FUJ00098223

Document Title: Host Applications Database Design and Interface Standards

Document Type: Standard

Abstract: This document defines the standards for the design of all database applications which run on the Pathway Host Systems. It also defines how these applications should interface with one another.

Status: Issued

Distribution: Library
Pathway Programmes Directorate
Pathway Development Teams
Oracle Development Team
A&TC Dublin

Author: Graham Lloyd

FUJ00098223

0 DOCUMENT CONTROL

0.1 DOCUMENT HISTORY

Version	Date	Reason
0.1	2/12/97	First draft containing only section entitled Communicating Between Applications
0.2	19/12/97	Second draft while still incomplete. Produced to get initial feedback from reviewers.
0.3	7/1/98	First draft for full review
0.4	9/2/98	Updated with review comments
1.0	9/3/98	Minor additional updates. Baselined version.
1.1	24/9/98	⇒ Security Section rewritten. ⇒ Section added on the use of shared Oracle libraries ⇒ Section added to cover ad hoc updating. ⇒ Section defining naming standards for sequences added. ⇒ Section added to define the documents which need to be produced fully to describe an application. ⇒ Archive_Type FK added to *Archived_Tables* definition. ⇒ Source type "B" added to *Action_Audit_Trails.* ⇒ <sign> declarative added to list of archive file directives. ⇒ Definition of null in archive data redefined to align with the PAS/CMS implementation.
2.0	9/11/98	⇒ Updated with review comments. ⇒ Definition of how Oracle users are delivered with an application added - CP1558. ⇒ Scope of Oracle auditing extended - CP1555. ⇒ Year 2000 standards added. Baselined on CP1582.
2.1	8/3/99	⇒ A definition of how Oracle users are set up for an application is added to section 15 (PinICL 17350). ⇒ Section 2.21 added specifying the character set to be used. ⇒ Section on the use of public synonyms added (PinICL 20698).
3.0	29/4/99	Baselined on approval of CP1910.

0.2 APPROVAL AUTHORITIES

Name	Position	Signature	Date
Dick Long	Design Manager		
	Development Manager		

FUJ00098223

	Testing Manager		

0.3 ASSOCIATED DOCUMENTS

Ref.	Vers.	Date	Title and Reference	Source
1	1.0	11/6/98	Host Systems Low Level Design Document Specification Pathway ref.: TD/STD/002	Pathway
3	2.0	24/2/98	Access Control Policy Pathway ref.: RS/POL/003	Pathway
4	3.0	3/12/97	Security Functional Specification Pathway ref.: RS/FSP/001	Pathway

0.4 ABBREVIATIONS

A&TC	Application & Technical Consultancy (ICL)
APS	Automated Payments System
BA	Benefits Agency
BES	Benefits Encashment Service
BPS	Benefit Payment System
CAPS	Customer Accounting and Payments Strategy
CBO	Cost Based Optimises
CD	Compact Disc
CM	Configuration Management
CMS	Card Management System
CPU	Central Processor Unit
CTAS	Create Table … As Select SQL construct
DBA	Database Administrator
DDL	Data Definition Language
DFD	Data Flow Diagram
DLR	De La Rue
DLT	Digital Linear Tape
DML	Data Manipulation Language
DW	Data Warehouse
ESBM	Enterprise Scalable Backup Manager
ESNS	Electronic Stop Notice System
ISO	International Standards Organisation
MIS	Management Information System
NINO	National Insurance Number
NUMA	Non-Uniform Memory Access
OBCS	Order Book Control System
OLTP	On-Line Transaction Processing
OO	Object Orientation
OSBM	Oracle Scalable Backup Manager
PAS	Payment Authorisation Service
PC	Personal Computer
PL/SQL	Oracle's Procedural Language extensions to SQL
PMS	Payment Management Service (equivalent to PAS)

FUJ00098223

POCL	Post Office Counters Limited
PQO	Parallel Query Option
RDBMS	Relational Database Management System
RDMC	Reference Data Management System
RM	Royal Mail
RPC	Remote Procedure Call
SLA	Service Level Agreement
SQL	Structured Query Language
SMP	Symmetric Multi-Processor
SRDF	Symmetric Remote Data Facility
TIP	Transaction Information Project
TMS	Transaction Management System
TPS	Transaction Processing System

0.5 GLOSSARY

Abend	A Maestro term for the abandonment of a job because of failure
Application	A self-contained collection of data and code which satisfies all the data processing requirements of a particular corporate function
Database Link	An Oracle term for a named object that describes a path from one database to another.
Encapsulation	Within the Object Oriented approach to software development, encapsulation is the term used for packaging data and methods together such that the structure of the data within an object is hidden from external methods.
Entity	Usually used in reference to the "Entity Relationship Model (ER Model). It generally (though not exclusively) is used to refer to some "real world thing" which is to be modelled by the system.
Functional	Pertaining to the business application using the database
Method	An Object Oriented term for the implementation of a unit of processing.
Object	a) In Object Oriented terms, an object is an instance of an data type or class. It is similar to an entity in that it is concerned with data. However, unlike an entity, an object is concerned also with the methods that manipulate that data. b) In an Oracle context, an object is an element of the schema (e.g. table, index, etc.)
Non-functional	Pertaining to those components of the system, such as backup and recovery, which have nothing directly to do with the application.
Schema	An Oracle term for a collection of logical structures of data or schema objects. A schema is owned by a user and has the same name as that user. There are therefore as many schemas within

FUJ00098223

	an Oracle database as there are users.
Schema Object	A logical data structure within an Oracle database, such as a table, view, synonym, database link, or index.
Synonym	An Oracle term for an alias for a table, sequence or program unit. The aliased can be within ones own schema, within another schema in the same database, or within a schema in a remote database.

0.6 CHANGES FORECAST

None

FUJ00098223

0.7 TABLE OF CONTENT

FUJ00098223

FUJ00098223

FUJ00098223

1. INTRODUCTION

1.1 INTRODUCTION

The ICL Pathway programme is made up of a growing number of applications all of which have the same basic structure. They have a counter layer which runs on the counter PCs at the Post Offices, an agent layer which runs on NT Server systems at the Data Centres, and a host systems layer which runs on large UNIX SMP machines also at the Data Centres.

In order to minimise development, support, and maintenance costs, it is essential that all these applications have the same basic structure. In other words, where there are requirements for functions such as exception handling or archiving, these functions must be performed in the same manner in all applications. To achieve this, it is necessary for Pathway as an organisation to impose a set of standards for the implementation of the database aspects of applications at the host layer, and then to ensure that those standards are adopted by all application developers.

This document satisfies the first of these requirements in that it defines the standards. To ensure that the standards are adopted, Pathway must have within it a central administration function whose responsibility it is to police these standards.

It is intended that these standards should be applied to all host applications that are developed by Pathway, whatever RDBMS is used to implement them. At present, Pathway's strategic RDBMS for these applications is Oracle. Consequently, in this document, where it is necessary to specify technical details or to give examples, Oracle terminology is used. If it is decided to use an RDBMS other than Oracle for an application, then the principles behind the standards should still be applied, even if the detail is not in all cases appropriate.

The standards defined are, where appropriate, loosely based on those used for the PAS/CMS Oracle application.

1.2 SCOPE

This document applies only to the design of the host system applications that are implemented on the Pathway Sequent Servers. It does not consider the requirements of the TMS layer or the Counter layer. Nor does it consider how the individual modules within a host system application are specified at the lower level. Although the application design principles defined within this document do have an effect upon how the individual modules are designed, the standards to be used for specifying these modules are defined in a different document (ref. [1]).

The standards specified herein apply to all current and future releases of the total ICL Pathway solution.

Further information about exactly how an application should be specified and documented, and about how this particular document fits into the overall hierarchy of standards documentation, can be found in the Pathway On-Line Standards.

FUJ00098223

2. DESIGN PRINCIPLES

2.1 USE OF REPOSITORY

All Pathway host applications, whether based on Oracle or not, must be defined and fully documented within a central Designer/2000 repository. This is necessary so that the development of all applications can be controlled, monitored and managed. More detailed information about the objects and properties which must be defined within the repository can be found in section 4.

2.2 APPLICATION DESIGN

An application is a self-contained collection of data and code which satisfies all the data processing requirements of one or more major corporate functions. Within Pathway there are a growing number of applications all of which have similar security, operational, support and manageability requirements. It is logical therefore that they should all be constructed in a similar manner. This reduces development, support and maintenance costs, and minimises the risks of a database becoming corrupt due to mistakes being made by support staff.

To achieve this aim, all applications must be designed using a standard set of framework tables.

The framework tables are designed to handle:

- application control;
- exception reporting;
- the auditing of:

  ⇒ batch processes;

  ⇒ file loading and unloading;

  ⇒ unplanned database access by support staff, and;
- archiving.

These framework tables are discussed in detail in sections 5, 6 and 8.

2.3 NAMING STANDARDS

Organisation-wide naming standards are essential for any organisation which aspires to have control over the development of its database applications. This is necessary so that there is no confusion within that organisation when particular terms are used to describe particular objects, attributes or activities.

The naming standards to be adopted by all Pathway applications are defined in section 3.

FUJ00098223

2.4 APPLICATION SELF-CONTAINMENT

It is important that object-orientation as a concept is embraced by Pathway at the application level. This means that all applications must be designed to be self-contained, or encapsulated in OO terms. This is necessary because if applications are not kept completely separate from one another, developing one application would inevitably result in any interfacing applications also having to be further developed, making the whole development and testing process unwieldy and practically unworkable.

Encapsulation at this level means that communication between applications is only allowed via pre-defined, documented interfaces which are in no way dependent on the applications' physical implementations. At the logical level, one application must not be dependent on how another application is implemented. This means that physically applications should be able either to co-exist within the same database, or to exist in separate databases, without this affecting the code within the applications.

Performance, operational and management considerations alone should then be used to dictate how individual applications are physically implemented. More details on this can be found in section 10.

A detailed description of how interfaces between applications should be developed can be found in section 9.

2.5 SCHEDULING

Within an organisation of Pathway's size, it is essential that batch jobs are scheduled reliably. If this does not happen the organisation is at risk of missing deadlines, or worse, if jobs are inadvertently run in the wrong order, of corrupting databases.

The only way to reduce this risk to an acceptable level is to use a robust workload management product to schedule all batch jobs. This product should ideally be able to:

- specify the sequences in which batch jobs should be run via both scripts and a graphical user interface;
- resolve inter-dependencies between jobs running on the same, and different, machines;
- launch and track each job;
- limit job concurrency;
- raise alerts if jobs fail in such a way that operator intervention is necessary, and;
- allow network administrators to control all clients on the network from a single graphical management console.

Within Pathway, the strategic scheduler to be used is Maestro, and it is this product which should be used to schedule all jobs within all the applications that Pathway operates. Consequently all host application modules must be written such that they are compatible with the requirements of Maestro.

Maestro must be used for all scheduling activities within an application. Application modules must not be written to usurp any such activities.

FUJ00098223

2.6 SUPPORTABILITY

2.6.1 THE REQUIREMENT FOR PASSIVE SUPPORT

No database application should require active support from operations or database support staff. This means that the application must not require any person to log into the database to update application data other than via scripts or forms which have been developed as an intrinsic part of the application, and have been delivered via CM.

The database must be designed such that it is capable of flagging up to support staff, via Patrol alerts, any problems detected within the application, or within the database engine itself. It should not require support staff to have to hunt out errors themselves.

More details on this can be found in section 6.

2.6.2 MONITORING DATABASE AND APPLICATION ACTIVITY

All database and application monitoring must be undertaken using the standard query tool for the platform. For the Oracle databases on the Host Systems this tool is Oracle's Discoverer.

2.6.3 INTERACTIVE USE OF ADHOC DATABASE ACCESS PRODUCTS

Ad hoc, interactive access of the live databases using sql*plus or equivalent low-level tools is not allowed. This is because the use of these products is:

- unauditable;
- can result in the corruption of live data;
- can impact the performance of production applications.

The methods that application designers should use to ensure that these tools can not be used are defined below.

- No users or roles should be defined which allow on-line access, unless they are part of the functional requirements of the application
- Updating the database should be allowed only via pre-defined access routes such as via stored procedures that a particular user has access to.
- Except for the database administrator, no user should exist which has default update privileges on all tables in the database.

2.7 PROGRAMMING LANGUAGES

The first choice of language for all server-based modules written for Pathway applications should be 'C++' with Pro*C being used for the interface with Oracle. Only when 'C++' can not easily be used, such as in database triggers, should consideration be given to using pl/sql or other languages.

'C++' is used in preference to other languages because:

- it is a mature language which is reliable and widely known;

FUJ00098223

- it provides the best performance for interfacing with Oracle, as well as with most other RDBMSs;
- other Pathway applications are written using it, so there is a knowledge base for the language within Pathway;
- it is portable between different RDBMSs.
- It is a "strongly typed" language that catches (at compile time) many of the errors that its "parent" 'C' language would only find in test or live.

Unless Class's simplify the implementation of a program, programs should be coded so that they can be compiled by either 'C' or 'C++' compilers. All Oracle interface modules should be coded in 'C' to take advantage of Pro*C's native support for 'C' structures and parameters.

2.8 DEFENSIVE PROGRAMMING

As a general principle all applications must be coded defensively. This means that there must be no assumptions made within an application about the quality of the data which it receives from other applications within Pathway or, more importantly, from external sources. All code should be written with the expectation that any exception condition which could conceivably be foreseen will actually occur at some point in the lifetime of the application.

If defensive programming principles are not adopted across the board within Pathway there is a significant probability that corrupt data will find its way onto one or more application databases, resulting in programs crashing and significant downtime.

The principles for defensive programming which should be adopted are as follows:

- All data input into an application's schema must first be validated to ensure it complies with the validation rules specified in the appropriate Application Interface Specification.
- The outcome of any SQL statement or RPC call must be explicitly checked for any sort of exception condition after each invocation.
- All modules must write away any exceptions which they encounter to "alerted" exceptions tables within the application's schema as described in section 8.
- All modules should be written so that they can "abend" to Maestro should they detect fatal errors.
- All modules should be written in such a way that they can be automatically restarted after any type of failure, notwithstanding database corruption, without any sort of manual intervention.
- One should always assume that there are residual faults within an application, even after it has been fully tested. Therefore, for critical information, consideration should be given to cross checking record counts or 'file' totals using an alternative processing path.
- All modules should be written to use structured exception handling:

FUJ00098223

⇒ "try, throw, catch" in C++ programs.

⇒ "DO_TRY, DO_THROW, DO_CATCH, DO_FINALLY" in C programs.

⇒ "BEGIN, RAISE, EXCEPTION" in PL/SQL.

⇒ "ON ERROR GOTO :label" in Visual Basic.

2.9 PERFORMANCE

The host applications are currently all implemented on Sequent SMP or NUMA machines. To obtain the best performance from these machines, it is necessary to code applications in such a way that they make best use of the multi-processor architectures involved. This involves using parallel SQL constructs where appropriate and also to make use of other techniques to optimise performance where circumstances so dictate.

This subject is discussed in more detail in section 11.

2.10 AUDITING

Pathway has contractual requirements to audit all events which occur on its databases and to ensure that the audit trails so produced can not be tampered with. The method by which this is achieved is described fully in section 7.

2.11 ARCHIVING

Internal and external auditors have a number of requirements to retain application data beyond the time when it is operationally necessary to retain that data on Pathway's databases. This data is archived to off-line media in the manner described in section 7.

2.12 HANDLING OF REFERENCE DATA

Nearly all applications require some reference data, or standing data as it is sometimes called, to enable them to function. Reference data is usually contained within an application's own schema in tables which contain a relatively small number of rows. The uses to which reference data can be put are varied. Examples of reference data are:

- Lists of codes together with their descriptions. These are usually used for expanding codes to meaningful descriptions when the data is displayed on a screen or in a report.
- Lists of values for validation where the values can change. If validation values are fixed, then it is more usual to allow the RDBMS to validate them using check constraints. If they can be changed, then the application carries out the validation using a list of values in a reference data table.
- Calendars.
- System and application operational data. E.g. File transfer destinations, Post Office addresses.

FUJ00098223

Reference data can either originate from requirements which are external to the application or from those which are internal. An example of external requirements would be Post Office addresses which must be used on orders for benefit cards which are sent to De La Rue from the CMS database. An internal requirement could be a list of validation values to be displayed on a screen. In order to safeguard the integrity of all the Pathway applications, it is essential that the loading of reference data of both types into an application's schema is closely controlled.

Within Pathway, all external reference data must be provided by way of the RDMC database. In this way, it can be guaranteed that all the data provided has gone through a stringent Change Control process and has been successfully validated.

For the internal reference data, if updates are expected to be frequent, the changes should ideally be made via a pre-defined form which has been designed specifically for the purpose. As well as performing the update, this form would ensure that the change is audited in the way described within section 7.2.5. If the updates are very infrequent, or are linked to a particular release of a product, then it is acceptable to make the changes using a tested, audited SQL script which has been delivered via CM.

2.13 BACKUP AND RECOVERY

All host applications must be regularly backed up such that there is no realistic possibility of data ever being lost should a media or system failure occur.

The method of achieving this for all types of host applications is described fully in section 12.

2.14 RESILIENCE

All the applications developed by Pathway must be resilient to media and site failure. The way this is to be achieved is described fully in section 13.

2.15 CONSISTENCY CHECKING

Routines which check the logical and physical consistency of all the data held within an application's schema must be implemented for all host applications. This subject is expanded upon in section 14.

2.16 SECURITY

All Pathway applications must be designed to be secure against unauthorised access. This is achieved by:

- enforcing explicit access control requirements for all database users;
- ensuring that links to the outside world are adequately protected.

More detailed information can be found in section 15.

FUJ00098223

2.17 MIGRATION

When an application is upgraded from one release to another, any associated migration of data between schema versions should be planned such that there is no requirement to unload and reload live data from the database. Instead SQL scripts should be developed to move the schema forwards. The upgraded version of the application must therefore be written such that it can operate on data which was in existence before the schema changes were made, as well as after.

The design documentation produced as a consequence of upgrading an application should not only contain comprehensive information on how the upgrade should be carried out, but also information specifying how the migrated schema can be regressed back to what it was, should the new version of the application not function as expected.

2.18 TIMEKEEPING

Applications must be designed such that their day to day operations are independent of the system clock. Instead they should use a logical clock which is maintained by modules scheduled by Maestro.

The system clock should only be used within an application for applying timestamps to records, reports and interactive screens.

2.19 USE OF DDL

Although necessary on occasion for performance reasons, the use of DDL within normal Oracle application modules should be avoided. Oracle DDL commands, such as TRUNCATE, which removes all rows from a table without recording before images in rollback segments, or the unrecoverable version of CREATE TABLE … AS SELECT… (CTAS), should not be used without due consideration for the consequences should the commands fail. This is because the effects of these commands can not simply be recovered by the RDBMS.

If, despite this, it is decided that the performance gains of using DDL commands are such that the benefits greatly outweigh the disadvantages, then the modules which use these commands should be designed such that, if they fail at any stage and have to be restarted, there is no possibility of data ever being lost or of the database becoming corrupt in any way.

2.20 YEAR 2000 COMPLIANCE

The year component of all dates used within an application must be stored within the database as a four digit number. It must also be displayed as a four digit number on all interactive forms and written as a four digit number on all printed reports.

If two-digit years are passed to an application, the date inferencing rules are the same as those used for the Oracle RR date format element. These rules are summarised below.

FUJ00098223

		If the specified two-digit year is:	
		0-49	50-99
If the last two digits of the current year are:	0-49	The return date is in the current century.	The return date is in the century before the current one.
	50-99	The return date is in the century after the current one.	The return date is in the current century.

2.21 CHARACTER SETS

All host system databases must use the ISO 8859-1 West European character set (Oracle acronym WE8ISO8859P1). This character set ensures that there is no confusion between the "#" and "£" symbols.

FUJ00098223

3. NAMING STANDARDS

3.1 GENERAL STANDARDS

Within the database environment there are two important requirements for all names given to objects, attributes and modules. They must be:

1. as meaningful as possible, and;
2. as short as is practicable.

This is necessary so that the schema definitions, the written code, and the documentation are all manageable and easily understood. The problem with trying to achieve this goal is that the two aims are incompatible, an incompatibility which grows with the size of an organisation. If uncontrolled, this inevitably leads to the same names being used to mean completely different, or, more dangerously, subtly different, things, often within the same application. The only way for an organisation to get control of the way objects and attributes are named is to impose organisation-wide naming standards.

At a high level of abstraction, there are some general standards which can be applied to all names that are adopted by Pathway. These general standards are defined below.

- Names should contain only recognised words or standard abbreviations.
- Names should contain only alphabetic and numeric characters, and the underline character.
- All words or abbreviations within a name should be separated using the underline character.
- Within an application's own schema, the application itself should *not* be identified in names (by the use of a prefix for example), unless that application explicitly uses a sub-set/specialisation of an entity. E.g. "Products" is defined within the RDMC, but the TPS subset is called "TPS Products".
- Once a name has been chosen for an object or an attribute, only that name should ever be used within the application for that object or attribute. E.g. If an event timestamp attribute is defined in the *Card_Events* table with a name of Event_Tsmp, then that column must have the same name on all the other *XXX_Events* tables within the application.

3.2 APPLICATION NAMES

Applications are self contained units of data and code which perform a single corporate function. Although frequently a single application runs in a single database, this is not a requirement and is often undesirable (see section 10). Each application within the Pathway domain should have a single schema which contains all the objects which are intrinsically used by that application.

FUJ00098223

The name used for identifying an application is specified within the Designer/2000 repository. It should be a short and meaningful name and contain within it no release or version information.

Each application must have applied to it a three character alias which is defined on the Designer/2000 repository within the application's Notes attribute. This alias is used only by distributed applications which require access to the defined application. It is not to be used as a prefix or suffix within the application's own schema.

3.3 MODULES

It is not recommended that names which attempt to describe the function of that module are used for its naming. This is primarily because it is essential for scheduling and the management of work to keep module names short. It is also because it is usually very difficult to describe a module's function meaningfully in a short name.

As a consequence, all Module names should be 7 characters in length. The first three characters must be the application name (or the sub-application if the application is logically broken down further), the fourth character defines the type of module (see below), the last three characters are a three digit number which uniquely identifies the module within the application.

Fourth Character	Module Type
C	Module written in C or C++
F	Developer/2000 Form
J	Module written in Java
L	SQL loader module
M	Developer/2000 Menu
P	Patrol Knowledge Module
R	Report
S	PL/SQL procedure
V	Visual Basic
X	UNIX Script

As an example, the "C" PAS/CMS module which validates cardholder data is called CMSC103.

3.4 TABLES

3.4.1 GENERAL RULES FOR TABLE NAMES

All database tables should have meaningful, plural names.

3.4.2 TABLE NAMES

Type	Description	Standard
Simple	A normal table in the local database which	A meaningful plural name less than

FUJ00098223

Type	Description	Standard
Functional Table	is part of an application	16 characters in length. E.g. Cardholders
Simple Non-Functional Table	A normal table in the local database which is not part of an application.	If it is a framework table defined in this document, then the name must be as specified herein. Otherwise as for simple functional tables.
Partitioned Table	An application table which has been partitioned using the application software rather than the RDBMS.	A meaningful plural name less than 16 characters in length followed by _N(x). E.g. *Encashments_3*
Flip-Flop Table	One of a pair (or possibly a trio) of application table which is flipped to and flipped back from, usually by means of Create Table As Select.	A meaningful plural name less than 16 characters in length followed by _A(x). E.g. *Payments_B*
Transient Interface Table	An interface table which contains data for a particular period of time.	A meaningful plural name of 16 characters or less in length followed by _YYYYMMDD_NN, where YYYYMMDD is the creation date and NN is an optional numeric second suffix if more than one version of this table can be created in a particular day. E.g. *Mis_Dw_Cda_19971124*

3.4.3 TABLE ALIASES

Every table within an application should have a three character alias assigned to it. This alias is used when naming a table's primary key, foreign key and check constraints.

3.5 CONSTRAINT NAMES

3.5.1 PRIMARY KEY CONSTRAINTS

A primary key constraint for a table should have a name of the form:

XXX_PK

where XXX is the three character alias of the table for which this is the primary key, as described in section 3.4.3, and _PK is a standard suffix defining the name as a primary key constraint.

For example, a primary key on the PAS/CMS *Cardholders* table is named: CDH_PK.

3.5.2 FOREIGN KEY CONSTRAINTS

A foreign key constraint should have a name of the form:

XXX_YYY_FK

FUJ00098223

where XXX is the three character alias of the table on which the foreign key is declared, YYY is the three character alias of the table for which this is the primary key, and _FK is a standard suffix defining the name as a foreign key constraint.

For example, a foreign key on the PAS/CMS *Cardholders* table linking it to the *Card_Types* table is named: CDH_CDT_FK.

3.5.3 CHECK CONSTRAINTS

A check constraint should have a name of the form:

XXX_column_name_CHK

where XXX is the three character alias of the table on which the check constraint is declared, column_name is the name of the column on which the check constraint acts, and _CHK is a standard suffix defining the name as a check constraint.

For example, a check constraint on the of_interest column within the PAS/CMS *Cardholders* table is named: CDH_of_interest_CHK.

3.6 SEQUENCES

The Oracle RDBMS includes within it a sequence generator for generating sequential numbers. This is mainly used for the automatic generation of unique primary keys.

Oracle sequences for the generation of unique primarykeys should have a name of the form:

XXX_SEQ

where XXX is the three character alias of the table on which this sequence is used, as described in section 3.4.3, and _SEQ is a standard suffix defining the name as a sequence.

For example, a sequence used for generating unique primary key values on the PAS/CMS *Beneficiary_Events* table is named: BEV_SEQ.

If sequences are used for other purposes, the name should have the form:

XXX_YYYY_SEQ

where XXX and _SEQ are as for primary key sequences and YYYY is a variable length descriptive string.

3.7 SYNONYMS TO OBJECTS IN OTHER DATABASES

Objects in remote databases can be accessed either directly using the database link name as a qualifier as in:

SELECT * FROM calls@pascms;

where pascms is the name given to the database link, and “calls” is the name of the table on the remote database. Alternatively a synonym can be defined on the local database.

FUJ00098223

To satisfy the requirements of encapsulation the direct method should never be used by Pathway applications. Remote objects should always be accessed using synonyms. Synonym names should have the format:

AAA_meaningful_name

where AAA is the three character application alias as described in section 3.2, and meaningful_name is a short, meaningful, plural name.

3.8 STANDARD EXPRESSION ABBREVIATIONS

All RDBMSs impose upper limits on the sizes of names which can be defined by the user for objects and their attributes within the database. In most cases, these limits are sufficient for meaningful names to be defined without the use of abbreviations. For instance the Oracle limit of 30 characters for table names would appear to be sufficient for most requirements. However, although allowed, names of this length are unwieldy to use and can make code and schema definitions difficult to understand. In practice, most designers and programmers would prefer that names did not exceed about sixteen characters in size. Also the use of prefixes and suffixes, some of which have been proposed in the above sections, further restrict the number of characters which can be used for the meaningful part of the object names.

With these de facto limitations to name sizes, comes a requirement for the use of abbreviations for all but the shortest of words which could be used within object or attribute names. For these abbreviations to be understandable by everyone within Pathway and to avoid confusion whereby the same abbreviations are used by different people to mean different things, it is necessary for the same abbreviation to be used for the same word in all cases within all applications. To achieve this, a centrally maintained register of standard abbreviations and their meanings must be implemented by Pathway. As an example the table below is the standard abbreviation list for all the tables and columns used within this document.

Abbreviation	Full Word
Appl	Application
Desc	Description
Distr	Distributed
Excptn	Exception
Id	Identifier
Info	Information
Msg	Message
Obj	Object
Seq	Sequence
Tsmp	Timestamp

FUJ00098223

3.9 NAME TYPES

Names used within applications can often be grouped together into types. Examples of types are: dates, times, timestamps, identifiers, sequence numbers, descriptions.

As with abbreviations, it greatly simplifies schema and code comprehension if standard methods are used for naming objects or attributes which are of the same type.

It is recommended that the type, which can itself be an abbreviation, is used as a suffix to the object or attribute, this suffix following after an underline character.

Again, it is necessary for Pathway to maintain a central register of the standards used for naming types. As an example the table below is the standard abbreviation list for all the types used within this document.

Suffix	Type
_code	codes which are used to simplify processing
_count	Numeric attributes holding counts
_date	Date attributes which do not have a time component
_desc	Descriptions
_excptn	Application exception tables
_id	Identifiers
_name	Peoples names
_seq	Sequence numbers
_status	Status information
_time	Time attributes which do not have a date component
_tsmp	Timestamp attributes which have both a date and time component
_type	Type fields

FUJ00098223

4. APPLICATION DOCUMENTATION

4.1 DOCUMENTS

All host system applications must be fully documented at all stages of development. This documentation should be made up of Word documents and an application within a Designer/2000 repository.

The list below defines the documents that should be produced to describe the application:

- Analysis Report
  Outlines the functional requirements for the development of the application.
- High Level Design
  Describes how the application is designed to meet the functional and non-functional requirements.
- Access Control Matrix
  Defines the access control policy for the application (see section 15.2.2). Where feasible, this document should be generated directly from the data held in the Designer/2000 repository.
- Application Volumes
  Specifies the detailed volumetrics information used as the basis for the physical design of the database. Where feasible, this document should be generated directly from the data held in the Designer/2000 repository.
- Physical Database Configuration
  Describes how the application's database is physically configured onto disc.
- Installation Guide
  Describes how the application should be delivered, installed and initialised.
- Application Interface Specification(s)
  One of these documents must exist for each external or internal interface to the each application. The document describes in detail all the physical and logical aspects of the data that is transferred across the interface.

All other documentation should be contained within the Designer/2000 repository.

4.2 THE REPOSITORY

All the objects, together with their properties, which need to be defined for a particular application, can be specified within the Designer/2000 repository. The central software toolset within Designer/2000 is the Repository Object Navigator. This toolset provides, for an application, an object hierarchy window and a properties window. The object hierarchy window tabulates all the objects which can be defined for an application. The properties window allows the properties of an object highlighted in the object hierarchy window to be defined.

FUJ00098223

Although there is scope within Designer/2000 to specify large numbers of objects and properties for a particular application, there is only a subset of them which are essential for the logical systems design of an application which is to be implemented by Pathway. This core of systems design objects and properties is defined below.

4.3 DOCUMENTATION OF DESIGN DECISIONS

Not only is it necessary to define all the schema objects (i.e., tables, views, indexes, etc.) of an application within the repository; it is also necessary that the design decisions behind those definitions are also documented. This is essential so that any future changes to the application are made with the full knowledge of why the application was designed the way it was in the first place.

The "Notes" property within each repository object type should be used for this purpose.

4.4 DATA DEFINITION

4.4.1 MODULES

Each module within the application must be defined as a separate object. Within the module object, the properties which must be specified are described in the table below.

Property	Description	Example
Application System	The name of the application system that owns the module	CMS
Short Name	The short name of the module	CMSC502
Name	name of the module	Process card exceptions
Purpose	The purpose or short description of the module.	To process card exception files sent from De La Rue to CMS
Language	The language in which the module is written	Pro*C
Type	The type of module	Function
Complexity	The complexity for this module	Average
Command Line	The command line used to invoke this module from the operating system	<MODULE> <UN>/<PW>
Description	This section should contain a reference to the Low level Design documentation (see ref. [1]) held within PCMS.	*PCMS Document Reference*
Notes	A version history of changes to the module under the headings: Version No, Reason, Who, Date.	History of versions 0.1, 0.2 and 0.3 of CMSC502

4.4.2 TABLES

All database tables used within the application must be accurately defined. The table below specifies only the logical properties which must be specified. For the successful

FUJ00098223

generation of schema DDL, a number of storage parameters, which are omitted from this table, also need to be specified. These storage parameters are defined using the standard Designer/2000 property elements.

Property	Description	Example
Application System	The name of the application system that owns the table	CMS
Name	The name of the table	*CARD_EXCPTNS*
Alias	A unique three character alias for the table name (as described in 3.4.3)	CDX
Display Title	The title to be displayed when an item based on this table's data is generated	Card Exceptions
Volumes: End Rows	An estimate of the maximum number of rows that the table will hold	100,000
Description	A brief description of the table	Contains a row for each card exception raised
Notes	Documentation of design and storage decisions	Records are only ever inserted into this table. Pctfree therefore set to 1.

4.4.3 VIEWS

Each view used within the application must be defined as a separate object. Within the view object, the properties which must be specified are described in the table below.

Property	Description	Example
Application System	The name of the application system that owns the view	PMS
Name	The name of the view	*V_PAYMENT_PAYEES*
Alias	A unique three character alias for the view name	VPP
Select Text	The SELECT statement that specifies the base tables and columns from which the snapshot is derived	PPY.PAYMENT_ID, BEN.FIRST_NAM, …….
Where/Validation Condition	The condition that qualifies the view	WHERE PAYEE_NINO = BENEFICIARY_NINO
Description	A brief description of the view	View of payment payee personal details
Notes	Documentation of design decisions	To simplify access to payees by module HLPF102 for Help Desk Encashments

4.4.4 COLUMNS

Subordinate to both the Tables and Views objects within the object hierarchy is the Columns object. The properties which have to be specified for this object are defined below.

FUJ00098223

Property	Description	Example
Table	Display only. The name of the table or view for which the columns are to be defined	*CARD_EXCPTNS*
Name	The name of the column	APPL_EXCPTNS_CODE
Sequence	The create sequence of the column within the table, view	8
Datatype	The standard Oracle datatype for the column	VARCHAR2
Average Length	The average length of data contained in this column	7
Maximum Length	The maximum length of the data that the column can contain	7
Optional?	Indicates whether the column can contain null values	TRUE
Description	A brief description of the column.	Code identifier of the exception

4.4.5 OTHER OBJECTS SUBORDINATE TO TABLES AND VIEWS

For every table specified for an application, there are likely to be subordinate objects other than columns, which, if they exist, must also be specified if the table is to be defined fully. These include primary and foreign key constraints, and synonyms. These objects are not described fully here as Pathway has no special requirements for their definition.

4.5 SCHEMA GENERATION

If the application is Oracle-based, the SQL DDL for the application must be generated directly from the Designer/2000 repository. This is necessary to guarantee the integrity of the data definitions used within the application's schema.

4.6 DIAGRAMS

A data diagram for an application must be drawn within the repository using the Data Diagrammer toolset.

4.7 USER DEFINITION

Every category of Oracle user that can have access to an application must be defined within the repository using the Oracle Database Users object type. Within this definition, all the database objects to which the user has been granted access permissions must be defined using the Database Object Grants property. The information inserted here must match that defined in the application's associated Access Control Matrix (see section 15.1).

If a user is to be granted any specific system privileges, these must be specified in the repository using the System Privilege Grants property.

FUJ00098223

5. APPLICATION CONTROL

5.1 OVERVIEW OF AN APPLICATION'S FRAMEWORK TABLES

Although each of the host applications implemented by Pathway satisfies a discrete business requirement, the overall structure of each of these applications is fundamentally the same, in that they have broadly similar requirements for active support, auditing, and exception handling. To simplify the support and maintenance tasks, it is therefore logical that each application should use the same basic mechanisms for controlling how the application operates. To achieve this each host application must have defined within it a common set of tables which are specifically designed to support this.

These tables fall into three categories

- Those that control the operation of the application.
- Those that contain audit and archive data. These are defined in more detail in section 6.
- Those that control exception handling. These are defined in more detail in section 8.

These tables are illustrated in the diagram below.

FUJ00098223

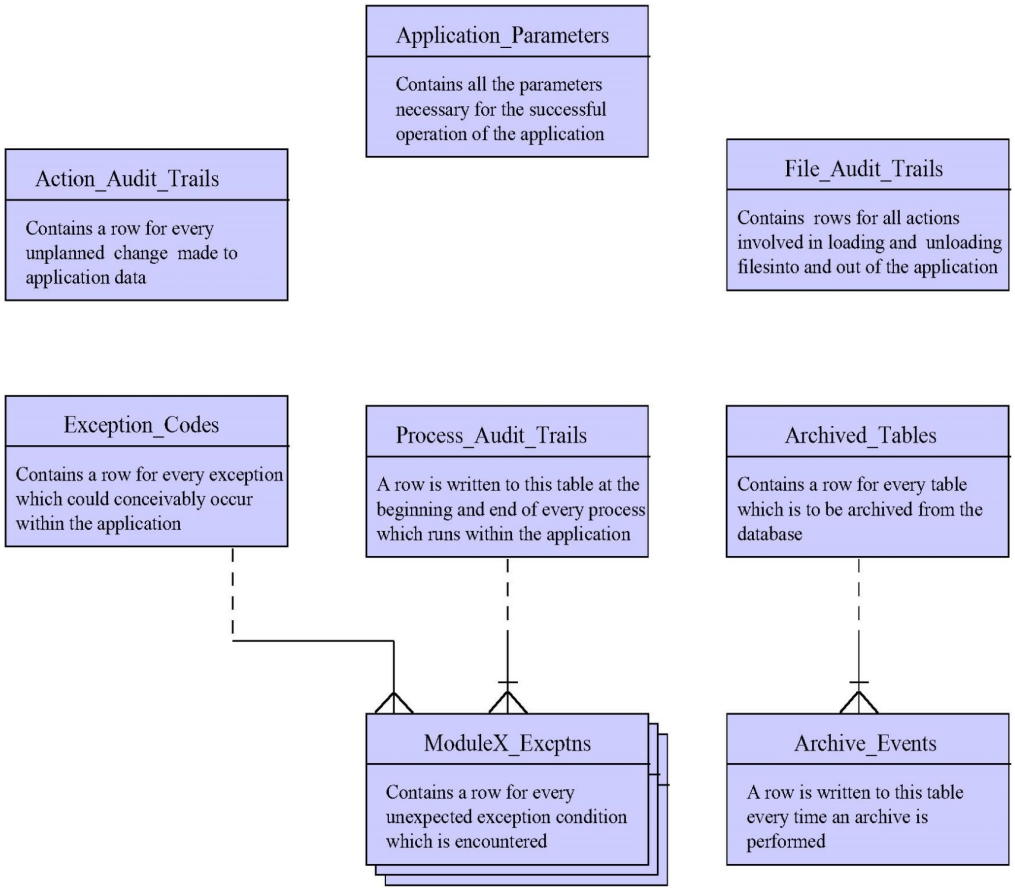

Figure 1- An Application's Framework Tables

All applications require an *Application_Parameters* table, a *Process_Audit_Trails* table and an *Exception_Codes* table. If ad hoc access to the application's data is allowed, there must also be an *Action_Audit_Trails* table; if files are loaded into or unloaded out of the database, there must be a *File_Audit_Trails* table, and if archiving is required, *Archived_Tables* and *Archive_Events* tables must be defined. A *ModuleX_Excptns* table also needs to be defined for every module, or group of related modules, which could conceivably fail for any reason.

5.2 OPERATIONAL PARAMETERS

5.2.1 SYSTEM PARAMETERS

System parameters define how a database operates. They exist within a single table within the database called *System_Parameters*. This table has a public synonym applied to it enabling all users to select records from it.

The table below defines the *System_Parameters* table.

FUJ00098223

System_Parameters			
Column name	*Null?*	*Datatype*	*Description*
Parameter_Name	N	varchar2(30)	The name of the parameter *Primary Key*
Parameter_Type	N	varchar2(1)	The datatype of the parameter. C Character string D Date in the format: DDMMYYYY_HH:MM (where _HH:MM is optional) N Number S SQL Script
Parameter_Date	Y	date	Contains the value of the parameter if Parameter_Type is D.
Parameter_Number	Y	number	Contains the value of the parameter if Parameter_Type is N.
Parameter_Text	Y	varchar2(1500)	Contains the value of the parameter if Parameter_Type is C or S.
Parameter_Desc	Y	varchar2(50)	A description of the parameter

Examples of the types of parameters which may be defined in this table are;

- system directory locations;
- table STORAGE values for dynamic SQL CTAS scripts;
- the degree of parallelism to be used in parallel hints. This assumes that dynamic SQL is to all hints to be configured at run-time;
- backup/recovery data.

5.2.2 APPLICATION PARAMETERS

Application parameters define how a particular application operates. There is one of these tables for each application running on the database. The table has the name *AAA_Application_Parameters*, where AAA is the application name. It exists within the application user's schema and is defined in exactly the same way as the *System_Parameters* defined above.

Examples of the types of parameters which may be defined in this table are

- default directory locations for files to be loaded or unloaded into or out of the database;
- processing dates;
- maximum and minimum values to be used in application processes;
- index creation scripts;
- retention periods;
- rollback segments to use for specific processes.

FUJ00098223

5.2.3 UPDATING PARAMETERS

A standard method must be provided to allow systems administrators to amend values within the system and application parameters tables. Whatever method is chosen to achieve this, it is essential that, as well as updating the appropriate tables(s), it ensures that a record of the change is written to a log table within the database. This table is called *Action_Audit_Trails* and is described fully in section 7.2.5.

5.3 PUBLIC SYNONYMS

Public synonyms must not be created for any objects that are contained within an application's schema. Although the provision of public synonyms for an application's objects simplifies the access of those objects from users that are used for support, their existence prevents multiple applications from defining objects with the same name, and thus prevents them from running in the same database instance.

FUJ00098223

6. UNPLANNED ACCESS TO A LIVE DATABASE

6.1 REQUIREMENT FOR STRUCTURED ACCESS

In an ideal world in which all applications are designed perfectly and the database never fails, it would not be necessary for support or operations staff ever to have unplanned access to a live database. This, however, is a forlorn aim. Even in the most well designed of systems, support staff need to access the database to investigate exceptions that have been reported by applications, to follow up on errors reported by the RDBMS, or to amend application data corrupted by incorrect code or by invalid data input from external systems.

In an environment where a database does not have associated stringent availability, integrity, security and auditability requirements, it is often acceptable for this type of support to be undertaken using ad hoc tools such as sql*plus together with a library of scripts which most database administrators have in their personal toolboxes.

In that sort of environment, it has to be accepted that an ill-advised action by a member of the support staff might result in impaired performance or even corruption of live data. It is a risk which can be chosen to be taken by an organisation. It is not however, a risk which can be taken by Pathway, upon whom swingeing financial penalties are applied when SLAs are missed.

Full details of Pathway's policy for access control can be found in ref. [2].

6.2 DISCOVERER FOR AD HOC ENQUIRIES

Within Pathway, the generic requirement to perform ad hoc enquiries on a database is not only limited to applications and database support staff. Auditors and fraud investigators may also need to enquire on a database on an ad hoc basis. These different classes of people all have their own requirements for accessing the data, each of which could be satisfied by the provision of a number of bespoke enquiry functions. This solution, however, would be difficult to implement and would be prohibitively expensive because of the large number and diversity of the enquiries involved.

The ad hoc enquiry requirement is therefore satisfied by the provision of a generic tool which is designed specifically for the purpose. For Pathway's host applications, this tool is Oracle's Discoverer product.

Discoverer has a two layer architecture. There is a meta layer which runs on the server that provides a logical view of the data to be presented to the client, and there is a client layer which runs on the PC that presents the data in the format required by the end-user.

One of the main advantages of using Discoverer is that access to data within an application can be controlled. It is, for instance, possible to prevent end-users from doing enquiries which will tie up the server for long periods of time when important live processes are being run.

FUJ00098223

6.3 AD HOC UPDATING OF NON-FUNCTIONAL DATA

6.3.1 FORM BASED UPDATING

6.3.1.1 UPDATING APPLICATION PARAMETERS

There should be many fewer reasons for updating data within a database on an ad hoc basis than there are for enquiring upon it. If this is not the case, then it is probably because the functional implementation of the application is incomplete. The only data which should have to be updated on an ad hoc basis are records within the *Application_Parameters* table, or within other control tables as a consequence of manual recovery actions being taken.

It is obviously more important than for enquiries for this type of access to be closely controlled. It is essential therefore that a change to *Application_Parameters*, for example, is properly authorised, is validated and is audited. To do this it is recommended that a bespoke generic screen-based form is developed which will satisfy all these requirements.

The simplest method of achieving this is to use an Oracle Developer/2000 form. A typical screen to amend the *Application_Parameters* table might appear as below.

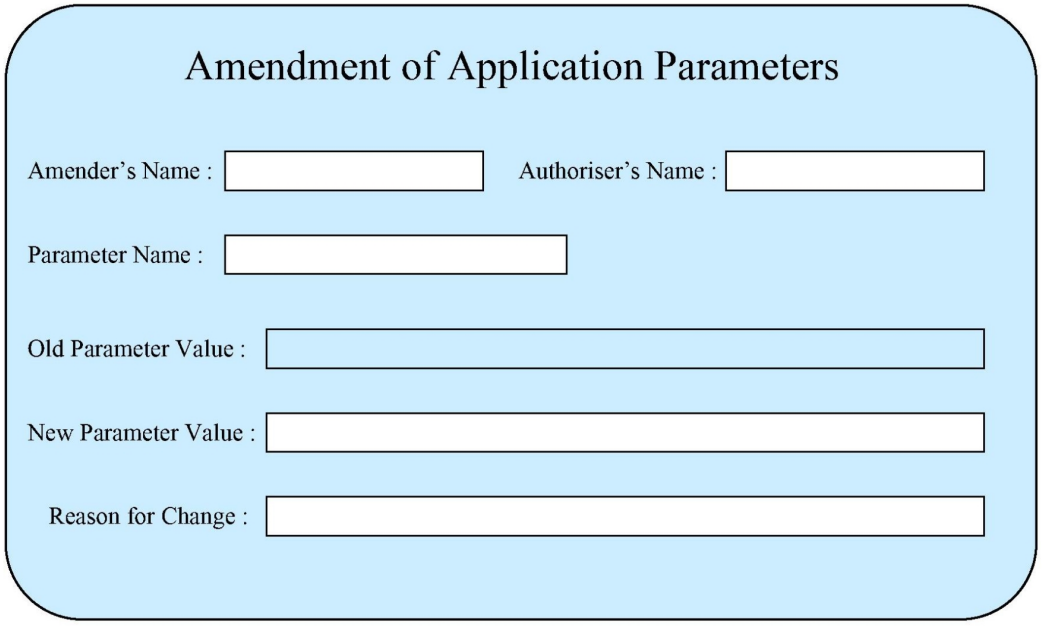

Figure 2 - Form for Amending Application Parameters

This form would not only update a record in the *Application_Parameters* table. It would also cause a record to be written to the *Action_Audit_Trails* table. This audit record would record the name of the person making the change, the name of the person authorising the change and a timestamp, as well as the contents of the record before and after the change.

FUJ00098223

Where feasible, the form should also attempt to validate the value of the new parameter submitted, though it is accepted that this would be difficult for a generic form such as that described above.

6.3.1.2 AUTHORISATION

Unless there are particular requirements for stricter security for an application, it is not expected that the authorisation process will involve any checks being made upon the value entered in the Authoriser's Name field. However, if the security requirements for a particular application are more stringent, the process could be enhanced to enforce two person working by requiring the database server to validate the authoriser's name against an application table held on the database. The form could also be extended to require passwords to be input by both the amender and the authoriser. These would also be matched against encrypted passwords held on the database.

6.3.2 UNPREDICTABLE AD HOC UPDATES

6.3.2.1 BATCH SCRIPTS

Although predefined forms should be provided for performing all predictable updating of data within an application, there are inevitably situations where data has to be amended unexpectedly and in an unpredictable manner, possibly as a result of database corruption. When this happens it is more important than ever for the changes that are made to be:

a) fully tested prior to application on the live database, and;

b) fully audited when applied to the live database.

The procedure for making this type of change is as follows:

1. Assess the changes that need to be made by querying the live database.
2. Write the script to make the changes required, ensuring that it inserts an audit record into the *Action_Audit_Trails* table for each record amended.
3. Test the script on a test database.
4. Run the script on the live database as a batch process.

6.3.2.2 EXAMPLE AD HOC UPDATE SCRIPT

Below is an example of a simple ad hoc script to delete a record from the *XXX_FILE_REGISTER* table.

```
Doc
   Name:          sc110998.sql
   Author:        John Doe
   Date written:  11/09/98
   PinICL No:     0099999
   Description:
      This script removes from the xxx_file_register table the record
      corresponding to the failed file described in PinICL 99999.
#

INSERT INTO action_audit_trails
   ( action_seq
    ,action_tsmp
```

FUJ00098223


```
     ,source_type
     ,source_id
     ,authoriser_id
     ,table_updated
     ,old_detail
     ,new_detail
     ,reason)
    SELECT
        act_seq.NEXT_VAL
       ,SYSDATE
       ,'B'
       ,'Joe Support-Bloke'
       ,'Jim Authoriser-Bloke'
       ,'XXX_FILE_REGISTER'
       ,x.file_type    || '|' ||
        x.file_seq     || '|' ||
        TO_CHAR(x.timestamp,'YYYYMMDD_HH24MISS')    || '|' ||
        x.file_status
       ,NULL
       ,'To fix failure as describe in PinICL 99999'
    FROM xxx_file_register x
    WHERE file_type  = 'C' AND
          file_seq   = 1234;

DELETE
FROM xxx_file_register
WHERE file_type  =  'C' AND
      file_seq   = 1234;

COMMIT;
```

FUJ00098223

7. AUDIT AND ARCHIVING

7.1 AN OVERVIEW OF AUDIT AND ARCHIVE

Audit and archive are words whose meanings have been blurred in recent years as different people have come to use them to mean different things. It is therefore worth starting this section with some clarification.

Audit.

The dictionary definition of the word is "the official scrutiny of accounts". In database terms, however, it has come to have a number of different meanings:

- As an abbreviation for an audit trail. An audit trail in this context is a track of user database activities and log violations. It provides evidence of unauthorised additions or deletions in a table, row or column and unsuccessful access attempts.
- To describe data which has exceeded its retention period on the database, and has been archived to off-line media for future scrutiny by auditors.
- As an abbreviation for an audit track. This is a term used to describe the traceability of items of data through the various components of an integrated computer environment.

Within this document, audit is taken to mean audit trail as described in the first description above.

Most RDBMSs provide the ability to collect this type of audit information, and to ensure that the logs so produced are secure. The problem with these default audit trails is that all too frequently they either provide too much information which wastes disc space and can impact on performance, or they provide insufficient information to satisfy the requirements of both audit and database support who would like to use the information to trace errors.

To fill the gap, it is therefore necessary for a generic process audit facility to be provided which satisfies the specific requirements of the applications which make use of the database.

Archive.

The dictionary definition of an archive is "a collection of documents or records". Again, in database terms, the term has been adopted to mean a number of entirely different things.

- An archive is most commonly used to mean a collection of historical data which has been offloaded from a database. The format of the archive files is usually such that the data within them can be restored into a database or schema other than the one from which it was offloaded. An archive is usually held on magnetic tape, though increasingly nowadays magneto optical CDs are used.
- In an Oracle environment, to describe the files produced by archiving the on-line redo log file. The redo log contains before and after images of database updates, inserts and deletes; the archive files are the off-line components of this redo log.

FUJ00098223

- As an alternative word for a backup. A backup is a copy of all or part of a database taken so that the database can be recovered without the loss of data should a hardware or software error occur. In general, backups are retained for short periods of time only and contain data in such a format that it can only be restored back into a database which is physically identical to that from which it was copied. Using the term archive to mean a backup is wrong.
- To describe audit data, as opposed to application data, which has been offloaded from the database to long term storage media.

Within this document, archive is taken to mean historical data which has been offloaded from the database, as in the first definition above. This description is also equivalent to the second definition of audit, though an archive is usually kept for more reasons than just to satisfy the requirements of auditors. It is also kept:

- as a source of data for the ad hoc analysis of business trends where an organisation decides not to keep such data in an expensive data warehouse;
- as a record of information for support purposes. An example of where this would be useful would be in an investigation to find the origin of a logical inconsistency which had appeared within the database. By trawling through the archive data, support staff could pinpoint exactly when, and possibly how, the corruption occurred.

7.2 AUDIT TRAILS

7.2.1 TYPES OF AUDIT DATA

The audit trails provided by the RDBMSs are primarily there as sources of information to aid investigations into suspicious activity, and to provide DBAs with statistical information on specific database activities. Examples of this would be information on how frequently specific tables are updated, or how many concurrent users connect at peak times.

What they are not so good at doing is providing non-generic information on the processing which occurs within specific applications. For instance an application might contain half a dozen batch modules which run daily, and which process different amounts of data each day depending on what on-line activity has been performed during the day.

What would be more interesting for the people who support this application would be to know how many records of a specific type were processed, how long each process took and whether any exceptions were encountered. These are obviously application specific requirements which can not be met by the RDBMS's own audit facility.

To provide this information, it is therefore necessary for an application to include within its schema tables that contain audit trails of:

- all processes that have been run;
- all files that have been loaded into or unloaded out of the application's schema, and;

FUJ00098223

- all actions that have been taken by people to change data within the application's schema.

Within Pathway, it is important that a common approach is taken to recording this information. To achieve this all applications must make use of audit tables which have a common structure. These table structures are described in the following sections.

7.2.2 PROCESS AUDIT TRAILS

The process audit trails table within an application is a historical record of what modules have been run and what were the results of their running. It is very important to discriminate between this type of table and a table which is used to control the actions of a module.

Control tables within an application are frequently used to hold checkpoint information to enable a module to pick up from where it left off after a system crash or other failure. The records within these tables are therefore frequently updated with status information. The records within a Process Audit Trails table on the other hand must never be updated as the table is there to provide a historical record of the business functions which have been executed. Updating the records within it would compromise the integrity of the audit trail itself.

The table to contain the audit trails of the processes within the Pathway domain is called *Process_Audit_Trails*. It is defined in the schema of the application in which the processes run and must be written to in the manner described below.

Records are written to the *Process_Audit_Trails* table by batch modules and daemon processes. Help Desk modules do not write to it: their activities are audited via call logging and event recording.

Each batch module or daemon must write and commit a record to the *Process_Audit_Trails* table as its first and as its last action. Where a module performs multiple steps, each of which is completing a distinct business task, such as "validation of payments against cardholders" the module should additionally write a record at the end of each of these tasks. For instance, if a validation module performed three distinct validation tasks, then a record should be written to the *Process_Audit_Trails* table after each of them.

Audit details are written in a consistent format to allow for their easy analysis by query tools. The information gathered can then be analysed to provide information on resource usage and to monitor database growth.

The format of the Process Audit Trails table should be as defined below.

Process_Audit_Trails			
Column name	*Null?*	*Datatype*	*Description*
Process_Seq	N	number(8)	A unique sequence number identifying the process. This sequence number is always incremented when a process is started, even if it is a restart following a failure. *Primary Key Component 1*

FUJ00098223

Process_Audit_Trails			
Column name	*Null?*	*Datatype*	*Description*
Process_Entry_Seq	N	Number(4)	A sequence number, starting from 1 for each Process_Seq, which uniquely identifies the audit entry within the process. *Primary Key Component 2*
Module_Id	N	Varchar2(7)	Module running the process
Version_No	N	Number(3)	Version number of the module specified in Module_Id
Process_Tsmp	N	Date	Date and time details written to table
PID	Y	Number(10)	The Operating System Process Id
Element_Type	Y	Varchar2(2)	This defines the type of element being processed, the count of which is in the Elements_Count attribute. These can be application specific, but must be unique and centrally registered in Pathway. For example, 'RC' Records in a file read or written. 'CT' Count of rows from a table which is scanned.
Element_Count	Y	Number(12)	A count of the elements, the type of which is defined in the Element_Type attribute.
Daemon_Alert_Count	Y	Number(8)	If the module is a daemon that reacts to database alerts, this is a count of the number of times the daemon was alerted.
Daemon_Idle_Count	Y	Number(8)	If the module is a daemon that reacts to database alerts, this is a count of the number of times the daemon was alerted, but found no work to do.
Daemon_Poll_Time	Y	Number(4)	If the module is a daemon that reacts to database alerts, the frequency at which it polls the target table as a precaution against missing any alerts.
Daemon_Delay_Time	Y	Number(4)	If the module is a daemon that reacts to database alerts, the time the daemon waits in between checking for alerts. The daemon_delay_time should divide exactly into daemon_poll_time (typically 3:1).
Further_Info_Type	Y	Varchar2(2)	This defines the type of further information provided in the Further_Info attribute. For example: 'IF' Full UNIX file name of incoming file. 'OF' Full UNIX file name of outgoing file.

FUJ00098223

Process_Audit_Trails			
Column name	*Null?*	*Datatype*	*Description*
Further_Info	Y	varchar2(255)	The further information as described in the Further_Info_Type attribute.

The definition of the *Process_Audit_Trails* table may also be extended to include statistical information specific to the RDBMS used by the application.

For Oracle applications these additional fields are defined below.

Column name	*Null?*	*Datatype*	*Description*
Oracle Sid	Y	Number	The database instance ID for the session, part one.
Oracle Serial	Y	Number	The database instance ID for the session, part two.
Physical Reads	Y	Number	The number of physical read IO's requested to fetch data from disk.
Block Gets	Y	Number	The number of database blocks read by the process since the session started.
Consistent Gets	Y	Number	
Block Changes	Y	Number	The number of database blocks written.
Consistent Changes	Y	Number	

7.2.3 FILE AUDIT TRAILS

All the actions involved in loading data into a database from non-database files, and unloading data from the database to non-database files, are registered in a table within the database. This is necessary for three reasons:

- to provide information to support staff about the loading and unloading of the data;
- to satisfy audit requirements, and;
- to support any subsequent archiving of the files.

Non-database files which are used for the loading and unloading of data are all registered within a table called *File_Audit_Trails*. A single table is used for all the types of all the files which are transferred into and out of the database.

File_Audit_Trails			
Column name	*Null?*	*Datatype*	*Description*
File_Event_Seq	N	number(8)	A unique sequence number identifying the file event. *Primary Key*
Input_Output	N	varchar2(1)	I The file is input into the application O The file is output from the application
Remote_Application	N	varchar2(8)	The identifier of the remote application to which the file is sent or from which the file is received.

FUJ00098223

File_Audit_Trails			
Column name	*Null?*	*Datatype*	*Description*
File_Type	N	varchar2(3)	A code identifying the type of file loaded or unloaded
Filename	N	varchar2(30)	The terminal name of the file as it is created in the specified File_Directory
File_Status	N	varchar2(1)	A File allocated to load process C File processing complete D Delivery of outgoing file complete E File has been rejected H File is being post-processed N Arrival of new file for loading detected, or new file for unloading created P File is being processed R File ready to transfer V File is being pre-processed X File has been archived
Status_Change_Tsmp	N	Date	Time of status change as specified in File_Status attribute
File_Directory	Y	Varchar2(80)	The name of the directory which contains the file.
Record_Count	Y	Number(10)	A count of the number of records in the file
Average_Record_Size	Y	Number(6)	The average size of the data records in the file

Although not declared as such, it is worth noting that the logical primary key of the table is Remote_Application / File_Type / Filename / File_Status.

7.2.4 AUDITING OF INTER-DATABASE TRANSFERS OF DATA

If the interface between applications or databases is implemented by means of database links as described in section 9, rather than by the exchange of files, the auditing requirements of the interface are satisfied by the information which is written to the *Application_Accesses* table in the interface user as specified in section 9.5.5.

7.2.5 ACTION AUDIT TRAILS

It is frequently necessary for Systems Administrators to change the way the database or an application operates. This is usually done by amending the operational parameters of the system or application as described in section 5.1. For security, auditing and support reasons, it is important that any changes of this type are adequately logged so that anyone coming in afterwards can see what changes have been made, why they were made and when they were made.

Any changes to the operational parameters, whether they are made to the systems or applications parameters tables, or to other non-generic parameters tables, must be written to the *Action_Audit_Trails* table. One record is written for each record changed.

The format of this table is defined below.

FUJ00098223

Action_Audit_Trails			
Column name	*Null?*	*Datatype*	*Description*
Action_Seq	N	Number(8)	A unique sequence number identifying the action. *Primary Key*
Action_Tsmp	N	Date	Date and time of the action
Source_Type	N	Varchar2(1)	The type of source making the change. M Application Module S Support staff interactively B Batch script
Source_Id	N	Varchar2(30)	The name of the source making the change. If source_type is 'M', this is the name of the module; if 'S' or 'B' it is the name or identifier of the person making the change.
Authoriser_Id	Y	Varchar2(30)	If source_type is 'S' or 'B' this is the name or identity of the person who authorised the change.
Table_Updated	N	Varchar2(30)	The full name of the table updated (e.g. Application_Parameters)
Old_Detail	Y	varchar2(1500)	Contains the complete record as it was before the change is made in the "trimmed" format as specified in section 7.4.4.4. Default values are assumed for all other format directives specified in section 7.4.4.4. = NULL if a record is inserted.
New_Detail	Y	varchar2(1500)	Contains the complete record as it is after the change has been made in the "trimmed" format as specified in section 7.4.4.4. Default values are assumed for all other format directives specified in section 7.4.4.4. = NULL if a record is deleted
Reason	Y	varchar2(800)	The reason for making the change as described by the amender.

7.3 ARCHIVE DATA

Archive data is data that has been offloaded from a production database into an off-line media store because it is no longer of current interest to the application which uses that database. Although often confused with backup data, archive data has a number of features which set it apart from backup data and which require it to be treated in an entirely different manner. The table below summarises these differences.

Feature	Archive Data	Backup Data
Data Retention	Usually measured in years	Days, or at most weeks.
Retrieval requirements	Data must be capable of being restored into any type of database	Data only required to be restored into the database from which it

FUJ00098223

	on any type of machine.	was archived, or one identical to it.
End User Data Access Requirements	Usually not known	Only via standard RDBMS or operating system modules.
Volume	Many gigabytes to many terabytes	Many megabytes to many gigabytes
Media	Usually Tape or CD	Either disc or fast tape

As a consequence of these distinctive features, archive data must be written in such a way that it contains within it the data dictionary definitions of the data itself. The data archived from the PAS/CMS database is currently archived in such a format, and therefore, for consistency, this is the basis of the mechanism which will be used for all other archive data produced by the host system applications.

7.4 THE MECHANISM FOR ARCHIVING DATA FROM A DATABASE

7.4.1 ARCHIVE CONTROL

The archiving of data from the database is a two-phase process. The first phase involves copying the archive data from the database to files within a UNIX filesystem directory. The second phase involves copying these filesystem files to a tape library using the ESBM backup manager product.

Archive control tables within the database from which the data is archived control the first phase of the process. There are two of these tables; one of which holds the details of the tables to be archived; the other, the details of each archive event. These tables are defined as follows:

Archived_Tables			
Column name	*Null?*	*Datatype*	*Description*
Application_Alias	N	varchar2(3)	The alias for the application as described in 3.2. *Primary Key Component 1*
Table_Alias	N	varchar2(3)	The alias for the table as described in 3.4.3. *Primary Key Component 2*

FUJ00098223

Archived_Tables			
Column name	*Null?*	*Datatype*	*Description*
Archive_Type	N	Varchar2(2)	FK Foreign Key archive. All records having logical foreign keys referencing primary keys of records which have been deleted are archived. FT Full table archive. All records in the table are archived. RP All records which are older than the period specified in the Retention_Period attribute are archived. AI All data with the attribute Actioned_Ind set to Y are archived. An actioned_ind column must exist on the archived table against which the comparison can be made.
Purge_After_Archive	N	Varchar2(1)	Y The archived records are removed from the database table after archiving. N The archived records are not removed from the database table after archiving.
Directory	N	Varchar2(80)	The full name of the UNIX directory to which the archive file is to be written.
Retention_Period	Y	Number(5)	Only used if Archive_Type = RP. The number of days after which records within the table become candidates for archiving. A timestamp column must exist on the table to be archived against which this attribute can be compared.
Archive_Threshold	Y	Number(2)	This attribute is used by the archive process to determine whether it is worth performing the archive. After a pre-scan of the table, if it is found that the percentage of records that would be archived from the table is below this threshold, then the archive is not performed.

Archive_Events			
Column name	*Null?*	*Datatype*	*Description*
Application_Alias	N	varchar2(3)	The alias for the application as described in 3.2. *Primary Key Component 1*
Table_Alias	N	varchar2(3)	The alias for the table as described in 3.4.3. *Primary Key Component 2*

FUJ00098223

Archive_Events			
Column name	*Null?*	*Datatype*	*Description*
Archive_Seq	N	number(8)	A sequence number of the archive event for the table specified in Table_Alias *Primary Key Component 3*
Archive_Status	N	varchar2(1)	P Archive in progress S Archive finished successfully F Archive failed
Creation_Tsmp	N	Date	Time of the archive file's creation
Status_Change_Tsmp	N	Date	Time of status change as specified in Archive_Status attribute
Archive_File_Name	N	Varchar2(110)	Full name of the file produced by the archive event
Record_Count	Y	Number(10)	A count of the number of records archived
Average_Record_Size	Y	Number(6)	The average size of the data records archived

7.4.2 NAMING OF THE ARCHIVE FILE

As with all file names, it is important that the names generated for the archive files by the first stage of the archiving process are meaningful and easy to understand. A standard naming convention is therefore used.

This convention specifies that the format of the filename within the archive directory must be as follows:

AAAXXXYYYYMMDDNNNN.arc

AAA	The application identifier as specified in section 3.2;
XXX	The alias of the table archived as specified in section 3.4.3;
YYYYMMDD	The date of the archive;
NNNN	The sequence number of the archive within the archive date. If only one file is ever created on one day, this will always be 0001;
.arc	Standard suffix for all archive files once the archive process is complete. Whilst the archive is in process, the suffix used is '.tmp'.

7.4.3 THE ARCHIVE PROCESS

Before the archiving process is started it is strongly recommended that the table to be archived and any dependent tables are backed up.

The archive process actively uses the archive control tables. The first action performed by the archive process is to examine the relevant record in the *Archived_Tables* control table to see what sort of archive is required. The actions performed during the archive are then dependent on the Archive Type.

- If the archive_type is FT, the archive process immediately goes ahead and archives the whole table. If then the Purge_After_Delete parameter is specified, the table from which the data was archived has all records within it removed. The quickest means of achieving this is by using the TRUNCATE command.

FUJ00098223

- If the archive_type is RP or AI and the Archive_Threshold attribute is not null, a pre-scan of the whole table is first performed to see what percentage of the table would be deleted by the archive process. If this value turns out to be less than the value specified in the Archive_Threshold attribute, then the archive is aborted as it wouldn't be worth doing.
  The archive is then performed according to the criteria specified in attributes of the *Archive_Tables* record. On completion, if the Purge_After_Delete parameter is specified, the archived records are removed from the table using either the delete command, or if the table is very large and there are many records to remove, using the parallel Create Table As Select construct.

When designing an archive process that includes a purge phase, it is important to consider whether the purging of records in the archived table requires any further purging of records in other tables. This may be required if there are logical master-detail links in existence which are not implemented physically by foreign key constraints.

7.4.4 ARCHIVE FILE FORMAT

7.4.4.1 ARCHIVE FILE STRUCTURE

The files produced by the archiving processes must all contain within them full definitions of the tables from which they are archived, as held within the database's data dictionary. This is necessary so that the file can in the future be read back into a database which has no knowledge of the table from which the data was archived, or, indeed, which may be of a completely different type.

The archive file is therefore in two parts. The first part contains the data mapping directives as extracted from the Oracle data dictionary view: *user_tab_columns*. The second part contains the data itself.

Each record in the file is terminated by a "carriage return" character. At the end of the file there is optionally also a trailer directive.

7.4.4.2 ARCHIVE FILE DIRECTIVES

The data mapping directives take the form of a number of records in the file, each of which has the format: <*directive*> followed by the directive itself. Each of these directives must start on a new line.

The directives are defined in the table below.

Directive	*Meaning*	*Notes*
<*col*>	Column details	At least one <col> directive must be included
<*date*>	Date and timestamp when file produced. Format is 'ddmmyyyy_hhmmss'	Directive must be included. The 24hour clock must be used
<*end*>	End of directives	Must be the last directive before the start of the data
<*map*>	Start of record definition	Must be first directive

FUJ00098223

Directive	*Meaning*	*Notes*
<notrim>	Output data is not trimmed. This means: · text has trailing spaces padded to full column width as specified in column directive. · numbers have leading zeroes padded to full column width as specified in column directive	Must not be specified if *<trim>* is declared.
<null>	NULL value identification for NULL values	Optional; default is '^'. Only used for untrimmed data
<separator>	Separation character(s) used if data trimmed	Optional; default is '|'
<sign>	If output data is not trimmed, and this directive is declared, then all numeric fields include a leading character position to accommodate the + or - sign	Optional; default is not to include a leading character position; all numbers are then treated as positive
<table>	Name of the table name from which the data was archived	Directive must be included. This name should be in the format schema.table_name. (For PAS/CMS, the "schema." part of the name is omitted.)
<trailer>	After all the data records have been written, this directive can optionally be written to designate the end of the archive file	If written, this must be the last record in the file
<trim>	Output data is trimmed. This means: · text has trailing spaces removed · numbers have leading zeros removed	Must not be specified if *<notrim>* is declared. Default value if neither *<trim>* nor *<notrim>* is declared.

7.4.4.3 COLUMN MAPPING

Within the archive file directives there is a separate directive for each column of the table to be archived.

The column directive has the structure: *<col>column_name|N|datatype* where:

Column_name is the name of the column as held in the database's data dictionary;

N defines whether a NULL value is allowed for that column on the database. N means the column can not be null; the absence of any value signifies that it can;

Datatype is the datatype defined for that column on the database. This may be abbreviated to the datatype's first character.

The separator in the column directive is always '|' regardless of the value given to the *<separator>* directive.

7.4.4.4 TRIMMED AND UNTRIMMED DATA

The <notrim> and <trim> directives define whether the data records are written as fixed length records, with each field in a set position, or as variable length records with each field being separated by the character defined in the <separator> directive.

FUJ00098223

Archive files are normally stored in the variable length, trimmed format as this ordinarily saves on disc space. The untrimmed format can, however, save disc space over the trimmed format if the data archived is all packed out to their maximum field sizes. This is because the field separator characters are not included in the untrimmed data.

The rules for storing untrimmed and trimmed data are given below.

Untrimmed data output rules

- If the column datatype is character then the column value is output with trailing spaces padded out to the full column width. If the column value is NULL then the <null> character string is output and again trailing spaces added.
- If the column datatype is numeric then the column value is output with leading zeros padded out to the full column width. Any decimal part is always output to the maximum number of digits specified (including, of course, the decimal point). If the column value is NULL then the <null> character string is output and again trailing spaces added.
- If the column datatype is date then the column value is as 'ddmmyyyy_hhmmss'. The time component is always output as it cannot be recognised whether or not the column is a date or a timestamp datatype. If the column value is NULL then the <null> character string is output and again trailing spaces added

Trimmed data output rules

- If the column datatype is character and not allowed to be null then, if the field contains all spaces, one space is output, otherwise the character string is output as it is held on the database, with no trailing spaces.
- If the column datatype is numeric and not allowed to be null then, if the value of the field is zero, one zero is output, otherwise the number is output as it is held on the database, with no leading zeros.
- If the column datatype is date and not allowed to be null then the field is output in the same format as it would were it to be untrimmed.
- If the column can be NULL, and on the database the field contains NULL, then no characters are output for that field.

Note that the trimmed format is also used for records written to the detail columns within the *Action_Audit_Trails* table and the *ModuleX_Excptns* tables.

FUJ00098223

7.4.4.5 ARCHIVE FILE EXAMPLE

Below is an example of the archive directives that might be used for the *Card_Events* table on the PAS/CMS database.

```
<map>
<table>CARD_EVENTS
<date>21021998_101227
<trim>
<separator>|
<col>EVENT_SEQ|N|NUMBER(16)
<col>EVENT_TSMP||DATE
<col>BATCH_ID|N|NUMBER(10)
<col>SOURCE_ID1|N|VARCHAR2(30)
<col>SOURCE_ID2|N|VARCHAR2(30)
<col>STATUS_CODE|N|VARCHAR2(3)
<col>CALL_ID|N|NUMBER(16)
<col>EVENT_TYPE||NUMBER(2)
<col>EVENT_SOURCE||VARCHAR2(1)
<col>SOURCE_DESC||NUMBER(2)
<col>PRIMARY_AC_NO||NUMBER(16)
<col>CARD_ISSUE_NO||NUMBER(3)
<end>
27|01021998_085721|50|Src1|Src2|STP|113|AB|H||1234567890123456|1
28|01021998_085831|51|Src3|Src2|ORD|115|CD|H|XX|9876543210654321|10
<trailer>
```

7.4.5 FUTURE ACCESS OF ARCHIVED DATA

A successful archiving strategy must permit archived data to be accessed many months or even years after it has been written. It is not practical therefore at the time when the archiving strategy is being developed to be definitive about how that data is to be accessed in the future when technology will surely have moved on and when almost certainly better methods of accessing the data will be available.

7.5 ARCHIVING OF FILES USED FOR LOADING AND UNLOADING

7.5.1 CONTROL OF THE ARCHIVING OF NON-DATABASE FILES

There are frequently requirements to archive the non-database files which are used for the loading or unloading of data to and from the database. As with the archiving of the database tables, it is necessary here also to have control over what files have been archived, when they were archived, and to where they have been archived. This control information is retained in the database into which or from which the data is loaded or unloaded. This information is all stored in the file register tables as described in section 0.

FUJ00098223

8. EXCEPTION HANDLING

8.1 EXCEPTION HANDLING PRINCIPLES

One of the ways of ensuring that an application functions successfully without the deployment of a huge support effort is to design and build it using defensive coding techniques. This means that the application is designed such that it can handle in a controlled manner any sort of unexpected exception condition that could feasibly occur.

Before going into detail about the handling of unexpected exception conditions, it is first worth defining exactly what is meant by such an exception condition. This is best done by example. The following are some examples of unexpected exception conditions:

- Invalid data which has been erroneously input into the database. This might, for example, be due to incorrect validation criteria being defined on a client form, or to a file of records being directly loaded into a database on which pre-validation had not been completely performed.
- Logical inconsistencies between database objects, such as may be encountered if a foreign key dependency had not been implemented using a constraint for performance reasons.
- The receipt of an error response from the RDBMS to a valid DML request.

What an exception condition is not is an expected business event such as the receipt of an order for a customer who, unexpectedly, does not exist on the database. For these business exception conditions, which are quite able to be predicted, the application itself must be designed to cater for and, if necessary report on.

8.2 HANDLING UNEXPECTED EXCEPTION CONDITIONS

To simplify support of an application, it is important that all unexpected exception conditions encountered in the Pathway environment, are handled in the same manner. The following are a series of design principles which can be employed to achieve this.

- All SQL statements must be coded to take into account the possibility that the statement may fail. This means that, not only should it do obvious things such as checking the SQL code returned by the RDBMS, it should also actively check the logical format of the data returned where this hasn't been guaranteed by the existence of check constraints on the records' columns.
- If a SQL statement fails or produces unexpected results, a minimum of one record must be written to an exceptions table as described below. If the SQL statement encounters more than one row which is invalid, it should write a record to the exceptions code for each invalid row.

FUJ00098223

- Whenever a record is written to an exceptions table an alert should automatically be raised to enable application support to react to the fault immediately. Within Pathway, the alert is signalled using the BMC Patrol product.
- If the exception condition encountered is deemed by the application designer to be serious enough for the module not to be allowed to continue, then the module should roll back all uncommitted updates and abend gracefully to Maestro.

8.3 EXCEPTIONS TABLES

For each process for which exceptions could conceivably be encountered, an exceptions table should be defined within the application's schema. It is usually impractical for there to be a single exceptions table for each and every module which can access the database; however, it is similarly impractical for there to be just one which is used by all modules within an application. In practice, a single exceptions table would normally be used by a number of different modules, so long as they all do broadly similar processing on broadly similar tables.

Each exceptions table must have a Patrol Knowledge Module associated with it that signals an alert to a central console whenever one or more records get written to the table.

The format of the table should be as defined below.

ModuleX_Excptns (where ModuleX identifies the module(s))			
Column name	*Null?*	*Datatype*	*Description*
Process_Seq	N	number(8)	The unique sequence number identifying the process which produced this exception report. As on *Process_Audit_Trails* (see section 7.2.2). *Primary Key Component 1*
Excptn_Seq	N	number(8)	A sequence number identifying the exception within Process_Seq. *Primary Key Component 2*
Module_Id	N	varchar2(7)	Module encountering the failure
Excptn_Tsmp	N	date	System timestamp of when the exception was encountered.
Appl_Excptn_Code	N	varchar2(7)	Application Exception Code (see below)
PID	Y	number(10)	The Operating System Process Id
Dbms_Excptn_Code	Y	number(6)	The RDBMS error code, if the exception is caused by the RDBMS.
Excptn_Object	Y	varchar2(30)	If this is a data error, the name of the table or view in which the invalid record has been encountered.
Excptn_Detail	Y	varchar2(800)	The full invalid row in the trimmed format specified in section 7.4.4.4.

FUJ00098223

In nearly all cases in which a record is written to an exceptions table, some action needs to be taken, either by business support if it is an application fault, or by database support if the problem is with the RDBMS, or maybe by both if the data within the database turns out to be corrupt.

For all the applications in the Pathway environment, it is probably essential that this action is taken as soon after an exception has been reported as possible. In many cases, of course, the exception condition itself will have halted the module. However, if corrupt data has been encountered by a module, this event in itself may not be deemed sufficiently serious by the application designer to cause the module to fail. For instance, if a million rows are to be processed, and a logical inconsistency is detected in one of them, this is not necessarily serious enough to hold up the processing of the rest of the rows. It is however a condition which needs to be investigated as a matter of some urgency as it may be symptomatic of more widespread corruption.

8.4 EXCEPTION CODES

Each application has its own unique range of exception codes that it uses for flagging up unexpected exception conditions. These exception codes are all seven characters in length and have a format of AAA9999, where AAA is the 3 character application alias (see section 3.2), and 9999 uniquely identifies the exception within the application.

All the exception codes that a particular application can use are defined within that application's own schema in a table called *Exception_Codes*.

The format of the table is defined below.

Exception_Codes			
Column name	*Null?*	*Datatype*	*Description*
Excptn_Code	N	varchar2(7)	Application Exception Code in the format AAA9999 *Primary Key*
Short_Excptn_Msg	N	varchar2(40)	A short exception message suitable for displaying on forms etc.
Full_Description	Y	varchar2(240)	Full description of the exception
Action	Y	varchar2(240)	Describes what user action is necessary to resolve the exception

FUJ00098223

9. COMMUNICATING BETWEEN APPLICATIONS

9.1 BACKGROUND

Most of the applications being developed as part of the Pathway programme are, at the host layer, logically separate from one another. There are, however, numerous requirements for data to be passed from one application to another.

The diagram in Figure 3 is a top-level DFD which illustrates the flow of data between the host system applications at Release 2. This diagram will obviously become more complex as more applications are introduced. It is therefore very important that this increased complexity does not make the overall Pathway solution correspondingly complex and unwieldy. Specifically, it is essential that when new applications are added or changed, there is no requirement to re-test all the other components of the system with which they interact.

To achieve this, it is necessary for applications to be designed using the principles taken from the Object Oriented approach to software development. In particular, it is essential that applications embrace encapsulation. In the domain of database building, this means that the data and process implementation of one application must be hidden from all other applications with which that application interacts. Applications must only be allowed access to one another by means of well defined interfaces, each of which must be documented in an Application Interface Specification.

The logical consequence of this is that a generic mechanism must be adopted within Pathway which allows applications both to communicate with one another in an efficient and straightforward manner and also to support fully the requirements of encapsulation.

FUJ00098223

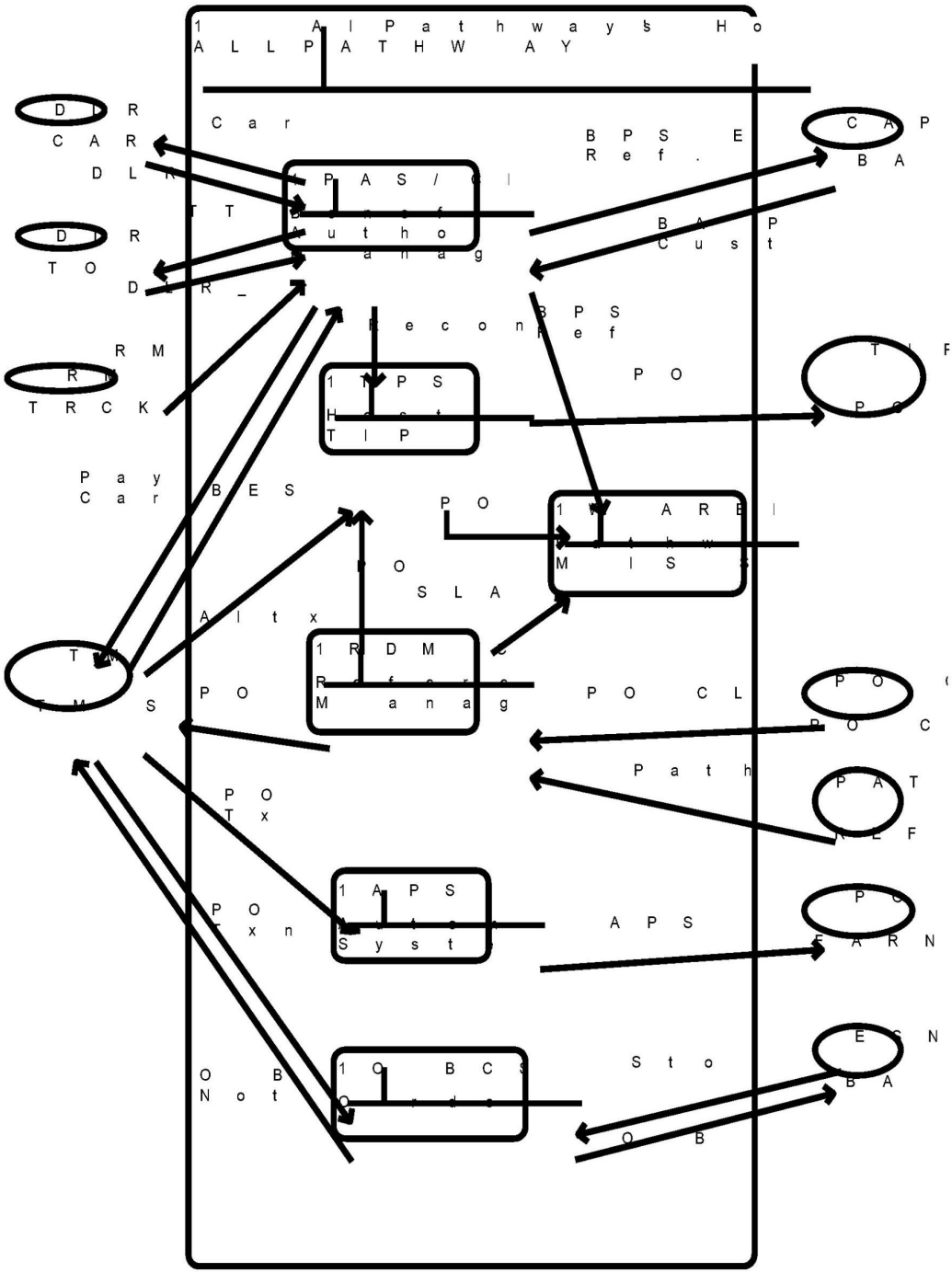

Figure 3 - DFD of Pathway's Host Applications

FUJ00098223

There are a number of ways that this intra-application communication can be implemented, all of which have both advantages and disadvantages. Up until Release 2, only two techniques, as described in the table below, had been used .

Technique	Description	Advantages	Disadvantages
Unloading / Loading of flat files.	Data is written from Application A on Database A to UNIX flat files. Application B then reads these files into Database B	• Clean interface. No dependency at all between applications. • No restrictions imposed on how the data is loaded into the destination database	• Poor Performance • Complex to develop • Files have to be managed and housekept • Recovery and Resilience processes have to be written
Intra-Database Direct Access using SQL (i.e. one application directly accessing another's objects using synonyms)	Application A and Application B physically exist in the same database and communicate directly with one another via SQL.	• Excellent Performance • Simple to develop • Recovery and Resilience managed by the database	• Applications must exist in the same database and can not be moved to different platforms • Applications are intrinsically dependent on one another. Changes to one require re-testing of the other

As can be seen from the table, the unloading/loading flat files is clean in OO terms, but has performance, developmental and system management disadvantages; whilst intra-database direct access, which is currently used for the interface between TPS and RDMC, has none of the disadvantages of the first technique, but does not satisfy in any way the requirements of encapsulation which are essential for Pathway.

If Pathway is to support the expansion in the number of host systems which is currently envisaged, it is necessary for a third technique to be employed that has the advantages of both the techniques described above, but none of the disadvantages.

9.2 ALTERNATIVE OPTIONS

There are two additional techniques over and above those defined in the previous section which can be used to enable applications in different database to communicate with one another. One is data replication, the other is data distribution.

9.2.1 REPLICATED DATA

With data replication, all updates to tables which are defined within a replication schema on the remote database are automatically replicated to copies of those tables on the local database by means of "snapshots" and by updates to those snapshots in "snapshot logs" that are transferred from the remote database at predetermined intervals. Replication can be a one-way process in which case it is called asynchronous replication, or it can be allowed in both directions when it is called symmetric replication.

The main advantages of replication are that:

- it uses the standard features of the RDBMS and therefore requires no special code to be written. It is therefore relatively easy and inexpensive to implement.

FUJ00098223

- One application is not dependent upon the availability of another.

The main disadvantages are that:

- Both databases have to use the same RDBMS.
- Performance of the data transfer is poor.
- Disc space for the replicated data has to be available at both local and remote sites.
- It is difficult for the remote site to know whether all local sites have received the replicated data.

9.2.2 DISTRIBUTED DATA

In a distributed system, the data exists on only one database. If an application on another database wants access to that data, it accesses it directly and transparently using standard SQL. This transparency is achieved by the use of database links and synonyms.

The advantages of data distribution are that:

- It can be configured to support encapsulation fully.
- By manually paralleling the access of data across the network, performance can be optimised.
- As the data can be accessed directly, it is not necessary to hold a copy of the data on the local database as well as on the remote one.

The main disadvantage is that:

- it requires the implementation of some additional schema objects and code in order to control the environment.
- Both databases have to be running and available.

Although under exceptional conditions asynchronous replication could be used for specific Pathway applications, it is expected that only database links and synonyms used in the controlled manner described below will be used as a matter of course.

9.3 DATABASE LINKS AND SYNONYMS

Within an Oracle environment, it is relatively simple to configure applications so that they transparently communicate with one another using standard SQL, even if the remote application exists on a different database at a different location. To do this, one first creates a database link, or "path", within the local application's schema which points to an object in the remote application's schema. One then creates a synonym for the remote object in the local application's schema, which can then be referenced as if it were the name of a local table. The local application can then use the synonym to access that information without concern for where the data is actually stored.

Before Release 2, this technique had not been used within Pathway because it creates inter-database and inter-application dependencies which are hard to manage. Specifically these dependencies are:

1. Scheduling.
   The remote database must be up and running, and not actively updating the data required, when the local application requires to access the remote database.

FUJ00098223

2. Data Retention.
   Data has to be retained on the remote database until that data has successfully been accessed by all distributed applications which have an interest in it. This is particularly pertinent to the Pathway environment where most of the data is transactional, and thus transient, in nature. If any of the applications were down for an extended period of time, data may have to be kept on the remote database for longer than its designed retention period, possibly resulting in tablespaces filling up.
3. Testing
   When testing applications, test versions of the distributed databases need to be available to provide the remote data, thus increasing the complexity of the test environment.
4. Transparency
   There is a perception that the use of database links and synonyms means that both the local and remote databases have to use the same RDBMS, which breaks the rules of encapsulation whereby applications must not be required to know how applications with which they communicate are implemented.

The first three of these dependencies are powerful arguments against the use of database links, and they must all be countered if this technique is to be used in the Pathway solution. This can be done, but only by the strict definition and enforcement of application interface standards. Most of the remainder of this section is therefore devoted to the provision of a generic standard which can be used in any situation.

The last of the dependencies numbered above, that the data can not be accessed transparently, can be countered by the use, where necessary, of transparent gateways.

9.4 TRANSPARENT GATEWAYS

For Oracle databases, Transparent Gateways are software products which fit between the local application's Oracle database and the remote non-Oracle database. (Of course, where the remote database is also Oracle then no gateway software would be necessary.) These products are available from Oracle Corporation itself as well as from many other suppliers.

An example of a gateway product is the Oracle Transparent Gateway to Microsoft SQL Server. This product allows Oracle client applications to access SQL Server data transparently. The gateway, in conjunction with the Oracle7 server, creates the appearance that all data resides on the local Oracle7 server, even though data might be widely distributed.

If in the future it were decided to change the implementation of the remote database, from SQL Server to, say, Oracle7 (or, for that matter, to any other RDBMS assuming the appropriate gateway product was available), no changes in client application code would be necessary because the gateway handles all differences in datatypes or SQL functions between the application and the database. The diagram below illustrates the architecture.

FUJ00098223

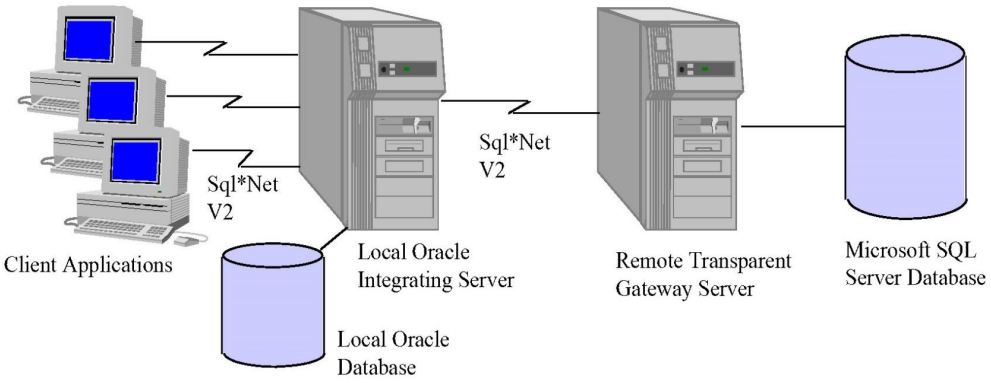

Figure 4 - Transparent Gateway Architecture and Components

The Oracle7 server connects directly to the gateway and thus facilitates heterogeneous queries against Oracle7 and SQL Server data. It also post-processes Oracle7 SQL functions which are not supported by SQL Server. The gateway runs as an NT service at the remote location.

The main drawback in using a transparent gateway is that the performance of access through such a link is never going to be as good as a link between databases of the same type. Careful consideration should therefore be given as to the wisdom of using heterogeneous databases where very large volumes of data have to be shipped quickly between them.

9.5 APPLICATION INTERFACE STANDARDS

9.5.1 NOMENCLATURE

In the following sections, the local application is defined as being the one which is accessing the data, the remote application as the one which owns the data to be accessed. Although a remote application is usually on a different database, there is no reason why it should not be within the same physical database.

9.5.2 DOCUMENTING THE INTERFACE

For every interface which involves the passing of data between applications, whether internally or externally, there must exist an Application Interface Specification, which defines all aspects, both physical and logical, the interface. This document must be written in the format required by Pathway's On-Line Standards.

9.5.3 DEFINING THE INTERFACE DATA

The data to be passed across the interface is defined within the remote application's database. This data must be presented to the local application as a set of objects existing within an interface user on the remote database. For each interface with different access requirements there must be a separate user. The schema for the interface user must contain only those objects which are relevant to that interface and which are defined in the associated Application Interface Specification.

FUJ00098223

A basic premise of using an interface user is that the remote application itself must be unaware of the existence of the interface user, even though it exists within the same database. If there are changes to the local application which require different data to be passed from the remote application, then this must only affect the schema of the interface user, it must not require changes to be made to the remote application itself. Likewise, if the schema of the remote application changes, this must not be allowed to affect the format of the data which the local application extracts from the interface user.

The diagram below illustrates the environment for a remote application, D, which provides data to distributed applications A and B, as well as to a third application, C, which happens to exist within the same database.

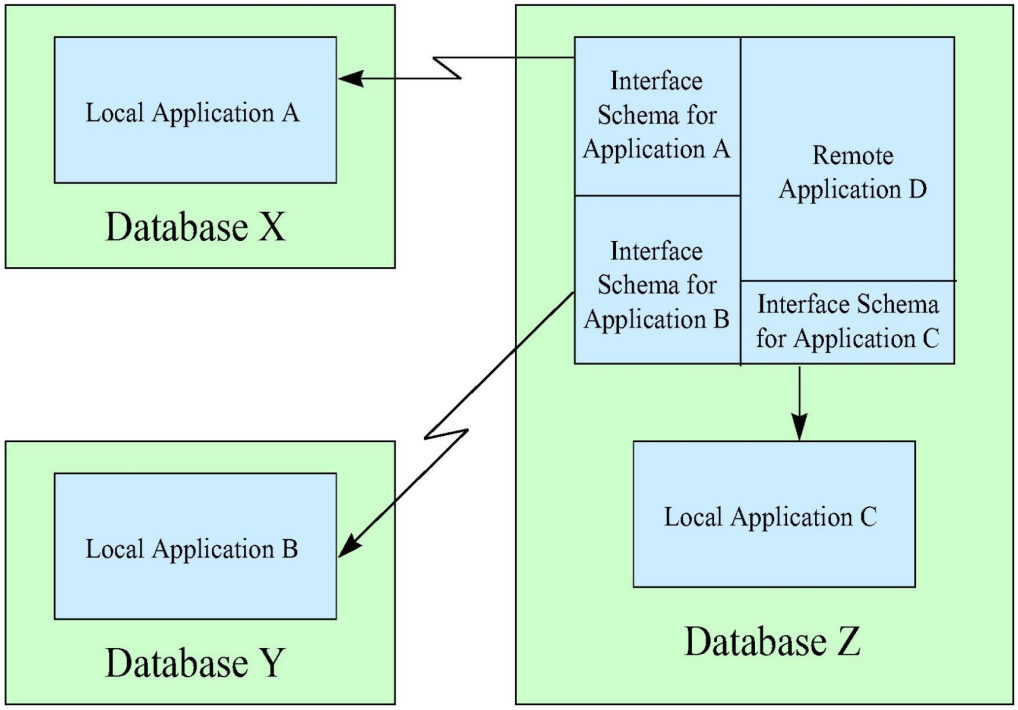

Figure 5 - Inter-Application Communication using Interface Users

The objects which exist in the interface user's schema can be:

- views of data which exist in base tables within the remote application's schema. Unless there is a time-dependency on the data which is to be passed across the interface, this is the mechanism which should be used;
- tables which are used only for the passing of data across the interface. Typically these tables would be created daily by a process which extracts a snapshot of data from other tables within the remote application. Transient data must be presented to local applications within this type of interface table;
- views of interface tables which exist in the schemas of other applications' interface users, within the remote application.

FUJ00098223

9.5.4 NAMING OBJECTS AND SYNONYMS IN THE INTERFACE USER

Within an interface user the only objects which should exist are the control tables, which are defined below, and the transient tables. All other objects that require to be accessed only exist as synonyms within the interface user.

If there are no transient tables to be transferred across the interface, no tables at all need to be defined within the interface user.

The naming standards for the transient tables can be found in section 3.4.2. The synonyms should just have, as for normal functional tables, short, meaningful, plural names. No further qualification is required as the objects and synonyms within the interface user on the remote machine must be accessed on the local machine either using the database link qualifier, or via a locally defined synonym.

The synonyms defined on the local machine should be named according to the standards in section 3.5.2.

9.5.5 INTERFACE CONTROL IN THE REMOTE APPLICATION

If transient tables are to be used for the transferral of data across an interface, then the schema for that interface user must contain a set of control tables to hold information about:

- any transient tables which are to be created in the interface user, and;
- all the distributed local applications which can access those transient tables.

The primary purpose of these tables is to ensure that data is not purged from the remote application before it has been accessed and secured by all local applications. It is also there to provide information to support staff to enable them to monitor remote access.

A Data Diagram of the control tables required is shown below.

FUJ00098223

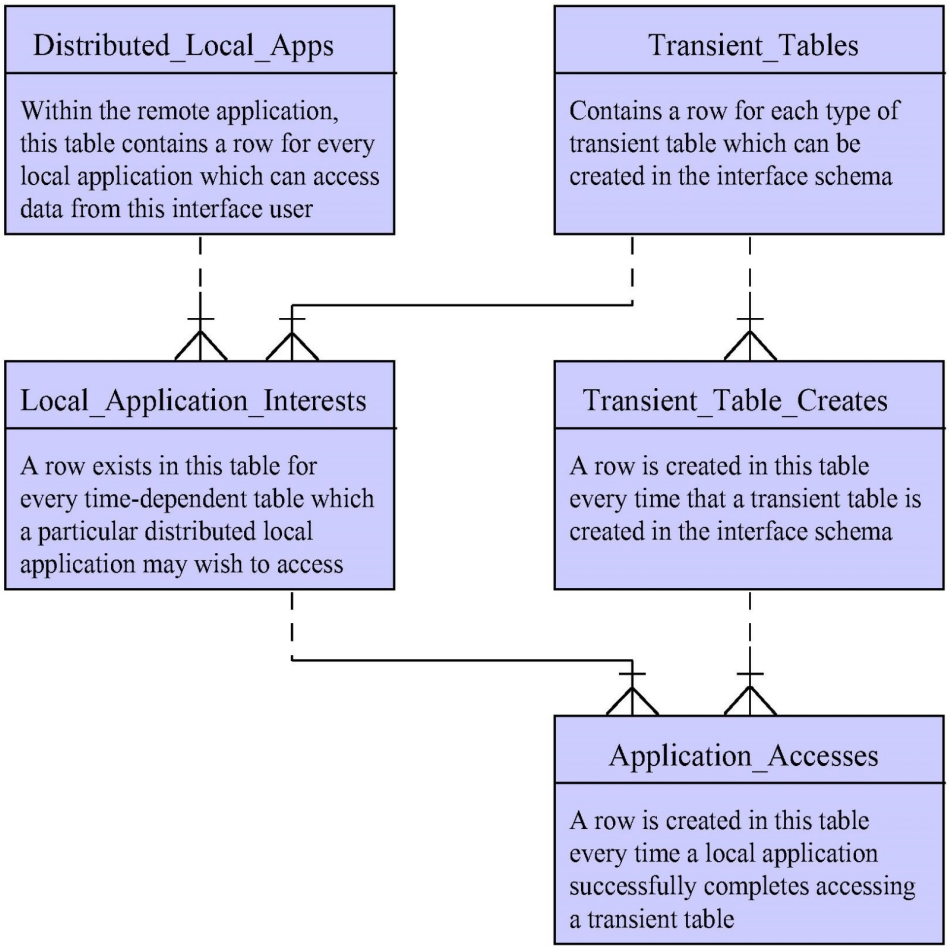

Figure 6 - Data Diagram of Control Tables within the Interface Schema of a Remote Application

These tables are defined in more detail below.

Distributed_Local_Apps			
Column name	*Null?*	*Datatype*	*Description*
Application_Name	N	varchar2(8)	The name of the distributed local application. *Primary Key*
Application_Desc	N	varchar2(240)	A description of the distributed local application
Access_Notes	Y	varchar2(240)	A free-text description of when this application is expected to access the data within the interface tables.

FUJ00098223

Transient_Tables			
Column name	*Null?*	*Datatype*	*Description*
Root_Table_Name	N	varchar2(16)	The name of the transient table before the suffix, as defined in section 3.4.2. *Primary Key*
Maximum_Allowed	N	number(4)	The maximum number of instances of this table which are allowed to exist simultaneously in the interface schema
Archive_Purge	N	varchar2(1)	A or P Defines whether the transient table is archived or purged when a table becomes the oldest table and the maximum number of instances is about to be exceeded.

Local_Application_Interests			
Column name	*Null?*	*Datatype*	*Description*
Root_Table_Name	N	varchar2(16)	The name of the transient table before the suffix, as defined in section 3.4.2. *Primary Key Component 1*
Application_Name	N	varchar2(8)	The name of the distributed local application. *Primary Key Component 2*

Transient_Table_Creates			
Column name	*Null?*	*Datatype*	*Description*
Root_Table_Name	N	varchar2(16)	The name of the transient table before the suffix, as defined in section 3.4.2. *Primary Key Component 1*
Table_Suffix	N	number(10)	Format YYYYMMDDNN where YYYYMMDD is the creation date and NN is an optional second suffix if more than one version of the table can be created on a particular day. *Primary Key Component 2*
Creation_Tsmp	N	date	The date and time of the transient table's creation
Number_of_Rows	Y	number(10)	The number of rows within the transient table

Application_Accesses			
Column name	*Null?*	*Datatype*	*Description*
Root_Table_Name	N	varchar2(16)	The name of the transient table before the suffix, as defined in section 3.4.2. *Primary Key Component 1*

FUJ00098223

Application_Accesses			
Column name	*Null?*	*Datatype*	*Description*
Table_Suffix	N	number(10)	Format YYYYMMDDNN where YYYYMMDD is the creation date and NN is an optional second suffix if more than one version of the table can be created on a particular day. *Primary Key Component 2*
Application_Name	N	varchar2(8)	The name of the distributed local application. *Primary Key Component 3*
Access_Tsmp	Y	date	A timestamp of when the application successfully completed accessing the transient table

In addition to being a control table, the *Application_Accesses* table can also be used as the source of data for an audit trail of transfers of transient data across the interface.

9.5.6 USING SQL TO ACCESS DATA IN DISTRIBUTED DATABASES

The transparent gateway products allow SQL to be used to access data in distributed databases. Ostensibly, any SQL constructs which can be used to access data on the local database can also be used to access a distributed database through a transparent gateway. However, to ensure that incompatibilities do not arise in the future, the following standards must be followed when writing SQL to access data within other applications:

- SQL must conform to the ANSI/ISO SQL standard;
- Row Ids must not be used in SQL statements;
- No assumption must be made about how the optimiser on the remote machine will cause the SQL to be executed. This means, for instance, that SQL "hints" can not be used to encourage the optimiser to perform particular SELECTs in parallel.

9.5.7 HOUSEKEEPING OF THE REMOTE INTERFACE USER

If the interface user within the remote application is used to transfer transient tables, it is necessary for these tables to be housekept at regular intervals. This is achieved by processes which access the control tables within the interface user, and, from the information within them, decide on whether the transient tables should be purged or archived. These processes are scheduled to run nightly after all the day's accesses by distributed applications are expected to have finished.

A typical housekeeping process operates by performing a table scan of the *Transient_Table_Creates* table. For every entry found there it reads the *Local_Application_Interests* table using the root_table_name as a key to discover which distributed local applications are expected to access this table. It then interrogates the *Application_Accesses* table to check that all distributed applications have successfully accessed the table. If this has happened then, depending on the setting of the archive_or_purge attribute on the *Transient_Tables* table, the interface table is either archived (see section 6 for details) and then dropped, or just dropped.

FUJ00098223

Simultaneously with dropping the interface table, the corresponding record in the *Transient_Table_Creates* table is deleted together with all subsidiary *Application_Accesses* records.

This mechanism should pose few problems if the housekeeping process can purge tables at regular intervals. If, however, it is unable to drop any of the transient tables for any length of time, perhaps because one of the distributed applications has been off-line for an extended period, then data is going to build up within the interface user. The housekeeping process must therefore be able to detect whether this is going to result in the number of tables exceeding a pre-defined maximum. It does this by counting the number of instances of a particular table which are in existence, and checking it against the maximum_allowed attribute on the objects table. If the number exceeds this maximum, an alert is raised to allow system support staff to take remedial action.

9.5.8 ACCESSING REMOTE DATA FROM A LOCAL APPLICATION

If a process which is running on a local application requires data from a remote application's database and that database is not available, maybe because the interface tables are locked by the remote application or because the remote database is shut down, then clearly that process is not going to be able to complete successfully. It is important, however, that the process within the local application does not just fall over. The module which is running the extraction process must be coded in such a way that it is aware that the information it is reading is being obtained from a remote source, and that that information may possibly not be accessible at the time.

There are a number of ways that the module can be coded to do this invisibly. The easiest is for the extraction process to run as a daemon which polls the interface tables *Transient_Table_Creates* and *Application_Accesses* in the remote interface user at regular intervals to see if there are any tables which require extraction. If these tables can not be accessed, or the tables to which they refer can not be accessed, then the daemon should wait for a pre-defined period of time and try again. Only if the daemon has waited for an excessive amount of time (again pre-defined) should it fail with an alert to systems support.

FUJ00098223

10. MAPPING APPLICATIONS TO DATABASES

10.1 GENERAL PRINCIPLES

In the Pathway environment there are many database applications which have to be implemented during the lifetime of the project. Most of these applications are intrinsically separate from one another but, at the same time, have some dependencies on one another. This poses a question as to whether particular applications should co-exist with others in the same database, or whether they should exist in separate databases, or whether there should be some sort of half way house whereby some go in one database and some in another.

When faced by this dilemma, the database designer's initial position should always be that all applications should be put together within a single database because this is obviously the simplest implementation. There is little point in adding complexity just for the sake of it. The designer should then allow himself to be argued out of this position.

There are clearly many valid reasons why applications should be put into separate databases. However there is frequently confusion as to what are and what are not valid reasons. It might initially appear logical to have each application in its own database. This impression is often reinforced by the way people refer to applications. For instance, if an organisation has an orders application, it inevitably will be referred to as the "Orders Database" rather than the "Orders Application running on Database X". However this apparently logical solution is only occasionally the best implementation.

Databases should be seen merely as receptacles for applications. As soon as applications are split into separate databases, the complexity of the environment immediately increases as each of the multiple databases has its own systems management, configuration and backup systems to administer.

10.2 APPLICATION SEPARATION CRITERIA

The criteria which need to be considered when deciding on whether applications should share the same database or not are:

1. Do the applications perform broadly similar sorts of functions? For instance, it is unlikely to be suitable for a data warehouse application to be in the same database as a volatile OLTP application.
2. Do the applications have similar availability requirements? It may be unsuitable to have an application which operates an on-line day with limited overnight batch work in the same database as one which operates a 24 hour-a-day Help Desk service.
3. Are the security requirements of the applications similar? It is, for instance, impractical for an application which can be accessed via the Internet to be in the same database as one which contains sensitive information.
4. Are the backup/recovery requirements broadly similar?

FUJ00098223

5. Is the performance of one application likely to be detrimental to that of another?
6. Are all the applications in the database equally well tested? If an application is likely to fail and require point-in-time recovery, it should not co-exist in the same database as those that are not. One of the reasons for this is that Oracle does not support partial recovery of a database to a point-in-time: either all the applications in the database are recovered to a particular point-in-time or none of them are.
7. Are the resilience requirements of the applications similar? Applications should not share the same database if one requires site failover disaster recovery using SRDF on the EMC discs and the other does not.
8. Is there are a large volume of data which has to be passed between applications? If so, then performance is likely to be optimal if they share the same database. If it is decided to do this, it is, of course, still essential for the applications to communicate with one another using the mechanisms described in section 9.5.

FUJ00098223

11. PERFORMANCE

This section contains advice on how applications should be written so that they perform optimally. Although it is aimed specifically at applications which make use of Oracle on the Sequent SMP machines using Dynix, the advice within it is also likely to be relevant to Oracle applications running on other platforms.

11.1 HINTS AND OPTIMISERS

There are two different optimisers provided with the Oracle RDBMS; the Cost Based Optimiser (CBO) and the older Rule Based Optimiser. Within Pathway all applications should make use of the CBO as this is likely to provide the best performance for the majority of queries.

The main problem with using the CBO is that its optimisations are dependent on the size of the tables within the database, and on how frequently those tables are analysed using the ANALYZE function. This means that the optimisations arrived at by the CBO are likely to be different for small test databases to those arrived at for full size live databases. This makes it difficult to test the performance of the live systems.

The recommendation is therefore that "hints" are always used with the CBO for all but the most simple of queries. A hint overrides the CBO and ensures that the query is performed in the manner intended by the designer, whatever the populations of the underlying tables.

11.2 OPTIMISATION OF SELECTS ON THE SMP PLATFORM

The host system Sequent SMPs used by Pathway each have within them a minimum of ten CPUs. Consequently the Parallel Query Option (PQO) should be used for nearly all queries on tables with significant populations.

It is essential that all queries which use PQO are volume tested on a life size test rig so that the optimal query mechanism can be specified within the query's hint.

11.3 INDEXES VS. PARALLEL JOINS

It is beyond the scope of this document to go into great detail about the use of PQO. Detailed information on the use of PQO can be found within the Oracle7 Server manual entitled "Tuning". However, it is worth noting that experience with PAS/CMS has shown that, for almost any batch process which accesses more than about 1% of a table, better performance is obtained by using a parallel, full table scan with a hash join than is obtained by using table indexes.

11.4 RECORD DELETION

Record deletion using the SQL DELETE function is usually slow, can impose significant performance overheads on the remainder of the system, and can lead to fragmented tables.

FUJ00098223

It is recommended therefore that, if many records are frequently to be deleted from a table, the records are not actually deleted at the time of their deletion but are instead marked as being logically deleted by updating a logical_deleted column defined on the table. The logically deleted records are then removed by a housekeeping procedure that runs at a time of low activity. If there are no foreign key constraints specified on the table, the most efficient method of achieving this is to use the "Create Table… As Select… Unrecoverable" construct to create a new table from the old using a query that would only select those records with a null logical_deleted column.

11.5 LOADING

Experience with PAS/CMS has shown that, if a large number of records are to be loaded into a database in as short a time as possible, it is more efficient to run a number of separate unrecoverable processes (DIRECT=TRUE, PARALLEL=TRUE) each of which loads a part of the table than it is to make use of the Oracle Parallel Direct Load function.

11.6 FOREIGN KEY CONSTRAINTS

Foreign key constraints should be used with caution as their presence imposes significant overheads for all processes which either insert new records into tables or update the foreign key columns within existing records. This is particularly pertinent to the host applications in which large numbers of transaction records are regularly loaded and unloaded.

If a foreign key constraint is defined, the join column within the superior table must be indexed unless there are compelling reasons against doing this.

11.7 WRITING OUT DATA TO FLAT FILES

Although still relatively slow, the most efficient method for writing records from an Oracle database to flat files is to use C programs which select arrays of records from the Oracle tables and then write them directly to files. If many files are to be written in as short a period of time as possible, then the number of these processes can be paralleled to a high degree.

11.8 USE OF ORACLE SHARED LIBRARIES

In order for compiled pro*C programs to run successfully, it is necessary for them to be linked to Oracle libraries. This linking can either be static or dynamic. If it is static, a physical copy of the library is attached to the executable program when it is compiled; if dynamic, the program dynamically links to a shareable Oracle library at run-time.

There are two main advantages to using dynamic linking:

1. The size of the executable program is considerably smaller leading to gains in memory efficiency.

FUJ00098223

2. If dynamic linking is used for all programs, it can be guaranteed that the same version of the Oracle libraries is being used by all programs.

Performance testing of the TPS application has shown that dynamic linking to shareable libraries imposes no perceptible performance costs.

All Pro*C programs developed for the Pathway host applications should therefore use dynamic linking to shared Oracle libraries.

FUJ00098223

12. BACKUP AND RECOVERY

12.1 THE REQUIREMENT FOR BACKUPS

All of the data for the live host system applications should be contained within databases which exist on the mirrored discs within the EMC Symmetrix Storage Units. The discs within these units are not only mirrored locally, they are also replicated and mirrored at a remote site by means of SRDF over the E3 telecommunications link. There are always therefore four copies of the live database in existence: two copies mirrored locally and two remotely.

The database is therefore inherently secure both against media failure and site loss. Although this means that it is exceedingly unlikely that database recovery will ever be necessitated by hardware failures, it does not remove the requirement to back up the database because corruptions on all four mirrors can still conceivably occur as a result of:

- operator or DBA errors;
- application program or scheduling errors;
- incorrect external data being erroneously loaded into the database;
- catastrophic failure of the EMC Storage Systems.

It is therefore essential to ensure that recent backup copies of the database exist at all times so that the database can be recovered, without the loss of any data, should any such corruption occur.

12.2 BACKUP STRATEGY

With the exception of the PAS/CMS application, none of the host applications which are currently known about have a requirement for a database which is up and running for twenty four hours a day. This means that all host applications can be backed up whilst the database is shut down. This makes for a considerably simpler backup strategy than would be necessary were it not possible to shut down the database.

The strategy to be adopted by all host applications involves the taking of a cold backup after the database has been shut down. The strategy must have the following features:

- The backup processes must all be scheduled by Maestro.
- After all the overnight batch processes have finished, the database is shut down for the backup.
- A cold backup of all the raw volumes and filesystems underlying all the database's control files, online redo logs, archived redo log files and data files is made to tape twice each day.

FUJ00098223

- All backups taken are made to DLT tape in the standalone tape libraries. The OSBM and ESBM software is used to perform the backups and manage the tapes so produced.
- Once the backup has finished successfully, all archived redo logs produced since the previous backup are deleted from disc.
- The database is restarted once the backup has been successfully completed.
- A cycle of seven backups (i.e. backups for a week) is maintained by ESBM.
- After the backup has been completed, the second tape copy is removed from the tape library and transported to the remote Pathway computer site where it can be used to recover the database there should this prove necessary.

Unless the application has some very specific backup and recovery requirements, no bespoke code should be written to implement any part of an application's backup or recovery strategy. All that is necessary is for the OSBM and ESBM backup management products to be configured to perform the backup, and for Maestro scripts to be updated to include the backup in the daily schedule.

FUJ00098223

13. RESILIENCE

13.1 MIRRORING

All the database data files for a live host application must be placed on EMC Symmetrix discs, which are locally mirrored using the standard EMC hardware mirroring facilities.

For Oracle databases, the online redo logs should also be mirrored at the EMC hardware level. Mirroring at the Oracle level using redo log groups is not recommended. If it is decided to invoke redo log archiving, the redo log archives should also be mirrored on the EMC discs.

13.2 DISASTER PROVISION

It must be possible to recover all host applications without the loss of any data should the site at which the applications normally run be totally destroyed.

This requirement is satisfied for the Sequent systems by placing all live host system databases on the EMC Symmetrix discs. These discs must be locally mirrored using the EMC hardware and remotely mirrored using SRDF, as illustrated in the diagram below.

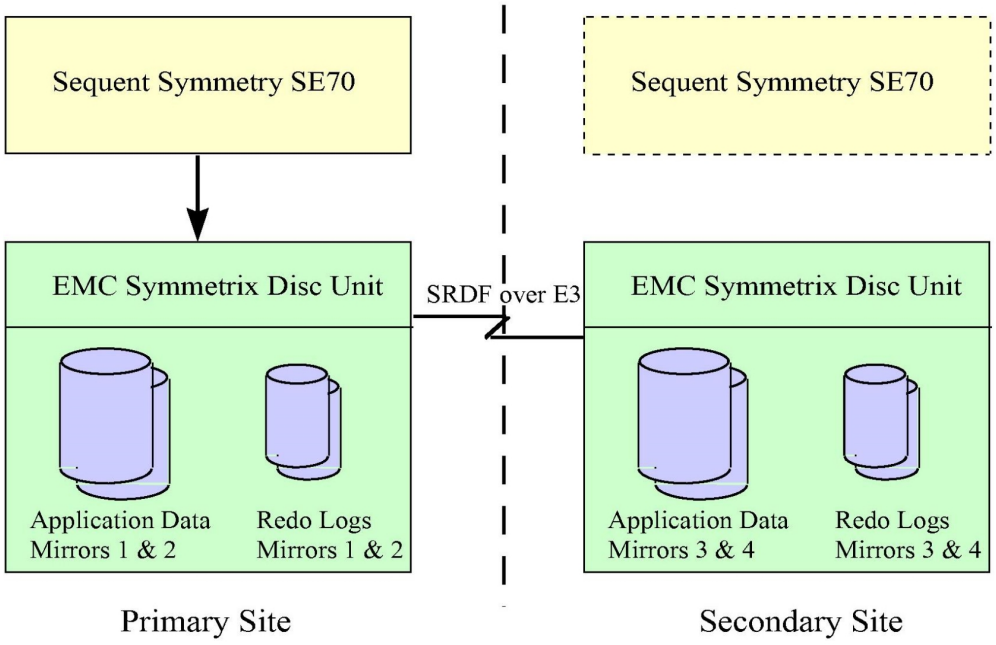

Figure 7 - Fully Resilient Host Application Configuration

FUJ00098223

14. CONSISTENCY CHECKING

14.1 REQUIREMENT FOR CONSISTENCY CHECKING

The logical and physical consistency of an application's data on the database must be checked at regular intervals. This is necessary for three reasons.

- If the database is shown by the consistency procedures to be corrupt, this allows the inconsistencies to be identified and fixed at an early stage, hopefully before they cause application failures or spawn more inconsistencies.
- If the database is shown by the consistency procedures to be clean, this gives support staff and users confidence in the quality of the data held within the database.
- If an application module fails or reports exceptions due to inconsistent data, a consistency checking procedure can be run to ascertain the extent of the corruption.

14.2 CONSISTENCY CHECKING PROCEDURES

For Oracle-based applications, the physical consistency of the data within the application's schema should be checked regularly by using the DB_VERIFY utility on the backup files.

If an application's relationships between tables are defined on the repository using foreign key constraints and the schema is generated directly from the repository, then the scope for there to be logical inconsistencies within the database is vanishingly small. This is because, when foreign keys are defined, the RDBMS checks for logical consistency before records are inserted, updated or deleted. Often, however, foreign keys are not so defined because their existence can lead to poor performance. This is caused by the requirement to access superior records using the join column whenever subsidiary records are inserted, deleted or have the join column updated (see section 11).

Wherever there are logical relationships between tables (or, in fact, between columns within the same table) that are not implemented using foreign key constraints, a module (or modules) should be written that specifically checks for the existence of records in the superior tables for all records with non-null values in the join columns in the subsidiary tables. This module should then be scheduled to run at least once a month.

If the module finds any inconsistencies, it should write them to an exceptions table (see section 8) and raise an alert.

14.3 RESOLUTION OF INCONSISTENCIES

Logical or physical inconsistencies are detected within an application's data either by the consistency checking procedures or by the application producing incorrect results. If such inconsistencies are detected then the first consideration should be given to

FUJ00098223

recovering the database using a recent backup and then performing point-in-time recovery to a time before the corruption occurred.

Often, however, this course of action is not practical, either because the source of the corruption can not be easily identified, or because the corruption happened so long ago that it is not possible to recover updates which occurred after the point-in-time to which the recovery must be performed. It should additionally be noted that point-in-time recovery is not likely to be a viable option at all for applications which include on-line updating via a Help Desk or via some other interactive function.

If point-in-time recovery is not possible, then the actions which should be taken to resolve the situation are as follows:

1. The scope of the problem must be ascertained by querying the live database using the appropriate query tool.
2. Assuming that recovery is not possible, the corrupt parts of (or all of) the live database are copied to a test database.
3. Scripts are developed to back out the corruption.
4. Said scripts are tested on the test database.
5. Live database is backed up.
6. Scripts are run on the live database.
7. Live database is checked for correctness using the appropriate query tool.

FUJ00098223

15. SECURITY

15.1 SECURE ORACLE BUILD

15.1.1 SECURITY REQUIREMENTS

The security requirements of Pathway's application databases are fully defined in refs. [2] and [3]. In order to meet these requirements, it is necessary for the application databases which make use of Oracle to be built such that they use the security facilities provided by the RDBMS. The following sections define the actions that need to be taken to satisfy the requirements for the Oracle RDBMS version 7.3.4.

15.1.2 DATABASE LINKS TO INTERFACE USERS

The use of database links to communicate information from one database to another is described in section 9.5. This method of communication requires the existence of an interface user for the link within the database from which the information is to be extracted. To ensure the security of the defined link, the following principles must be applied:

- When a database link is created on the local machine, it is necessary to specify both the username and the password of the interface user on the remote machine to which the local application is to connect[1]. As a consequence of this, the password of the interface user is visible on the local machine within the SYS.LINK$ table. Although only those with the DBA role applied can normally see this table, it is still necessary to minimise the security risk which this limited visibility provides.

  As a consequence, it is essential that the interface user on the remote database has the minimum number of privileges granted to it and that only those objects which are required for the link exist in that user. Neither UPDATE, INSERT, DELETE nor ALTER privileges must not be granted to the interface user for any of the application tables which exist outside of that user. In this way any damage which could be done by malicious access via the database link is minimised.

- The initialisation parameter DBLINK_ENCRYPT_LOGIN should be set to TRUE for all databases. This ensures that the password is always encrypted when passed across the link.

15.1.3 SESSION LEVEL AUDITING

All Pathway databases must have auditing enabled such that audit information is written to the database table SYS.AUD$[2]. This is achieved by setting the

[1] It is possible to get around this restriction by ensuring that the userid and password on the local machine are the same as those of the interface user on the remote machine. This technique is, however, not recommended as it creates artificial dependencies between the databases.

[2] By default, the SYS.AUD$ table is placed in the system tablespace. To avoid contention with system objects, this table should be re-created in another tablespace.

FUJ00098223

AUDIT_TRAIL initialisation parameter to DB, and invoking the AUDIT SESSION command.

Auditing should be enabled using the following statement audit options:

ALTER SYSTEM	Audits all changes to the Oracle system.
ROLE	Audits all creates, alters, sets and drops of roles.
SESSION	Audits all connects and disconnects.
SYSTEM GRANT	Audits all grants and revokes of roles/privileges to/from users/roles.
USER	Audits all creates, alters, sets and drops of users.

The audit trail should itself be protected by invoking the command below as part of the initial build.

```
AUDIT INSERT, UPDATE, DELETE
  ON sys.aud$
  BY ACCESS;
```

15.1.4 ORACLE USERS

The minimum number of Oracle users with the minimum number of privileges granted to them should be set up for a host system application. The standard users with which an application should be delivered by developers are defined in section 15.2.4. The manner in which these and other users should be set up are described in this section.

15.1.4.1 APPLICATION USERS AND BATCH/DAEMON PROCESSES

Most of the batch and daemon processes that are run within a host system application are processes that are directly invoked via Maestro from the host system's operating system. Oracle does not therefore need to perform user authentication for these processes as this will already have been performed by the operating system.

Consequently, for a Pathway application, the Oracle user that owns the application's schema, and under which all batch and daemon processes must be run, should be set up as an externally authenticated user. An externally authenticated user, in Oracle terms, is a user for which password authentication is not done by the RDBMS.

By default, for an Oracle database, externally authenticated users all have a prefix of ops$, though this prefix can be changed for particular databases by setting the OS_AUTHENT_PREFIX initialisation parameter. For Pathway applications, the ops$ prefix should be retained on all databases so that it is obvious which are the externally authenticated users.

As with all other Oracle users that are set up within a Pathway database, the user that owns the application schema should be set up with the minimum number of system privileges necessary to run the application. This means, for example, that the application owning user should never be granted the DBA role.

FUJ00098223

15.1.4.2 HUMAN USERS

All host system Oracle databases should be delivered to CM by the development teams with no users defined through which humans can connect in client/server mode. The only exceptions to this rule are SYS and SYSTEM, as described in section 15.1.4.3, and those users that are necessary for the successful operation of products as described in section 15.1.4.4.

As part of the build process, the SYSTEM user should be used to create the necessary human users to enable the application to be run and supported in the manner required by the environment into which the application is being installed.

15.1.4.3 SYS AND SYSTEM USERS

The developers of an application should initially assign obvious passwords, which are defined in the database's Installation Guide (see section 4.1), to the SYS and SYSTEM users. During the installation process, the installers should use the SYSTEM user to perform any necessary system installation procedures and to set up any necessary human users. Once this has been completed, the SYS and SYSTEM users should be disabled by assigning to them impossible[3] passwords, such that no human user can connect into them.

Subsequent to this, any necessary database administration activities are undertaken by a CFM user who has the CFM_DBA role enabled.

15.1.4.4 PRODUCT USERS

Many products that make use of an Oracle database have their own users created for them in the database. Two products that are used on most Pathway Oracle databases and that fall into this category are BMC's Patrol and Oracle's Discoverer. The set up of Patrol on the server creates a user called PATROL and the Discoverer administrator creates a user for the public end user layer.

As with the SYS and SYSTEM users, any users created for products should be assigned obvious passwords when the database is first delivered. The installers of the application should then change these passwords and secure them once the installation of the application is complete.

15.1.5 USE OF INTERACTIVE SQL*PLUS

Under normal circumstances SQL*Plus should not be used interactively on any of the Pathway production databases. If, however, an emergency requires that the product is used in this manner, then the following safeguards must be taken to minimise the possibility of misuse.

- SQL*Plus must never be used in client/server mode. To access SQL*Plus, a user must first log into a Dynix session via COS/Manager and then use operating system authentication (i.e. using an OPS$username if the OS_AUTHENT_PREFIX initialisation is set to OPS$) to connect to Oracle using the command 'sqlplus /'.

[3] An impossible password is one that contains one or more characters that are not alphanumeric characters from the database character set. To set such a password, the command used could be: ALTER USER scott IDENTIFIED BY "no way@all!".

FUJ00098223

- There exists a table in the SYSTEM account, *PRODUCT_USER_PROFILE*, which is there to provide additional security at the product-level. By inserting rows into this table, users can be prevented from using SQL*Plus to execute specified DML, DDL or SQL*Plus commands. The table below defines the minimum rows which should be inserted into this table for all Pathway production databases.

Columns in PRODUCT_USER_PROFILE				*Reason*
PRODUCT	*USERID*	*ATTRIBUTE*	*CHAR_VALUE*	
SQL*Plus	%	CONNECT	DISABLED	Prevents all users from connecting to the database as a different user.
SQL*Plus	%	HOST	DISABLED	Prevents all users from invoking host system commands from within the SQL*Plus session.
SQL*Plus	%	NOAUDIT	DISABLED	Prevents any user from disabling auditing.

15.2 ACCESS CONTROL

15.2.1 ACCESS CONTROL DESCRIPTION

Each application that is developed as part of the Pathway programme must have associated with it an Access Control Matrix. This matrix specifies who can access what. It cross references the various groups of users with the schema objects to which they require access.

Assuming that the roles and users have been set up correctly in the Designer/2000 repository, the matrix could be generated directly from the data therein.

15.2.2 STANDARD ACCESS CONTROL MATRIX FOR ALL PATHWAY APPLICATIONS

Users requiring access to an application database are first of all grouped together by virtue of their requirements to access common objects within the database. These user groupings map on to roles as defined for Oracle databases. A role is made up of a set of privileges to perform various actions upon various objects within the database[4]. A role can be assigned to a user or another role.

The minimum set of roles that must be defined for all of Pathway's application databases are defined in the table below.

Role	***Expected Users***	***Human Users?***	***Objects Accessed and Type of Access***
AUDITOR	Internal and External Auditors	Yes	As for MONITOR, plus the ability to interrogate the Oracle audit table (SYS.AUD$) and views.

[4] In Oracle terms, these privileges are Object Privileges. Oracle Roles can also include System Privileges which allow the grantee to perform particular database operations or classes of database operations. Other than the CFM_DBA role which has the DBA role applied to it, none of the roles defined in the table above include any system privileges other than those bestowed by virtue of having the CONNECT and RESOURCE roles applied.

FUJ00098223

Role	*Expected Users*	*Human Users?*	*Objects Accessed and Type of Access*
BMC	BMC Patrol & Knowledge Modules	No	Select and update access on all the exception tables in the database, plus the ability to interrogate the Oracle audit table (SYS.AUD$) and views.
BSU	Pathway Business Support Unit	Yes	As for MONITOR, plus the ability to update / insert application objects for which the BSU have pre-defined forms provided.
CFM_DBA	CFM database administration - privileged DBA group	Yes	Full DBA privileges for the CFM database administrators
MONITOR	All users who require query only access to the application. This includes all users of query tools such as Discoverer.	Yes	Select only access to all objects within the application.
SECURITY_ MANAGER	Support staff who are authorised to administer support users and to investigate security breaches.	Yes	Systems privileges for maintaining users and for selecting all tables and views in the database. Systems privileges assigned: GRANT ANY ROLE CREATE ANY USER ALTER ANY USER DROP USER SELECT ANY TABLE
SSC	Pathway SSC	Yes	As for MONITOR, plus the ability to update / insert application objects for which the SSC have pre-defined forms provided.
TMS	TMS Agents only. No human users are expected to use this role	No	Select / Update / Insert privileges as required on all objects which are accessed by the TMS Agents.

In addition to these roles, one or more application specific roles must be provided to allow the users assigned for running the batch processes access to the application objects.

15.2.3 HYPOTHETICAL EXAMPLE OF AN ACCESS CONTROL MATRIX

In order to clarify what is required, there follows an example of an access control matrix for a hypothetical cut-down CMS database. This database might contain the following ten tables which are split into object groupings.

Object Name	*Object Group*
action_audit_trails	Application Control
application_parameters	Application Control
card_order_requests	CMS Objects
cardholder_excptns	Exception Objects
cardholders	CMS Objects
cards	CMS Objects
mis_ordered_cards	MIS Objects shared with CMS
process_audit_trails	Application Control
tms_rx_card_events	TMS Objects shared with CMS
tms_tx_cardholder_changes	TMS Objects shared with CMS

FUJ00098223

The groups of users who may access these object groups could be as described in the table below.

Role	*Target Users*	*Human User?*
AUDITOR	Auditors	Yes
BMC	BMC Patrol & Knowledge Modules	No
BSU	Pathway Business Support Unit	Yes
CFM_DBA	CFM database administration - privileged DBA group	Yes
CMS	All CMS functional processes	No
DW	Data Warehouse extraction procedures	No
MIS	MIS extraction procedures	No
MONITOR	All query only users	Yes
SSC	Pathway SSC	Yes
TMS	TMS Agents	No

The access types used in the access control matrix itself are defined below.

Access Type		*Meaning*
D	-	Delete rows from table
E	-	Execute DDL (e.g. TRUNCATE, CTAS)
I	-	Insert into table
Q	-	Query, or select of rows from table
U	-	Update rows in table
All	-	All privileges available (Q, I, U, D and E)
A	-	Alter Tables (e.g. Add/Delete columns to/from tables)
--	-	No access allowed

The Access Control Matrix would then appear as below.

Role	*Object Groupings*					
	Application Control	CMS Objects	TMS Objects (shared with CMS)	MIS Objects (shared with CMS)	Exception Objects	Oracle Audit Objects
AUDITOR	Q	Q	Q	Q	Q	Q,I
BMC	--	--	--	--	Q,U	I
BSU	Q	Q	Q	Q	Q	I
CFM_DBA	All	All	All	All	All	All
CMS	Q,I,U	All	Q,I,U,D	Q,I,U	Q,I	I
MIS	Q	--	--	All	--	I
MONITOR	Q	Q	Q	Q	Q	I
SSC	Q,I,U,D	Q	Q	Q	Q	I
SECURITY_ MANAGER	Q	Q	Q	Q	Q	Q
TMS	Q	--	All	--	--	I

15.2.4 STANDARD USERS FOR ALL PATHWAY APPLICATIONS

When an application is first handed over by the developers, it must have a basic set of Oracle users set up. These users, and the roles which need to be assigned, are defined in the table below.

User	*Grants*
bmc_user	granted CONNECT, MANAGE TABLESPACE privileges granted BMC role
ops$application (this is the user that contains the application's	granted CONNECT, RESOURCE privileges + any other application specific privileges

FUJ00098223

User	*Grants*
schema)	
tms_user	granted CONNECT, RESOURCE privileges granted TMS role

In addition, when the application is first delivered, the SYSTEM and SYS users are enabled. It is only after the system has been built and the "human" users set up, that the SYSTEM and SYS users are disabled as described in section 15.1.4.

15.2.5 DOCUMENTATION OF THE ACCESS CONTROL MATRIX

The access control matrix for an application must be documented fully on the Designer/2000 repository using the method described in section 4.7.

15.3 SECURITY OF EXTERNAL INTERFACES

All access to or by agencies which are external to Pathway must conform to the requirements laid down in ref. [2] and [3].