USERS' GUIDE TO AGSO'S ORACLE DATABASE SYSTEM

by S L Lenz, R J Ryburn & M Kucka

Record 1993/81

AGSO

AUSTRALIAN
GEOLOGICAL SURVEY
ORGANISATION

USERS' GUIDE TO AGSO'S

ORACLE DATABASE SYSTEM

Record 1993/81

S.L. Lenz, R.J. Ryburn and M. Kucka

AUSTRALIAN GEOLOGICAL SURVEY ORGANISATION

DEPARTMENT OF PRIMARY INDUSTRIES AND ENERGY

Minister for Resources: Hon. Michael Lee, MP
Secretary: Greg Taylor

AUSTRALIAN GEOLOGICAL SURVEY ORGANISATION

Executive Director: Harvey Jacka

ISSN: 1039-0073

ISBN: 0 642 19880 2

CONTENTS

ABSTRACT

This Record gives database users in the Australian Geological Survey Organisation (AGSO) an overview and basic understanding of the corporate Oracle database system. It supplies enough information for 'average' users to begin using the system effectively. SQL*Plus underpins the Oracle relational database management system and a subset of the SQL syntax is provided that will allow users to accomplish most database tasks successfully. An appendix includes some more sophisticated routines.

Although individual databases are not discussed in depth, and the information given on the Oracle system is not exhaustive, the authors believe that the increasing numbers of AGSO staff expected to use Oracle databases without comprehensive training should benefit considerably from this Record. It is also the basis of an in-house Oracle training course. Another important function of this users' guide is to remove the need to duplicate this type of information in the guides to individual databases.

1 - INTRODUCTION

As Australia's leading earth science research agency the Australian Geological Survey Organisation plays a major role in collecting, processing and disseminating data on the Australian continent and its surroundings. Over the last 8 years AGSO's attribute data collections have been progressively transferred to the corporate relational database management system (RDBMS), Oracle. This, together with advances in networking technology, has meant that the data can now be more readily accessed and shared between different groups within AGSO. Synergy has resulted.

Information Systems Branch is aware that as more databases are transferred to Oracle, and as more users access these data sets, the need grows to train staff in the use of Oracle and its tools. Commercially available training courses are rather expensive in these times of financial constraint, and they are generally not focussed on the needs of 'normal' AGSO users. This Record is the first step in providing more relevant Oracle training to AGSO staff, to be followed by an in-house course with hands-on training beginning in the latter part of 1993.

2 - RELATIONAL DATABASES

A relational database is one in which the data can be thought of as residing in tables with horizontal rows and vertical columns. A table contains data items which logically belong together. These data items are attributes of an entity that is being represented in the database. Columns consist of named fields, generally of fixed lengths, accommodating numbers, text, dates or binary data (eg. images). The rows are records, usually identified by a unique primary key of some kind. Tables are related to one another by common data and there are no predefined linkages. Thus a row in one table may share some data values with a row in another

table - the ROCKS and K_AR tables in the NGMA Field Database both include sample numbers. Also, an item in one table can be repeated many times in another.

If one record in a table relates to a single record in another table we talk of a one-to-one relationship. In the NGMA Field Database, for example, a record in the SITES table corresponds with only one record in the OUTCROPS table. Similarly, if one record in the first table corresponds to more than one record in the second table, we are looking at a one-to-many relationship. A record in the OUTCROPS table may correspond to many rock samples in the ROCKS table. Finally, a many-to-many relationship occurs when several records in the first table correspond to more than one record in the second table. A rock sample in the ROCKS table generally contains many minerals recorded in the AGSOMINERALS table, while a single mineral in that table can occur in thousands of rock samples. (Many-to-many relationships like this are represented in the database through an intermediary table between the two entities - not shown in the diagram below.)

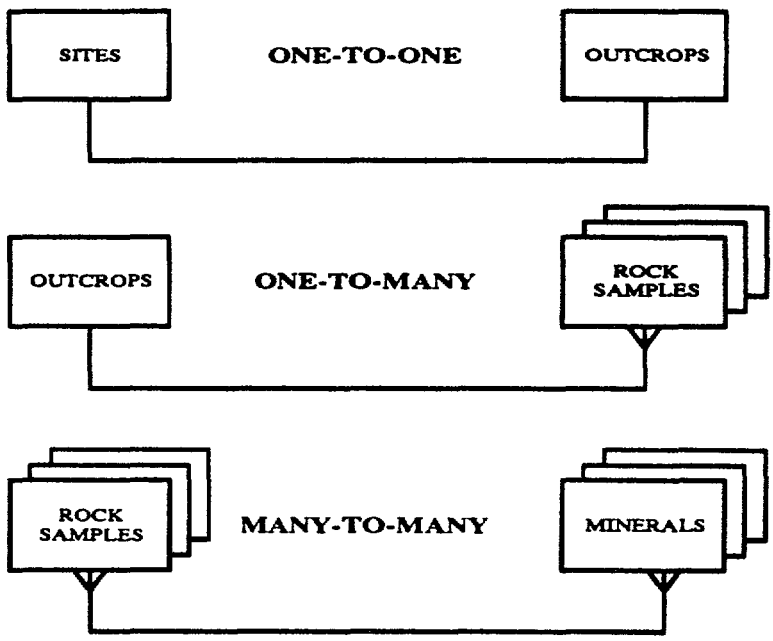


Figure 1. Some possible relationships between tables in a relational database.

The strengths of the relational model are its conceptual simplicity, flexibility and inherent data integrity. Because the links between tables are not 'hard-wired' into the structure of the database, all conceivable retrievals are theoretically possible. On the negative side, a highly normalised relational database (i.e. one with minimal duplication of data) can have so many tables that the commands required to retrieve data become unduly complicated and maintenance becomes a problem.

A relational database is the logical choice for many of AGSO'S scientific data collections due to the great flexibility and productivity gained by using a relational database manager. Most geographic information systems (GISs) allow on-line links to relational database systems, and all geographic data so stored can be plotted on a variety of maps and other projections.

3 - ORACLE TOOLS AND UTILITIES

AGSO's corporate database management system is currently Oracle RDBMS version 6 running on a DG AViiON 6240 database server. However, Oracle will soon be upgraded to version 7. A suite of tools and utilities belong to this relational database management system of which the most commonly used are briefly described in the following paragraphs.

Users access the Oracle relational database management system via **Structured Query Language (SQL)**, a language that is now the industry standard for data manipulation and retrieval in relational databases.

SQL*Plus is Oracle's extended version of SQL which allows for limited report formatting in addition to the standard ANSI SQL functions. SQL*Plus now includes **PL/SQL** which adds all the procedural capabilities found in a third-generation language (3GL) such as FORTRAN. It can, however, only be used for data manipulation from the SQL prompt.

SQL*Menu is used to set up and operate a menu interface for executing different data processing tools. It can be used with Oracle products (eg. SQL*Forms, SQL*Plus) or any other software products that run on the AViiON (eg. any operating system command can be invoked from SQL*Menu). SQL*Menu also provides an additional layer of security.

SQL*Forms is a full-screen interface tool for creating, modifying and using application forms for data entry and retrieval in an Oracle database. Data manipulation and query, as well as some of the data control statements, are invoked from within SQL*Forms without the user having to learn the SQL syntax. Records may also be updated or deleted through SQL*Forms.

SQL*Net is database communications software that enables sharing of information stored in different databases and client applications (eg. Arc/Info) across a network.

SQL*Report is Oracle's report generation program. (**SQR** is a third-party report program.)

SQL*Loader is an Oracle tool used to load data from ASCII files into Oracle tables.

The Oracle utilities **Export** and **Import** are used for moving database objects and data to and from operating systems files on systems running Oracle. They are discussed briefly in Appendix D: Additional Notes and Handy Hints.

For more detailed information on the use of these tools and utilities refer to the user guides and reference guides listed in Section 15 - Bibliography. SQL*Plus, SQL*Menu, SQL*Forms and SQL*Net are discussed in detail in Sections 8 to 11.

4 - AGSO's ORACLE SYSTEM

Dual Oracle Environment and Change Control Management

AGSO runs two separate instances of the Oracle RDBMS. One database environment used for testing and creating or making changes to databases, is called the Test Environment. The other instance is the Production Environment which contains AGSO's 'production' databases. It is more tightly controlled. Changes are migrated to the Production Environment by the Database Administrator (DBA) in Information Systems Branch (ISB) after the system owner has tested them sufficiently in the Test Environment (for more information on the dual Oracle environment and change control management see Kucka,1992).

The Test Environment is much smaller than the Production Environment. It should contain copies of all database objects such as tables, views, indexes etc., but only subsets of the data - ideally just enough to execute and test all programs which run against the database.

To use Oracle, the environment you wish to work in - Test or Production - has to be specified. Two commands are provided on the AViiON to effect this: ***setoratest*** for the Test Environment, and ***setoraprod*** for making the Production Environment accessible. These commands set up some UNIX environment variables which Oracle needs for its programs to run.

Oracle User IDs

Oracle user IDs or user names are created by the ISB Database Administrator on the users' request. Each database has a system owner and, usually, a number of other users. The system owner ID is the same as the name of the database, eg. GEODX, OZCHRON, RTMAP, GEOMAG are both names of databases and names of the owners of these systems. The system owner has the *resource* attribute, ie. all privileges for creating and using a database, including the right to *grant* other user IDs certain privileges. The other users have *connect* rights which means they can use a database; some users can be granted *select-only* access, others can be granted *insert*, *delete*, *update*, *index* and/or *grant* access, according to their needs. All these rights can be to the entire database or to parts of the database, ie. to some tables, only to certain table columns, or to particular views of the database.

In the Test Environment both the system owner ID and the other users can have resource privileges so that not only the system owner can experiment with the database, set up data entry forms, create and drop tables etc.

All users of a particular database are allocated to a group, a so-called 'role', by the ISB Database Administrator. This role usually has the same name as the database system, and role membership is used by SQL*Menu to determine which users have access to menus or menu items. In the next version of Oracle (Oracle7) these roles will also be used to control data access, thereby replacing individual grants (see Section 5).

Database Directory Structure

AGSO's Oracle databases on the AViiON are each given a 'home' directory, generally named after the database or the owner's login. Under this home directory there are subdirectories for forms, menus, SQL scripts, SQR reports, other programs, and data files. The use of standard names for these directories will help users who want to find out what objects are available for each database. The recommended directory structure and directory names for Oracle databases on the AViiON are shown in Appendix B.

5 - SECURITY AND ACCESS

User Names and Passwords

To be able to use AGSO's Oracle databases you must request ISB to provide you with personal user names (also known as 'logins') and passwords for both the AGSO UNIX network and the Oracle database environment. UNIX and Oracle user names are normally identical and are formed in lower case from the user's first initial and surname. The user name must be truncated to 8 characters if longer than this (eg. 'D. Mackenzie' becomes 'dmackenz'). If the resulting user name is not unique a number is appended - or substituted for the last letter if the user name is already 8 characters long (eg. 'dmacken1'). UNIX is case sensitive, but Oracle doesn't mind if you enter lower or/and upper-case user names and passwords. However, the safest policy is to always use lower case.

The passwords that are given to you initially should be changed straight away. To change your UNIX password just enter the UNIX command ***passwd*** after first logging into UNIX with your initial password. UNIX will then prompt you again for your old password and twice for your new password. Your new UNIX password must include at least one non-alphabetic character.

To change your Oracle password you must first log into SQL*Plus with your Oracle user name and initial password - as described in Section 8 - then enter the following command:

> ***grant connect to*** [your Oracle user name] ***identified by*** [your new password];
>
> eg. - ***grant connect to pkeating identified by mabo93;***

Don't elect to use such an obvious password, though, and don't forget the semicolon that terminates the SQL*Plus statement. Enter the ***exit*** command to exit SQL*Plus.

As required by departmental security guidelines, the UNIX system on the AViiON now insists that you change your password every month, and that the new password must differ from the old one in at least three character places. When you first log on to the AViiON at the beginning of the month you are informed that your old password has expired and that a new one must be entered. You are prompted for your old password, then your new one - just follow the prompts.

Oracle version 6 does not have the ability to age passwords, so there is currently no compulsion to change your Oracle password regularly, however, it is good practice to do so. If at any time you think the security of your current password may have become compromised you should change it immediately.

Select-Only Database Access

Oracle allows all internal AGSO users select-only access to publicly available data tables. Most databases are accessible through a menu system (see Section 9 and Appendix A). All users can retrieve (view) data in the databases including existing validation tables, but they cannot insert, update or delete data. When in SQL*Plus (see Section 8), all users may select data from public tables and views provided they prepend the owner's name to the table or view name. In the following example NGMA is the owner of the OUTCROPS table:

```
select siteid from ngma.outcrops
    where origno = 56;
```

Some table or view names are very long and/or not very meaningful. To make life easier for the SQL*Plus user, the database owner can give the tables or views additional, shorter or more meaningful names, so-called *synonyms*. *Public synonyms* exist for some publicly accessible database tables whose usage instead of the actual tablename eliminates the need to add the owner's name to the table name. To find out which synonyms exist, at the SQL> prompt type

```
select * from all_synonyms;
```

(* stands for 'all fields' - see Section 8).

Insert, Update and Delete Oracle Access

The system owner alone (or the DBA in an emergency) can grant other users the privilege to insert, update or delete records. This is done for individual users and individual database objects.

Custodians' Access Privileges

Data custodians are responsible for looking after the data in databases. Some custodians have responsibility for a whole database, others just for a particular authority or validation table, especially in large multi-user databases such as the NGMA Field Database system. In the case of smaller databases the system owner is often also the data custodian of the whole database.

Custodians are given appropriate access privileges to the data or authority tables which they administer. They may insert, update and delete all data in these tables via screen forms or from SQL*Plus. Users of validation tables who find they need additional data values have to ask the custodian to add records for them. Usually the custodian will either consult with other users of

the table or make his/her own decision as to the appropriateness of the suggested addition. The same applies to deletions and amendments. Care is required with deletions and amendments because other tables in the database may need to be updated to reflect changes (especially in the case of an authority table).

Owners' Access Rights

The database owner is the user name under which the database is created. This user name is the same as the name of the database, eg. PALEO owns the PALEO Fossil Collections Database, RTMAP the RTMAP Regolith Terrain Mapping Database, NGMA the NGMA Field Database. Usually, there is one member of the database user group who has the function of the system owner (see Appendix A). Owners have resource privileges to all tables and other objects in their database, and only they can grant other users access privileges. If you need access to a particular database you must ask the owner to grant it to you.

In the Test Environment the database owner's resource privileges also include the ability to *alter*, *create*, *drop* and *rename* objects (see Appendix C). In the Production Environment these SQL commands cannot be used.

6 - TERMINAL EMULATION

Open Systems Terminal Access

The change in AGSO to an open-systems UNIX environment has required particular attention to the equipment and software used to access corporate Oracle databases. Although the DEC vt220 terminal type has been adopted by AGSO as the standard character terminal for accessing corporate databases, most users now have IBM compatible PCs, Macintosh PCs or Sun workstations running vt220 terminal emulation software. In the near future, the 'X11' graphics terminal standard for accessing Oracle in graphics mode is likely to become important with Oracle Forms version 4.

The genuine vt220 keyboard is different from those of PCs and Sun workstations, and keyboard mappings vary in some aspects between various vt220 emulators. Oracle's SQL*Forms and SQL*Menu require many different key combinations. Oracle's default key combinations for vt220 terminals are excessively cumbersome, with the result that nearly all Oracle sites remap Forms/Menu functions to a friendlier set of keys. In AGSO we have tried to match the Forms/Menu keys as closely as possible to the much better set used in PC implementations of Oracle. However, the default vt220 setup is still available if needed.

Other terminal types can also be used. The obsolescent DG-411 terminal may be used (with only one or two minor irritations). The DG-412 terminal behaves as a 411 or can be operated in vt220 mode. The use of vt100 terminals is also possible. Sun workstations and IBM compatible PCs can be used in X11 terminal mode. For information on accessing the AViiON from PCs using X11 devices refer to Chopra (1991a,b). X11 terminal emulations are likely to become more popular when SQL*Forms Version 4.0 is installed, as a graphical user interface (GUI) with displayed images should then become a realistic proposition. Other means of

providing GUI front ends generally rely on expensive client software that entails yet another learning curve and much time spent on application development. Client software is being trialled by some groups of users in AGSO.

Oracle uses the term *device type* to indicate the terminal type. In AGSO, Oracle understands several device types that have been specifically designed to work with Oracle SQL*Forms and SQL*Menu. The available device types are:

d412-dg	native DG mode 410, 411 and 412 terminals
d412-vt	412 terminal in vt220 emulation - based on the 'pc-vt' mapping below
pc-vt	PC vt220 emulation, resembles key mapping of PC Oracle. The Racal Interlan also uses this key mapping with some minor differences (see below for more information on Racal)
sun-k4	Sun terminal type 4 mapping, again based on 'pc-vt' key mapping
vt220	standard, real vt220 device, based on 'pc-vt' key mapping
vt100	standard, real vt100 device, currently as supplied by Oracle.

When invoking SQL*Forms/Menu, Oracle automatically looks at the UNIX environment variable TERM to determine which device type is being used. However, it is possible to instruct Oracle to use a different device type than that specified by TERM.

Issuing the command ***runform30 -c <terminal device> <form name>*** to run a form, for instance, overrides the TERM variable and substitutes the name represented by <terminal device>. Note that this will only work if <terminal device> is one of the above choices. The same parameters can be used with the commands ***sqlforms30***, ***sqlmenu50*** and ***runmenu50***. It is important to specify the right terminal type, otherwise the screen will show garbage when you try to display Oracle screen forms.

PC Keyboard Mapping

Both Oracle Production and Test can use the device type 'pc-vt' to map PC keys to Oracle functions for PCs that are in vt220 emulation mode. The preferred method to make Oracle use this key mapping is to set the TERM variable to 'pc-vt' as Oracle by default uses the value of TERM to determine which device type to use:

TERM=pc-vt (using Bourne shell)
export TERM

or ***setenv TERM pc-vt*** (using C shell).

LAN Workplace for DOS - 'TNVT220'

For the majority of users of IBM compatible PCs in AGSO the currently recommended terminal emulation software is the 'TNVT220' emulator supplied with Novell's LAN Workplace for DOS package. This package provides the functionality required for users connected to a Novell Netware PC LAN to talk to remote systems and to transfer files.

Although the package also provides a Microsoft Windows vt220 emulator ('Host Presenter'), its use is not recommended as it pre-empts some keys for Windows functions and does not provide a full width screen on a standard VGA colour monitor. (However, it could be used in future if someone devotes time to working out a suitable keyboard mapping).

LAN Workplace for DOS requires a tailored file **bmr220.bin**, which in conjunction with the appropriate Bourne or C shell commands (see previous page) correctly maps the PC keyboard with Oracle. This file can be obtained from ISB. We suggest it reside in a network directory, for instance *xln/bin40*, if it exists on your PC. In this case set up a batch file called **av.bat** in your c:\ directory as follows:

tnvt220 -f c:\xln\bin40\bmr220.bin av

which you run to log on to the AViiON from outside of Windows.

Racal Interlan

It is possible to access Oracle using PCs through the Racal Interlan vt220 emulator. There are several known differences with the Racal key mapping. The keys 'Insert', 'Home', 'PageUp', 'Delete', 'End' and 'PageDown' map to different Oracle functions, as described below, and also have to be invoked with a 'Ctrl' key.

PC Key	Normal Oracle function	Racal function	Racal PC Keys
Insert	Insert/Replace	Next Block	Ctrl Insert
Home	Next Block	Insert/Replace	Ctrl Home
PageUp	Previous Record	Previous Block	Ctrl PageUp
PageDown	Next Record	Next Record	Ctrl PageDown
Delete	Delete character (backwards)	Previous Record	Ctrl Delete
End	Previous Block	Delete character (backwards)	Ctrl End

Additionally, under Racal, the 'Esc Z' key does not work, which equates to Oracle function 'Delete Line'.

VersaTerm-PRO

Macintosh users can access Oracle through VersaTerm-PRO's vt220 emulator. The software takes you through several setup screens. The TCP/IP address to host av is 192.104.43.110. In the keyboard settings screen set <RETURN> to New Line, the Delete key to <BACKSPACE>, highlight the Numeric option for the DEC vt100 keypad, and DEC vt220 keyboard as extended keyboard option. In the Extras screen under the Settings menu the following options should be set on:

Text Emulation Options:

vt220/7-bit, Auto DEC vt220 Entry, Auto Horizontal Scroll, Auto Wraparound, Add NL after CR, Return is New Line, Standard Tab Stops, Ignore vt100 Answerback.

General Options:

Multifinder "Auto Zoom", Enable Sounds, Enable Remote File Access, Enable DA's while Printing.

Graphics Emulation Options:

High Resolution PICT's.

There are also other options you can set according to your needs, such as the shape of the cursor, the format of text files created, word wrapping at a particular column, prompt character, and printer type. The available options are somewhat different in newer versions of the software. With a bit of trial and error, and consultation with other Mac users you will find the most appropriate settings.

7 - LOGGING ON AND OFF THE AViiON SERVER

AGSO's corporate database server is currently a DG AViiON 6240 multi processor computer running the UNIX (System 5, Revision 4) operating system and Oracle's relational database management system (RDBMS). Most people connected to AGSO's Ethernet-TCP/IP local area network (LAN) can now log on to the AViiON computer, but the procedure varies widely throughout AGSO according to the user's circumstances.

If you have LAN Workplace for DOS installed on your PC and you are connected to a Novell Netware PC LAN and you have set up the batch file av.bat (see Section 6), run it from outside of Windows (type ***av <ENTER>***). This will take you to the AViiON's login invitation.

Enter your UNIX user name - eg. ***swatanab <ENTER>***.
[your UNIX password] <ENTER>.

Provided your user name and password were correctly entered, the AViiON will now prompt you for your terminal type, eg.

Terminal Type is vt220 (vt100, sun, d412-dg...)
Press <Enter> to accept, - for default or enter new type :-

Press ***<ENTER>*** to accept the terminal type if it is the right one. People using a vt220 terminal emulation on an IBM-type PC will enter ***pc-vt***. If you are using a DG 400-series terminal in its native DG mode, or a PC emulation thereof, you must enter ***d412-dg*** (a short-hand way of doing this is to enter a minus sign). If you are using the vt220 emulator on a DG terminal, enter ***d412-vt***. See Section 6 for a list of available terminal types.

Your AViiON UNIX prompt is now displayed (Bourne Shell) - **av:/home1/swatanab $ _**

Some users are still connected to the older Sytek LAN and have to use the DG 400 terminals or PC emulations that they previously used to log on to the old DG MV/20000 to log on to the AViiON. Proceed as before to obtain the Sytek LAN's "#" prompt, then type:

av <RETURN>
<ENTER>

From here on follow the above instructions from UNIX login. (Logging on through the current MV9300 is not encouraged as it will be phased out soon.)

To log off the AViiON type

exit <ENTER>

or press ***<CTRL>-D*** (**^D**).

8 - SQL*PLUS

If you wish to make full use of the power of Oracle, you should learn to use SQL*Plus to make retrievals and updates. SQL*Plus is Oracle's version of SQL (Structured Query Language). Although screen forms allow you much freedom to select and update data in tables, you can use SQL*Plus for some of the more complex operations - for example, queries involving several tables. SQL is essentially a non-procedural language without loops, 'go-to' statements and subroutines (PL/SQL is a procedural extension to SQL in Oracle - see the PL/SQL User's Guide for more information). It is an interpreted interactive language. There are four different types of SQL statements:

-	data definition	eg.	***create table...***
-	data control	eg.	***grant select...***
-	data manipulation	eg.	***insert into...***
-	data query	eg.	***select * from ...***

The data definition and data control statements are used when setting up, or changing the structure of, a database, and for controlling data access. These are mainly tasks performed by the system owner. Data manipulation and data query statements are used for inserting, updating and deleting records as well as for data retrieval.

Entering and Exiting SQL*Plus

You may enter SQL*Plus from most databases' menu system. Alternatively, from the UNIX prompt, type ***setoraprod <ENTER>*** to select the Oracle Production Environment (or ***setoratest*** to get into the Test Environment), followed by ***sqlplus <ENTER>*** to log into SQL*Plus. Supply your Oracle user name and password when asked. Once in SQL*Plus the SQL> prompt is displayed (SQL-Prod> or SQL-Test>).

To exit from SQL*Plus, just type ***exit <ENTER>***.

Most users require only a small subset of the total SQL vocabulary. In this section we will look at the most commonly used commands: ***select***, ***update***, ***insert***, ***delete***, ***commit*** and ***rollback***.

Basic queries

The *select* command is used to retrieve data from Oracle databases. *Note that the semicolon is needed to terminate all SQL statements.* The general statement for retrieving data is:

SELECT columnname FROM owner.tablename
WHERE condition; (optional)

A statement like

```
select qdate,origin_time from quakes.details;
```

retrieves the values for fields *qdate* and *origin_time* for **all** records in table DETAILS of the QUAKES database (all 450 000 of them!). A *where* clause restricts the selection to user-defined conditions, eg. to retrieve date, origin time, latitude, longitude and maximum magnitude of all events with magnitudes greater than 7 type:

```
select qdate, origin_time, dlat, dlong, m_max
    from quakes.details
        where m_max > 7;
```

Combinations of conditions are also possible:

```
select qdate, origin_time, dlat, dlong, m_max
    from quakes.details
        where qdate between 19800101 and 19801231
        and m_max > 7;
```

Selecting all Fields

A shorthand way to select all columns from a table is to use an * (asterisk) in place of the column names. For instance, when using the QUAKES database to retrieve all the records from the SOURCES table, at the SQL> prompt type:

```
select * from quakes.sources;
```

Another common requirement is to select all **distinct** values of a particular field. For example:

```
select distinct hmapno from ngma.sites
    where geoprovno = 54          - [Mount Isa Inlier]
    and origno = 50;              - [Wybom. L.A.I.]
```

This particular selection will retrieve all 1:100 000 sheet numbers in the Mount Isa Inlier that have sites for which L. Wybom is the Originator.

Table Joins

It is possible to retrieve data items (columns) from more than one table if they are related to one another. Usually, two tables with related data have values in at least one column in common. For example, here is a select statement that retrieves the bibliographic details of articles in GEODX which refer to the ACT. Table joins like this are a common requirement:

```
select code, title, publication, year
    from geodx.bibliog, geodx.article_state
    where article_state.nm_state = 'ACT'
    and article_state.cd_article = bibliog.code
    order by code;
```

The last equation in this statement shows which attributes in the two tables are to be used for *joining* them. If this condition is left out a so-called 'Cartesian join' is done which combines **every** record in one table with **every** record in the other table. This has the potential of creating a huge output listing - probably not what the user was looking for. Care should therefore be taken to ensure that the conditions imposed on the retrieval are tight enough to retrieve the required information. The **order by** clause ensures that the result is sorted in ascending order of the field *code*.

Sometimes, a nested select using '**in**' may be used:

```
select count(*) from geodx.stratname
    where stratigraphic_name in (
        select reserved_name
        from geodx.reserved_names);
```

This last select statement returns the number of stratigraphic names in GEODX that were originally input into GEODX as reserved names.

Outer Joins

To join two or more tables and also return those rows from one table which have no direct match in the other table, a so-called outer join (indicated by a + sign in round brackets) is used:

```
SELECT column1, column2, ...
    FROM table1, table2
        WHERE table1.column = table2.column (+);
```

Using MINUS and UNION

When setting up a select statement to retrieve data from one table based on a join involving data which is NOT in the second table it is better to use the MINUS operator between the sets of retrieved items from both tables. For example, instead of:

```
select id_stratname
    from geodx.stratname
where not id_stratname in
    (select id_stratname
        from geodx.stratname_article);
```

it is better to use:

```
select id_stratname
    from geodx.stratname
minus
select id_stratname
    from geodx.stratname_article;
```

The difference in performance between the two methods can be dramatic.

A UNION can be used to combine similar kinds of data retrievals, eg. data from the Stratigraphic Index database GEODX can be combined with data from the corporate references database GEOREF with the following command:

```
select code, to_char(year), title, publication
    from geodx.bibliog;
union
select refid, year, title, source
    from georef.references;
```

Sending Retrieved Data to a File and Printing it

To select data from OZCHRON's K-Ar table -

```
select sampno, mineral, age_ma, std_dev
    from ozchron.k_ar
        where origno = 37
        and k_wtpct > 0.5;
```

and *print* the output, type ***spool filename***, eg. ***spool temp <ENTER>***, before entering the select statement. When the listing has finished, type ***spool out <ENTER>*** and the listing will be printed in the printer room on the ground floor of AGSO, ready for collection from the pigeonholes. Note that file 'temp.lst' (*.lst* is the suffix automatically appended to the filename by SQL*Plus, if you don't supply one yourself) remains in your current directory, and can be edited or transmitted over the network to a PC. Use ***spool off <ENTER>*** to end spooling without printing.

If a large listing (say, larger than 1 Mbyte) is about to be created, use the AViiON directory '/scratch' to contain the spool file -

spool /scratch/temp <ENTER>.

This directory is for files too big to fit into your personal directory. Any user can create files in /scratch and anyone else can read them. The file cannot be kept in /scratch forever, though, as this directory is only meant for temporary storage. It will eventually be wiped, so make sure you print it out, manipulate the data or transfer it to a different storage device (your PC for instance), as soon as possible.

Formatting Output

SQL*Plus commands can be used to format the output of a select statement. To find out what the current settings are, type ***show all.***

The default formatting parameters set up in SQL*Plus on the AViiON - for instance the length of pages and lines equate to a normal screen display (14 lines and 80 characters), and column headers are always printed at the top of each page of output - are often not appropriate for your output. To change formatting parameters, the ***set*** command is used, for instance

set linesize 132 and
set pagesize 64

would format the output of a select statement for a wide line printer. By setting pagesize to 0 all headings, page breaks, titles, the initial blank line and other formatting information can be suppressed in output.

Some formatting parameters are set either ***on*** or ***off***, this applies to ***heading*** (column headers), ***echo***, ***pause***, ***feedback*** and ***termout***. To suppress column headings in a report, type ***set heading off***; to get the listing of command lines in a command file during execution, type ***set echo on***; to make SQL*Plus pause at the beginning of each page of output on the screen, type ***set pause on*** (you must press ***<RETURN>*** to make the terminal scroll again after each page); to suppress the message **n rows selected**, type ***set feedback off***; and to suppress the display of output from a command file on your screen, type ***set termout off*** (these last two only make sense if you are spooling the output to a file).

When selecting all columns from a table with many columns it might be necessary to change Oracle's default buffer size as it might be too small to handle the large number of columns. For instance, change the arraysize:

```
set arraysize 10
select * from ozchron.u_pb where origno = 15;
```

For more help in formatting contact the DBA.

Updating the Database

To make changes throughout a table, the SQL *update* statement is used. For example -

```
update ngma.rocks
    set lithname = 'metabasalt'
    where lithname = 'metabasite';
```

The % symbol as substitute for any number of characters is very useful when updating character fields -

```
update ngma.samples
    set stratunit = 'Timbuktu Formation'
        where stratgroup like 'Timbuktu%'
        and stratunit is not null;
```

This will update "Timbuktu Group", "Timbuktu Sandstone", and any other variants that might exist. Note that the **like** operator, rather than =, is required whenever the % symbol is used in this way.

When you are satisfied that the update has occurred according to plan (ie. the message '25000 records updated' does not appear when you were expecting about 2!), you must *commit* the changes by typing

```
commit;
```

The reason for this is that anyone else using the table (eg. from a form) will be suspended from any updates or inserts on the same records until you commit your update (however, other users can still *select* the records). Updates and/or deletes from SQL*Plus place an exclusive lock on rows until a commit is executed, or until you log out of Oracle. You cannot roll back after a commit.

When attempting an update, be sure to include the **where** clause, otherwise **all** records in the database will be updated! However, all is not lost - you can do a *rollback*. Just type

rollback;

and the last SQL command (in actual fact, all commands back to your last explicit commit command or implicit commit in logging out of SQL*Plus) will be undone! That is, all records just updated will be 'backdated'. Note that a rollback can be done only if you have not logged off in the meantime as an implicit commit occurs on exit from SQL*Plus.

Inserting New Records

Users with insert privileges to a particular database's tables can *insert* records from within SQL*Plus. The general insert statement is:

INSERT INTO owner.tablename (column1, column2,...)
VALUES (value1, value2,...);

For instance, to insert a new source of information on nuclear explosions into the XSOURCES table of the Nuclear Explosions Database NUCEXP, the staff at the Australian Seismological Centre can type:

insert into nucexp.xsources (source, description)
values ('Ans', 'A new source');

At this stage typing ***commit;*** writes the new record to the database, ***rollback;*** makes the change undone.

Deleting Records

To be able to *delete* records in a particular database a user must have delete privileges, either to all tables, to particular tables or to certain records. Care must be exercised when deleting records so as not to cause referential problems by deleting information without which information in other tables becomes meaningless (in version 7 referential integrity will be looked after by the database management system). The general delete statement is:

```
DELETE FROM tablename
    WHERE condition;
```

The **where** clause is very important - it determines how many records in the table are going to be deleted. If it is left out **all** records in the table will be deleted. It is good practice to do a *count* of the records which satisfy your delete condition/s first to find out how many records are about to be wiped before actually specifying the delete statement.

For example, if all reserved names entered into the RESERVED_NAMES table of the GEODX Stratigraphic Names Database on a certain day, say the 25th of August 1993, were incorrectly entered, after finding out how many records were entered on that day by doing the following *select* :

```
select count(*) from geodx.reserved_names
    where dt_reserved = '25-AUG-93';
```

(it is often useful just to select all the records that you want to delete first as a last visual check) they could be deleted from the database with the following statement:

```
delete from geodx.reserved_names
    where dt_reserved = '25-AUG-93';
```

SQL*Plus tells you how many records it has flagged to delete. If the number coincides with your expectations, type ***commit;*** to make the deletions permanent.

Alternatively, type ***rollback;*** if you realise you are about to wipe the wrong number of records.

Finding out About the Structure of the Database

To be able to use SQL properly you need to know the names of the relevant tables and the names of the columns in them. Table names and their fields are to be found in the database schema which is usually part of the database documentation. Alternatively, to find out what tables comprise a particular database which is accessible to you, eg. the publicly accessible GEODX database, type

```
select * from all_catalog
        where owner = 'GEODX';
```

This *select* statement without the **where** clause gives you a list of all tables and views to which you have at least select (= read) access.

Use the ***describe tablename*** command in SQL*Plus for a listing of all column names, their datatype and size, eg.

```
describe rtmap.landf
```

for the landforms table in the RTMAP database. (Actually, SQL*Plus can understand the abbreviated command ***desc tablename***.)

> ***Hint***: *If at any time you are in the middle of a SQL statement and you realise that you have, for example, forgotten what the column names are, you can temporarily jump out of the first statement and issue another command by prefixing it with # (eg.* ***# describe tablename****). After executing the second command, SQL*Plus takes you back to where you interrupted the first one.*

9 - SQL*MENU

Access to most AGSO databases is through menus using Oracle's SQL*Menu. The menu system provides access to data entry and retrieval screen forms associated with the databases, reporting programs, SQL*Plus and operating system commands. Most ad-hoc queries, inserts and updates are done via forms, although you should also know that batch retrievals and updates are often more effective if done interactively using SQL*Plus (see Section 8).

The following describes the steps required to run the NGMA Database Menu:

After logging on to the AViiON server and specifying your terminal type, enter the command ***ngma*** to log into Oracle and the NGMA Database Menu. This brings up the following SQL*Menu login screen:

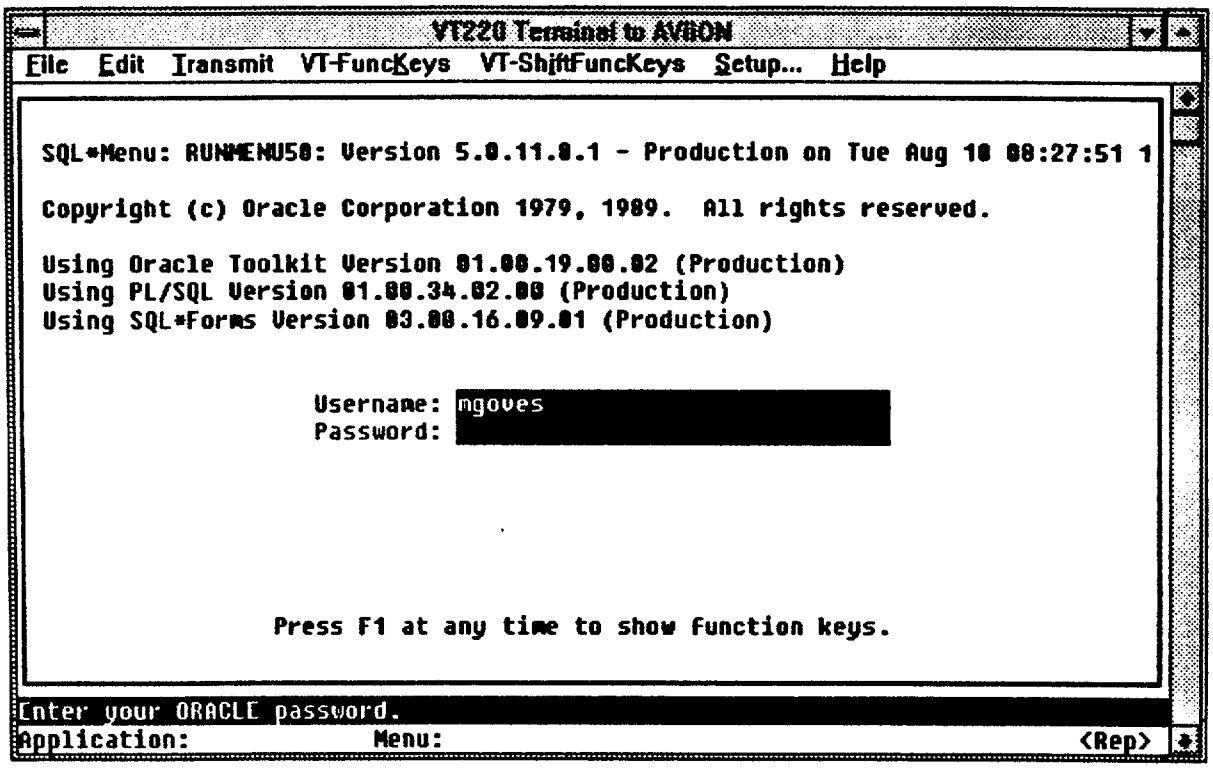


Figure 2. SQL*Menu login screen.

The key required to display your function keys in SQL*Menu and SQL*Forms is indicated at the bottom of the screen. Remember this key, as it allows you to find out how to navigate within both SQL*Menu and SQL*Forms in the absence of a keyboard overlay.

The screen invoked by this key is context sensitive but looks similar to this:

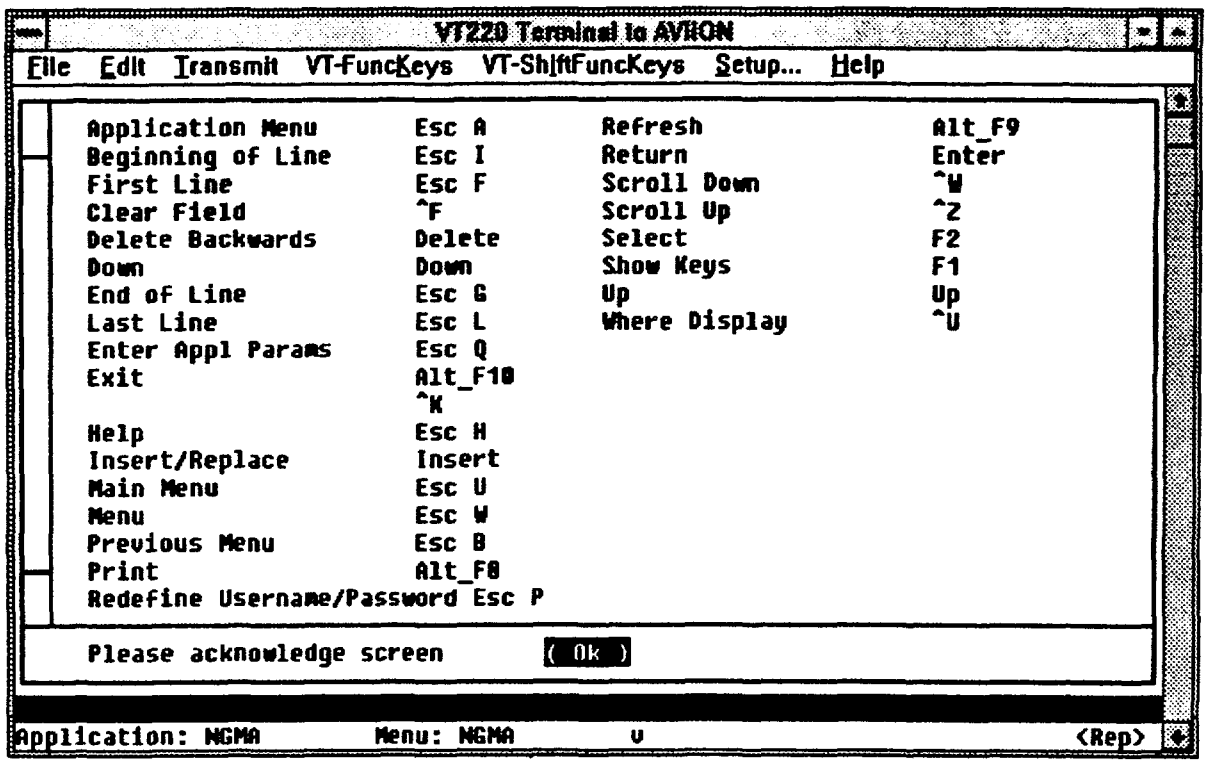


Figure 3. SQL*Menu's key mapping screen.

The key mapping screens for SQL*Menu and SQL*Forms are context sensitive - they only list the keys which are valid in the current environment. In some environments not all the available keys can be displayed on one screen - you may have to use the down and up arrows (or ^N and ^P depending on your terminal) to scroll through the whole list.

Keyboard templates containing the available functions for both PCs and DG terminals have been designed and are available from the ISB Database Administrator.

Press ***<ENTER>*** to return to the SQL*Menu login screen.

To complete logging in, supply your Oracle user name and password as requested by the SQL*Menu login screen. If successful, the opening menu is displayed:

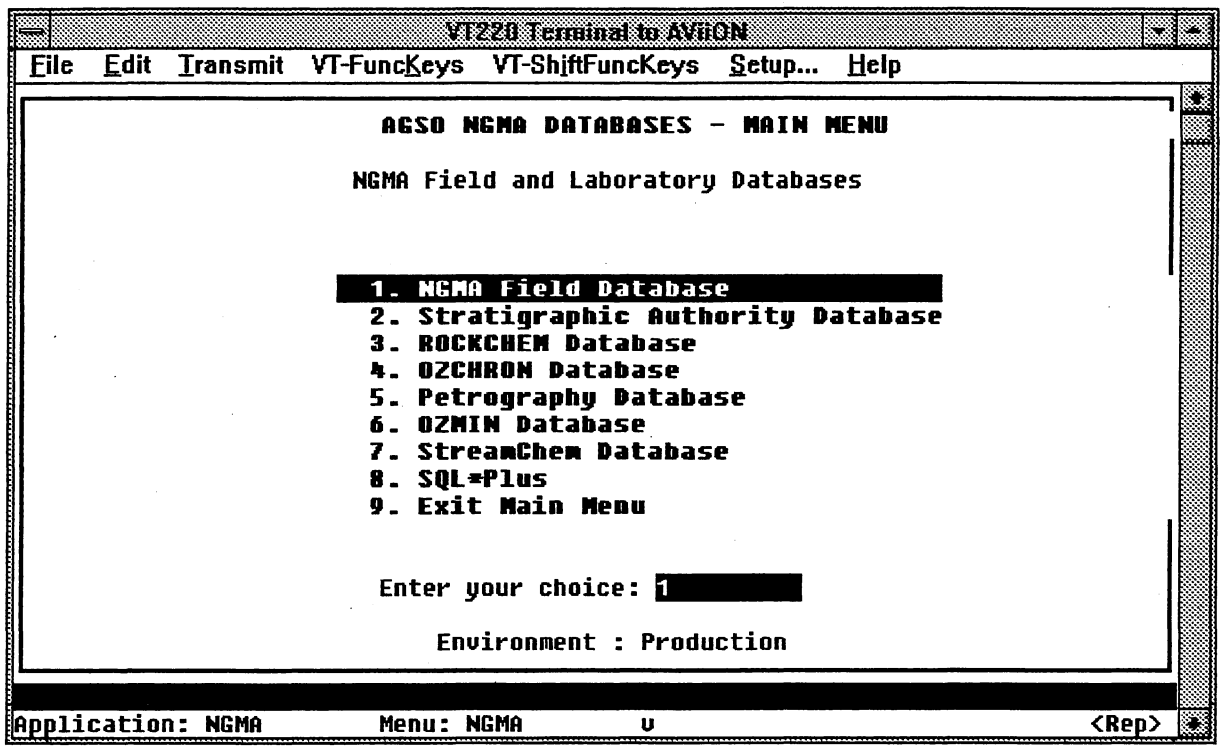


Figure 4. NGMA System, Main Menu.

You can either enter the number corresponding to the menu item required or move the highlighted bar down and up with the **<DOWN>** and **<UP>** function keys. Either way, press ***<ENTER>*** to register your choice. Note that most menus have a menu item which allows you to engage SQL*Plus directly without having to supply your Oracle user name and password again.

To exit from a menu enter the number corresponding to 'Exit Menu', 'Exit Submenu', 'Return to operating system', 'Return to main menu' or similar. A somewhat annoying aspect of SQL*Menu in 'Full-Screen Mode' (as opposed to 'Pull-Down Mode') is that the **<CANCEL/EXIT>** key drops you straight back to the UNIX prompt and cannot be used to back out of submenus to menus higher in the tree.

The **<HELP>** key is useful if you want to find out what the various menu items are supposed to achieve. Place the cursor on the menu item and press ***<HELP>***.

Some databases have only one menu whose items allow the user to access it in its entirety. The more complicated databases, in contrast, often have an elaborate system of submenus branching off the main menu, as for instance the NGMA Field Database system or the PALEO Fossil Collections Database. See Appendix A for a list of publicly available AGSO databases and the commands to access them.

10 - SQL*FORMS

General Comments

Most users enter a form through a menu. Before going into the menu you are requested to supply your Oracle user name and password (see Section 9). The same screen tells you which key or key combination gives you access to the key mapping screen - **remember it!** The query/entry forms are made up of **blocks**. Each block corresponds to a different table (= base table) in the database.

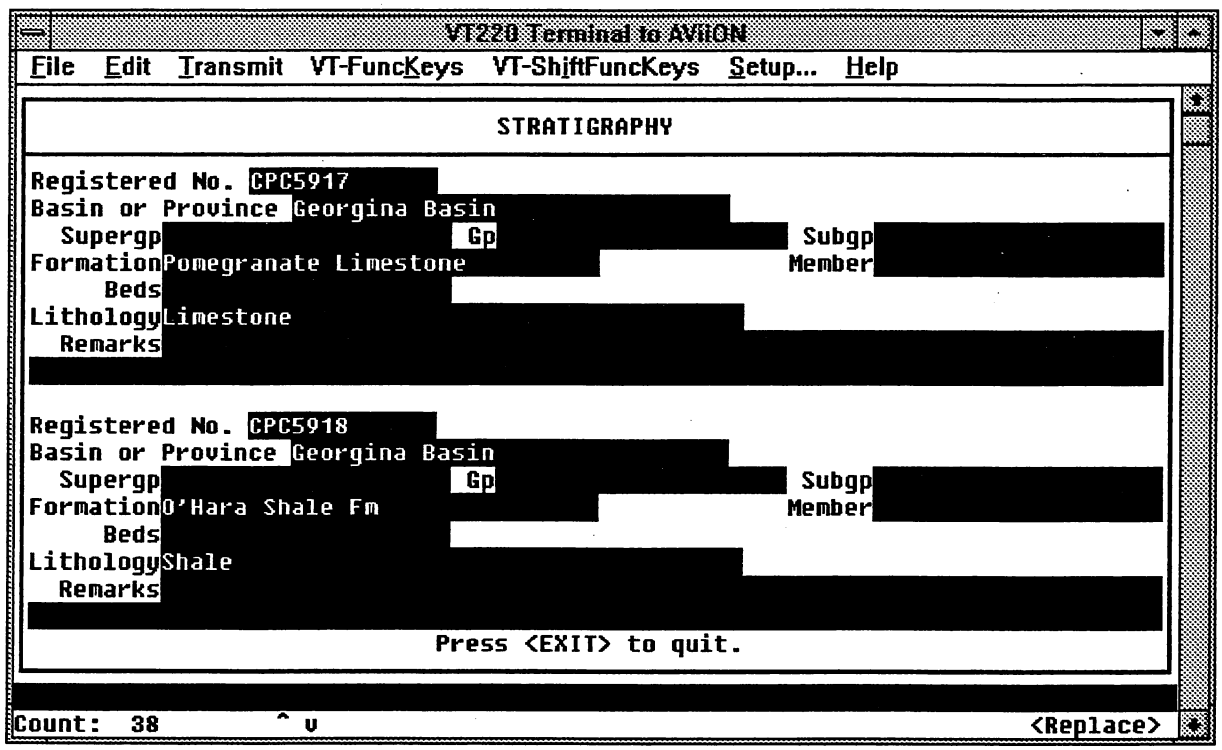


Figure 5. A simple data entry form displaying two records.

The **cursor** moves within the screen in a set sequence, generally from left to right and from top to bottom. Watch the **message line** at the bottom of the screen closely as you move through the fields using the **<NEXT FIELD>** key to go forwards and the **<PREVIOUS FIELD>** key to go backwards. The message line displays help messages for data entry and error messages should something go wrong. Should you encounter an **Oracle error** while using a form it is reported in the message line at the bottom of the screen. Use the **<DISPLAY ERROR>** key to try to find out what the error is. In some cases you will find the explanation uninformative, but '**duplicate value in index**' means that an attempt has been made to enter a record with the same primary key - eg., the same combination of Originator Number and Site ID in the NGMA Sites form. This error will occur if you attempt to commit any duplicate value in a field or fields covered by a unique index. Use key **<CANCEL/EXIT>** to return to the form after inspecting the error message. Do not hesitate to contact the Database Administrator if you run into an intractable error.

Blocks can either display only one record per screen, or they can display several records at a time. Some fields are **mandatory** which means a valid value must be entered before the cursor can move out of the field. Look at the help message or the database schema which is usually part of the system documentation if you are uncertain about the data type for a particular field.

Some forms have two or more 'blocks', corresponding to two or more underlying tables. Use the **<NEXT BLOCK>** key to move the cursor to the beginning of the next block; **<PREVIOUS BLOCK>** to move back to the previous block. Some forms display more than one block per screen. If the blocks are related to one another, for instance, in a one-to-many (master-detail) relationship, they are *coordinated.* That means that the retrieval of data in the first ('master') block will retrieve all data related to the current master record in subsequent blocks ('details'). (In forms created under previous versions of SQL*Forms there may be no block coordination.) Data entry into coordinated blocks must proceed from master to details as the field on which the blocks are joined is usually invisible in details blocks (it gets automatically copied down from the master block). For this reason the corresponding master data has to be displayed in the master block when new records are entered into a details block.

If **<NEXT BLOCK>** is used to place the cursor in the detail block, this block may be queried independently, as if it were a separate form. If there is data in the master block, then only detail records related to this master can be retrieved and updated. However, if the master block is empty, the detail block can be used to query all the data in the corresponding table.

The following 'Rocks & Structures' Form from the NGMA Field Database has 3 blocks corresponding to the ROCKS, LITHDATA and STRUCTURES tables. The ROCKS block is the 'master block', while the other two blocks are 'detail blocks' :

Figure 6. The Rocks and Structures form, NGMA Field Database system.

Should you realise after leaving a field that you have made a mistake while entering data, you can always take the cursor back by pressing the **<PREVIOUS FIELD>** key for moving within a block or the **<PREVIOUS BLOCK>** key to take you back through blocks. Correct the mistake by typing over it. The *edit* facility in SQL*Forms version 3 with features like word-wrap is useful when entering or editing data in long fields. Press the **<EDIT>** key to call up the editor.

It is generally recommended that users save the added/updated record/s in each block to the database (*commit* the data) before moving to the next block or exiting the form. The message **'n records posted and committed'** indicates that changes have been committed to the database. You may exit any screen form by pressing the **<CANCEL/EXIT>** key.

Querying the Database

Querying the database means selecting or retrieving data from the database. In an Oracle form, data from a database may be displayed by *entering* and *executing* a query. Various conditions may be included to limit the records retrieved and place them in a specified order.

Access the required form by entering the appropriate number on the menu. To initiate a query of the underlying table press the **<ENTER QUERY>** key. (Some forms are already in *query mode* when they are first displayed - in these cases the **<ENTER QUERY>** key does not have to be pressed.) The message '**ENTER QUERY**' appears at the bottom of the screen.

Press **<EXECUTE QUERY>** with an empty screen to retrieve all records from the database you have access to. To retrieve only the records that satisfy certain conditions, enter the appropriate query data into the relevant field(s) using the **<NEXT FIELD>** and **<PREVIOUS FIELD>** keys to move around the form from field to field. For example, enter the required site ID into the Site ID field in the Sites form of the NGMA database to retrieve a particular site.

Alternatively, in text fields (fields of character datatype) a **%** symbol may be used as a wildcard character which substitutes for any sequence of characters. For example, if you enter ***Coopers%*** you will get 'Coopers Creek' and 'Coopers Crossing'; ***%Smith%*** will get you 'John Smith', as well as 'Charlie Smithers and Son'. A consequence of this convention is that if a **%** symbol happens to occur in a character value you wish to retrieve, double up the **%** in the entered query value. In other words, enter ***100%%*** to find '100%'. A disadvantage with using **%** in front of the entered value is that it negates any index on the field and can drastically slow a query. The wildcard character _ substitutes for one character at a time.

Now *execute* the query by pressing the **<EXECUTE QUERY>** key. After a variable but usually slight delay, during which the word **'Working...'** is displayed in the status line at the bottom of the form, the data appears in the form.

In order to observe more than one record retrieved, use the **<NEXT RECORD>** key to step through the data, record by record - the **<PREVIOUS RECORD>** key to go backwards. In forms that display many records at once you may step forwards to the **next set of records** (ie., the next screen-full) with the **<NEXT SET>** key. Unfortunately no key exists for 'previous set of records'.

It is also possible to enter < and > symbols (less than & greater than) before a number entered as a query in a field of number datatype. However, this will not work with fields of character datatype - even if they contain only numeric characters and are actually called 'number'. See the database schema for the particular database you are looking at or, alternatively, use the ***describe tablename*** command in SQL*Plus for information on data types (see Section 8).

More complex queries can be done by entering a substitution variable, eg. ***&x***, ***&y***, etc. into one or more fields. When **<EXECUTE QUERY>** is pressed the form displays a pop-up window in the middle of the screen that allows you to enter a SQL *where* or *order by* clause. For example, if ***&x*** is entered in the Site ID field of the OZCHRON Rb-Sr form, the following *where* clause will extract all Rb-Sr records from the Marraba 1:100 000 sheet in the Mount Isa Inlier:

&x in (select siteid from ngma.sites where hmapno = 6956) - Marraba Sheet

This facility can also be used to find records with fields where ***not &x is null*** or ***&x is null*** - ie., where there is, or is not, any data.

There are a few pitfalls when querying a form. If a name like 'O'Shannassay' must be entered into a form field, the apostrophe must be doubled up like this - ***O''Shannassay*** (these are two single quotes, not one double quote). Beware of using the wrong case - 'basalt' will not get 'BASALT'. However, it is possible to use the SQL UPPER or LOWER functions - eg.,

where lower(fieldname) = 'basalt'.

Updating the Database

Forms may be used to *update* individual records in a table. Where updates are required that apply equally to many records, it is usually quicker and easier to use SQL*Plus to 'bulk-update' tables in the database (see Section 8).

To update the data retrieved in a form as the result of a query, just change the values that appear on the form using the editing keys (**<LEFT ARROW>**, **<RIGHT ARROW>**, **<BACKSPACE>** or **<INSERT/REPLACE>**) and by typing over the existing values. For long fields press the **<EDIT>** key to call up the pop-up editor. If many records have been retrieved (say, by originator number in some of the NGMA forms) you may step through them with the **<NEXT RECORD>** or **<PREVIOUS RECORD>** keys to make amendments.

Understand that no changes occur in the database itself until the changes have been committed. This may be done with the **<COMMIT>** key or on exit from the form. On exiting the form with the **<CANCEL/EXIT>** key you are asked if you want to commit the changes you have just made. Answer yes if you are really sure.

The corollary of this is that you should answer no if you are unsure as to what changes have happened. Better to have to re-enter data, rather than inadvertently corrupt the database.

Committing changes to the database is exactly like saving a document during an editing session with a word processor. The changes are made on disk only when they are committed ('saved'). In general it is better to commit changes frequently, rather than at the end of a long spell of editing. Some errors only become apparent on committing changes. Also, the system can go down occasionally.

The delays that were sometimes experienced in SQL*Forms when other people were updating the same table via SQL*Plus are no longer a problem. Oracle Version 6 with the Transaction Processing Option (TPO) prevents whole tables from being locked for any significant periods of time. Only individual rows are locked for short periods during the update process.

The **<ROLLBACK>** key can be used to make all changes that have not yet been committed to the database revert to the original values.

Deleting Records

To *delete* a record currently on the screen use the **<DELETE RECORD>** key. Typically, a group of records will have been retrieved with a query and you will then step through them with the **<NEXT RECORD>** key, deleting some or all records. In multi-record screens make sure the cursor is really positioned on the record you want to delete before pressing the **<DELETE RECORD>** key.

You must use the **<COMMIT>** key to make all deletions permanent. This can be done after deleting each record, after deleting a batch of records, or when leaving the form. The function **<CLEAR RECORD>** removes a record temporarily from the retrieve list, making it effectively invisible. But it is not 'deleted' and can be retrieved again with a new query.

Here again, all deletions not yet committed to the database can be made undone by pressing the **<ROLLBACK>** key.

Entering Data

To enter new records into the database you **must** start with a blank form - either that which presents itself on first entering the form, or that produced by pressing the **<INSERT RECORD>** key. **If not, you are repeatedly updating an existing record, not inserting a new one!** In general, one should press **<INSERT RECORD>** before any new data are entered, even if this is not, in fact, needed with an empty screen.

However, it is often useful to retrieve a previously entered record to save having to retype all fields. To retrieve all data that were on the screen prior to using the **<INSERT RECORD>** key, press **<DUPLICATE RECORD>** and enter, modify or delete fields as required. Alternatively, if you only want to duplicate certain field values, call up the previously entered record, press **<INSERT RECORD>**, then press **<DUPLICATE FIELD>** only in the fields you want to copy across from the previous record.

Like updated data, new records must be committed to the database with the **<COMMIT>** key, or when prompted on exit from the form. Pressing **<ROLLBACK>** before committing them or answering NO on the exit prompt wipes them from the entry form.

Pop-up Lists of Values

In version 3 forms, most fields with a look-up table have a pop-up list from which you can select a value. The message **'Pick list available - press LIST'** is usually displayed at the bottom of the screen when the cursor is in the field. A list of Originators from the Sites form in the NGMA Field Database is shown in Figure 7. Press **<LIST>** to display the pop-up list, **<NEXT FIELD>** to place the cursor in the *Find* field, enter *s*, and press **<NEXT FIELD>** again to show all values starting with 's'. Several letters may be entered in the *Find* field without the need to add a '%' symbol to the end of the string. To select a value back into the form just place the highlight bar on it and press **<ACCEPT/COMMIT>**.

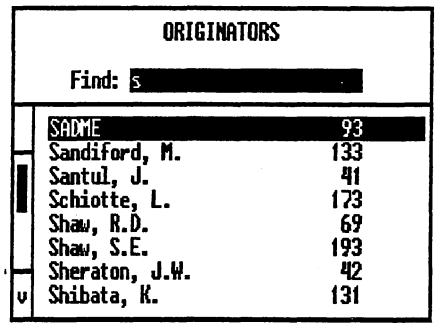


Figure 7. A pop-up list of values

Modifying/Creating Forms

Only the system owner can modify an existing form or create a new one for general use within the database, and it can only be done in the Test Environment. The modified or new form can be migrated to the Production Environment after sufficient testing (see AGSO Record 92/85). However, users with resource privileges and access rights to a particular database can create forms for their own use.

The need for modifying an existing form arises when the underlying table (= base table) is changed, ie. column/s added, column size/s and/or data type/s changed (see Appendix C). If the form is not modified after such changes are made to the table structure it may not be possible to use the form for correct data entry. Indeed, sometimes it cannot be used at all and the attempted use will only create error messages.

Likewise, a form might have to be modified if a new table is added to the database or a table is dropped. Sometimes a completely new form might have to be created in these cases.

Refer to the SQL*Forms Designer's Reference or Designer's Tutorial for full instructions for creating or modifying a form.

11 - SQL*NET

SQL*Net has been installed on the AViiON to provide distributed access to AGSO databases, eg. direct access to Oracle databases on the AViiON from within the Arc/Info GIS system. It also provides access to data in one instance of Oracle (Test or Production) from within the other instance. Access is via so-called *links*.

Publicly accessible database links which automatically pass on the current user's login combination to the remote database have been set up and should be used whenever possible. They are called **oratest** for accessing the Test Environment from the Production Environment and **oraprod** for accessing the Production Environment from the Test Environment. These links assume that the current username/password combination exists on the other database environment. If the user wants to access the remote database with a different username/password combination, a private database link has to be created (see below).

Thus, a user in the Test Environment can select data in the Production Environment by issuing a command like:

select * from rtmap.landf@oraprod;

to retrieve all the landforms currently in the landforms lookup table of the RTMAP Regolith Terrain Mapping Database which is the AGSO authority table for landforms.

Links can be created by typing:

CREATE DATABASE LINK linkname
CONNECT TO username IDENTIFIED BY password
USING 'connect string';

where *linkname* is the name of the link being created;
username and *password* are an existing username/password combination on the remote database to be accessed through the link;
connect string is the connect ID of the remote database ('T:av:oraprod' or 'T:av:oratest').

Users can create their own private links using the above command. The problem is that this way the user's username/password combination is coded into the link. At this stage there appears to be no way of creating a dynamic database link which prompts for the username and password on invoking it. A private database link can only be used by its creator ('owner'), other users cannot be granted access to it.

12 - AUTHORITY TABLES AND DATA STANDARDS

In much the same way that standards are pivotal to the automation of manufacturing industries, data standards are essential when automating the management and presentation of information. Authority tables, also known as validation or lookup tables, are the main method of imposing standards in databases, and AGSO databases use them extensively.

AGSO has developed and used standards in nomenclature and classifications in many geoscience disciplines over many years. These standards have now found their appropriate representation in several AGSO Oracle databases, eg. STRATLEX, a list of Australian stratigraphic names; LITHNAMES for lithological nomenclature; AGSOMINERALS containing mineral names; COMMODS for commodities within a deposit; GEOPROVS, a list of Australian geological provinces (all part of the NGMA Field Database and OZMIN, the NGMA Mineral Deposits and Occurrences Database); LANDF, a list of landforms as part of the RTMAP Regolith Terrain Mapping Database; and VEGET, a classification of Australian vegetation types, used by QUATDB, the Quaternary Climates Database.

Ideally, standard nomenclature and classification tables should only exist once within AGSO's database system under the proviso that any user within the organisation may access and use the information contained in them when necessary. Thus, most authority tables are publicly accessible to all internal AGSO users (this accessibility is to be extended to eligible outside users in due course). The data in these authority tables are kept up-to-date by designated data custodians who are usually the most qualified people within the organisation to do so through their intimate knowledge of the particular subject matter, and their standing within the scientific community.

Authority tables are important mechanisms in AGSO's relational databases for controlling data inputs and imposing standard classification schemes. They can also relieve data input personnel of much of the tedium of entering repetitive data, and they facilitate comprehensive data retrieval. For example, with the help of the NGMA.ORIGINATORS table it is not necessary to type in the full name of the originator of a batch of samples for each sample, all that is needed is to input the originator's number into each record which can be copied out of the lookup table. The full name of the originator can be related to this number anytime, and is usually displayed in data entry forms as a means of data validation.

Also, when entering an originator number in the NGMA.SITES table for instance, only the numbers that are already in the NGMA.ORIGINATORS table may be used. Thus, it is not possible for 'Brown, B.B.' to be mistakenly entered as 'Brown B.B.' (without the comma after 'Brown') and for later retrievals to omit that record because of a missing comma. Other authority tables which define standard nomenclature or classifications prevent the ambiguities that inevitably arise when free-text fields are used.

13 - CLIENT-SERVER METHODS

Most users will have already heard the term 'client-server'. AGSO currently uses one type of client-server computing in its extensive Sun UNIX network, but in the sense of the term commonly applied to database systems AGSO has yet to wholeheartedly embrace client-server methods. An example, though, is provided by the Arc/Info system, which can directly access Oracle via SQL*Net (Chopra & Ryburn, 1993). In this symbiotic relationship the Oracle RDBMS on the AViiON acts as the database server to the GIS client.

In client-server database architecture a centralised server specialises in the core database functions while the distributed client applications handle all the other aspects of the computing system (Ryburn & Lenz, 1991). The current situation in AGSO with respect to SQL*Forms does not conform to this idea, as all forms are stored and controlled centrally on the AViiON server. However, client versions of Oracle's SQL*Forms are available that can run on a PC or UNIX work station under DOS, Macintosh System, Microsoft Windows, Open Look, Motif. etc. Current AGSO forms can be quickly and easily converted to run under these environments. Oracle supplies a variety of other client packages such as Oracle Graphics, Data Browser and Card. Client software for directly connecting Microsoft Excel, and other popular PC software packages, is also available. Client 'front ends' that will work with an Oracle server are also available from a number of independent software suppliers.

Client systems have a number of advantages. They isolate the application from the database, permitting a choice of 'front-end' software from a variety of suppliers. In theory, different database servers can also be used but in practice this ideal has yet to be properly realised. Client-server architecture allows the corporate database to be used with sophisticated graphical user interfaces - with all the power and user-friendliness of modern PC or UNIX work station software. Images and vector linework can be handled and a mouse used for pointing and selecting. Client-server systems also act to minimise the amount of traffic on corporate networks.

The disadvantages of client-server methods, which largely account for AGSO's reluctance to jump too quickly into this technology, are to do with cost, complexity, duplication of effort and lack of control. Most client software packages require the user to purchase an expensive run-time licence and to pay annual maintenance fees. An IBM compatible PC, for example, needs to be a 386/486 with 8 to 16 Mbyte of memory. SQL*Net software will have to be purchased and installed, and TCP/IP network drivers may be needed over and above those already used for Novell networks. New programming languages and software development environments must be mastered by the applications developer. Different versions of software and applications inevitably wind up on different PCs, and when the structure of the central database is changed each copy of a client application may also need to be changed, creating additional work.

A recent development, that may further inhibit the adoption of client software, is the ability of version 4 of Oracle Forms to function fully on a UNIX X-terminal, thus providing the user with a proper graphical user interface with centrally controlled applications software. With the help of software like HCL-eXceed, PCs can act as X-terminals (Chopra, 1991a & b). X-terminals will allow the use of a mouse and the display of images held in an Oracle database. Ironically, X-terminal software is itself a form of client-server computing.

Where client packages clearly have a future in AGSO is in specialised areas to do with imaging, and graphing and mapping of data. In addition to the established Arc/Info GIS software for interactive mapping, there is also a need for simple PC packages for one-way plotting, projecting, mapping and analysis of corporate data. Oracle Graphics is one example of a client graphing software, while MapInfo, which is already used in AGSO for stand alone applications, is also capable of acting as a mapping client to the corporate database.

14 - FUTURE DEVELOPMENTS

Oracle Version 7 is scheduled for installation on the AViiON in the first half of 1994. It will provide increased performance, security and functionality. In particular, referential integrity and trigger-based rules can be built into the database kernel, rather than into forms and other applications. This makes it easier for files of data to be properly validated when loaded directly into the database with SQL*Loader. Distributed databases become a more practical proposition with Version 7, although there appears to be no immediate need for this in AGSO. With Version 7 comes PL/SQL version 2, providing most of the functionality found in third generation languages such as Fortran. This will eliminate most of the present need for Pro-Fortran programs with embedded SQL, and will allow complex triggers to be included in forms for tasks such as map coordinate conversion.

At the same time as Version 7 is installed, SQL*Forms 4 will begin to be used. The main change with SQL*Forms 4 will be the ability to run centrally generated forms (as opposed to 'client' versions of forms) in a windowed graphics environment on a UNIX X11 terminal, or a PC running X-terminal software. This will allow images to be displayed, and also support the use of a mouse. The PALEO database, for example, will be able to incorporate photographs of fossils in the database. SQL*Menu becomes a part of SQL*Forms Version 4 rather than a separate application.

Further in the future is the probable adoption of 'object-oriented' database techniques. Object-oriented DBMSs, which stem from the concepts embodied in object-oriented languages like Smalltalk, permit manipulation of complex self-defined data types. Whether this step will take place by evolution or revolution cannot yet be foretold, although Oracle appears to be moving rapidly towards the adoption of new and user-definable data types in its databases. This should enable the development of 'multimedia' databases with images, vector diagrams, sound, and video.

Although a number of pure object-oriented database management systems (OODBMSs) are now available (eg. Object Store), none of these appear mature or robust enough, or financially viable, to risk using them in mission-critical applications in a shared corporate environment. Most of these OODBMSs require programs to be written in C++, or similar, for all database operations - including the selection of data. No accepted standards for OODBMSs yet exist, and they have no theoretical basis like the relational algebra and set theory that provides the foundation for relational databases. Nevertheless, for some applications like computer-aided design (CAD) and GIS they offer considerable performance advantages.

The most likely scenario is that relational and object-oriented databases will begin to merge. There are already several hybrid systems on the market that retain a standard database manipulation language like SQL, or some extension of it. The database management system of the future will have to cater for the regular data of 'high cardinality' currently handled by RDBMSs, as well as the complex self-defined data types currently best managed by OODBMSs.

15 - BIBLIOGRAPHY

Boston, T., 1989, Copy Service Database User Manual. *Bureau of Mineral Resources, Australia, Record* 1989/7.

Chopra, P.N., 1991, PC X11 Windows Servers Providing Network Access to UNIX Graphics. *Bureau of Mineral Resources, Australia, Record* 1991/97.

Chopra, P.N., 1991, Configuration options for the HCL-eXceed/W PC X11 server. *Bureau of Mineral Resources, Australia, Record* 1991/106.

Chopra, P.N. & Ryburn, R.J., 1993, Linking continental databases in the Oracle RDBMS with project data in the Arc/Info GIS. *Australian Geological Survey Organisation, Record* 1993/12.

Kucka, M., 1992, Dual Oracle Database Environment and Change Control Management. *Australian Geological Survey Organisation, Record* 1992/85.

Lenz, S. & Modrak, K., 1990, The Stratigraphic Index Database GEODX, User Manual. *Bureau of Mineral Resources, Australia, Record* 1990/16.

Lenz, S. 1991, RTMAP: BMR Regolith Database, Users' Manual. *Bureau of Mineral Resources, Australia, Record* 1991/30.

Lenz, S.L., McCue, K.F. & Small, G.R., 1992, QUAKES: BMR-ASC World Earthquake Database, Users' Manual. *Bureau of Mineral Resources, Australia, Record* 1992/14.

Lenz, S.L., 1993, NUCEXP: Nuclear Explosions Database, Users' Manual. *Bureau of Mineral Resources, Australia, Record* 1993/39.

Page, R.J., Wyborn, L.A.I., Hazell, M.S. & Ryburn, R.J., 1993, OZCHRON documentation. *Australian Geological Survey Organisation, Record* 1993/44.

Ryburn, R.J., 1990, User's Guide to the Petchem Database. *Bureau of Mineral Resources, Australia, Record* 1990/19.

Ryburn, R.J., 1992, Relational Databases for continent-wide data. *In*: Geographic Information Systems, Cartographic and Data Standards, Workshop Proceedings. *Bureau of Mineral Resources, Australia, Record* 1992/27.

Ryburn, R.J., Blewett, R.L., Stuart-Smith, P.G. & Williams, P.R., 1993, Users' Guide to the NGMA Field Database. *Australian Geological Survey Organisation, Record* 1993/49.

Ryburn, R.J. & Lenz, S., 1991, Geoscientific relational databases in BMR and the client-server method. *Proceedings of the National Conference on the Management of Geoscience Information and Data, Adelaide, 1991*, 15pp.

Ryburn, R.J., Page, R.W. & Richards, J.R., 1993, Users' Guide to the OZCHRON Database. *Australian Geological Survey Organisation, Record* 1993/11.

Tucker, A., 1993, Procedures to Access Point Spatial and Attribute Data in an Oracle Database from within the ARC/INFO GIS. *Australian Geological Survey Organisation, Record* 1993/73.

Oracle User Guides and Reference Manuals:

ORACLE RDBMS V.6.0 Database Administrator's Guide, *Oracle Corporation.*

ORACLE V.6.0 Utilities User's Guide, *Oracle Corporation.*

PL/SQL User's Guide and Reference V.1.0, *Oracle Corporation.*

SQL*Forms V.3.0 Designer's Reference, *Oracle Corporation.*

SQL*Forms V.3.0 Designer's Tutorial, *Oracle Corporation.*

SQL*Plus V.3.0 User's Guide and Reference, *Oracle Corporation.*

SQL Language Reference Manual V.6.0. *Oracle Corporation 1990.*

SQL*Menu User's Guide and Reference V.5.0, *Oracle Corporation.*

SQL*Net TCP/IP User's Guide V.1.2, *Oracle Corporation.*

SQL*Report User's Guide V.1.1, *Oracle Corporation.*

SQR Structured Query Report Writer User Guide, 1990, *SQ Software, Inc.*

16 - GLOSSARY

ADD part of SQL command ALTER TABLE for adding a new column to a table;

ALTER SQL command for making changes to the database structure;

ANSI **American National Standards Institute**

ASCII **American Standard Code for Information Interchange;**

ATTRIBUTE column in a database table;

BACKUP duplicate of database objects which enables them to be restored later to the state they were in at the time of backup;

CAD **Computer-Aided Design;**

COLUMN fields representing one kind of data (one attribute) in a table;

COMMIT SQL*Plus command for storing (making permanent) changes made to database tables;

CREATE SQL command for setting up new database objects;

DATA CUSTODIAN user charged with looking after the data in database objects;

DATABASE SCHEMA SQL script for creating all objects of a database, ideally including a detailed description of the objects and their data items;

DBA Database Administrator;

DELETE SQL command to remove rows from a table;

DESCRIBE SQL*Plus command for listing names and specifications of all columns in a table;

DROP SQL command to permanently remove database objects;

ENTITY distinguishable system object represented in the database;

EXPORT Oracle utility for moving database objects and data to operating system files;

GIS **Geographic Information System;**

GRANT assign access rights to a database user;

GUI Graphical User Interface;

IMPORT Oracle utility for moving exported database objects and data back into the database;

INDEX database object that enables finding a specific row without examining the whole table;

INSERT SQL command for adding new rows to a table;

JOIN retrieve parts of a row from two or more tables at the same time;

KEY column/s in a table whose values uniquely identify the records in that table;

LIKE comparison operator for matching of a pattern which includes a wildcard;

MODIFY part of SQL command ALTER TABLE to change the definition of an existing table column;

OBJECT named database element maintained by the Oracle RDBMS;

OODBMS Object-Oriented Database Management System;

OUTER JOIN combine data from two or more tables and also return those rows from one table which have no direct match in the other table;

ORACLE RDBMS relational database management system sold by Oracle Corporation;

OWNER user who has created a database object and has all access rights to it;

PL/SQL extension of SQL; adds procedural capabilities to Oracle's SQL*Plus and SQL*Forms;

PRODUCTION Oracle database environment which is tightly controlled containing all final working databases;

RDBMS Relational Database Management System: data storage and retrieval program which organises data into tables whose rows all have the same set of data items;

RECORD data in one row of a table; instance of an entity;

ROLE group of database users with access rights to certain menus or menu items in SQL*Menu;

ROLLBACK SQL*Plus command to discard changes made to tables before they have been committed;

SELECT SQL command to retrieve data from one or more tables and/or views;

SQL	Structured Query Language; user interface for storing and retrieving data in a database;
SQL*FORMS	interface tool for creating, modifying and using forms to access an Oracle database;
SQL* LOADER	tool for loading data from ASCII files into an Oracle database;
SQL*NET	communications software that enables sharing of information stored in Oracle databases across a network;
SQL*PLUS	extension of SQL for producing formatted reports from an Oracle database;
SQR	Structured Query Report Writer used for accessing relational databases;
SYNONYM	name assigned to a table or view in addition to its actual name;
TABLE	grid of columns and rows; basic unit of data storage in an RDBMS;
TEST	Oracle database environment for creating, developing and trialling databases;
UNIX	operating system on the AViiON corporate database server;
UTILITY	program run by an operating system command to perform functions associated with Oracle;
VIEW	database object similar to a table derived from another/other table/s; has no storage of its own;
WHERE	clause in SQL to specify conditions for an operation on a relational database;
X11	X Window System Version 11 developed by the Massachussetts Institute of Technology; a hardware and software independent graphics environment.

APPENDIX A: ORACLE DATABASES ON AGSO'S AViiON SERVER

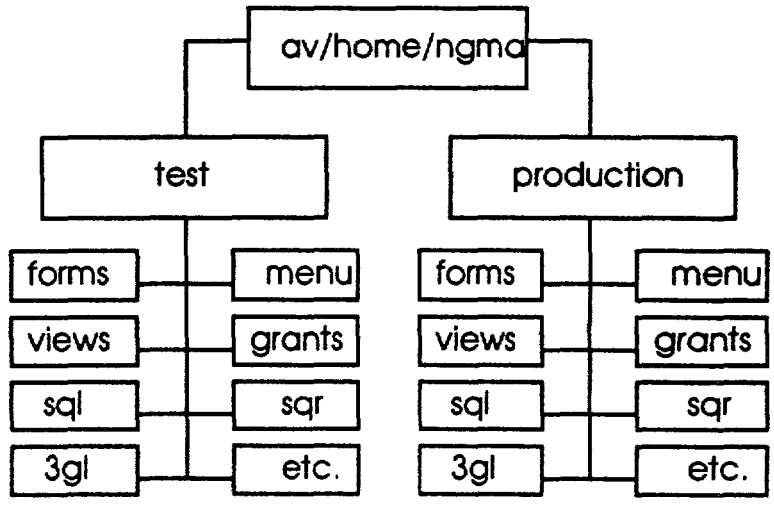
Database	System owner	Access command
Publicly accessible:		
GEODX - Stratigraphic Index Database	Anne Franklin	*geodx*
LOCATION - Australian Locations	David Walton	-
MINLOC - Mineral Locations Database	Brian Elliott	*minloc*
NGMA - NGMA Field Data	Richard Blewett	*ngma*
NPD - National Petroleum Database	Sandy Radke	*npd*
NUCEXP - Nuclear Explosions Database	Lesley Hodgson	*nucexp/nucexpsun*
PALEO - Fossil Collections Database	Des Strusz	*paleo*
PETROG - Petrography Database	Jan Knutson	-
QUAKES - World Earthquakes Database	Kevin McCue	*quakes/quakessun*
RTMAP - Regolith Terrain Mapping Database	Colin Pain	-
STRATA - Stratigraphic Authority Database	Rod Ryburn	-
Not publicly accessible:		
EABASIN - Eastern Australian Basins	Malcolm Nicoll	
COPYSERV - Copy Service Database	Gillian Tidey	
ESU - ESU Project Costing Database	Neville Esau	
GAB - Great Artesian Basin	Andrew Tucker	
INFO - AGSO Corporate Mailing List	Sonja Lenz	
ORGCHEM - Organic Geochemistry	Chris Boreham	
OZMIN - Australian Mineral Deposits	Greg Ewers	
OZCHRON - Isotope Geochronology	Rod Page	
PALMAG - Palaeomagnetic Database	John Giddings	
QUATDB - Quaternary Climates Database	Evert Bleys	
REMOS - Remote Sensing Database	Taro Macias	
RESFACS - Petroleum Reservoirs, Shows, Facies	John Bradshaw	
RGMAG - Regional Magnetics Database	Andrew McEwin	
ROCKCHEM - Whole-rock Geochemistry	Lesley Wyborn	
SEISMIC - Seismic Reflection Surveys	Clive Collins	
STRATDAT - Stratigraphic Dating Database	Clinton Foster	
VOLCBIB - Volcanic Bibliography	David Palfreyman	

Current: November 1993

APPENDIX B: RECOMMENDED DIRECTORY STRUCTURE

Under the home directory of an Oracle database are directories ***test*** and ***production*** which each have separate subdirectories for SQL*Forms (***forms***), SQL*Plus (***sql***), SQR reports (***sqr***), SQL*Menu (***menu***), views (***views***) and others.

For example, this is the directory structure for the NGMA Field and Laboratory Database System:



The Test and Production directories should ideally each contain a copy of the current version of the main program files as a backup in case some disaster happens to files in either directory. Files in the Production subdirectories belonging to the database username (in the above diagram: NGMA) should be the ones that all users of the particular database access and use from the system menu. That is not to say that users cannot have copies or different versions of the various system files in their directories but the database system administrator should take care to ensure that the latest working versions of the files are the ones that all users of the database are executing from within the system's menu. A shell script similar to this one for the NGMA database should reside in the home directory for accessing the database in the Production environment:

```
. setoraprod
ORACLE_PATH=$ORACLE_PATH:/home/ngma/prod/forms
export ORACLE_PATH
SQR_PATH=/home/ngma/prod/sqr
export SQR_PATH
SQLPATH=/home/ngma/prod/sql
export SQLPATH
runmenu50 ngma -m f
```

APPENDIX C: CHANGING THE DATABASE STRUCTURE

As a database is used it will most certainly become apparent that the present design is not perfect and tables might have to be added or dropped, columns added to existing tables or their width and/or datatype changed. The system owner (or anyone who has been granted this right by the system owner) has the right to make changes like these to the structure of the database in the Test Environment (see Section 5).

The following sets out the basic procedures for making those changes using the commands ***create***, ***drop***, ***alter***, ***rename***. Set the Oracle environment to Test (***setoratest***) and log into SQL*Plus by typing

sqlplus

You will be prompted for your Oracle user name and password.

First a Word on Oracle Tablespaces

A tablespace is a storage space for Oracle data. Besides the system tablespace which contains all the system data, several tablespaces for user data have been created by ISB's database administrator. Both Oracle environments use the same tablespace naming convention. The tablespaces are called: TBSPA, TBSPB, TBSPC, and the indexspaces are called: INDXA, INDXB and INDXC. All six tablespaces are accessible to you in the Oracle Test environment.

All userids have a default tablespace assigned to them, which may be TBSPA, B or C. Only the table data should reside in tablespace TBSPA, B or C. The table's indexes should reside in tablespace INDXA, B or C, respectively. That is if table FRED resides in tablespace TBSPB, then all of FRED's indexes should reside in tablespace INDXB, etc.

Note that when creating tables, the storage for the initial and next extent is in bytes, not blocks as in the previous version of Oracle. For example:

```
create table FRED
      (fredkey      number not null primary key,
      fredfield1    char(5),
      lastfield     char(10))
      tablespace tbspb
      storage (initial 10k next 2k);
```

will create the table in tablespace TBSPB, with an initial extent (= space allocation) of 10,240 bytes and next extent of 2048 bytes. Note: you can use the suffix 'm' to indicate Megabytes, and 'k' to indicate Kilobytes. If no suffix is provided, bytes is assumed.

The same storage parameters apply to indexes.

Adding a Database Object

The *create* command lets you create new tables, indexes, synonyms and views. For examples of create statements, see database schemas which are part of the database documentation.

The general statement for creating a new *table* is:

```
CREATE TABLE tablename
        (column1 datatype1(size1),
         column2 datatype2(size2),
         etc)
TABLESPACE tablespace
STORAGE (INITIAL size NEXT size);
```

Mandatory columns must be specified as NOT NULL.

Indexes on big tables are a means of speeding up retrieval times. They can also enforce uniqueness of records. However, too many indexes will slow down update activity unduly. An index can be **concatenated** which means that a combination of columns is used for indexing.

To create an index in SQL*Plus, specify its name and the table with its column/s that contain/s the information to go into the index:

```
CREATE INDEX indexname
        ON tablename (column1,column2,...)
TABLESPACE tablespace
STORAGE (INITIAL size NEXT size);
```

If the index is specified as a **unique** index Oracle will make sure that there are no two entries (records) in the table with the same value/s in the specified column or combination of columns.

A *view* is a table-like database object which is derived from columns from one or more tables (or other views). It has no storage of its own. The syntax for creating a view is:

```
CREATE VIEW viewname AS query;
```

The query is a select statement for selecting all columns which make up the view.

If you are the owner of a table or a view or you have select access to them you can create *synonyms* as alternate names for these objects (for security or convenience reasons). The syntax is:

```
CREATE SYNONYM synonym_name FOR tablename;
```

Dropping a Database Object

The *drop* command is used to permanently remove tables, indexes, synonyms and views.

Before dropping a table make sure that it does not contain data which are still needed (the table could still be used by other systems) as it can NOT be restored (there is no ***undrop*** command) and the data are permanently lost. At the SQL-Test> prompt, type

DROP TABLE tablename;

All existing indexes and grants on the table will be dropped at the same time. Synonyms and views remain in the database but they are invalid and have to be dropped and redefined.

Indexes, views and synonyms can be dropped with the same statement (substitute *TABLE* and *tablename*).

Altering a Database Object

The *alter tablename* command in conjunction with a *modify column* or *add column* clause is used to make changes to tables.

A column's width and/or datatype can be changed with the following SQL statement:

ALTER TABLE tablename
MODIFY (columnname datatype(size));

To modify more than one column at a time, use commas within the parentheses to separate each column and its definitions from the next.

For instance, the length of fields *site* and *description* in the NUCEXP.XSITES table could be altered in the following way:

alter table xsites
modify (site char(5) not null primary key,
description char(60) not null);

To change a mandatory field to a non-mandatory one, add the NULL clause to the end of the column specification. A non-mandatory field can only be changed to a mandatory one (NOT NULL) if there are no null values in the column.

A column can be added to an existing table with the command:

ALTER TABLE tablename
ADD (columnname datatype(size));

To add more than one column, use commas within the parentheses to separate each column and its definitions from the next.

For instance, field *m_max* was added to table QUAKES.DETAILS with the statement:

alter table details
add(m_max number(3,2));

All fields in a new column initially have a value of null. Therefore, a new column added to an existing table cannot be defined as NOT NULL when the table already contains records. If a new column is to be made mandatory, add the column, then give every record a non-null value, then use the ALTER TABLE command with the MODIFY clause to change it to NOT NULL.

After altering table definitions, ie. adding or modifying columns, it might be necessary to re-create existing views. Also, existing data entry forms might have to be amended to accommodate the changes.

Renaming a Database Object

The owner of a table, view or synonym has the right to *rename* these objects. The command is:

RENAME oldname TO newname;

eg. table OLDDATA in the QUAKES database was renamed to the more appropriate name NOMAGS ('no magnitude values') with the following statement:

rename olddata to nomags;

Indexes and grants which belong to the object are transferred to the new name.

APPENDIX D: ADDITIONAL NOTES AND HANDY HINTS

Data Export, Import and Backup

Export and **Import** are two Oracle utilities. Export transfers Oracle data to binary operating system files, Import moves the data back into the database. The export files can be used for archiving data (= data backup) or moving data between operating systems or Oracle databases. The following types of Oracle data can be stored in this way: table definitions, table data, indexes, space definitions, grants, synonyms, and view definitions.

With the export utility, the database definitions and/or the data in the database are written to a special kind of backup file. This export file is in a special format and, therefore, it should not be edited. It can only be used by the import utility which restores the exported data into an Oracle database - each table is re-created and its data loaded back into it. Depending on the options selected, problems with uniqueness of records can be caused if data are added to a table which already contains records.

Export and Import are interactive utilities - you are asked questions and the utility proceeds according to your answers. To run the utilities, type

exp or ***imp***

and respond to the questions (first, you will be prompted for your Oracle user name and then your password). It is possible to set up files with appropriate responses so that the utilities can be run in batch.

For details and further information refer to Oracle Utilities User's Guide.

On AGSO's AViiON, full backups of all databases are done on a regular basis by the Information Systems Branch computer operators. In the case of a system failure or human error with consequent data loss or corruption, the databases can be restored. Contact the database administrator in Informations Systems Branch if this kind of help is needed.

Execution of SQR Programs

Structured Query Report Writer (SQR) is a third-party report writing program. With it you can select data from the Oracle database and create neatly formatted output. SQR does not use path variables. If SQR is not run from the directory where its programs are stored, the full path name of the program has to be specified. For example, the following shell script can be set up and run from any directory to retrieve all reserved names currently in the GEODX Stratigraphic Index database and put them into a file called allreslist:

```
sqr /home/geodx/prod/sqr/allreslist allreslist
```

Use of Oracle Pseudo Columns

SYSDATE and ***USER*** are two Oracle pseudo columns which return the current date and the username of the current user. The syntax is:

```
select SYSDATE,USER from dual;
```

Use of Oracle Functions

Functions are used to manipulate individual data items. Following are examples of the most commonly used functions in Oracle:

1. Number functions

```
select COUNT(recno),MAX(recno),MIN(recno) from quakes.details;
```

returns the number of records and the maximum and minimum value of the identifier *recno* from the earthquakes details table.

```
select ROUND(dlat,4) from paleo.location where not dlat is null;
```

retrieves the decimal latitude from the fossil locality table rounded to 4 decimals right of the decimal point.

```
select SUM(copies) from info.publications_sent
    where pubref = 55;
```

calculates the number of copies of the *Database News* that have to be produced each issue (AGSO Mailing List Database).

2. Character functions

```
select * from paleo.acquisition
    where INITCAP(collector) like 'Chapr%';
```

will retrieve all records in the acquisition table of the PALEO database whose entry in field

collector starts with any upper or lower case combination of 'Chapr'.

```
select max(LENGTH(description)) from nucexp.xsources;
```

tells you what the maximum length of the entries in the *description* field of the NUCEXP XSOURCES table is.

```
select * from all_catalog
      where LOWER(owner) = 'geodx';
```

returns all tables and views belonging to the GEODX database to which you have access, no matter whether the word GEODX is in the database in upper or lower case.

```
select * from paleo.store
      where SUBSTR(refno,1,3) = 'MFP';
```

retrieves the storage records for all microfossil specimens in the PALEO database whose identifier (*refno*) starts with the three letters MFP.

```
select * from nucexp.explosions
      where UPPER(site) = 'URAL';
```

will retrieve all the explosions from the Nuclear Explosions Database which were detonated in the Ural Mountains, no matter whether the site name is in the database in upper or lower case.

3. Conversion functions

To perform operations on field values which can be null, the NVL function may be used to substitute a value for all nulls. For instance,

```
update temp_details set m_max =
GREATEST(NVL(mb_aver,0),NVL(ms_aver,0),NVL(ml_aver,0));
```

is part of the command the staff at the Australian Seismological Centre use to update the maximum magnitude (field *m_max*) to the greatest value of all the different magnitude determinations. In this case, mb_aver,ms_aver and ml_aver are assigned a value of 0 if the field is null. This example also shows the use of the GREATEST function. The corresponding function for the smallest value is LEAST.

Values in number fields can be changed to values of character data type with the TO_CHAR function, eg.

```
select acronym||TO_CHAR(num)||suffix from paleo.specimen
    where not acronym is null;
```

returns the three parts of the unique identifier for each specimen concatenated to form the key field *refno*. (Fields *acronym*, *suffix* and *refno* are character fields, whereas *num* is a number.) Accordingly, fields of character data type which consist wholly of numerals can be used in arithmetic operations after their values have been converted to the number data type with the TO_NUMBER function.

Finding out about Duplicate Values

If a certain attribute value is repeated in more than one record the following statement can be used to find out how often that occurs in a table:

```
SELECT fieldname FROM tablename
    GROUP BY fieldname
    HAVING COUNT(*) > 1;
```

Use of Views in the NGMA Field Database

All internal Oracle users on the production environment can add, change or delete their own data in the NGMA Field Database. This is accomplished via views on the respective base tables.

The 'Insert-Update' forms cover these views. The restrictions applying to the views are the same in each case. For example, the view USITES of the SITES tables is defined as:

```
create view usites as
    select * from sites
        where enteredby = user;
```

The word *user* in the above statement is an Oracle function that returns the current user name. The tables have the mandatory field *enteredby* for the user name of the person entering the data. This scheme guarantees that the users see only their own records in the 'Insert-Update' forms, and only they or the data custodians can alter or delete them.

Users wishing to use SQL*Plus to insert, update or delete records in the tables (or SQL*Loader to load records from an ASCII file) must use the views.

Map Coordinate Conversion

A common requirement in AGSO is to be able to convert between Australian Map Grid (AMG) metric eastings and northings, and latitudes and longitudes. Both types of coordinates are required by many AGSO databases, and an ability is needed to convert between the two types of coordinates in both directions.

A Pro*Fortran program called SETLONG can be used with any Oracle table containing columns for AMG Zone, metres east, metres north, decimal latitude and decimal longitude for converting map coordinates in either direction. If a column specifically for AMG zone is not present, then the middle two digits of a 1:250 000 map ID can be used instead (eg. 'substr(qmapid,3,2)'). The direction of the conversion is determined by the presence and absence of data in individual records, and conversions in both directions take place in a single pass through a table. If data is present in the fields for AMG Zone, easting and northing - but latitude and longitude are null - then the latter values will be updated in the record. If lats and longs are present, but the zone, easting and northing are empty, then the AMG reference will be calculated and updated in the record. The program will not update any records that already have data in all the relevant fields. In other words it will not overwrite any pre-existing data. The spheroid used is the 1966 AMG spheroid, and the accuracy of conversion is to within a fraction of a metre.

To run SETLONG just enter ***setlong*** from the AViiON's UNIX prompt. After prompting you for your Oracle username and password, SETLONG asks for the name of the Oracle table and the names of each of the required five columns in the target table. It then processes the entire table, reporting on the screen each record as it is updated. If you wish to obtain the source code for SETLONG you can copy the Pro*Fortran source file into your current directory with the UNIX command

cp /home/ngma/test/setlong/setlong.pfo .

In addition to this program, we have written another Pro*Fortran program, COORDCONV, that works in conjunction with a trigger for two-way conversion of coordinates in Oracle forms. The latest version of the NGMA.SITES table uses a 'NEXT KEY' trigger to convert coordinates just entered into the form. However, this program is a rather inefficient interim measure that will be replaced by a much quicker and more elegant PL/SQL trigger in version 4 of Oracle Forms. PL/SQL cannot be used at present as it does not have the trigonometric functions required for coordinate conversion.

Map Sheet Area from Latitude and Longitude

The NGMA.HMAPS and NGMA.QMAPS tables allow the 1:100 000, 1:250 000 or 1:1 000 000 map sheet area to be found for any point defined as a latitude and longitude. These tables contain a complete list of all map sheet areas in Australia at the three scales. The HMAPS table

also contains PNG 1:100 000 maps and is defined as follows :

```
create table hmaps (
    hmapno      number    (4,0)  not null  primary key,
    mmapid      char      (4),
    qmapno      number    (2,0),
    hmapname    char      (22),
    n_lat       number    (3,1),
    w_long      number    (4,1),
    meast       number    (6),
    mnorth      number    (7),
    state1      char      (3),
    state2      char      (3) );
```

The minimum latitude and longitude at the north west corner of each sheet are given by the fields *n_lat* and *w_long*. Similarly, the fields *meast* and *mnorth* record the minimum AMG easting and northing at the southmost corner of each sheet. The following statement retrieves number and name of the 1:100 000 map on which a given latitude [lat] and longitude [long] falls:

```
select hmapno, hmapname from ngma.hmaps
    where [lat] >= n_lat  and  [lat] < n_lat+.5
    and  [long] >= w_long and [long] < w_long+.5;
```

Note that the 'between' SQL operator should not be used in the above select statement, otherwise as many as four maps can be returned in some cases. The statement required to obtain the 1:250 000 map ID from a given 1:100 000 map number [hmap number] is :

```
select mmapid || lpad(to_char(qmapno),2,'0') from ngma.hmaps
    where hmapno = [hmap number];
```

The somewhat clumsy concatenation in the above statement is needed because the *qmapno* column in HMAPS contains a number between 1 and 16 indicating one of sixteen 1:250 000 sheets on any given 1:1 000 000 map.

To obtain the name of a 1:250 000 sheet from its 6-character ID you must refer to the NGMA.QMAPS table. This table is defined as follows :

```
create table qmaps (
    mapno       char      (6) not null primary key,
    mapname     char      (22),
    n_lat       number    (3,1),
    w_long      number    (4,1) );
```

In this table the 1:250 000 map ID, which is a 6-character string like 'SF5403', is somewhat perversely called *mapno* rather than *qmapid*. It is probably too late to change this attribute name now, as too many other tables, forms and reports depend on it as it is.

Australian States from Latitude & Longitude

For any given latitude and longitude the correct Australian state can be determined - in most cases unambiguously - from the NGMA.HMAPS table. The table now has two additional columns, *state1* and *state2*, that give the state or states in which the 1:100 000 map sheet lies. For the vast majority of 1:100 000 sheets there is only one state, and only the *state1* field has the abbreviation for an Australian state as defined in the NGMA.AGSOSTATES table; *state2* is left null. For the small number of 1:100 000 sheets that straddle state boundaries, *state1* indicates the state covering the larger area on the sheet, and *state2* the state with the lesser area represented on the sheet. Note that the 30 metre discrepancy in the boundary between Victoria and South Australia has been ignored!

To determine the Australian state within which any given decimal latitude and longitude pair lies the following SQL select statement can be used -

```
select state1, state2 from ngma.hmaps
    where [lat] >= n_lat and [lat] < n_lat+.5
    and [long] >= w_long and [long] < w_long+.5;
```

If only *state1* returns a value then that value is unambiguous. If both *state1* and *state2* return values then the point could lie in either state.

Handling Hierarchical Tables

Although all Oracle tables are defined as simple two-dimensional structures, you can build up tree-like data structures within tables by using 'self pointers' - ie., pointers to the 'parent' record in the same table. A number of tree-structured tables exist in AGSO databases - eg. STRATA.STRATLEX, STRATA.GEOPROVS and STRATA.GEOTIME, all from the Stratigraphic Authority Database. In the Stratigraphic Lexicon (STRATLEX), **members** point to **formations**, which point to **groups**, which point to **supergroups**.

In the case of the STRATA.GEOTIME table you may want to obtain a listing of all records with the name of the parent rather than just its record number. To find out that the Bendigonian is an Australian stage of the Middle Ordovician is much more informative than the fact that its parent record is number 56. To achieve this result you must include two references to the table in the 'where clause' of the select statement, and distinguish between them by using aliases. For example, the following select statement :

```
SELECT
        SUBSTR(TO_CHAR(X.AGENO),1,3)        NO,
        SUBSTR(X.AGENAME,1,17)              AGE_NAME,
        SUBSTR(RANKNAME,1,7)                RANK,
        SUBSTR(SCOPENAME,1,13)              SCOPE,
        SUBSTR(STATUSNAME,1,8)              STATUS,
        SUBSTR(TO_CHAR(X.YNGBOUND),1,5)AGE1,
        SUBSTR(TO_CHAR(X.OLDBOUND),1,5)AGE2,
        SUBSTR(Y.AGENAME,1,17)              PARENT
FROM STRATA.GEOTIME X, TIMERANK, TIMESCOPE, TIMESTATUS,STRATA.GEOTIME Y
        WHERE X.RANK   = RANKNO
        AND  X.SCOPE  = SCOPENO
        AND  X.STATUS = STATUSNO
        AND  X.PARENT = Y.AGENO
        ORDER BY X.RANK, X.YNGBOUND, X.AGENAME;
```

produces the following report :

```
NO  AGE_NAME          RANK    SCOPE         STATUS   AGE1  AGE2  PARENT
--- ----------------- ------- ------------- -------- ----- ----- ----------------
163 Archaean          Eon     International Current  2600  4000  Precambrian
165 Cainozoic         Erathem International Current  0     65    Phanerozoic
166 Mesozoic          Erathem International Current  65    250   Phanerozoic
167 Palaeozoic        Erathem International Current  250   544   Phanerozoic
145 Neoproterozoic    Erathem International Current  544   1600  Proterozoic
146 Mesoproterozoic   Erathem International Current  1600  2000  Proterozoic
147 Paleoproterozoic  Erathem International Current  2000  2600  Proterozoic
27  Quaternary        Period  International Current  0     1.8   Cainozoic
38  Neogene           Period  International Current  1.8   24    Tertiary
37  Tertiary          Period  International Current  1.8   65    Cainozoic
39  Palaeogene        Period  International ..............
```

Oracle has a specific syntax for handling hierarchical tables, involving the terms 'CONNECT BY', 'PRIOR' and 'START WITH'. You will need to consult the SQL*Plus Users' Guide, but the following example shows you how to select a whole hierarchy of names from the STRATA.GEOTIME table - going from the lowest ranked term to the highest.

```
select ageno, agename, rank
        from strata.geotime
        connect by  ageno = prior parent
        start with agename = 'Bendigonian';
```

```
AGENO AGENAME                        RANK
----- ---------------------- ----------
  202 Bendigonian                     6
  127 Early Ordovician                5
  125 Ordovician                      3
  167 Palaeozoic                      2
  168 Phanerozoic                     1

6 rows selected.
```

It is also possible to traverse the tree downwards from any given 'start with' point :

```
select ageno, agename, rank, yngbound, oldbound from strata.geotime
    connect by  parent  = prior ageno
    start with  agename = 'Triassic'
    order by rank, yngbound;
```

```
    AGENO AGENAME                        RANK   YNGBOUND   OLDBOUND
--------- ---------------------- ---------- ---------- ----------
       86 Triassic                        3        204        250
       87 Late Triassic                   5        204        230
       88 Middle Triassic                 5        230        242
       89 Early Triassic                  5        242        250
       90 Rhaetian                        6        204        210
       91 Norian                          6        204        220
       92 Carnian                         6        220        230
       93 Ladinian                        6        230        235
       94 Anisian                         6        235        242
       95 Scythian                        6        242        250
      260 Spathian                        6        242        244
      261 Nammalian                       6        244        247
      262 Griesbachian                    6        247        250

13 rows selected.
```

The STRATA.GEOTIME table is defined as follows :

```
create table geotime (
    ageno      number    (4,0)  not null,
    agename    char      (24)   not null,
    scope      number    (2,0)  not null,
    rank       number    (1,0)  not null,
    status     number    (1,0)  not null,
    parent     number    (4,0),
    yngbound   number    (8,3),
    oldbound   number    (8,3),
    comments   char      (64),
    geodxid    number    (5,0),
    lastalt    date );
```