1992/85
copy 4

AUSTRALIAN GEOLOGICAL SURVEY ORGANISATION

Record Number

1992/85

Dual Oracle Database Environment and Change Control Management

Mirek Kucka

AGSO

AUSTRALIAN GEOLOGICAL SURVEY ORGANISATION

AGSO
AUSTRALIAN GEOLOGICAL SURVEY ORGANISATION

Dual Oracle Database Environment and Change Control Management

Record 1992/85

Release 1.6

M. Kucka
Database Administrator
Information Systems Branch

October 1992

R9208501

ISSN 1039-0073
ISBN 0 642 18497 6

Table of contents

SCOPE

The objective of this document is two-fold. Firstly it formally establishes the concept of AGSO's dual Oracle environments, namely the Test Oracle environment and the Production Oracle environment, and secondly it establishes the concept of change control management within both Oracle environments.

These guidelines are to be implemented as part of the migration of databases from the current Data General proprietary MV environment to the newly acquired AViiON 6240 Oracle Database server. They also apply to new databases to be established on the AViiON.

INTRODUCTION

One of the difficulties in writing a paper discussing technical points is avoiding the use of specific technical terminology (jargon). However, in reality it is sometimes necessary to use this jargon to avoid circumlocution and to ensure that when communicating ideas, the terms used will be clearly understood by both parties.

This paper uses jargon only where necessary to synchronise the concepts, ideas and terminology presented in this paper with other Oracle users. For example, when the term **table** is used, it will mean a relational database concept of a two dimensional matrix composed of columns (description of the fields within a record, such as names, phone, address etc) and rows (the actual data of the record, i.e.: Fred Bloggs, 06 2499111, GPO Box 378). To use any other terminology other than **table** in this case, would be unworkable.

To this end, a glossary of terms has been included.

This document assumes that the reader has had some experience with the Oracle Relational Database Management System (RDBMS), and is also familiar with the concepts and terms discussed in 'Guideline for Oracle userids' which has been circulated to all identified system owners and Heads of Program. In particular, the terms 'system owner' and 'system administrator' are synonymous.

BACKGROUND

The corporate data held on AGSO's Oracle database has grown not only in size, but also in importance and realised commercial value. Coupled with the need to share the data not only between different areas within AGSO but also with outside organisations, the Oracle database environment must now be revised to ensure that it can support current and future needs.

WHY THE NEED FOR MULTIPLE ORACLE ENVIRONMENTS

When the Oracle RDBMS was first purchased in 1986 for use within BMR, very little effort was expended towards planning, organising and managing the Oracle database environment.

As a result, AGSO has one instance of Oracle RDBMS running on the Data General MV computer, which comprises both production and development/maintenance data and workloads. Consequently within this single environment integrity may be compromised both in the structure of the data and also in the data itself (for example, programs under development are usually tested against real live production data), with potentially disastrous consequences.

It must also be recognised that many of these databases are National Geoscience reference collections, where in some cases AGSO is acting as the physical and systems custodian of data owned or deposited by outside organisations.

AGSO also intends to allow on-line external access to legitimate users of the data and associated tables, once the new Unix environment is secured and auditable.

The means for addressing these problems are through the establishment of two separate and distinct Oracle environments, each with its own data and data dictionaries, which are termed Test Oracle environment, and Production Oracle environment - which in this paper we will simply term Test environment and Production environment, both residing on the same hardware platform (see Appendix A).

Advantages of multiple environments

There are many advantages in having multiple environments, some of the more important ones from a user perspective are:

- development, maintenance and testing of the programs occur in a non critical database, therefore 'unexpected' and 'undocumented feature' changes to data (more commonly known as bugs), can be identified and fixed before corrupting live production data
- since the Test environment will hold less data than Production, all programs will run faster, therefore increasing turnaround
- much higher levels of data integrity can be maintained, both in data structure and in the actual data
- an auditable chain of changes to the Production systems can be maintained

Production Environment

The Production environment is where only production work is to be carried out. It will be a tightly controlled environment, having system particulars and userids as set out below:

system particulars:

- high system priority
- high level of recoverability
- no access to SQL*Forms generation
- eventual performance monitoring of both the RDBMS and user programs
- the possibility of automatic enforcement of password changes, based on time interval
- no development work

system userid: this class of userids is the only group in Production that will have the 'resource' attribute. However, in Oracle Version 6, these userids will **not** have access to 'alter','create' and 'drop' data definition language (DDL) verbs. They will still retain the ability to grant and revoke access rights to the system's objects.

general userid: this group of userids will have no 'resource' attribute, only the 'connect' attribute. This group will still retain the ability to create synonyms and views, but will only be able to manipulate the data that they have been granted access to, through SQL*Forms, SQL*Plus, SQR etc.

Test Environment

The Test environment is where training, and all the development and testing will be performed, such as designing new databases or modifying existing ones, as well as developing or modifying programs (such as SQL*Forms, SQL*Plus reports etc). The Test environment will be basically unrestricted, having system particulars and userids as set out below:

system particulars:

- most userids will be restricted in the total space that they can allocate, as the Test environment should generally only contain test data or subsets/fraction of the Production data. However, due to the nature of AGSO's work there will be instances where certain userids will not be limited, as they may require large space allocations for their particular needs (such as massaging data from other sources, AGSO or external)
- lower priority to that of Production. It should be noted that even though the Test environment will have lower priority than Production, in most cases, the response time from Test will be faster than Production due to the smaller size of data that has to be accessed/manipulated
- lower level of recoverability

system userid: will have 'resource' attribute with no restrictions

general userid: will have 'resource' attribute with no restrictions, provided that the system owner decides to give 'resource' to that general userid

DEVELOPING DATABASES WITHIN THE TEST ENVIRONMENT

All the database design, implementation and maintenance occurs in the Test environment. Even though this section does not really deal with the issues of change control management, it is nevertheless included to give suggestions on database design.

Data analysis and database design are fundamental in designing a successful system. They are also quite complex and cannot be really adequately covered here. However, since they play such an important role in system design, a very brief overview of the three major 'tools' of data analysis/database design are described.

The most commonly used 'tools' in database design can be grouped into transaction analysis, normalisation process and volume sizing information, which are briefly described below. While not being the only 'tools' used, they are usually the most often either ignored or misunderstood. [Note that CASE (Computer Aided Software Engineering) tools make the process of database design much easier, provide a consistent approach to database design methodology, as well as providing system level (as opposed to user level, i.e. guidelines) documentation].

Transaction Analysis

Anticipated transaction rates through the system, when used with the normalisation process (and to a lesser extent sizing information), will largely dictate the final physical database design, as well as providing useful documentation.

Transaction analysis:

- allows prioritisation of transactions that have to be developed first, and depending on how many transactions are going to be run within a timeframe, which transactions need special attention with regard to optimising response time
- enables grouping of transactions, i.e. this group of transactions will modify the invoices, this group will access customer information etc, which may be important from a security viewpoint, or this group of transactions needs to be written first etc.
- allows you to double check what entities within the system are going to be accessed, and which are not (in which case either the transaction was missed out, or the entity is superfluous and hence should be deleted)

Transaction details should contain:

- details of each type of transaction (update/retrieval transaction, tables accessed, expected number of rows processed)
- the frequency with which the transactions are going to be run
- response time required for that transaction

A blank form in Appendix B has been provided for this purpose.

Normalisation

The process of normalisation is designed to prevent update anomalies and data inconsistencies. By normalising the database design, multiple occurrences of a particular data item are avoided (for example, a person's address is only stored once, so when updating the person's address, the update takes place only once, and on only one record).

Even though there are now 5 normal forms in relational database theory, generally only the first three normal forms are used.

1st Normal form	repeating groups are removed
2nd Normal form	non-key attributes are removed which are dependent on part of the key attribute
3rd Normal form	non-key attributes are removed which are dependent on any other non-key attribute

While it is highly desirable to normalise the database design to 3rd Normal form from an update viewpoint, the retrieval of data can dramatically slow down, as many table joins may be needed in order to get the desired information.

It is recommended that all systems be normalised to 3rd Normal form, and then, based upon the transaction rates that are anticipated to run against the system and the performance required of the system, be de-normalised to achieve faster response time.

By normalising the system to 3rd Normal form and then de-normalising the design for performance, an accurate list of data redundancies can be produced. These redundancies will then have to be maintained across updates (to ensure data integrity) by the programs that manipulate the data. (Oracle version 7 allows this to be built into the database design itself).

Sizing Estimates

Even though it is acknowledged that it is difficult to obtain accurate sizing figures, this information is necessary for calculating the space requirements for tables and indexes. Sizing information should include details such as:

- the average and the maximum expected number of rows in each table
- the expected growth rate of each table
- total size required for the tables (based on the expected average number of rows). Blank forms for table sizing estimates are in Appendix C.
- total size required for the indexes (based on the expected average number of rows). Blank forms for index sizing estimates are in Appendix D.

Please refer to the chapter on Space Management of the *'ORACLE Database Administrator's Guide'* for a detailed discussion on how to calculate the space requirements for tables and indexes.

General issues

There are several other issues to keep in the back of the mind when designing/modifying database systems:

- all new systems or new system objects should be created by the system userid
- clustering should be avoided, unless there are definite benefits in clustering tables
- keep the number of indexes to a minimum. Generally there is no benefit in indexing tables that have less than 50 rows.
- use fully qualified table names (i.e. <creation userid>.<tablename>), as this increases performance
- when using fully qualified table names in SQL*Forms, there is no need to have individual synonyms set up for each userid that uses the forms
- when altering or deleting attributes within a table, be aware of possible repercussions for other dependant systems.

- be aware that views may need to be re-created if one of the base tables being used in the view has been altered
- all tables must have informative comments about the table included in the table definition. Use the SQL*Plus command:

  COMMENT ON TABLE *<tablename>* IS *<information about the table>*;
- do not store SQL*Forms in the Oracle dictionary - store them in the Operating System directories
- document the system thoroughly. The reasoning behind the chosen physical design should be well documented so that new staff will not puzzle over the database design.

MIGRATION

Since all the development and testing is carried out in the Test environment, and the Production environment is secured against changes to the structure of the data, a mechanism to migrate changes from the Test environment to the Production environment needs to be established.

There are really two types of migration, namely the migration from the Test environment to the Production environment (which will be through the Database Administrator, or a program librarian), and the internal migration within the Test environment itself (which is totally under the control of the system owner).

Note that only the object definitions/structures will be migrated into the Production environment, and not the actual data. Also, we will be only referring to database objects such as tables, views etc, and not programs such as SQL*Forms etc.

MIGRATION WITHIN THE TEST ENVIRONMENT

Within a typical area, there may be several people working simultaneously on the development/maintenance of different parts of the system. Each of these people should be working on their own copy of the system objects, so that the data and the data structure of the existing system will remain intact while development is taking place.

System owners should test the new modifications and if accepted, the system owner should either copy the new, or alter the existing system objects using the system userid (so that all the objects belonging to the system are again owned by the system userid).

The copies of objects that were just promoted by the system userid should be deleted to stop the proliferation of duplicate objects.

One of the great advantages of having dual environments, is that there are always two copies of the data structure. That is, the data structure can always be copied back into Test by the system owner, should there be an 'accident' with the Test data structure.

The actual mechanism for this internal migration is up to the system owner to decide and control.

MIGRATION FROM TEST TO PRODUCTION ENVIRONMENT

Once the testing has been completed and the objects accepted in the Test environment, the system can then be migrated into Production. The process of migrating objects from the Test environment to the Production environment can only be instigated by the system owner, to ensure that he/she is fully aware of all the changes to the Production system that he/she is responsible for.

It should be stressed that only the objects owned by the system userid will be migrated into the Production environment. Under no circumstances will objects owned by general userids be migrated.

The rationale behind this controlled migration is to ensure that :

- changes to the Production environment only occur at the system owner's request
- the changes to Production systems are auditable
- Test and Production environments are 'synchronised', with the Test environment being at the same level or slightly ahead of the Production environment
- all objects owned by the system userid make up the whole of the system

Responsibilities of the DBA

The migration of objects from the Test environment to the Production environment will be through the Database Administrator (DBA) or the Program Librarian (see section 'The role of the Database Administrator or Program Librarian' later on). It will be the DBA's responsibility to ensure that the requested objects are migrated to the Production environment properly, i.e. the objects will be the same in both the Test and Production environments after the migration.

The DBA will coordinate the timing of the migration to the Production environment with the system owner, as the affected system will be 'effectively down' while the migration is taking place.

Responsibilities of the System Owner

The system owner is responsible for maintaining the system to ensure that it operates smoothly and as expected. This responsibility is particularly important when altering or deleting existing system objects, in particular:

Tables

dropping tables: this is a potentially dangerous situation, as there may be application programs not only within the system, but also other systems using that particular table.

altering tables:

new attributes:	the system owner should be aware that some views may need to be regenerated
altered attributes:	as discussed earlier in 'dropping tables', potentially dangerous. The system owner must be aware of the impact on his/her own system, as well as other systems that share his/her data.
dropped attributes:	same as above.

Views

Views do not contain data, they are a means of changing the appearance of existing data. Caution should be applied when dropping or altering views, for the abovementioned reasons.

Indexes

Indexes do not alter the functionality of the system, however response time may be severely degraded if a key index is dropped.

Again, it cannot be overstressed that it is the system owner's responsibility to not only keep track of all the changes to the system and ensuring that all the changes are synchronised (i.e. if a table definition is changed, then all affected entities such as views and indexes will also have to be included in the same migration), but also to be aware of the impact the changes will have on his/her system, and on other systems which may share his/her data.

REQUIREMENTS FOR MIGRATING OBJECTS FROM TEST TO PRODUCTION

Before a migration into Production can proceed, certain system information as described below is required. Many of these requirements should be part of the usual system documentation.

New Systems

- brief summary of the system's purpose
- database schema (details of the whole system)
- Entity Relationship (ER) diagram showing both primary and foreign keys and the relationship between the entities
- sizing estimates, including:
    - the average and the maximum number of rows expected in each table
    - the expected growth rate of each table
    - space required for each table and its respective indexes (based on the average number of rows expected).
    - the amount of space that is reserved in each block for future updates to a particular table, expressed as a percentage of the allocated space (PCTFREE)

  Blank forms for sizing estimates are included in Appendices C and D

Existing Systems

The SQL code or statements used to create or alter tables, views or indexes must be provided. In the case of table changes, extra information is required depending on the type of change:

New tables

- sizing estimates, as in 'New Systems' above
- updated Entity Relationship (ER) diagram

Existing tables

- if there is significant increase in the size of the table, new sizing details are to be provided

The system owner must be aware of the points discussed in 'Responsibilities of the System Owner' above.

STEPS IN THE MIGRATION

Currently there is no software package available to automate the process of migrating objects from the Test environment into the Production environment. Should a suitable package become available, it will become incorporated into the migration path to automate the migration as far as possible.

A typical migration into the Production environment will have the following steps:

- A request from the system owner is forwarded to the DBA. The request should clearly specify what objects are to be migrated, and any conditions or peculiarities which relate to that request (i.e. 'desired date/time for the migration', 'urgent' etc). The information specified above must also be provided.

  At this stage there are no forms available for requesting migration from the Test into the Production environment, as the likely types of requests are unknown (however, after a period of time, it may be possible to design a form suitable for the majority of requests).

- The DBA will migrate the specified objects into the Production environment, at a time arranged between the DBA and the system owner (the system will be 'down' when the migration takes place).

- The DBA will notify the system owner when the migration is completed.

It is anticipated that a normal, non-urgent migration will be generally completed within one day of receiving the request.

AUDITABILITY

It is envisaged that all the SQL code or statements used in the migration be kept in a read-only file, grouped by system id, for audit related purposes.

KEEPING TRACK OF CHANGES WITHIN A SYSTEM

The concept of several people working on different parts of the system can easily get out of control, unless successful project management is introduced to monitor, and above all, control the developments that are taking place. This applies particularly to the way developments/changes are migrated into the Production environment.

The system owner must be fully cognisant with what is happening to his/her system, and it is his/her responsibility to ensure that changes to the system do not get out of control.

THE ROLE OF THE DATABASE ADMINISTRATOR OR PROGRAM LIBRARIAN

The role of the Database Administrator (DBA) or the Program Librarian (a function which may be set up to specifically administer the migration between the two environments) is to ensure that a controlled and auditable migration takes place. All migrations to the Production environment are instigated at the system owner's request. Upon receipt of an authorised request, the DBA will ensure that the specified objects are transferred thereby ensuring that the object definitions are the same in both the Test and the Production environments.

WHAT ABOUT THE ENVIRONMENT OUTSIDE ORACLE ?

This document has described dual environments, and the mechanism to promote changes from Test to Production. This, however, only addresses issues which are internal to the Oracle environments themselves, and not the other major problem, namely how does one manage programs that access Oracle, such as SQL*Forms, SQL*Plus, SQR etc, which all reside outside Oracle, in the form of files within directories.

These programs are totally under the control of the Operating System (OS) userids, and are outside the scope of Database Administration. It is possible however, to take advantage of the change control management in the dual Oracle environments, and apply the same principles in managing the programs.

It is highly recommended that the notion of system ownership be adopted for all the programs that comprise the system. The same concept of dual environments in Oracle as described above, can be applied to the programs as well.

Up to now we have used the terms 'system userid' and 'general userid' as they are defined within Oracle. Let us now broaden the definition of these userids to include the operating system (OS) environment. The OS system userid can again be used to own all programs that make up the system. Likewise, the OS general userid can be used for both running the system, and enhancing/developing the system.

The directory structure that could be used is as follows: The OS system userid is the home directory, under which two subdirectories exist, lets call these Test and Production (See Appendix A).

Production directory

This directory is secured, and the only access the general userids have to this userid is read and execute. No other access should be allowed. Within this directory are numerous subdirectories, which hold different types of programs, such as SQL*Forms(both executable <u>and</u> source), SQR etc, which have the same access as the Production directory.

Test directory

This directory contains the same subdirectories as the Production directory, except the security is lower than the Production directory. There are several ways to set up this directory in terms of usage to suit user requirements. Two possible ways might be:

- have all the maintenance/development staff working in their own personal directories, and then when they are finished, copy the programs to the OS system Test directory for testing either by themselves, or by the OS system owner, depending on the security of the Test directory chosen (this way is probably the preferred method)
- or have all the maintenance/development staff work directly in the OS system Test directory (much more complex to manage)

The OS system owner should again do the testing of the programs in the test directory, and when he/she is happy with the program(s), he/she copies them to the production directory, thereby having a secure copy of the program(s).

Whichever way users decide to set up their directory structure, they should all work in the following way: Test programs residing in the Test directory should only execute against the Oracle Test environment, and the programs in the Production directory should run against the Production Oracle environment.

It may be a good idea to set up an archive of all the programs that are promoted to the system Production directory.

There are several commercially available program library control packages, which will do all of the functions described here, as well as keeping track of changes, version releases of the same program, consolidation of different changes to the same program by different people, flexible reporting, flexible security, etc, etc. It is highly recommended that each area should look seriously at such packages.

SUMMARY

This document describes and formally establishes the concept of the dual Test and Production Oracle environments in AGSO. The establishment of these two environments will serve to isolate production work from that of development/maintenance work, thereby increasing the stability and integrity of the programs and the actual data in the Production environment.

The functions and procedures of change control management between the dual Oracle environments have been described and formalised. The change control management will make the operation of AGSO Oracle database environment more secure and auditable. These management procedures will apply to all systems, existing and new, on the AViiON database server.

The requirements for promoting/migrating new systems or changes to existing systems into the Oracle Production environment are established, as well as the responsibilities of the Database Administrator (DBA) and of the system owners.

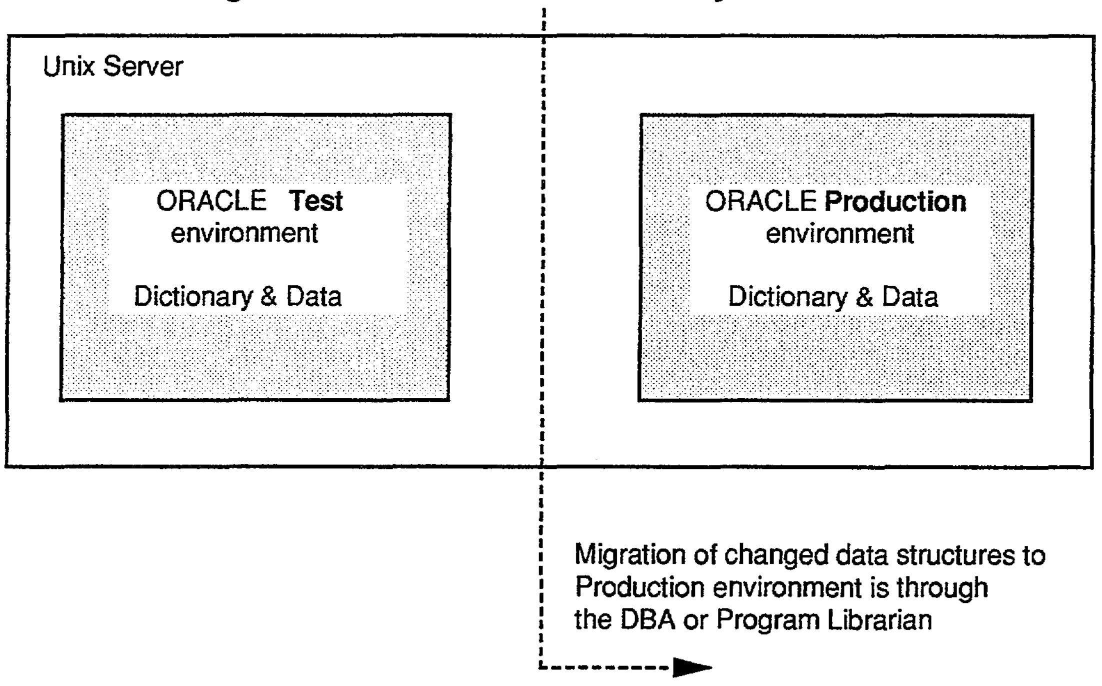
Glossary of Terms

ALTER	Change the structure of a table
ATTRIBUTE	Generally implies a column or field within a table or view.
CONNECT	Privilege granted by the DBA, which gives a user the authority to log on to Oracle. This allows the user to: - access Oracle - access other user's data (such as SELECT from tables and views), providing access has been granted - manipulate the data in other user's tables, (such as INSERT, MODIFY, etc), providing the corresponding access has been granted - create views and synonyms
DATA DICTIONARY	an accurate and up-to-date description of the physical database (data about the data stored on the database), maintained by the Oracle, in order for the database management system to access the data
DBA	Database Administrator
DDL	Data Definition Language, which is a category of SQL to define or delete database objects such as tables and views
DROP	Remove objects such as tables, views, synonyms, indexes etc
ENTITY	Generally implies a table
GRANT	Pass on to other database users the privileges which you have
JOIN	A relational database operation which returns the result of combining 2 or more tables or views based on common key/s
NORMALISATION	A data analysis technique which organises data in such a way as to minimise redundancies (multiple occurrences of the same data). For extra information please refer to any Data Analysis book
NORMAL FORM 1	Repeating groups are removed
NORMAL FORM 2	Attributes are removed which are dependant on only some of the identifying attributes
NORMAL FORM 3	Attributes are removed which are not directly dependent on the identifying attributes
OBJECTS	Database structures, such as tables, views, synonyms, clusters and indexes
RDBMS	Relational Database Management System
RESOURCE	Privilege granted by the DBA, which gives a user all the privileges of CONNECT plus the ability to create tables and other database objects
SCHEMA	Exact and total description of the system (i.e. all tables, views, indexes etc that comprise the system)

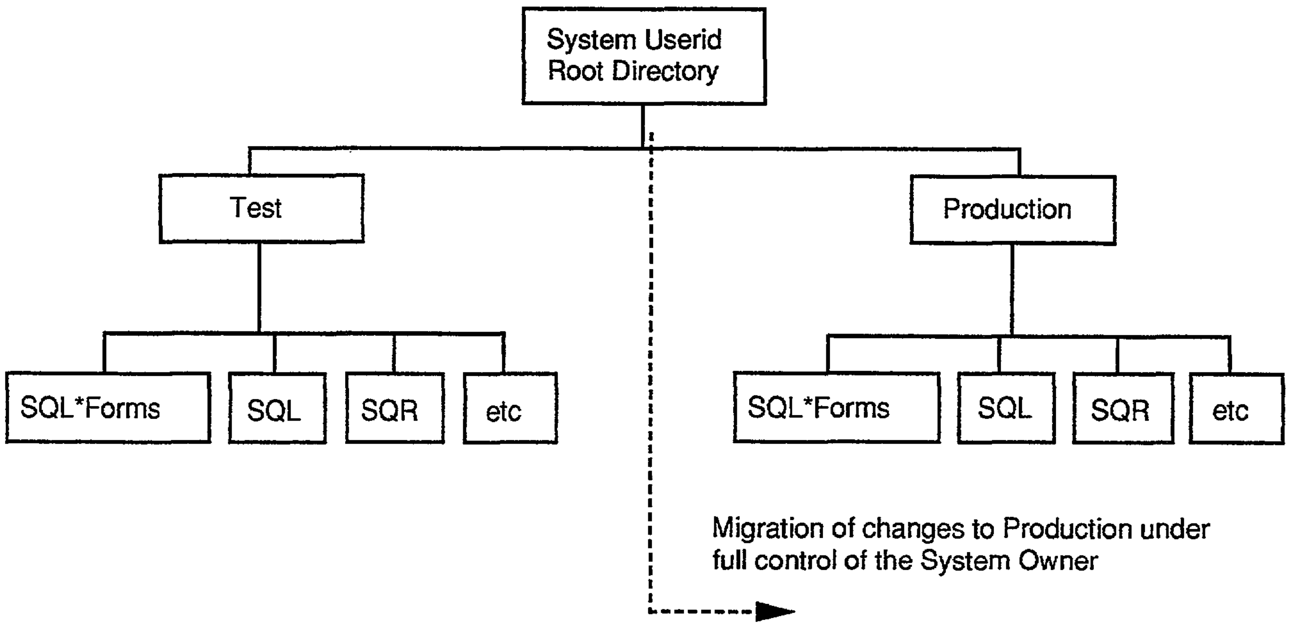
SYNONYM	Name assigned to a table (other than the actual table name), which can be used to refer to that table
SQL	Structured Query Language, used to define and manipulate data stored in the database
SQL*Forms	A full-screen forms interface, which allows users to create, modify and use full-forms for data entry or query
SQL*Plus	An interactive command-driven interface to Oracle databases, useful for ad-hoc queries and report writing, extension of SQL
SQR	Structured Query Report writer, marketed by SEQUEL Pty Ltd
TABLE	This is where the actual data is stored. A table is defined as a number of columns (fields or attributes) and rows (records, tuples or observations)
VIEW	Stored query against one or more database tables which returns results in tabular format

Appendix A

Change control management for ORACLE Database systems


Unix Server
ORACLE Test environment
Dictionary & Data
ORACLE Production environment
Dictionary & Data
Migration of changed data structures to Production environment is through the DBA or Program Librarian

Suggested directory structure for programs belonging to the System


System Userid Root Directory
Test
Production
SQL*Forms
SQL
SQR
etc
SQL*Forms
SQL
SQR
etc
Migration of changes to Production under full control of the System Owner

TRANSACTION DETAILS

SYSTEM: Date:

Transaction name:

Frequency Avg: Based on: Response:
Max: (hours/days etc) required

Table Accessed	Selection Criteria used	Access Type (S,U,I or D)	Rows Accessed	
			Average	Maximum

Type of Access: Select, Update, Insert or Delete

TABLE SIZING DETAILS

SYSTEM : Date:

Table name	Average expected number of rows	Average record row size (bytes)	Expected growth rate per year %	PCTFREE	Space required [in MEGABYTES] (based on average expected rows)

INDEX DETAILS

SYSTEM : Date:

Table name:

Index name	**Indexed on columns** (list columns in the desired sequence, first column to last column)	**Unique** (Y/N)	**PCTFREE**	**Space required** [in MEGABYTES] (based on average expected rows)

TABLE SIZING DETAILS

SYSTEM : XMAS					Date: 22/12/90
Table name	Average expected number of rows	Average record row size (bytes)	Expected growth rate per year %	PCTFREE	Space required [in MEGABYTES] (based on average expected rows)
CHILD	22 500	120	1.5%	5%	2.7
PRESENTS	500 000	60	1.5%	5%	30
HISTORY	340 000	46	1.5%	5%	15.6
PRESENTS_MADE	50 000	78	10%	10%	3.9

INDEX DETAILS

SYSTEM : XMAS Date: 22/12/90
Table name: CHILD

Index name	Indexed on columns (list columns in the desired sequence, first column to last column)	Unique (Y/N)	PCTFREE	Space required [in MEGABYTES] (based on average expected rows)
CHILD_NAME_IX	LAST_NAME	N	5%	1.5 M
	FIRST_NAME			
	DOB			
CHILD_TAX_NBR_IX	TAX_FILE_NBR	Y	5%	0.5 M

TRANSACTION DETAILS

SYSTEM: XMAS Date: 22/12/90
Transaction name: ALLOCATE_PRESENTS

Frequency Avg: Max: Based on: (hours/days etc) Response: required

Table Accessed	Selection Criteria used	Access Type (S,U,I or D)	Rows Accessed Average	Rows Accessed Maximum
CHILD	TAX_FILE_NBR	S	1	—
HISTORY	TAX_FILE_NBR	S	15	40
PRESENTS	TAX_FILE_NBR	S	20	50
SEQUENCE	SEQUENCE_TYPE	U	3	5
PRESENTS		I	3	5