8837-EN-01
DTIC

Final Report
(Interim Period 9 to 12)

submitted to
European Research Office
USARDSG-UK
Edison House
223 Old Marylebone Road
London, NW1 5TH
England

31 May 2000

Modeling Transport Phenomena within the FSU Information Analysis System

R&D 8837-EN-01
Contract N68171-99-C-9027

Principal investigator:
Dr. J.M.F. Masuch

Institution:
Applied Logic Laboratory (ALL)
University Foundation
Amsterdam

Plantagekade 57
1018 ZV Amsterdam
the Netherlands
tel: # .31.20.422.97.67
fax: # .31.20.422.97.68

Modeling Transport Phenomena within the FSU Information Analysis System
Final Report

R&D 8837-EN-01

1. Title

Modeling Transport Phenomena within the FSU Information Analysis System: Final Report (Interim Period 9 to 12)

2. Abstract

Within this project report, we describe the technical methods for organizing, documenting, and archiving an Oracle 8.i Relational Database Management System (RDBMS). These procedures are required by the Ministry of Defense (MOD), Scientific Center for the Safety and Security of Technical Means (SCSSTM) in St. Petersburg, Russia for emergency operations within the Information Analysis System (IAS) and the emerging Center for Technical Diagnostics (CTD). Within the IAS system, the Oracle procedures are required to organize geo-spatial data for analysis and display by five major field teams. Within the CTD, the Oracle algorithms are required to organize destructive and non- destructive materials data for analysis by technical diagnostics teams. This report builds upon a series of course work developed for the MOD by ALL/ERO presented in St. Petersburg during 31 January to 4 February 2000 by Mr. Alexander Starshinin (IT/Peter-Service - St. Petersburg) and expanded by Dr. Alexandr Ptitzyn (IT- Moscow) on 16-18 February 2000. Further documentation (technical abstracts) were provided to MOD during the period 22-23 February 2000 as a component of the basic laboratory support for the IAS.

The course curriculum was designed to assist MOD in their use of RDBMS within a highly distributed field based system. Since one or more server may be accessed by a mobile team, the Database Administrator (DBA) is commonly required to organize and distribute information in an effective manner. This is the occurrence during emergency operations. When accessing records from a *blind* server, a common problem is the organization of class data, and the recreation (or modification) of the database access tables. If export and import can be used, this task is fairly easy to accomplish; however, sometimes the DBA is required to provide DDL (Data Definition Language) for this purpose. The DBA may wish to document existing procedures, views, constraints and such other structures as they see fit, with human readable (easily understood) output. The standard Oracle 8.i techniques fall short of the mark in providing the DBA with documentation, so alternative techniques are required. Hence, the DBA must develop SQL, PL/SQL and SQLPLUS code to delve into the inner workings of the Oracle Data Dictionary tables and re-generate the required DDL.

In this final report, an integrated methodology for documenting an existing Oracle 8.i *Instance* is provided. In this terminology, the word *Instance* is used to define the organization and operating state of the database (IAS or CTD). By using these steps a DBA can create a series of scripts that allow documentation and, if required, recreation

of an Oracle 8.i RDBMS under emergency operations. This research demonstrates techniques that can be used in creating these scripts. The source code for the database object information is provided.

This research is the final component within this BAA project. The documentation is required for the future development of the CTD laboratory information management systems (LIMs). Since these new systems may interface with the current IAS Oracle RDBMS, considerable attention is required to ensure the proper exchange of data structures (GIS versus materials testing) and related LIMs measurements. This report describes the data dictionary (lookup) components that will be required for efficient operations, and examines the basic methods for controlling two or more systems (log documentation). It is understood that one system may reside within a single local area network (the IAS) and the second system may be used within an entirely separate network with an independent operating environment (the CTD). It is also understood, that these systems may be physically disconnected from one another due to security limitations within the SCSSTM. For this reason, a common data dictionary must be developed that will be used by each independent network. Once this common structure is established, data can be transferred on physical media (tape, cartridge, etc.) without loss of attribute information. This research, and future efforts, will ensure a safe and effective transfer of topographic data between the IAS and CTD for MOD mission requirements.

3. Contents

4. Introduction

The Oracle Data Dictionary is a collection of C language constructs, Oracle tables and Oracle views. At the lowest level are the "hidden" C structures known as the X$ tables. These X$ tables are usually best left alone. Indeed, to even see the contents a DBA has to jump through a few hoops and once they do get to them, they are not well documented and have such logical attributes as "mglwmp" or "tabsrpr". Needless to say, unless the DBA has several long nights available and a penchant for solving riddles it is suggested that they leave these to Oracle.

The next layer of the Oracle Data Dictionary is the $ tables. These are more human readable cuts of the X$ tables and have such names as COL$ or TAB$. While being a step above the X$ structures, it is still suggested that the DBA only use them if they are really needed.

The third layer of the dictionary is the V$ views and their cousins the V_$ tables (actually, Siamese twins since for nearly all of the V$ views there is a corresponding V_$ view). These are the workhorses of the Data Dictionary and what most of your scripts should deal with.

The final layer of the dictionary (for our purposes) is the DBA_ series of views. These views are made from the V$ and $ sets of views and tables and provide a very friendly interface for viewing the insides of your Oracle Data Dictionary. The USER_ and ALL_ views are based on the DBA_ views.

A good set of build scripts will provide excellent documentation for such an instance. Essentially there are two types of documentation, "hard" documentation such as DDL that an experienced database developer can understand, and "soft" documentation such as generalized reports showing structures and relationships that anyone with a smattering of database experience can use. This report will deal more with creating "hard" documentation, that is, the scripts to rebuild the database.

5. Organization of the RDBMS

Within Oracle 8.i the database contain the following items:

Hard Objects	Stored Objects
Tables, places and their associated Data files	View definitions
Control Files	Constraints
Redo Logs	Procedures
Rollback Segments	Functions
Tables	Packages
Indexes	Package Bodies
Database Initialization file	Triggers
Clusters	User definitions
	Roles
	Grants
	Database Links
	Snapshots and Snapshot Logs

In the following pages this report will cover how to generate the required DDL to recreate or document each of the above database objects using SQL, PL/SQL and SQLPLUS.

5.1 Control Files

Initially, we document the Control file since this is the only location where the information for MAXDATAFILES, MAXLOGFILES MAXMEMBERS and other instance specific data can be found.

Having a script around that rebuilds the control file is vital to emergency operations. In the context where the Oracle instance was built, with very little understanding of how Oracle builds their default instances, a highly dysfunctional database is created. This is particularly apparent when using the default Oracle procedures for the *vanilla* build. The DBA will find their MAXDATAFILES will be set to 20 or maybe 32 and that other hard database limits are probably set too low. Once the DBA generates a script to rebuild their control files, they can change these hard database limits without rebuilding the database. This is accomplished by simply rebuilding the control files. In other situations, a disk crash can wipe out the control file (hopefully there is more than

one, but maybe a DBA didn't create the database?), and leave the database unable to start. With a rebuild script on hand, the DBA can operate the RDBMS without significant loss of data or operating function.

A DBA can backup the control file with the command:

ALTER DATABASE BACKUP CONTROLFILE to 'filename' REUSE;

However, the above command only makes a machine-readable copy (which is an excellent idea for when changes are made to structures). A DBA should usually backup the control file with each set of backups. Of course, this doesn't provide documentation of the control file.

Since there are no tables which document the control files (other than v$parameter which only documents their location) the DBA must depend on a system command to provide them with the required script:

```
ALTER DATABASE
BACKUP CONTROLFILE TO TRACE NORESTLOGS;
```

The output from this script is the following:

```
*********************************************************************

Dump file H:\ORAWIN\RDBMS81\trace\ORA14071.TRC
Thur Mar 30 10:05:53 2000
vsnsta=0
vsnsql=a vsnxtr=3
MS-WINDOWS
Thur Mar 30 10:05:52 2000
Thur Mar 30 10:05:53 2000

*** SESSION ID:(5.3)
# The following commands will create a new control file and use it
# to open the database.
# No data other than log history will be lost. Additional logs may
# be required for media recovery of offline data files. Use this
# only if the current version of all online logs are available.
STARTUP NOMOUNT
CREATE CONTROLFILE REUSE DATABASE "ORACLE" NORESETLOGS
NOARCHIVELOG
    MAXLOGFILES 32
```

```
    MAXLOGMEMBERS 2
    MAXDATAFILES 32
    MAXINSTANCES 16
    MAXLOGHISTORY 1600
  LOGFILE
   GROUP 1 'H:\ORAWIN\DBS\wdblog1.ora'  SIZE 500K,
   GROUP 2 'H:\ORAWIN\DBS\wdblog2.ora'  SIZE 500K
  DATAFILE
   'H:\ORAWIN\DBS\wdbsys.ora' SIZE 10M,
   'H:\ORAWIN\DBS\wdbuser.ora' SIZE 3M,
   'H:\ORAWIN\DBS\wdbrbs.ora' SIZE 3M,
   'H:\ORAWIN\DBS\wdbtemp.ora' SIZE 2M
  ;

  # Recovery is required if any of the datafiles are restored backups,
  # or if the last shutdown was not normal or immediate.
  RECOVER DATABASE

  # Database can now be opened normally.
  ALTER DATABASE OPEN;

*******************************************************************
```

As shown within this output, all the required "hard" database setpoints are now documented. The output from the command is located wherever the system places its system level trace files (check v$parameter on a wildcard name search for '%dump%').

5.2 Documenting the Database Initialization

The initialization file, commonly called INIT.ORA or initSID.ora where the SID is the database instance name, is another key file for the database documentation set. Sadly, the initialization file is often overlooked. All of the values used in the initialization process are stored in the V$PARAMETER table. A fairly simple SQLPLUS script can be used to document these values.

Once the control file settings and the initialization file settings are documented, it is time to document the database itself. In this instance, MOD must create a script that will generate a "bare bones" CREATE DATABASE command. The script should have the MAX set of parameters from the CREATE CONTROLFILE command edited into it between the DATAFILE and NOARCHIVELOG (or ARCHIVELOG) clauses. The tables used to document the

database for a CREATE DATABASE command would consist of DBA_DATA_FILES for the location of the SYSTEM tablespace datafiles. In addition, the V$LOGFILE for the locations and sizes of the redo log files and the V$LOG view for the structure of redo logs, groups and threads.

Essentially, the script for documenting the database creation procedure would define cursors for the database datafiles, the redo log related items and the redo groups. The scripts would then use loop constructs to populate a temporary table with the required command syntax and once complete, a simple selection from this temporary table builds your command file. Since this CREATE DATABASE process uses all of the techniques that can be used to document other database structures, the script used to generate the CREATE DATABASE command is a good example to show how this technique is applied:

```
**************************************************************

REM FUNCTION: SCRIPT FOR CREATING DB
REM
REM       This script must be run by a user with the DBA role.
REM
REM       This script is intended to run with Oracle8.
REM
REM       Running this script will in turn create a script to
REM       rebuild the database. This created
REM       script, crt_db.sql,  is run by SQLDBA
REM
REM       Only preliminary testing of this script was performed.
REM       Be sure to test it completely before relying on it.
REM
REM

set verify off termout off feedback off echo off pages 0
set termout on
prompt Creating db build script...
set termout off;

create table db_temp
   (lineno NUMBER,  text VARCHAR2(255))
/
DECLARE
  CURSOR dbf_cursor IS
     select  file_name,bytes
     from  dba_data_files
     where tablespace_name='SYSTEM';
  CURSOR mem_cursor (grp_num number) IS
     select   member
```

```
    from    v$logfile
    where   group#=grp_num
    order by member;
  CURSOR thread_cursor IS
    select thread#, group#
    from v$log
    order by thread#;

 grp_member        v$logfile.member%TYPE;
 db_name           varchar2(8);
 db_string     VARCHAR2(255);
 db_lineno     number := 0;
 thrd        number;
 grp         number;
 filename          dba_data_files.file_name%TYPE;
 sz              number;
 begin_count  number;

 procedure write_out(p_line INTEGER,
              p_string VARCHAR2) is
     begin
       insert into db_temp (lineno,text)
             values (db_lineno,db_string);
       end;

BEGIN
     db_lineno:=db_lineno+1;
  SELECT 'CREATE DATABASE '||value into db_string
     FROM v$parameter where name='db_name';
    write_out(db_lineno,db_string);
     db_lineno:=db_lineno+1;
  SELECT 'CONTROLFILE REUSE' into db_string
     FROM dual;
    write_out(db_lineno,db_string);
     db_lineno:=db_lineno+1;
  SELECT 'LOGFILE (' into db_string
     FROM dual;
    write_out(db_lineno,db_string);
commit;
if thread_cursor%ISOPEN then
      close thread_cursor;
      open thread_cursor;
else
      open thread_cursor;
end if;
loop
      fetch thread_cursor into thrd,grp;
      exit when thread_cursor%NOTFOUND;
      db_lineno:=db_lineno+1;
      db_string:= 'THREAD '||thrd||' GROUP '||grp||' (';
```

```
        write_out(db_lineno,db_string);
        if mem_cursor%ISOPEN then
            close mem_cursor;
            open mem_cursor(grp);
        else
            OPEN mem_cursor(grp);
        end if;
        db_lineno:=db_lineno+1;
        begin_count:=db_lineno;
        loop
            fetch mem_cursor into grp_member;
            exit when mem_cursor%NOTFOUND;
            if begin_count=db_lineno then
                db_string:=''||grp_member||'';
                write_out(db_lineno,db_string);
                db_lineno:=db_lineno+1;
            else
                db_string:=','||'''||grp_member||'';
                write_out(db_lineno,db_string);
                db_lineno:=db_lineno+1;
            end if;
        end loop;
        db_lineno:=db_lineno+1;
        db_string:=' )';
        write_out(db_lineno,db_string);
    end loop;
    db_lineno:=db_lineno+1;
    SELECT ')' into db_string from dual;
    write_out(db_lineno,db_string);
    commit;
    if dbf_cursor%ISOPEN then
        close dbf_cursor;
        open dbf_cursor;
    else
        open dbf_cursor;
    end if;
    begin_count:=db_lineno;
    loop
        fetch dbf_cursor into filename, sz;
        exit when dbf_cursor%NOTFOUND;
        if begin_count=db_lineno then
            db_string:='DATAFILE '||''''||filename||''''||' SIZE '||sz||' REUSE';
        else
            db_string:=','||''''||filename||''''||' SIZE '||sz||' REUSE';
        end if;
        db_lineno:=db_lineno+1;
        write_out(db_lineno,db_string);
    end loop;
    commit;
    SELECT decode(value,'TRUE','ARCHIVELOG','FALSE','NOARCHIVELOG')
```

```
        into db_string from v$parameter where name='log_archive_start';
        db_lineno:=db_lineno+1;
        write_out(db_lineno,db_string);
    SELECT ';' into db_string from  dual;
        db_lineno:=db_lineno+1;
        write_out(db_lineno,db_string);
    CLOSE dbf_cursor;
    CLOSE mem_cursor;
    CLOSE thread_cursor;
    commit;
    END;
    /
    column dbname new_value db noprint
    select value dbname from v$parameter where name='db_name';
    set heading off pages 0 verify off
    set recsep off
    spool rep_out\&db\crt_db.sql
    col text format a80 word_wrap
    select   text
    from    db_temp
    order by  lineno;
    spool off
    set feedback on verify on termout on
    drop table db_temp;
    prompt Press enter to exit
    exit
```

****

The reason this script is so long is that there may be multiple datafiles for the SYSTEM tablespace, and there most certainly will be multiple redo logs, redo log groups and possibly redo log threads. This multiplicity of files results in the need for a cursor for each of the possible recursions and a loop-end loop construct to support the selection of the data from the database.

It is ironic that all of the above code is used to produce the following output:

****

```
CREATE DATABASE ORCSPCD1
CONTROLFILE REUSE
LOGFILE (
THREAD 1 GROUP 1 (
'/vol2/oracle1/ORCSPCD1/log1ORCSPCD1.dbf'
)
```

```
THREAD 1 GROUP 2 (
'/vol2/oracle2/ORCSPCD1/log2ORCSPCD1.dbf'
)
THREAD 1 GROUP 3 (
'/vol3/oracle3/ORCSPCD1/log3ORCSPCD1.dbf'
)
)
DATAFILE '/vol2/oracle1/ORCSPCD1/systORCSPCD1.dbf' SIZE 41943040 REUSE
NOARCHIVELOG
;
```

****

As previously stated, the script should have the MAX set of parameters from the CREATE CONTROLFILE command edited into it between the DATAFILE and NOARCHIVELOG (or ARCHIVELOG) clauses. One should also consider creating soft documentation for database related items. This would consist of reports on the redo logs, system parameters, and datafiles for the SYSTEM tablespace.

5.3 Documenting Tablespaces and Rollback Segments

Tablespaces and their underlying datafiles should be created immediately after the database itself. A script should be written using information in DBA_TABLESPACES and DBA_DATA_FILES to create the DDL required to make an exact duplicate of a system's existing tablespace/datafile profile. Note that each tablespace may have a number of datafiles associated with it, so PL/SQL supporting this recursive relationship will be required.

Before anything else in the database can be created, the rollback segments have to be built. At this point, the DBA must have a functioning database, a copy of the respective tablespaces, but only the default SYSTEM rollback segment.

The next step is to create a script to generate DDL to make an exact copy of the existing rollback segment profile using information in the DBA_ROLLBACK_SEGS and V_$ROLLSTAT views.

The script should document the rollback segments as they currently exist. The DBA can then edit the scripts to make any improvements that are required or desired before executing it on the new database.

5.4 Documenting Roles, Grants and Users

Before database objects such as tables, indexes, constraints and the like can be created, there must be users (or schemas) with the appropriate roles and grants to create and own them. The next scripts to be created should generate DDL to create users, roles and grants. Oracle has spread this data to numerous tables. Fortunately most have the same structure and can be queried using a simple query structure. The information can be obtained from the following tables:

DBA_TS_QUOTAS,
DBA_USERS,
DBA_COL_PRIVS,
DBA_ROLE_PRIVS,
DBA_ROLES,
DBA_TAB_PRIVS and
DBA_SYS_PRIVS.

Once you have fully documented all users, roles and grants, your next set of scripts should provide documentation for the tables and their associated indexes, constraints, procedures, triggers and views.

5.5 Documenting Tables

Information for documenting application tables can be found in the DBA_TABLES and DBA_TAB_COLUMNS views. MOD must use PL/SQL so that the recursive nature of the relationship between tables and table columns can be easily resolved using cursors and loop constructs.

It is recommended that the DBA should not capture table constraints at this point. The reason for this is to allow data loading to occur before the constraints are enabled. A separate script should be written to re-create all primary key, unique, foreign key, non-"not null" check type and default value constraints.

5.6 Documenting Database Constraints

In many cases the constraints have been specified during the CREATE TABLE command, but are not named within Oracle 8.i. This lack of naming for a

constraint will result in Oracle selecting a name consisting of SYS_C with a sequence number appended to it, such as SYS_C00123. Using PL/SQL the DBA can re-build, and, at the same time, re-name these constraints.

A simple naming convention for constraints (and their associated indexes) would be to provide a suffix (PK for primary key, UK for unique, FK for foreign key and CK for check or default values). The naming convention should be followed by the table name, and for those constraints for which there can be more than one per table, a sequential number. For example, FK_ACTCNTL_1 would be the first foreign key constraint on table ACTCNTL.

Constraints come in several forms, primary key (P), foreign key (R), unique (U), check (C) and default value (V).

Check constraints can be the standard NOT-NULL type constraints or can be a full-blown verification algorithm. Default value constraints provide a default value for those fields effected by RDBMS. Since all of the constraints with the exception of check and default values constraints can be associated with one or more columns in a table, PL/SQL is again indicated for the script to rebuild (and possibly rename) the database constraints.

Information needed to document your table constraints can be found in the DBA_INDEXES (data such as index tablespace, index storage parameters, index name). The DBA_CONS_COLUMNS (columns involved with the constraint), and DBA_CONSTRAINTS (showing constraints and their related primary constraints if they are a foreign key, and all other information for other constraints) views are also informative for context documentation.

5.7 Documenting Indexes and Sequences in the Database

Indexes are usually created for one of four purposes:

(i) Enforcing a primary key,
(ii) Enforcing a unique value,
(iii) Lookup support on a foreign key, or
(iv) Performance enhancement.

Your constraints rebuilding script should handle primary key, unique and foreign key type indexes. Your next script should allow you to rebuild all

indexes in a database. It should pull information from the DBA_IND_COLUMNS and DBA_INDEXES views.

Since constraints will create indexes for their support (at least for primary and unique keys), the DBA should create constraints before running any index build scripts that are strictly look-up in nature. This will ensure that the naming convention for indexes is enforced. Oracle will only allow one index per a specified set of column names in a specific table. If a constraint enforcement index already exists, it will not be rebuilt and renamed by another script.

Since sequences are not a component of standard SQL, many analysts tend to overlook their function, until they break. Sequences can break if you exceed their maximum (for an ascending sequence) or minimum (for a descending sequence) values or place too much stress on a single sequence. By documenting your sequences you can see how they were created and have a script to recreate them if it becomes needed. In some situations, such as import or use of SQL*Loader, sequences used for key values can become out of sync with the tables they relate to and will require resetting. Use of a documentation script can speed this process. A SQLPLUS or PL/SQL script can be created to generate the DDL to rebuild sequences using information from DBA_SEQUENCES.

6. Documenting Packages Bodies, Procedures and Functions

Under Oracle 8.i, the PL/SQL constructs can be stored in the database as imbedded objects. The lowest level for any object is a procedure or a function. The procedures and functions in an Oracle database can be grouped by application or related function into packages (and associated package bodies). All of the DDL to define PL/SQL stored objects such as packages, package bodies, procedures and functions is stored in the DBA_SOURCE view and additional information on them is stored in the DBA_OBJECTS view. By creating a script which joins these two tables you can re-create any of the objects mentioned above.

To demonstrate how convoluted not using PL/SQL in recursive type relationships can become, the process of documenting package bodies, procedures and functions is shown below using standard SQL and SQLPLUS scripts.

The first script is the procedure that does the actual work of selecting the text from the database and reconstructing the words into a command. It is called fprc_rct.sql:

```
**************************************************************

REM
REM NAME:    FPRC_RPT.SQL
REM
REM FUNCTION: Build a script to re-create functions, procedures,
REM             packages or package bodies.
REM
REM
set termout off verify off feedback off lines 132 pages 0 heading off
set recsep off space 0
column text format a79
column line noprint
select 'create or replace '||text,line
from
      dba_source
where
      owner = upper('&&1') and
      type = upper('&&2') and
      name = upper('&&3') and
      line = 1;
select text,line
from
      dba_OBJECTS s1,
      dba_source s2
where
      s1.OBJECT_type = upper('&&2') and
      s1.owner = upper('&&1') and
      s1.object_name = upper('&&3') and
      s1.OBJECT_type = s2.type and
      s1.owner = s2.owner and
      s1.OBJECT_NAME = s2.name and
      line > 1
order by 2;

**************************************************************
```

This script can be run in a standalone mode by calling it with the values for owner, object type and object name. However, it was designed to be called by a reiterative program called do_fprc.sql which is created by a script run_fprc.sql, shown below:

```
**************************************************************
```

```
REM
REM NAME            : RUN_FPRC.SQL
REM FUNCTION        : Generate and execute the crt_fprc.sql procedure
REM USE             : Document the procedures and packages and functions
REM                 for a user or users
REM Limitations     : Must have access to dba_source and dba_objects.
REM                 The FPRC_RCT.SQL procedure must be in same directory
REM
column dbname new_value db noprint
pause Use % for a wildcard - Press enter to continue
accept owner prompt 'Enter object owner:'
accept type  prompt 'Enter object type:'
accept name  prompt 'Enter object name:'
prompt Working....
set echo off heading off verify off feedback off
select value dbname from v$parameter where name='db_name';
spool rep_out\&db\do_fprc.sql
select
      unique('start fprc_rct '||owner||''''||''''||type||''''||''''||name)
from   dba_source
where
      owner like upper('&owner') and
      type like upper('&type') and
      name like upper('&name');
spool off
set termout off
spool rep_out\&db\crt_fprc.sql
start rep_out\&db\do_fprc.sql
spool off
exit
```

****

This script generates a sequence of calls to fprc_rct.sql:

****

```
start fprc_rct CSPCDBA "PROCEDURE" ADDRMANOTE
start fprc_rct CSPCDBA "PROCEDURE" ENTERRULES
start fprc_rct CSPCDBA "PROCEDURE" ESCALATE
start fprc_rct CSPCDBA "PROCEDURE" GETDETAILDATA
start fprc_rct CSPCDBA "PROCEDURE" GETLEAST
start fprc_rct CSPCDBA "PROCEDURE" GETMAXACTSEQNBR
start fprc_rct CSPCDBA "PROCEDURE" GETSUMMARYHEADERS
start fprc_rct CSPCDBA "PROCEDURE" GET_PAWS0001
start fprc_rct CSPCDBA "PROCEDURE" INSADMINAREA
start fprc_rct CSPCDBA "PROCEDURE" INSAFFLNROL
```

```
start fprc_rct CSPCDBA "PROCEDURE" INSAUTSYSAREA
start fprc_rct CSPCDBA "PROCEDURE" INSCO
start fprc_rct CSPCDBA "PROCEDURE" INSCOORG
start fprc_rct CSPCDBA "PROCEDURE" INSCOORGAREA
start fprc_rct CSPCDBA "PROCEDURE" INSCOORGAUTH
start fprc_rct CSPCDBA "PROCEDURE" INSEMP
start fprc_rct CSPCDBA "PROCEDURE" INSERTTMPTABLE
start fprc_rct CSPCDBA "PROCEDURE" INSINDIV
start fprc_rct CSPCDBA "PROCEDURE" INSINDIVAUTH
start fprc_rct CSPCDBA "PROCEDURE" INSORGEMP
start fprc_rct CSPCDBA "PROCEDURE" INSSUM
start fprc_rct CSPCDBA "PROCEDURE" PAWS1
start fprc_rct CSPCDBA "PROCEDURE" POSTERROR
```

The script run_fprc.sql then executes the do_fprc.sql script spooling output to the crt_fprc.sql script. The crt_fprc.sql script contains the commands to rebuild the specified object or objects:

```
**************************************************************

  create or replace PROCEDURE AddRmaNote (
       hv_eventid      IN event.evntid%TYPE,
       hv_actseqnbr IN act.actseqnbr%TYPE,
       hv_user  IN INDIV.CUIDNBR%TYPE,
       hv_notes        IN VARCHAR2,
       hv_status       IN OUT NUMBER
       )
  AS   leid     NUMBER(10);
       date1   DATE;
  BEGIN
       hv_status := 0;
       leid := 0;
       BEGIN
       SELECT FK_LELEID INTO leid FROM INDIV WHERE ( CUIDNBR = hv_user );
       EXCEPTION WHEN OTHERS THEN
         hv_status := 1;
       END;
       SELECT SYSDATE INTO date1 FROM dual;
       IF ( leid > 0 ) THEN
       INSERT INTO actvy_rmk ( ACTVYRMKDT, ACTVYRMKTXT,
FK_ACTFK_EVENTEVNT,
            FK_ACTACTSEQNBR, FK_INDIVFK_LELEID )
       VALUES ( date1, hv_notes, hv_eventid, hv_actseqnbr, leid );
       END IF;
       COMMIT;
  END AddRmaNote;
  /
  ...
  create or replace PROCEDURE wkgrp_usr (
```

```
    hv_wrkgrp     IN COORG.COORGCD%TYPE,
    hv_grptyp     IN COORG.COORGTYPCD%TYPE,
    hv_status     IN OUT NUMBER
    )
AS
  CURSOR wkgrpusr_cur is
    SELECT indiv.CUIDNBR, indiv.INDIVLASTNM, indiv.INDIVFRSTNM
  FROM orgemp, emp, afflnrol, le, indiv
  WHERE (
    indiv.FK_LELEID = le.leid AND
    le.letypcd = 'EMP' AND
    afflnrol.FK_LELEID = le.leid AND
    emp.FK_AFFLNROLAFFLNRO = afflnrol.AFFLNROLCD AND
    emp.FK_AFFLNROLFK_AFFL = afflnrol.FK_AFFLNAFFLNID AND
    emp.FK_AFFLNROLFK_LELE = afflnrol.FK_LELEID AND
    orgemp.FK_EMPEMPSSN = emp.EMPSSN AND
    orgemp.FK_COORGCOORGTYPCD = hv_grptyp AND
    orgemp.FK_COORGCOORGCD = hv_wrkgrp
    );
  id  indiv.CUIDNBR%TYPE;
  fnm indiv.INDIVFRSTNM%TYPE;
  lnm indiv.INDIVLASTNM%TYPE;
BEGIN
  hv_status := 0;
  BEGIN
  OPEN wkgrpusr_cur;
  FETCH wkgrpusr_cur INTO id, lnm, fnm;
  WHILE NOT wkgrpusr_cur%NOTFOUND LOOP
  INSERT INTO user_table (cuid,lastnm,frstnm) VALUES (id,lnm,fnm );
  FETCH wkgrpusr_cur INTO id, lnm, fnm;
  END LOOP;
  CLOSE wkgrpusr_cur;
  END;
  COMMIT;
END wkgrp_usr;

**********************************************************************
```

The crt_fprc.sql script, with minor editing if the developers tended to string commands over several lines, can then be executed to rebuild whichever object or objects it was invoked to recreate. If the DBA edits the crt_fprc.sql (the code for the packages) the package bodies, procedures or functions can be modified to be more readable and to adhere to any coding standards. In laboratories where many developers may be working on a single project (SCSSTM - St. Petersburg), the diversity of coding techniques can be astounding. This allows the DBA to enforce some semblance of order on the code in the database.

6.1 Documenting Triggers and Views

Triggers are used to enforce referential integrity constraints, enforce snapshot logic, provide updates on calculated table values, and other database functions. The triggers required for constraint enforcement, and snapshot upkeep, are automatically generated by the Oracle kernel. It is suggested that all triggers created by non-Oracle processes be preceded by some unique set of characters. In this manner, the script can be made more selective (to ignore the Oracle created triggers).

The DBA should consider using the CREATE command rather than the CREATE OR REPLACE command in the program to provide some measure of safety that existing triggers (created automatically by Oracle) will not be harmed by running the script. Information for documenting triggers can be found in the DBA_TRIGGERS and DBA_TRIGGER_COLS views.

The final set of stored objects that should be documented are database views. Views are used to pre-join tables, provide security or make it easier for non-SQL literate users to get at complex data. Often times, a developer may create a view and then forget to document it. Information for documenting views can be found in the DBA_VIEWS. You may need to set the value for longs and adjust your session linesize parameters to get output that is readable. You can query the DBA_VIEWS text_length column for the max value: "select max(text_length) from dba_views;" and set your long value accordingly with the SQLPLUS SET command.

6.2 Snap Shot and Snap Shot Log Documentation

The DBA may create report scripts that will document snapshots and snapshot logs in the instance using the DBA_SNAPSHOTS and DBA_SNAPSHOT_LOGS procedure. Using PL/SQL scripts to rebuild existing snapshots and create new snapshot logs. However, since these objects seem to change with each new release it may be difficult to keep up with Oracle's improvements. With new features such as refresh groups, and the entire new set of replication options, it may be wise to hold off efforts at generating documentation until Oracle finalizes all they wish to do with these objects.

7. Process Termination

At some point an Oracle administrator will be forced to terminate non-essential Oracle sessions, or will have to conduct a shutdown regardless of database activity. This is known as killing an Oracle session. Non-essential Oracle sessions typically consist of active terminal sessions left connected by end users. These active sessions cause problems when the database is shutdown for backup or maintenance operations. If there is an active session, a normal mode shutdown will hang waiting for the session to end.

Killing an Oracle session can be accomplished via one of three methods:

(i) Using the shutdown command with the abort option;
(ii) Using the ALTER SYSTEM kill option; or
(iii) Using an operating system process killer.

Once a session is killed, its status in the V$SESSION view goes to "KILLED", and the user receives a "session killed" message if they try to reactivate that session. The session entry in the V$SESSION view will not be physically removed until the user physically disconnects.

Oracle provides the shutdown command with the abort option, which will shutdown the database. However, the abort option does not shutdown the session gracefully. After using the abort option, you have to re-start the session and perform a normal shutdown before conducting a backup operation[1].

Besides the shutdown command with the abort option, you can either issue the ALTER SYSTEM KILL SESSION command, or you can issue an operating system process kill command such as the UNIX 'kill -9 pid'. You should not do both.

If you kill both the Oracle process and the operating system process, you could cause a database hang forcing you to perform a shutdown abort.

---

[1] Shutdown immediate and shutdown normal are supposed to be able to handle killed sessions properly but there have been reports of problems up to version 8.i, on some hardware platforms.

7.1 Killing an Oracle Session from the Oracle Side

You can either issue a series of ALTER SYSTEM commands manually, or use a dynamic SQL script to perform the operation. A PL/SQL procedure is shown below that performs a kill of a process using the dynamic SQL package; DBMS_SQL:

```
**********************************************************************

    create or replace procedure kill_session ( session_id in varchar2,
    serial_num in varchar2)
    as
    cur integer;
    ret integer;
    string varchar2(100);
    begin
     string :=
          'alter system kill session '||''''||session_id||','||serial_num||'''';
      cur := dbms_sql.open_cursor;
      dbms_sql.parse(cur,string,dbms_sql.v7);
      ret := dbms_sql.execute(cur) ;
      dbms_sql.close_cursor(cur);
    exception
      when OTHERS then
        raise_application_error(-20001,'Error in execution',TRUE);
        if dbms_sql.is_open(cur) then
          dbms_sql.close_cursor(cur);
        end if;
    end;
    /

**********************************************************************
```

Using the above procedure, the DBA can then create a quick SQL procedure to remove the non-required Oracle sessions from the Oracle RDBMS side. An example of this procedure is shown below:

```
**********************************************************************

    REM
    REM ORA_KILL.SQL
    REM FUNCTION   : Kills non-essential Oracle sessions (those that aren't owned
    REM            : by SYS or "NULL"
```

```
REM DEPENDANCIES: Depends on kill_session procedure
REM MRA 3/30/00
REM
set heading off termout off verify off echo off
spool kill_all.sql
select 'execute kill_session('||chr(39)||sid||chr(39)||','||
chr(39)||serial#||chr(39)||');' from v$session
where username is not null
or username <> 'SYS'
/
spool off
start kill_all.sql
exit
```

****

An example of the output from ora_kill.sql (kill_all.sql) is:

****

```
execute kill_session('10','212');
execute kill_session('13','1424');
```

****

7.2 Killing an Oracle Session From the UNIX Operating System

A second method of removing unwanted sessions is to kill them from the operating system side. In UNIX environments this is accomplished with the "kill -9' command executed from a privileged user (see the Solaris 7 OS for similar commands).

A sample command script (shell script) that will remove the non-essential Oracle sessions for all currently active Oracle databases on the UNIX server is shown below:

****

```
#!/bin/ksh
ORATAB=/etc/oratab
trap 'exit' 1 2 3
# Set path if path not set (if called from /etc/rc)
```

```
case $PATH in
  "") PATH=/bin:/usr/bin:/etc
    export PATH ;;
esac
rm kill.lis
rm proc.lis
touch kill.lis
touch proc.lis
#
# Loop for every entry in oratab
#
cat $ORATAB | while read LINE
do
  case $LINE in
    \#*)        ;;    #comment-line in oratab
    *)
    ORACLE_SID=`echo $LINE | awk -F: '{print $1}' -`
  if [ "$ORACLE_SID" = '*' ] ; then
        ORACLE_SID=""
  fi
     esac
     if [ "$ORACLE_SID" <> '*' ] ; then
          proc_name='oracle'$ORACLE_SID
          ps -ef | grep $proc_name>>proc.lis
     fi
done
cat proc.lis | while read LINE2
do
     command=`echo $LINE2 | awk -F: 'BEGIN { FS = ",[ \t]*|[ \t]+" }
                              { print $2}' -`
     test_it=`echo $LINE2 | awk -F: 'BEGIN { FS = ",[ \t]*|[ \t]+" }
                              { print $8}' -`
     if [ "$test_it" <> 'grep' ] ; then
          command='kill -9 '$command
          echo $command>>kill.lis
     fi
done
rm proc.lis
chmod 755 kill.lis
kill.lis
rm kill.lis
```

****

The ora_kill.sh script utilizes a technique used in the dbshut and dbstart shell scripts. This technique uses the /etc/oratab file to determine what databases should be operating. An alternative to using the oratab file would be to do "ps -ef | grep smon", redirecting output into a file, and using awk to strip out the SID

names (similar to the technique used below). Each operating instance will have one "smon" process, so this makes a logical string value to grep out of the "ps -ef" process list.

Killing the sessions from the operating system side will remove their entries from the V$SESSION view. An example of the output from ora_kill.sql (kill.lis) is:

```
**********************************************************************

    kill -9 11240
    kill -9 11244
    kill -9 11248
    kill -9 11252
    kill -9 11256
    kill -9 9023
    kill -9 9025
    kill -9 9028
    kill -9 9030

**********************************************************************
```

8. Training

During the period 25 January to 4 February 2000 comprehensive training was conducting in ESRI GIS systems and Oracle RDBMS systems. The Oracle instruction continued from 16 to 18 February 2000 conducted by Dr. Alexandr Ptitzyn (IT Corporation Moscow) and Mr. Alexandr Starshinin (TeleconInvest-PS, LTd - St. Petersburg). In this section, we provide sample images that describe the training process with the principal MOD students.

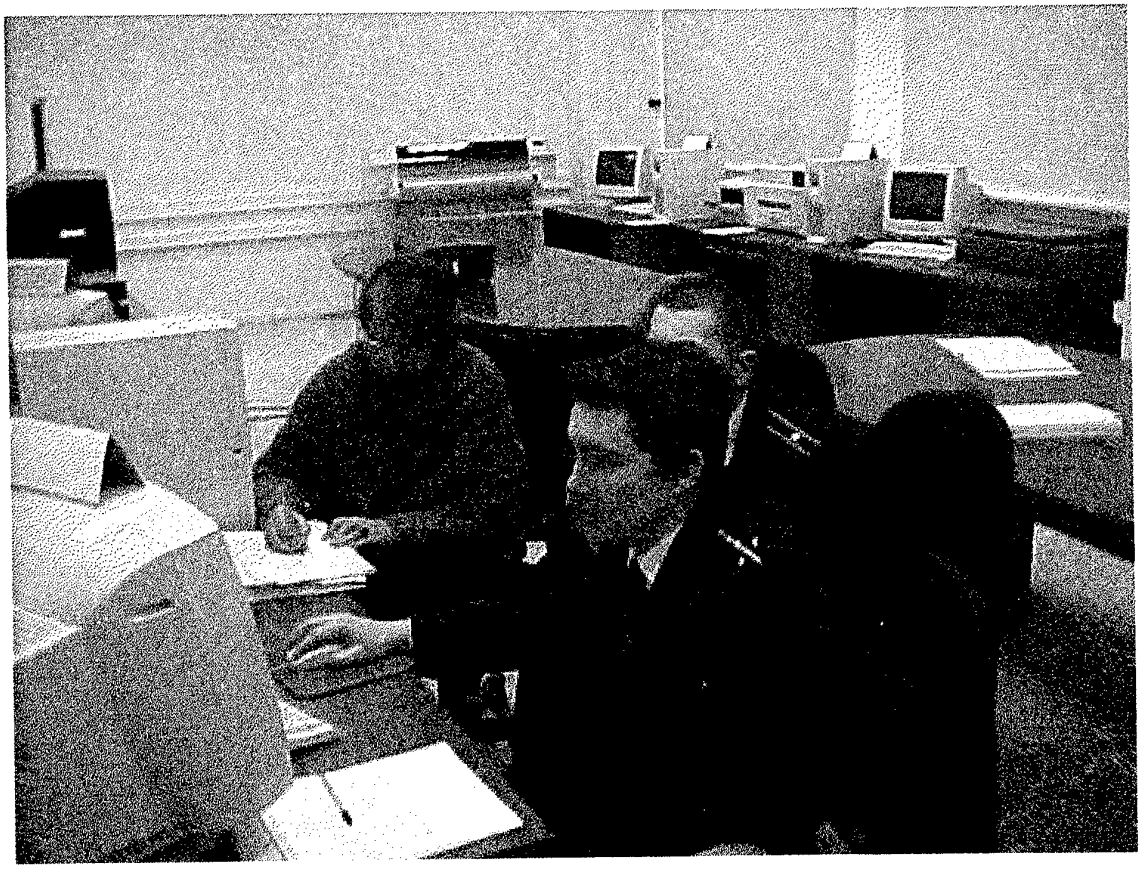

Figure 1: Oracle RDBMS Students. The group worked within each IAS laboratory to remotely access geo-spatial data. In this example, the students are located within the mobile laboratory using a standard PC workstation that accesses the UNIX Oracle data structures.

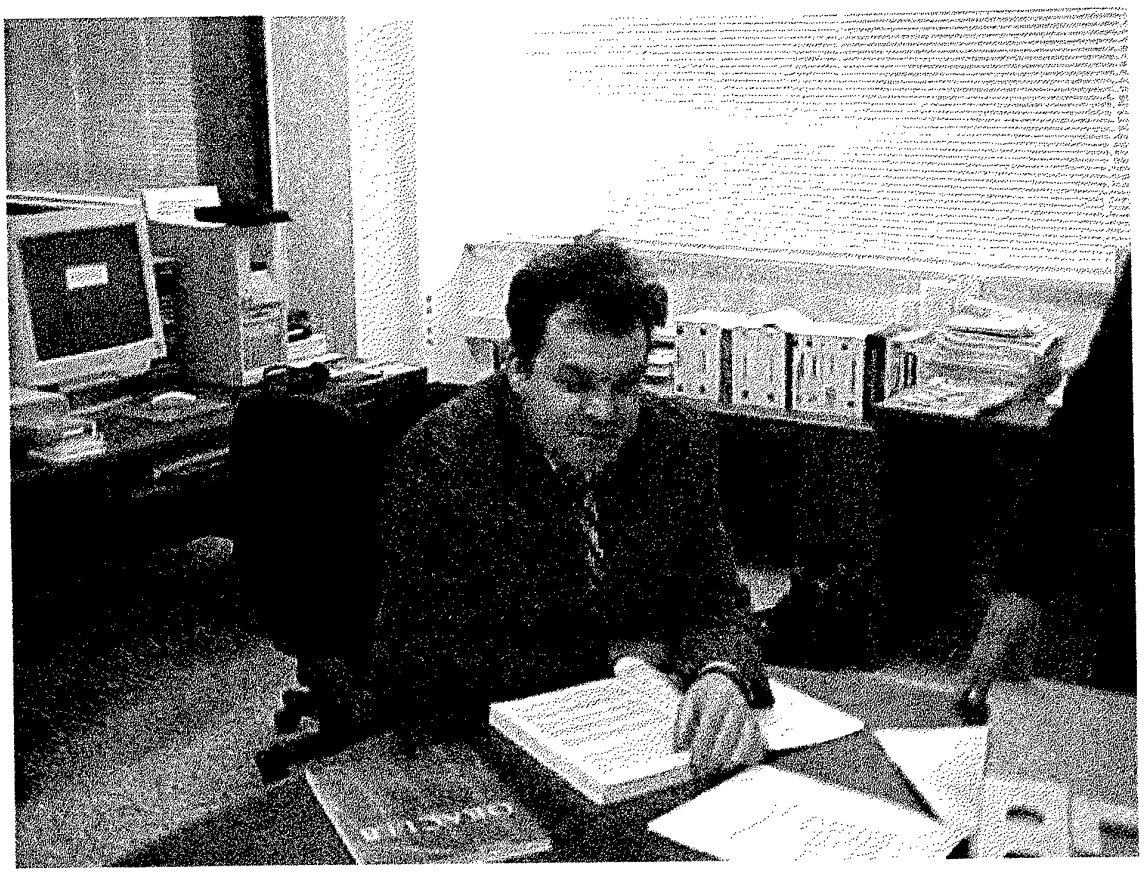

Figure 2: Principal Instructor Alexandr Starshinin. Mr. Starshinin works closely with MOD to manage the system level installation of the Oracle 8 RDBMS. ALL works closely with Oracle representatives in St. Petersburg and Moscow to assure efficient operations and development of IAS- RDBMS models.

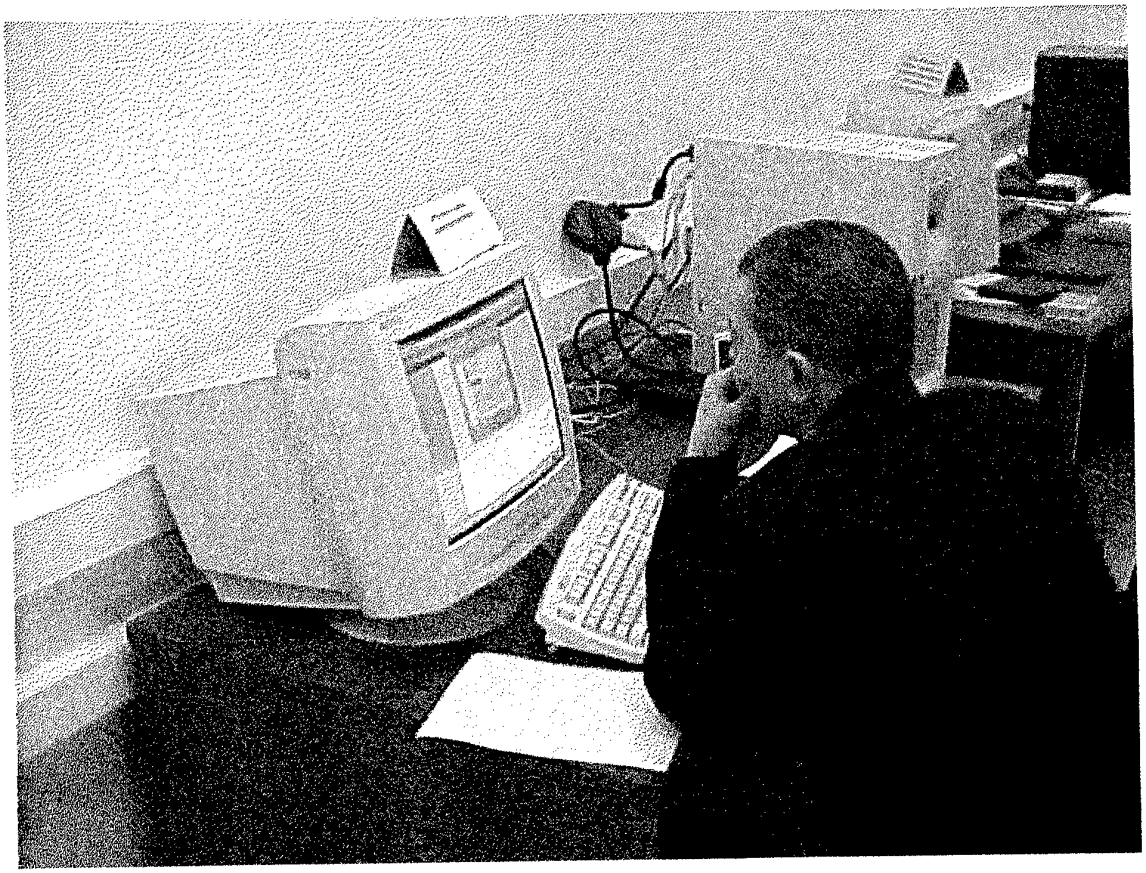

Figure 3: MOD Student Tutorials. ALL has developed stand-alone student tutorials (laboratory exercises) that may be used by MOD to test and evaluate their knowledge about a particular topic. In this case, an MOD student is working with an SQL link that ties Oracle 8 RDBMS to the ESRI-GIS.

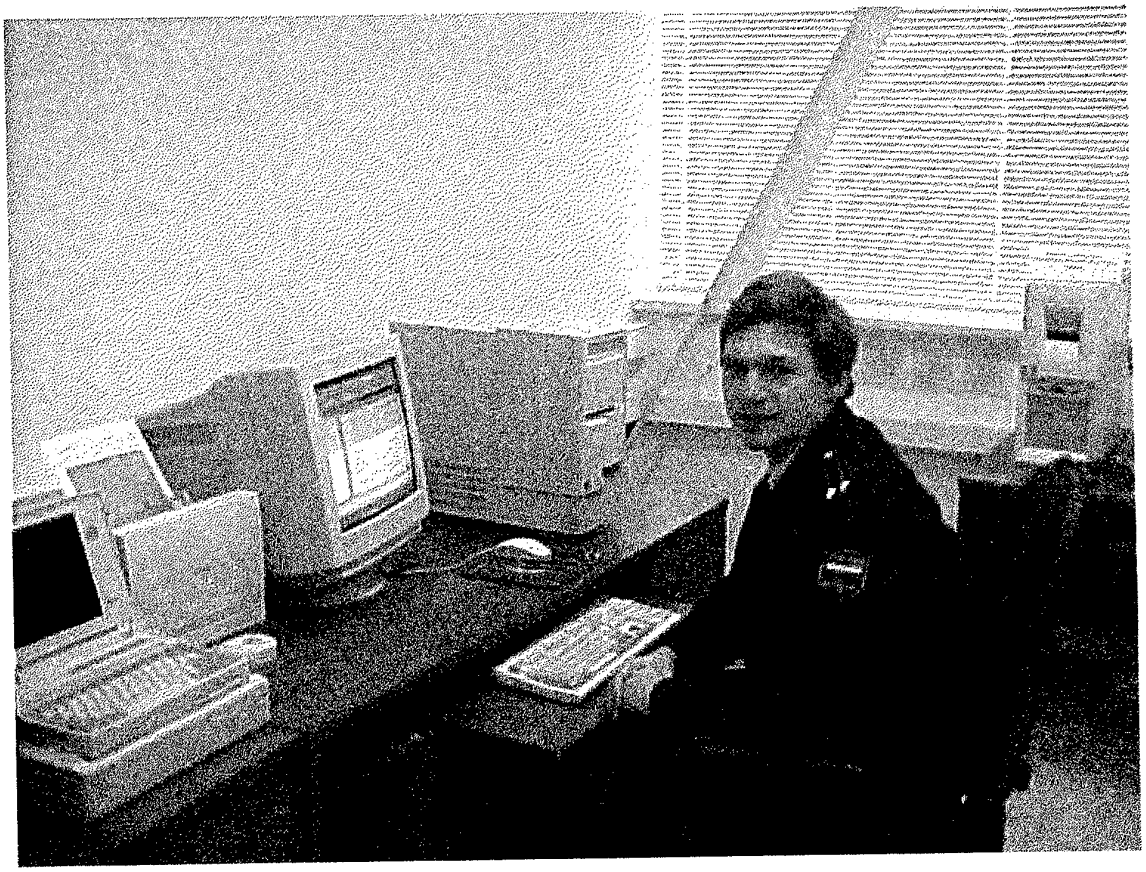

Figure 4: MOD Trained Professionals. In this figure, Igor demonstrates a new program he has developed to archive and display geo-spatial data using the Oracle 8 RDBMS and the ESRI-GIS. As shown, the student is working within the Vibro-Acoustic Laboratory using the 100Base-T ethernet network designed by ALL. The Oracle data and the GIS data are located in separate laboratories.

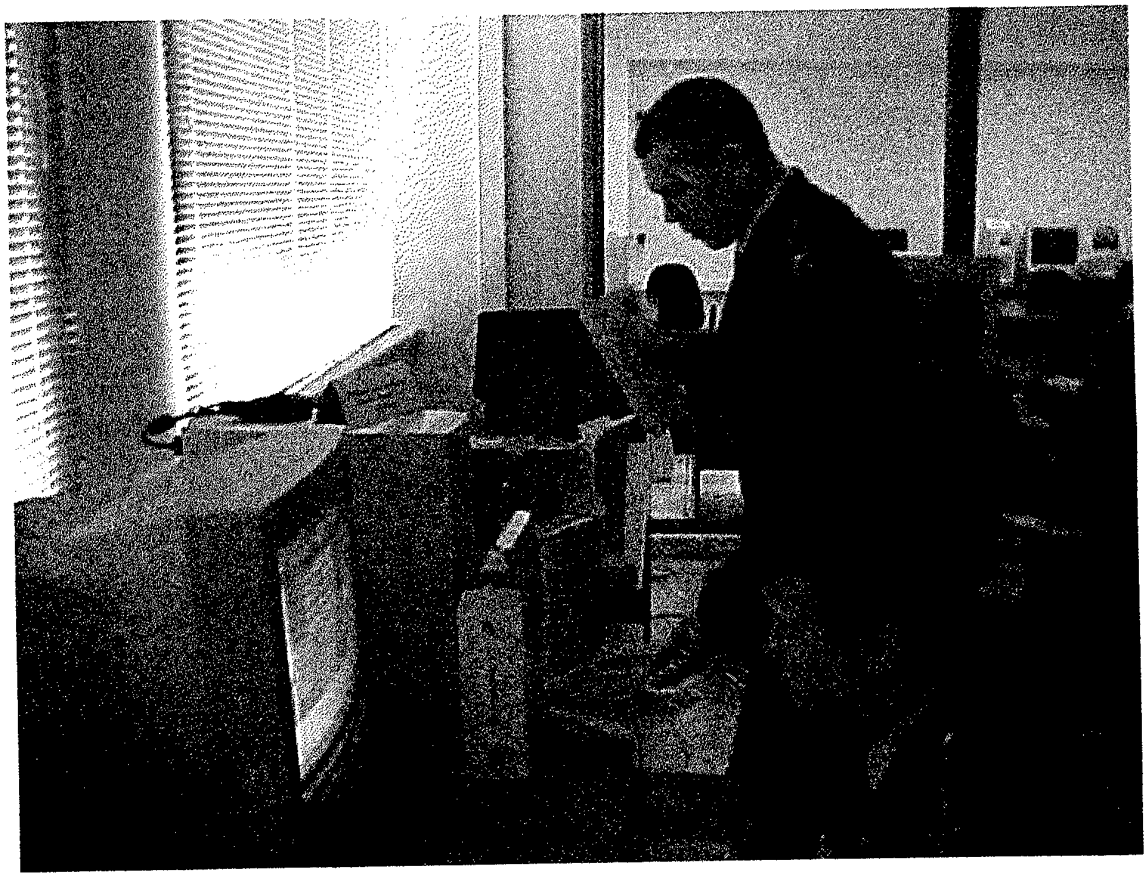

Figure 5: MOD Geo-Spatial Data Analysis. In this figure, Sergei demonstrates a new technique for modeling transport phenomena. The algorithm uses data stored within the Oracle 8 RDBMS, but relies on the ESRI-GIS for visual display. Once again, the 100Base-T ethernet network is used to automatically mount all data structures from the UNIX server.

Figure 6: Technical Development Team. Col. Sergei Shibunin (second from right) leads the MOD development effort for IAS operations. In this figure, he is conferring with his principal development team for mobile operations.

Figure 7: IAS Security. Mr. Stanislav Myasoedov (left) leads the IAS security team. He is responsible for securing laboratory rooms within the IAS, and is also a member of the mobile laboratory development team.

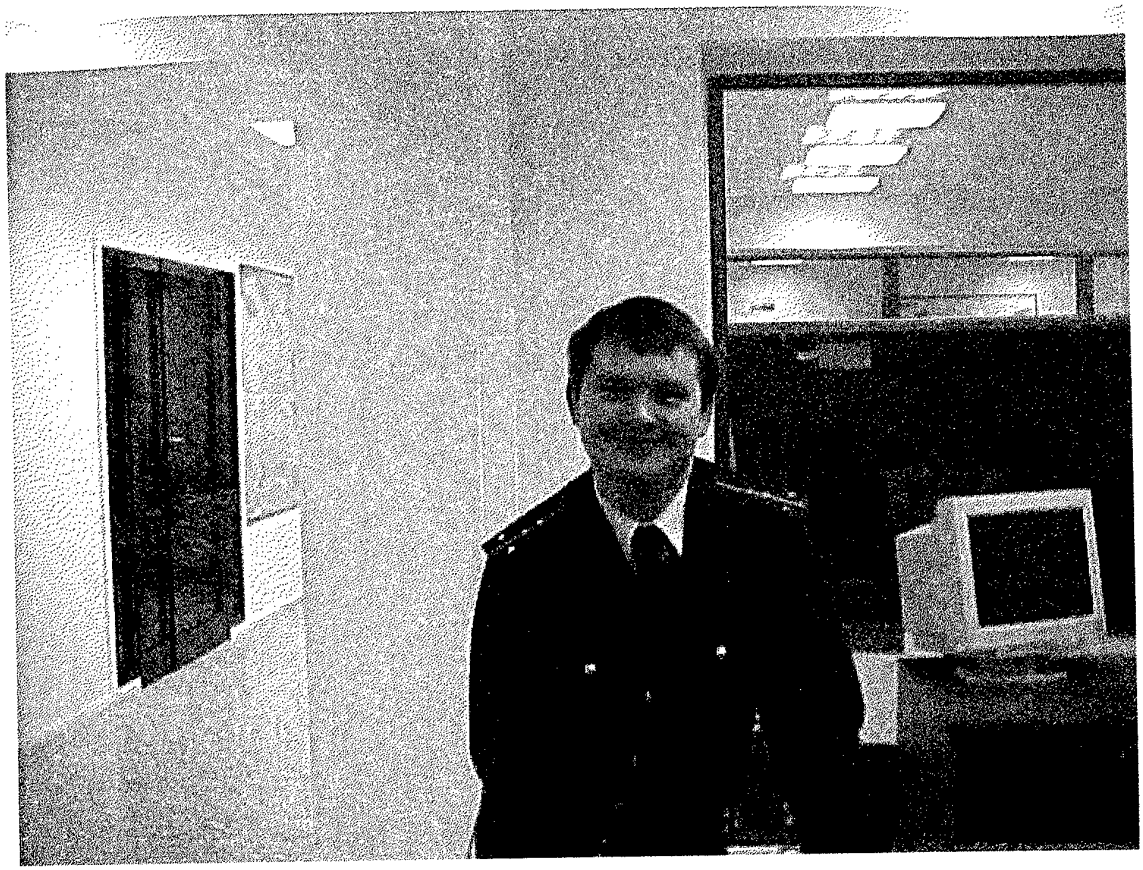

Figure 8: IAS Systsems Administration. Mr. Taras Marenich is the principal MOD systems administrator for the Oracle RDBMS and the ESRI-GIS. Taras has excellent hardware development skills, and a good working knowledge of all major systems found within the IAS. During the course of this R&D training effort, we have developed a very close working relationship.

9. Conclusions

Techniques for documenting an Oracle RDBMS are inherently complex since multiple users and tasks are running simultaneously on servers within five primary MOD IAS laboratories. Since individual scientists create specialized databases for their respective investigation, a common framework is required to insure that a single DBA can ascertain the fidelity of the RDBMS and correct deficiencies before serious problems affect the performance of the central processor. In this final report, we provide scripts and procedures for adequately documenting the Oracle RDBMS using standard SQL functions. The SQL scripts outlined in this text have been presented to MOD for their respective use, and general suggestions have been made to help their systems administrator (Taras Marenich) configure and design a comprehensive documentation program. The scripts also show safe techniques for terminating Oracle processes and rebuilding the Oracle RDBMS during emergency operations. These algorithms may be directly applied to the CTD laboratory for safe administration of the materials database.

In future efforts, ALL will work with MOD to develop cross-platform methods for linking GIS topographic data structures with CTD materials information. The CTD information is not spatially oriented, but is chronologically organized according to test procedure. Hence, each data point from the CTD contains a unique time stamp and instrument attribute. The time stamp is used to simply organize all available information according to the relevant testing sequence. For example, if two or more tests are performed on a common material, it is important to know which material was tested first -- particularly if the same sample is used in future non-destructive testing. The instrument label is acquired from the LIMs data structure. Since all LIMs use a proprietary data structure that may not be SQL compliant, software may be required to translate the instrument information into a more universal format that is easily used within the IAS. The translation is also required to export this data to other systems outside the SCSSTM. Since both the IAS and the emerging CTD will use a common spatial data engine (Oracle), the exchange requirements may be minimized by using a common database structure. The structure should have variables in place to store the (x,y,z) spatial data and related fields for storing the instrument and time attribute information. In additional, in situ samples from mobile operations will require additional fields that link the field sample location to the related within laboratory measurement.

It is anticipated that ALL will assist MOD in the development of future database systems for the CTD. This research is essential to the mission requirements of DTRA and ERO, and has produces some very fruitful results in the field of computer science, applied mathematics, and engineering. We look forward to close collaboration with ERO in future research efforts.

9. References

Black, James D., "Fusing RDBMS and GIS," *GIS World*, July 1999, pp. 44–47.

Conway, J.W., K.D. Hondl, M.J. Moreland, J.M. Cordell, and R.J. Harron, 1995: Improvements in the WSR-88D dealiasing algorithm: The pursuit of the final, most important gate. Preprints, 27th Conf. Radar Meteor., Vail, CO, 145-147.

Donaldson R.J. Jr. and P.R. Desrochers, 1994: Ecosystem detection and warning by GIS. The Tornado: Its Structure, Dynamics, Prediction, and Hazards. C. Church, Ed., Amer. Geophys. Union, 203-221.

Doswell, C.A. III, S.J. Weiss, and R.H. Johns, 1993: Ecosystem forecasting: Structure, Dynamics, Prediction, and Hazards. C. Church, Ed., Amer. Geophys. Union. 557-571.

ESRI Systems Administrators Guide 1999. The Environmental Systems Research Institute, Redlands, California, 285-289.

Green, G.D., E.D. Mitchell, and J.A. Haro: Mini-supercell interaction: The February 13, 1995 Mesa Event. WRH Technical Attachment, No. 96-32.

Healey, R.G., "Database Management Systems," *Geographical Information Systems, Volume 1: Principles*, 1999, pp. 251–267, Longman Scientific and Technical, New York.

Horn, B. K. P., Understanding Image Intensities, J. Artif. Intell., 8, 1997.

Masuch, J.M.F., 1999: Interim Project Report I-V, ERO III, The European Research Office, London, England.

Oracle Systems Support - Reference Manual, 1999. The Oracle Corporation, Provo, Utah, 110-193.

Spitzer, Tom, "A Database Perspective on GIS," *DBMS*, November 1999, pp. 95–102.

Williams, L., Pyramidal Parametrics, , J. Comput. Graphics, 22, 1998.

Watts, B.A., and F.L. Ogden, 1998: Formation of Saturated Areas on Hillslopes with Shallow Soils , submitted for publication,Water Resources Research, AGU.

AD NUMBER	DATE	DTIC ACCESSION

1. REPORT IDENTIFYING INFORMATION

A. ORIGINATING AGENCY UNIV. FOUNDATION, APPLIED LOGIC LABORATORY, AMSTERDAM.

B. REPORT TITLE AND/OR NUMBER MODELING TRANSPORT PHENOMENA WITHIN THE FSU INFORMATION ANALYSIS SYSTEM

C. MONITOR REPORT NUMBER R+D 8837-EN-01

D. PREPARED UNDER CONTRACT NUMBER N68171-99-C-9027

2. DISTRIBUTION STATEMENT

APPROVED FOR PUBLIC RELEASE

DISTRIBUTION UNLIMITED

FINAL

REQUES

1. Put your
on revers
2. Complete
3. Attach for
mailed ic
4. Use uncl
informat
5. Do not on
for 6 to

DTIC:

1. Assign A
2. Return t

DITIONS ARE OBSOLETE