Working Paper and Preprint No. 88/8

A MODEL FOR A SOILS AND TERRAIN DIGITAL DATABASE

John H.M. Pulles

September, 1988

INTERNATIONAL SOIL REFERENCE AND INFORMATION CENTRE

Working Paper and Preprint 88/8

A MODEL FOR A SOIL AND TERRAIN DIGITAL DATABASE

Development of a database structure for SOTER and implementation in a Relational Database Management System (RDBMS)

Prepared by John H.M. Pulles
as a M.Sc. study, Wageningen Agricultural University,
in the framework of the GLASOD project, ISRIC.

September, 1988

TABLE OF CONTENTS

SUMMARY

In August 1986, the SOTER Project was started. Its objectives were to produce a World Soils and Terrain Digital Database at a map scale of 1:1M, in order to improve the mapping and monitoring of world soils and terrain resources, and to be able to provide accurate information about them to planners.

The first phase of the project is to be completed at the end of 1989. At that time, a universal legend has to be adopted, a database must have been tested on data from selected pilot areas, and a geographic information system must have been selected.

At this early stage, the future use of the database, which kind of transactions will prevail, is still not clear. Therefore, it is not possible to select a database structure which is optimal for the future transaction profile.

For the reason of having a flexible, sound structure, it is decided to construct the database in accordance with the relational model.
Normalization has not been done to its highest forms; especially in a "large database" environment a high normal form is not favourable in case of a frequently occurring linkage of objects during retrieval.
The SOIL table of the SOTER database has two (composite) candidate keys which are considered acceptable, for by doing this both the terrain component and the soil component have a direct relation towards profile data.
A disadvantage is the introduction of some redundancy.

PREFACE

The World Soils and Terrain Digital Database (SOTER) was first mentioned to me in early 1987. In March 1988, contact was made with the SOTER Project in order to make a contribution to it as major part of a MSc. study in Computer Science at the Agricultural University of Wageningen.
The Project was in its early phase and it was still vague how computers were to be used for it. It was decided to design a database structure for it, and an input application.

The first chapter of this report gives an introduction to the SOTER Project. The second chapter describes the objectives of the first phase of the SOTER project and the objectives of this study (in short). Chapter 3 makes an analysis of the system requirements, and the kind of data involved. Chapter 4 of this report describes the data model and its implementation. The created ORACLE application forms are discussed in chapter 5.

Finally, I want to express my acknowledgements to Mr. P. Driessen for throwing out a fish, to Mr. W. Peters for catching it, and to Mr. C. Valenzuela of the I.T.C. (Enschede).
Especially, I would like to thank Mr. J. den Dulk of the Department of Computer Science for his help in initiating this study and his guidance during it.

1. INTRODUCTION

Soils throughout the world differ much in their suitability for agriculture or other kinds of land use. Accurate information about the land is essential for a rational planning related to the development, management and sustainable productivity of these resources.

The increasing pressure on the land, the often indiscriminate destruction of forests and woodlands, and the spectre of land degradation results in decreased productivity with dire social consequences. This provides a strong reason for improved mapping and monitoring of world soils and terrain resources, and the development of an information system capable of delivery of accurate, useful, and timely information about resources to decison-makers and planners.

In response to this, the urge was felt to create a World Soils and Terrain Digital Database (SOTER) in order to provide in the need for detailed and up-to-date information on land resources. In August 1986, the SOTER project proposal for a world soils and terrain digital database at a map scale of 1:1M, was endorsed.

As general requirements of the database were considered (SOTER Project Proposal, 1986):

1. amenable to updating and purging of obsolete and/or irrelevant data,

2. possibility of exchange of data with data bases of other global environmental resources,

3. accessible to a broad array of international, regional and national decision-makers to provide them with interpretative maps and tabular information essential for development, management, and conservation of environmental resources,

4. transferable to developing countries for national database development at larger scale.

The present study is a contribution to the SOTER project on the issue of database design and construction of an application for data entry.

2. OBJECTIVES OF STUDY

The 1st phase of SOTER, which is to be completed on December 31st 1989, has as its objectives (:

1. adoption of a universal legend, and definition of soils and terrain parameters and specifications to be included in the database,

2. development of a detailed set of specifications that define the minimum set of capabilities required for the database,

3. selection of three pilot areas in developing countries for initial database construction, and acquisition and correlation of all relevant maps and data,

4. construction of a prototype database, data entry, and test of the reliability, accuracy and utility of this database,

5. assessment of current geographic information systems and development of recommendations on the optimal system for the SOTER project.

At short terms, the objectives of SOTER are to collect and input data from selected pilot areas and to use this data in testing different existing geographic information systems.

The main objective of the present study is to develop an appropriate database structure, and to implement some first applications necessary for data entry.

The data can be distinguished in data that describe the location and extent of objects, and data describing their characteristics. This study is not concerned with the geo-graphical (spatial) characteristics of objects, but merely with their soil and terrain properties.

3. SYSTEM ANALYSIS

With the SOTER project being in its initial phase, it is unknown at present what the future use of the database will be. Especially during the first phase of the project the flexibility of the system is an important requirement; the attributes to be included may change due to experiences in the pilot areas, and developing countries must be able to tailor the system in accordance with their own requirements.
One of the requirements is that the system can be purchased at limited costs. This makes a system that can work on a micro-computer preferable (not considering the amount of data).

It was decided to use ORACLE as a Database Management System, for reasons of being a relational DBMS, suitable for use on a micro computer and compatible with UNEP's Global Resource Information Database (GRID) in Geneva that uses ARC/INFO. Also, ORACLE can be linked to the Integrated Land and Watershed Management Information System (ILWIS), which is developed at the ITC (Enschede) and a strong candidate to be selected as a GIS for the SOTER project.

The SOTER Project distinguished three objects to include in the SOTER Database (SOTER Report 88/2), polygon, terrain and soil.

For most attributes of these objects, it is proposed to specify, when a value cannot be given, whether it is unknown or not applicable in the context of that record/tuple. The report uses the symbols '?' for 'unknown' and '#' for 'not applicable'.

For attributes of quantitative nature, it has been decided that the lower values of class limits are stored. It should be documented when this value is an 'expert estimate', in case of a lack of data. It must be noted, that these values should not be used for calculations, but merely as a code for the class that applies to an attribute.

On the long term, especially when the system is to be transferred to a larger scale, the qualification of 'measured/exact value' will be desirable. On the short term, during the first phase of the project, this problem does not occur; all values are class values.

Therefore, it is proposed not to make any distinction at first between the different nature of the attributes; both 'unknown' and 'not applicable' may be stored as NULL values.

For the pilot areas, the input of data is the primary process. It will be done manually at a micro computer; it is not known how this will be in future phases with a much larger amount of data. With much data being extracted from existing soil maps, most data can be expected to be supplied on paper forms.
The total area of the pilot area in South-America amounts to 250,000 km^2. With the requirement for a map polygon of 1 cm^2 at a scale of 1:1M, the maximum number of polygons is 2500. Each polygon (15 entries) has a maximum of 3 soils (66 entries) and 3 terrain components (24 entries), which may make the data entry too much when done manually.

Required functions of the system are:

1. input, update and deletion of data at a micro computer
2. retrieval of data, e.g. in tabular form.
3. maintain data integrity

At present, the system is thought of as a single user system, which excludes concurrency problems. In its final form, it is to be expected that the database will be a multi-user system. The present database has to be reviewed than on concurrency and security aspects.

Most of the attributes of the objects are related to one another. Some combinations of attribute values can be considered highly improbable.
In case that the soil layer file contains exact, measured values, some functional dependencies appear, which are not there when the values are in classes.

4.1 The data model

All data available to the users of an information system are stored in the database. These data describe the objects that are considered relevant.
A logical data structure can be represented by using the entity-relationship approach (see figure 1.). In this approach an entity is "a thing that can be distinctly identified", and a relationship is "an association among entities" (Date, 1986).

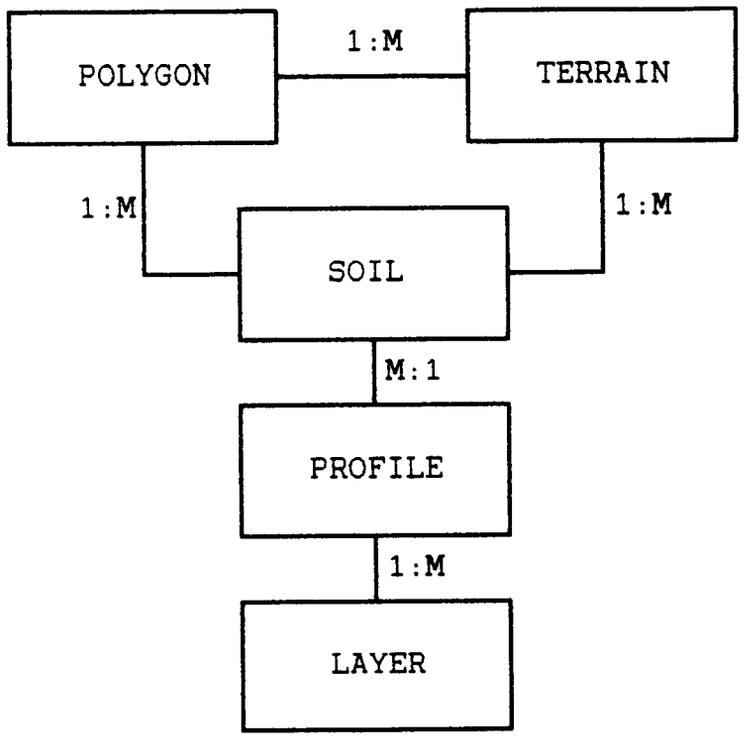


Figure 1. Entity-relationship diagram for entities of data model for the SOTER database.

The logical data structure, as a model of the universe of discourse, must have the following characteristics:

1. recognizable to users,
2. contain all relevant entities and their attributes,
3. implementable without redundancies.

General concepts in the SOTER Procedures Manual (1988) are polygon, terrain component and soil. A polygon is defined as an enclosed map delineation characterized by a certain regional landform.
Each polygon may include a maximum of three terrain components which are defined as segments of the polygon with comparable topographic (e.g. local surface form) and/or soil patterns.

For each terrain component at least one soil is characterized, with a maximum of three soils per polygon. The representation of a soil in the database has a maximum of four layers within a depth of 150 cm.
These limitations must be implemented in the applications on the database; the database structure itself does not have them.

Soil data can be distinguished in 'external' data, such as slope position, that describe the location of the soil, and 'internal' data that describe the profile (below surface). Profile data can be representative for the whole profile, or for a layer only.

The SOTER Procedures Manual (1988) proposes both a maximum of three terrain components and a maximum of three soils for each polygon, to "encourage the map compilator to be very selective in choosing those soils which are most important to the subsequent interpretations on the database". The possibility of having three soils whose distribution over the polygon does not coincide with the distribution of the terrain components, is not taken into account.

The occurrence of nine (3 * 3) terrain/soil combinations within one polygon can be realized in two ways:

1. by inclusion of the attributes soil_proportion, slope_positon and profile_id in the terrain table for each soil component (thus 3 times), or

2. by creation of a separate table that contains the polygon_id, terrain_number, soil_number, soil_proportion, slope_position and profile_id.

Both solutions have their merits and demerits; the first solution has the advantage that it eliminates the presence of a separate soil table. However, in general the number of soil components amounts to only one, which makes the attribute fields for the other components redundant. The second approach has the disadvantage of an extra table and henceforth will worsen queries that concern both the attributes of the terrain component and of the soil component. Because of its advantages of the increased flexibility of the data structure towards future changes, and the limitation of redundancy, the latter solution is preferred.

In the original concept of the manual, the user identifies a soil (including the profile data) by the identification of the polygon together with a soil component number. Thus, at the external level (the one concerned with the way the data is viewed by individual users), the unique profile identification is not visible.
However, in the database it is necessary, for the profile data cannot be (uniquely) identified by a soil component because many soils can have the same profile (data). The soil component in the database can be considered a relationship between terrain, polygon and profile.

Both the relations of polygon with terrain and polygon with soil are one to many, with many ranging from 1 to 3. The relation between terrain and soil is has also become 1 to 3 because of the soil table.
The soil-profile relation is many-to-one; several soils may refer to the same profile, with a minimum of one soil.

4.2 Implementation of the data structure

The declarations of the database tables for the entities in the diagram of figure 1, are presented in Annex A.

Important aspects of databases are integrity and security. Integrity refers to the accuracy or validity of data, security refers to the protection of data against unauthorized disclosure, alteration of destruction.

Integrity

The integrity part of the relational model consists of the so-called 'entity integrity' and 'referential integrity', concerning primary keys and foreign keys respectively. Date (1986) defines a foreign key as an attribute (or attribute combination) in one relation whose values are required to match those of the primary key of another relation.

To ensure entity integrity, for each table of the database, all fields in the primary key are specified as NOT NULL, and a UNIQUE index is created over the combination of all fields in the primary key.

The SOIL table has <u>two</u> unique indexes; an index on POLYID, TERRID and SOILID to ensure that every terrain-component/soil-component combination will be unique within a polygon, and an index on PROFID, POLYID and SOILID to guard that each soil polygon within a polygon has the same profile number (when occurring in more than one terrain component).
It must be noted that this implies that the soil table could be further normalized into the relations (polyid, terrid, soilid, soilprop, slopepos) and (polyid, soilid, profid).

The indexes on the primary keys are created with the option NOCOMPRESS. This option is particularly useful when a query involves a small subset of a table's data (the indexed attributes) and might be done entirely in the index, without having to access the data block.

Constraints on the database can be considered at different, increasing levels:

1. attribute constraints (the domain of an attribute)
2. tuple constraints (between attributes within one tuple)
3. table constraints (between tuples within one table)

4. database constraints (between tables within a database)

Tuple and table constraints for each table, and the database constraints are given in Annex B.
An example of a database constraint is the foreign key or subset constraint. With an increasing level of the constraints, they are more difficult to guard (Bots, 1985). The disadvantage of the normalization of tables is the introduction of database constraints.

Foreign keys in the database are:

TERRAIN.POLYID to POLYGON.POLYID
SOIL.POLYID+SOIL.TERRID to TERRAIN.POLYID+TERRAIN.TERRID
PROFILE.PROFID to SOIL.PROFID
LAYER.PROFID to PROFILE.PROFID

To enforce referential integrity, the authorization subsystem (see Security) can be used to prohibit on-line operations that may violate the constraints. The foreign key constraints must be part of the requirements of application programs (see paragraph 5.1). It is advisable to have a utility program that can be run periodically to check for and report on any constraint violations.
This check can easily be done by asking selected queries on the pseudo keys of the different tables.

Unfortunately, the implementation of constraints is only to a limited extent supported by the DBMS. For example, attribute constraints are slightly supported by defining attributes of a certain datatype.
More complex constraints must be implemented in specific applications. Note that when the databased is approached by other means, these constraints may be violated.

Security

Security refers to the protection of data against unauthorized disclosure, alteration or destruction. The ORACLE DBMS offers the means to authorize users for different kinds of access to specified tables or views.
The following operations on database tables may violate foreign key constraints (Date, 1986):

1. DELETE on the referenced table,
2. UPDATE on the referenced table's primary key,
3. INSERT on the referencing table,
4. UPDATE on the referencing table's foreign key.

5. APPLICATIONS

5.1 ORACLE Interactive Application Facility

The use of Oracle's Interactive Application Facility (IAF) is a relatively easy way of generating screen forms for the manipulation of data in tables.
IAF provides the possibility of defining SQL-triggers in order to maintain the integrity of the data base. A trigger is an SQL-statement that will be executed at a certain event. These triggers can be defined at two levels: field level or block level; field level triggers are executed when a screen field's value is changed, block level triggers are executed when the transaction
is committed to the database.

Attribute constraints can be implemented by specifying a datatype and/or a range and/or a trigger for that attribute, in which the validity of the entered value is checked.
In case of a trigger, the use of a subquery from a domain table is preferred for this is faster than using OR-predicates or IN-value lists (ORACLE Database Administrator's Guide, 1984). It is evident that attribute constraints are checked at field level.

To maintain an acceptable speed of data input, the so-called subset-requirements are not checked at field level (when field is left), but at block level (when commit).
Because update of primary key fields is not allowed, what needs to be triggered is:

DELETE on the referenced table, and
INSERT on the referencing table.

The subset requirement between soil.polyid and polygon.polyid is not necessary to check, for this one must be valid if soil.polyid+terrid is a subset of terrain.polyid+terrid (transitive).

ORACLE provides eight types of block triggers:

PRE-QUERY	PRE-INSERT	PRE-UPDATE	PRE-DELETE
POST-QUERY	POST-INSERT	POST-UPDATE	POST-DELETE

The two query triggers are of no importance for integrity, but they are useful at the profile and layer table for the retrieval of a polygon number and a soil number to display instead of the profile number that must remain invisible. The triggers that are used in the IAP forms are listed in Annex C.

The insert on the referencing table can be checked by a statement such as:

```
"select <referenced_key> from <referenced_table>
 where <referenced_key> = <inserted_value>"
```

When this query is successful, the insert is permitted, if it fails, the transaction will be rolled back.

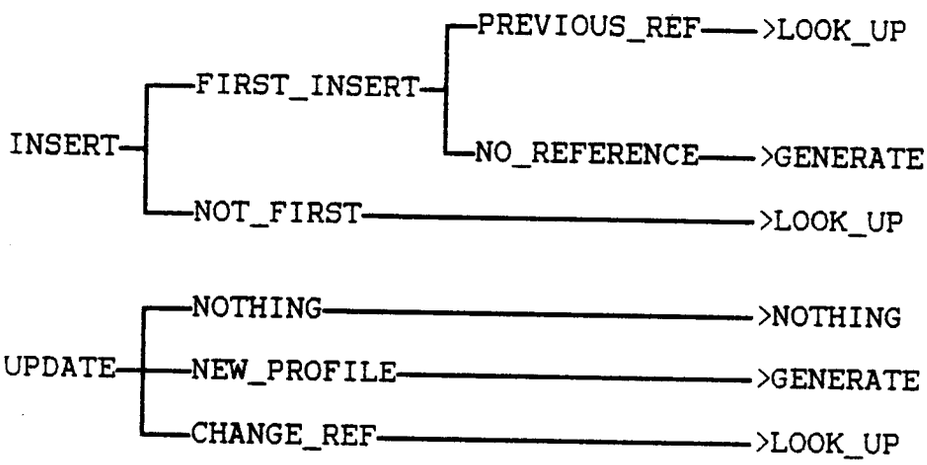
The delete on the referenced table is permitted when the following query fails:

"select <referencing_key> from <referencing_table>
where <referencing_key> = <value_to_be_deleted>".

The soil table presents the most constraints for maintaining integrity. One of the reasons is that it is not fully normalized; e.g. when a soil's profile changes, the profile number must be changed at each occurrence of that soil (1 to 3 times).

The unique profile identification numbers are generated sequentially by means of a table 'SEQNOS' that contains the highest profile number.
When using the soil form, the user refers to a certain profile by a polygon number and a soil component. When the given soil holds the same profile as another soil, he also must enter the polygon number and soil component of that previous soil.

The profile number is generated when the soil is inserted for the first time, or when during update the user wants to assign a new profile to it. In all other cases, the profile number is looked up in the table. The following scheme shows the different possible actions and the reactions to them.



The screens of the IAP-applications are shown in Annex D.

5.2 Creation and maintenance of domains

To create, update or drop domains, a utility has been written in the C-language, interfacing with ORACLE by means of the Oracle Call Interface (OCI, see Annex E). With the DOMAINS application it is possible to create, update or drop domains with explanatory texts that are of a length limited by the maximum size of the ORACLE LONG datatype.
To keep track of the present domains, an additional table was created, in which the name of the domain (equal to the attribute name), type, width (storage size), scale (decimals) and table name ('parent table') are stored (see Annex A.).

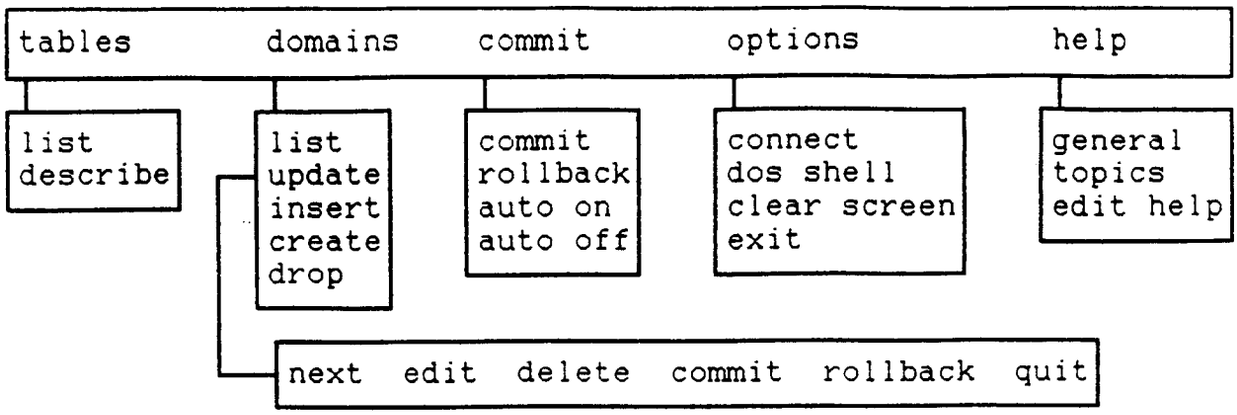


Figure 2. Command structure, option bars and boxes, of DOMAINS application.

The DOMAINS application (see figure 2.) uses the TAB and COL views of the system catalog to get a list of tables for the present user, or for a description of a specific table.

A list of domains is kept in the above mentioned additional DOMAINS table. To create a domain, the user must first select a table, and then the desired attribute in the description of the selected table. All domain tables are then created in the same way; a CODE field of type 'DNAME' and size 'DWIDTH' (and DSCALE decimals for numbers), with an attribute TXT of type LONG, and one attribute TXTLEN of type NUMBER specifying the length of the description contained in TXT. The newly created domain table is added to the domains table.
'Update' is to be used when the present texts must be changed or deleted. When the whole domain table is dropped, it is also removed from domains.

It must be noted that whenever a domain is created, it is necessary to adjust the IAF input file so that it uses the domain table, and to generate a new application form.

6. CONCLUSIONS AND RECOMMENDATIONS

It appears to be quite early to start the design of a database when it is still unknown what the use of the database will be at the time that it is operational.

Especially with large databases it is important for the performance to optimize the data structure with regard to the main transactions that are expected to be done on the database.

The database from this study was implemented with ORACLE version 4.1.4. The expected availability of ORACLE 5 will certainly facilitate the implementation of triggers in the application forms, because it offers the possibility of making tree structured trigger steps, and of macros.

When it will be desired to label values as unknown, unapplicable, class, estimate or measured, it is not recommended to use a numeric code (e.g. '-1'), which may be a member of the attribute's domain, for this purpose. The introduction of a separate label field in a table is to be preferred.

When the final system will be working in a multi-user environment, it will be neccessary to revise the present applications concerning concurrency and safety aspects, which is of more importance then.

LITERATURE

Ammeraal, L., 1987. De programmeertaal C, 4e druk. Academic Service, Den Haag.

Bots, J.M. and J.L. Simons, 1985. Bestuurlijke Informatie Verzorging, Syllabus. Landbouwuniversiteit, Wageningen.

Date, C.J., 1986. An Introduction to Database Systems, Volume 1, Fourth Edition. Addison Wesley Publishing Company.

Hofstede, G.J., 1986. Handboek Databases, Syllabus. Landbouwuniversiteit, Wageningen.

ORACLE Programmer's Guide, 1984. Oracle Corporation, Menlo Park, California.

ORACLE Database Administrator's Guide, 1984. Oracle Corporation, Menlo Park, California.

Peters, W.L. (ed.), 1988. Proceedings of the First Regional Workshop on a Global Soils and Terrain Digital Database and Global Assessment of Soil Degradation. ISSS SOTER Report 3, Wageningen.

Pro*C User's Guide.

Shields, J.A. and D.R. Coote, 1988. SOTER Procedures manual for small scale map and data base compilation. ISSS SOTER Report 88/2, Wageningen.

SOTER Project Proposal, 1986. World Soils and Terrain Digital Database. Wageningen.

SOTER, World Soils and Terrain Digital Data Base at a scale 1:1 M. Wageningen, The Netherlands, 1986.

ANNEX A. Creation of SOTER database tables.

```
REM  'soterdba' = SOTER Database Administrator
REM  'soterdata' = insert/update/delete/retrieval
REM  'public' = retrieval

drop table seqnos;
create table seqnos (
     item_name       char (30) not null,
     max_itemno      number not null
);

insert into seqnos values ('PROFID', 0);
grant select, update on seqnos to soterdata;
create public synonym seqnos for seqnos;

drop table dual;
create table dual (
     dummy           char (1)
);

insert into dual values ('^A');
grant select on dual to public;
create public synonym dual for dual;


drop table polygon;
create table polygon (
     polyid          number (4) not null,
     country         char (4),
     statprov        number (2),
     basemap         char (4),
     reportref       number (4),
     landform        char (1),
     relief          number (4),
     elevation       number (4),
     lithology       char (4),
     lakesurf        number (3),
     seasinund       number (3),
     riverdist       number (3),
     draindens       char (1),
     landuse         char (4),
     yearrec         number (4)
);
create unique index polygonx on polygon (polyid) nocompress;

grant select on polygon to public;
grant insert, update, delete on polygon to soterdata;
create public synonym polygon for polygon;


drop table terrain;
create table terrain (
     polyid          number (4) not null,
     terrid          number (1) not null,
     terrprop        number (3),
     parentmat       char (2),
     textgroup       char (1),
```

```
    surfform        char (1),
    slopgrad        number (2),
    slopleng        number(3),
    stoniness       number (3,1),
    rockiness       char (3),
    grwdepth        char (5),
    grwqual         number(4),
    rootdepth       number (3),
    vegetation      char (2),
    flooding        char (3),
    crusting        char (3),
    surfdrain       char (4),
    overwash        number(3),
    overblow        number (3),
    waterstat       char (1),
    windstat        char (1),
    complexmat      char (1),
    permafrost      char (1),
    icecontent      char (1)
);
create unique index terrainx on terrain (polyid, terrid)
    nocompress;

grant select on terrain to public;
grant insert, update, delete on terrain to soterdata;
create public synonym terrain for terrain;


drop table soil;
create table soil (
    polyid          number (4) not null,
    terrid          number (1) not null,
    soilid          number (1) not null,
    soilprop        number (3),
    slopepos        char (3),
    profid          number (5) not null
);
create unique index soilx_1 on soil (polyid, terrid, soilid)
    nocompress;
create unique index soilx_2 on soil (profid, polyid, soilid)
    nocompress);

grant select on soil to public;
grant insert, update, delete on soil to soterdata;
create public synonym soil for soil;


drop table profile;
create table profile (
    profid          number (5) not null,
    intdrain        char (4),
    sysclass        char (3),
    soildev         char (4),
    refpedon        char (6),
);
create unique index profilex on profile (profid) nocompress;

grant select on profile to public;
```

```
grant insert, update, delete on profile to soterdata;
create public synonym profile for profile;


drop table layer;
create table layer (
     profid          number (5) not null,
     layerid         number (1) not null,
     lowerdepth      number (3),
     abruptness      char (2),
     moisthue        char (5),
     moistval        number (1),
     moistchr        number (1),
     dryhue          char (5),
     dryval          number (1),
     drychr          number (1),
     degrdecomp      char (3),
     biolact         char (3),
     claymin         char (4),
     contrast        char (1),
     diagnhor        char (4)
);
alter table layer add (
     coarse          number (2),
     sand            number (2),
     very_fine       number (2),
     silt            number (2),
     clay            number (2),
     textclass       char (4),
     upwat_kpa       number (2),
     lowat_kpa       number (4),
     upwat_vol       number (2),
     lowat_vol       number (2),
     bulkdens        number (3,2),
     infiltrat       number (4,1),
     sathydcon       number (4,1),
     structure       char (2),
     stabaggr        number (2)
);
alter table layer add (
     orgcarbon       number (3,1),
     totnitro        number (3,2),
     cec_soil        number (2),
     cec_clay        number(3,1),
     cec_eff         number (3,1),
     aec_soil        number(3,2),
     ca_exch         number (4,2),
     mg_exch         number(3,2),
     na_exch         number (3,2),
     k_exch          number(3,2),
     mn_exch         number (3,2),
     al_exch         number(2,1),
     ca_mg_rat       number (2,1),
     ca_k_rat        number(2,1),
     mg_k_rat        number (2,1),
     al_satperc      number(3)
);
alter table layer add (
```

```
     p_avail          char (1),
     p_fixation       char (4),
     s_avail          char (4),
     trace_def        char (4),
     toxic_pot        char (4),
     base_sat         number(3),
     ph_h2o           number (3,1),
     ph_cacl2         number(3,1),
     elect_cond       number (3),
     esp              number(2),
     caco3            char (3),
     gypsum           number(2)
);
create unique index layerx on layer (profid, layerid) nocompress;

grant select on layer to public;
grant insert, update, delete on layer to soterdata;
create public synonym layer for layer;


drop table domains;
create table domains (
     dname            char (30) not null,
     dtype            char (6) not null,
     dwidth           number not null,
     dscale           number,
     tname            char (30)
);
```

ANNEX B. Tuple, table and database constraints.

TABLE POLYGON

Tuple Constraints

1. Permanent lake surface + seasonally inundated <= 100%
2. Relations between attributes; e.g. landform and relief, landform and lithology, landform and land use, distance between rivers and drainage density.

Table Constraints

-

Primary Key

1. Polygon number

END TABLE POLYGON

TABLE TERRAIN

Tuple Constraints

1. Overwash + overblow <= 100%
2. Relations between different attributes.

Table Constraints

1. Total of terrain components within one polygon <= 100%

Primary Key

1. Polygon Number
2. Terrain Component Number

END TABLE TERRAIN

TABLE SOIL

Tuple Constraints

-

Table Constraints

1. Total of soil components within one terrain component <= terrain component's proportion of polygon
2. The combination of polygon+soil component has one profile number, no more, no less (candidate key).

Primary Key

1. Polygon number
2. Terrain component number

3. Soil component number

Candidate Key

1. Profile number
2. Polygon number
3. Soil component number

END TABLE SOIL

TABLE PROFILE

Tuple Constraints

-

Table Constraints

-

Primary Key

1. Profile Number

END TABLE PROFILE

TABLE LAYER

Tuple Constraints

1. UpLimkpa < lolimkpa
2. Uplimvol > lolimvol
3. Sum of $CaCO_3$ and gypsum <100%.
4. pH-$CaCl_2$ < pH-H_2O

Table Constraints

1. Layer number must increase with the lower depth of the layers.
2. For layer number n > 1, layers 1 to n-1 must also be stored.

Primary Key

1. Profile Number
2. Layer Number

END TABLE LAYER

DATABASE SOTER

Tables

polygon
terrain
soil
profile
layer

Subset requirements

1. {terrain.polyid} {polygon.polyid}
2. {soil.polyid+soil.terrid}
   {terrain.polyid+terrain.terrid}
3. {profile.profid} {soil.profid}
4. {layer.profid} {profile.profid}

Database constraints

1. Total area of soil components cannot exceed the area of the terrain component in which they reside.

ENDDATABASE SOTER

ANNEX C. Block triggers in IAP forms.

The block triggers are given per table, together with a label 'F' or 'S', indicating if the trigger has to Fail or Succeed for the commit to proceed.

TABLE PROFILE

PRE-DELETE: the polygon may not be present in the terrain table

```
F    select profid from terrain
     where profid = &poly.profid
```

END PROFILE

TABLE TERRAIN

PRE-DELETE: terrain component may not be present in soil table

```
F    select terrid from soil
     where polyid = &terr.polyid and terrid = &terr.terrid
```

PRE-INSERT: polyid must occur in polygon table

```
S    select polyid from polygon
     where polyid = &terr.polyid
```

POST-INSERT: the total proportion of components must be <= 100%

```
S    select * from dual
     where 100 <= select nvl(sum(terrprop),0) from terrain
     where polyid = &terr.polyid
```

POST-UPDATE: idem.

```
S    see POST-INSERT trigger
```

END TERRAIN

TABLE SOIL

The soil form uses the fields 'prevpoly' and 'prevsoil', which have no corresponding attribute in the soil table. These fields must both be NULL or both be NOT NULL.
The triggers for INSERT and UPDATE operations are conform the scheme given in paragraph 5.1.

POST-INSERT
POST-UPDATE

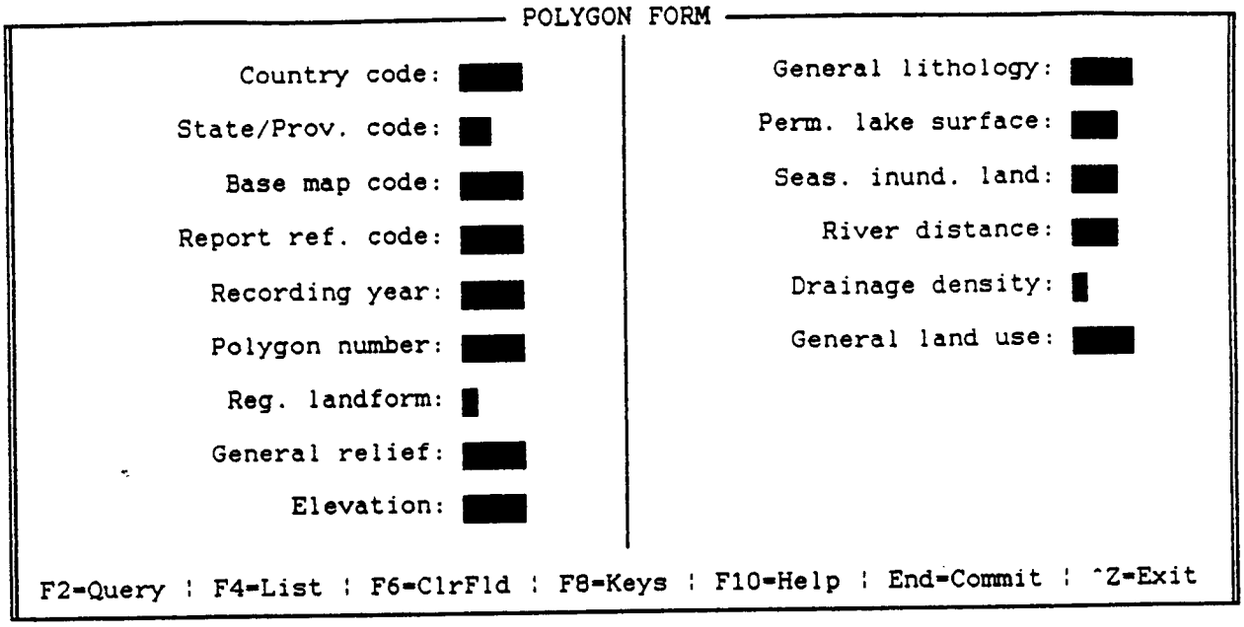
POST-DELETE: no delete if it was the only reference to a profile

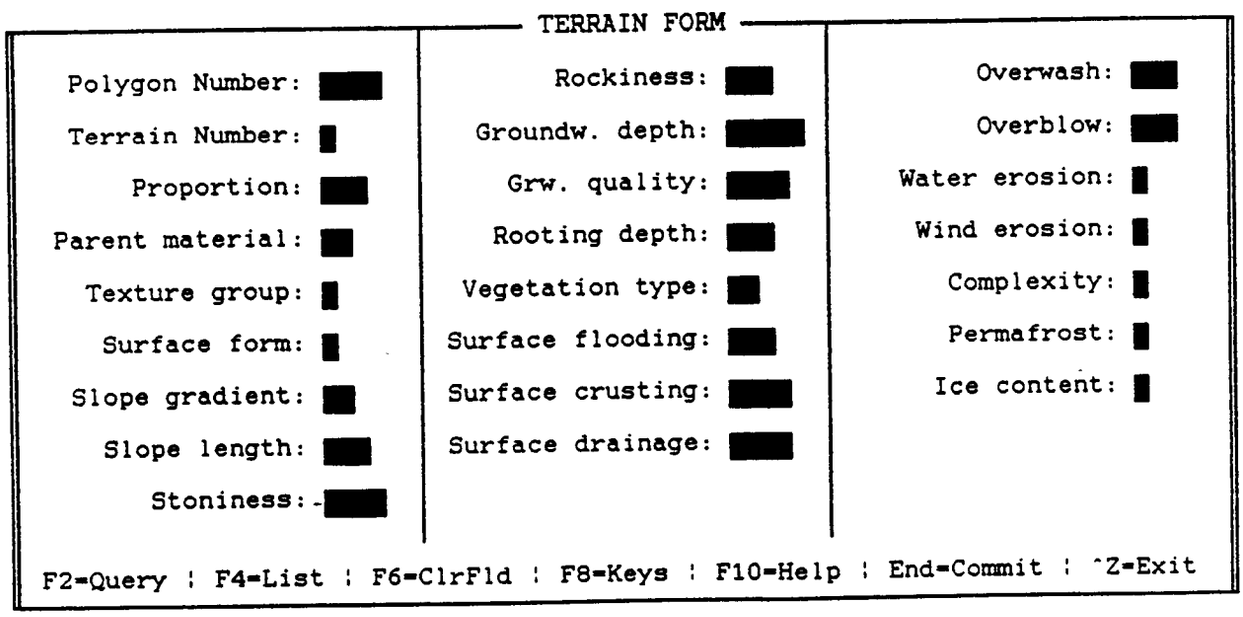
```
S    select profid from profile
     where profid = &soil.profid
```

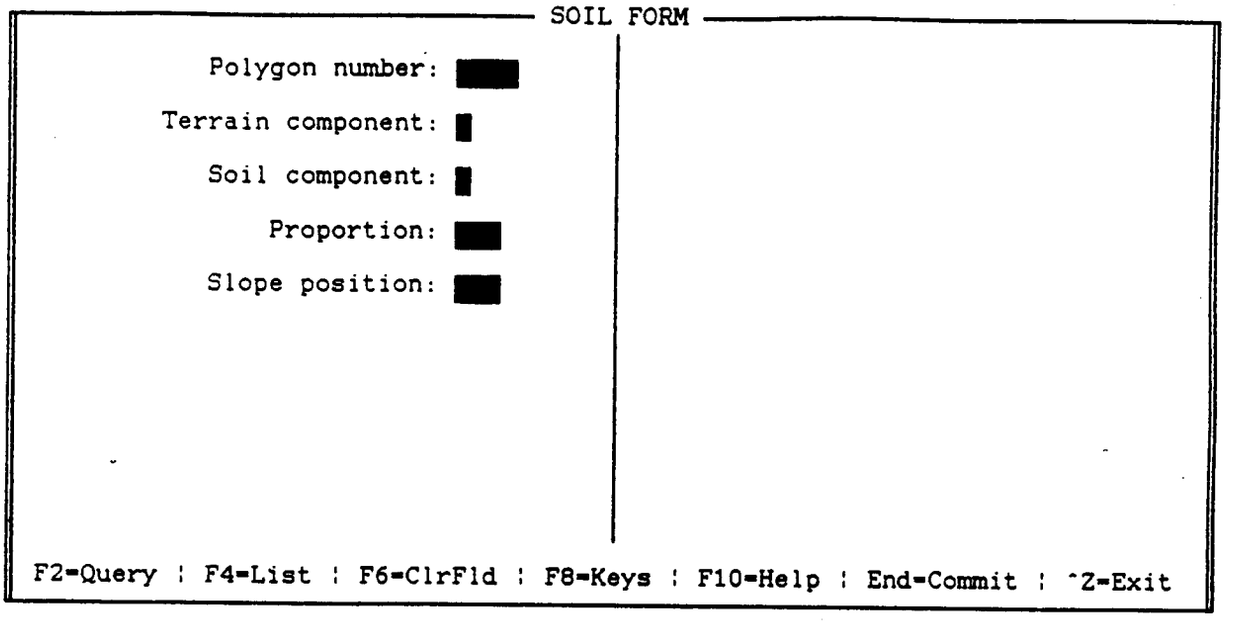
END SOIL

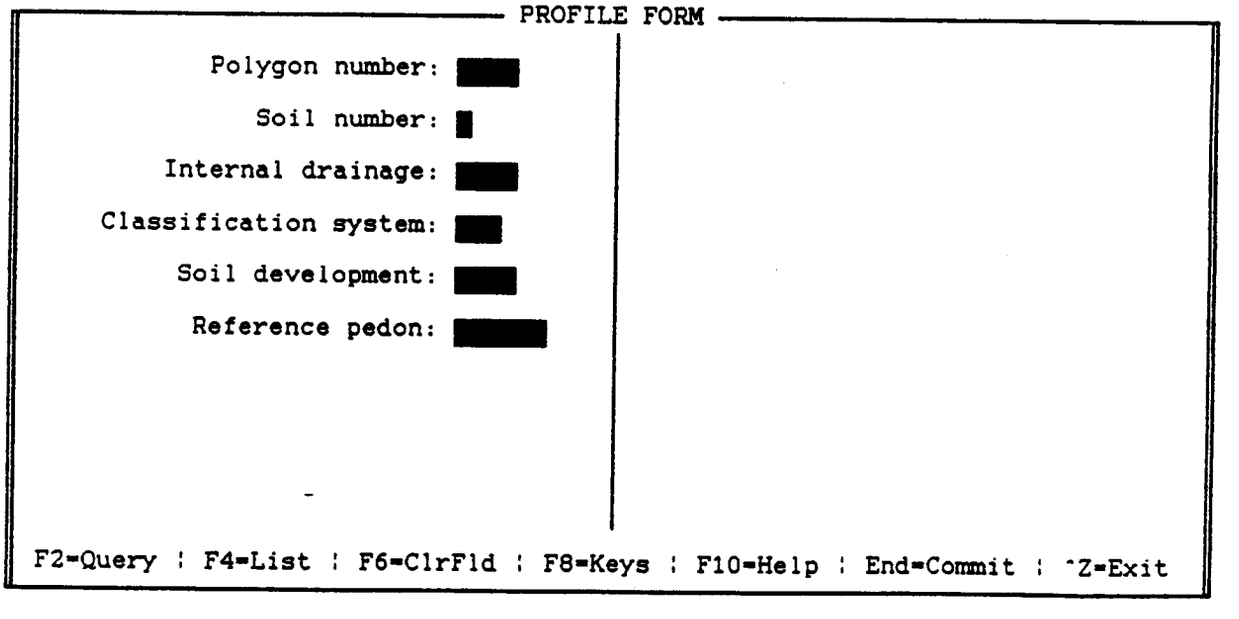
```
TABLE PROFILE

POST-QUERY: select a value for polyid and soilid into the form

     S    select polyid, soilid into prof.polyid, prof.soilid
          from soil
          where profid = &prof.profid

PRE-INSERT: get the appropriate profile number from soil

     S    select profid into prof.profid
          from soil
          where polyid = &prof.polyid and soilid = &prof.soilid

PRE-DELETE: the profile number may not be present in a layer

     F    select profid from layer
          where profid = &prof.profid

END PROFILE


TABLE LAYER

POST-QUERY: idem.

     S    see POST-QUERY in PROFILE table

PRE-INSERT: profid must be present in profile table

     S    select profid from profile
          where profid = &layer.profid

POST-INSERT: check increase of depth with increase of layerid

     F    select one.profid from layer one, layer two
          where one.profid = two.profid
          and one.layerid > two.layerid
          and one.lowerdepth < two.lowerdepth

POST-UPDATE: idem.

     F    see POST-INSERT

END LAYER
```

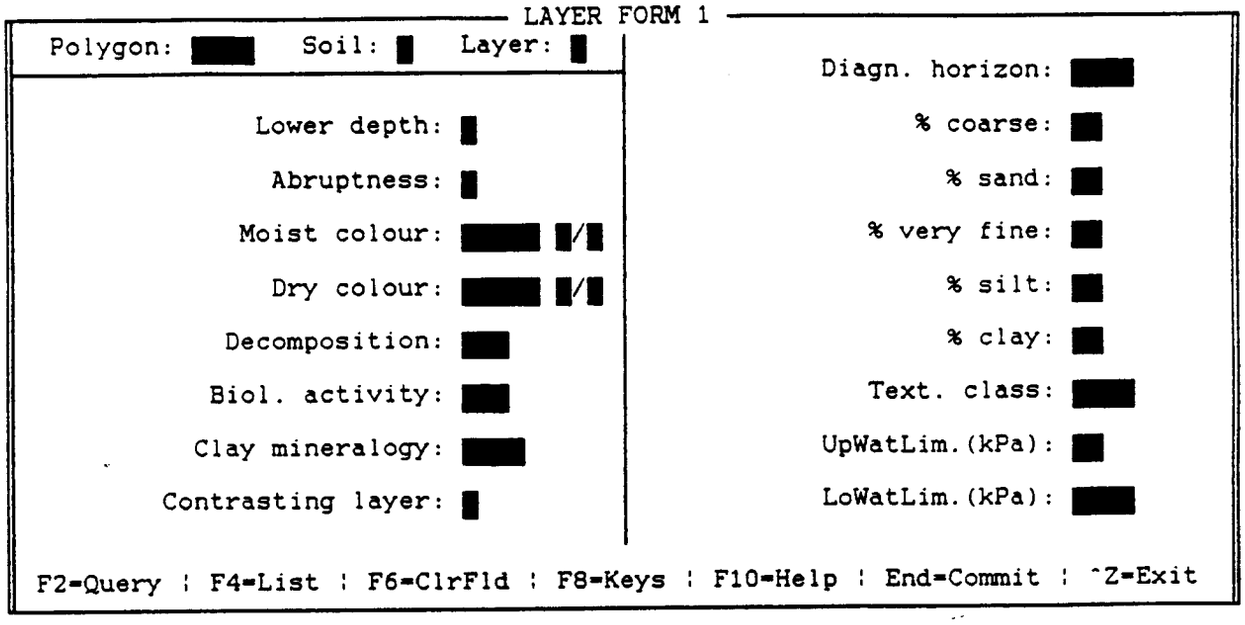
ANNEX D. IAP Screens

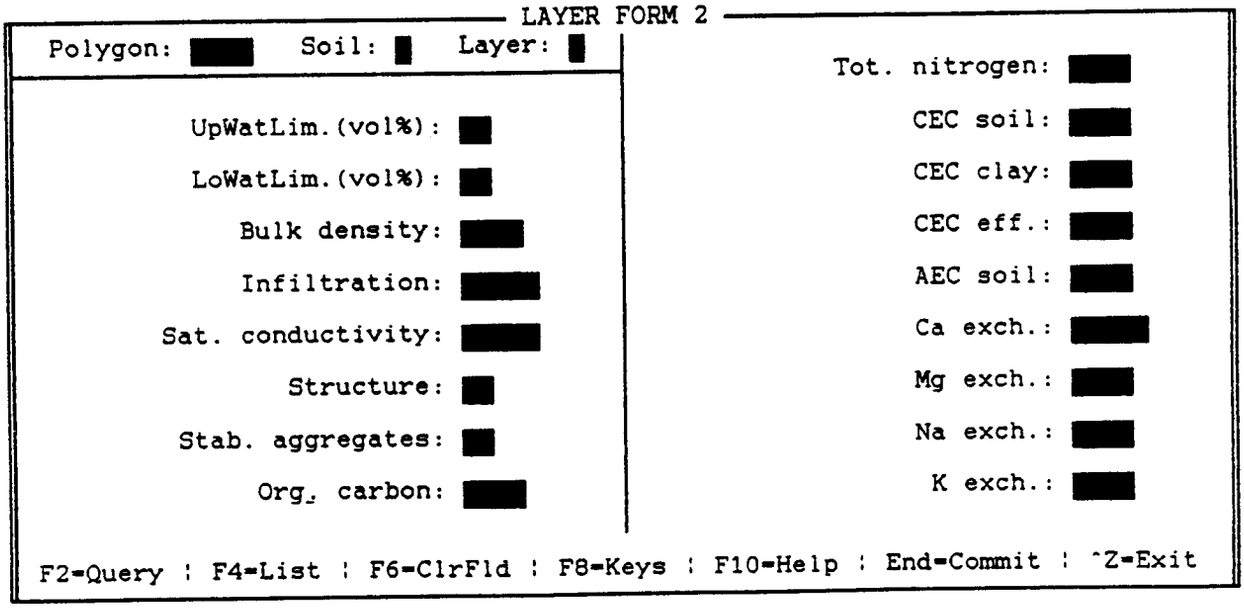


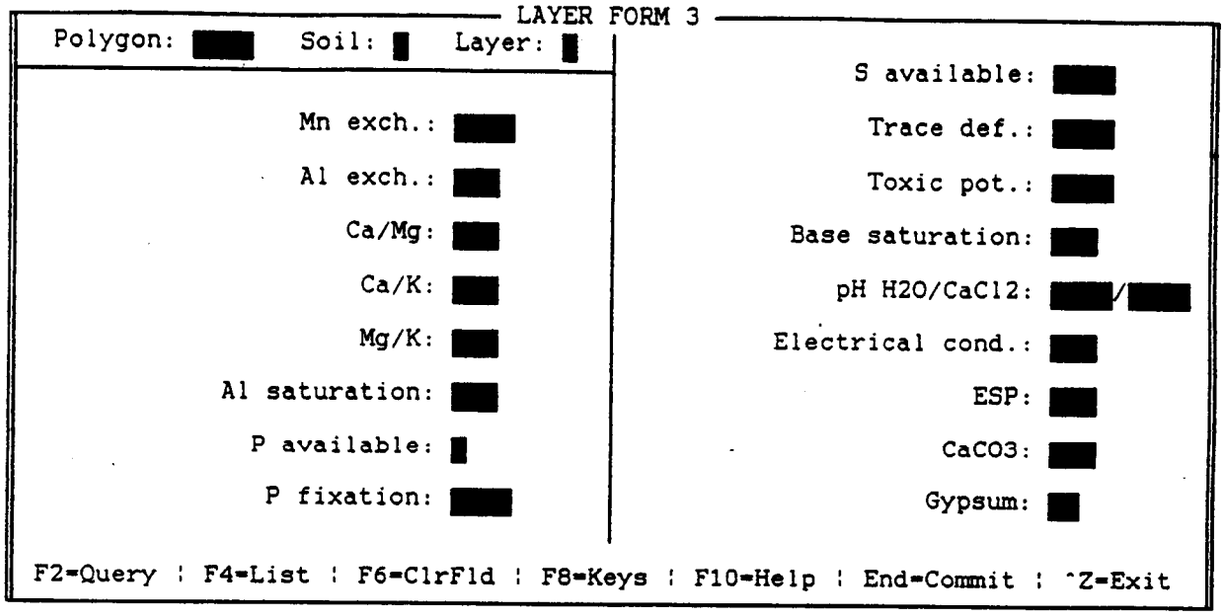



SOIL FORM
Polygon number:
Terrain component:
Soil component:
Proportion:
Slope position:
F2=Query ¦ F4=List ¦ F6=ClrFld ¦ F8=Keys ¦ F10=Help ¦ End=Commit ¦ ^Z=Exit


PROFILE FORM
Polygon number:
Soil number:
Internal drainage:
Classification system:
Soil development:
Reference pedon:
F2=Query ¦ F4=List ¦ F6=ClrFld ¦ F8=Keys ¦ F10=Help ¦ End=Commit ¦ ^Z=Exit


LAYER FORM 1
Polygon:
Soil:
Layer:
Lower depth:
Abruptness:
Moist colour:
Dry colour:
Decomposition:
Biol. activity:
Clay mineralogy:
Contrasting layer:
Diagn. horizon:
% coarse:
% sand:
% very fine:
% silt:
% clay:
Text. class:
UpWatLim.(kPa):
LoWatLim.(kPa):
F2=Query ¦ F4=List ¦ F6=ClrFld ¦ F8=Keys ¦ F10=Help ¦ End=Commit ¦ ^Z=Exit


LAYER FORM 2
Polygon:
Soil:
Layer:
UpWatLim.(vol%):
LoWatLim.(vol%):
Bulk density:
Infiltration:
Sat. conductivity:
Structure:
Stab. aggregates:
Org. carbon:
Tot. nitrogen:
CEC soil:
CEC clay:
CEC eff.:
AEC soil:
Ca exch.:
Mg exch.:
Na exch.:
K exch.:
F2=Query ¦ F4=List ¦ F6=ClrFld ¦ F8=Keys ¦ F10=Help ¦ End=Commit ¦ ^Z=Exit


LAYER FORM 3
Polygon:
Soil:
Layer:
Mn exch.:
Al exch.:
Ca/Mg:
Ca/K:
Mg/K:
Al saturation:
P available:
P fixation:
S available:
Trace def.:
Toxic pot.:
Base saturation:
pH H2O/CaCl2:
Electrical cond.:
ESP:
CaCO3:
Gypsum:
F2=Query ¦ F4=List ¦ F6=ClrFld ¦ F8=Keys ¦ F10=Help ¦ End=Commit ¦ ^Z=Exit

```
/***
*oradef.h - definitions for ORACLE interface routines.
*
*    Copyright (c) 1988, J.H.M. Pulles.
*
*Purpose:
*    Defines the struct of an ORACLE cursor data area which is
*    used by ORACLE interface functions; values for ORACLE
*    constants and masks; provides function prototypes for ORACLE
*    Interface calls.
*
*******************************************************************/

#ifndef NO_EXT_KEYS /* extensions enabled */
     #define _CDECL cdecl
     #define _NEAR  near
#else /* extensions not enabled */
     #define _CDECL
     #define _NEAR
#endif /* NO_EXT_KEYS */


/* definition of ORACLE cursor data area */

struct oracle_area
{
     int ret_code;              /* return code for operation */
     int func_type;             /* function type of SQL command */
     long row_count;            /* number of rows processed */
     int parserr_offset;        /* offset of parse error */
     char func_code;            /* code for requested OCI call */
     char fill_char;
     int cursor_nr;             /* ORACLE internal cursor number */
     int v4_err;                /* error message number */
     char flag1;                /* 1st warning flag */
     char flag2;                /* 2nd warning flag */
     char int_rowid[13];        /* row ID in internal format */
     char osd_err[4];
     char chk_byte;
     char ora_parm[28];         /* used internally by ORACLE */
};


/* Masks for warnings of flag1 */

#define   IS_WARNING           0x01
#define   TRUNC_ON_FETCH       0x02 /* data truncated on fetch */
#define   NULL_VALUES          0x04
#define   NO_WHERE_CLAUSE      0x10 /* at UPDATE or DELETE op. */
#define   ROLLBACK_PERF        0x40
#define   INCONSISTENT         0X80

/* Masks for warnings of flag2 */

#define   ORES_POSSIBLE        0x01
#define   FATAL_ERROR          0x02
```

```
#define   ROW_LEVEL_ROLLB      0x04
#define   NO_AUDIT             0x08


/* code values for data types */

#define   VAR_STRING           1
#define   INT_NUMERIC          2
#define   INT                  3
#define   NULL_STRING          5
#define   RAW_DATA             6
#define   LONG                 8
#define   UNKNOWN              10
#define   INT_ROWID            11
#define   INT_DATE             12


/* values for constants used in SQL.ORA */

#define   M_IDEN               30
#define   S_DTFL               6


/* 'Oracle Call Interface' function prototypes */

int _CDECL olon (struct oracle_area *, char *, int, char *, int,
     int);

int _CDECL oopen (struct oracle_area *, struct oracle_area *,
     int, int, int, char *, int);

int _CDECL osql3 (struct oracle_area *, char *, int);

int _CDECL odsc (struct oracle_area *, int, int *, int *, int *,
     int *, char *, int *, int *);

int _CDECL oname (struct oracle_area *, int, int, int, char *,
     int *);

int _CDECL odefin (struct oracle_area *, int, char *, int, int,
     int, int *, int, int, int, int *, int *);

int _CDECL obndrv (struct oracle_area *, char *, int, char *,
     int, int, int, int *, int, int, int);

int _CDECL obndrn (struct oracle_area *, int, int, int, int, int,
     int *, int, int, int);

int _CDECL oexec (struct oracle_area *);

int _CDECL ofetch (struct oracle_area *);

int _CDECL ocmn (struct oracle_area *, char *, int);

int _CDECL ocof (struct oracle_area *);

int _CDECL ocom (struct oracle_area *);
```

```
int _CDECL ocon (struct oracle_area *);
int _CDECL orol (struct oracle_area *);
int _CDECL oermsg (int, char *);
int _CDECL oclose (struct oracle_area *);
int _CDECL ologof (struct oracle_area *);
int _CDECL oopt (struct oracle_area *, int, int);
int _CDECL ores (struct oracle_area *);
```