL_BATCHES	47310	35320	11990
APPS.GL_JE_HEADERS	69800	51346	18454
APPS.GL_JE_LINES	5933726	4430704	1503022

Table B.3 Record volumes – Standard tables (01-1995 → 14-1995)

The same measurements were performed on the archive schema (see Table B.4), that was initially empty (i.e. second column with zeros). The table owner is now "OP_ARCH" to indicate that the calculation refers to the archive tables.

Destination table	**Totals**		
	Before archiving	After archiving	Difference
OP_ARCH.GL_BALANCES	0	921065	921065
OP_ARCH.GL_BATCHES	0	11990	11990
OP_ARCH.GL_JE_HEADERS	0	18454	18454
OP_ARCH.GL_JE_LINES	0	1503022	1503022

Table B.4 Record volumes – Archive tables (01-1995 → 14-1995)

By comparing the last column of each table among each other, the same values must be observed. This essentially means that all records selected and removed from the standard schema are actually inserted into the corresponding tables in the archive schema.

Table B.5 and Table B.6 below show other results, now for the period range *"01-1995 → 01-1995"* (one accounting period only – same business unit).

[16] The underlying concept applies also to the other main financial module *Accounts Receivables*

Source table	Totals		
	Before archiving	After archiving	Difference
APPS.GL_BALANCES	2733646	2649915	83731
APPS.GL_BATCHES	35320	34230	1090
APPS.GL_JE_HEADERS	51346	49666	1680
APPS.GL_JE_LINES	4430704	4292053	138651

Table B.5 Record volumes – Standard tables (01-1996 → 01-1996)

Destination table	Totals		
	Before archiving	After archiving	Difference
OP_ARCH.GL_BALANCES	921065	1004796	83731
OP_ARCH.GL_BATCHES	11990	13080	1090
OP_ARCH.GL_JE_HEADERS	18454	20134	1680
OP_ARCH.GL_JE_LINES	1503022	1641673	138651

Table B.6 Record volumes – Archive tables (01-1996 → 01-1996)

Appendix C

C.1 Data not archived by standard routines

This section shows data-loss details resulting from investigations conducted on the data-aggregation performed by standard archive routines on financial module *Account Receivables*.

Investigations were initially conducted comparing source tables columns with destination tables columns. For a given source table, each column was looked up in one of the three destination tables (AR_ARCHIVE_HEADER, AR_ARCHIVE_DETAIL, AR_ARCHIVE_PURGE_INTERIM). This was done by issuing the following commands:

Command 1

```
SELECT column_name
FROM all_tab_columns
WHERE table_name = <table_name>
AND owner = <table_owner>
```

This command accesses data dictionary information. It provides the list of columns (in this case, their names) for the table `<table_name>` owned by the database user `<table_owner>`.

Command 2

```
DESC <table_name>;
```

This command accesses information related to the definition of table `<table_name>` and returns the definition of its set of columns (name, datatype, etc.).

Command 3

```
a)      SELECT * FROM <table_name>;
b)      SELECT * FROM <table_name> WHERE ROWNUM = n;
```

These commands are ordinary "select" SQL statements used to analyse data recorded in source tables. When needed, a restriction was added (command b) in order to limit the result set to a few rows so as to facilitate analysis on data.

After such first investigations, source code of standard archive routines had been analysed in order to understand how columns in destination tables were actually populated. That is, analysis were carried out in order to track back a destination column to its corresponding source column, regardless of name correspondences.

Then followed a higher level investigation, carried out by running the archive routines on test environments and tracing data. In this case, such activities had been supported by the adoption of the tool “TOAD” (described in section 8.3.2).

When feasible, an automatic control had been arranged by generating output files containing the list of columns and then by parsing them by means Unix commands (e.g. *grep*, *sed*).

Below are listed all source tables for which information indicated on the right are not archived.

Source tables	Information not archived
AR_CORRESPONDENCE_PAY_SCHED AR_CORRESPONDENCES	Invoice and correspondence information related to dunning letters
AR_CASH_BASIS_DISTRIBUTIONS	Information related to cash basis accounting
RA_CUST_TRX_LINE_SALESREPS	Information concerning sales
AR_ACTION_NOTIFICATIONS AR_NOTES AR_CALL_ACTIONS AR_CUSTOMER_CALL_TOPICS	Information related to calls and notification issued to customers.
AR_PAYMENT_SCHEDULES	Information related to detail payment schedules
AR_RATE_ADJUSTMENTS	Currency exchange adjustments
AR_ADJUSTMENTS (where status = ’U’)	Unaccrued adjustments

Table C.1 Information not archived by standard archive routines

References

Sommerville, I. (2001) *Software Engineering 6th edn.* Harlow, UK: Addison-Wesley. (Chs 2, 3, 5, 6, 7)

James D. *et al.* (2002) *Oracle E-Business Suite Financial Handbook..* Berkley, CA: McGraw-Hill/Osborne. (Chs 1, 2, 3, 6, 18)

Cyran M. *et al.* (2002) *Oracle9i Database Concepts, Release 2.* Oracle Corporation. (Chs 2, 3, 11)

Baylis R. *et al.* (2002) *Oracle9i Database Administrator's Guide, Release 2*. Oracle Corporation. (Chs 11, 17)

Burroughs T. *et al.* (2002) *Oracle 9i Database New Features, Release 2*. Oracle Corporation. (Chs 1, 2)

Oracle Corporation (2001) *Oracle Applications Developer's Guide, Release 11i Volume 1.* Oracle Corporation. (Chs 1, 27, 31)

Studdard L. (2001) *Oracle Applications User's Guide, Release 11i.* Oracle Corporation. (Chs 1, 2, 3, 6)

Oracle Corporation (2002) *Oracle Applications Concepts, Release 11i.* Oracle Corporation. (Chs 1, 2, 3)