Pesticide Use Report Loading and Error-Handling Processes

By

Larry Wilhoit

January 2002

California Department of Pesticide Regulation
Pest Management and Licensing Branch
Pest Management Analysis and Planning Program
1001 I Street, Sacramento, California 95812

PM 02–01

Table of Contents

List of Tables and Figures

Introduction

The California Department of Pesticide Regulation's (DPR) Pesticide Use Report (PUR) is probably the largest and most complete database on pesticide use in the world. A system to collect information on California pesticide use has been in operation in some form for more than 40 years, but the current, full use reporting system began in 1990. The PUR database discussed in this report refers only to the data collected since 1990.

The PUR contains information on nearly all production agricultural pesticide use and on some non-agricultural use in California. The data collected includes what pesticide product was used, the date it was applied, which field was treated, and location to a square-mile section. Production agricultural use includes applications to growing crops, agricultural fields, and most applications to forest trees and ornamental turf. For brevity, these uses will all be referred to as "agricultural uses." Other pesticide uses reported to DPR include post-harvest commodity treatments, rights of ways, landscapes, structural use, and other non-agricultural uses by commercial applicators. These heterogeneous applications will be referred to as "non-agricultural use."

DPR expanded pesticide use collection in 1990 primarily to more accurately assess dietary risks from pesticide exposure. However, the PUR is now used for a wide variety of environmental and public health purposes, including risk assessments, promoting farm worker health and safety, analyzing human exposure patterns, protecting threatened and endangered species, monitoring and investigating environmental issues, and improving pest management. The PUR is used extensively by state and U.S. governments, universities, farmer organizations, the pesticide industry, and public interest groups.

This report describes the operation of only one part of this data collection system. It provides some of the technical details of the computer program and procedures for loading PUR data received from the counties into an Oracle database and checking the data for errors. This program is known as the loader program.

The PUR Process

To understand the function of the loader program, it will help to understand the larger PUR process that it is part of. Data collection starts with pesticide users filling out a form to report each pesticide application. Most of these forms are paper but some are electronic. These reports are sent to the county where staff enter the data into a local database. Every few weeks most counties produce a text file containing the latest data collected. This file is either emailed or copied to a floppy disk and mailed to the state headquarters at DPR. At DPR, the loader program runs every day and checks if any new county data files have been received. If the program finds a new file the data are run through a series of error checks. Error-free data are entered into a database table and errors are recorded in another table. The error table is used to generate an error report, listing each error. This is printed and sent back to the county where the data came from. Staff at the county must try to correct each error in this error report. This usually means that they must find the original pesticide use report submitted by the pesticide user. The correction is made to the county database and marked on the printed error report, which is sent back to the DPR where staff make the correction to DPR database.

Generally the counties send the PUR data to DPR every few weeks, but sometimes it can take several months. By April, DPR should have nearly all the county data from the previous year. However, the error corrections take several more months. The final version of the PUR is typically available in December and contains the prior year's data.

The loader process refers to the movement of the PUR data from the county file into a central DPR database and the identification of errors in the data. One of the most complex parts of the PUR process is the error-handling process, which includes not only the error identification and tracking carried out by the loader program but also the error correction process.

The Importance of Error-handling

Because of the importance of the PUR for many groups and individuals, it is critical that the database be as accurate and complete as possible. People who use the data need to feel confident that the data are sufficiently accurate for their purposes.

Bad data are worse than no data. If a system fails to minimize errors, two serious problems will arise. Some people will use the data not knowing about the errors and, therefore, make wrong conclusions. If laws and regulations are based on these conclusions, there is the potential for serious consequences. Other people will not trust the data and therefore ignore it. In any case, the huge effort at data collection is wasted.

More than 50,000 errors (about 2% of the total number of records) are found each year in the PUR data. Most of these detected errors are corrected before the final version of the PUR is available. However, not all errors can be identified and so the true error rate is unknown. Also, even a 0.1% error rate in pounds applied, if errors are large, could seriously affect some kinds of conclusions.

Although this report describes the error-handling process at DPR, it is important to realize that most error-handling should occur at the time of data entry. If paper reporting forms are used, these forms should be as clear as possible to minimize errors due to misunderstandings. The people filling out the forms should be instructed on the proper procedures. Computer-based forms provide a much more powerful mechanism for preventing errors from being entered because the computer program can check for many possible errors and prevent the entry of invalid values.

Another step in the PUR process is data entry from the PUR reports into the county database. This provides another opportunity for creating errors. Although, the county data entry programs do carry out some error checking routines, many errors are not caught and remain in the data that DPR receives. On the other hand, most county staff have a good understanding of the PUR process and requirements, and they are invaluable in finding and correcting errors in the reports.

Because of insufficient error-handling procedures in the different county data entry programs, incorrect values can be entered in their databases. Thus, another mechanism must exist at DPR to ferret out these errors. Trying to eliminate and correct errors at this stage in the overall process is much more difficult, much more expensive, and much less successful than preventing errors in the first place. It may be months before county staff find time to investigate the errors, at which time much information may have been forgotten or lost. Also, as time passes people are less motivated to make the corrections.

The difficulties in handling errors at this late stage should become evident from the description of the process in this report.

Overview of the Loader Program

Conventions

This report uses a few conventions to help the reader understand the processes. Database table names are in bold capitals; file names are in italics; the column names in a table are in Arial type. The columns are also called "fields", which should not be confused with agricultural fields, which are always referred to as "agricultural fields". A database record is just a row in a table. As mentioned above, the PUR contains two primary types of records: production agricultural applications and all others. For brevity these are referred to as agricultural applications and non-agricultural applications, respectively.

PUR data loading procedures

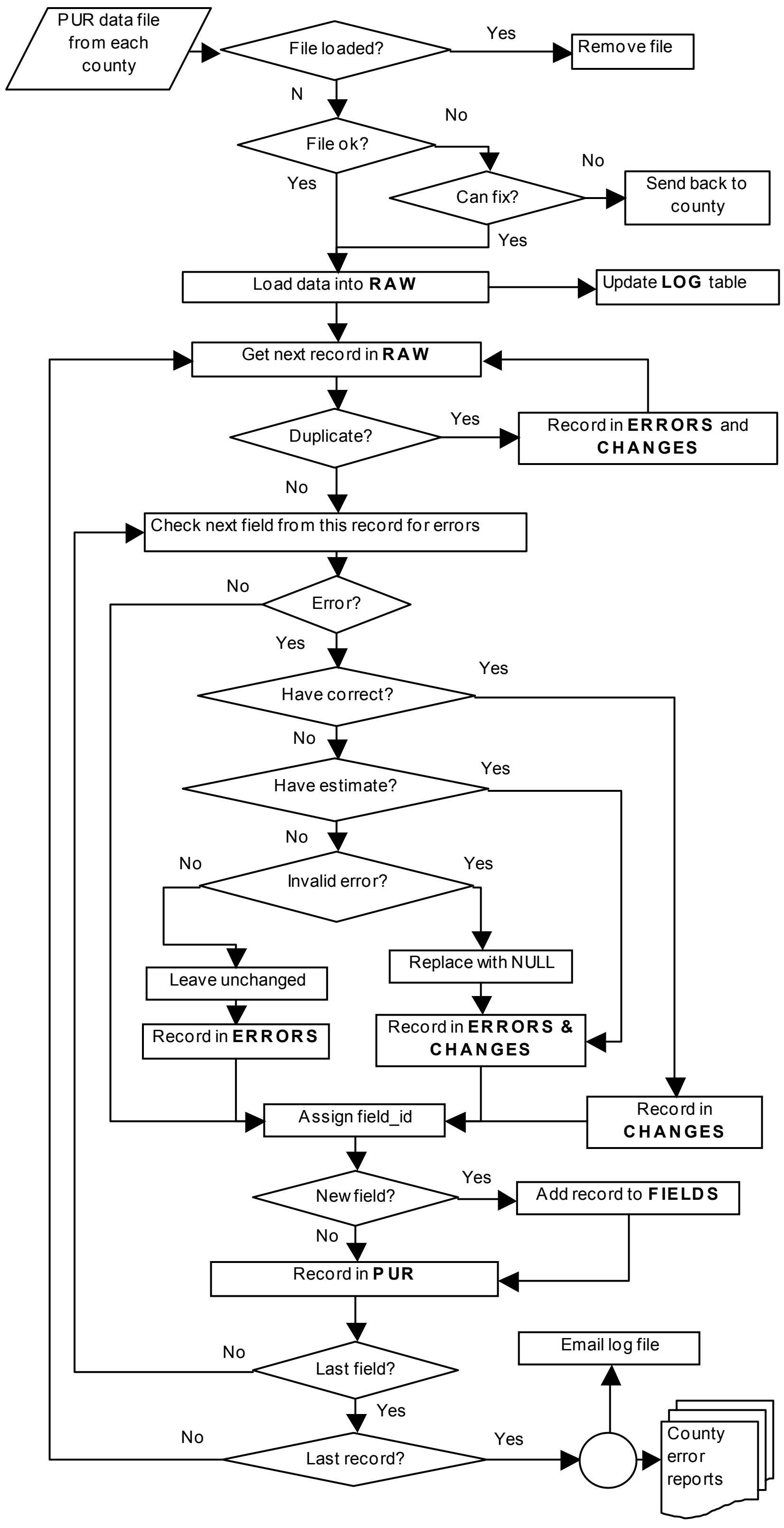
The loader program runs on a UNIX server every day checking for any newly received county PUR data files. When it finds a new file it runs through several steps (Figure 1):

1. checks that data from a PUR data file with the same name as the new file has not already been loaded;
2. checks that the new data file has the correct structure and fixes certain kinds of errors;
3. loads all the data from the new file directly into an Oracle table (**RAW**);
4. logs the name and date of the county file, number of rows of data entered, and the date loaded (**LOG**);
5. checks that each record in the new file is not an erroneous duplicate of records already loaded (not all duplicate records are errors);
6. checks each data field in this table for a series of possible errors; these error checks are described in the next section;
7. where possible, corrects errors or makes estimates to replace invalid data, otherwise replaces the value with a blank or leaves it unchanged;
8. records any uncorrected errors it finds, including both the original value and new value, and the kind of error (**ERRORS**);

9. records any changes made to the data, the date of the change, and whether it was corrected, estimated, or replaced with a null (**CHANGES**);
10. identifies each agricultural field, assigns it a field_id, determines the most likely acres planted and location of the field, and creates a record of this agricultural field (**FIELDS**);
11. if the agricultural field has records in the **PUR** with different values reported for its acres planted or location, the agricultural field is marked as inconsistent in **FIELDS** and a list of all inconsistent values of acres planted and location is made for this field (**FIELD_MTRS_ACRES**);
12. makes some calculations and conversions (such as getting the DPR product number from the pesticide registration number and the pounds of product used from the amount of product used);
13. loads the valid and converted data into another table (**PUR**);
14. creates and prints an error report with all the errors found in the PUR data; and
15. emails loader log files to the loader administrators; these report the data files successfully loaded, the files not loaded because of errors, and any database or operating system error messages that may have been produced during loading.

A more complete description of this process is given in the section “What the loader program does”.

Figure 1. Loader program: Overview of procedures to load pesticide use data into DPR database
Database table names in bold capitals



The database tables
The loader program uses, creates, and updates several database tables. It uses several tables from DPR's pesticide label database, such as **PRODUCT**, which lists the names, product numbers (prodno), registration numbers, registration dates, inactive dates if any, and some other information for all California registered pesticide products; **CHEMICAL**, which lists all active ingredients that appear in products; **PROD_CHEM**, which lists the active ingredients in each pesticide product; **SITE**, which lists all the sites that can appear on pesticide labels; and **PROD_SITE**, which lists all sites that appear on the label of each product. It creates and updates other tables, which are described here (and whose fields are described in Tables 1A - D):

PUR: contains all processed and corrected PUR data. It should be used for all official analyses of the PUR data. In order to identify the PUR version being used, analyses should include the date that the table was last updated. This table will also be used to produce annual PUR summary reports. The "preliminary" report will be produced in June and will include all data received and processed by that time. The "final" report will be produced in December and will incorporate the majority of error corrections for that year's data. The December report is called the "final" version because it is the last published report, not because no more errors corrections will be made. This table will continue to be updated as errors are detected and corrected.

RAW: contains original data as it was received from the counties. No changes are made to these data. This table is used to create the other PUR tables and would not normally be used in analyses; previous years' tables would be archived.

ERRORS: contains the current list of all errors found in the PUR data. When values are corrected, the corresponding record in **ERRORS** is deleted.

CHANGES: contains a list of all the changes that have been made to the **PUR** and the time the change was made. Records are never deleted from **CHANGES**, so that it provides a complete history of all changes made to the **PUR**. Thus, this table could be used to recreate an analysis using any particular version of the **PUR**.

LOG: contains a list of all county files that have been loaded into the **PUR**, the number of rows from each file, and the date each file was loaded.

FIELDS: contains a list of each agricultural field in the PUR, identified by the combination of county_cd, grower_id, site_loc_id, and site_code. It contains information about the field such as the acres planted, its location, its region, and a flag that indicates whether different applications on this field reported different values for acres planted or location. This table should be used to calculate agricultural field level statistics and total acres planted.

FIELD_MTRS_ACRES: If a field has inconsistent acres planted or location, records are created in this table listing the field_id, each MTRS and acres planted reported for this field, and the number of records reported for each combination of MTRS and acres planted. This table would only be used to investigate possible errors in the data.

PUR_SITES: contains all the site or commodity codes and names used in the PUR. This table is a subset of the label database table **SITE,** because not all sites that can appear on pesticide labels are used for use reporting. The field extra contains an 'X' for sites that are not in the list of sites that should be used in the PUR but that were used in some of the earlier years of the PUR. Also, the site names are those used in the PUR explanations given to the counties, which differ somewhat from the names used in **SITE**.

PUR_SITE_QUALIFIER: contains the same sites listed in **PUR_SITES**, but includes the qualifier codes used with each site.

There are a few other tables that are used internally by the loader and error-checking programs. **RAWI, ERRORSI,** and **CHANGESI** are intermediate tables used to temporarily store the data from the current county file that is being loaded and error checked.

Table 1A. All data fields (columns) in the Oracle tables associated with the PUR: **PUR**.

Column name	Type	Description
acre_planted	NUMBER(9, 2)	Size of field that the application occurred on
acre_treated	NUMBER(9, 2)	Size of area or volume treated
aer_gnd_ind	VARCHAR2(1)	Method of application (air, ground, or other)
amt_prd_used	NUMBER(12, 4)	Amount of product applied, units given by unit_of_meas
applic_cnt	NUMBER(6)	Number of pesticide applications for this record
applic_dt	DATE	Date pesticide was applied
applic_time	VARCHAR2(4)	Time of day of application
base_ln_mer	VARCHAR2(1)	Baseline & meridian for application location
batch_no	NUMBER(4)	Number assigned by county to a group of PUR reports
cedts_ind	VARCHAR2(1)	Indicates if report was submitted using the CEDTS system
corr_cntr	NUMBER(1)	Not currently used
county_cd	VARCHAR2(2)	County code for county where pesticide was applied
document_no	VARCHAR2(8)	Number of report in a batch of reports
field_id	NUMBER(8)	System assigned number uniquely identifying each agricultural field
grower_id	VARCHAR2(11)	Value that uniquely identifies each grower or operator
grwr_fut_suf	VARCHAR2(1)	Not currently used
last_up_dt	DATE	Last time any field in this record was changed
lbs_prd_used	NUMBER(14, 4)	Pounds of product applied
license_no	VARCHAR2(13)	License number of pest control operator or business
mtrs	mtrs_type	Meridian-township-range-section value
nursery_ind	VARCHAR2(1)	Not currently used
planting_seq	NUMBER(1)	Number to indicate multiple plantings of a crop on a field
process_mt	NUMBER(2)	Month the PUR report was processed by county
process_yr	NUMBER(4)	Year the PUR report was processed by county
prodno	NUMBER(7)	System assigned pesticide product number
qc_flag1	VARCHAR2(1)	Not currently used
qc_flag2	VARCHAR2(1)	Not currently used
qualify_cd	NUMBER(2)	Code that identifies commodity more precisely than site_code
range	VARCHAR2(2)	Range number for application number
range_dir	VARCHAR2(1)	Range direction for application number
record_id	VARCHAR2(1)	Value indicating the kind of PUR report
section	VARCHAR2(2)	Section number for application number
site_code	NUMBER(6)	Code of site or commodity that was treated
site_loc_id	VARCHAR2(8)	Value that indicates the particular field treated
summary_cd	NUMBER(4)	Line number on PUR report
township	VARCHAR2(2)	Township number for application number
tship_dir	VARCHAR2(1)	Township direction for application number
unit_of_meas	VARCHAR2(2)	Unit of measure for amount product used
unit_planted	VARCHAR2(1)	Unit of area for planted field (acres, square feet, etc)
unit_treated	VARCHAR2(1)	Unit of area or volume treated (acres, cubic feet, etc)
use_no	NUMBER(8)	System assigned number to uniquely identify each record
year	NUMBER(4)	Year of application

Table 1B. All data fields (columns) in the Oracle tables associated with the PUR: **RAW**.

Column name	Type	Description
use_no	NUMBER(8)	System assigned number to uniquely identify each record
record_id	VARCHAR2(1)	Value indicating the kind of PUR report
process_mt	VARCHAR2(2)	Month the PUR report was processed by county
process_yr	VARCHAR2(2)	Year the PUR report was processed by county
batch_no	VARCHAR2(4)	Number assigned by county to a group of PUR data files
nursery_ind	VARCHAR2(1)	Not currently used
county_cd	VARCHAR2(2)	County code for county where pesticide was applied
section	VARCHAR2(2)	Section number for application number
township	VARCHAR2(2)	Township number for application number
tship_dir	VARCHAR2(1)	Township direction for application number
range	VARCHAR2(2)	Range number for application number
range_dir	VARCHAR2(1)	Range direction for application number
base_ln_mer	VARCHAR2(1)	Baseline & meridian for application location
aer_gnd_ind	VARCHAR2(1)	Method of application (air, ground, or other)
grower_id	VARCHAR2(11)	Value that uniquely identifies each grower or operator
cedts_ind	VARCHAR2(1)	Indicates if report was submitted using the CEDTS system
site_loc_id	VARCHAR2(8)	Value that indicates the particular field treated
acre_planted	VARCHAR2(8)	Size of field that the application occurred on
unit_planted	VARCHAR2(1)	Unit of area for planted field (acres, square feet, etc)
applic_dt	VARCHAR2(6)	Date pesticide was applied
site_code	VARCHAR2(6)	Code of site or commodity that was treated
qualify_cd	VARCHAR2(2)	Code that identifies commodity more precisely than site_code
planting_seq	VARCHAR2(1)	Number to indicate multiple plantings of a crop on a field
acre_treated	VARCHAR2(8)	Size of area or volume treated
unit_treated	VARCHAR2(1)	Unit of area or volume treated (acres, cubic feet, etc)
mfg_firmno	VARCHAR2(7)	Manufacturing firm number, component of registration number
label_seq_no	VARCHAR2(5)	Label sequence number, component of registration number
revision_no	VARCHAR2(2)	Revision code, component of registration number
reg_firmno	VARCHAR2(7)	Registrant firm number, component of registration number
amt_prd_used	VARCHAR2(10)	Amount of product applied, units given by unit_of_meas
unit_of_meas	VARCHAR2(2)	Unit of measure for amount product used
document_no	VARCHAR2(8)	Number of report in a batch of reports
summary_cd	VARCHAR2(4)	Line number on PUR report
applic_cnt	VARCHAR2(6)	Number of pesticide applications for this record
applic_time	VARCHAR2(4)	Time of day of application
license_no	VARCHAR2(13)	License number of pest control operator or business
file_date	DATE	Date PUR county file was created
file_name	VARCHAR2(12)	Name of PUR county file

Table 1C. All fields (columns) in the Oracle tables associated with the PUR: **ERRORS, CHANGES, LOG, FIELDS, and FIELD_MTRS_ACRES**.

Fields in ERRORS table

Column name	Type	Description
transaction_no	NUMBER	Uniquely identifies row in errors table
use_no	NUMBER(10)	Foreign key to the PUR tables
error_code	NUMBER(3)	Error code for use by counties
column_name	VARCHAR2(50)	Name of column or field in PUR
old_value	VARCHAR2(50)	Original value in this column (the one with error)
new_value	VARCHAR2(50)	Value used to replace old_value (if do not know correct value or estimate, leave NULL)
duplicate_set	NUMBER(13)	If this record is an error duplicate of other records (error_code = 80), this field gives the number to identify what set of duplicate records this record is in.
error_type	VARCHAR2(20)	"invalid" (value not allowed in this column), "probable" (value allowed but incorrect), "possible" (value may be incorrect), "duplicate" (this row is a duplicate of another one)
replace_type	VARCHAR2(20)	"null" (replace with null) "estimate" (replace with estimate or good guess) "same" (do not replace, leave value same) "delete" (delete entire row),
require_type	VARCHAR2(20)	"required" (value is required for this column) "optional" (value is optional) "not_allowed" (value is not allowed) "unknown" (value may or may not be required)
who	VARCHAR2(30)	Who or which program found the error; if loader, include version number
comments	VARCHAR2(2000)	Comments about the error

Fields in CHANGES table

Column name	Type	Description
transaction_no	NUMBER	Uniquely identifies row in changes table
use_no	NUMBER(10)	Foreign key to the PUR tables
error_code	NUMBER(3)	Error code for use by counties
column_name	VARCHAR2(50)	Name of column or field in PUR
old_value	VARCHAR2(50)	Previous value in this column
new_value	VARCHAR2(50)	New or corrected value
action_taken	VARCHAR2(20)	"null" (replace with null) "estimate" (guess correct value), "correct" (change to correct value), "delete" (delete row), "insert" (insert row), "validate" (indicates that a possible error is correct)
action_time	DATE	Date and time that the tables were updated
who	VARCHAR2(30)	Who made the change; if loader, include version number
county_validated	VARCHAR2(1)	Whether or not the correction was validated by county: Y or N
comments	VARCHAR2(2000)	Comments about the change

Fields in LOG table

Column name	Type	Description
file_name	VARCHAR2(12)	Name of file from county
file_date	DATE	Date of county file
load_date	DATE	Date file loaded
num_of_records	NUMBER(7)	Number of records loaded
start_use_no	NUMBER(8)	First use_no of the records in RAW table
end_use_no	NUMBER(8)	Last use_no

Fields in FIELDS table

Column name	Type	Description
field_id	NUMBER(8)	Unique number assigned by the loader program to identify each agricultural field
county_cd	VARCHAR2(2)	County code
operator_id	VARCHAR2(7)	Operator id, taken from the grower_id
site_loc_id	VARCHAR2(8)	Grower assigned name of the field
site_code	NUMBER(6)	Crop or commodity code
mtrs	mtrs_type	The geographical location of the field using MTRS; if different applications report different values for this field, the value chosen is the one with the largest number of records. Mtrs_type is a user defined type containing the components base_ln_mer, township, tship_dir, range, range_dir, and section. Each component can be identified, for example, by mtrs.base_ln_mer, mtrs.township, etc. You can also get the MTRS value as a single string using mtrs.Get_value().
acres_planted	NUMBER(92)	The acres planted for this field. The unit used is always acres so there is not need for another field for units. If different applications report different values for this field, the value chosen is the one with the largest number of records.
region	VARCHAR2(30)	The name of the region in California where the field is located.
inconsistent_field	VARCHAR2(1)	A flag (an 'X') to indicate that different applications on this field report different acres planted or MTRS.

Fields in FIELD_MTRS_ACRES table

field_id	NUMBER(8)	Unique number assigned by the program to identify each agricultural field. This table lists only inconsistent fields, that is, fields with more than one reported value for acres planted or MTRS. These fields have an 'X' in the inconsistent_field column in FIELDS.
mtrs	mtrs_type	Each MTRS value reported in the PUR for this field.
acres_planted	NUMBER(10,2)	Each acres planted reported in the PUR for this field.
num_recs	NUMBER(12)	The number of records in the PUR with this combination of MTRS and acres planted.

Table 1D. All fields (columns) in the Oracle tables associated with the PUR: **PUR_SITE_QUALIFIER, PUR_SITES**. These tables are subsets of table **SITE** containing only sites that are actually used in the PUR. Also, names are the ones used in the PUR explanations given to the counties.

Fields in PUR_SITE_QUALIFIER table

Column name	Type	Description
site_code	NUMBER(6)	Code for site or commodity that was treated
qualify_cd	NUMBER(2)	Code for additional site specificity
site_name	VARCHAR2(50)	Name of site or commodity
extra	VARCHAR2(1)	An 'X' indicates a site that is not in the list of sites that should be used in PUR reporting but which have appeared in earlier PUR years.

Fields in PUR_SITES table

Column name	Type	Description
site_code	NUMBER(6)	Code for site or commodity that was treated
site_name	VARCHAR2(50)	Name of site or commodity
extra	VARCHAR2(1)	An 'X' indicates a site that is not in the list of sites that should be used in PUR reporting but which have appeared in earlier PUR years.

Error-handling Procedures

Types of Errors and How They Are Handled

The most complex job of the loader program is checking for errors in the PUR data. These errors are classified in a number of ways. **Invalid** errors are values that are not allowed for a field, that is, are not in the set of valid values; for example, a character in a number field, or a unit treated that is not one of the valid values such as A (acres), S (square feet), etc. **Probable** errors are values that are in the set of valid values for a field, but are almost surely wrong, for example, a product registration number that is not in our label database (this could be a correct registration number, the error being in the label database), or an unreasonably high value for the pounds applied. All probable errors, like invalid errors, are reported to the county. **Possible** errors are values that are unusual in some way but not clearly known to be an error, for example, a high value for pounds applied but not so high that it could not have actually been used. Possible errors are not reported to the county, but are left in the **PUR** and **ERRORS** table so they could be identified by anyone wanting to check the data more carefully.

Errors are handled in several ways. (1) If the correct value is known, replace the erroneous value with the correct value. This will probably be an unusual situation, but one example is an error in the first two characters of the grower_id. These characters should be the county code for the county where the pesticide was applied. Since we know the correct county code, we can use that code when there is an error in these characters of the grower_id. (2) If the correct value is unknown but can be guessed or estimated, replace it with the estimate. For example, if the home county code in the grower_id is invalid (does not correspond to one of the county codes) you could assume that it is the same as the county of application, since this is usually true. Another example is estimating an extreme outlier in the rate of use with the median rate for all applications that used this product on this crop or site. (3) If no reasonable estimate can be made and the error is not invalid, leave the value unchanged; this could be considered a kind of estimate. (4) If no reasonable estimate can be made and the value is invalid, replace it with a null. Different characters are used to indicate a null value: '?' is used if the field is a character, -1 if the field is a numeric code for which arithmetic operations are meaningless, and a blank is used if the field is numeric. A '?' cannot be used in a numeric field since it is not a number and '-1' should not be used since it will certainly be an incorrect value and would affect the results of arithmetic operations. An actual character for field and code fields rather than a blank was used to make it easier to link these values to a label such as "UNKNOWN".

An important consideration for determining when values are errors is whether a value is required or not. If a value is not required it may be optional, that is, sometimes reported and sometimes not. In all other cases it is labeled as "not allowed" in the database. Values for some fields are required for all situations, but other values, such as acres planted and geographical location, are required only in certain situations, the most common one being production agricultural reports (identified by record_id equal to 1, 4, A, or B).

Whenever an error is found and is not corrected, whether the error was invalid, probable, or possible, a record of this is made in the **ERRORS** table. Included in the **ERRORS** record is the use_no, which uniquely identifies the record in the PUR; the data field name; the original value reported that was in error; the value actually entered in the **PUR** table; the error type (invalid, probable, or possible); how the value entered was chosen (by estimation, correction, leaving unchanged, or replacing with null); and whether the value was required, optional, or not required. If the value entered into the **PUR** was different from the originally reported value, even if an error record was not created, a record is made in the **CHANGES** table.

All errors found during error checking are compiled into a printed error report and sent to the appropriate county. County staff make corrections on the printed report and send the marked report back to DPR.

Another program is used by DPR staff to make these corrections to the database. This error correction program calls the same error checking procedures as the loader program to check the new values that staff enter. If a value is invalid, the error correction program will not allow the person to enter the value. If a value is a probable or possible error it will let the person know about the problem, but will allow the person to then override that error check. In some cases values marked as probable errors by the loader program will be found to be correct. The error correction program can be used to change any estimated, incorrect values back to their original, correct values and the record for that error in the **ERRORS** table will be deleted. As with all changes, a record will be made in the **CHANGES** table. The error correction program has not yet been completed and is not described here.

Error Checking Procedures

A description of the errors checked for each of the PUR data fields is given in Tables 2 and 3. Table 2 lists the procedures ordered by error_code and provides a short description of each error; Table 3 provides a more detailed explanation of the procedures for each field in the database. Most of the errors will not be described in this text since Table 3 provides sufficient explanation. However, a few error-checking procedures are rather complex and require further explanation (see below).

Table 2. Error codes and descriptions used for the error reports sent to the counties starting in 2000.

Error Code	Error Description
1	record id is not 1, 2, 4, A, B, or C
2	batch no. is not a number or is 0
3	process month is not a number or not between 1 and 12
4	process year is not a number, is less than 1998, or is greater than 2000
5	county code is not a number or not between 1 and 58
6	section number is not a number or not between 1 and 36
7	township is not a number or not between 1 and 48
8	township direction is not N or S
9	range is not a number or not between 1 and 47
10	range direction is not W or E
11	base meridian is not H, M, or S
12	application method is not A, G, or O
13	commodity code is not a number
14	commodity code is not in commodity table
15	application month is not a number or not between 1 and 12
16	application day is not a number or not between 1 and 31
17	application year is not the current year
18	acres treated is not a number or is 0 where a value is required
19	unit treated is not A, S, C, K, P, T, or U, or is not valid for this type of record
22	area treated is greater than 700 acres or equals 999999 square feet for non production ag
23	area treated is greater than the area of its section
24	registration number (firm) is not a number or is 0
25	registration number (label) is not a number or is 0
30	number of applications is not a number
31	amount used is not a number or is 0
32	unit of measure is not LB, OZ, GA, QT, PT, KG, GR, LI, or ML
34	document number is not a number
35	line item is not a number or is 0
37	product is not in the label database
38	unit of measure is invalid for this formulation
39	commodity is not on the list of commodities for which this pesticide is registered
43	grower id is invalid
44	acres planted is not a number or is 0
45	unit planted is not A, S, C, K, or U
47	area treated is greater than the area planted
48	township-range-section does not exist in this county
51	application date is invalid (e.g. day is not in this month or it is after the current date)
52	specific gravity < 0.0001 if formulation is wet (internal use only)
60	area planted is greater than the area for that section
61	unit treated is not compatible with the unit planted
62	site_loc_id has trailing or leading spaces or missing for production ag record
63	grower_id -site_loc_id-site_code has inconsistent values for MTRS or acre_planted
64	cedits indicator is not E, 0, or space
65	qualify code is not a number

Error Code	Error Description
66	planting sequence is not a number
67	base line meridian-township-range-section does not exist in California
68	application time is not a number or is an invalid time
69	license_no is not valid when no grower id is given
70	pounds of product divided by the acre treated is greater than the first outlier limit
71	pounds of product divided by the acre treated is greater than the second outlier limit
72	pounds of product divided by the acre treated is greater than the third outlier limit
75	rate of use (pounds of product divided by the acres treated) is high
80	record is an erroneous duplicate of another record

Table 3. Description of error checking procedures used in the PUR loader program. The error code is an arbitrary number assigned to each kind of error check. Field is the name of the field in the PUR that is checked. Requirements indicate whether a value is required for this PUR field. Error type indicates whether errors found by this check are invalid, probable, or only possible errors. Replacement indicates whether an error value was replaced with a null character ('?', -1, or null), an estimate of the value, a correct value, or left unchanged.

Error Code	Field	Requirements	Error type	Replace	Error Description
1	record_id	required	invalid	?	Record id is not 1, 2, 4, A, B, or C.
5	county_cd	required	invalid	?	County code is not a number or not between 1 and 58.
13	site_code	required	invalid	-1	Commodity code is not a number.
14	site_code	required	invalid	-1	Commodity code is not in the commodity table.
65	qualify_cd	optional	invalid	-1	Qualifier code is not a number
66	planting_seq	optional for production ag.	invalid	-1	Planting sequence is not a number. Value '0' is replaced with null.
24	mfg_firmno	required	invalid	null	Registration firm number is not a number or is 0
25	label_seq_no	required	invalid	null	Registration label sequence number is not a number or is 0
NA	revision_no	optional	NA	null	Revision code is not between AA and ZZ but no errors are reported. People often do not know the revision code because it is not on some labels. In this case, a default value of AA is used. This may or may not be correct but it is looked at in the check for prodno.
NA	reg_firmno	optional	NA	null	Sub registration number is not a number or is 0. Handled same as for revision code.
37	prodno	required	invalid or possible	-1	Pesticide product is not in DPR's label database. If the program can't find a product with the exact given registration number, it will look for a product with the same 2-part or 3-part registration number and report possible error. If it does not find such a product, it will give an invalid error. If there are several such products, it will choose a product based on several criteria discussed in the loader documentation.

Error Code	Field	Requirements	Error type	Replace	Error Description
39	prodno, site_code	required	invalid or possible	no change	Site code is not on the list of commodities for which this product is registered. This will generate an error message only if prodno is not null, site_code >= 1000, product is not a spreader/sticker, fumigant, rodenticide, or exempt from mill assessment. Possible error occurs where site code is not listed for reported product, but is found for another product with the same 2 or 3-part registration number as reported product.
52	spec_gravity	required	invalid	NA	Specific gravity < 0.0001 if product formulation is wet. Specific gravity value comes from the label database, so it is an error in the label database and is not reported to the counties.
31	amt_prd_used	required	invalid	null	Amount used is not a number or is 0
32	unit_of_meas	required	invalid	?	Unit of measure is not LB, OZ, GA, QT, PT, KG, GR, LI, or ML
38	unit_of_meas	required	probable	no change	Unit of measure is invalid for this formulation; changing this requires that lbs_prd_used be recalculated
18	acre_treated	required if prod ag; required if non-ag with site_code > 100; others optional	invalid	null	Acres treated is null or is not a number or <= 0 when it is required; or acres treated is null and unit treated contains a valid value, when it is optional.
22	acre_treated	required if prod ag; required if non-ag with site_code > 100; others optional	probable	estimate	For non production ag, acres treated is > 700 (unit = A) or acre_treated > 999990 (unit = S) and site is not rangeland, pastureland, forest trees, uncultivated ag, or uncultivated non-ag areas. If error, set unit_treated = U.
23	acre_treated	required if prod ag; required if non-ag with site_code > 100; others optional	probable	estimate	Acres treated is > 10% more than the area of its section, when unit is acres and site is not rangeland, pastureland, forest trees, uncultivated ag, or uncultivated non-ag areas. If error, set acres treated equal to area of the section.
19	unit_treated	required if prod ag; required if non-ag with site_code > 100; others optional	invalid	?	Unit treated is null or is not A, S, C, K, or U (for prod ag) or A, S, C, K, P, T, or U (for others), when it is required. Unit treated is null and acre treated has valid value when it is optional.

Error Code	Field	Requirements	Error type	Replace	Error Description
44	acre_planted	required for production ag.	invalid	null	Acres planted is null or is not a number or <= 0.
60	acre_planted	required for production ag.	probable	estimate	Acres planted is > 10% more than the area of its section, when unit is acres and site is not rangeland, pastureland, forest trees, uncultivated ag, or uncultivated non-ag areas. If error, set acres treated equal to area of the section.
45	unit_planted	required for production ag.	invalid	?	Unit planted is not A, S, C, K, or U.
47	acre_planted, acre_treated	required for production ag.	probable	estimate	Area (or volume) treated is greater than the area (or volume) planted. If error, set area treated equal to area planted. To do the comparison, covert square feet to acres and thousand cubic feet to cubic feet. Special case is if unit_treated = A (or K) and unit_planted = S (or C) and acre_treated < acre_planted, then assume error is with unit_treated and set unit_treated equal to S (or C).
61	unit_treated, unit_planted	required for production ag.	probable, possible, or invalid	estimate, no change, or ?	Unit treated is not compatible with the unit planted, that is, one unit is a measure of area and other is a measure of volume; or one unit is U and other unit is C, K, P, or T.
6	section	required for production ag.	invalid	?	Section number is not a number or not between 1 and 36
7	township	required for production ag.	invalid	?	Township is not a number or not between 1 and 48
8	tship_dir	required for production ag.	invalid	?	Township direction is not N or S
9	range	required for production ag.	invalid	?	Range is not a number or not between 1 and 47
10	range_dir	required for production ag.	invalid	?	Range direction is not W or E
11	base_ln_mer	required for production ag.	invalid	?	Base meridian is not H, M, or S
67	MTRS	required for production ag.	invalid	?	MTRS does not exist in CA
48	MTRS	required for production ag.	invalid	?	MTRS does not exist in this county
43	grower_id	required for production ag; required for non ag. when license_no is null; others optional	invalid	?, estimate, or correct	Grower_id is null, or first two digits not equal to county_cd, or 2nd and 3rd digits not between last two digits of PUR year - 3 and last two digits of PUR year + 3, or 5th and 6th digits not between 1 and 58. Further explanation of the grower_id is given in the documentation.

Error Code	Field	Requirements	Error type	Replace	Error Description
69	license_no	required for non ag. when grower_id is null; optional for non ag. when grower_id has value	invalid	?	License number is null or equals 0.
62	site_loc_id	required for production ag.	invalid if null; probable if extra spaces	? or correct	Site location id is null. If it has trailing or leading spaces, remove the spaces and mark as a probable but corrected error.
63	site_loc_id	required for production ag.	possible	no change	Agricultural field (as identified by grower_id, site_loc_id, and site_code) has inconsistencies. Different applications on a field (identified by the combination of county_cd, grower_id, site_loc_id, and site_code or by the program assigned field_id) report different values for acres planted or MTRS. Further explanation of inconsistent agricultural fields is given in the documentation.
15	applic_dt	required	invalid	null	Application month is not a number or not between 1 and 12.
16	applic_dt	required	invalid	null	Application day is not a number or not between 1 and 31.
17	applic_dt	required	probable	estimate	Application year is not the PUR year. As with process_yr it is converted to 4 digit value. If it is an error, it is estimated as the PUR year.
51	applic_dt	required	invalid	null	Application date is an invalid date (eg day does not exist for the month) or is later than the current date. This error is checked only if there were no errors found for error codes 15, 16, or 17.
68	applic_time	optional for production ag.	invalid	?	Application time's first two digits is not between 0 and 24 or last two digits is not between 0 and 60. Replace any spaces with 0; replace '0000' with null.
30	applic_cnt	optional for non ag.	invalid	null	Number of applications is not a number or < 0 for non ag; replace reported values of 0, blank, or "." with null. For production ag., applic_cnt is always set equal to 1 regardless of value reported and no error report is generated.
12	aer_gnd_ind	required for production ag.	invalid	?	Application method is not A, G, or O

Error Code	Field	Requirements	Error type	Replace	Error Description
2	batch_no	required	invalid	-1	Batch number is not a number or is 0.
34	document_no	required except for CalTrans records	invalid	-1	Document number is not a number. Document_no is not required for reports from CalTrans, so it is set to null in these cases. CalTrans records are identified as records with record_id = 2 and application day = 28.
35	summary_cd	required	invalid	-1	Line item is not a number or is 0.
3	process_mt	required	invalid	null	Process month is not a number or not between 1 and 12.
4	process_yr	required	invalid	null	Process year is not a number, is less than the PUR year, or is greater than 2 + the PUR year. The value from the county file is a two digit number. It is coverted here to a 4 digit year by assuming a number between 50 and 99 is a year in the 1900's and a number between 0 and 49 is a year in the 2000's.
64	cedts_ind	required	invalid	?	Cedits indicator is not null, 0, or E. Value '0' is replaced with null.
75	lbs_prd_used, acre_treated, unit_treated, amt_prd_used	required if prod ag; required if non-ag with site_code > 100; others optional	probable	estimate	Pounds of product divided by the acre treated (the rate of use) is greater than either the first or second outlier limit. Further explanation of extreme rate values is given in the loader documentation.
72	lbs_prd_used, acre_treated, unit_treated, amt_prd_used	required if prod ag; required if non-ag with site_code > 100; others optional	possible	no change	Pounds of product divided by the acre treated (the rate of use) is greater than the third outlier limit. Further explanation of extreme rate values is given in the loader documentation.
80	county_cd, grower_id, site_loc_id, site_code, qualify_cd, prodno, amt_prd_used, unit_of_meas, acre_treated, unit_treated, acre_planted, unit_planted, applic_dt	required for production ag.	probable	delete	Values for these fields are the same as those for 1or more other records and the sum of acres treated for this set is greater than the acres planted. Further explanation of duplicate records is given in the documentation.

Registration number, prodno, and site_code. The person filling out a pesticide use report enters the registration number for the pesticide used. The loader program checks if the given registration number exists in DPR's label database. If not, it will generate an error record.

However, there may be some difficulties in determining which product was used because the submitted form may not give the full pesticide registration number. The United States Environmental Protection Agency (U.S. EPA) pesticide registration number consists of a firm number that identifies the company that is the primary registrant (mfg_firmno), a number assigned to each company's individual product as it is registered (label_seq_no), and a distributor or sub-registrant's number that identifies any company that is marketing a product owned by another company (reg_firmno). In addition, California requires companies to register and license individual brand names, so the California registration number includes an additional code, which uniquely identifies each brand (revision_no). In DPR's database each product with a distinct California registration number is also given a unique identifying number (prodno).

The pesticide label usually displays only the parts assigned by U.S. EPA. If the reg_firmno is different from the mfg_firmno, the label will display the 3-part registration number (mfg_firmno, label_seq_no, and reg_firmno); if the reg_firmno is the same as the mfg_firmno, the label will display the 2-part registration number (mfg_firmno and label_seq_no). The revision_no usually does not appear on the label so it is often not given in the pesticide use reports. If the revision_no is not given, most county data entry programs will provide a default value of 'AA', which may be incorrect. Note that in the county PUR file a reg_firmno value of '0000000' means that reg_firmno is the same as mfg_firmno; in the label database for these cases, reg_firmno will equal mfg_firmno.

To help determine if a reported registration number is valid, the loader program uses information on the reported site treated and date of application. DPR's database includes tables that list all sites that each product is registered for and the dates during which all products had valid registrations. In some situations determining the most likely exact pesticide product is difficult. For example, a reported registration number with a 'AA' revision_no may exist in the database but the reported site may not be on its label but may be on the label for another product that matches the 2 or 3-part registration number. Since a 'AA' revision_no may have been a default value entered by the computer rather than an actual value entered, it could be incorrect. (Of course, it could be incorrect even if someone entered it, but, hopefully, it is more likely to be correct in that situation.)

In this situation, there are several choices:

1. Leave the registration number as reported and report no errors: this choice leaves an unreported error in the PUR (site not on the label) so is not acceptable.

2. Leave the registration number as reported but report the error “site not on label”: this does not indicate that there exists a similar product which could be the actual product used so we lose information.
3. Change the value to the registration number with the site on its label and report no errors: this does not give feedback to the county (or users) that there was a problem.
4. Change to the registration number with the site on its label but report site not on label: this is not quite accurate since the site is on the chosen product and it does not distinguish this case from the case where the site code is not on any similar products.
5. Leave the registration number as reported but report a possible error that the site is not on its label and give the registration number of the product with the site on its label.

Choice 5 seems to be the most accurate option and to provide the most information, so was used in the current loader program. There are several other difficult situations, but rather than explain all possibilities, we summarize the algorithm used by the loader:

```
First check the existence of the reported product registration number:

IF a product is found in DPR’s label database with the 4-part registration number,
THEN return the prodno and report no errors
ELSE IF only one product is found in DPR’s label database with the three-part
    registration number reported by the applicator (mfg_firmno, label_seq_no, and
    reg_firmno)
THEN return the prodno as an estimate and report possible error 37
ELSE IF more than one product is found in DPR’s label database with the three-part
    registration number reported by the applicator (mfg_firmno, label_seq_no, and
    reg_firmno)
THEN return prodno with a NULL value and report possible error 37 [one of these will
    be chosen below]
ELSE return prodno with value –1 and report invalid error 37.
END IF.

Second, check site on label and choose a prodno:

IF a prodno was found [it will have a positive value] and the reported site code is on its
pesticide label
THEN report no errors
ELSE IF a prodno was found and the site was not on its label
THEN check if site is on the label for other products
    IF site is on any product with the same mfg_firmno and label_seq_no
    THEN report possible error 39 and provide registration number for that product in the
        comments field
    ELSE report invalid error 39
    END IF.
```

```
ELSE IF more than one product was found with the reported 3-part registration number
    [prodno is NULL]
THEN choose one of them
    IF one of the products has a valid specific gravity, has the site on its label, and was
        registered at the time of application,
    THEN report that product as an estimate with possible error 39
    ELSE IF a product has the site on its label,
    THEN report that product as an estimate with possible error 39
    ELSE IF a product has a valid specific gravity,
    THEN report that product as an estimate and check other products for site:
        IF site is on any product with the same mfg_firmno and label_seq_no
        THEN report possible error 39 and provide registration number for that product
        ELSE report invalid error 39
        END IF
    ELSE choose any product in the list, report that product as an estimate, and check
        other products for site:
        IF site is on any product with the same mfg_firmno and label_seq_no
        THEN report possible error 39 and provide registration number for that product
        ELSE report invalid error 39
        END IF
    END IF
ELSE [no product was found with the reported 3-part registration number; prodno = -1]
THEN find all the products in DPR's label database with the same mfg_firmno and
label_seq_no as that of the reported product and choose one:
    IF no product found, THEN report invalid error 39
    ELSE IF a product has a valid specific gravity, has the site on its label, and was
        registered at the time of application,
    THEN report that product as estimate with possible error 39
    ELSE IF a product has the site on its label,
    THEN report that product as estimate with possible error 39
    ELSE IF a product has a valid specific gravity,
    THEN report that product as estimate with invalid error 39
    ELSE choose any product in the list, report that product as estimate with invalid error
        39
    END IF
END IF
```

There are a few details in determining if a site_code is on the list of legal sites on the pesticide product label. All sites with site_code < 1000, which are the nursery and mostly non-agricultural sites, are acceptable for any product even if the site is not listed on that product's label. Also, all sites are acceptable for any product that is a spreader/sticker, a fumigant, a rodenticide, or is exempt from mill assessments.

The product is considered to have a valid registration date at the time of application if the time of application is between the original registration date and the date it became inactive, or after the registration date if the inactive date is null, or before the inactive date if the registration date is null.

Grower_id and license_no. The grower_id is an identifier assigned to the operator or owner of the property where the application occurred. It is composed of: (reporting county_cd) + (year) + (permitting county_cd) + (permit number). The reporting county_cd is the code for the county where the pesticide application occurred; the application year is either the last two digits of the application year, or the expiration year of the pesticide permit, or the year the permit was issued; the permitting county_cd is the code for the county that issued the permit, and the permit number is an arbitrary number assigned to the permit (consisting of five characters). Most counties issue permits each year and in this case the year part of the grower_id should be the year of application. However, some counties issue multi-year permits which can be valid for up to three years. For multi-year permits, some counties set the year part of the grower_id to the year the permit was issued and some counties set it to the year the permit will expire. All counties except for Imperial issue permits in January; Imperial issues them in July. Because of these possibilities, the year part of the grower_id could be any number between the application – 3 to application year + 3.

The license_no is the license number of the person or business who applied the pesticide; the license_no is a new database field added to the PUR in 1999. For production agricultural reports the grower_id is required but the license_no is not required. For non-agricultural reports after 1999, applicators must provide either the grower_id or license_no. They can provide both but that is optional. In 1999, the license_no was considered optional for all reports.

Errors are checked for each of the different parts of the grower_id. If the number in the first two digits of the grower_id is not equal to the county_cd, it is changed by setting it equal to county_cd for the current application. This is not reported as an error, but is recorded in the **CHANGES** table. If the next two digits of the grower_id are not equal to the PUR year – 3 and PUR year + 3, it is also changed to the current year but reported as invalid and as an estimated value. If the value in 5th and 6th place is not a valid county_cd, then it is estimated by setting it equal to the current county_cd. The permitting county could obviously be another county, but in most cases it is the same county so this should be correct in most cases. In any case, this is reported as an error, so the counties have the opportunity to correct it. If the last 7 characters of the grower_id are spaces or all 0, then the grower_id is set to null and is treated as null for all error checking.

No error checking is done for license_no; any set of characters is allowed. However, sometimes an applicator will use the same number for both grower_id and license_no. In this case one of these is likely to be an error so the program assumes that if the number is the format grower_id, it will leave the value in the grower_id field and replace the license_no will null; if the value is not in the grower_id format the program will leave the value in license_no and replace grower_id with null. No error reports are generated, but the changes are recorded in **CHANGES**.

Inconsistent values for an agricultural field. An agricultural field should be identified by the combination of county_cd, grower_id, site_loc_id, and site_code. Since the grower_id contains the county_cd, it is not necessary to include county_cd in this list, but I include it to make the dependency explicit and to provide an important bit of information about the field. However, not all counties or all growers use the site_loc_id in a way to clearly identify different fields, so this can create problems with this interpretation.

If these values do identify an agricultural field, the acres planted and its geographic location should be the same for all reported applications to a given field. Ideally, the PUR database should identify each agricultural field with a unique identifier and contain a separate table with a list of all fields and their unique identifiers, sizes, geographic locations, and maybe other properties. However, the PUR table currently repeats the field properties for each application record. This can result in inconsistencies if different applications to a field are reported with different values for acres planted or geographic location.

At first, during the processing of each county PUR file, the loader would check for such inconsistencies in acres planted and location for each agricultural field. However, this procedure was very slow so the process was changed so that this procedure was done only after all files for one year had been first loaded. The procedure creates and populates a **FIELDS** table. In **FIELDS** the procedure assigns each agricultural field a unique identifier, which is stored in field_id, and stores a set of information about each field.

Location of agricultural fields is given using the Public Lands Survey System of geographical coordinates. These coordinates consist of values for base line meridian, township, township direction, range, range direction, and section. In the PUR each of these components is assigned to a different column. To make handling these values easier, the loader program defines a new user type called mtrs_type which is a composite value containing all these components. This value is referred to here as the "MTRS".

The error checking procedure first determines if the agricultural field reported in the **PUR** record being checked has been identified previously by looking for the field in the **FIELDS** table. If no record for this field appears in **FIELDS**, a new field_id is assigned to the field and a record is added to **FIELDS** containing values for field_id, county_cd, operator_id, site_loc_id, site_code, MTRS, acres_planted, and region. The value for acres planted is assigned only if the reported unit planted was 'A' (acres) or 'S' (square feet). If the units were square feet, the value is converted to acres, otherwise the value in acre_planted is used as is and recorded in **FIELDS**. Thus, acres_planted in **FIELDS** is always in units of acres so there is no need for a units field. **FIELDS** contains one other field called "inconsistent_field" which uses the value "X" as a flag to indicate if any inconsistencies were found for this agricultural field. Since we are discussing here the first record for an agricultural field there can be no inconsistencies and inconsistent_field is left null.

The operator_id is that part of the grower_id which uniquely identifies the operator or grower. The last 5 characters of the grower_id give the operator's permit number. Since different operators in different counties could be assigned the same permit number, to uniquely identify each operator in the state, we need to include the permitting county_cd which are in the 5th and 6th characters of the grower_id. Thus, the last 7 characters are used as the operator_id in **FIELDS**. However, Alameda County has an additional procedure. They will sometimes give one operator two different grower_ids, one that ends in 'R' and another that ends in 'U'. For Alameda County, the operator_id is set equal to the last 7 characters of grower_id, but if there is an 'R' or 'U' on the end, that character is removed.

If the agricultural field for the record being checked had previously reported applications and is, therefore, recorded in **FIELDS**, the procedures checks to see if this field had previously found inconsistencies. If there were no inconsistencies before (that is, the field inconsistent_field is blank) and the acres planted and MTRS for the current record is the same as the previous reported values for this field, nothing else is done. If the current values for acres planted or MTRS are different from the previously consistent values for this field, an 'X' is recorded in inconsistent_field of **FIELDS** and two new rows are inserted into the table **FIELD_MTRS_ACRES**. This table records all the inconsistent values of acres and location for an agricultural field and the number of records in the PUR for each combination of acres and MTRS. We are now discussing the situation of finding the first inconsistent record for an agricultural field, so one of the new rows in **FIELD_MTRS_ACRES** will give the field_id, MTRS, and acres_planted for the current record, and the value 1 for num_recs. The other row will be the same field_id, and the MTRS and acres_planted previously recorded for this field and the number of records in the **PUR** for this agricultural field. Finally, for this situation, an error record will be created in **ERRORSI** for the current record as well as for all the other records in the **PUR** for this agricultural field. These are all flagged as possible errors because we do not know which value is correct. However, the program tries to determine which of the different acres planted or MTRS values are most likely to be the correct values. It skips any missing acres planted values or any MTRS values with missing components and chooses the value that has the largest number of records.

If there were previously found inconsistencies for this field and if the current record has values for acres planted and MTRS that are in the list of previously found values, the record count is increased for this combination of acres planted and MTRS in **FIELD_MTRS_ACRES**. If this combination was not found previously, a new row is added to **FIELD_MTRS_ACRES**. Since the record count has now changed, the program reevaluates the most likely values for acres planted and MTRS as before by choosing the value with the largest number of records and making any changes in **FIELDS**. Of course, a record is added to **ERRORSI** to record the new possible error.

High rates of use. Some PUR records may have extremely large rates of use (pounds of pesticide per area or volume treated). Sometimes unusually large rates are actually applied even when they are larger than the legal maximum rate, but often these values are errors.

How can we determine which values are probably errors? It would help if our database had the maximum label rates, but it doesn't. Even if it did, this would not tell us how large a value must be before it is flagged as a possible error. Several different criteria have been developed to help identify possible errors in rate of use (described in the report "A Computer Program to Identify Outliers in the Pesticide Use Report Database" by Larry Wilhoit, PM 98-01). Three of these criteria were used in the loader program. These criteria are used to flag a record as a possible or probable error if the rate is larger than:

1. 200 pounds of AI per acre treated,
2. the median pounds of product per acre for all uses of that product on a site or commodity, and
3. a value determined by a neural network.

The neural network is the most unusual technique. Basically, it is a computer programming technique to recognize patterns in a set of complex and noisy data. In this case, the distribution of rates have such unusual distributions, standard statistical techniques are of limited use.

The outlier program is run for an entire year of PUR data and creates a table of the outlier rate limits from the different criteria for each combination of product, site or commodity, unit treated, and record type used in the PUR. The loader program uses the previous year's outlier table to decide if each rate value in the file to be loaded is unusually high.

The program treats outliers found by the first two criteria differently from the third (neural network) criterion. If a rate is larger than either limit from the first two criteria it is recorded as a probable error in **ERRORS** and replaced by the median rate of all uses of the product on that site and unit treated and a record of the change is made in **CHANGES**. If it is higher than the third criterion it is left unchanged and recorded as a possible error in **ERRORS**.

Replacing a rate value needs further explanation since there is not a field for rate in the PUR database. A high rate can result from errors in any of five different database fields: unit treated, area treated, pounds of product, amount of product, or the amount's unit of measure. The program goes through the following algorithm for all records with rates of use higher than one of the criteria to determine which value is likely incorrect and changes the value so that it equals or is close to the median rate.

The program first checks the unit_treated. If the unit_treated is not acres, the record is for a production agricultural application, and the site is not commodity or other fumigation, mushrooms, or any of the sites with site_codes between 40010 and 66000 (these include sites such as animals, structures, and aquatic areas), then it recalculates the rate assuming that the units are acres. If the recalculated rate is less than all outlier limits, the loader program changes the units to acres. In the PUR for years after 1994, 97% of all applications to the sites not excluded used acres for the unit treated, so it is reasonable to assume that in most of these situations the units should be in acres.

For other sites, or where units are already in acres, or where changing to acres still does not make the rate less than the outlier limits, the program then checks the value for acres treated. If this value is less than 1 for any site other than greenhouses, nurseries, any flavoring and spice crop, cilantro, mint, or mustard and if lbs_prd_used/(median rate) is less than acre_planted, then the acre_treated is set equal to lbs_prd_used/(median rate). The median rate is the median rate of use of this product on this site for all applications during the previous year. Making this change to acre_treated will make the rate of use equal to the median rate of use. The above list of sites was excluded because queries revealed that applications are often reported with acre_treated < 1.

For all other situations or where these changes still do not make the rate of use less than the outlier limits, the lbs_prd_used is changed to equal median rate times the acres treated and amt_prd_used is changed proportionally the same. Note that the applicators report only amount of product and unit of measure in their use reports. The loader program calculates the pounds of product from these values.

Duplicate records. Another kind of error that is difficult to handle is an erroneous duplication of an entire record. This could happen, for example, if two different people (say a grower and a PCO) both sent in reports for the same application, if the data for an application was entered twice into the county database, or if the county accidentally sent the same data file twice to DPR. However, some apparent duplicates are correct, for example, if a grower applied 2 or more spot treatments of the same AI to the same field at the same time and reported these as different applications.

Records are considered duplicate if 2 or more records have the same values for county_cd, grower_id, site_loc_id, site_code, qualify_cd, prodno, amt_prd_used, unit_of_meas, acre_treated, unit_treated, acre_planted, unit_planted, applic_dt. The column, applic_time was added in 1999 to help separate error from valid duplicates. However, this data field is not used consistently by all counties so currently is not used in the checks for duplicate records.

The column record_id is not included in the set because the same application could have been reported by a grower (who uses a report with record_id equal to 4 or B) and a PCO (who uses a report with record_id equal to 1 or A). Also, we don't include the MTRS (location) because site_loc_id is supposed to uniquely identify each field for each grower. Errors from using the same site_loc_id for two different fields in different sections were identified as inconsistent field errors (described above). This procedure, of course, will still not find situations where two different fields in the same section had the same site_loc_id.

In order to decide which record duplications are probably errors, the program adds all the acres treated for a set of duplicate records and if this sum is greater than the acres planted for the field, all these records are marked as error duplicates. Not all of these records are inserted into the **PUR** table. Records from a duplicate set are inserted until the sum of acres treated equals the acres planted. Because this procedure depends on values for acres planted, it can only be used for production agricultural records. Duplicates for non-agricultural records (record_ids of 2 and C) are not flagged as errors.

This procedure, however, will not always correctly identify error duplicates. For example, apparent error duplications would appear if a grower made 2 spot treatments to a field, reported these as two different treatments, but reported the acres treated each time as the entire field. Another example would be two treatments to two different fields that had the same site identification (grower_id and site_loc_id).

We also have to decide what to do if any of these fields are NULL. Since all these fields are required to have a value, the presence of a NULL indicates an error. Most likely, the original erroneous values were the same for each record, so NULLs are considered the same value in each record.

Error duplicate records have error code 80 in **ERRORS**, but to see which records are duplicates of one another, each set of duplicates are given a duplicate set number which is recorded in the column duplicate_set in **ERRORS**. Records that are not inserted into **PUR** are recorded in **CHANGES** with a replace_type value of "delete" to indicate that the entire record was not inserted into the PUR.

Dealing with values that are optional or have unknown requirements. Some field values do not need to be reported but if a value is given it is accepted (such as qualify_cd and planting_seq); these fields are marked as optional. If optional values are given (other than '0' or '.') they are error checked as if they were required; if no values are given (or '0' or '.' which are treated the same as nulls) no error record is created.

Some fields should not have a value (such as site_loc_id for non-agricultural records); these fields are marked as not allowed. If a value is given for fields not allowed, then the value is set to the null character and is not recorded as an error.

In some rare cases, value requirements are unknown. This would happen if the value used to determine requirements (such as the record_id) were unknown. In these cases, the loader decides what to do based on the following algorithm, assuming that the record_id is missing. If a value is optional or not allowed for some record_id values and if a value is required for other record_ids, then if the value is null or 0, record it as a possible error, or if the value is not null but an error, record it as an error. Again, if a value is optional or not allowed for some record_id values, but if no value is required for other record_ids, then if the value is null, do not record it as an error, or if the value is not null but an error, record it as an error.

Calculating Pounds of Product

The loader program during error checking will make some corrections and conversions as described above. However, the most important conversion is calculating pounds of product used (lbs_prd_used) from the amount of product used and unit of measure as reported by the applicator (Table 4A). In most cases the calculation is simply a matter of converting from one set of units to another. For liquid formulations (reported in volume units such as gallons, pints, liters) you also need the product specific gravity, which the loader gets from the label database. The only difficulty is that the unit of measure of ounces ('oz') has two different meanings. It can refer to either a volume measure or a weight measure. The loader determines which meaning to give ounces by using Table 4B to determine whether a product should be considered liquid or dry based on its formulation, again which comes from the label database. A further difficulty is that pressurized gas formulation for most purposes is considered a liquid (and this is what it is called in the label database), but pesticide applicators typically report amount of pressured gas in measures of weight. Thus the loader assumes ounces of pressured gas measures a weight.

Table 4A. Formulas for calculating the pounds of pesticide product from the reported amount used, for each unit of measure. The specific gravity (density of product/density of water) is needed to convert from volume units to weight units. The density of water is 8.33 lb/gal. 'OZ' can refer to either a liquid or dry measure and this will determine which formula to use. We can determine whether a product is liquid or dry by the pesticide formulation using the relationship in Table 4B.

Unit Meas.	Dry/Wet Meas.	Calculation of lbs_prd_used
LB	D, W	amt_prd_used
OZ	D	amt_prd_used / (16 oz/lb)
GR	D	amt_prd_used / (453.59 g/lb)
KG	D	amt_prd_used * 2.2046 lb/kg
OZ	W	amt_prd_used * specific_gravity * 8.33 lb/gal / (128 oz/gal)
GA	W	amt_prd_used * specific_gravity * 8.33 lb/gal
QT	W	amt_prd_used * specific_gravity * 8.33 lb/gal / (4 qt/gal)
PT	W	amt_prd_used * specific_gravity * 8.33 lb/gal / (8 pt/gal)
LI	W	amt_prd_used * specific_gravity * 8.33 lb/gal / (3.785 li/gal)
ML	W	amt_prd_used * specific_gravity * 8.33 lb/gal / (3785. ml/gal)

Table 4B. Formulation table

Dry/Wet	Formula-tion code	Formulation description
D	A0	DUST/POWDER
W	B0	EMULSIFIABLE CONCENTRATE
W	C0	FLOWABLE CONCENTRATE
W	D0	GEL, PASTE, CREAM
D	E0	GRANULAR/FLAKE
D	F0	IMPREGNATED MATERIAL
W	G0	MICROENCAPSULATED
W	H0	OIL
W	I0	PAINT/COATINGS
D	J0	PELLET/TABLET/CAKE/BRIQUET
D	K0	PRESSURIZED DUST
D	L0	PRESSURIZED GAS
W	M0	PRESSURIZED LIQUID/SPRAYS/FOGGERS
D	N0	SOLUBLE POWDER
W	O0	SOLUTION/LIQUID (READY-TO-USE)
D	P0	WETTABLE POWDER
W	Q0	SUSPENSION
D	R0	DRY FLOWABLE
W	S0	LIQUID CONCENTRATE
W	T0	OTHER (LIQUID)
D	U0	OTHER (DRY)

Computer Program Description

Location of the Loader Program and Output Files

The scripts to load and error check the data are on Environmental Monitoring Branch's UNIX server *pelia*; the script files are all owned by the UNIX user purload and all loader program and output files are in *pelia:/home/purload*. Nearly all directories and files mentioned will be located here. The exception is the directory *scalos:/ora05/data* where DPR staff place the county PUR data files.

Each year's program is in a separate directory in *sql* (such as *sqlpur2000*, *sql/pur2001*, etc.). The program consists of many files, but the main loader program is a perl program, *loader.pl*. This program manages the overall file processing, Oracle and operating system error-handling, and runs several SQL and PL/SQL scripts (with filenames that end in ".sql") and Oracle loader scripts (filenames end in ".ctl"). Most of the other files in this directory are the scripts called by *loader.pl*. The remaining files serve various purposes: *loader.vpj* and *loader.vtg* are created by Visual SlickEdit, which is the programmer editor I used to create the program files; *afiedt.buf* is a temporary file created by SQL*Plus when editing scripts in the buffer; files ending in ".t" are temporary files created from each county file during loading; and ".out" files contain results or error messages from the cron process that runs the loader.

Version 1.0 of the loader program, which was used to load the 1997, 1998, and the preliminary 1999 PUR data are in the directories *sql/pur97*, *sql/pur98*, and *sql/pur99_vers_1*. Version 2.0 of the loader, used for the final 1999 PUR, is in the directory *sql/pur99*. This directory contains two configurations of the program—batch processing and individual county file processing. The batch configuration was used to create the final release of the 1999 PUR and these files are in the directory *sql/pur99/batch*. The county file processing scripts are in the directory *sql/pur99/test*. The loader program will normally be used with the individual county processing and this is the version that is described in this document. The batch configuration was only used after DPR had received most or all of the county files, so these were all loaded at once, rather than file by file. The files in *sql/pur99/test* were used to test the new version, and will be implemented starting in 2000.

All county files and files produced by the loader program are in the directories *pur1997*, *pur1998*, *pur1999*, *pur2000*, etc.. The directory *pur2000* and for later years contains several subdirectories. Directory *pur2000/data* contains copies of the county files; *pur2000/load_logs* contains log files of the loading process; *pur2000/loaded* contains the county files that were successfully loaded; *pur2000/loading* contains county files that are in the process of being loaded; *pur2000/not_loaded* contains files that could not be loaded because of some kind of errors; and *pur2000/reports* contains the county error reports that are sent to the counties. Directories for the other years contain similar kinds of subdirectories.

This document and other documentation files are in *docs/loader*.

Monitoring the Loader Program
The main program, loader.pl, is started automatically by a cron process each day at a certain time. You can see the time it will start by logging onto UNIX as user purload and typing: “crontab –l”. The first two numbers of the output gives the min and hour of the day that *loader.pl* will start. You can change this by editing the cron file using “crontab –e”.

After each day’s run, the loader will email one or two log files to the list of loader administrators. One log file gives a more detailed description, results, and error messages, if any, from each major step in the loader process. The other log file simply gives a list of county files found, a list of the files that were successfully loaded, and a list of files that were not loaded because of errors.

If no problems appeared, then the loader process needs no other attention. If there were problems, the detailed log file will explain the problem and its location (for example, which script and where in the script). This may involve checking either the county file for problems or the loader script.

To set up a new year’s loading process, you need to create the necessary directories and copy the code to the new directories. To help in that process, you can use the perl script *setup.pl*

What the Loader Program Does
A complete explanation of what the program does is found in the source code. These files are extensively commented to help a programmer understand the code. All source code is printed in Appendix 2.

Before the loading process starts, the loader checks several things. First it checks that the program is not still running from a previous time; if it is still running it will not start another process. If there are large PUR county files or many files to process, the program may take longer than one day to run. If it is not already running, it will open a log file to record the program’s progress and any errors than may occur. This file is saved into the directory *pur2000/load_logs* and will have the current date and time appended to the filename. The loader then checks that all needed files are present and will quit if any are missing. The log file will contain a list of the missing files. It will then check that the compiled procedures are up to date; if not, it will recompile them from the necessary files. Finally, it will check if any new PUR files have been copied to the designated directory. If so, it will first copy the files (for backup) to *pur2000/data* and then move the files to *pur2000/loading*. At this point, the normal PUR file processing can begin.

The loader program will first get the name of each PUR county file in *pur2000/loading* and examine the county file’s basic structure. First, it creates a temporary copy of the file, and in that copy it replaces a string of one or more control characters with the UNIX end of line character. It also finds the number of characters in each line of the file and the number of lines in the file. If the line lengths are different it will print error messages giving the line lengths for the different line numbers and will not load that file. Finally, it adds the name of the file and last modification date of the original PUR file to the end of each line in the temporary copy.

Next, if the file contains more than 15,000 lines, it will split the file into nearly equal sized smaller files and name each file with the name of the original file with "_1", "_2", etc. added to the end of name. The reason for splitting large files is that the program runs inefficiently for large files. It usually takes more than one day to process files with more than 15,000 lines; for files much bigger, it could take the program weeks to finish. However, the same data distributed in several smaller files would be loaded much sooner.

It next checks if data from a file with the same name was loaded previously. If a file with the same name was previously loaded, the program will not load the data from the new file. Instead it will move the file to *pur2000/not_loaded*, generate an error message, and then go to the next file in *pur2000/loading*. If the file was not previously loaded, the loader will then check if the intermediate tables (**RAWI**, **ERRORSI**, and **CHANGESI**) are ready to process another file; unless something went seriously wrong with the loader program these tables should have no data because they only store data temporarily while processing each file.

Next, the loader calls SQL*Loader to load all the data in the file into the Oracle table **RAWI**. This table consists of character fields, so the data gets loaded exactly as it is in the file. There are four common types of files: one contains production agricultural records, another non-agricultural records, and, for each record type, there are the old file formats (before 1999 with no fields for license_no and applic_time) and the new file structures (starting in 1999 with the two new fields). The loader still needs to check for old file formats because not all counties have switched to the new format. There is, also, a special format for files from CalTrans, which have their own unique structure. The loader determines which file format to use based on the number of characters in each line. After SQL*Loader runs, it creates a log file with information and possible errors messages. These files are saved in the directory *pur2000/load_logs/oracle*. Also, if the data were successfully loaded, a record is inserted in **LOG** with the name of the file, number of lines loaded, and the current date. The loader checks that the number of new rows in **RAWI** equals the number of lines in the PUR file found previously when the loader checked the file structure.

The next check is the error checking of the PUR data, now loaded into **RAWI**. The code for this is in two compiled PL/SQL packages. One package, Co_error, contains one primary procedure that calls separate procedures from another package, Check_value, that checks for errors in each field of the PUR. The Check_value procedures take as input parameters the use_no or the current record and the value of the current field being checked. Some checks require other information such as the record_id or county_cd. Each procedure returns values for an error_code, error_type, replace_type, and require_type. The error_code will return a number corresponding to the error found, or, if no errors are found, it will return a null. The error_type can be 'invalid', 'probable', 'possible', 'duplicate', or 'N' (for no errors). The replace_type can be 'null', 'estimate', 'same', or 'delete'. The require_type can be 'required', 'optional', 'not_allowed', or 'unknown'.

The purpose and meaning of each of these terms is explained above in the section "Types of Errors and How They Are Handled". The procedures in Check_value only checks for errors.

The procedures in Co_error use the Check_value procedures to decide how the data will be handled. Values with no errors will be recorded into the table **PUR**. If an error is found, a null character, an estimated value, a corrected value, or the original value will be recorded into **PUR** depending on the parameters returned about from the Check_value procedures. Also, if errors were found and not corrected a record will be inserted in **ERRORSI** and if the value entered in **PUR** was in anyway different from the original value in the PUR file, a record will be inserted into **CHANGESI**.

Because each field has its own separate procedure, these procedures can also be used in the later stages of the PUR system when corrections need to be made. Staff use another program to make corrections to the PUR database. This error correction program calls the Check_value procedures to determine if new values entered by staff contain errors.

After each row from the PUR file is checked for errors, the program assigns the next sequential number to the use_no value, which is then stored in **RAWI** and **PUR**. After all data are error checked, the starting and ending use_no values for this file are then recorded in **LOG**.

An error report is then created from the data in **RAWI** and **ERRORSI** and saved in the directory *pur2000/reports*. The loader program prints the error report and staff mail the report to the county that produced the original data file. This report contains a list of each line in **RAWI** (which is the same as a line in the PUR file) that contains at least one error and, for each line, a list of all the errors found in that line. It also includes a table listing all product registration numbers and site_codes where the site_code was not found on the list of commodities for that product; a table listing each error found with the number of records that had that error; the number of invalid and valid records and the total number of errors for each record_id; and the total number of records for each record_id, application month, and batch_no. Since the loader program runs every day and prints an error report for each county file, the error reports normally are sent back to the counties after a few days from the time the county files were received at DPR.

The loader then moves all records from the intermediate tables to the permanent tables, moves the PUR file in *pur2000/loading* to the directory *pur2000/loaded*; and deletes the temporary copy of the county PUR file.

Finally after all the PUR files have been processed, one of two log files are emailed to a list of people interested in monitoring the loading process. One log file contains a list of all PUR county files found, files that were successfully loaded, and files were not loaded because of errors. The other log file contains the same information but in addition it contains a detailed record of the loader progress during each major step of the process and all error messages, if any, that were generated during the process (such as out of memory errors, missing code scripts, or problems in the structure of any data files).

Rationale for the Program Structure
The processes described here may seem excessively complex, but there were clear and conscious reasons why it was designed this way. The database and error tracking tables need to be flexible and complete enough for several different purposes. Relating the history of the loader program may help in understanding why the program was created in the way it was.

The loader program was originally written by the Information Systems Branch using the database program FOCUS. In 1999, the responsibility for the PUR was divided between the Enforcement Branch and the Environmental Monitoring and Pest Management Branch (EMPM), with Enforcement taking responsibility for the interaction with the counties and EMPM processing and storing the data. Also, the database system was changed from Focus to Oracle. In the process of rewriting the loading and error-checking procedures for Oracle, we had the opportunity to improve the system and add other error-checking procedures.

Based on my experience with the PUR data I suggested a few changes and additions to the error checks that were previously done on the data. These suggestions were sent to several DPR staff and were discussed at a meeting of these staff on March 3, 1999. A description of the agreements we made on the error checks are given in the file *doc/new_validations.doc*. Appendix 1 describes the major differences between the error checks done previously with the checks described in this documentation.

Responsibility for rewriting the loader program was originally given to Yihua Lin. When she left the department, the responsibility was moved to Steve Kishaba. However, Steve was later moved off that project, which was then handed over to Larry Wilhoit. The code Yihua and Steve developed is in the directory *sql/yihua_steve_khoas_files*. They had made a good start, but did not finish it.

Yihua's approach to error checking was to write separate SQL scripts for each field. These scripts would scan through the entire PUR table for each field and record any errors found in another table. For simple error checking this worked fairly well, but more complicated error checking required the use of a procedural language. So I rewrote the routines in PL/SQL. This also vastly improved the efficiency because rather than scanning the entire PUR table every time for each column, the PL/SQL code read each row or record into memory and checked all values in that row for errors. Thus, the PUR table was scanned through just once. Also, many of the fields depend on or are affected by values in other fields. In the SQL scripts these other fields would have to be read in separately again (another costly operation) while the PL/SQL code had all values for a row already in memory. Another difficulty with the SQL scripts was that if any unexpected errors occurred the entire script was stopped. The PL/SQL code can handle unexpected errors in appropriate ways. Finally, the PL/SQL structure made the code much easier to understand.

However, my first attempt at creating this code (which are the ones I have used for loading the 1998 and 1999 data and are in the directories *sql/pur98* and *sql/pur99_vers_1*) had a number of problems. First, it was written as one function and as it got longer with additional and more complex error checks, it became difficult to follow especially when conditional statements spanned several pages. It was difficult to know where each statement was in relation to the various conditions and difficult to follow the logic.

Also, this function was written to both load the data into Oracle tables and check for errors. However, error checking should also be done when any changes are made to the PUR. The error checking routines in the loading code could not be used as they were since they were embedded in code for another function. So, to make the code easier to follow and to separate the error checking function from the loading function, yet another rewriting of code was started and this became version 2.

This latest version creates separate PL/SQL functions for each value and each function only checks for errors. These functions require several input values and several output values that can be used for different functions. The input values include the value from the county file as a character and other values from the current record as necessary, such as record_id, to determine what error checks were appropriate. The output values included the correct value in the correct type, or null value if it was invalid, or the correct or estimated value, and the error code and type of error if there was an error, whether the value was corrected or estimated, and whether the value was required.

In addition, version 2 improved on many of the error procedures previously written and added some additional error checks that I never had time to do in the first version, the most important ones being duplicate records and inconsistent agricultural fields. Version 2 was also extensively tested for programming bugs. Finally, the Oracle table structure was improved, including more information on errors found, and made it easier to use and understand. For example, in version 1 both errors found and changes made were recorded in one Oracle table. These two very different kinds of records are now kept in different tables.

The **CHANGES** table is important to track the history of changes to the PUR which occur through the year and even after the "final" PUR is released. Thus, queries run at different times may use slightly different versions of the PUR. It may be important to know which version of the PUR was used, which can be done by reporting the last PUR update along with the data from the query. If for some reason a query from an earlier PUR version needs to be replicated, a person can use the **CHANGES** table to recreate the data from a previous version of the PUR.

Conclusion

All PUR data received from all counties in California are now processed by the loader program described in this document. The loader program runs every day and processes all the files received from the counties, checks the data for errors, loads valid data into the PUR table, flags any errors it finds, and generates an error report that is sent back to the appropriate county for correction. The original data received from the counties is kept in a separate database table. The loader program also maintains another table that lists all the errors found and another table that keeps a record of all changes that have been made to the PUR. All these tables can be viewed or queried by DPR staff.

This program adds several improvements over the previous loader program:

1. it adds several additional error checks thus improving the accuracy of the data;
2. it leaves out of the database tables only invalid values (or replaces them with estimates) not entire records that contain an error in any of their fields;
3. it includes records for illegal applications;
4. it is continually being updated and corrected so that queries will always have the most accurate data available;
5. it contains a table listing all current errors or possible errors so that staff can see what problems may exist in the data;
6. it contains a table giving the history all changes made to the database;
7. it contains a table listing all the county files received and the number of lines from the file that were successfully loaded;
8. it automates many of the loader tasks, possessing data and emailing the results of the process every day without the need for staff intervention;
9. the program and how to manage the process is fully documented;

Appendix 1: Differences Between the New and Previous Error Procedures

This new error checking procedure has several significant differences from the procedure used in previous years. One difference is that all errors found and all changes made are recorded and tracked in database tables. This allows people to see which values have errors and the entire history of which values have been corrected or changed.

In contrast to the previous procedure where, if a value was found to be an error, the entire record for that application was not entered into the PUR, the new procedure enters the record but changes (usually by making it null) only the fields that contain errors. Thus the data will be more complete and accurate. Another significant difference is that previously any error found that was a legal violation (such as applying a pesticide to a crop that was not on the label) the record of that application was not entered. With the new procedure these records are entered into the PUR database, again making the data more complete.

Finally, several new error procedures were added and previous ones improved. The primary additions are the checks for extremely high rates of use, agricultural field consistencies, duplicate records, but there are several others. Further details of the differences can be seen by comparing the Appendix Table, which lists the error procedures carried out in 1997, and Table 2, which lists the new error procedures. The Appendix Table includes a column that flags error procedures that differ significantly from the new procedures. These are the only descriptions or explanations DPR supplied for the error checks and it is not always clear what some of these descriptions mean or to what kinds of records they applied to. In some cases they appear to differ from what was in the actual code.

Appendix 1 Table. Error codes and descriptions used in the loader program in 1997. The column "Differ" contains an "X" if the new error checks differ from these. The new error checks also include several additional error checks.

Error Code	Differ	Error Description
1		record id not numeric or equal 1, 2, 4, 9, A, B, C, D
2		batch no. not numeric or less than 0001
3		process date (month) not numeric or not between 1 and 12
4	X	process date (year) not numeric or not between 90 and 99
5		county no. not numeric or not = 01 thru 58
6		section no. not numeric or not = 01 thru 36
7		township no. not numeric or not = 01 thru 48
8		township direction not equal N, S, or space
9		range not numeric or not = 01 thru 47
10		range direction not equal E, W, or space
11		base meridian not equal H, S, M, or space
12		application method not equal A, G, or O
13		commodity code not numeric
14		commodity code not on commodity table
15		date applied (month) not numeric or not = 01 thru 12
16		date applied (day) not numeric or not = 01 thru 31
17		date applied (year) not equal 97
18	X	acres treated not numeric
19	X	units not equal A, C, K, P, S, T, U or space
21	X	units = A, C, K, P S, T, or U and units treated equal zero
22	X	units = space, and units treated not equal zero
23	X	units = A, acres treated greater than 700.00
24		reg. no. (firm) not numeric, or equal zeros
25		reg. no. (label) not numeric, or equal zeros
26	X	reg. no. (rev code) not numeric, or equal spaces
27		reg. no. (sub reg) not numeric
30	X	number of applications not numeric or equal zeros
31		amount used not numeric, equal zero
32		unit of meas. not = lb, oz, ga, qt, pt, gr, kg, li, ml)
34		document no. not numeric or equal spaces
35		line item not numeric
37		product not on master label file
38		error in lbs conversion (check formulation)
39		commodity not on master label file
40	X	units not A, C, K, P, S, T, U or units treated = zero
41	X	units not space or units treated not zero
43	X	invalid grower id
44		acres planted must be numeric
45	X	invalid unit planted, see 19
47	X	treated units greater than planted units
48		invalid combination of county + township + range + section
50	X	amount product used > 3000 and unit of meas is GA
51		invalid application date (i.e. 04/31/97 or > current date)
52		check spec. gravity & formulation (internal used only)

Appendix 2: Loader Program Source Code

The source code for the loader program exists in several files, all located on the UNIX computer *pelia* in the directory */home/purload/sql/pur2001*.

The main program managing the operation of the loader program is a Perl program, name *loader.pl*. This calls several SQL and PL/SQL scripts to carry out the various database functions.

Loader.pl

```
#!/usr/local/bin/perl

# This file is the main program for loading PUR county data into
# the Oracle database. It processses the county files and
# calls several SQL scripts to load the data into Oracle tables
# and checks the data for errors.

# This program is started ever night at 10 PM by a cron process.
# Originally:
# (00 20 * * 1-5 . o8iprofile_load; cd sql/loader;
#  /usr/local/bin/perl loader.pl > cron.out 2> cronerr.out)
# Now:
# (00 20 * * 1-5 . /home/oracle/.profile; $HOME/sql/loader/loader.pl >
#  $HOME/sql/loader/cron.out 2> $HOME/sql/loader/cron.err)
# If any changes are made to check_value2001.sql or co_error2001.sql,
# these files must be run to compile their procedures.

# As this program runs it records success and failures at each step,
# saves this information in a file named "load_log_<date>.txt".
# The expression <date> will contain the date and time that the
# program ran.  This log file will be emailed to the PUR administrators.

# More specifically, the program:
# 1. checks that the program is not still running from a previous time;
# 2. checks that all needed SQL files are present;
# 3. checks for PUR county files in a certain directory;
# 4. makes a backup copy of the files in the "data" directory;
# 5. moves all the files to the "loading" directory;
# 6. checks that each file has not been previously loaded into the database;
# 7. checks that the Oracle tables are ready for loading;
# 8. checks that each file has the correct format and fixes certain
#    kinds of errors;
# 9. loads the data into the RAW table and records the number of rows
#    loaded in the LOG table;
# 10. checks the data in RAW for errors;
# 11. adds the valid data to the PUR table;
# 12. updates the LOG table;
# 13. creates the county error reports and prints them;
# 14. cleans up the intermediate tables;
# 15. moves the county files to the LOADED directory;
# 16. emails the log files to administrators.

# This code is normally used to process each individual county PUR file
# by reading the data into intermediate tables which then get inserted
# into the RAW table.
# However, sometimes we need to recreate the PUR tables from data already
# loaded into the RAW table (for example, if we change some of the error
# checks after the data was loaded).  I call this a batch run.
# To do a batch run, first make the necessary changes to co_error2001.sql
# (see the file for what changes to make).  Then, to run the program.
# use batch_run.sql rather than this perl script.

# Also, sometimes you might not want to produce an error report for each
# county PUR file.  Instead, you might want to load a bunch of files
# and produce an error report later.  In this case, set
# $create_error_report = 0, and then run "run_err_by_county.pl" which
# calls create_use_error_report.sql, error_report_by_county.sql, and
# error_report_caltrans.sql.

# If you want the error reports printed to Enforcement's printer,
# set $print_error_report = 1,
# otherwise set $print_error_report = 0.
# To print files from the UNIX prompt type:  lp -c -d hp8k_vili *

# If the loader failed to load some county files because of some kind of error,
# then after fixing the problem you need to load these files,
# move the files to directory /home/purload/pur2001/2001data
# and set variable $reload = 1.

# I have sometimes found C files which had values for license_no
# (which is the last field in the county file) except for a few records
# which had only spaces.  The code in Fix_file() function strips all trailing
# spaces, which made that line appear the wrong length.  Since the
# files are ok, in order to load that file you need to stop stripping
# of the trailing spaces to load those files.
# You can do that by setting variable $strip_trailing_spaces = 0;

# Another situation found occasionally for C files where values for
# license_no were not space filled.  SO if license_no contained
# no value or short values that were left justified, the line would
# be too short.  In these cases, you need to add spaces to the end
# of short lines to make them all the correct lenght.
# The perl program make_line.pl will do this for you.
# At some point, I may add that program to loader, but currently
# this has to run separately for each file.

# If you do not see any files in /Scalos_ora05/data from pelia when you know
# they are there, you probably need to mount that directory
# To mount a directory, you need to you full path name, and type the following
# as system administrator:
#      mount /Scalos_ora05/data


##############################################################################
###################     Main processing procedure     #######################

# List of email names for people who will receive email of the log files.
# $administrators receive the detailed log file;
# $mail_short_log receive the short log file.
$administrators = "wilhoit";
#$mail_short_log = "lwilhoit\@cdpr.ca.gov";
#$mail_short_log = "purdata\@cdpr.ca.gov lwilhoit\@cdpr.ca.gov";
$mail_short_log = "purdata\@cdpr.ca.gov";

# If you want an email acknowledgment of each file whose error report
# was printed, set $email_file_print = 1.
# If you don't want email for each file, set $email_file_print = 0.
$email_file_print = 1;

# If you want to create an error report file for each county file loaded,
# set $create_error_report = 1,
# otherwise set $create_error_report = 0.
# If you want to generate error report for all files by county,
# use run_err_by_county.pl.
$create_error_report = 1;

# If you want to print the error report,
# set $print_error_report = 1;
# otherwise, set $print_error_report = 0.
$print_error_report = 1;

# If you want to run loader for files that are put into directory
# /pur2001/2001data, set $reload = 1.
# For normal processing of data stored in /Scalos_ora05/data/2001data
# set $reload = 0.
$reload = 0;

# Normally, you want to strip trailing spaces in the county file.
# However, if there are any rows in a county file
# which have all spaces in the last field and if
# no lines have extra trailing spaces you should not
# strip spaces.  To strip spaces,
# set $strip_trailing_spaces = 1,
# otherwise set $strip_trailing_spaces = 0.
$strip_trailing_spaces = 1;

# Maximum acceptable number of lines for PUR county data file.
# If a county data file has more lines than this,
# the program will split the file into multiple files,
# each new file named as the original file with extenstion
# "_1", "_2", etc.
$max_num_lines = 15000;

# Perform various preliminary procedures and
# return the number PUR county files found by the program.
$number_of_files = &Setup;

print LOGFILE "\nProcess each PUR county file...\n";
while($nextfile = </home/purload/pur2001/loading/*>) {
   $file_name = $nextfile;
   $file_name =~ s/.*\///; # remove part before last slash.

   # If file is a plain file, process it
   if(-f $nextfile) {
      print LOGFILE "\n".("_" x 25)." $file_name ".("_" x 25)."\n";

      ($num_lines, $line_size, $error) = &Fix_file($nextfile, $file_name);
      &Handle_error($error, $nextfile, $file_name);

      %split_files = &Split_file($file_name, $num_lines, $max_num_lines);
      $g_split_file = %split_files - 1;
      $number_of_files = $number_of_files + $g_split_file;

      $any_error = 0;
      while ( ($split_file_name, $num_lines) = each (%split_files) ) {
         if ($g_split_file) {
            print LOGFILE "\n".("_" x 25)." $split_file_name ".("_" x 25)."\n";
         }
         &Intermediate_check;
         $error = &File_check($split_file_name);
         $any_error = $any_error + $error;
         &Handle_error($error, "/home/purload/sql/pur2001/$split_file_name.t",
                       $split_file_name);

         $error = &Load_raw($split_file_name, $num_lines, $line_size);
         $any_error = $any_error + $error;
         &Handle_error($error, "/home/purload/sql/pur2001/$split_file_name.t",
                       $split_file_name);

         &Check_errors;
         &Update_log($split_file_name);
         if ($create_error_report) {
            &Error_report($split_file_name);
         }

         &Move_intermediates;

         # If got here, then file loaded successfully.
         push @loaded_files, $split_file_name;
         &Remove_temp("$split_file_name.t");
      }
      if ($any_error) {
         &Move_file($nextfile, "/home/purload/pur2001/not_loaded", "print2");
      } else {
         &Move_file($nextfile, "/home/purload/pur2001/loaded", "print2");
      }
   }
}
```

```
&Print_results($number_of_files);
&Loaded_files;

close(LOGFILE);
close(SHORTLOG);

&Email_logs;



##############################################################################
######################      Function definitions        ######################


##############################################################################
# File_check determines whether the current county file has already been loaded
# into the RAW table.  If so, the file is moved to the error_file directory
# and the program continues to process the next county file.
sub File_check {
   my $p_file_name = @_[0];
   my $v_error;

   print LOGFILE "Checking if $p_file_name has already been loaded...\n";

   # The SQL script file_check finds the number of records in the RAW table
   # which were loaded from a file with the same names as the current file.
   # This number then appears in the variable $? (multiplied by 256).
   # open(SQL_OUTPUT, "sqlplus -S wilhoit/ \@file_check $p_file_name |") or

   unless (open(SQL_OUTPUT, "sqlplus -S / \@file_check $p_file_name |")) {
      print LOGFILE "**** Error! Couldn't start file_check.sql.\n";
      die "**** Error! Couldn't start file_check.sql, stopped";
   }

   # If any Oracle errors were generated, this will print them.
   print LOGFILE <SQL_OUTPUT>;
   close(SQL_OUTPUT);

   $v_error = $?/256;

   if ($v_error == 0) {
      print LOGFILE "---- Success! File $p_file_name has not previously been loaded.\n";
   } else {
      print LOGFILE "**** Error! $p_file_name has already been loaded.\n";
      #print LOGFILE "**** $p_file_name was not loaded. \n";
      #Move_file($nextfile, "/home/purload/pur2001/not_loaded", "print");
      #push @not_loaded_files, ($p_file_name);
      #next;
   }

   return $v_error;
}


##############################################################################
# Intermediate_check determines if there are any data in tables RAWI, ERRORSI,
# and CHANGESI.
# If there are data or some other error, the program aborts.
sub Intermediate_check {
   my $v_error;

   print LOGFILE "Checking if intermediate tables have any data...\n";

   unless (open(SQL_OUTPUT, "sqlplus -S / \@intermediate_check |")) {
      print LOGFILE "**** Error! Couldn't start intermediate_check.sql.\n";
      die "**** Error! Couldn't start intermediate_check.sql, stopped";
   }

   # If any Oracle errors were generated, this will print them.
   print LOGFILE <SQL_OUTPUT>;
   close(SQL_OUTPUT);

   $v_error = $?/256;

   if ($v_error == 0) {
      print LOGFILE "---- Success! intermediate tables are ready.\n";
   } else {
      print LOGFILE "**** Error! some intermediate tables already have data. \n";
      print LOGFILE "**** $p_file_name will not be loaded. \n";
      die "**** Error! Loader ended because of problem with intermediate tables, stopped";
   }
}


###############################################################################
# This function cleans up PUR files received from the county
# and adds the file date and file name to the end of each line.
# It returns the number of lines that were created in the temporary file.

# It replaces a string of one or more control characters by a UNIX end-of-line.
# Three control characters looked for are:
# '\x1A' (control-z)
# '\x0D' (carriage return, \r)
# '\x0A' (newline char, \n)

# End of line in DOS files is \x0D\x0A
# End of line in UNIX files is \x0A
sub Fix_file {
   my $p_nextfile = @_[0];
   my $p_file_name = @_[1];

   my $v_num_lines = 0;
   my $v_line_size = -1;
   my $v_prev_line_size = -1;
   my $v_num_subs;
   my $v_mtime = (stat($p_nextfile))[9];
   my $v_file_date = strftime("%d-%b-%Y %H:%M:%S", localtime($v_mtime));
   my $v_different_lengths = 0;

   print LOGFILE "Strip control characters, add file name and date, and save in \n";
   print LOGFILE "     temporary file $p_file_name.t...\n";

   unless (open(INFILE, "$p_nextfile")) {
      print LOGFILE "**** Error! Couldn't read file $p_nextfile \n";
      print LOGFILE "**** $p_file_name will not be loaded.\n";
      die "**** Error! Couldn't read file $p_nextfile, stopped";
   }

   unless (open(TMPFILE, ">$p_file_name.t")) {
      print LOGFILE "**** Error! Couldn't write to file $p_file_name.t \n";
      print LOGFILE "**** $p_file_name will not be loaded.\n";
      die "**** Error! Couldn't write to file $p_file_name.t, stopped";
   }
   # close LOGFILE,

   while(<INFILE>) {
      # Strip control characters that start a line
      s/^[\x1A\n\r]+//;

      # Substitutes one or more of either x1A, \r, or \n with
      # the file date, file name, and \n
      # If the file does not have a end of line control character
      # at the end, just add the file date and name to the end
      # of the last line.
      if (/[\x1A\n\r]$/) {
         $v_num_subs = s/[\x1A\n\r]+/$v_file_date$p_file_name\n/g;
      } else {
         if (length($_) > 0) {
            $_ .= "$v_file_date$p_file_name\n";
            $v_num_subs = 1;
         } else {
            $v_num_subs = 0;
         }
      }

      # Strip all trailing spaces.
      # Do not strip spaces if there are any rows in a county file
      # which have all spaces in the last field and if
      # no lines have extra trailing spaces.
      if ($strip_trailing_spaces) {
         s/ +$v_file_date$p_file_name\n/$v_file_date$p_file_name\n/;
      }

      $v_num_lines += $v_num_subs;

      $v_prev_line_size = $v_line_size;
      if (length($_) > 0) {
         $v_line_size = length($_) - length("$v_file_date$p_file_name\n");
      }

      if ($v_prev_line_size >= 0 and $v_prev_line_size != $v_line_size) {
         if ($v_different_lengths) {
            print LOGFILE "**** Line $v_num_lines has $v_line_size characters";
         } else {
            $v_different_lengths = 1;
            print LOGFILE "**** Error! Lines in file $p_file_name are not of same length \n";
            print LOGFILE "**** Previous lines had $v_prev_line_size characters\n";
            print LOGFILE "**** Line $v_num_lines has $v_line_size characters";
         }

         if ($v_num_subs > 1) {
            print LOGFILE " (it is actually $v_num_subs lines)\n";
         } else {
            print LOGFILE "\n";
         }
      }
      print TMPFILE $_;
   }

   close(INFILE);
   close(TMPFILE);

   if ($v_different_lengths) {
      print LOGFILE "**** $p_file_name.t has $v_num_lines lines. \n";
      #Move_file($p_nextfile, "/home/purload/pur2001/not_loaded", "print");
      #print LOGFILE "**** $p_file_name was not loaded. \n";
      #push @not_loaded_files, ($p_file_name);
      # delete temporary file.
      #next;
   } else {
      print LOGFILE "---- Success! Temporary file $p_file_name.t created;\n";
      print LOGFILE "     it has $v_num_lines lines each with $v_line_size characters. \n";
   }
   return ($v_num_lines, $v_line_size, $v_different_lengths);
}

##############################################################################
# If the county data file has more than $v_max_num_lines lines
# Split_file will split the file into multiple files.
# Each new file is named as the original file with extenstion
# "_1", "_2", etc.
# This function returns a hash variable containing a list of
# names of all the files created and the number of lines
# in each file.

sub Split_file {
   my $p_file_name = @_[0];
   my $p_orignal_num_lines = @_[1];
   my $p_max_num_lines = @_[2];

   my %v_split_files;
   my $v_split_file;
   my $v_num_split_file;
   my $v_new_num_lines = 0;

   my $v_line_num_original = 0;
   my $v_line_num_new = 0;
   my $v_extension = 0;

   if ($p_orignal_num_lines > $p_max_num_lines) {
      $v_num_split_file = int(($p_orignal_num_lines - 5)/$p_max_num_lines) + 1;
      $v_new_num_lines =  int $p_orignal_num_lines/$v_num_split_file;

      print LOGFILE "The county file $p_file_name contains more than $p_max_num_lines lines. \n";
      print LOGFILE "     It will be split into $v_num_split_file new files each\n";
      print LOGFILE "     containing close to $v_new_num_lines lines and with \n";
      print LOGFILE "     \"_1\", \"_2\", etc added to the end of the file names.\n";

      open ( INPUT_FILE, "< $p_file_name.t" ) ;

      while ($line = <INPUT_FILE>) {
         $v_line_num_original++;

         if ($v_line_num_new == 0) {
            $v_extension++;
            $v_split_file = "$p_file_name"."_"."$v_extension";
            open ( NEW_FILE, "> $v_split_file.t" );
         }

         $v_line_num_new++;
         chop($line);
         print NEW_FILE "$line"."_"."$v_extension\n";

         if ($v_line_num_original >= $v_new_num_lines*$v_extension && $v_extension < $v_num_split_file) {
            $v_split_files{"$v_split_file"} = $v_line_num_new;
            $v_line_num_new = 0;
            close(NEW_FILE);
         }
      }
      close(NEW_FILE);
      close(INPUT_FILE);
      &Remove_temp("$p_file_name.t");
      $v_split_files{"$v_split_file"} = $v_line_num_new;
   } else {
      $v_split_files{"$p_file_name"} = $p_orignal_num_lines;
      close(INPUT_FILE);
   }

   return %v_split_files;
}
```

```
##############################################################################
# Load_raw loads the data into Oracle table raw2001i.
# This uses the appropriate control file for either files with 'C' records
# or 'A' and 'B' records (file names start with "f").  Some files
# use the new format, some the old format.  Which control file
# to use can be determined by the file length.
# Note, that previously accepted another kind of new F file
# (which used load_pur_f_new2.ctl) but we should not accept this file structure.
sub Load_raw {
   my $p_file_name = @_[0];
   my $p_num_lines = @_[1];
   my $p_line_size = @_[2];

   my $v_error = 0;
   my $v_result = 0;
   my $v_control_file;
   my $v_log_dir = '/home/purload/pur2001/load_logs/oracle';

   print LOGFILE "Load county data into RAW table...\n";

   if ($p_line_size == 130) {
      print LOGFILE "---- Uses the old C file format.\n";
      $v_control_file = 'load_pur_c_old.ctl';
   } elsif ($p_line_size == 143) {
      print LOGFILE "---- Uses the new C file format.\n";
      $v_control_file = 'load_pur_c_new.ctl';
   } elsif ($p_line_size == 124) {
      print LOGFILE "---- Uses the old F file format.\n";
      $v_control_file = 'load_pur_f_old.ctl';
   } elsif ($p_line_size == 128) {
      print LOGFILE "---- Uses the new F file format.\n";
      $v_control_file = 'load_pur_f_new.ctl';
   } elsif ($p_line_size == 134) {
      print LOGFILE "---- Uses the CalTrans file format.\n";
      $v_control_file = 'load_pur_caltrans.ctl';
   } else {
      $v_error = 1;
   }

   unless ($v_error) {
      unless (open(SQL_OUTPUT,
         "sqlldr USERID=/ SILENT=HEADER, FEEDBACK CONTROL=$v_control_file DATA=$p_file_name.t ".
         "LOG=$v_log_dir/$p_file_name.log BAD=$v_log_dir/$p_file_name.bad |"))
      {
         $v_result = $?/256;
         print LOGFILE "**** Error! Couldn't start sqlldr, error = $v_result. \n";
         die "**** Error! Couldn't start sqlldr, stopped";
      }

      if (SQL_OUTPUT) {
         # If any Oracle errors were generated, print them.
         print LOGFILE <SQL_OUTPUT>;
         close(SQL_OUTPUT);
         $v_result = $?/256;
      }
   }

   if ($v_error) {
      print LOGFILE "**** Error! $p_file_name.t has the wrong file length.\n";
   } elsif ($v_result) {
      print LOGFILE "**** Error! Error occurred in sqlldr while loading $p_file_name.t.\n";
   } else {
      print LOGFILE "---- Success! Data from $p_file_name was loaded;\n";
      print LOGFILE "     the log file for this load is\n";
      print LOGFILE "     $v_log_dir/$p_file_name.log.\n";

      # Make a record in the LOG table that this data was successfully loaded.
      &Load_log($p_file_name, $p_num_lines);
   }
   return $v_error + $v_result;
}
```

```
###############################################################################
# Check_errors examines each value in RAW for possible errors, flags any errors
# found, makes corrections where possible, records results in ERRORS and CHANGES,
# and produces error reports.
sub Check_errors {
   my $v_error;

   print LOGFILE "Check data for errors...\n";

   unless (open(SQL_OUTPUT, "sqlplus -S / \@check_errors |")) {
      print LOGFILE "**** Error! Couldn't start check_errors.sql.\n";
      die "**** Error! Couldn't start check_errors.sql, stopped";
   }
   unless (<SQL_OUTPUT> eq "No errors.\n") {
      print LOGFILE <SQL_OUTPUT>; # If any Oracle errors were generated, this will print them.
   }

   close(SQL_OUTPUT);

   $v_error = $?/256;

   if ($v_error) {
      print LOGFILE "**** Error! Problem in running error checking procedures. \n",
      die "**** Error! Loader ended because of problem with error checking procedures, stopped";
   } else {
      print LOGFILE "---- Success! Data has been error checked.\n";
   }
}


###############################################################################
# Update_log updates the log2001 table
# by adding the min and max use_no from the current file.
sub Update_log {
   my $p_file_name = @_[0];
   my $v_error;

   print LOGFILE "Update the log2001 table...\n";

   unless (open(SQL_OUTPUT, "sqlplus -S / \@update_log $p_file_name |")) {
      print LOGFILE "**** Error! Couldn't start update_log.sql.\n";
      die "**** Error! Couldn't start update_log.sql, stopped";
   }

   # If any Oracle errors were generated, this will print them.
   print LOGFILE <SQL_OUTPUT>;

   close(SQL_OUTPUT);

   $v_error = $?/256;

   if ($v_error) {
      print LOGFILE "**** Error! Problem in updating log2001. \n",
      die "**** Error! Loader ended because of problem with updating log2001, stopped";
   } else {
      print LOGFILE "---- Success! Log2001 was updated.\n";
   }
}
```

```
###############################################################################
# Error_report creates an error report file
# THe error report is saved in directory /home/purload/pur2001/reports and
# printed to printer in Enforecment.  This report will be sent to the county.
sub Error_report {
   my $p_file_name = @_[0];
   my $v_error;

   print LOGFILE "Generating an error report...\n";

   unless (open(SQL_OUTPUT, "sqlplus -S / \@error_report $p_file_name |")) {
      print LOGFILE "**** Error! Couldn't start error_report.sql.\n";
      die "**** Error! Couldn't start error_report.sql, stopped";
   }

   unless (<SQL_OUTPUT> eq "No errors.\n") {
      print LOGFILE <SQL_OUTPUT>;
   }

   close(SQL_OUTPUT);
   $v_error = $?/256;

   if ($v_error) {
      print LOGFILE "**** Error! Problem in generating error report. \n",
      die "**** Error! Loader ended because of problem with generating error report, stopped";
   } else {
      print LOGFILE "---- Success! Error report was created and saved in file:\n";
      print LOGFILE "     /home/purload/pur2001/reports/error_report_"."$p_file_name"."_2001.txt.\n";

      if ($print_error_report) {
         print LOGFILE "Printing the error report...\n";
         unless (open(PRINT_OUTPUT, "/usr/bin/lp -c -d hp8k_vili
/home/purload/pur2001/reports/error_report_".
            "$p_file_name"."_2001.txt |")) {
            print LOGFILE "**** Error! Couldn't start printing.\n";
            die "**** Error! Couldn't start printing, stopped";
         }
         unless (<PRINT_OUTPUT> eq "No errors.\n") {
            print LOGFILE <PRINT_OUTPUT>;
         }

         close(PRINT_OUTPUT);

         $v_error = $?/256;

         if ($v_error) {
            print LOGFILE "**** Error! Problem in printing error report. \n",
         } else {
            print LOGFILE "---- Success! Error report was printed\n";

            if ($email_file_print) {
               unless (open(SHORT_MAILER, "mailx -s 'PUR file '"."$p_file_name"."' was printed'
$mail_short_log < /dev/null |")) {
                  die "**** Error! Couldn't email printing notice for $p_file_name, stopped";
               }

               print <SHORT_MAILER>;
               close(SHORT_MAILER);
            }

            Move_file("/home/purload/pur2001/reports/error_report_"."$p_file_name"."_2001.txt",
                      "/home/purload/pur2001/reports/printed");
            print LOGFILE "     Error report was moved to directory\n";
            print LOGFILE "     /home/purload/pur2001/reports/printed.\n";
         }
      } else {
         print LOGFILE "     Error report was not printed.\n";

      }
   }
}
```

```
##############################################################################
# Move_intermediates moves all the records from the intermediate table
# raw2001i to raw2001, errors2001i to errors2001,
# changes2001i to changes2001, and delete data from use_error_report2001i.
sub Move_intermediates {
   my $v_error;

   print LOGFILE "Move data from the intermediate tables...\n";

   unless (open(SQL_OUTPUT, "sqlplus -S / \@move_intermediates |")) {
      print LOGFILE "**** Error! Couldn't start move_intermediates.sql.\n";
      die "**** Error! Couldn't start move_intermediates.sql, stopped";
   }
   print LOGFILE <SQL_OUTPUT>; # If any Oracle errors were generated, this will print them.

   close(SQL_OUTPUT);

   $v_error = $?/256;

   if ($v_error) {
      print LOGFILE "**** Error! Problem in moving data from intermediate tables. \n",
      die
      "**** Error! Loader ended due to problem with moving data from intermediate tables, stopped";
   } else {
      print LOGFILE "---- Success! Data from intermediate tables moved.\n";
   }
}


##############################################################################
# Remove_temp deletes the temporary file.
sub Remove_temp {
   my $p_file_name = @_[0];
   my $v_result;

   print LOGFILE "Deleting temporary file $p_file_name...\n";

   $v_result = unlink($p_file_name);

   if ($v_result) {
      print LOGFILE "---- Success! Temporary file $p_file_name deleted.\n";
   } else {
      print LOGFILE "**** Error! Problem in deleting temporary file $p_file_name. \n";
   }
}


##############################################################################
# Clear_intermediates deletes all the records from the intermediate tables
# raw2001i and errors2001i. This is called from Load_raw if there
# was some kind of error during loading data into RAW.
sub Clear_intermediates {
   my $v_error;

   unless (open(SQL_OUTPUT, "sqlplus -S / \@clear_intermediates |")) {
      print LOGFILE "**** Error! Couldn't start clear_intermediates.sql.\n";
      die "**** Error! Couldn't start clear_intermediates.sql, stopped";
   }
   print LOGFILE <SQL_OUTPUT>; # If any Oracle errors were generated, this will print them.

   close(SQL_OUTPUT);

   $v_error = $?/256;

   if ($v_error) {
      print LOGFILE "**** Error! Problem in deleting data from intermediate tables. \n",
      die "**** Error! Loader ended due to problem with deleting data from intermediate tables, stopped";
   }
}
```

```
################################################################################
# Load_log records the name of the file loaded and the number of rows loaded.
# This is called from Load_raw.
sub Load_log {
   my $p_file_name = @_[0];
   my $p_num_lines = @_[1];

   my $v_error;

   unless (open(SQL_OUTPUT, "sqlplus -S / \@load_log $p_file_name $p_num_lines |")) {
      print LOGFILE "**** Error! Couldn't start load_log.sql.\n";
      die "**** Error! Couldn't start load_log.sql, stopped";
   }
   print LOGFILE <SQL_OUTPUT>;

   close(SQL_OUTPUT);

   $v_error = $?/256;

   if ($v_error) {
      &Clear_intermediates;
      print LOGFILE "**** Error! Problem in recording data in log2001. \n",
      die "**** Error! Loader ended because of problem with log2001, stopped";
   }
}



################################################################################
# Move a file $from to directory $to. Variable $from should contain
# full pathname of the file to be moved. Variable $print_move should
# contain value "print" if you want to print a sentence saying
# the file was moved.
# I previously used move function from File::Copy, but this changes the file dates.
sub Move_file {
   #use File::Copy;
   my ($p_from, $p_to, $p_print_move) = @_;

   my $v_file_name = $p_from;
   my $v_dir_name = $p_from;

   $v_file_name =~ s/.*\///;
   $v_dir_name =~ s/(.*)\/.*/\1/;

   #if(move($p_from, $p_to."/$v_file_name")) {
   unless(system("mv $p_from $p_to/$v_file_name")) {
      if ($p_print_move eq "print") {
         print LOGFILE "**** Moved file $v_file_name from $v_dir_name\n";
         print LOGFILE "     to $p_to.\n";
      } elsif ($p_print_move eq "print2") {
         print LOGFILE "Moved file $v_file_name from $v_dir_name\n";
         print LOGFILE "     to $p_to.\n";
      }
   } else {
      print LOGFILE "**** Could not move file $v_file_name from $v_dir_name\n";
      print LOGFILE "     to $p_to.\n";
   }
}
```

```
###############################################################################
# Copy a file $from to directory $to. Variable $from should contain
# full pathname of the file to be copied.
# I previously used File::Copy, but this changes the file dates.
sub Copy_file {
   my ($p_from, $p_to) = @_;

   my $v_file_name = $p_from;
   my $v_dir_name = $p_from;

   $v_file_name =~ s/.*\///;
   $v_dir_name =~ s/(.*)\/.*/\1/;


   if(system("cp -p $p_from $p_to/$v_file_name")) {
      print LOGFILE "**** Could not copy file $v_file_name from $v_dir_name\n";
      print LOGFILE "     to $p_to.\n";
   }
}


###############################################################################
# SQL_check determines if all the SQL files needed to run this program
# and present and can be read.  If not, the program aborts.
# It will also check if any procedures need to be recompiled.  Procedures
# are defined in co_error2001.sql, check_value2001.sql, get_prodno.sql, and outlier.sql.
sub SQL_check {
   my $v_error = 0;

   @sql_files = ("file_check.sql", "intermediate_check.sql",
                 "move_intermediates.sql", "clear_intermediates.sql",
                 "check_errors.sql",  "load_log.sql", "update_log.sql",
                 "error_report.sql", "co_error2001.sql", "check_value2001.sql",
                 "get_prodno.sql", "outlier.sql", "dates.sql",
                 "procedure_validity.sql",
                 "load_pur_c_old.ctl", "load_pur_f_old.ctl",
                 "load_pur_c_new.ctl", "load_pur_f_new.ctl",
                 "load_pur_caltrans.ctl");

   # print LOGFILE "Checking if all needed files are present...\n";

   foreach $sql_file (@sql_files) {
      unless (-e $sql_file) {
         $v_error = 1;
         print LOGFILE "**** Error! The SQL file $sql_file does not exist.\n";
         print LOGFILE "**** This file is needed for this program to run.\n";
         next;
      }
      unless (-r $sql_file) {
         $v_error = 1;
         print LOGFILE "**** Error! The SQL file $sql_file cannot be read.\n";
         print LOGFILE "**** This file is needed for this program to run.\n";
      }
   }

   if ($v_error) {
      die "**** Error! Some needed SQL files are not present or readable, stopped";
   }

   # Check if any procedures need to be recompiled.
   &Check_procedures("get_prodno.sql");
   &Check_procedures("outlier.sql");
   &Check_procedures("check_value2001.sql");
   &Check_procedures("co_error2001.sql");
}
```

```
###############################################################################
# Check_procedures determines if the package in $p_file_name needs to be
# recompiled based on the relative last dates of modifications of the
# package and the package source code.
#
sub Check_procedures {
   my $p_file_name = @_[0];
   my $v_file_date;
   my $v_result;
   my $v_error;

   use POSIX qw(strftime);

   $v_mtime = (stat($p_file_name))[9];
   $v_file_date = strftime("%d-%b-%Y_%H:%M:%S", localtime($v_mtime));

   unless (open(SQL_OUTPUT, "sqlplus -S / \@dates $p_file_name $v_file_date |")) {
      die "**** Error! Couldn't start dates.sql, stopped";
   }
   print LOGFILE <SQL_OUTPUT>;

   close(SQL_OUTPUT);

   $v_result = $?/256;

   if ($v_result == 1) {
      die "**** Error! Loader ended because of problem in getting date, stopped";
   } elsif ($v_result == 2) {
      print LOGFILE "---- Package in $p_file_name needs to be compiled\n";

      # Try to recompile it:
      unless (open(SQL_OUTPUT, "sqlplus -S / \@$p_file_name |")) {
         die "**** Error! Couldn't start $p_file_name, stopped";
      }

      unless (<SQL_OUTPUT> eq "No errors.\n") {
         print LOGFILE <SQL_OUTPUT>;
      }

      close(SQL_OUTPUT);

      $v_error = $?/256;

      if ($v_error) {
         die "**** Error! Loader ended because of problem with $p_file_name, stopped";
      }

      print LOGFILE "---- Package $p_file_name was compiled.\n";

      # Even if no errors appeared here, the procedure could still be invalid.
      # If it is invalid, quit the program.
      unless (open(SQL_OUTPUT, "sqlplus -S / \@procedure_validity $p_file_name |")) {
         die "**** Error! Couldn't start procedure_validity, stopped";
      }

      unless (<SQL_OUTPUT> eq "No errors.\n") {
         print LOGFILE <SQL_OUTPUT>;
      }

      close(SQL_OUTPUT);

      $v_error = $?/256;

      if ($v_error) {
         die "**** Error! Loader ended because procedure $p_file_name is invalid, stopped";
      }

      print LOGFILE "---- Package $p_file_name is valid.\n\n";
   }
}
```

```
##############################################################################
# Email administrators the log file.
#
sub Email_logs {
   unless (open(MAILER, "mailx -s 'Detailed results of loading PUR data' $administrators < $logfile_name
|")) {
      die "**** Error! Couldn't email log file, stopped";
   }

   unless (open(SHORT_MAILER, "mailx -s 'Summary results of loading PUR data' $mail_short_log <
$short_logfile |")) {
      die "**** Error! Couldn't email short log file, stopped";
   }

   print <MAILER>;
   close(MAILER);

   print <SHORT_MAILER>;
   close(SHORT_MAILER);
}


##############################################################################
# Setup performs various preliminary procedures.
#
sub Setup {
   my $v_number_of_files;

   # Check if another process of this program is running.
   # If so, exit.
   unless (open(PROCESS, "ps -ef | grep 'loader.pl' |")) {
      die "**** Error! Couldn't start ps, stopped";
   }

   @result = <PROCESS>;
   close(PROCESS);

   $process_num = 0;
   foreach $process (@result) {
      print $process;

      # Ignore any process involving grep or profile
      # (which appears during cron run)
      unless ($process =~ /grep|profile/) {
         $process_num++;
      }
      if( $process_num > 1) {
         die "**** Found other loader process running, stopped";
      }
   }

   # Move the files back for testing
   #while($nextfile = </home/purload/pur2001/loading/*>) {
   #   $file_name = $nextfile;
   #   $file_name =~ s#.*/##;
   #   rename($nextfile, "/home/purload/pur2001/2001data/$file_name");
   #}

   #while($nextfile = </home/purload/pur2001/loaded/*>) {
   #   $file_name = $nextfile;
   #   $file_name =~ s#.*/##;
   #   rename($nextfile, "/home/purload/pur2001/2001data/$file_name");
   #}

   #while($nextfile = </home/purload/pur2001/not_loaded/*>) {
   #   $file_name = $nextfile;
   #   $file_name =~ s#.*/##;
   #   rename($nextfile, "/home/purload/pur2001/2001data/$file_name");
   #}
   # End of testing move.
   # Get current time:
   use POSIX qw(strftime);
   $current_date = strftime("%d-%b-%Y_%Hh%Mm%Ss", localtime(time));

   # Open log files:
   $logfile_name = "/home/purload/pur2001/load_logs/load_log_"."$current_date".".txt";
   $short_logfile = "short_log.txt";
   open(LOGFILE, ">$logfile_name") or
      die "**** Error! Couldn't create $logfile_name, stopped";
   open(SHORTLOG, ">$short_logfile") or
      die "**** Error! Couldn't create $short_logfile, stopped";

   # Disable buffering so that logfile will contain all messages,
   # even when errors prematurely abort the program.
   select(LOGFILE);
   $| = 1;

   select(SHORTLOG);
   $| = 1;

   print LOGFILE "Results from loading and error checking PUR county files on $current_date.\n\n";
   print SHORTLOG "Results from loading and error checking PUR county files on $current_date.\n\n";

   # Check if all needed files are present.
   &SQL_check;

   # This procedure will traverse the file tree in /Scalos_ora05/data/2001data
   # (for testing use /home/purload/pur2001/2001data) and
   # for each plain file found in any directory
   # make a backup copy in the data directory, and
   # move it to the loading directory.
   # This ignores any files that start with "d" or "D" which are the
   # "D" records.
   use File::Find;
   $v_number_of_files = 0;
   if ($reload) {
      find(\&wanted, '/home/purload/pur2001/2001data');
   } else {
      find(\&wanted, '/Scalos_ora05/data/2001data');
   }

   sub wanted  {
      $file_name = $_;
      $nextfile = $File::Find::name;

      if(-f $nextfile and $file_name =~ /^[cCfF]/) {
         $v_number_of_files++;
         if ($v_number_of_files == 1) {
            print LOGFILE "The following county files were found:\n";
            print SHORTLOG "The following county files were found:\n";
         }
         print LOGFILE "$file_name\n";
         print SHORTLOG "$file_name\n";
         Copy_file($nextfile, "/home/purload/pur2001/data");
         Move_file($nextfile, "/home/purload/pur2001/loading");
      }
   }

   if ($v_number_of_files == 0) {
      # print LOGFILE "No county files were found.\n";
      print SHORTLOG "No county files were found.\n";
      close(LOGFILE);
      close(SHORTLOG);
      unlink($logfile_name);
      exit 1;
   } else {
      print LOGFILE "\nTotal number of files = $v_number_of_files.\n";
      print SHORTLOG "\nTotal number of files = $v_number_of_files.\n";
   }
   return $v_number_of_files;
}
```

```
###############################################################################
# Print_results prints a list of files that were successfully loaded, and
# that were not loaded.
sub Print_results {
   my $p_number_of_files = @_[0];

   my $v_nextfile;
   my $v_num_loaded_files = @loaded_files;
   my $v_num_not_loaded_files = @not_loaded_files;

   print LOGFILE "\n".("_" x 25)." Summary ".("_" x 25)."\n";
   print SHORTLOG "\n".("_" x 60)."\n";

   if ($v_num_loaded_files == $p_number_of_files) {
      print LOGFILE "All files were successfully loaded.\n";
      print SHORTLOG "All files were successfully loaded.\n";
      if ($print_error_report) {
         print LOGFILE "Error reports for all files were printed.\n";
         print SHORTLOG "Error reports for all files were printed.\n";
      }
   } elsif ($v_num_loaded_files > 0 and $v_num_not_loaded_files > 0) {
      print LOGFILE "$v_num_loaded_files files were successfully loaded:\n";
      print SHORTLOG "$v_num_loaded_files files were successfully loaded:\n";

      foreach $v_nextfile (@loaded_files) {
         print LOGFILE "$v_nextfile\n";
         print SHORTLOG "$v_nextfile\n";
      }

      print LOGFILE "\n $v_num_not_loaded_files files were not loaded because of errors:\n";
      print SHORTLOG "\n $v_num_not_loaded_files files were not loaded because of errors:\n";

      foreach $v_nextfile (@not_loaded_files) {
         print LOGFILE "$v_nextfile\n";
         print SHORTLOG "$v_nextfile\n";
      }

      if ($print_error_report) {
         print LOGFILE "\nError reports for successfully loaded files were printed.\n";
         print SHORTLOG "Error reports for successfully loaded files were printed.\n";
      }
   } elsif ($v_num_not_loaded_files == $p_number_of_files) {
      print LOGFILE "None of the files were loaded because of errors.\n";
      print SHORTLOG "None of the files were loaded because of errors.\n";
   } else {
      print LOGFILE "The total number of files processed does not match number found.\n";
      print SHORTLOG "The total number of files processed does not match number found.\n";
   }
}

###############################################################################
# Loaded_files creates a file listing all the county files loaded so far
# for this year.
sub Loaded_files {
   my $v_error;

   unless (open(SQL_OUTPUT, "sqlplus -S / \@loaded_files |")) {
      print LOGFILE "**** Error! Couldn't start loaded_files.sql.\n";
      die "**** Error! Couldn't start loaded_files.sql, stopped";
   }
   print LOGFILE <SQL_OUTPUT>; # If any Oracle errors were generated, this will print them.

   close(SQL_OUTPUT);
   $v_error = $?/256;

   if ($v_error) {
      print LOGFILE "**** Error! Problem in creating file listed all loaded county files. \n",
      die "**** Error! Loader ended due to problem creating file listed all loaded county files, stopped";
   }
}
```

```
###############################################################################
# Handle_errors carries out procedures after an error is found.
# $p_file_to_move should be file (including pathname) of file.
sub Handle_error {
   my $p_error = @_[0];
   my $p_file_to_move = @_[1];
   my $p_file_name = @_[2];

   #my $v_file_name =~ s/.*\///; # Remove the part before last slash.
   #$v_file_name =~ s/\.t$//; # If the name has a ".t" on end, remove it.

   if ($p_error) {
      print LOGFILE "**** $p_file_name was not loaded. \n";
      push @not_loaded_files, $p_file_name;
      &Move_file($p_file_to_move, "/home/purload/pur2001/not_loaded", "print");
      &Clear_intermediates;
      next;
   }
}
```

The SQL and PL/SQL scripts:
Check_value2001.sql

```
/*
   Package for all the error checking procedures
   Each field in the PUR has a separate procedure to check for errors.
   Each procedure needs the use_no and p_value and
   returns the error_code, converted new value with correct type
   (variable with same name as field name in the PUR).
 */

set termout on
SET document off
SET FEEDBACK ON
SET verify off
SET pause off
set serveroutput on size 1000000 format word_wrapped
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

CREATE OR REPLACE PACKAGE Check_value2001 AS
   v_pur_year        NUMBER(4) := 2001;
   TYPE     use_no_array_type IS TABLE OF pur2001.use_no%TYPE
               INDEX BY BINARY_INTEGER;

   PROCEDURE Record_id
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_record_id OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Batch_no
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_batch_no OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Process_mt
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_process_mt OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Process_yr
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_process_yr OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE County_cd
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_county_cd OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Site_code
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_site_code OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Applic_month
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_month OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Applic_day
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_day OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Applic_year
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_month IN NUMBER,
       p_applic_day IN NUMBER,
       p_applic_year OUT NUMBER,
       p_estimated_date OUT DATE,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Applic_dt
      (p_use_no IN NUMBER,
       p_applic_month IN NUMBER,
       p_applic_day IN NUMBER,
       p_applic_year IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_dt OUT DATE,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Document_no
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_applic_day IN NUMBER,
       p_value IN VARCHAR2,
       p_document_no OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE Summary_cd
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_summary_cd OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);
```

```
PROCEDURE Site_loc_id
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_site_loc_id OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Inconsistent_site_loc_id
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_county_cd IN VARCHAR2,
    p_grower_id IN VARCHAR2,
    p_site_loc_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_acre_planted IN NUMBER,
    p_unit_planted IN VARCHAR2,
    p_base_ln_mer IN VARCHAR2,
    p_township IN VARCHAR2,
    p_tship_dir IN VARCHAR2,
    p_range IN VARCHAR2,
    p_range_dir IN VARCHAR2,
    p_section IN VARCHAR2,
    p_field_id OUT NUMBER,
    p_site_loc_id_state OUT VARCHAR2,
    p_operator_id OUT VARCHAR2,
    p_this_mtrs OUT mtrs_type,
    p_this_acres OUT NUMBER,
    p_previous_mtrs OUT mtrs_type,
    p_previous_acres OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Cedts_ind
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_cedts_ind OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Qualify_cd
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_qualify_cd OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Planting_seq
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_planting_seq OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Section
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_section OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Township
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_township OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Tship_dir
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_tship_dir OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Range
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_range OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Range_dir
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_range_dir OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Base_ln_mer
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_base_ln_mer OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Mtrs
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_base_ln_mer IN OUT VARCHAR2,
    p_township IN OUT VARCHAR2,
    p_tship_dir IN OUT VARCHAR2,
    p_range IN OUT VARCHAR2,
    p_range_dir IN OUT VARCHAR2,
    p_section IN OUT VARCHAR2,
    p_ca_mtrs_acres OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);
```

```
PROCEDURE Comtrs
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_county_cd IN VARCHAR2,
    p_base_ln_mer IN OUT VARCHAR2,
    p_township IN OUT VARCHAR2,
    p_tship_dir IN OUT VARCHAR2,
    p_range IN OUT VARCHAR2,
    p_range_dir IN OUT VARCHAR2,
    p_section IN OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Aer_gnd_ind
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_aer_gnd_ind OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Applic_time
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_applic_time OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Grower_id
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_county_cd IN VARCHAR2,
    p_license_no IN VARCHAR2,
    p_applic_year IN NUMBER,
    p_value IN VARCHAR2,
    p_grower_id OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Grower_id_error
   (p_grower_id IN OUT VARCHAR2, p_county_cd IN varchar2,
    p_applic_year IN NUMBER, p_error_code OUT VARCHAR2,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

PROCEDURE License_no
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_grower_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_license_no OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Mfg_firmno
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_mfg_firmno OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Label_seq_no
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_label_seq_no OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Revision_no
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_revision_no OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Reg_firmno
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_reg_firmno OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Amt_prd_used
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_amt_prd_used OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Unit_of_meas
   (p_use_no IN NUMBER,
    p_value IN VARCHAR2,
    p_unit_of_meas OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Prodno
   (p_use_no IN NUMBER,
    p_mfg_firmno IN NUMBER,
    p_label_seq_no IN NUMBER,
    p_revision_no IN VARCHAR2,
    p_reg_firmno IN NUMBER,
    p_prodno OUT NUMBER,
    p_new_revision_no OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Prod_site
   (p_use_no IN NUMBER,
    p_mfg_firmno IN NUMBER,
    p_label_seq_no IN NUMBER,
    p_revision_no IN VARCHAR2,
    p_reg_firmno IN NUMBER,
    p_unit_of_meas IN VARCHAR2,
    p_site_code IN NUMBER,
    p_applic_dt IN DATE,
    p_prodno IN OUT NUMBER,
    p_new_revision_no OUT VARCHAR2,
    p_new_reg_firmno OUT NUMBER,
    p_other_product OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Spec_gravity
   (p_use_no IN NUMBER,
    p_prodno IN NUMBER,
    p_unit_of_meas IN VARCHAR2,
    p_spec_gravity OUT NUMBER,
    p_formula_cd OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Unit_of_meas2
   (p_use_no IN NUMBER,
    p_unit_of_meas IN VARCHAR2,
    p_formula_cd IN VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Lbs_prd_used
   (p_use_no IN NUMBER,
    p_prodno IN NUMBER,
    p_amt_prd_used IN NUMBER,
    p_unit_of_meas IN VARCHAR2,
    p_formula_cd IN VARCHAR2,
    p_spec_gravity IN NUMBER,
    p_lbs_prd_used OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Acre_treated
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_value IN VARCHAR2,
    p_acre_treated OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Unit_treated
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_value IN VARCHAR2,
    p_unit_treated OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Acre_unit_treated
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_acre_treated IN NUMBER,
    p_unit_treated IN VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Acre_treated_section
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_ca_mtrs_acres IN NUMBER,
    p_acre_treated IN OUT NUMBER,
    p_unit_treated IN VARCHAR2,
    p_grower_id IN VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Acre_treated_nonag
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_acre_treated IN NUMBER,
    p_unit_treated IN OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Acre_planted
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_acre_planted OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Unit_planted
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_unit_planted OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Acre_planted_section
   (p_use_no IN NUMBER,
    p_site_code IN NUMBER,
    p_ca_mtrs_acres IN NUMBER,
    p_acre_planted IN OUT NUMBER,
    p_unit_planted IN VARCHAR2,
    p_grower_id IN VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Acre_treated_planted
   (p_use_no IN NUMBER,
    p_unit_planted IN VARCHAR2,
    p_unit_treated IN OUT VARCHAR2,
    p_acre_planted IN NUMBER,
    p_acre_treated IN OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);
```

```
PROCEDURE Unit_treated_planted
   (p_use_no IN NUMBER,
    p_acre_treated IN NUMBER,
    p_acre_planted IN NUMBER,
    p_unit_treated IN OUT VARCHAR2,
    p_unit_planted IN OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Applic_cnt
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_value IN VARCHAR2,
    p_applic_cnt OUT NUMBER,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Outliers_nn
   (p_use_no IN NUMBER,
    p_record_id VARCHAR2,
    p_prodno IN NUMBER,
    p_site_code IN NUMBER,
    p_lbs_prd_used IN NUMBER,
    p_acre_treated IN NUMBER,
    p_unit_treated IN VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Outliers
   (p_use_no IN NUMBER,
    p_record_id VARCHAR2,
    p_prodno IN NUMBER,
    p_site_code IN NUMBER,
    p_amt_prd_used IN OUT NUMBER,
    p_lbs_prd_used IN OUT NUMBER,
    p_acre_treated IN OUT NUMBER,
    p_unit_treated IN OUT VARCHAR2,
    p_acre_planted IN NUMBER,
    p_unit_planted IN VARCHAR2,
    p_estimated_field OUT VARCHAR2,
    p_criteria OUT VARCHAR2,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2);

PROCEDURE Duplicates
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_county_cd IN VARCHAR2,
    p_grower_id IN VARCHAR2,
    p_site_loc_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_qualify_cd IN NUMBER,
    p_prodno IN NUMBER,
    p_amt_prd_used IN NUMBER,
    p_unit_of_meas IN VARCHAR2,
    p_acre_treated IN NUMBER,
    p_unit_treated IN VARCHAR2,
    p_acre_planted IN NUMBER,
    p_unit_planted IN VARCHAR2,
    p_applic_dt IN DATE,
    p_use_no_array OUT use_no_array_type,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2);

   PROCEDURE No_error
      (p_error_code OUT VARCHAR2, p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2);

   PROCEDURE Invalid_error
      (p_value IN VARCHAR2, p_out_value OUT NUMBER,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Invalid_char_error
      (p_value IN VARCHAR2, p_out_value OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Invalid_num_error
      (p_value IN VARCHAR2, p_out_value OUT NUMBER,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Invalid_date_error
      (p_value IN VARCHAR2, p_out_value OUT DATE,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Probable_estimated_error
      (p_estimate IN VARCHAR2, p_out_value OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Probable_error
      (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Probable_corrected_error
      (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Not_allowed
      (p_value IN VARCHAR2, p_out_value OUT VARCHAR2, p_error_code OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Possible_invalid_error
      (p_value IN VARCHAR2, p_out_value OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Possible_invalid_char_error
      (p_value IN VARCHAR2, p_out_value OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Possible_error
      (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Duplicate_error
      (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);


   PROCEDURE Other_error
      (p_value IN VARCHAR2, p_out_value OUT VARCHAR2,
       p_error_code IN OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2,
       p_use_no IN NUMBER);

   FUNCTION Null_or_same (p_value IN VARCHAR2)
      RETURN VARCHAR2;

END Check_value2001;
/
show errors
```

```
CREATE OR REPLACE PACKAGE BODY Check_value2001 AS
   PROCEDURE Other_exceptions(p_use_no IN NUMBER, p_error_code IN BINARY_INTEGER);

/*  Error code 1: check record_id
 */
   PROCEDURE Record_id
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_record_id OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 1;
      p_require_type := 'required';
      p_record_id := p_value;

      IF p_record_id IS NULL OR
         p_record_id NOT IN ('1','2','4','A','B','C')
      THEN
         Invalid_char_error(p_value, p_record_id, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_record_id, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_record_id, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Record_id;


/* Error code 2: check batch_no
 */
   PROCEDURE Batch_no
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_batch_no OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 2;
      p_require_type := 'required';
      p_batch_no := TO_NUMBER(p_value);

      IF p_batch_no IS NULL OR p_batch_no = 0 THEN
         Invalid_num_error(p_value, p_batch_no, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_num_error(p_value, p_batch_no, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_batch_no, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Batch_no;
```

```
/* Error code 3: check process_mt
 */
   PROCEDURE Process_mt
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_process_mt OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 3;
      p_require_type := 'required';
      p_process_mt := TO_NUMBER(p_value);

      IF p_process_mt IS NULL OR p_process_mt NOT BETWEEN 1 AND 12 THEN
         Invalid_error(p_value, p_process_mt, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_process_mt, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_process_mt, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Process_mt;


/* Error code 4: check process_yr
 */
   PROCEDURE Process_yr
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_process_yr OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 4;
      p_require_type := 'required';
      p_process_yr := TO_NUMBER(p_value);

      IF p_process_yr BETWEEN 50 AND 99 THEN
         p_process_yr := 1900 + p_process_yr;
      ELSIF p_process_yr BETWEEN 0 AND 49 THEN
         p_process_yr :=2000 + p_process_yr;
      END IF;

      IF p_process_yr IS NULL OR
         p_process_yr NOT BETWEEN v_pur_year AND v_pur_year+2
      THEN
         Invalid_error(p_value, p_process_yr, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_process_yr, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_process_yr, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Process_yr;
```

```
/* Error code 5: check county_cd
 */
   PROCEDURE County_cd
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_county_cd OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_county_cd   NUMBER(2);
   BEGIN
      p_error_code := 5;
      p_require_type := 'required';
      v_county_cd := TO_NUMBER(p_value);

      IF v_county_cd IS NULL OR v_county_cd NOT BETWEEN 1 AND 58 THEN
         Invalid_char_error(p_value, p_county_cd, p_error_type, p_replace_type);
      ELSE
         /* Note LTRIM is needed because TO_CHAR adds an extra space to beginning */
         p_county_cd := LTRIM(TO_CHAR(v_county_cd, '09'));
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_county_cd, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_county_cd, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END County_cd;

/* Error code 13, 14: check site_code
 */
   PROCEDURE Site_code
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_site_code OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_dummy       VARCHAR2(11);
   BEGIN
      p_require_type := 'required';
      p_site_code := TO_NUMBER(p_value);

      IF p_site_code IS NULL OR p_site_code = 99999 THEN
         p_error_code := 13;
         Invalid_num_error(p_value, p_site_code, p_error_type, p_replace_type);
      ELSE
         -- pur_site is a table of only the sites that allowed
         -- to be reported on the PUR forms.
         SELECT  site_code
         INTO    v_dummy
         FROM    pur_site
         WHERE   site_code = p_site_code AND
                 extra IS NULL;

         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;
   EXCEPTION
      WHEN VALUE_ERROR THEN
         p_error_code := 13;
         Invalid_num_error(p_value, p_site_code, p_error_type, p_replace_type);
      WHEN NO_DATA_FOUND THEN
         p_error_code := 14;
         Invalid_num_error(p_value, p_site_code, p_error_type, p_replace_type);
      WHEN OTHERS THEN
```

```
         Other_error(p_value, p_site_code, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Site_code;


/* Error code 15: check applic_month
 */
   PROCEDURE Applic_month
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_month OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 15;
      p_require_type := 'required';
      p_applic_month := TO_NUMBER(p_value);

      IF p_applic_month IS NULL OR p_applic_month NOT BETWEEN 1 AND 12 THEN
         Invalid_error(p_value, p_applic_month, p_error_type, p_replace_type);
         p_replace_type := 'null';
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_applic_month, p_error_type, p_replace_type);
         /* replace type is always null because we replace applic_dt with null,
            not just the month part of the data
          */
         p_replace_type := 'null';
      WHEN OTHERS THEN
         Other_error(p_value, p_applic_month, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Applic_month;

/* Error code 16: check applic_day
 */
   PROCEDURE Applic_day
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_day OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 16;
      p_require_type := 'required';
      p_applic_day := TO_NUMBER(p_value);

      IF p_applic_day IS NULL OR p_applic_day NOT BETWEEN 1 AND 31 THEN
         Invalid_error(p_value, p_applic_day, p_error_type, p_replace_type);
         p_replace_type := 'null';
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_applic_day, p_error_type, p_replace_type);
         p_replace_type := 'null';
      WHEN OTHERS THEN
         Other_error(p_value, p_applic_day, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Applic_day;
```

```
/* Error code 17: check applic_year
 */
   PROCEDURE Applic_year
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_month IN NUMBER,
       p_applic_day IN NUMBER,
       p_applic_year OUT NUMBER,
       p_estimated_date OUT DATE,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 17;
      p_require_type := 'required';
      p_applic_year := TO_NUMBER(p_value);

      IF p_applic_year BETWEEN 50 AND 99 THEN
         p_applic_year := 1900 + p_applic_year;
      ELSIF p_applic_year BETWEEN 0 AND 49 THEN
         p_applic_year :=2000 + p_applic_year;
      END IF;

      IF p_applic_year IS NULL OR
         p_applic_year <> v_pur_year
      THEN
         IF p_applic_day IS NOT NULL AND p_applic_month IS NOT NULL THEN
            Probable_estimated_error(v_pur_year, p_applic_year, p_error_type, p_replace_type);
            p_estimated_date := TO_DATE(LTRIM(TO_CHAR(p_applic_month, '09'))||
                              LTRIM(TO_CHAR(p_applic_day, '09'))||
                              p_applic_year, 'MMDDYYYY');
         ELSE
            Invalid_error(p_value, p_applic_year, p_error_type, p_replace_type);
            p_replace_type := 'null';
         END IF;
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Probable_estimated_error(v_pur_year, p_applic_year, p_error_type, p_replace_type);
         p_estimated_date := TO_DATE(LTRIM(TO_CHAR(p_applic_month, '09'))||
                              LTRIM(TO_CHAR(p_applic_day, '09'))||
                              p_applic_year, 'MMDDYYYY');
      WHEN OTHERS THEN
         Other_error(p_value, p_applic_year, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Applic_year;


/* Error code 51: check applic_dt
   This flags error only if no previous errors were found for application
   month, day, or year.

   Must be called after checks for applic_month, applic_day, and applic_yr.
 */
   PROCEDURE Applic_dt
      (p_use_no IN NUMBER,
       p_applic_month IN NUMBER,
       p_applic_day IN NUMBER,
       p_applic_year IN NUMBER,
       p_value IN VARCHAR2,
       p_applic_dt OUT DATE,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      e_invalid_day_of_month  EXCEPTION;
      PRAGMA EXCEPTION_INIT(e_invalid_day_of_month, -1839);
   BEGIN
      p_error_code := 51;
      p_require_type := 'required';
      p_applic_dt := NULL;

      IF p_applic_month IS NOT NULL AND p_applic_day IS NOT NULL AND
         p_applic_year IS NOT NULL
      THEN
         p_applic_dt :=
            TO_DATE(LTRIM(TO_CHAR(p_applic_month, '09'))||
                    LTRIM(TO_CHAR(p_applic_day, '09'))||
                    p_applic_year, 'MMDDYYYY');

         IF p_applic_dt >= SYSDATE THEN
            Invalid_date_error(p_value, p_applic_dt, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;

      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN e_invalid_day_of_month OR VALUE_ERROR THEN
         Invalid_date_error(p_value, p_applic_dt, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_applic_dt, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Applic_dt;


/* Error code 34: check document_no

   Must be called after checks for p_record_id and applic_day.
 */
   PROCEDURE Document_no
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_applic_day IN NUMBER,
       p_value IN VARCHAR2,
       p_document_no OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 34;

      IF p_record_id = '2' AND p_applic_day = 28 THEN
         -- This is a CalTrans record, which does not have a document_no
         p_require_type := 'not_allowed';
      ELSE
         p_require_type := 'required';
      END IF;

      IF p_require_type = 'required'  THEN
         p_document_no := TO_NUMBER(p_value);

         IF p_document_no IS NULL THEN
            Invalid_num_error(p_value, p_document_no, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;

      ELSE
```

```
         Not_allowed(p_value, p_document_no, p_error_code,
                      p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_num_error(p_value, p_document_no, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_document_no, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Document_no;


/* Error code 35: check summary_cd
 */
   PROCEDURE Summary_cd
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_summary_cd OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 35;
      p_require_type := 'required';
      p_summary_cd := TO_NUMBER(p_value);

      IF p_summary_cd IS NULL OR p_summary_cd = 0 THEN
         Invalid_num_error(p_value, p_summary_cd, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_num_error(p_value, p_summary_cd, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_summary_cd, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Summary_cd;


/* Error code 62: check site_loc_id

   Must be called after checks for p_record_id.
 */
   PROCEDURE Site_loc_id
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_site_loc_id OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 62;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;
      IF p_require_type = 'required'  THEN
         p_site_loc_id := RTRIM(LTRIM(p_value));

         IF p_value IS NULL OR p_value = '        ' THEN
            Invalid_char_error(p_value, p_site_loc_id, p_error_type, p_replace_type);
         ELSIF p_site_loc_id <> p_value THEN
            Probable_corrected_error(p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed'  THEN
         Not_allowed(p_value, p_site_loc_id, p_error_code,
                      p_error_type, p_replace_type);
      ELSE /* p_require_type = 'unknown' */
         p_site_loc_id := RTRIM(LTRIM(p_value));

         IF p_value IS NULL OR p_value = '        ' THEN
            Possible_invalid_char_error(p_value, p_site_loc_id, p_error_type, p_replace_type);
         ELSIF p_site_loc_id <> p_value THEN
            Probable_corrected_error(p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_site_loc_id, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_site_loc_id, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Site_loc_id;

   /* error code 63: Inconsistent site_loc_ids. Different records
      with same ag field (id by grower_id, site_loc_id, and site_code)
      have different MTRS.

      Must be called after checks for record_id, county_cd, site_loc_id, grower_id,
      Site_code, MTRS, acre_planted, unit_planted.
    */
   PROCEDURE Inconsistent_site_loc_id
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_county_cd IN VARCHAR2,
       p_grower_id IN VARCHAR2,
       p_site_loc_id IN VARCHAR2,
       p_site_code IN NUMBER,
       p_acre_planted IN NUMBER,
       p_unit_planted IN VARCHAR2,
       p_base_ln_mer IN VARCHAR2,
       p_township IN VARCHAR2,
       p_tship_dir IN VARCHAR2,
       p_range IN VARCHAR2,
       p_range_dir IN VARCHAR2,
       p_section IN VARCHAR2,
       p_field_id OUT NUMBER,
       p_site_loc_id_state OUT VARCHAR2,
       p_operator_id OUT VARCHAR2,
       p_this_mtrs OUT mtrs_type,
       p_this_acres OUT NUMBER,
       p_previous_mtrs OUT mtrs_type,
       p_previous_acres OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_inconsistent_field VARCHAR2(1);
      v_dummy              VARCHAR2(1);
   BEGIN
      /* Need to change error code for cedts_ind
    */
   p_error_code := 63;

   IF p_record_id IN ('1','4','A','B') THEN
      p_require_type := 'required';
   ELSIF p_record_id IN ('2','C') THEN
      p_require_type := 'not_allowed';
   ELSIF p_record_id IS NULL THEN
      p_require_type := 'unknown';
   ELSE
      p_require_type := 'unknown';
   END IF;


   IF p_require_type = 'required' THEN
      /* Get the mtrs and acres_planted for the current record.
         Convert square feet acre_planted values to acres.
       */
      p_this_mtrs :=  mtrs_type(p_base_ln_mer, p_township, p_tship_dir,
                                p_range, p_range_dir, p_section);

      /* Need to round to two decimel places because in the Oracle tables
         acres is stored to 2 decimal places, while the division
         here can create a number with values beyond 2 decimal places--
         despite the fact that the the number format of the variable
         passed to this function has a scale of 2.
       */
      IF p_unit_planted = 'A' THEN
         p_this_acres := ROUND(p_acre_planted, 2);
      ELSIF p_unit_planted = 'S' THEN
         p_this_acres := ROUND(p_acre_planted/43560, 2);
      ELSE
         p_this_acres := NULL;
      END IF;

      /* In Alameda County, some operators get two grower_ids,
         one ending in 'R' and another ending in 'U';
         they should really have only one which can get be
         created by stripping off the R or U.
       */
      IF p_county_cd = '01' AND
         SUBSTR(p_grower_id, LENGTH(p_grower_id), 1) IN ('R', 'U')
      THEN
         p_operator_id := SUBSTR(p_grower_id, 5, LENGTH(p_grower_id) - 5);
      ELSE
         p_operator_id := SUBSTR(p_grower_id, 5);
      END IF;

      IF p_operator_id IS NULL THEN
         p_operator_id := '-1';
      END IF;

      /* See if there were previous records for this ag field;
         if so, get the information about the field.
       */

      BEGIN
         SELECT   field_id, mtrs, acres_planted,
                  inconsistent_field
         INTO     p_field_id, p_previous_mtrs, p_previous_acres,
                  v_inconsistent_field
         FROM     fields2001
         WHERE    operator_id = p_operator_id AND
                  site_loc_id = p_site_loc_id AND
                  site_code = p_site_code AND
                  county_cd = p_county_cd;
      EXCEPTION
         WHEN NO_DATA_FOUND THEN
            p_field_id := NULL;
         WHEN OTHERS THEN
            RAISE;
         END;

         /* Summary of following algorithm; note that all the
            updating actions are carried out in Co_error2001.Update_fields():

            If this is the first record that used this ag field,
            then add new record to fields2001;

            else (i.e. there are other records for this ag field)
               if this field was consistent before and
                  if it is still consistent,
                  then update fields2001;

                  if this is the first inconsistent found,
                  then determine best estimate for mtrs and acres
                     planted, add new record to field_mtrs_acres table,
                     mark as inconsistent_mtrs
               else (previously found inconsistent mtrs for this field)
                  if this combination of mtrs and acres is already in the
                     list of values for this field
                  then increase the count for this mtrs-acres by 1;

                  else (it is not in the list)
                  then add a new row in mtrs_acres for this mtrs-acres;

                  Determine which acres and mtrs values are most likely correct and
                  Update the fields table with the new values.
          */

         IF p_field_id IS NULL THEN
            p_site_loc_id_state := 'first_record';
            No_error(p_error_code, p_error_type, p_replace_type);

         ELSE --  Other applications were reported for this ag field

            IF v_inconsistent_field IS NULL THEN --Field was consistent before
               IF p_previous_mtrs = p_this_mtrs AND
                  NVL(p_previous_acres, -1) = NVL(p_this_acres, -1)
               THEN
                  p_site_loc_id_state := 'still_consistent';
                  No_error(p_error_code, p_error_type, p_replace_type);
               ELSE
                  p_site_loc_id_state := 'first_inconsistent';
                  Possible_error(p_error_type, p_replace_type);
               END IF;
            ELSE
               p_site_loc_id_state := 'previous_inconsistent';
               Possible_error(p_error_type, p_replace_type);
            END IF;

         END IF;

      ELSE
         p_field_id := NULL;
         p_site_loc_id_state := 'not_allowed';
         p_this_mtrs := NULL;
         p_this_acres := NULL;
         p_previous_mtrs := NULL;
         p_previous_acres := NULL;
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(v_dummy, v_dummy, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Inconsistent_site_loc_id;
```

```
/* Error code 64: check cedts_ind
 */
   PROCEDURE Cedts_ind
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_cedts_ind OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 64;
      p_require_type := 'required';
      IF p_value IN ('0', ' ') THEN
         p_cedts_ind := NULL;
      ELSE
         p_cedts_ind := p_value;
      END IF;

      IF p_cedts_ind <> 'E' THEN
         Invalid_char_error(p_value, p_cedts_ind, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_cedts_ind, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_cedts_ind, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Cedts_ind;


/* Error code 65: check qualify_cd
 */
   PROCEDURE Qualify_cd
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_qualify_cd OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 65;
      p_require_type := 'optional';
      p_qualify_cd := TO_NUMBER(p_value);

      No_error(p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_num_error(p_value, p_qualify_cd, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_qualify_cd, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Qualify_cd;
```

```
/* Error code 66: check planting_seq
   Planting_sec is optional for prod ag records.  No value allowed for other records.

   Must be called after checks for record_id.
 */
   PROCEDURE Planting_seq
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_planting_seq OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 66;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'optional';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'optional' OR p_require_type = 'unknown'  THEN
         p_planting_seq := TO_NUMBER(p_value);

         IF p_planting_seq = 0 THEN
            p_planting_seq := NULL;
            --Assumes value of 0 means no value.
         END IF;

         IF p_planting_seq NOT BETWEEN 1 AND 9 THEN
            Invalid_num_error(p_value, p_planting_seq, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;

      ELSE  /* p_require_type = 'not_allowed' record_id is 2 or C. */
         Not_allowed(p_value, p_planting_seq, p_error_code,
                      p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_num_error(p_value, p_planting_seq, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_planting_seq, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Planting_seq;
```

```
/* Error code 6: check section

   Must be called after checks for record_id.
 */
   PROCEDURE Section
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_section OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_section   NUMBER(2);
   BEGIN
      p_error_code := 6;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         v_section := TO_NUMBER(p_value);

         IF v_section IS NULL OR v_section NOT BETWEEN 1 AND 36 THEN
            Invalid_char_error(p_value, p_section, p_error_type, p_replace_type);
         ELSE
            p_section := LTRIM(TO_CHAR(v_section, '09'));
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
            Not_allowed(p_value, p_section, p_error_code,
                        p_error_type, p_replace_type);
      ELSE
         v_section := TO_NUMBER(p_value);

         IF v_section = 0 OR v_section IS NULL THEN
            Possible_invalid_char_error(p_value, p_section, p_error_type, p_replace_type);
         ELSIF v_section NOT BETWEEN 1 AND 36 THEN
            Invalid_char_error(p_value, p_section, p_error_type, p_replace_type);
         ELSE
            p_section := LTRIM(TO_CHAR(v_section, '09'));
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;
   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_section, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_section, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Section;
```

```
/* Error code 7: check township

   Must be called after checks for record_id.
 */
   PROCEDURE Township
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_township OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_township   NUMBER(2);
   BEGIN
      p_error_code := 7;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         v_township := TO_NUMBER(p_value);

         IF v_township IS NULL OR v_township NOT BETWEEN 1 AND 48 THEN
            Invalid_char_error(p_value, p_township, p_error_type, p_replace_type);
         ELSE
            p_township := LTRIM(TO_CHAR(v_township, '09'));
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
         Not_allowed(p_value, p_township, p_error_code,
                      p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         v_township := TO_NUMBER(p_value);

         IF v_township = 0 OR v_township IS NULL THEN
            Possible_invalid_char_error(p_value, p_township, p_error_type, p_replace_type);
         ELSIF v_township NOT BETWEEN 1 AND 48 THEN
            Invalid_char_error(p_value, p_township, p_error_type, p_replace_type);
         ELSE
            p_township := LTRIM(TO_CHAR(v_township, '09'));
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_township, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_township, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Township;
```

```
/* Error code 8: check tship_dir

   Must be called after checks for record_id.
*/
   PROCEDURE Tship_dir
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_tship_dir OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 8;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_tship_dir := p_value;

         IF p_tship_dir IS NULL OR p_tship_dir NOT IN ('N','S') THEN
            Invalid_char_error(p_value, p_tship_dir, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
            Not_allowed(p_value, p_tship_dir, p_error_code,
                        p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         p_tship_dir := p_value;

         IF p_tship_dir = '0' OR p_tship_dir IS NULL THEN
            Possible_invalid_char_error(p_value, p_tship_dir, p_error_type, p_replace_type);
         ELSIF p_tship_dir NOT IN ('N','S') THEN
            Invalid_char_error(p_value, p_tship_dir, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_tship_dir, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_tship_dir, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Tship_dir;
```

```
/* Error code 9: check range

   Must be called after checks for record_id.
*/
   PROCEDURE Range
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_range OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_range   NUMBER(2);
   BEGIN
      p_error_code := 9;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         v_range := TO_NUMBER(p_value);

         IF v_range IS NULL OR v_range NOT BETWEEN 1 AND 47 THEN
            Invalid_char_error(p_value, p_range, p_error_type, p_replace_type);
         ELSE
            p_range := LTRIM(TO_CHAR(v_range, '09'));
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
         Not_allowed(p_value, p_range, p_error_code,
                     p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         v_range := TO_NUMBER(p_value);

         IF v_range = 0 OR v_range IS NULL THEN
            Possible_invalid_char_error(p_value, p_range, p_error_type, p_replace_type);
         ELSIF v_range NOT BETWEEN 1 AND 47 THEN
            Invalid_char_error(p_value, p_range, p_error_type, p_replace_type);
         ELSE
            p_range := LTRIM(TO_CHAR(v_range, '09'));
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_range, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_range, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Range;
```

```
/* Error code 10: check range_dir

   Must be called after checks for record_id.
*/
   PROCEDURE Range_dir
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_range_dir OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 10;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_range_dir := p_value;

         IF p_range_dir IS NULL OR p_range_dir NOT IN ('W','E') THEN
            Invalid_char_error(p_value, p_range_dir, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
            Not_allowed(p_value, p_range_dir, p_error_code,
                        p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         p_range_dir := p_value;

         IF p_range_dir = '0' OR p_range_dir IS NULL THEN
            Possible_invalid_char_error(p_value, p_range_dir, p_error_type, p_replace_type);
         ELSIF p_range_dir NOT IN ('W','E') THEN
            Invalid_char_error(p_value, p_range_dir, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_range_dir, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_range_dir, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Range_dir;
```

```
/* Error code 11: check base_ln_mer

   Must be called after checks for record_id.
*/
   PROCEDURE Base_ln_mer
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_base_ln_mer OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 11;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_base_ln_mer := p_value;

         IF p_base_ln_mer IS NULL OR p_base_ln_mer NOT IN ('H','M','S') THEN
            Invalid_char_error(p_value, p_base_ln_mer, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
         Not_allowed(p_value, p_base_ln_mer, p_error_code,
                          p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         p_base_ln_mer := p_value;

         IF p_base_ln_mer = '0' OR p_base_ln_mer IS NULL THEN
            Possible_invalid_char_error(p_value, p_base_ln_mer, p_error_type, p_replace_type);
         ELSIF p_base_ln_mer NOT IN ('H','M','S') THEN
            Invalid_char_error(p_value, p_base_ln_mer, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_base_ln_mer, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_base_ln_mer, p_error_code,
                         p_error_type, p_replace_type, p_use_no);
   END Base_ln_mer;
```

```
/* Error code 67: check that mtrs exist in California

   Must be called after checks for record_id, base_ln_mer, township, tship_dir,
   range, range_dir, and section.
*/
   PROCEDURE Mtrs
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_base_ln_mer IN OUT VARCHAR2,
       p_township IN OUT VARCHAR2,
       p_tship_dir IN OUT VARCHAR2,
       p_range IN OUT VARCHAR2,
       p_range_dir IN OUT VARCHAR2,
       p_section IN OUT VARCHAR2,
       p_ca_mtrs_acres OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 67;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;


      IF p_base_ln_mer IS NOT NULL AND p_base_ln_mer <> '?' AND
         p_township IS NOT NULL AND p_township <> '?' AND
         p_tship_dir IS NOT NULL AND p_tship_dir <> '?' AND
         p_range IS NOT NULL AND p_range <> '?' AND
         p_range_dir IS NOT NULL AND p_range_dir <> '?' AND
         p_section IS NOT NULL AND p_section <> '?'
      THEN
          SELECT  acres
          INTO    p_ca_mtrs_acres
          FROM    mtrs_acres
          WHERE   mtrs = p_base_ln_mer||p_township||p_tship_dir||
                  p_range||p_range_dir||p_section;
      ELSE
         p_ca_mtrs_acres := NULL;
      END IF;

      No_error(p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         p_error_type := 'invalid';
         p_replace_type := 'null';
         p_base_ln_mer := '?';
         p_township := '?';
         p_tship_dir := '?';
         p_range := '?';
         p_range_dir := '?';
         p_section := '?';
         p_ca_mtrs_acres := NULL;
      WHEN OTHERS THEN
         Other_error(NULL, p_ca_mtrs_acres, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Mtrs;
```

```
/* Error code 48: check that comtrs exist in the county

   Must be called after checks for record_id, county_cd,
   base_ln_mer, township, tship_dir, range, range_dir, and section.
 */
   PROCEDURE Comtrs
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_county_cd IN VARCHAR2,
       p_base_ln_mer IN OUT VARCHAR2,
       p_township IN OUT VARCHAR2,
       p_tship_dir IN OUT VARCHAR2,
       p_range IN OUT VARCHAR2,
       p_range_dir IN OUT VARCHAR2,
       p_section IN OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_dummy  NUMBER;
   BEGIN
      p_error_code := 48;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;


      IF p_county_cd IS NOT NULL AND p_county_cd <> '?' AND
         p_base_ln_mer IS NOT NULL AND p_base_ln_mer <> '?' AND
         p_township IS NOT NULL AND p_township <> '?' AND
         p_tship_dir IS NOT NULL AND p_tship_dir <> '?' AND
         p_range IS NOT NULL AND p_range <> '?' AND
         p_range_dir IS NOT NULL AND p_range_dir <> '?' AND
         p_section IS NOT NULL AND p_section <> '?'
      THEN
          SELECT  acres
          INTO    v_dummy
          FROM    comtrs_acres
          WHERE   comtrs = p_county_cd||p_base_ln_mer||p_township||p_tship_dir||
                  p_range||p_range_dir||p_section;
      END IF;

      No_error(p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         p_error_type := 'invalid';
         p_replace_type := 'null';
         p_base_ln_mer := '?';
         p_township := '?';
         p_tship_dir := '?';
         p_range := '?';
         p_range_dir := '?';
         p_section := '?';
      WHEN OTHERS THEN
         Other_error(NULL, v_dummy, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Comtrs;
```

```
/* Error code 12: check aer_gnd_ind

   Must be called after checks for record_id.
 */
   PROCEDURE Aer_gnd_ind
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_aer_gnd_ind OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 12;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_aer_gnd_ind := p_value;

         IF p_aer_gnd_ind IS NULL OR p_aer_gnd_ind NOT IN ('A','G','O') THEN
            Invalid_char_error(p_value, p_aer_gnd_ind, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
            Not_allowed(p_value, p_aer_gnd_ind, p_error_code,
                        p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         p_aer_gnd_ind := p_value;

         IF p_aer_gnd_ind = '0' OR p_aer_gnd_ind IS NULL THEN
            Possible_invalid_char_error(p_value, p_aer_gnd_ind, p_error_type, p_replace_type);
         ELSIF p_aer_gnd_ind NOT IN ('A','G','O') THEN
            Invalid_char_error(p_value, p_aer_gnd_ind, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_aer_gnd_ind, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_aer_gnd_ind, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Aer_gnd_ind;
```

```
/* Error code 68: check applic_time: new field in 1999

   Must be called after checks for record_id.
 */
   PROCEDURE Applic_time
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_applic_time OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 68;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'optional';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'optional' OR p_require_type = 'unknown' THEN
         p_applic_time := TRANSLATE(p_value, ' ', '0');

         IF TO_NUMBER(p_applic_time) = 0 THEN
            p_applic_time := NULL;
         END IF;

         IF LENGTH(p_applic_time) < 4 OR
            TO_NUMBER(SUBSTR(p_applic_time, 1, 2)) NOT BETWEEN 0 AND 24 OR
            TO_NUMBER(SUBSTR(p_applic_time, 3, 2)) NOT BETWEEN 0 AND 60
         THEN
            Invalid_char_error(p_value, p_applic_time, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
         Not_allowed(p_value, p_applic_time, p_error_code,
                     p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_applic_time, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_applic_time, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Applic_time;
```

```
/* Error code 43: check grower_id.
   Grower_id is required for prod ag uses and, starting after 1999,
   required for non ag if license_no is missing; not required otherwise.

   Must be called after checks for record_id, county_cd.

   For 1999, ignore requirement that either grower_id or license_no be given for
   non ag records
 */
   PROCEDURE Grower_id
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_county_cd IN VARCHAR2,
       p_license_no IN VARCHAR2,
       p_applic_year IN NUMBER,
       p_value IN VARCHAR2,
       p_grower_id OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 43;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         IF v_pur_year <= 1999 OR p_county_cd = '38' THEN
            p_require_type := 'optional';
         ELSE
            IF p_license_no = '             ' OR
               p_license_no = '0000000000000' OR
               p_license_no IS NULL
            THEN
               p_require_type := 'required';
            ELSE
               p_require_type := 'optional';
            END IF;
         END IF;
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_value IS NULL OR
         SUBSTR(p_value, 7) = '     ' OR
         ((p_county_cd <> '39' OR p_record_id NOT IN ('1','4','A','B')) AND
           SUBSTR(p_value, 7) = '00000')
      THEN
         p_grower_id := NULL;
      ELSE
         p_grower_id := p_value;
      END IF;

      IF p_require_type = 'required' THEN
         IF p_grower_id IS NULL THEN
            Invalid_char_error(p_value, p_grower_id, p_error_type, p_replace_type);
         ELSE
            Grower_id_error(p_grower_id, p_county_cd, p_applic_year,
                            p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'optional' THEN
         /* We can not use "between 1 and 58" in this condition because if the value
            in SUBSTR(p_grower_id, 5, 2) is not a number, a VALUE_ERROR will be
            raised which takes the procedure to the exception clause.
          */
         IF (SUBSTR(p_grower_id, 1, 2) = p_county_cd AND
             SUBSTR(p_grower_id, 3, 2) BETWEEN
                SUBSTR(p_applic_year, 3, 2) - 3 AND SUBSTR(p_applic_year, 3, 2) + 3 AND
             SUBSTR(p_grower_id, 5, 2) IN
             ('01','02','03','04','05','06','07','08','09','10',
              '11','12','13','14','15','16','17','18','19','20',
              '21','22','23','24','25','26','27','28','29','30',
              '31','32','33','34','35','36','37','38','39','40',
              '41','42','43','44','45','46','47','48','49','50',
              '51','52','53','54','55','56','57','58')) OR
            p_grower_id IS NULL OR
            p_county_cd = '38'
         THEN
            No_error(p_error_code, p_error_type, p_replace_type);
         ELSE
            /* To get here, license_no and p_grower_id must both
               be present and grower_id must be invalid.
               If grower_id is same as license_no (removing all spaces),
               then set it to NULL;
               else report error.
             */
            IF RTRIM(LTRIM(p_license_no)) = RTRIM(LTRIM(p_grower_id)) OR
               RTRIM(LTRIM(p_license_no)) = RTRIM(LTRIM(SUBSTR(p_grower_id, 5))) OR
               RTRIM(LTRIM(p_license_no)) = RTRIM(LTRIM(SUBSTR(p_grower_id, 7)))
            THEN
               Not_allowed(p_value, p_grower_id, p_error_code,
                           p_error_type, p_replace_type);
            ELSE
               Invalid_char_error(p_value, p_grower_id, p_error_type, p_replace_type);
            END IF;
         END IF;
      ELSE /* p_require_type = unknown */
         IF p_grower_id IS NULL THEN
            Possible_invalid_char_error(p_value, p_grower_id, p_error_type, p_replace_type);
         ELSE
            Grower_id_error(p_grower_id, p_county_cd, p_applic_year,
                            p_error_code,  p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_grower_id, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_grower_id, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Grower_id;


   /* Called to check for errors with grower_id
    */
   PROCEDURE Grower_id_error
      (p_grower_id IN OUT VARCHAR2, p_county_cd IN VARCHAR2,
       p_applic_year IN NUMBER, p_error_code OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
   IS
      v_home_county_cd  NUMBER(2);
   BEGIN
      p_error_code := NULL;
      p_error_type := 'N';
      p_replace_type := 'same';

      IF SUBSTR(p_grower_id,1,2) <> p_county_cd THEN
         p_grower_id := p_county_cd||SUBSTR(p_grower_id, 3);
         p_error_code := 43;
         p_error_type := 'invalid';
         p_replace_type := 'correct';
      END IF;

      IF SUBSTR(p_grower_id, 3, 2) NOT BETWEEN
         SUBSTR(p_applic_year, 3, 2) - 3 AND SUBSTR(p_applic_year, 3, 2) + 3
      THEN
         p_grower_id := p_county_cd||SUBSTR(p_applic_year, 3, 2)||SUBSTR(p_grower_id, 5);
         p_error_code := 43;
```

```
         p_error_type := 'invalid';
         p_replace_type := 'estimate';
      END IF;

      v_home_county_cd := SUBSTR(p_grower_id,5,2);
      IF v_home_county_cd NOT BETWEEN 1 AND 58 THEN
         RAISE VALUE_ERROR;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         /* This should be raised only for checks on home county_cd
          */
         p_grower_id := SUBSTR(p_grower_id, 1, 4)||p_county_cd||
                        SUBSTR(p_grower_id, 7);
         p_error_code := 43;
         p_error_type := 'invalid';
         p_replace_type := 'estimate';

   END Grower_id_error;


/* Error code 69: check license_no: new field in 1999

   Must be called after checks for record_id, p_grower_id.

   For 1999, ignore requirement that either grower_id or license_no be given for
   non ag records
 */
   PROCEDURE License_no
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_grower_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_license_no OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 69;
      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IN ('2','C') THEN
         IF v_pur_year <= 1999 THEN
            p_require_type := 'optional';
         ELSE
            IF p_grower_id IS NULL THEN
               p_require_type := 'required';
            ELSE
               p_require_type := 'optional';
            END IF;
         END IF;
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_value IS NULL OR
         p_value = '             ' OR
         p_value = '0000000000000' /* Is this an error? ask counties */
      THEN
         p_license_no := NULL;
      ELSE
         p_license_no := p_value;
      END IF;

      IF p_require_type = 'required' THEN
         IF p_license_no IS NULL THEN
```

```
            Invalid_char_error(p_value, p_license_no, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'optional' THEN
         IF p_license_no IS NULL THEN
            No_error(p_error_code, p_error_type, p_replace_type);
         ELSE
            /* To get here, license_no and p_grower_id must both
               be present and grower_id must be valid.
               If license_no is same as grower_id,
               then set it to NULL;
               else leave it as is.
             */
            IF RTRIM(LTRIM(p_license_no)) = RTRIM(LTRIM(p_grower_id)) OR
               RTRIM(LTRIM(p_license_no)) = RTRIM(LTRIM(SUBSTR(p_grower_id, 5))) OR
               RTRIM(LTRIM(p_license_no)) = RTRIM(LTRIM(SUBSTR(p_grower_id, 7)))
            THEN
               Not_allowed(p_value, p_license_no, p_error_code,
                           p_error_type, p_replace_type);
            ELSE
               No_error(p_error_code, p_error_type, p_replace_type);
            END IF;
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
         Not_allowed(p_value, p_license_no, p_error_code,
                          p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         IF p_license_no IS NULL THEN
            Possible_invalid_char_error(p_value, p_license_no, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_license_no, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_license_no, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END License_no;

/* Error code 31: check amt_prd_used
 */
   PROCEDURE Amt_prd_used
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_amt_prd_used OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 31;
      p_require_type := 'required';
      p_amt_prd_used := TO_NUMBER(p_value)/10000;

      IF p_amt_prd_used IS NULL OR p_amt_prd_used = 0 THEN
         Invalid_error(p_value, p_amt_prd_used, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;
   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_amt_prd_used, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_amt_prd_used, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Amt_prd_used;
```

```
/* Error code 32: check unit_of_meas
 */
   PROCEDURE Unit_of_meas
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_unit_of_meas OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 32;
      p_require_type := 'required';
      p_unit_of_meas := p_value;

      IF p_unit_of_meas IS NULL OR
         p_unit_of_meas NOT IN ('LB','OZ','GA','QT','PT','KG','GR','LI','ML')
      THEN
         Invalid_char_error(p_value, p_unit_of_meas, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_unit_of_meas, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_unit_of_meas, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Unit_of_meas;


/* Error code 24: check mfg_firmno
 */
   PROCEDURE Mfg_firmno
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_mfg_firmno OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 24;
      p_require_type := 'required';
      p_mfg_firmno := TO_NUMBER(p_value);

      IF p_mfg_firmno IS NULL OR p_mfg_firmno = 0 THEN
         Invalid_error(p_value, p_mfg_firmno, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_mfg_firmno, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_mfg_firmno, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Mfg_firmno;
```

```
/* Error code 25: check label_seq_no
 */
   PROCEDURE Label_seq_no
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_label_seq_no OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 25;
      p_require_type := 'required';
      p_label_seq_no := TO_NUMBER(p_value);

      IF p_label_seq_no IS NULL OR p_label_seq_no = 0 THEN
         Invalid_error(p_value, p_label_seq_no, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_label_seq_no, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_label_seq_no, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Label_seq_no;

/* check revision_no  (previously Error code 26)
   This is optional and no errors are reported because
   counties and growers often do not know the revision_no or reg_firmno.
   Default value of 'AA' is often used instead.
 */
   PROCEDURE Revision_no
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_revision_no OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := NULL;
      p_require_type := 'optional';
      p_revision_no := p_value;

      IF p_revision_no NOT BETWEEN 'AA' AND 'ZZ' THEN
         p_revision_no := NULL;
      END IF;

      No_error(p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(p_value, p_revision_no, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Revision_no;

/* check reg_firmno (previously Error code 27)
   This is optional and no errors are reported because
   counties and growers often do not know the reg_firmno.
 */
   PROCEDURE Reg_firmno
      (p_use_no IN NUMBER,
       p_value IN VARCHAR2,
       p_reg_firmno OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := NULL;
      p_require_type := 'required';
      p_reg_firmno := TO_NUMBER(p_value);

      /* In this case 0 means same as null, so make null.
         Note that p_reg_firmno will be assiged 0 if p_value has a
         value of one or more spaces (even though to_number(' ') will
         give an invalid number error)'
       */
      IF p_reg_firmno = 0 THEN
         p_reg_firmno := NULL;
      END IF;

      No_error(p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN VALUE_ERROR THEN
         p_reg_firmno := NULL;
         No_error(p_error_code, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_reg_firmno, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Reg_firmno;

/* Error code 37: check prodno.  This looks for a product in the
   label database with the reported reg. no.

   The error_type and replace_type are set in the
   procedure Get_prodno.FInd.

   Error_type = 'N' (replace_type = 'same') means found prodno that
      matched the reported 4-part registration number.

   Error_type = 'possible', replace_type = 'estimate' means did not
      find prodno that matched the reported 4-part registration number,
      but did find only one prodno that matched 3-part registration number;
      this prodno was returned in variable p_prodno and its
      revision_no was returned in variable p_new_revision_no.

   Error_type = 'possible', replace_type = 'null' means did not
      find prodno that matched the reported 4-part registration number,
      but did find more than one prodno that matched 3-part registration
      number. No prodno was chosen (prodno set to NULL); a prodno will
      be chosen in Procedure Prod_site.

   Error_type = 'invalid', replace_type = 'null' means did not
      find prodno that matched the reported 4-part registration number,
      or the 3-part registration number. Prodno set to -1; a prodno will
      be chosen in Procedure Prod_site if there exists any products
      matching the 2-part registration number.

   Must be called after checks for mfg_firmno, label_seq_no, revision_no,
   reg_firmno.
 */
   PROCEDURE Prodno
      (p_use_no IN NUMBER,
       p_mfg_firmno IN NUMBER,
       p_label_seq_no IN NUMBER,
       p_revision_no IN VARCHAR2,
       p_reg_firmno IN NUMBER,
       p_prodno OUT NUMBER,
       p_new_revision_no OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 37;
      p_require_type := 'required';

      Get_prodno.Find
         (p_mfg_firmno, p_label_seq_no, p_revision_no, p_reg_firmno,
          p_prodno, p_new_revision_no,
          p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(p_prodno, p_prodno, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Prodno;



 /* Error code 39: check if site is on label for prodno.
    If prodno is NULL (which happens if full registration number was
    not found in product table and more than one product exists
    with the reported 3-part registration number (mfg_firmno,
    label_seq_no, and reg_firmno)), then
    this procedure will choose one and set prodno equal to that product.
    site_code is not changed.

    Because of the somewhat intricate logic, the error_type and
    replace_type are set in the procedure Get_prodno.Check_site.
    In this case, replace_type refers to prodno, not site_code,
    since only prodno may be changed, never site_code.

   Error_type = 'N', replace_type = 'same' means
      site_code was on the label for that prodno.

   Error_type = 'possible', replace_type = 'same' means
      site_code was not on the label for that prodno, but
      site_code was found on the label for another product which
      matched the 2-part registration number.  The other product
      found is given in the comments field.

   Error_type = 'invalid', replace_type = 'same'
      site_code was not on the label for that prodno, and
      it was not on any other product matching the 2-part
      registration number.


   Error_type = 'possible', replace_type = 'estimate' means
      exact 4-part registration was not found, but
      prodno matching the 3-part number was found.
      If the site_code was not on the label for that prodno,
      but was on the label for another product which
      matched the 2-part registration number, this other product
      is given in the comments field.  If site_code was on
      the label, comments field will not mention other product.
      If site_code was not on any of these products,
      the program reports the next situation.

   Error_type = 'invalid', replace_type = 'estimate' means
      exact 4-part registration was not found, but
      prodno matching the 3-part number (or 2-part number) was found and
      site_code was not on the label for that product nor was
      site_code found on the label for any other product which
      matched the 2-part registration number.


   Error_type = 'invalid', replace_type = 'null' means
      exact 4-part registration was not found, and
      no other product matching the 2-part number was found.

    Must be called after checks for prodno, unit_of_meas,
    site_code, applic_dt.
 */
```

```
   PROCEDURE Prod_site
      (p_use_no IN NUMBER,
       p_mfg_firmno IN NUMBER,
       p_label_seq_no IN NUMBER,
       p_revision_no IN VARCHAR2,
       p_reg_firmno IN NUMBER,
       p_unit_of_meas IN VARCHAR2,
       p_site_code IN NUMBER,
       p_applic_dt IN DATE,
       p_prodno IN OUT NUMBER,
       p_new_revision_no OUT VARCHAR2,
       p_new_reg_firmno OUT NUMBER,
       p_other_product OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 39;
      p_require_type := 'required';

      Get_prodno.Check_site
         (p_mfg_firmno, p_label_seq_no, p_revision_no, p_reg_firmno,
          p_unit_of_meas, p_site_code, p_applic_dt, p_prodno,
          p_new_revision_no, p_new_reg_firmno, p_other_product,
          p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(NULL, p_prodno, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Prod_site;


/* Error code 52: Get spec_gravity from label database, and
   report error if spec_gravity is not found.  This indicates
   an error in the label database, not in the county PUR file.

   Must be called after checks for prodno, unit_of_meas.
 */
   PROCEDURE Spec_gravity
      (p_use_no IN NUMBER,
       p_prodno IN NUMBER,
       p_unit_of_meas IN VARCHAR2,
       p_spec_gravity OUT NUMBER,
       p_formula_cd OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 52;
      p_require_type := 'required';

      IF p_prodno IS NOT NULL AND p_prodno <> -1 THEN
         SELECT   spec_gravity, formula_cd
         INTO     p_spec_gravity, p_formula_cd
         FROM     product
         WHERE    prodno = p_prodno;

         IF NOT Get_prodno.HasSpecificGravity
            (p_spec_gravity, p_formula_cd, p_unit_of_meas)
         THEN
            Invalid_error(NULL, p_spec_gravity, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN NO_DATA_FOUND OR TOO_MANY_ROWS OR VALUE_ERROR THEN
         Invalid_error(NULL, p_spec_gravity, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(NULL, p_spec_gravity, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Spec_gravity;

/* Error code 38: second check of unit_of_meas--invalid measure for the formualation
   of this product.  This is a probable error because error could be
   the formulation listed in label database.

   Must be called after checks for unit_of_meas and
   spec_gravity (which gets formula_cd).
 */
   PROCEDURE Unit_of_meas2
      (p_use_no IN NUMBER,
       p_unit_of_meas IN VARCHAR2,
       p_formula_cd IN VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_dummy  VARCHAR2(1);
   BEGIN
      p_error_code := 38;
      p_require_type := 'required';

      IF (p_unit_of_meas IN ('GA','QT','PT','LI','ML') AND
          p_formula_cd IN ('A0','E0','F0','J0','K0','N0','P0','R0','U0')) OR
         (p_unit_of_meas IN ('GR','KG') AND
          p_formula_cd IN ('B0','C0','G0','H0','I0','M0','O0','Q0','S0','T0'))
      THEN
         Probable_error(p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Probable_error(p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(NULL, v_dummy, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Unit_of_meas2;

/* Calculate lbs_prd_used.  Does not generate any error messages.
   If unit_of_meas is wrong, this will generate wrong lbs_prd_used.

   Must be called after checks for proedno, amt_prd_used, unit_of_meas, and
   spec_gravity (which gets formula_cd).
 */
   PROCEDURE Lbs_prd_used
      (p_use_no IN NUMBER,
       p_prodno IN NUMBER,
       p_amt_prd_used IN NUMBER,
       p_unit_of_meas IN VARCHAR2,
       p_formula_cd IN VARCHAR2,
       p_spec_gravity IN NUMBER,
       p_lbs_prd_used OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_lbs_prd_used := NULL;
      IF p_amt_prd_used IS NOT NULL AND
         p_prodno IS NOT NULL AND p_prodno <> -1
      THEN
         IF p_unit_of_meas = 'LB' THEN
            p_lbs_prd_used := p_amt_prd_used;
         ELSIF p_unit_of_meas = 'OZ'  AND
            p_formula_cd in ('A0','E0','F0','J0','K0','L0','N0','P0','R0','U0') THEN
            p_lbs_prd_used := p_amt_prd_used/16;
         ELSIF p_unit_of_meas = 'GR' THEN
            p_lbs_prd_used := p_amt_prd_used/453.59;
         ELSIF p_unit_of_meas = 'KG' THEN
            p_lbs_prd_used := p_amt_prd_used*2.2046;
         ELSIF p_spec_gravity IS NOT NULL THEN
            IF p_unit_of_meas = 'OZ'  AND
               p_formula_cd in ('B0','C0','D0','G0','H0','I0','M0','O0','Q0','S0','T0') THEN
               p_lbs_prd_used := p_amt_prd_used*p_spec_gravity*8.33/128;
            ELSIF p_unit_of_meas = 'GA' THEN
               p_lbs_prd_used := p_amt_prd_used*p_spec_gravity*8.33;
            ELSIF p_unit_of_meas = 'QT' THEN
               p_lbs_prd_used := p_amt_prd_used*p_spec_gravity*8.33/4;
            ELSIF p_unit_of_meas = 'PT' THEN
               p_lbs_prd_used := p_amt_prd_used*p_spec_gravity*8.33/8;
            ELSIF p_unit_of_meas = 'LI' THEN
               p_lbs_prd_used := p_amt_prd_used*p_spec_gravity*8.33/3.785;
            ELSIF p_unit_of_meas = 'ML' THEN
               p_lbs_prd_used := p_amt_prd_used*p_spec_gravity*8.33/3785;
            END IF;
         END IF;
      END IF;

      IF p_lbs_prd_used < 0.0001 AND p_amt_prd_used > 0 THEN
         p_lbs_prd_used := 0.0001;
      END IF;

      No_error(p_error_code, p_error_type, p_replace_type);

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(NULL, p_lbs_prd_used, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Lbs_prd_used;

/* Error code 18: check acre_treated

   Must be called after checks for record_id, site_code.
 */
   PROCEDURE Acre_treated
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_site_code IN NUMBER,
       p_value IN VARCHAR2,
       p_acre_treated OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 18;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         IF p_site_code > 100 THEN
            p_require_type := 'required';
         ELSIF p_site_code BETWEEN 1 AND 100 THEN
            p_require_type := 'optional';
         ELSE
            p_require_type := 'unknown';
         END IF;
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_acre_treated := TO_NUMBER(p_value)/100;
         IF p_acre_treated IS NULL OR p_acre_treated <= 0 THEN
            Invalid_error(p_value, p_acre_treated, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'optional' THEN
         IF p_value = '.' OR p_value = ' ' THEN
            p_acre_treated := NULL;
         ELSE
            p_acre_treated := TO_NUMBER(p_value)/100;
         END IF;

         IF p_acre_treated = 0 THEN
            p_acre_treated := NULL;
         END IF;

         IF p_acre_treated < 0 THEN
            Invalid_error(p_value, p_acre_treated, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSE /* p_require_type = unknown */
         IF p_value = '.'  OR p_value = ' ' THEN
            p_acre_treated := NULL;
         ELSE
            p_acre_treated := TO_NUMBER(p_value)/100;
         END IF;

         IF p_acre_treated IS NULL OR p_acre_treated = 0 THEN
            Possible_invalid_error(p_value, p_acre_treated, p_error_type, p_replace_type);
         ELSIF p_acre_treated < 0 THEN
            Invalid_error(p_value, p_acre_treated, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_acre_treated, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_acre_treated, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Acre_treated;
```

```
/* Error code 19: check unit_treated .
   [Ask counties what kinds of prod ag treatments use C, K, or U:
    U can be things like trees, plants, greenhouses]

   Must be called after checks for record_id, site_code.
 */
   PROCEDURE Unit_treated
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_site_code IN NUMBER,
       p_value IN VARCHAR2,
       p_unit_treated OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 19;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         IF p_site_code > 100 THEN
            p_require_type := 'required';
         ELSIF p_site_code BETWEEN 1 AND 100 THEN
            p_require_type := 'optional';
         ELSE
            p_require_type := 'unknown';
         END IF;
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_unit_treated := p_value;

         IF p_unit_treated IS NULL OR
            (p_record_id IN ('1','4','A','B') AND
             p_unit_treated NOT IN ('A','S','C','K','U')) OR
            (p_record_id IN ('2','C') AND
             p_unit_treated NOT IN ('A','S','C','K','P','T','U'))
         THEN
            Invalid_char_error(p_value, p_unit_treated, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'optional' THEN
         IF p_value = '.' OR p_value = ' ' OR p_value = '0' THEN
            p_unit_treated := NULL;
         ELSE
            p_unit_treated := p_value;
         END IF;

         IF p_unit_treated NOT IN ('A','S','C','K','P','T','U') THEN
            Invalid_char_error(p_value, p_unit_treated, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSE /* p_require_type = unknown */
         IF p_value = '.' OR p_value = ' ' OR p_value = '0' THEN
            p_unit_treated := NULL;
         ELSE
            p_unit_treated := p_value;
         END IF;

         IF p_unit_treated IS NULL THEN
            Possible_invalid_char_error(p_value, p_unit_treated, p_error_type, p_replace_type);
         ELSIF p_unit_treated NOT IN ('A','S','C','K','U') THEN
```

```
            Invalid_char_error(p_value, p_unit_treated, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;
   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_unit_treated, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_unit_treated, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Unit_treated;


/* Error code 18, 19: check acre_treated, unit_treated both present
   when optional

   Must be called after checks for record_id, site_code, acre_treated, unit_treated.
 */
   PROCEDURE Acre_unit_treated
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_site_code IN NUMBER,
       p_acre_treated IN NUMBER,
       p_unit_treated IN VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_dummy VARCHAR2(1) := NULL;
   BEGIN
      IF p_record_id IN ('2','C') AND
         p_site_code BETWEEN 1 AND 100
      THEN
          p_require_type := 'optional';
      ELSIF p_record_id IS NULL OR p_record_id = '?' OR
         p_site_code IS NULL OR p_site_code = -1
      THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'required';
      END IF;

      IF p_require_type = 'optional' OR p_require_type = 'unknown' THEN
         IF p_acre_treated IS NULL AND
            p_unit_treated IS NOT NULL AND p_unit_treated <> '?'
         THEN
            p_error_code := 18;
            Invalid_char_error(NULL, v_dummy, p_error_type, p_replace_type);
         ELSIF p_acre_treated IS NOT NULL AND
            (p_unit_treated IS NULL OR p_unit_treated = '?')
         THEN
            p_error_code := 19;
            Invalid_char_error(NULL, v_dummy, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(NULL, v_dummy, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Acre_unit_treated;
```

```
/* Error code 23: check acre_treated > area of its section.  If it is,
   mark as error but estimate acre_treated = area of section.

   Includes a special exception to allow large fields for
   certain growers in year=2000.

   Must be called after checks for site_code, acre_treated, unit_treated,
   and mtrs() (which get value for p_ca_mtrs_acres).
 */
   PROCEDURE Acre_treated_section
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_site_code IN NUMBER,
       p_ca_mtrs_acres IN NUMBER,
       p_acre_treated IN OUT NUMBER,
       p_unit_treated IN VARCHAR2,
       p_grower_id IN VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 23;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         IF p_site_code > 100 THEN
            p_require_type := 'required';
         ELSIF p_site_code BETWEEN 1 AND 100 THEN
            p_require_type := 'optional';
         ELSE
            p_require_type := 'unknown';
         END IF;
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_acre_treated IS NOT NULL AND
         p_ca_mtrs_acres IS NOT NULL AND
         p_unit_treated = 'A' AND
         p_site_code NOT IN (30000, 28035, 28045, 66000, 67000) AND
         p_acre_treated > 1.1*p_ca_mtrs_acres
      THEN
         IF v_pur_year =2000 AND
            p_grower_id IN ('270027K003A', '270027S333A', '270027S324A', '270027K022A')
         THEN
            No_error(p_error_code, p_error_type, p_replace_type);
         ELSE
            Probable_estimated_error(p_ca_mtrs_acres, p_acre_treated,
                          p_error_type, p_replace_type);
         END IF;
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(p_acre_treated, p_acre_treated, p_error_code,
                  p_error_type, p_replace_type, p_use_no);
   END Acre_treated_section;
```

```
/* Error code 22: for non ag check if acre_treated > 700 (unit = A) or
   acre_treated >= 999999 (unit = S).
   For estimate, let unit_treated = U.

   Must be called after checks for site_code, acre_treated, unit_treated.
 */
   PROCEDURE Acre_treated_nonag
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_site_code IN NUMBER,
       p_acre_treated IN NUMBER,
       p_unit_treated IN OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 22;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         IF p_site_code > 100 THEN
            p_require_type := 'required';
         ELSIF p_site_code BETWEEN 1 AND 100 THEN
            p_require_type := 'optional';
         ELSE
            p_require_type := 'unknown';
         END IF;
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_record_id IN ('2','C') AND
         p_acre_treated IS NOT NULL AND
         p_site_code NOT IN (30000, 28035, 28045, 66000, 67000)
      THEN
         /* Maximum value for acre_treated = 999999.99 */
         IF (p_unit_treated = 'A' AND p_acre_treated > 700) OR
            (p_unit_treated = 'S' AND p_acre_treated >= 999999)
         THEN
            /* Estimate unit as miscellaneous */
            Probable_estimated_error('U', p_unit_treated,
                         p_error_type, p_replace_type);
         END IF;
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(p_unit_treated, p_unit_treated, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Acre_treated_nonag;
```

```
/* Error code 44: check acre_planted

   Must be called after checks for record_id.
 */
   PROCEDURE Acre_planted
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_acre_planted OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 44;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_acre_planted := TO_NUMBER(p_value)/100;
         IF p_acre_planted IS NULL OR p_acre_planted <= 0 THEN
            Invalid_error(p_value, p_acre_planted, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
         Not_allowed(p_value, p_acre_planted, p_error_code,
                     p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         IF p_value = '.'  OR p_value = ' ' THEN
            p_acre_planted := NULL;
         ELSE
            p_acre_planted := TO_NUMBER(p_value)/100;
         END IF;

         IF p_acre_planted IS NULL OR p_acre_planted = 0 THEN
            Possible_invalid_error(p_value, p_acre_planted, p_error_type, p_replace_type);
         ELSIF p_acre_planted < 0 THEN
            Invalid_error(p_value, p_acre_planted, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_acre_planted, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_acre_planted, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Acre_planted;
```

```
/* Error code 45: check unit_planted
    [Ask counties what kinds of fields use C, K, or U]

   Must be called after checks for record_id.
 */
   PROCEDURE Unit_planted
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_unit_planted OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 45;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'not_allowed';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_unit_planted := p_value;

         IF p_unit_planted IS NULL OR
            p_unit_planted NOT IN ('A','S','C','K','U')
         THEN
            Invalid_char_error(p_value, p_unit_planted, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSIF p_require_type = 'not_allowed' THEN
         Not_allowed(p_value, p_unit_planted, p_error_code,
                     p_error_type, p_replace_type);
      ELSE /* p_require_type = unknown */
         IF p_value = '.' OR p_value = ' ' OR p_value = '0' THEN
            p_unit_planted := NULL;
         ELSE
            p_unit_planted := p_value;
         END IF;

         IF p_unit_planted IS NULL THEN
            Possible_invalid_char_error(p_value, p_unit_planted, p_error_type, p_replace_type);
         ELSIF p_unit_planted NOT IN ('A','S','C','K','U') THEN
            Invalid_char_error(p_value, p_unit_planted, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;
   EXCEPTION
      WHEN VALUE_ERROR THEN
         Invalid_char_error(p_value, p_unit_planted, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_unit_planted, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Unit_planted;
```

```
/* Error code 60: check acre_planted > area of its section.  If it is,
   mark as error but estimate acre_planted = area of section.

   Includes a special exception to allow large fields for
   certain growers in year=2000.

   Must be called after checks for site_code, acre_planted, unit_planted,
   and mtrs() (which get value for p_ca_mtrs_acres).
 */
   PROCEDURE Acre_planted_section
      (p_use_no IN NUMBER,
       p_site_code IN NUMBER,
       p_ca_mtrs_acres IN NUMBER,
       p_acre_planted IN OUT NUMBER,
       p_unit_planted IN VARCHAR2,
       p_grower_id IN VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 60;

      IF p_acre_planted IS NOT NULL AND
         p_ca_mtrs_acres IS NOT NULL AND
         p_unit_planted = 'A' AND
         p_site_code NOT IN (30000, 28035, 28045, 66000, 67000) AND
         p_acre_planted > 1.1*p_ca_mtrs_acres
      THEN
         IF v_pur_year =2000 AND
            p_grower_id IN ('270027K003A', '270027S333A', '270027S324A', '270027K022A')
         THEN
            No_error(p_error_code, p_error_type, p_replace_type);
         ELSE
            Probable_estimated_error(p_ca_mtrs_acres, p_acre_planted,
                            p_error_type, p_replace_type);
         END IF;
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(p_acre_planted, p_acre_planted, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Acre_planted_section;


/* Error code 47: check acre_treated > acre_planted.  If it is,
   mark as probable error but estimate acre_treated = acre_planted,
   unless unit_treated = A (or K) and unit_planted = S (or C) and
   acre_treated < acre_planted, then assume error is with
   unit_treated and estimate unit_treated as S (or C).

   Must be called after checks for acre_treated, unit_treated,
   acre_planted, unit_planted.
 */
   PROCEDURE Acre_treated_planted
      (p_use_no IN NUMBER,
       p_unit_planted IN VARCHAR2,
       p_unit_treated IN OUT VARCHAR2,
       p_acre_planted IN NUMBER,
       p_acre_treated IN OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_acres_treated         NUMBER(10, 2) := NULL;
      v_acres_planted         NUMBER(10, 2) := NULL;
      v_cubic_feet_treated    NUMBER(15, 2) := NULL;
      v_cubic_feet_planted    NUMBER(15, 2) := NULL;
      v_misc_treated          NUMBER(10, 2) := NULL;
      v_misc_planted          NUMBER(10, 2) := NULL;
   BEGIN
      p_error_code := 47;

      IF p_unit_treated = 'A' THEN
         v_acres_treated := p_acre_treated;
      ELSIF p_unit_treated = 'S' THEN
         v_acres_treated := p_acre_treated/43560;
      ELSIF p_unit_treated = 'C' THEN
         v_cubic_feet_treated := p_acre_treated;
      ELSIF p_unit_treated = 'K' THEN
         v_cubic_feet_treated := p_acre_treated * 1000;
      ELSIF p_unit_treated = 'U' THEN
         v_misc_treated := p_acre_treated;
      END IF;

      IF p_unit_planted = 'A' THEN
         v_acres_planted := p_acre_planted;
      ELSIF p_unit_planted = 'S' THEN
         v_acres_planted := p_acre_planted/43560;
      ELSIF p_unit_planted = 'C' THEN
         v_cubic_feet_planted := p_acre_planted;
      ELSIF p_unit_planted = 'K' THEN
         v_cubic_feet_planted := p_acre_planted * 1000;
      ELSIF p_unit_planted = 'U' THEN
         v_misc_planted := p_acre_treated;
      END IF;

      IF v_acres_treated > v_acres_planted THEN
         IF p_unit_treated = 'A' AND
            p_unit_planted = 'S' AND
            p_acre_treated < p_acre_planted
         THEN
            Probable_estimated_error('S', p_unit_treated,
                            p_error_type, p_replace_type);
         ELSIF p_unit_treated = 'A' THEN
            Probable_estimated_error(v_acres_planted, p_acre_treated,
                            p_error_type, p_replace_type);
         ELSIF p_unit_treated = 'S' THEN
            Probable_estimated_error(v_acres_planted*43560, p_acre_treated,
                            p_error_type, p_replace_type);
         END IF;
      ELSIF v_cubic_feet_treated > v_cubic_feet_planted THEN
         IF p_unit_treated = 'K' AND
            p_unit_planted = 'C' AND
            p_acre_treated < p_acre_planted
         THEN
            Probable_estimated_error('C', p_unit_treated,
                            p_error_type, p_replace_type);
         ELSIF p_unit_treated = 'C' THEN
            Probable_estimated_error(v_cubic_feet_planted, p_acre_treated,
                            p_error_type, p_replace_type);
         ELSIF p_unit_treated = 'K' THEN
            Probable_estimated_error(v_cubic_feet_planted/1000, p_acre_treated,
                            p_error_type, p_replace_type);
         END IF;
      ELSIF v_misc_treated > v_misc_planted THEN
         Probable_estimated_error(v_misc_planted, p_acre_treated,
                         p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(p_acre_treated, p_acre_treated, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Acre_treated_planted;
```

```
/* Error code 61: check unit_treated and unit_planted are consistent.

   Previously I treated situation where unit_treated = 'U' and unit_planted = 'A'
   as a special case.  Apparently, this situation is ok
   for production ag (U usually means trees, plants, or greenhouses),
   so these are now accepted.
     [The previous procedure was:
      Here we deal separately with the situation where
      unit_treated = 'U' and unit_planted = 'A'.  I find this
      fairly frequently and in most cases other records for the
      same ag field have the same values for acre_planted and unit_planted,
      so I assume this part is correct.  Also, in most of these
      cases, the unit_treated is also 'A', so most likely,
      value of 'U' for unit_treated is wrong.  I will estimate
      the unit_treated with 'A', but only if acre_treated < acre_planted;
      otherwise I leave unit_treated as 'U' and report as a
      possible error. ]


   Must be called after checks for acre_treated, acre_planted,
   unit_treated, and unit_planted.
 */
   PROCEDURE Unit_treated_planted
      (p_use_no IN NUMBER,
       p_acre_treated IN NUMBER,
       p_acre_planted IN NUMBER,
       p_unit_treated IN OUT VARCHAR2,
       p_unit_planted IN OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 61;

      IF (p_unit_treated IN ('A', 'S') AND p_unit_planted IN ('C', 'K')) OR
         (p_unit_treated IN ('C', 'K') AND p_unit_planted IN ('A', 'S')) OR
         (p_unit_treated = 'U' AND p_unit_planted NOT IN ('U', 'A', 'S')) OR
         (p_unit_treated NOT IN ('U', 'A', 'S') AND p_unit_planted = 'U')
      THEN
         Invalid_char_error(p_unit_treated, p_unit_treated, p_error_type, p_replace_type);
         Invalid_char_error(p_unit_planted, p_unit_planted, p_error_type, p_replace_type);
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

      /* Previous version:
      IF (p_unit_treated IN ('A', 'S') AND p_unit_planted IN ('C', 'K')) OR
         (p_unit_treated IN ('C', 'K') AND p_unit_planted IN ('A', 'S')) OR
         (p_unit_treated = 'U' AND p_unit_planted <> 'U') OR
         (p_unit_treated <> 'U' AND p_unit_planted = 'U')
      THEN


         IF p_unit_treated = 'U' AND p_unit_planted = 'A' THEN

            IF p_acre_treated <= p_acre_planted  THEN
               Probable_estimated_error('A', p_unit_treated, p_error_type, p_replace_type);
            ELSE
               Possible_error(p_error_type, p_replace_type);
            END IF;

         ELSE
            Invalid_char_error(p_unit_treated, p_unit_treated, p_error_type, p_replace_type);
            Invalid_char_error(p_unit_planted, p_unit_planted, p_error_type, p_replace_type);
         END IF;
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;
      */

   EXCEPTION
      WHEN OTHERS THEN
         Other_error(p_unit_treated, p_unit_treated, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Unit_treated_planted;

/* Error code 30: check applic_cnt.  For production ag, applic_cnt should always = 1

   Must be called after checks for record_id.
 */
   PROCEDURE Applic_cnt
      (p_use_no IN NUMBER,
       p_record_id IN VARCHAR2,
       p_value IN VARCHAR2,
       p_applic_cnt OUT NUMBER,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
   BEGIN
      p_error_code := 30;

      IF p_record_id IN ('1','4','A','B') THEN
         p_require_type := 'required';
      ELSIF p_record_id IN ('2','C') THEN
         p_require_type := 'optional';
      ELSIF p_record_id IS NULL THEN
         p_require_type := 'unknown';
      ELSE
         p_require_type := 'unknown';
      END IF;

      IF p_require_type = 'required' THEN
         p_applic_cnt := 1;
         No_error(p_error_code, p_error_type, p_replace_type);
      ELSIF p_require_type = 'optional' THEN
         IF p_value = '.' OR p_value = ' ' THEN
            p_applic_cnt := NULL;
         ELSE
            p_applic_cnt := TO_NUMBER(p_value);
         END IF;

         IF p_applic_cnt = 0 THEN
            p_applic_cnt := NULL;
         END IF;

         IF p_applic_cnt < 0 THEN
            Invalid_error(p_value, p_applic_cnt, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSE /* p_require_type = unknown */
         IF p_value = '.'  OR p_value = ' ' THEN
            p_applic_cnt := NULL;
         ELSE
            p_applic_cnt := TO_NUMBER(p_value);
         END IF;

         IF p_applic_cnt IS NULL OR p_applic_cnt = 0 THEN
            Possible_invalid_error(p_value, p_applic_cnt, p_error_type, p_replace_type);
         ELSIF p_applic_cnt < 0 THEN
            Invalid_error(p_value, p_applic_cnt, p_error_type, p_replace_type);
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      END IF;

   EXCEPTION
```

```
      WHEN VALUE_ERROR THEN
         Invalid_error(p_value, p_applic_cnt, p_error_type, p_replace_type);
      WHEN OTHERS THEN
         Other_error(p_value, p_applic_cnt, p_error_code,
                     p_error_type, p_replace_type, p_use_no);
   END Applic_cnt;


/* Error code 75: check rate of use > outlier value 1 or 2.

   Previously, used error code 70 if higher than outlier criterion 1
   and error code 71 if higher than outlier criterion 2.
   Now, use error code 75 if higher than either criteria
   and record which outlier criteria were not met in p_criteria.

   The outlier rate could result from errors in
   unit_treated, acre_treated, or lbs_prd_used.
   If any of these can be estimated, replace the value.
   --should be able to estimate at least one of these, but check to be sure
      this can always be done.

   -- Before you change unit_treated, check that it does not cause
      acre_treated > acre_planted.

   -- If change lbs_prd_used, need to also change amt_prd_used.

   !!!!!To do:
   -- Error could be with unit_pf_meas.  How check for this? One idea is
      to look at other records with of this product by this grower.


   Must be called after checks for record_id, prodno, site_code, lbs_prd_used,
   acre_treated, unit_treated, acre_planted, unit_planted, amt_prd_used.
 */
   PROCEDURE Outliers
      (p_use_no IN NUMBER,
       p_record_id VARCHAR2,
       p_prodno IN NUMBER,
       p_site_code IN NUMBER,
       p_amt_prd_used IN OUT NUMBER,
       p_lbs_prd_used IN OUT NUMBER,
       p_acre_treated IN OUT NUMBER,
       p_unit_treated IN OUT VARCHAR2,
       p_acre_planted IN NUMBER,
       p_unit_planted IN VARCHAR2,
       p_estimated_field OUT VARCHAR2,
       p_criteria OUT VARCHAR2,
       p_error_code OUT BINARY_INTEGER,
       p_error_type OUT VARCHAR2,
       p_replace_type OUT VARCHAR2,
       p_require_type OUT VARCHAR2)
   IS
      v_ai_a_1000_200   NUMBER;
      v_prd_u_50m       NUMBER;
      v_median          NUMBER;
      v_rate            NUMBER;
      v_dummy           VARCHAR2(1);
      v_old_lbs_prd_used NUMBER;
   BEGIN
      IF p_record_id IS NOT NULL AND p_record_id <> '?' AND
         p_prodno IS NOT NULL AND p_prodno <> -1 AND
         p_site_code IS NOT NULL AND p_site_code <> -1 AND
         p_acre_treated > 0 AND
         p_unit_treated IS NOT NULL AND p_unit_treated <> '?' AND
         p_lbs_prd_used IS NOT NULL
      THEN
         p_error_code := 75;

         /* Normally use previous years usetypeXXstats table.
          */
         SELECT   ai_a_1000_200, prd_u_50m, median
         INTO     v_ai_a_1000_200, v_prd_u_50m, v_median
         FROM     usetype99stats
         WHERE    prodno = p_prodno AND
                  site_code = p_site_code AND
                  unit_treated = p_unit_treated AND
                  record_id_type =
                     TRANSLATE(p_record_id, 'C2G9DHAB14EF', 'NNNNNNAAAAAA');

         v_rate := p_lbs_prd_used/p_acre_treated;

         p_criteria := NULL;
         IF v_rate > v_ai_a_1000_200 THEN
            p_criteria := p_criteria||'criterion 1 ';
         END IF;
         IF v_rate > v_prd_u_50m THEN
            p_criteria := p_criteria||'criterion 2 ';
         END IF;

         /* More English-like statement of the following code:
         IF rate of use is unusally high THEN
            IF replacing unit_treated with 'A' makes rate reasonable THEN
               estimate unit_treated with 'A';
            ELSIF replacing acre_treated using median rate
                  gives acre_treated < acre_planted THEN
               estimate acre_treated;
            ELSE
               estimate lbs_prd_used using median rate and acre_treated;
            END IF;
         END IF;
         */
         IF p_criteria IS NOT NULL THEN
            IF Outlier.Wrong_unit
                  (p_record_id, p_site_code, p_prodno, v_rate,
                   p_acre_planted, p_unit_planted,
                   p_acre_treated, p_unit_treated)
            THEN
               Outlier.Estimate_unit
                  (p_estimated_field, p_error_type, p_replace_type);
            ELSIF Outlier.Wrong_acres
                  (p_record_id, p_site_code, p_lbs_prd_used, v_median,
                   p_acre_planted, p_unit_planted,
                   p_acre_treated, p_unit_treated)
            THEN
               Outlier.Estimate_acres
                  (p_estimated_field, p_error_type, p_replace_type);
            ELSE
               v_old_lbs_prd_used := p_lbs_prd_used;

               p_lbs_prd_used := v_median * p_acre_treated;
               p_amt_prd_used := p_amt_prd_used * p_lbs_prd_used / v_old_lbs_prd_used;

               IF p_lbs_prd_used < 0.0001 THEN
                  p_lbs_prd_used := 0.0001;
               END IF;

               IF p_amt_prd_used < 0.0001 THEN
                  p_amt_prd_used := 0.0001;
               END IF;

               Outlier.Estimate_lbs
                  (p_estimated_field, p_error_type, p_replace_type);
            END IF;
         ELSE
            No_error(p_error_code, p_error_type, p_replace_type);
         END IF;
      ELSE
         No_error(p_error_code, p_error_type, p_replace_type);
      END IF;

   EXCEPTION
      WHEN NO_DATA_FOUND THEN
```

```
        No_error(p_error_code, p_error_type, p_replace_type);
     WHEN OTHERS THEN
        Other_error(v_dummy, v_dummy, p_error_code,
                    p_error_type, p_replace_type, p_use_no);
  END Outliers;


/* Error code 72: check rate of use > neural network outlier value.
  Record as possible error but do not estimate values--leave values unchanged.

  This could be called before Outliers() which  may change some of the rate values.
  In most cases, it will flag these as outliers as well.
  I suggest call it afterwards, and then it will only flag records not flagged by
  the other criteria. It should not flag records flagged by the other criteria
  because Outliers() changes the rates to be at the median rate.

  Must be called after checks for record_id, prodno, site_code, lbs_prd_used,
  acre_treated, unit_treated.
*/
  PROCEDURE Outliers_nn
     (p_use_no IN NUMBER,
      p_record_id VARCHAR2,
      p_prodno IN NUMBER,
      p_site_code IN NUMBER,
      p_lbs_prd_used IN NUMBER,
      p_acre_treated IN NUMBER,
      p_unit_treated IN VARCHAR2,
      p_error_code OUT BINARY_INTEGER,
      p_error_type OUT VARCHAR2,
      p_replace_type OUT VARCHAR2,
      p_require_type OUT VARCHAR2)
  IS
     v_nn4             NUMBER;
     v_rate            NUMBER;
     v_dummy           VARCHAR2(1);
  BEGIN
     IF p_record_id IS NOT NULL AND p_record_id <> '?' AND
        p_prodno IS NOT NULL AND p_prodno <> -1 AND
        p_site_code IS NOT NULL AND p_site_code <> -1 AND
        p_acre_treated > 0 AND
        p_unit_treated IS NOT NULL AND p_unit_treated <> '?' AND
        p_lbs_prd_used IS NOT NULL
     THEN
        p_error_code := 72;

        SELECT   nn4
        INTO     v_nn4
        FROM     usetype99stats
        WHERE    prodno = p_prodno AND
                 site_code = p_site_code AND
                 unit_treated = p_unit_treated AND
                 record_id_type =
                    TRANSLATE(p_record_id, 'C2G9DHAB14EF', 'NNNNNNAAAAAA');

        v_rate := p_lbs_prd_used/p_acre_treated;

        IF v_rate > v_nn4 THEN
           Possible_error(p_error_type, p_replace_type);
        ELSE
           No_error(p_error_code, p_error_type, p_replace_type);
        END IF;

     ELSE
        No_error(p_error_code, p_error_type, p_replace_type);
     END IF;

  EXCEPTION
     WHEN NO_DATA_FOUND THEN
        No_error(p_error_code, p_error_type, p_replace_type);
     WHEN OTHERS THEN
        Other_error(v_dummy, v_dummy, p_error_code,
                  p_error_type, p_replace_type, p_use_no);
END Outliers_nn;


/* Error code 80: Check for duplicate records that appear to be errors.
   p_use_no_array stores all the use_no which are duplicates of
   each other, including the current record, which is the last value
   in the array.

   Records are considered duplicate errors if 2 or more records have
   the same values for county_cd, grower_id, site_loc_id,
   site_code, qualify_cd, prodno, amt_prd_used, unit_of_meas, acre_treated,
   unit_treated, acre_planted, unit_planted, applic_dt
   (and TRS???) and
   if the sum of acre_treated for these records is greater than
   the acre_planted.
   I would say don't include TRS, since site_loc_id is supposed
   to indicate field.  If site_loc_id does represent different
   fields, then that is a kind of error, too.

   In version 2.2, took out applic_time on recommendation of
   Linda Lichtenberger because not all counties use applic_time,
   or don't use it correctly.

   These duplicates could be caused by the records getting submitted twice,
   or by spot treatments to field where acre_treated was set = acre_planted,
   or by different fields being given the same site_loc_id

   You need to check if there are any other records that duplicate the
   current record in PUR2001.
   Keep track of error duplicate sets in errors table with error_code = 80
   and recording the set number in field duplicate_set to identify
   a set of duplicate records.

   Don't include record_id since same record may be have entered using
   different record_ids (eg if both grower and PCO submitted forms).

   Check only prod ag records? Only for prod ag can we check
   sum(acre_treated) > acre_planted, but for non prod ag we could
   flag all duplicates.

   Check only records where all dup fields are not null?
   Probably we do want to flag them, because the raw data
   are most likely the same, so that is what I've done.

   Here I check for duplicates in pur2001, not raw2001.  It might make
   more sense to test raw2001 but what people who query the data
   will see if pur, not raw, so I think possible duplicates in pur
   should be flagged.  In any case, duplicates in pur and raw will
   nearly always be the same.

   Note that p_use_no_array will only contain two use_nos for
   error duplicates if acre_treated = acre_planted
   because only one record from such an error duplicate set
   is recorded in the pur.  The first use_no in p_use_no_array
   will be the one record from the duplicate set already in the pur
   and the second use_no will be for the current record, which
   will not be inserted into pur.  If acre_treated < acre_planted,
   there can be more than 2 use_nos in p_use_no_array.


   Must be called after checks for record_id, county_cd, grower_id,
   site_loc_id, prodno, site_code, amt_prd_used, unit_of_meas,
   acre_treated, unit_treated, acre_planted, unit_planted,
   applic_dt, applic_time.

   -- Why all records from duplicate set not in errors table?
      Because need outlier join when displaying join
      of errors and pur.

   -- Test for set of records with treated < planted, but where
```

```
      sum(treated) > planted. Here we want to record in PUR all records
      except for those that make sum(treated) > planted.

   -- Add records to changes table

 */
PROCEDURE Duplicates
   (p_use_no IN NUMBER,
    p_record_id IN VARCHAR2,
    p_county_cd IN VARCHAR2,
    p_grower_id IN VARCHAR2,
    p_site_loc_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_qualify_cd IN NUMBER,
    p_prodno IN NUMBER,
    p_amt_prd_used IN NUMBER,
    p_unit_of_meas IN VARCHAR2,
    p_acre_treated IN NUMBER,
    p_unit_treated IN VARCHAR2,
    p_acre_planted IN NUMBER,
    p_unit_planted IN VARCHAR2,
    p_applic_dt IN DATE,
    p_use_no_array OUT use_no_array_type,
    p_error_code OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2,
    p_replace_type OUT VARCHAR2,
    p_require_type OUT VARCHAR2)
IS
   v_dup_no   BINARY_INTEGER;
   v_area_vol_treated   NUMBER(14, 2) := NULL;
   v_area_vol_planted   NUMBER(14, 2) := NULL;
   v_dummy           VARCHAR2(1);

   /* Does this query run even if it is not called in the code below
      (it will not be called if the record is non ag)?
    */
   CURSOR   dup_cur IS
      SELECT   use_no
      FROM     pur2001
      WHERE    county_cd = p_county_cd AND
               grower_id = p_grower_id AND
               site_loc_id = p_site_loc_id AND
               site_code = p_site_code AND
               qualify_cd = p_qualify_cd AND
               prodno = p_prodno AND
               NVL(amt_prd_used, 0) = NVL(p_amt_prd_used, 0) AND
               unit_of_meas = p_unit_of_meas AND
               NVL(acre_treated, 0) = NVL(p_acre_treated, 0) AND
               unit_treated = p_unit_treated AND
               NVL(acre_planted, 0) = NVL(p_acre_planted, 0) AND
               unit_planted = p_unit_planted AND
               NVL(applic_dt, '01-JAN-1900') = NVL(p_applic_dt, '01-JAN-1900');

BEGIN
   p_error_code := 80;

   IF p_record_id IN ('1','4','A','B') THEN
      p_require_type := 'required';
   ELSIF p_record_id IN ('2','C') THEN
      p_require_type := 'not_allowed';
   ELSIF p_record_id IS NULL THEN
      p_require_type := 'unknown';
   ELSE
      p_require_type := 'unknown';
   END IF;

   IF p_require_type = 'required' THEN
      /* Change area treated and planted to common measures.
         Note that we have already checked that both measures
         are compatiable, that is, if area treated is A or S
         then area planted is A or S, etc.
```

```
        */
       IF p_unit_treated IN ('A', 'C', 'U') THEN
          v_area_vol_treated := p_acre_treated;
       ELSIF p_unit_treated = 'S' THEN
          v_area_vol_treated := p_acre_treated/43560;
       ELSIF p_unit_treated = 'K' THEN
          v_area_vol_treated := p_acre_treated * 1000;
       END IF;

       IF p_unit_planted IN ('A', 'C', 'U') THEN
          v_area_vol_planted := p_acre_planted;
       ELSIF p_unit_planted = 'S' THEN
          v_area_vol_planted := p_acre_planted/43560;
       ELSIF p_unit_planted = 'K' THEN
          v_area_vol_planted := p_acre_planted * 1000;
       END IF;


       /* Find if there any other records already in pur2001
          with the same values as the current record.
          [Note that the current record has not uet been
           inserted into pur2001]
        */
       v_dup_no := 0;
       FOR dup_rec IN dup_cur LOOP
          v_dup_no := v_dup_no + 1;
          p_use_no_array(v_dup_no) := dup_rec.use_no;
       END LOOP;

       v_dup_no := v_dup_no + 1;
       IF v_dup_no > 1 AND
          v_area_vol_treated * v_dup_no > v_area_vol_planted
       THEN
          /* Add current record to this duplicate set
             and report as probable error.
           */
          p_use_no_array(v_dup_no) := p_use_no;
          Duplicate_error(p_error_type, p_replace_type);
       ELSE
          No_error(p_error_code, p_error_type, p_replace_type);
       END IF;

    ELSE
       No_error(p_error_code, p_error_type, p_replace_type);
    END IF;

 EXCEPTION
    WHEN NO_DATA_FOUND THEN
       No_error(p_error_code, p_error_type, p_replace_type);
    WHEN OTHERS THEN
       Other_error(v_dummy, v_dummy, p_error_code,
                   p_error_type, p_replace_type, p_use_no);
 END Duplicates;





 /* Called if no errors were found
  */
 PROCEDURE No_error
    (p_error_code OUT VARCHAR2, p_error_type OUT VARCHAR2,
     p_replace_type OUT VARCHAR2)
 IS
 BEGIN
    p_error_code := NULL;
    p_error_type := 'N';
    p_replace_type := 'same';
 END No_error;
```

```
-- Invalid errors:
/* Called if invalid errors were found in p_out_value.  These errors
   include values not allowed for this variable and can be caused
   by throwing INVALID_ERROR.
 */
PROCEDURE Invalid_error
   (p_value IN VARCHAR2, p_out_value OUT NUMBER,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_out_value := NULL;
   p_error_type := 'invalid';
   p_replace_type := Null_or_same(p_value);
END Invalid_error;

/* Called if invalid errors were found in character p_out_value.
   These errors include values not allowed for this variable and
   can be caused by throwing INVALID_ERROR.
 */
PROCEDURE Invalid_char_error
   (p_value IN VARCHAR2, p_out_value OUT VARCHAR2,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_out_value := '?';
   p_error_type := 'invalid';
   p_replace_type := Null_or_same(p_value);
END Invalid_char_error;

/* Called if invalid errors were found in character p_out_value.
   These errors include values not allowed for this variable and
   can be caused by throwing INVALID_ERROR.
 */
PROCEDURE Invalid_num_error
   (p_value IN VARCHAR2, p_out_value OUT NUMBER,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_out_value := -1;
   p_error_type := 'invalid';
   p_replace_type := Null_or_same(p_value);
END Invalid_num_error;

/* Called if invalid errors were found in p_out_value.  These errors
   include values not allowed for this variable and can be caused
   by throwing INVALID_ERROR.
 */
PROCEDURE Invalid_date_error
   (p_value IN VARCHAR2, p_out_value OUT DATE,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_out_value := NULL;
   p_error_type := 'invalid';
   p_replace_type := Null_or_same(p_value);
END Invalid_date_error;


-- Probable errors:
/* Called if probable errors were found in p_out_value.
   Value is left unchanged.
   These errors include values that could be valid
   but are incorrect for some other reason.
 */
PROCEDURE Probable_error
   (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_error_type := 'probable';
   p_replace_type := 'same';
END Probable_error;
```

```
/* Called if probable errors were found in p_out_value but
   value is estimated with value in p_estimate.
   These errors include values that could be valid
   but are incorrect for some other reason.
 */
PROCEDURE Probable_estimated_error
   (p_estimate IN VARCHAR2, p_out_value OUT VARCHAR2,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_out_value := p_estimate;
   p_error_type := 'probable';
   p_replace_type := 'estimate';
END Probable_estimated_error;

/* Called if probable errors were found in p_out_value and
   value was corrected.
   These errors include values that could be valid
   but are incorrect for some other reason.
   Because they are corrected, they do not show up in ERRORS table
 */
PROCEDURE Probable_corrected_error
   (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_error_type := 'probable';
   p_replace_type := 'correct';
END Probable_corrected_error;


-- Possible errors:
/* Called if possible errors were found in p_out_value and
   value is left unchanged.
   Currently, only situations that generates these are
   rate values greater than the neural network criterion,
   mtrs or acres planted were inconsistent for an
   agricultural field, or where unit_treated = U,
   unit_planted = A, and acre_treated > acre_planted.
   Values are left unchanged
 */
PROCEDURE Possible_error
   (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_error_type := 'possible';
   p_replace_type := 'same';
END Possible_error;

/* Called if possible invalid errors were found in p_out_value
   when requirements were unknown and where some values are
   optional or not required and other values are required.
 */
PROCEDURE Possible_invalid_error
   (p_value IN VARCHAR2, p_out_value OUT VARCHAR2,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_out_value := NULL;
   p_error_type := 'possible';
   p_replace_type := Null_or_same(p_value);
END Possible_invalid_error;

/* Called if possible invalid errors were found in char p_out_value
   when requirements were unknown and where some values are
   optional or not required and other values are required.
 */
PROCEDURE Possible_invalid_char_error
   (p_value IN VARCHAR2, p_out_value OUT VARCHAR2,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
```

```
BEGIN
   p_out_value := '?';
   p_error_type := 'possible';
   p_replace_type := Null_or_same(p_value);
END Possible_invalid_char_error;


-- Duplicate errors
/* Called if duplicate errors were found in current record.
   Only one record from a duplicate set is left in the PUR;
   the others are not reported.
 */
PROCEDURE Duplicate_error
   (p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_error_type := 'probable';
   p_replace_type := 'delete';
END Duplicate_error;

/* Called when require_type is not_allowed.  In these cases, any
   errors are ignored.
 */
PROCEDURE Not_allowed
   (p_value IN VARCHAR2, p_out_value OUT VARCHAR2, p_error_code OUT VARCHAR2,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
IS
BEGIN
   p_out_value := NULL;
   p_error_code := NULL;
   p_error_type := 'N';
   p_replace_type := Null_or_same(p_value);
END Not_allowed;




PROCEDURE Other_error
   (p_value IN VARCHAR2, p_out_value OUT VARCHAR2, p_error_code IN OUT BINARY_INTEGER,
    p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2,
    p_use_no IN NUMBER)
IS
BEGIN
   Other_exceptions(p_use_no, p_error_code);
   p_out_value := NULL;
   p_error_code := 0;
   p_error_type := 'invalid';
   p_replace_type := Null_or_same(p_value);
END Other_error;

/* Function used in error sections of each field error check
   Returns 'null' or 'same' depending whether the value should be
   replaced by a null (or '?' or -1) or left unchanged.
   p_value is the original field value;
   v_num_chars is the number of characters in p_value
 */
FUNCTION Null_or_same (p_value IN VARCHAR2)
RETURN VARCHAR2 IS
   v_replace_type    VARCHAR2(20);
   v_empty_string    VARCHAR2(100);
   v_num_chars       NUMBER;
BEGIN
   v_num_chars := NVL(LENGTH(p_value),0);
   v_empty_string := '';
   FOR i IN 1..v_num_chars LOOP
      v_empty_string := v_empty_string||' ';
   END LOOP;

   IF p_value IS NULL OR p_value = v_empty_string THEN
      v_replace_type := 'same';
   ELSE
```

```
         v_replace_type := 'null';
      END IF;

      RETURN v_replace_type;
   END Null_or_same;



   /* Print out error message...
 */
   PROCEDURE Other_exceptions(p_use_no IN NUMBER, p_error_code IN BINARY_INTEGER)
   IS
      error_msg       VARCHAR2(300) := SQLERRM;
   BEGIN
      DBMS_OUTPUT.PUT_LINE('**** Check_value2001: Other error for error code '||p_error_code||
       ' and use_no '||p_use_no||  ': ' || SQLERRM);
   END;
END Check_value2001;
/
show errors

EXIT 0
```

Co_error2001.sql:

```
set termout on
SET document off
SET FEEDBACK ON
SET verify off
SET pause off
set serveroutput on size 1000000 format word_wrapped
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

/* When you change the version, you should change the variable
   v_loader_name

   version 2.0 was used for pur99.
   vertions 2.1, 2.2 was used for pur2000.

   version 2.1 had a few minor changes:
      error code 17:
         Before, if year was wrong, replace applic_dt with null.
         Now, estimate year as current PUR year and flag as probable error.

      error codes 67, 48:
         Befeore, if MTRS not found in California or County, made MTRS values NULL.
         Now, set each MTRS value = '?'.

   version 2.2 changes:
      error code 61:
         before, I treated situation where unit_treated = 'U' and unit_planted = 'A'
            as a special case.
         now, this situation is considered acceptable and is not flagged as an error.

      error code 14:
         before, used label database site table to check if site_code exists.
         now, use PUR site table to check site_code.

      error code 17:
         before, if year was not equal to current PUR year, flagged as error
         now, if year is not equal to current or previous yera, flagged as error.

      error codes 47 and 80:
         increased the number precision of v_cubic_feet_treated, v_cubic_feet_olanted,
         v_area_vol_treated, and v_area_vol_planted.

      error code 67:
         before, in exception seciton set p_ca_mtrs_acres = '?' which generated
         an error message because p_ca_mtrs_acres is a number type.
         Now p_ca_mtrs_acres set = null.
         Note that even though an error message was generated, this error
         had no effect on any of the data, but p_ca_mtrs_acres was set to null
         by default in any case.

      error code 80:
         took out applic_time from fields used in duplicate check.
         Also, at some point between version 2.0 and 2.2, in last condition of where
         clause in dup_cur, "applic_dt" was changed to "p_applic_dt", i.e.
         it read NVL(p_applic_dt, '01-JAN-1900') = NVL(p_applic_dt, '01-JAN-1900');
         it should be "applic_dt" as originally.

   version 2.21 changes:
      error code 14:
         site_code = 99999 is flagged as error.

   version 2.3 changes:
      error code 39:
         Changed the procedure for site code not on label.
         Now if site does not appear on the label for the reported pesticide
         product, the program will check if the site is on the label for
         any product which has the same first two parts of the registration
         number as the reported product.  If it finds another product
         with the site on the label, it will mark this as a possible error
         but will not be reported to the county.

       Also, added another procedure to the loader.pl.  When the loader if
       finished with a batch of county files, it will produce a new list
       of all county files loaded.  This file is at:
       /Scalos_ora05/data/2001data/loaded_files_2001.txt

   version 2.4 changes:
      error code 43:
         For C records in years after 1999, if license_no is present, grower_id
         is not required (before was optional).  In this case, the loader
         checks to see if they grower_id and license_no contain the same values;
         if they do, grower_id is set to null and no errors are reported;
         if they contain different values an error is reported.

   version 2.41 changes:
      error code 43:
         Still trying to get this check right.
         For C records in years after 1999, user can give either license_no or
         grower_id or both, but they must provide at least one. This was the
         requirement before version 2.4 and is still the requirement.
         What is different now is that
         if grower_id and license_no have the same values
         (or if license_no is the same as substr(grower_id, 5)) then
            if grower_id is in the right format, then grower_id is accepted and
               license_no is replaced with null, no error report is given;
            if grower_id is not in right format, then grower_id is replaced
               with null, and license_no is accepted, no error report given.

   version 2.42 changes:
      error codes 23 and 60:
         Previously, values stored in old_value for acre_treated or acre_planted
         for error codes 23 and 60 were the values from the RAW table
         without dividing by 100 but for other error codes (47 and 75)
         the values in old_value were divided by 100.
         Now all values for old_value are divided by 100, except for error codes
         18 and and 44 which are invalid character errors (so they could
         for example not be numeric).
         I have also changed all such previous old_values so they are all
         divided by 100.

   version 2.43 changes:
      error codes 23, 60. Added special exception for PUR year=2000 to
         allow large fields for certain growers.  Ada Ann Scott explains:
         "It seems that Monterey County issues permits in year=2000 to one or two
         growers for sites over the 640 acre limitations and have submitted
         hundreds of PURs for sites exceeding the section acreage. ...
         It seems that Monterey misinterpreted the 'Site Definition'.
         Bottom line, they have agreed to correct the problem in their
         2001 permit/operator ID database.  Any site that might possibly
         exceed the section limits should be identified as errors
         in the 2001 PUR."

      loader.pl.  Added procedure to split county data files with more
         than 15,000 lines, append "_1", "_2", etc to end of the new
         smaller files.

   version 2.44 changes:
      error code 43: Allow any value for grower_id for San Francisco County
         (county_cd = 38) and record_id 2 or C.  Since SF County has no
         production agricultural reporting, they store additional data
         related to pest control businesses in this field in their permit database.


   version 2.5 changes:
       error_code 63:  Inconsistent_site_loc_id is not called now
         because it slowed loading too much, it was not called
         during error corrections, and it really needs to be
         improved.  This procedure should be run after loading.
       error_code 43: Where grower_id is compared to license_no to
         see if they are the same, added condition:
         RTRIM(LTRIM(p_license_no)) = RTRIM(LTRIM(SUBSTR(p_grower_id, 7))).
         There were no records meeting this condition, but it
         seemed like a possibility in the future.
         Also, in if condition just above that, changed
         SUBSTR(p_grower_id, 5, 2) BETWEEN 1 and 58
         to
         SUBSTR(p_grower_id, 5, 2) IN
             ('01','02','03','04','05','06','07','08','09','10',
              '11','12','13','14','15','16','17','18','19','20',
              '21','22','23','24','25','26','27','28','29','30',
              '31','32','33','34','35','36','37','38','39','40',
              '41','42','43','44','45','46','47','48','49','50',
              '51','52','53','54','55','56','57','58')).
         I made this change because if the value
         in SUBSTR(p_grower_id, 5, 2) is not a number, a VALUE_ERROR will be
         raised which takes the procedure to the exception clause.

   version 2.51 changes:
      Error code 43: Now program does not flag substr(grower_id,7) = '00000' as error
         for production ag records from San Joaquin County (county_cd = '39').
         Grower_id = '00000' was assigned as first grower_id in San Joaquin County.

   version 2.6 changes: Made at start of 2001 PUR (10/3/01).
      Error code 22: New error check added. For non ag check if acre_treated > 700 (unit = A) or
         acre_treated > 999999 (unit = S). For estimate, let unit_treated = U.

      Error code 80: Added qualify_cd to list of fields to include in duplicate check.

      Error code 17: Previously, the program accepted reported year for the previous year.
         However, now this is considered a probable error: some may be correct,
         but most seem to be errors and there were only a few such cases in 2000.

      Error code 43: If year part of grower_id is not current year, previously
         replace_type = correct, now replace_type = estimate.

      Error code 37 and 39: Many changes were made to procedures for identifying and
         reporting errors for prodno. See loader documentation for full details.

         Some example changes:
         1) If product is found with reported full registration number, the program
            uses this even if site_code is not on label--before it only did so
            if site was on label.

         2) If no product is found with same first two parts of the reg. no. the reported
            registration is given in the errors table--before this was left blank.

         3) If either revision_no or reg_firmno was changed, it is reported as a
            possible error--before is was considered a correction.

   version 2.61 changes: 10/17/01.
      Error code 17: Change code so that it will not try to estimate year if
         applic_day or applic_month is NULL; previously, this would generate
         error code 0 message because null applic_day or applic_month
         would cause failure in TO_DATE() function.

      Error code 43: Previously, error report was generated if year part of grower_id
         was not equal to year of application.  Now, year part of grower_id can equal
         any year between current year and current year + 2.  The year part of the
         grower_id can be (or should be?) the expiration date of the permit,
         and permits can be valid for up to three years.

   version 2.62 changes: 1/3/02.
       Error codes 22, 23, 60: Added pastureland (28035) to list of sites that are
         excluded in checking size of acres treated and planted.

      Error code 43: Allow year part of grower_id to equal any year between
         current year - 3 and current year + 3.
         Permits can be valid for up to three years.
         In some counties, the year is the expiration date of the permit,
         and in other counites, the year is the date the permit was issued.
         Because permits are issued in July in Imperial the year part of the
         grower_id differ from the PUR year by 3.

   version 2.63 changes: 1/330/02.
      Error code 14: Added site code 28509 (orchard floor) to list of valid
         site codes that can be used.

 */

/* This code is normally used to process each individual county PUR file
   by reading the data into intermediate tables which then get inserted
   into the RAW table.
   However, sometimes we need to recreate the PUR tables from data already
   loaded into RAW table (for example, if we change some of the error
   checks after the data was loaded).  In that case, we need to do
   a batch run.  The changes that need to be made to the code for batch
   processing are marked by the comments:

   /*************************
    For normal PUR processing use:

    For batch PUR processing  use:

 */

CREATE OR REPLACE PACKAGE Co_error2001 AS
   v_loader_name     VARCHAR2(20) := 'loader 2.63';

   PROCEDURE Check_records;

   PROCEDURE Log_error_change
      (p_use_no IN NUMBER, p_error_code IN BINARY_INTEGER,
       p_column_name IN VARCHAR2, p_old_value IN VARCHAR2,
       p_new_value IN VARCHAR2, p_error_type IN VARCHAR2,
       p_replace_type IN VARCHAR2, p_require_type IN VARCHAR2,
       p_who IN VARCHAR2, p_comments IN VARCHAR2);

   PROCEDURE Log_duplicates
      (p_use_no IN NUMBER, p_use_no_array IN Check_value2001.use_no_array_type,
       p_error_code IN BINARY_INTEGER, p_error_type IN VARCHAR2,
       p_replace_type IN VARCHAR2, p_require_type IN VARCHAR2,
       p_who IN VARCHAR2, p_comments IN VARCHAR2);

END Co_error2001;
/
show errors

CREATE OR REPLACE PACKAGE BODY Co_error2001 AS
/* Forward declarations of private procedures
 */
   PROCEDURE Other_exceptions(p_use_no IN NUMBER, p_error_code IN BINARY_INTEGER);
   PROCEDURE General_exceptions;
   PROCEDURE Update_fields
      (p_use_no IN NUMBER,
       p_county_cd IN VARCHAR2,
       p_operator_id IN VARCHAR2,
       p_site_loc_id IN VARCHAR2,
       p_site_code IN NUMBER,
       p_site_loc_id_state IN VARCHAR2,
       p_this_mtrs IN mtrs_type,
       p_this_acres IN NUMBER,
       p_previous_mtrs IN mtrs_type,
       p_previous_acres IN NUMBER,
       p_error_code IN BINARY_INTEGER,
       p_field_id IN OUT NUMBER);
```

```
/* Check for errors in each column of all records in raw2001i
 */
   PROCEDURE Check_records
   AS
      v_use_no          NUMBER(8);
      v_error_code      BINARY_INTEGER := 0;
      v_error_type      VARCHAR2(20);
      v_replace_type    VARCHAR2(20);
      v_require_type    VARCHAR2(20);

      v_record_id       VARCHAR2(1);
      v_batch_no        NUMBER(4);
      v_process_mt      NUMBER(2);
      v_process_yr      NUMBER(4);
      v_county_cd       VARCHAR2(2);
      v_site_code       NUMBER(6);
      v_applic_month    NUMBER(2);
      v_applic_day      NUMBER(2);
      v_applic_year     NUMBER(4);
      v_applic_dt       DATE;
      v_document_no     NUMBER(8);
      v_summary_cd      NUMBER(4);
      v_site_loc_id     VARCHAR2(8);
      v_cedts_ind       VARCHAR2(1);
      v_qualify_cd      NUMBER(2);
      v_planting_seq    NUMBER(1);

      v_section         VARCHAR2(2);
      v_township        VARCHAR2(2);
      v_tship_dir       VARCHAR2(1);
      v_range           VARCHAR2(2);
      v_range_dir       VARCHAR2(1);
      v_base_ln_mer     VARCHAR2(1);
      v_ca_mtrs_acres   NUMBER(10,2);
      v_aer_gnd_ind     VARCHAR2(1);
      v_applic_time     VARCHAR2(4);
      v_grower_id       VARCHAR2(11);
      v_license_no      VARCHAR2(13);

      v_mfg_firmno      NUMBER(7);
      v_label_seq_no    NUMBER(5);
      v_revision_no     VARCHAR2(2);
      v_reg_firmno      NUMBER(7);
      v_new_revision_no VARCHAR2(2);
      v_new_reg_firmno  NUMBER(7);
      v_other_product   VARCHAR2(50);
      v_amt_prd_used    NUMBER(12,4);
      v_amt_prd_used_old NUMBER(12,4);
      v_unit_of_meas    VARCHAR2(2);
      v_spec_gravity    NUMBER(6,4);
      v_formula_cd      VARCHAR2(2);
      v_prodno          NUMBER(6);
      v_lbs_prd_used    NUMBER(14,4);
      v_lbs_prd_used_old NUMBER(14,4);
      v_acre_treated    NUMBER(10,2);
      v_acre_treated_old NUMBER(10,2);
      v_unit_treated    VARCHAR2(1);
      v_unit_treated_old VARCHAR2(1);
      v_acre_planted    NUMBER(10,2);
      v_acre_planted_old NUMBER(10,2);
      v_unit_planted    VARCHAR2(1);
      v_unit_planted_old VARCHAR2(1);

      v_applic_cnt      NUMBER(6);

      v_estimated_field VARCHAR2(20);
      v_estimated_date   DATE;
      v_criteria        VARCHAR2(100);

      v_use_no_array    Check_value2001.use_no_array_type;
      v_error_duplicate BOOLEAN;
      v_field_id           NUMBER(8);
      v_this_mtrs          mtrs_type;
      /*
      v_operator_id        VARCHAR2(7);
      v_site_loc_id_state  VARCHAR2(50);
      v_this_acres         NUMBER(10, 2);
      v_previous_mtrs      mtrs_type;
      v_previous_acres     NUMBER(10, 2);
      */

      /************************
       For normal PUR processing use:
       */
      CURSOR record_cur IS
         SELECT        *
         FROM          raw2001i
         FOR UPDATE OF use_no
         NOWAIT;

      /* **********************
       For batch PUR processing use:
      CURSOR record_cur IS
         SELECT        *
         FROM          raw2001;
      */

   BEGIN
      /* It seems that the first output line never appears, so
         if no output statements occur before an error message,
         the error message will not appear.
       */
      DBMS_OUTPUT.PUT_LINE('Start Check_records()');
      DBMS_OUTPUT.PUT_LINE('Start Check_records()');
      FOR rec IN record_cur LOOP

         /************************
          For normal PUR processing use:
          */
         SELECT pur_seq2001.NextVal
         INTO   v_use_no
         FROM   dual;

         /* **********************
          For batch PUR processing use:
         v_use_no := rec.use_no;
         */

         /* For testing...
          */
         --DBMS_OUTPUT.PUT_LINE('1: use_no = '||v_use_no);

         /* Error code 1: Record_id
          */
         Check_value2001.Record_id
            (v_use_no, rec.record_id, v_record_id,
             v_error_code, v_error_type, v_replace_type, v_require_type);
         IF v_error_type <> 'N' THEN
            Log_error_change
               (v_use_no, v_error_code, 'record_id', rec.record_id,
                v_record_id, v_error_type, v_replace_type, v_require_type,
                v_loader_name, NULL);
         END IF;

         /* For testing...
          */
         --DBMS_OUTPUT.PUT_LINE('2: use_no = '||v_use_no);

         /* Error code 2: batch_no
          */
         Check_value2001.Batch_no
      (v_use_no, rec.batch_no, v_batch_no,
       v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      Log_error_change
         (v_use_no, v_error_code, 'batch_no', rec.batch_no,
          v_batch_no, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   END IF;

   /* Error code 3: process_mt
    */
   Check_value2001.Process_mt
      (v_use_no, rec.process_mt, v_process_mt,
       v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      Log_error_change
         (v_use_no, v_error_code, 'process_mt', rec.process_mt,
          v_process_mt, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   END IF;

   /* Error code 4: process_yr
    */
   Check_value2001.Process_yr
      (v_use_no, rec.process_yr, v_process_yr,
       v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      Log_error_change
         (v_use_no, v_error_code, 'process_yr', rec.process_yr,
          v_process_yr, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   END IF;

   /* Error code 5: county_cd
    */
   Check_value2001.County_cd
      (v_use_no, rec.county_cd, v_county_cd,
       v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      Log_error_change
         (v_use_no, v_error_code, 'county_cd', rec.county_cd,
          v_county_cd, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   END IF;

   /* Error code 13, 14: site_code
    */
   Check_value2001.Site_code
      (v_use_no, rec.site_code, v_site_code,
       v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      Log_error_change
         (v_use_no, v_error_code, 'site_code', rec.site_code,
          v_site_code, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   END IF;

   /* Error code 15: applic_month
    */
   Check_value2001.Applic_month
      (v_use_no, SUBSTR(rec.applic_dt, 1, 2), v_applic_month,
       v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      Log_error_change
         (v_use_no, v_error_code, 'applic_dt', rec.applic_dt,
          NULL, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   END IF;

   /* Error code 16: applic_day
    */
```

```
Check_value2001.Applic_day
   (v_use_no, SUBSTR(rec.applic_dt, 3, 2), v_applic_day,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'applic_dt', rec.applic_dt,
       NULL, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 17: applic_year
 */
Check_value2001.Applic_year
   (v_use_no, SUBSTR(rec.applic_dt, 5, 2), v_applic_month, v_applic_day,
    v_applic_year, v_estimated_date,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'applic_dt', rec.applic_dt,
       v_estimated_date, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 51: applic_dt
 */
Check_value2001.Applic_dt
   (v_use_no, v_applic_month, v_applic_day, v_applic_year,
    rec.applic_dt, v_applic_dt, v_error_code,
    v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'applic_dt', rec.applic_dt,
       v_applic_dt, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 34: check document_no
 */
Check_value2001.Document_no
   (v_use_no, v_record_id, v_applic_day, rec.document_no,
    v_document_no, v_error_code, v_error_type,
    v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'document_no', rec.document_no,
       v_document_no, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 35: summary_cd
 */
Check_value2001.Summary_cd
   (v_use_no, rec.summary_cd, v_summary_cd,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'summary_cd', rec.summary_cd,
       v_summary_cd, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 62: site_loc_id
 */
Check_value2001.Site_loc_id
   (v_use_no, v_record_id, rec.site_loc_id, v_site_loc_id,
    v_error_code, v_error_type, v_replace_type, v_require_type);
```

```
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'site_loc_id', rec.site_loc_id,
       v_site_loc_id, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 64: cedts_ind
 */
Check_value2001.Cedts_ind
   (v_use_no, rec.cedts_ind, v_cedts_ind,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'cedts_ind', rec.cedts_ind,
       v_cedts_ind, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 65: qualify_cd
 */
Check_value2001.Qualify_cd
   (v_use_no, rec.qualify_cd, v_qualify_cd,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'qualify_cd', rec.qualify_cd,
       v_qualify_cd, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 66: planting_seq
 */
Check_value2001.Planting_seq
   (v_use_no, v_record_id, rec.planting_seq, v_planting_seq,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'planting_seq', rec.planting_seq,
       v_planting_seq, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;


/* Error code 6: section
 */
Check_value2001.Section
   (v_use_no, v_record_id, rec.section, v_section,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'section', rec.section,
       v_section, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 7: township
 */
Check_value2001.Township
   (v_use_no, v_record_id, rec.township, v_township,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'township', rec.township,
       v_township, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;
```

```
/* Error code 8: tship_dir
 */
Check_value2001.Tship_dir
   (v_use_no, v_record_id, rec.tship_dir, v_tship_dir,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'tship_dir', rec.tship_dir,
       v_tship_dir, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 9: range
 */
Check_value2001.Range
   (v_use_no, v_record_id, rec.range, v_range,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'range', rec.range,
       v_range, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 10: range_dir
 */
Check_value2001.Range_dir
   (v_use_no, v_record_id, rec.range_dir, v_range_dir,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'range_dir', rec.range_dir,
       v_range_dir, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 11: base_ln_mer
 */
Check_value2001.Base_ln_mer
   (v_use_no, v_record_id, rec.base_ln_mer, v_base_ln_mer,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'base_ln_mer', rec.base_ln_mer,
       v_base_ln_mer, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 67: mtrs
 */
Check_value2001.Mtrs
   (v_use_no, v_record_id, v_base_ln_mer, v_township, v_tship_dir, v_range,
    v_range_dir, v_section, v_ca_mtrs_acres,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'mtrs', rec.base_ln_mer||rec.township||
       rec.tship_dir||rec.range||rec.range_dir||rec.section,
       NULL, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 48: comtrs
 */
Check_value2001.Comtrs
   (v_use_no, v_record_id, v_county_cd, v_base_ln_mer, v_township, v_tship_dir, v_range,
    v_range_dir, v_section,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'comtrs', v_county_cd||rec.base_ln_mer||rec.township||
       rec.tship_dir||rec.range||rec.range_dir||rec.section,
       NULL, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 12: aer_gnd_ind
 */
Check_value2001.Aer_gnd_ind
   (v_use_no, v_record_id, rec.aer_gnd_ind, v_aer_gnd_ind,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'aer_gnd_ind', rec.aer_gnd_ind,
       v_aer_gnd_ind, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 68: applic_time.
 */
Check_value2001.Applic_time
   (v_use_no, v_record_id, rec.applic_time, v_applic_time,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'applic_time', rec.applic_time,
       v_applic_time, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 43: grower_id
 */
Check_value2001.Grower_id
   (v_use_no, v_record_id, v_county_cd, rec.license_no, v_applic_year,
    rec.grower_id, v_grower_id, v_error_code, v_error_type,
    v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'grower_id', rec.grower_id,
       v_grower_id, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 69: license_no
 */
Check_value2001.License_no
   (v_use_no, v_record_id, v_grower_id, rec.license_no,
    v_license_no, v_error_code, v_error_type,
    v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'license_no', rec.license_no,
       v_license_no, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 31: amt_prd_used
 */
Check_value2001.Amt_prd_used
   (v_use_no, rec.amt_prd_used, v_amt_prd_used,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'amt_prd_used', rec.amt_prd_used,
       v_amt_prd_used, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 32: unit_of_meas
 */
Check_value2001.Unit_of_meas
   (v_use_no, rec.unit_of_meas, v_unit_of_meas,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'unit_of_meas', rec.unit_of_meas,
       v_unit_of_meas, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 24: mfg_firmno
 */
Check_value2001.Mfg_firmno
   (v_use_no, rec.mfg_firmno, v_mfg_firmno,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'mfg_firmno', rec.mfg_firmno,
       v_mfg_firmno, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 25: label_seq_no
 */
Check_value2001.Label_seq_no
   (v_use_no, rec.label_seq_no, v_label_seq_no,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'label_seq_no', rec.label_seq_no,
       v_label_seq_no, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* revision_no
   Previously had Error code 26, but does not generate error now
   because if there is error in revision_no or reg_firmno,
   this program (in function Check_value2001.Prodno) will search
   for valid product with given mfg_firmno and label_seq_no.
   We still call this function to replace value with null
   if it is invalid value.
 */
Check_value2001.Revision_no
   (v_use_no, rec.revision_no, v_revision_no,
    v_error_code, v_error_type, v_replace_type, v_require_type);

/* reg_firmno
   (Previously had Error code 27)
 */
Check_value2001.Reg_firmno
   (v_use_no, rec.reg_firmno, v_reg_firmno,
    v_error_code, v_error_type, v_replace_type, v_require_type);

/* Error code 37: prodno
   No value for prodno is given in the county files.
   This value is found from the label database using the
   registration numbers.  Only the 3-part reg numbers
   are consistently given; defaults are often used for the
   revision_no so these may be wrong.

   Check_value2001.Prodno() will return the likely correct
   values for revision_no (in variables
   v_new_revision_no and v_new_reg_firmno.
 */
Check_value2001.Prodno
   (v_use_no, v_mfg_firmno, v_label_seq_no, v_revision_no, v_reg_firmno,
    v_prodno, v_new_revision_no,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   IF v_replace_type = 'null' THEN
      Log_error_change
         (v_use_no, v_error_code, 'registration number',
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_revision_no||'-'||v_reg_firmno,
          NULL, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   ELSE
      Log_error_change
         (v_use_no, v_error_code, 'registration number',
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_revision_no||'-'||v_reg_firmno,
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_new_revision_no||'-'||v_reg_firmno,
          v_error_type, v_replace_type, v_require_type, v_loader_name,
          'prodno = '||v_prodno);

      v_revision_no :=  v_new_revision_no;
   END IF;
END IF;


/* Error code 39: Prod_site: check if site is on label for prodno.
 */
Check_value2001.Prod_site
   (v_use_no, v_mfg_firmno, v_label_seq_no, v_revision_no, v_reg_firmno,
    v_unit_of_meas, v_site_code, v_applic_dt,
    v_prodno, v_new_revision_no, v_new_reg_firmno, v_other_product,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   IF v_replace_type = 'null' THEN
      Log_error_change
         (v_use_no, v_error_code, 'registration number',
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_revision_no||'-'||v_reg_firmno,
          NULL, v_error_type, v_replace_type, v_require_type,
          v_loader_name, NULL);
   ELSIF v_other_product IS NOT NULL THEN
      Log_error_change
         (v_use_no, v_error_code, 'registration number',
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_revision_no||'-'||v_reg_firmno,
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_new_revision_no||'-'||v_new_reg_firmno,
          v_error_type, v_replace_type, v_require_type, v_loader_name,
          'prodno = '||v_prodno||', site_code = '||v_site_code||
          ': found on label for '||v_other_product||', not on label for '||
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_new_revision_no||'-'||v_new_reg_firmno);
   ELSE
      Log_error_change
         (v_use_no, v_error_code, 'registration number',
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_revision_no||'-'||v_reg_firmno,
          v_mfg_firmno||'-'||v_label_seq_no||'-'||v_new_revision_no||'-'||v_new_reg_firmno,
          v_error_type, v_replace_type, v_require_type, v_loader_name,
          'prodno = '||v_prodno||', site_code = '||v_site_code);
   END IF;
END IF;


   /*
   ELSE * v_error_type = 'N' which means here that a prodno was found.
        However, there could be probable errors in revision_no or
        reg_firmno.
        Even if there were probable errors, Prodno returns
        error_type = 'N' because we want to record errors and changes
        to either revision_no or reg_firmno.
      *
  * If revision_no was replaced, record as probable error.
   *
  IF v_revision_no <> v_new_revision_no OR
     (v_revision_no IS NULL AND v_new_revision_no IS NOT NULL) THEN
     Log_error_change
        (v_use_no, v_error_code, 'revision_no', rec.revision_no,
         v_new_revision_no, 'probable', 'estimate', v_require_type,
         v_loader_name, NULL);
  END IF;

  * If reg_firmno was replaced, record as probable error.
     However, do not record change if v_reg_firmno is NULL
     and v_new_reg_firmno = v_mfg_firmno because in the
     PUR reports from the county, reg_firmno is left blank
     or set = 0 if reg_fimrno = mfg_firmno.  In Reg_firmno()
     v_reg_firmno was set to null when reg_firmno was 0 or ' '.
  *
  IF v_reg_firmno <> v_new_reg_firmno OR
     (v_reg_firmno IS NULL AND v_new_reg_firmno <> v_mfg_firmno) THEN
     Log_error_change
        (v_use_no, v_error_code, 'reg_firmno', rec.reg_firmno,
         v_new_reg_firmno, 'probable', 'estimate', v_require_type,
         v_loader_name, NULL);
  END IF;

  *  Previously:
  IF v_revision_no <> v_new_revision_no OR
     (v_revision_no IS NULL AND v_new_revision_no IS NOT NULL) THEN
     Log_error_change
        (v_use_no, v_error_code, 'revision_no', rec.revision_no,
         v_new_revision_no, 'invalid', 'correct', v_require_type,
         v_loader_name, NULL);
  END IF;

  IF v_reg_firmno <> v_new_reg_firmno THEN
     Log_error_change
        (v_use_no, v_error_code, 'reg_firmno', rec.reg_firmno,
         v_new_reg_firmno, 'invalid', 'correct', v_require_type,
         v_loader_name, NULL);
  END IF;
  */



/* Error code 52: Spec_gravity: check if there is valid specific gravity
   value for prodno.  This is a check for DPR's label database, not
   for PUR data entry.
 */
Check_value2001.Spec_gravity
   (v_use_no, v_prodno, v_unit_of_meas, v_spec_gravity, v_formula_cd,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'spec_gravity', v_spec_gravity,
       v_spec_gravity, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 38: second unit_of_meas check
 */
Check_value2001.Unit_of_meas2
   (v_use_no, v_unit_of_meas, v_formula_cd,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'unit_of_meas', v_unit_of_meas,
       v_unit_of_meas, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* No Error code: Calculate lbs_prd_used.  If error occurs it must
   be an "other" error, error code = 0.
 */
Check_value2001.Lbs_prd_used
   (v_use_no, v_prodno, v_amt_prd_used, v_unit_of_meas, v_formula_cd,
    v_spec_gravity, v_lbs_prd_used,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'lbs_prd_used', v_lbs_prd_used,
       v_lbs_prd_used, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;
```

```
/* Error code 18: acre_treated
 */
Check_value2001.Acre_treated
   (v_use_no, v_record_id, v_site_code, rec.acre_treated, v_acre_treated,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'acre_treated', rec.acre_treated,
       v_acre_treated, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 19: unit_treated
 */
Check_value2001.Unit_treated
   (v_use_no, v_record_id, v_site_code, rec.unit_treated, v_unit_treated,
    v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   Log_error_change
      (v_use_no, v_error_code, 'unit_treated', rec.unit_treated,
       v_unit_treated, v_error_type, v_replace_type, v_require_type,
       v_loader_name, NULL);
END IF;

/* Error code 18, 19: acre_treated, unit_treated both present when optional
*/
Check_value2001.Acre_unit_treated
  (v_use_no, v_record_id, v_site_code, v_acre_treated, v_unit_treated,
   v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
  IF v_error_code = 18 THEN
     Log_error_change
        (v_use_no, v_error_code, 'acre_treated', rec.acre_treated,
         v_acre_treated, v_error_type, v_replace_type, v_require_type,
         v_loader_name, NULL);
  ELSIF v_error_code = 19 THEN
     Log_error_change
        (v_use_no, v_error_code, 'unit_treated', rec.unit_treated,
         v_unit_treated, v_error_type, v_replace_type, v_require_type,
         v_loader_name, NULL);
  END IF;
END IF;

/* Error code 23: acre_treated  > area of its section?
*/
v_acre_treated_old :=  v_acre_treated;
Check_value2001.Acre_treated_section
  (v_use_no, v_record_id, v_site_code, v_ca_mtrs_acres, v_acre_treated, v_unit_treated,
   v_grower_id, v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
  Log_error_change
     (v_use_no, v_error_code, 'acre_treated', v_acre_treated_old,
      v_acre_treated, v_error_type, v_replace_type, v_require_type,
      v_loader_name, NULL);
END IF;

/* Error code 22: acre_treated  > area of its section?
*/
v_unit_treated_old :=  v_unit_treated;
Check_value2001.Acre_treated_nonag
  (v_use_no, v_record_id, v_site_code, v_acre_treated, v_unit_treated,
   v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
  Log_error_change
     (v_use_no, v_error_code, 'unit_treated', v_unit_treated_old,
      v_unit_treated, v_error_type, v_replace_type, v_require_type,
      v_loader_name, NULL);
END IF;
```

```
/* Error code 44: acre_planted
*/
Check_value2001.Acre_planted
  (v_use_no, v_record_id, rec.acre_planted, v_acre_planted,
   v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
  Log_error_change
     (v_use_no, v_error_code, 'acre_planted', rec.acre_planted,
      v_acre_planted, v_error_type, v_replace_type, v_require_type,
      v_loader_name, NULL);
END IF;

/* Error code 45: unit_planted
*/
Check_value2001.Unit_planted
  (v_use_no, v_record_id, rec.unit_planted, v_unit_planted,
   v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
  Log_error_change
     (v_use_no, v_error_code, 'unit_planted', rec.unit_planted,
      v_unit_planted, v_error_type, v_replace_type, v_require_type,
      v_loader_name, NULL);
END IF;

/* Error code 60: acre_planted  > area of its section?
*/
v_acre_planted_old := v_acre_planted;
Check_value2001.Acre_planted_section
 (v_use_no, v_site_code, v_ca_mtrs_acres, v_acre_planted, v_unit_planted,
  v_grower_id, v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
 Log_error_change
    (v_use_no, v_error_code, 'acre_planted', v_acre_planted_old,
     v_acre_planted, v_error_type, v_replace_type, v_require_type,
     v_loader_name, NULL);
END IF;

/* Error code 47: acre_treated > acre_planted?
*/
v_acre_treated_old := v_acre_treated;

Check_value2001.Acre_treated_planted
 (v_use_no, v_unit_planted, v_unit_treated, v_acre_planted, v_acre_treated,
  v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
 Log_error_change
    (v_use_no, v_error_code, 'acre_treated', v_acre_treated_old,
     v_acre_treated, v_error_type, v_replace_type, v_require_type,
     v_loader_name, NULL);
END IF;

/* Error code 61: unit_treated and unit_planted consistent?
   Log_error_change is called twice, once to record error for
   unit_treated and once for unit_planted.
*/
v_unit_treated_old := v_unit_treated;
v_unit_planted_old := v_unit_planted;

Check_value2001.Unit_treated_planted
 (v_use_no, v_acre_treated, v_acre_planted, v_unit_treated, v_unit_planted,
  v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
 Log_error_change
    (v_use_no, v_error_code, 'unit_treated', v_unit_treated_old,
     v_unit_treated, v_error_type, v_replace_type, v_require_type,
     v_loader_name, NULL);
 IF v_replace_type != 'estimate' THEN
    Log_error_change
       (v_use_no, v_error_code, 'unit_planted', v_unit_planted_old,
        v_unit_planted, v_error_type, v_replace_type, v_require_type,
        v_loader_name, NULL);
 END IF;
END IF;


/* Error code 30: applic_cnt
*/
Check_value2001.Applic_cnt
 (v_use_no, v_record_id, rec.applic_cnt, v_applic_cnt,
  v_error_code, v_error_type, v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
 Log_error_change
    (v_use_no, v_error_code, 'applic_cnt', rec.applic_cnt,
     v_applic_cnt, v_error_type, v_replace_type, v_require_type,
     v_loader_name, NULL);
END IF;


/* Error code 80: Check for duplicate records that appear to be
   errors.
   Here, only one record from a error duplicate set is recorded
   in the pur table, though all duplicates from the raw table
   are flagged in the raw table.
   Thus, if current record is a duplicate of another record
   already in pur, do not insert the current record into pur.
 */
Check_value2001.Duplicates
 (v_use_no, v_record_id, v_county_cd, v_grower_id, v_site_loc_id,
  v_site_code, v_qualify_cd, v_prodno, v_amt_prd_used, v_unit_of_meas,
  v_acre_treated, v_unit_treated, v_acre_planted, v_unit_planted,
  v_applic_dt, v_use_no_array, v_error_code, v_error_type,
  v_replace_type, v_require_type);
IF v_error_type <> 'N' THEN
   v_error_duplicate := TRUE;
   Log_duplicates
    (v_use_no, v_use_no_array, v_error_code,
     v_error_type, v_replace_type, v_require_type,
     v_loader_name, NULL);
ELSE
   v_error_duplicate := FALSE;
END IF;

IF NOT v_error_duplicate THEN

/* Error code 63: inconsistent site_loc_id
   This procedure determines if there are inconsistent values for
   mtrs or acres planted reported for this agricultural field.
   Should be called after error checks for all values used and
   after duplicate check (so duplicates do not get added in
   num_rec field of the field_mtrs_acres table.
 */
   /* Inconsistent_site_loc_id is not called because it slowed
      loading too much, and because it was not called
      during error corrections, and it really needs to be
      improved.
      It can also be run after loading.
   Check_value2001.Inconsistent_site_loc_id
      (v_use_no, v_record_id, v_county_cd, v_grower_id, v_site_loc_id,
       v_site_code, v_acre_planted, v_unit_planted, v_base_ln_mer,
       v_township, v_tship_dir, v_range, v_range_dir, v_section,
       v_field_id, v_site_loc_id_state, v_operator_id,
       v_this_mtrs, v_this_acres, v_previous_mtrs, v_previous_acres,
       v_error_code, v_error_type, v_replace_type, v_require_type);

   Update_fields(v_use_no, v_county_cd, v_operator_id, v_site_loc_id, v_site_code,
                 v_site_loc_id_state, v_this_mtrs, v_this_acres,
                 v_previous_mtrs, v_previous_acres, v_error_code,
                 v_field_id);

   IF v_error_type <> 'N' THEN
      Log_error_change
         (v_use_no, v_error_code, 'site_loc_id', v_site_loc_id,
          v_site_loc_id, v_error_type, v_replace_type, v_require_type,
          v_loader_name, 'field_id = '||v_field_id);
   END IF;
   */
   v_field_id := NULL;
   IF v_record_id IN ('1','4','A','B') THEN
      v_this_mtrs :=  mtrs_type(v_base_ln_mer, v_township, v_tship_dir,
                                v_range, v_range_dir, v_section);
   ELSE
      v_this_mtrs := NULL;
   END IF;

   /* Error code 75: outlier flagged by criteria 1 or 2
   */
   v_unit_treated_old := v_unit_treated;
   v_acre_treated_old := v_acre_treated;
   v_lbs_prd_used_old := v_lbs_prd_used;
   v_amt_prd_used_old := v_amt_prd_used;

   Check_value2001.Outliers
    (v_use_no, v_record_id, v_prodno, v_site_code, v_amt_prd_used,
     v_lbs_prd_used, v_acre_treated, v_unit_treated, v_acre_planted,
     v_unit_planted, v_estimated_field, v_criteria,
     v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      IF v_estimated_field = 'unit_treated' THEN
       Log_error_change
          (v_use_no, v_error_code, v_estimated_field, v_unit_treated_old,
           v_unit_treated, v_error_type, v_replace_type, v_require_type,
           v_loader_name, v_criteria);
      ELSIF v_estimated_field = 'acre_treated' THEN
       Log_error_change
          (v_use_no, v_error_code, v_estimated_field, v_acre_treated_old,
           v_acre_treated, v_error_type, v_replace_type, v_require_type,
           v_loader_name, v_criteria);
      ELSIF v_estimated_field = 'lbs_prd_used' THEN
       Log_error_change
          (v_use_no, v_error_code, v_estimated_field, v_lbs_prd_used_old,
           v_lbs_prd_used, v_error_type, v_replace_type, v_require_type,
           v_loader_name, v_criteria);
       Log_error_change
          (v_use_no, v_error_code, 'amt_prd_used', v_amt_prd_used_old,
           v_amt_prd_used, v_error_type, v_replace_type, v_require_type,
           v_loader_name, v_criteria);
      END IF;
   END IF;

   /* Error code 72: rate of use > neural network outlier value.
    I suggest to call this after Outliers() to flag only records
    not flagged by other criteria.
   */
   Check_value2001.Outliers_nn
    (v_use_no, v_record_id, v_prodno, v_site_code, v_lbs_prd_used,
     v_acre_treated, v_unit_treated,
     v_error_code, v_error_type, v_replace_type, v_require_type);
   IF v_error_type <> 'N' THEN
      Log_error_change
       (v_use_no, v_error_code, 'rate_of_use', v_lbs_prd_used/v_acre_treated,
        v_lbs_prd_used/v_acre_treated,
        v_error_type, v_replace_type, v_require_type,
        v_loader_name, NULL);
   END IF;

END IF; --v_error_duplicate
```

```
         /*************************
         For normal PUR processing use:
         */
         UPDATE   raw2001i
            SET   use_no = v_use_no
            WHERE CURRENT OF record_cur;

         /*************************
         For batch PUR processing comment out above update.
         */

         IF NOT v_error_duplicate THEN
            INSERT INTO
            pur2001(use_no, record_id, batch_no, process_mt, process_yr,
                       county_cd, site_code, applic_dt, document_no, summary_cd,
                       site_loc_id, cedts_ind, qualify_cd, amt_prd_used, unit_of_meas,
                       prodno, lbs_prd_used, acre_treated, unit_treated,
                       acre_planted, unit_planted, applic_cnt,
                       planting_seq, section, township, tship_dir, range, range_dir,
                       base_ln_mer, aer_gnd_ind, applic_time,
                       grower_id, license_no, last_up_dt, field_id, mtrs)
            VALUES(v_use_no, v_record_id, v_batch_no, v_process_mt, v_process_yr,
                   v_county_cd, v_site_code, v_applic_dt, v_document_no, v_summary_cd,
                   v_site_loc_id, v_cedts_ind, v_qualify_cd, v_amt_prd_used, v_unit_of_meas,
                   v_prodno, v_lbs_prd_used, v_acre_treated, v_unit_treated,
                   v_acre_planted, v_unit_planted, v_applic_cnt,
                   v_planting_seq, v_section, v_township, v_tship_dir, v_range, v_range_dir,
                   v_base_ln_mer, v_aer_gnd_ind, v_applic_time,
                   v_grower_id, v_license_no, SYSDATE, v_field_id, v_this_mtrs);
            --COMMIT;
         END IF;

      END LOOP;
      COMMIT;

   EXCEPTION
      WHEN OTHERS THEN
         General_exceptions;
   END Check_records;


/* Record information in the errors and/or changes table.
 */
   PROCEDURE Log_error_change
      (p_use_no IN NUMBER, p_error_code IN BINARY_INTEGER,
       p_column_name IN VARCHAR2, p_old_value IN VARCHAR2,
       p_new_value IN VARCHAR2, p_error_type IN VARCHAR2,
       p_replace_type IN VARCHAR2, p_require_type IN VARCHAR2,
       p_who IN VARCHAR2, p_comments IN VARCHAR2)
   AS
   BEGIN

      /* p_replace_type can be null, estimate, same, or correct
       */

      /*************************
       For normal PUR processing use:  errors2001i
       */
      IF p_replace_type != 'correct' THEN
         INSERT INTO errors2001i
            (transaction_no, use_no, error_code, column_name,
             old_value, new_value, error_type, replace_type,
             require_type, who, comments)
         VALUES
            (errors_seq2001.NextVal, p_use_no, p_error_code, p_column_name,
             p_old_value, p_new_value, p_error_type, p_replace_type,
             p_require_type, p_who, p_comments);
         --COMMIT;
      END IF;
      /*************************
       For normal PUR processing use:  changes2001i
       */
      IF p_replace_type != 'same' THEN
         INSERT INTO changes2001i
            (transaction_no, use_no, error_code,
             column_name, old_value, new_value, action_taken,
             action_time, who, county_validated, comments)
         VALUES
            (changes_seq2001.NextVal, p_use_no, p_error_code,
             p_column_name, p_old_value, p_new_value, p_replace_type,
             SYSDATE, p_who, 'N', p_comments);
         --COMMIT;
      END IF;

      /*************************
       For batch PUR processing use:  errors2001 and changes2001 in above code.
       */
   EXCEPTION
      WHEN OTHERS THEN
         Other_exceptions(p_use_no, p_error_code);
   END Log_error_change;


/* Record information about error duplicates.  Record both in errors and changes.
   We record in changes because all but one duplicate records from raw are
   not recorded in the PUR.
 */
   PROCEDURE Log_duplicates
      (p_use_no IN NUMBER, p_use_no_array IN Check_value2001.use_no_array_type,
       p_error_code IN BINARY_INTEGER, p_error_type IN VARCHAR2,
       p_replace_type IN VARCHAR2, p_require_type IN VARCHAR2,
       p_who IN VARCHAR2, p_comments IN VARCHAR2)
   IS
      v_new_duplicate_set_no  BINARY_INTEGER;
      v_old_duplicate_set_no  BINARY_INTEGER;
      v_new_duplicate_set     BOOLEAN := FALSE;
      v_dup_no                BINARY_INTEGER;
   BEGIN
      /* First, check if there is already a duplicate set which
         the current record belongs to.
       */
      /******************************
         For normal PUR processing use both errors2001 and errors2001i;
         For batch processing use only errors2001.
       */
      BEGIN
         SELECT duplicate_set
         INTO   v_old_duplicate_set_no
         FROM   errors2001
         WHERE  use_no = p_use_no_array(1) AND
                error_code = p_error_code;
      EXCEPTION
         WHEN OTHERS THEN
            v_new_duplicate_set := TRUE;
      END;

      IF v_new_duplicate_set THEN
         BEGIN
            SELECT duplicate_set
            INTO   v_old_duplicate_set_no
            FROM   errors2001i
            WHERE  use_no = p_use_no_array(1) AND
                   error_code = p_error_code;

            v_new_duplicate_set := FALSE;
         EXCEPTION
            WHEN OTHERS THEN
               v_new_duplicate_set := TRUE;
         END;
      END IF;
```

```
/* If this record is not in a previous duplicate set,
   make an error record using a new duplicate set number.
   Otherwise, make an error record using the previous
   duplicate set's number.
 */
IF v_new_duplicate_set THEN
   SELECT dup_set_seq2001.NEXTVAL
   INTO   v_new_duplicate_set_no
   FROM   dual;

   /* Record error for each record in the duplicate set; the last record
      in the array is the current array and this record is not inserted
      in the PUR but the other records are--note that replace_type is set to 'same'.
    */

   /*************************
    For normal PUR processing use:  errors2001i and changes2001i in all inserts below;
    for batch processing use errors2001 and changes2001.
    */
   FOR v_dup_no IN 1..(p_use_no_array.LAST - 1) LOOP
      INSERT INTO errors2001i
         (transaction_no, use_no, error_code,
          column_name, old_value, new_value, error_type, replace_type,
          require_type, who, duplicate_set, comments)
      VALUES
         (errors_seq2001.NextVal, p_use_no_array(v_dup_no), p_error_code,
          NULL, NULL, NULL, p_error_type, 'same',
          p_require_type, p_who, v_new_duplicate_set_no, p_comments);
      --COMMIT;
   END LOOP;

   /* Only the current record is not recorded in the PUR,
      so report this in ERRORS table (note that p_replace_type 'delete') and
      as a change from what is in the RAW table.
      (Note that p_use_no is the same as p_use_no_array(p_use_no_array.LAST))
    */
   INSERT INTO errors2001i
      (transaction_no, use_no, error_code,
       column_name, old_value, new_value, error_type, replace_type,
       require_type, who, duplicate_set, comments)
   VALUES
      (errors_seq2001.NextVal, p_use_no, p_error_code,
       NULL, NULL, NULL, p_error_type, p_replace_type,
       p_require_type, p_who, v_new_duplicate_set_no, p_comments);
   --COMMIT;

   INSERT INTO changes2001i
      (transaction_no, use_no, error_code,
       column_name, old_value, new_value, action_taken,
       action_time, who, county_validated, comments)
   VALUES
      (changes_seq2001.NextVal, p_use_no, p_error_code,
       NULL, NULL, NULL, p_replace_type,
       SYSDATE, p_who, 'N', p_comments);
   --COMMIT;
ELSE
   INSERT INTO errors2001i
      (transaction_no, use_no, error_code, column_name,
       old_value, new_value, error_type, replace_type,
       require_type, who, duplicate_set, comments)
   VALUES
      (errors_seq2001.NextVal, p_use_no, p_error_code, NULL,
       NULL, NULL, p_error_type, p_replace_type,
       p_require_type, p_who, v_old_duplicate_set_no, p_comments);
   --COMMIT;

   INSERT INTO changes2001i
      (transaction_no, use_no, error_code,
       column_name, old_value, new_value, action_taken,
       action_time, who, county_validated, comments)
      VALUES
         (changes_seq2001.NextVal, p_use_no, p_error_code,
          NULL, NULL, NULL, p_replace_type,
          SYSDATE, p_who, 'N', p_comments);
      --COMMIT;
   END IF;

EXCEPTION
   WHEN OTHERS THEN
      Other_exceptions(p_use_no, p_error_code);
END Log_duplicates;


/* This procedure is called after Inconsistent_site_loc_id()
   which determines if there are inconsistent values for mtrs or acres planted reported
   for this agricultural field.
   This procedure updates the table fields2001 with the information found.
   There are 4 possible outcomes from Inconsistent_site_loc_id():
      1. This is the first report of an application to this field;
      2. There were other reports for this field, all consistent, and
         this one is consistent as well
      3. This is the first inconsistent report for this field.
      4. Previous records were inconsistent.

   If this is not the first report for this field, p_field_id will have the field_id
   if for this field.  If it is the first report, a new field_id is generated
   and returned to the calling function.
 */
PROCEDURE Update_fields
   (p_use_no IN NUMBER,
    p_county_cd IN VARCHAR2,
    p_operator_id IN VARCHAR2,
    p_site_loc_id IN VARCHAR2,
    p_site_code IN NUMBER,
    p_site_loc_id_state IN VARCHAR2,
    p_this_mtrs IN mtrs_type,
    p_this_acres IN NUMBER,
    p_previous_mtrs IN mtrs_type,
    p_previous_acres IN NUMBER,
    p_error_code IN BINARY_INTEGER,
    p_field_id IN OUT NUMBER)
IS
   v_region             VARCHAR2(20);
   v_best_mtrs          mtrs_type;
   v_best_acres         NUMBER(10, 2);
   v_num_recs           NUMBER(10);

   v_highest_mtrs_num_recs   INTEGER;
   v_highest_acres_num_recs  INTEGER;

   v_require_type       VARCHAR2(20);

   CURSOR pur_cur(fid IN NUMBER) IS
      SELECT   use_no, site_loc_id, record_id
      FROM     pur2001
      WHERE    field_id = fid;

   CURSOR field_cur(fid IN NUMBER) IS
      SELECT   mtrs, acres_planted, num_recs
      FROM     field_mtrs_acres2001
      WHERE    field_id = fid;

BEGIN

   IF p_site_loc_id_state = 'first_record' THEN
      /* This is first record for this ag field.
         Add new record to fields2001 with values for this field.
        */
      BEGIN
         SELECT   region
         INTO     v_region
      FROM     co_regions
      WHERE    county_cd = p_county_cd;
   EXCEPTION
      WHEN OTHERS THEN
         v_region := 'UNKNOWN';
   END;

   SELECT   field_id_seq2001.NEXTVAL
   INTO     p_field_id
   FROM     dual;

   INSERT INTO fields2001 VALUES
      (p_field_id, p_county_cd, p_operator_id,
       p_site_loc_id, p_site_code, p_this_mtrs, p_this_acres, v_region, NULL);
   --COMMIT;

ELSIF p_site_loc_id_state = 'first_inconsistent' THEN
   /* Found first inconsistent mtrs.
      Determine best estimate for mtrs and acres
      planted, add new record to field_mtrs_acres table,
      mark as inconsistent_mtrs.

      Need to report all use_no with this field_id as error 63.
    */

   /* Determine which mtrs value is most likely correct
    */
   IF p_previous_mtrs.Has_missing_info AND
      NOT p_this_mtrs.Has_missing_info
   THEN
      v_best_mtrs := p_this_mtrs;
   ELSE
      v_best_mtrs := p_previous_mtrs;
   END IF;

   /* Determine which acre value is most likely correct
    */
   IF p_previous_acres IS NULL AND
      p_this_acres IS NOT NULL
   THEN
      v_best_acres := p_this_acres;
   ELSE
      v_best_acres := p_previous_acres;
   END IF;

   UPDATE fields2001
      SET inconsistent_field = 'X',
          mtrs = v_best_mtrs,
          acres_planted = v_best_acres
      WHERE field_id = p_field_id;
   --COMMIT;

   /* Mark other records in the PUR with this ag field as having an
      inconsistent field.
    */
   v_num_recs := 0;
   FOR pur_rec IN pur_cur(p_field_id) LOOP
      -- Only required records should have field_id, but just in case check...
      IF pur_rec.record_id IN ('1','4','A','B') THEN
         v_require_type := 'required';
      ELSIF pur_rec.record_id IN ('2','C') THEN
         v_require_type := 'not_allowed';
      ELSIF pur_rec.record_id IS NULL THEN
         v_require_type := 'unknown';
      ELSE
         v_require_type := 'unknown';
      END IF;

      v_num_recs := v_num_recs + 1;

      Log_error_change
```

```
         (pur_rec.use_no, p_error_code, 'site_loc_id', pur_rec.site_loc_id,
          pur_rec.site_loc_id, 'possible', 'same', v_require_type,
          v_loader_name, 'field_id = '||p_field_id);
   END LOOP;

   /* Record the inconsistent values and number of records of each
      in field_mtrs_acres2001.
    */
   INSERT INTO field_mtrs_acres2001 VALUES
      (p_field_id, p_previous_mtrs, p_previous_acres, v_num_recs);

   INSERT INTO field_mtrs_acres2001 VALUES
      (p_field_id, p_this_mtrs, p_this_acres, 1);
   --COMMIT;

ELSIF p_site_loc_id_state = 'previous_inconsistent' THEN
   /* Check if this combination of mtrs and acres is already in the
      list of values for this field.
    */

   /* This will increment num_recs if previous records exists
      with this mtrs and acres.
    */
   UPDATE   field_mtrs_acres2001
   SET      num_recs = num_recs + 1
   WHERE    field_id = p_field_id AND
            mtrs = p_this_mtrs AND
            acres_planted = p_this_acres;

   /* If no records exist with this mtrs and acres, add new row.
    */
   IF SQL%NOTFOUND THEN
      INSERT INTO field_mtrs_acres2001 VALUES
         (p_field_id, p_this_mtrs, p_this_acres, 1);
      --COMMIT;
   END IF;

   v_highest_mtrs_num_recs := 1;
   v_best_mtrs := p_previous_mtrs;

   v_highest_acres_num_recs := 1;
   v_best_acres := p_previous_acres;

   FOR field_rec IN field_cur(p_field_id) LOOP
      IF NOT field_rec.mtrs.Has_missing_info THEN
         IF field_rec.num_recs > v_highest_mtrs_num_recs THEN
            v_highest_mtrs_num_recs := field_rec.num_recs;
            v_best_mtrs := field_rec.mtrs;
         END IF;
      END IF;

      IF field_rec.acres_planted IS NOT NULL THEN
         IF field_rec.num_recs > v_highest_acres_num_recs THEN
            v_highest_acres_num_recs := field_rec.num_recs;
            v_best_acres := field_rec.acres_planted;
         END IF;
      END IF;
   END LOOP;

   IF v_best_mtrs != p_previous_mtrs AND
      v_best_acres != p_previous_acres
   THEN
      UPDATE   fields2001
      SET      mtrs = v_best_mtrs,
               acres_planted = v_best_acres
      WHERE    field_id = p_field_id;
   ELSIF v_best_mtrs != p_previous_mtrs THEN
      UPDATE   fields2001
      SET      mtrs = v_best_mtrs
      WHERE    field_id = p_field_id;
   ELSIF v_best_acres != p_previous_acres THEN
```

```
            UPDATE   fields2001
            SET      acres_planted = v_best_acres
            WHERE    field_id = p_field_id;
         END IF;
         --COMMIT;

      ELSIF p_site_loc_id_state = 'not_allowed' OR
            p_site_loc_id_state = 'still_consistent'
      THEN
         NULL;
      ELSE
         Other_exceptions(p_use_no, p_error_code);
      END IF;

   EXCEPTION
      WHEN OTHERS THEN
         Other_exceptions(p_use_no, p_error_code);
   END Update_fields;


/* Print out error message...
 */
   PROCEDURE Other_exceptions(p_use_no IN NUMBER, p_error_code IN BINARY_INTEGER)
   IS
   BEGIN
      DBMS_OUTPUT.PUT_LINE('Co_error2001: Other error for error code '||p_error_code||
       ' and use_no '||p_use_no||  ': ' || SQLERRM);
   END;

/* Print out error message...
 */
   PROCEDURE General_exceptions
   IS
   BEGIN
      DBMS_OUTPUT.PUT_LINE('Other Error: ' || SQLERRM);
   END;

END Co_error2001;
/
show errors

EXIT 0
```

Check_errors.sql

```
/* Run the error checking procedures
 */

SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET document off
SET serveroutput ON
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

EXECUTE Co_error2001.Check_records;
show errors

EXIT 0
```

Clear_intermediates.sql

```
/* Delete all the records from the intermediate tables
   raw2001i,  errors2001i, and changes2001i.
 */

SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET serveroutput ON
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

VARIABLE v_error NUMBER

BEGIN
   :v_error := 0;

   EXECUTE IMMEDIATE 'TRUNCATE TABLE raw2001i';
   EXECUTE IMMEDIATE 'TRUNCATE TABLE errors2001i';
   EXECUTE IMMEDIATE 'TRUNCATE TABLE changes2001i';

EXCEPTION
   WHEN OTHERS THEN
      :v_error := 1;
      DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
END;
/

EXIT :v_error
```

Dates.sql

```
SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
set serveroutput on size 1000000 format word_wrapped

WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

VARIABLE v_result NUMBER

DECLARE
   v_package_date    DATE;
   v_file_date       DATE;
   v_file_name       VARCHAR2(30);
BEGIN
   v_file_name := RTRIM(LOWER('&1'), '.sql');

   SELECT   last_ddl_time
   INTO     v_package_date
   FROM     user_objects
   WHERE    object_name = UPPER(v_file_name) AND
            object_type = 'PACKAGE BODY';

   v_file_date := TO_DATE('&2', 'DD-MON-YYYY_HH24:MI:SS');

   IF v_file_date > v_package_date THEN
      --DBMS_OUTPUT.PUT_LINE('Package '||v_file_name||' needs to be recompiled.');
      :v_result := 2;
   ELSE
      --DBMS_OUTPUT.PUT_LINE('Package '||v_file_name||' is up to date.');
      :v_result := 0;
   END IF;

EXCEPTION
   WHEN NO_DATA_FOUND THEN
      --DBMS_OUTPUT.PUT_LINE('Package '||v_file_name||' has not yet been compiled.');
      :v_result := 2;
   WHEN OTHERS THEN
      :v_result := 1;
      DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
END;
/

EXIT :v_result
```

Error_report.sql

```
/* Prints error reports that are sent to counties
 * Before you can the error report, this script fills the table use_error_report2001i.
 * This is a table of each use_no from errors2001 and
 * a string of all the error_codes for that use_no,
 * except for errors with error_codes 63 and 72.
 *
 * In order for this to print correctly, need to change settings on an HP printer.
 * In Printing Menu, use the following setting:
 *    paper = letter
 *    configure custom paper = no
 *    form = 45 lines
 *    orientation = landscape
 *    PCL font source = internal
 *    PCL font number = 0
 *    PCL font pitch = 14.00
 *    PCL symbol set = PC-8
 *    courier = regulatr
 *    wide A4 = no
 *    append CR to LF = no
 *
 *
 * To find default printer, and status of all printers, type: "lpstat -t".
 *
 * To change or set your default printer to, say, bashful, add to your .profile:
 * LPDEST=bashful export LPDEST
 *
 * To print on Enforcement's printer, use printer hp_phill
 * (which is in Phil Wilson's cubicle)
 * To print all county files to Enforcement's printer
 * without changing default printer, type:
 *
 * lp -d hp_phill county*
 *
 * Print out CalTrans records in separate file, not with the county files
 */

SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET document off
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

DECLARE
   v_all_error_codes VARCHAR2(200);
   v_prev_error_code VARCHAR2(10);
   v_first_error     BOOLEAN;

        /* Get each use_no.
         */
        CURSOR use_cur IS
               SELECT  DISTINCT use_no
               FROM           errors2001i
      WHERE    error_type != 'possible';

        /* Get all error_codes for this use_no.
         */
        CURSOR errors_cur(u_no IN NUMBER) IS
               SELECT  error_code
               FROM           errors2001i
      WHERE    use_no = u_no AND
               error_type != 'possible'
      ORDER BY error_code;
BEGIN
        FOR use_rec IN use_cur LOOP
      v_all_error_codes := '';
      v_first_error := TRUE;
      FOR errors_rec IN errors_cur(use_rec.use_no) LOOP
```

```
         IF v_first_error THEN
            v_all_error_codes := errors_rec.error_code;
            v_first_error := FALSE;
         ELSIF errors_rec.error_code IS NULL OR
            errors_rec.error_code = v_prev_error_code
         THEN
            NULL;
         ELSE
            v_all_error_codes :=
               v_all_error_codes||', '||errors_rec.error_code;
         END IF;

         v_prev_error_code := errors_rec.error_code;

      END LOOP;

      INSERT INTO use_error_report2001i VALUES
         (use_rec.use_no, v_all_error_codes);
      COMMIT;

   END LOOP;
EXCEPTION
       WHEN OTHERS THEN
               DECLARE
                       error_msg      VARCHAR2(300) := SQLERRM;
         error_code  NUMBER := SQLCODE;
               BEGIN
                       DBMS_OUTPUT.PUT_LINE('Error Number: '||error_code||': ' || error_msg);
               END;
END;
/
show errors

/* Now create the county error report files.
 */
SET newpage 0
SET pagesize 43
SET linesize 144
SET underline off
SET feedback Off
SET termout Off
SET verify off
SET pause off

COLUMN current_date NEW_VALUE report_date NOPRINT
SELECT TO_CHAR(SYSDATE, 'DD-MON-YYYY') current_date FROM dual;

COLUMN county_name NEW_VALUE report_county NOPRINT
SELECT INITCAP(LOWER(coname)) county_name FROM county WHERE county_cd = substr('&1', 2, 2);

COLUMN grower_id_title HEADING ''
COLUMN grower_no HEADING ''
COLUMN license_no_title HEADING ''
COLUMN license_no HEADING ''
COLUMN site_loc_id_title HEADING ''
COLUMN site_loc_id HEADING ''
COLUMN batch_no_title HEADING ''
COLUMN batch_no HEADING ''
COLUMN document_no_title HEADING ''
COLUMN document_no HEADING ''
COLUMN summary_cd_title HEADING ''
COLUMN summary_cd HEADING '' FOLD_AFTER

COLUMN u_no HEADING 'USE NO|--------'
COLUMN record_id HEADING 'R|C| |I|D|-'
COLUMN process_mt HEADING 'PR|MT|--'
COLUMN process_yr HEADING 'PR|YR|--'
COLUMN report_month HEADING 'RP|MT|--'
COLUMN report_year HEADING 'RP|YR|--'
COLUMN nursery_ind HEADING 'N|U|R|S|Y|-'
COLUMN county_cd HEADING 'CT|CD|--'
COLUMN section HEADING 'S|E|C|T|N|--'
COLUMN township HEADING 'T|W|N|S|P|--'
COLUMN tship_dir HEADING 'T|W|N|D|R|-'
COLUMN range HEADING 'R|A|N|G|E|--'
COLUMN range_dir HEADING 'R|N|G|D|R|-'
COLUMN base_ln_mer HEADING 'B|S| |M|R|--'
COLUMN aer_gnd_ind HEADING 'A|R| |G|D|-'
COLUMN acre_planted/100 FORMAT 999999.00 HEADING 'ACRE|PLANTED|----------'
COLUMN unit_planted HEADING 'U|N|T| |P|-'
COLUMN applic_dt HEADING 'APPL|DATE|------'
COLUMN applic_time HEADING 'APPL|TIME|------'
COLUMN site_code HEADING 'SITE|CODE|------'
COLUMN qualify_cd HEADING 'QL|CD|--'
COLUMN planting_seq HEADING 'P| |S|E|Q|-'
COLUMN acre_treated/100 FORMAT 999999.00 HEADING 'ACRE|TREATED|----------'
COLUMN unit_treated HEADING 'U|N|T| |T|-'
COLUMN mfg_firmno HEADING 'MFG|FIRM|NO|-------'
COLUMN label_seq_no HEADING 'LABEL|SEQ|NO|-----'
COLUMN revision_no HEADING 'RV|NO|--'
COLUMN reg_firmno HEADING 'REG|FIRM|NO|-------'
COLUMN amt_prd_used/10000 FORMAT 999999.0000 HEADING 'AMOUNT|USED|------------'
COLUMN unit_of_meas HEADING 'U|O|M|--'
COLUMN applic_cnt HEADING 'APPLIC|COUNT|------' FOLD_AFTER

COLUMN error_type HEADING ''
COLUMN all_error_codes HEADING '' FORMAT A100

BREAK ON grower_no SKIP 4 DUPLICATES

spo  /home/purload/pur2001/reports/error_report_&1._2001.txt
/* Print a title page
 */
SET newpage none
SET heading off
TTITLE   LEFT '' SKIP 3 -
         CENTER 'Department of Pesticide Regulation' SKIP 1 -
         CENTER '2001 Pesticide Use Report' SKIP 1 -
         CENTER 'Error Reports for ' report_county ' County ' SKIP 2 -
         CENTER 'Reports created on ' report_date

SELECT NULL FROM dual;

/* Print data
 */
SET newpage 0
SET heading ON
TTITLE   LEFT 'Page ' FORMAT 999 SQL.PNO SKIP 1 -
         LEFT report_date -
         CENTER 'Department of Pesticide Regulation' -
         RIGHT 'County: ' report_county SKIP 1 -
         CENTER '2001 Pesticide Use Report' -
         RIGHT 'Report: &1' SKIP 1 -
         CENTER 'Batch Error Report' SKIP 1 -
         LEFT '                          --- LOCATION ----                                       -
-- REGISTRATION NUM ---'

SELECT   '    GROWER ID:' grower_id_title,
            SUBSTR(grower_id, 5, 2) ||' '|| SUBSTR(grower_id, 7, 5) grower_no,
         '    LICENSE NO:' license_no_title, license_no,
         '  SITE:' site_loc_id_title, site_loc_id,
         '  BATCH NO:' batch_no_title, batch_no,
         '  DOCUMENT NO:' document_no_title, document_no,
         '  SUMMARY CODE:' summary_cd_title, summary_cd,
         praw.use_no u_no, record_id, process_mt, process_yr,
         SUBSTR(applic_dt, 1, 2) report_month, SUBSTR(applic_dt, 5, 2) report_year,
         nursery_ind, county_cd, section, township, tship_dir,
         range, range_dir, base_ln_mer, aer_gnd_ind,
         acre_planted/100, unit_planted, applic_dt, applic_time, site_code,
         qualify_cd, planting_seq, acre_treated/100, unit_treated,
         mfg_firmno, label_seq_no, revision_no, reg_firmno,
         amt_prd_used/10000, unit_of_meas, applic_cnt,
         '    -Error Type' error_type, all_error_codes
FROM     raw2001i praw, use_error_report2001i err
WHERE    praw.use_no = err.use_no
ORDER BY grower_no, site_loc_id, mfg_firmno, label_seq_no;


SET linesize 100
SET underline ON

COLUMN mfg_firmno clear
COLUMN label_seq_no clear
COLUMN revision_no clear
COLUMN reg_firmno clear
COLUMN site_code clear

COLUMN mfg_firmno format a10
COLUMN label_seq_no format a12
COLUMN revision_no format a11
COLUMN reg_firmno format a10
COLUMN site_code format a10
COLUMN num_recs heading 'No. Records' format 999,999

BREAK ON mfg_firmno NODUPLICATES -
      ON label_seq_no NODUPLICATES -
      ON revision_no NODUPLICATES -
      ON reg_firmno NODUPLICATES

TTITLE   LEFT 'Page ' FORMAT 999 SQL.PNO SKIP 1 -
         LEFT report_date -
         CENTER 'Department of Pesticide Regulation' -
         RIGHT 'County: ' report_county SKIP 1 -
         CENTER '2001 Pesticide Use Report' -
         RIGHT 'Report: &1' SKIP 1 -
         CENTER 'Summary Report of site_code Errors For Pesticide Products' SKIP 2

SELECT   mfg_firmno, label_seq_no, revision_no, reg_firmno, site_code,
         count(*) num_recs
FROM     raw2001i praw, errors2001i err
WHERE    praw.use_no = err.use_no AND
         error_code = 39 AND
         error_type != 'possible'
GROUP BY mfg_firmno, label_seq_no, revision_no, reg_firmno, site_code;

SET linesize 120
SET underline ON


COLUMN err_code HEADING 'ERROR|CODE' format 99999
COLUMN error_desc HEADING 'ERROR DESCRIPTION' format A90
COLUMN no_errors HEADING 'NO. OF|ERRORS' format 99,999

BREAK ON REPORT
CLEAR COMPUTES
COMPUTE SUM LABEL 'TOTAL NUMBER OF ERRORS' OF no_errors ON REPORT

TTITLE   LEFT 'Page ' FORMAT 999 SQL.PNO SKIP 1 -
         LEFT report_date -
         CENTER 'Department of Pesticide Regulation' -
         RIGHT 'County: ' report_county SKIP 1 -
         CENTER '2001 Pesticide Use Report' -
         RIGHT 'Report: &1' SKIP 1 -
         CENTER 'Error Summary Report' SKIP 2

SELECT   DECODE(err.error_code, 70, 75, 71, 75, err.error_code) err_code,
         RPAD(description, 90, '.') error_desc,
         COUNT(DISTINCT praw.use_no) no_errors
FROM     raw2001i praw, errors2001i err, error_code_descriptions ec
WHERE    praw.use_no = err.use_no AND
         DECODE(err.error_code, 70, 75, 71, 75, err.error_code) = ec.error_code AND
         err.error_type != 'possible'
```

```
GROUP BY DECODE(err.error_code, 70, 75, 71, 75, err.error_code), description;

SET linesize 100
SET underline ON

COLUMN record_id clear
COLUMN record_id HEADING 'RECORD|TYPE' format a6
COLUMN invalid_recs HEADING 'TOTAL INVALID|RECORDS' format 9,999,999
COLUMN valid_recs HEADING 'TOTAL VALID|RECORDS' format 9,999,999
COLUMN invalid_values HEADING 'TOTAL NUM|ERRORS' format 9,999,999

BREAK ON REPORT
CLEAR COMPUTES
COMPUTE SUM LABEL 'TOTAL' OF invalid_recs ON REPORT
COMPUTE SUM LABEL 'TOTAL' OF valid_recs ON REPORT
COMPUTE SUM LABEL 'TOTAL' OF invalid_values ON REPORT

TTITLE   LEFT 'Page ' FORMAT 999 SQL.PNO SKIP 1 -
         LEFT report_date -
         CENTER 'Department of Pesticide Regulation' -
         RIGHT 'County: ' report_county SKIP 1 -
         CENTER '2001 Pesticide Use Report' -
         RIGHT 'Report: &1' SKIP 1 -
         CENTER 'Number of Error Records Summary Report' SKIP 2

SELECT   record_id, COUNT(*) invalid_recs
FROM     raw2001i praw
WHERE    EXISTS
         (SELECT *
          FROM    errors2001i
          WHERE   praw.use_no = use_no AND
                  error_type != 'possible')
GROUP BY record_id;

TTITLE   LEFT 'Page ' FORMAT 999 SQL.PNO SKIP 1 -
         LEFT report_date -
         CENTER 'Department of Pesticide Regulation' -
         RIGHT 'County: ' report_county SKIP 1 -
         CENTER '2001 Pesticide Use Report' -
         RIGHT 'Report: &1' SKIP 1 -
         CENTER 'Number of Valid Records Summary Report' SKIP 2

SELECT   record_id, COUNT(*) valid_recs
FROM     raw2001i praw
WHERE    NOT EXISTS
         (SELECT *
          FROM    errors2001i
          WHERE   praw.use_no = use_no AND
                  error_type != 'possible')
GROUP BY record_id;


TTITLE   LEFT 'Page ' FORMAT 999 SQL.PNO SKIP 1 -
         LEFT report_date -
         CENTER 'Department of Pesticide Regulation' -
         RIGHT 'County: ' report_county SKIP 1 -
         CENTER '2001 Pesticide Use Report' -
         RIGHT 'Report: &1' SKIP 1 -
         CENTER 'Number of Errors Summary Report' SKIP 2

SELECT   record_id, COUNT(*) invalid_values
FROM     raw2001i praw, errors2001i err
WHERE    praw.use_no = err.use_no AND
         error_type != 'possible'
GROUP BY record_id;


COLUMN applic_month HEADING 'APPLIC|MONTH' format a6
COLUMN batch_no HEADING 'BATCH|NUM' format a5
COLUMN all_recs HEADING 'TOTAL ALL|RECORDS' format 9,999,999
```

```
COMPUTE SUM LABEL 'TOTAL' OF all_recs ON REPORT

TTITLE   LEFT 'Page ' FORMAT 999 SQL.PNO SKIP 1 -
         LEFT report_date -
         CENTER 'Department of Pesticide Regulation' -
         RIGHT 'County: ' report_county SKIP 1 -
         CENTER '2001 Pesticide Use Report' -
         RIGHT 'Report: &1' SKIP 1 -
         CENTER 'Number of All Records Summary Report' SKIP 2

SELECT   record_id, SUBSTR(applic_dt, 1, 2) applic_month, batch_no,
         COUNT(*) all_recs
FROM     raw2001i praw
GROUP BY record_id, SUBSTR(applic_dt, 1, 2), batch_no;

spo off

EXIT 0
```

File_check.sql

```
SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
WHENEVER SQLERROR EXIT FAILURE ROLLBACK
WHENEVER OSERROR EXIT FAILURE ROLLBACK

VARIABLE v_error NUMBER

DECLARE
   v_number_rec NUMBER;
BEGIN
   SELECT   COUNT(*)
   INTO     v_number_rec
   FROM     raw2001
   WHERE    file_name = '&1';

   IF v_number_rec = 0 THEN
      :v_error := 0;
   ELSE
      :v_error := 1;
   END IF;
END;
/

EXIT :v_error
```

Get_prodno.sql

```
/* Package to find prodno that best matches the given registration number.
   First look for prodno with reported full registration number;
   if not found look for product with same 3-part registration number.
   Most people only enter 3-part reg. number; entry program
   will enter default values for revision_no therefore
   these values might be wrong.
   ALso, check if reported site_code is on label for reported product;
   sometimes use this information to choose likely product used.
 */

set termout on
SET document off
SET FEEDBACK ON
SET verify off
SET pause off
set serveroutput on size 1000000 format word_wrapped
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

CREATE OR REPLACE PACKAGE Get_prodno AS
   --v_rec_num_debug   INTEGER := 0;

   PROCEDURE Find (p_mfg_firmno IN NUMBER,
                   p_label_seq_no IN NUMBER,
                   p_revision_no IN VARCHAR2,
                   p_reg_firmno IN NUMBER,
                   p_prodno IN OUT NUMBER,
                   p_new_revision_no OUT VARCHAR2,
                   p_error_code IN OUT BINARY_INTEGER,
                   p_error_type OUT VARCHAR2,
                   p_replace_type OUT VARCHAR2);

   PROCEDURE Check_site (p_mfg_firmno IN NUMBER,
                   p_label_seq_no IN NUMBER,
                   p_revision_no IN VARCHAR2,
                   p_reg_firmno IN NUMBER,
                   p_unit_of_meas IN VARCHAR2,
                   p_site_code IN NUMBER,
                   p_applic_dt IN DATE,
                   p_prodno IN OUT NUMBER,
                   p_new_revision_no OUT VARCHAR2,
                   p_new_reg_firmno OUT NUMBER,
                   p_other_product OUT VARCHAR2,
                   p_error_code IN OUT BINARY_INTEGER,
                   p_error_type OUT VARCHAR2,
                   p_replace_type OUT VARCHAR2);

   FUNCTION HasValidSite
      (p_prodno IN NUMBER, p_site_code IN NUMBER,
       p_confid_sw IN CHAR, p_fumigant_sw IN CHAR,
       p_rodent_sw IN CHAR, p_agriccom_sw IN CHAR)
      RETURN BOOLEAN;

   FUNCTION HasValidSiteOther
      (p_mfg_firmno IN NUMBER, p_label_seq_no IN NUMBER, p_site_code IN NUMBER,
       p_other_product OUT VARCHAR2)
      RETURN BOOLEAN;

   FUNCTION HasSpecificGravity
      (p_specific_gravity IN NUMBER, p_formula_cd IN CHAR,
       p_unit_of_meas IN CHAR)
      RETURN BOOLEAN;

   FUNCTION HasValidRegDate(p_prodno IN NUMBER, p_applic_dt IN DATE)
      RETURN BOOLEAN;

END Get_prodno;
/
show errors
```

```
CREATE OR REPLACE PACKAGE BODY Get_prodno AS
   /* Find the prodno for given registration number.
    */
   PROCEDURE Find (p_mfg_firmno IN NUMBER,
                   p_label_seq_no IN NUMBER,
                   p_revision_no IN VARCHAR2,
                   p_reg_firmno IN NUMBER,
                   p_prodno IN OUT NUMBER,
                   p_new_revision_no OUT VARCHAR2,
                   p_error_code IN OUT BINARY_INTEGER,
                   p_error_type OUT VARCHAR2,
                   p_replace_type OUT VARCHAR2)
   IS
      v_num_products       INTEGER;

   BEGIN
      p_new_revision_no := p_revision_no;

      SELECT   prodno
      INTO     p_prodno
      FROM     product
      WHERE    mfg_firmno = p_mfg_firmno AND
               label_seq_no = p_label_seq_no AND
               revision_no = p_revision_no AND
               reg_firmno = DECODE(p_reg_firmno, 0, p_mfg_firmno, NULL, p_mfg_firmno,
                  p_reg_firmno);

      p_error_type := 'N';
      p_error_code := NULL;
      p_replace_type := 'same';

   EXCEPTION
      WHEN NO_DATA_FOUND THEN
         SELECT   count(*)
         INTO     v_num_products
         FROM     product
         WHERE    mfg_firmno = p_mfg_firmno AND
                  label_seq_no = p_label_seq_no AND
                  reg_firmno = DECODE(p_reg_firmno, 0, p_mfg_firmno, NULL, p_mfg_firmno,
                     p_reg_firmno);

         IF v_num_products = 1 THEN
            SELECT   prodno, revision_no
            INTO     p_prodno, p_new_revision_no
            FROM     product
            WHERE    mfg_firmno = p_mfg_firmno AND
                     label_seq_no = p_label_seq_no AND
                     reg_firmno = DECODE(p_reg_firmno, 0, p_mfg_firmno, NULL, p_mfg_firmno,
                        p_reg_firmno);

            p_error_type := 'possible';
            p_replace_type := 'estimate';
         ELSIF v_num_products > 1 THEN
            p_prodno := NULL;
            p_error_type := 'possible';
            p_replace_type := 'null';
         ELSE
            p_prodno := -1;
            p_error_type := 'invalid';
            p_replace_type := 'null';
         END IF;

      WHEN OTHERS THEN
         RAISE;
   END Find;


   /* CHeck if site_code is on the label for given product.
    */
   PROCEDURE Check_site (p_mfg_firmno IN NUMBER,
                   p_label_seq_no IN NUMBER,
                   p_revision_no IN VARCHAR2,
                   p_reg_firmno IN NUMBER,
                   p_unit_of_meas IN VARCHAR2,
                   p_site_code IN NUMBER,
                   p_applic_dt IN DATE,
                   p_prodno IN OUT NUMBER,
                   p_new_revision_no OUT VARCHAR2,
                   p_new_reg_firmno OUT NUMBER,
                   p_other_product OUT VARCHAR2,
                   p_error_code IN OUT BINARY_INTEGER,
                   p_error_type OUT VARCHAR2,
                   p_replace_type OUT VARCHAR2)
   IS

      TYPE     num_table_type IS TABLE OF NUMBER;
      TYPE     char_table_type IS TABLE OF VARCHAR2(2);

      t_prodno             num_table_type;
      t_formula_cd         char_table_type;
      t_spec_gravity       num_table_type;
      t_confid_sw          char_table_type;
      t_fumigant_sw        char_table_type;
      t_rodent_sw          char_table_type;
      t_agriccom_sw        char_table_type;
      t_revision_no        char_table_type;
      t_reg_firmno         num_table_type;

      v_confid_sw          VARCHAR2(1);
      v_fumigant_sw        VARCHAR2(1);
      v_rodent_sw          VARCHAR2(1);
      v_agriccom_sw        VARCHAR2(1);
      v_found_product      BOOLEAN := FALSE;
      v_num_products       INTEGER;

   BEGIN
      --DBMS_OUTPUT.ENABLE (1000000);

      p_new_revision_no := p_revision_no;
      p_new_reg_firmno := p_reg_firmno;
      p_other_product := NULL;

      /*
      v_rec_num_debug := v_rec_num_debug + 1;
      DBMS_OUTPUT.PUT_LINE('Record '||v_rec_num_debug||'; prodno '||NVL(p_prodno, 0)||
                        '; regno '||p_mfg_firmno||'-'||p_label_seq_no||'-'||p_revision_no||'-'||
                        p_reg_firmno||'; site '||p_site_code);
       */
      /* If previously found valid prod, check if a site_code is on label for this product.
       */

      IF p_prodno > 0 THEN
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'prodno > 0');

         SELECT   UPPER(NVL(confid_sw,'0')),
                  UPPER(NVL(fumigant_sw,'0')),
                  UPPER(NVL(rodent_sw,'0')),
                  UPPER(NVL(agriccom_sw,'0'))
         INTO     v_confid_sw, v_fumigant_sw, v_rodent_sw, v_agriccom_sw
         FROM     product
         WHERE    prodno = p_prodno;

         IF HasValidSite(p_prodno, p_site_code, v_confid_sw,
                         v_fumigant_sw, v_rodent_sw, v_agriccom_sw)
         THEN
            --DBMS_OUTPUT.PUT_LINE(chr(9)||'Has valid site');
            p_replace_type := 'same';
            p_error_type := 'N';
            p_error_code := NULL;
         ELSIF HasValidSiteOther(p_mfg_firmno, p_label_seq_no, p_site_code, p_other_product)
         THEN
```

```
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'Has valid site other');
         p_replace_type := 'same';
         p_error_type := 'possible';
      ELSE
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'No valid site');
         p_replace_type := 'same';
         p_error_type := 'invalid';
      END IF;

   /* prodno = NULL means that more than one product was found with the same
      3-part registration number, but none equal the reported 4-part reg. num.
      For this situation, loader has already return possible error code 37.
    */
   ELSIF p_prodno IS NULL THEN
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'prodno is NULL');
      /* Get all products with this 3-part registration number.
       */
      SELECT   prodno, formula_cd,
               DECODE(spec_gravity, 0, NULL, spec_gravity),
               UPPER(NVL(confid_sw,'0')),
               UPPER(NVL(fumigant_sw,'0')),
               UPPER(NVL(rodent_sw,'0')),
               UPPER(NVL(agriccom_sw,'0')),
               revision_no
      bulk collect INTO
               t_prodno, t_formula_cd, t_spec_gravity, t_confid_sw,
               t_fumigant_sw, t_rodent_sw, t_agriccom_sw, t_revision_no
      FROM     product
      WHERE    mfg_firmno = p_mfg_firmno AND
               label_seq_no = p_label_seq_no AND
               reg_firmno = DECODE(p_reg_firmno, 0, p_mfg_firmno, NULL, p_mfg_firmno,
                  p_reg_firmno);

      /*
      DBMS_OUTPUT.PUT_LINE(chr(9)||'Found these products:');
      FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
         DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||t_prodno(i));
      END LOOP;
      */

      /* Look for any products that have valid specific gravity,
         application site on its label, and was registered at
         the time of application.
       */
      --DBMS_OUTPUT.PUT_LINE(chr(9)||'First try...');
      FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
         IF HasSpecificGravity(t_spec_gravity(i), t_formula_cd(i), p_unit_of_meas) AND
            HasValidSite(t_prodno(i), p_site_code, t_confid_sw(i), t_fumigant_sw(i),
                         t_rodent_sw(i), t_agriccom_sw(i)) AND
            HasValidRegDate(t_prodno(i), p_applic_dt)
         THEN
            p_prodno := t_prodno(i);
            p_new_revision_no := t_revision_no(i);
            p_replace_type := 'estimate';
            p_error_type := 'possible';
            v_found_product := TRUE;
            --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has valid site +; prodno = '||p_prodno);
            EXIT;
         END IF;
      END LOOP;

      /* If have not found valid product, just look for product which has
         the site on its label.
       */
      IF NOT v_found_product THEN
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'Second try...');
         FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
            IF HasValidSite(t_prodno(i), p_site_code, t_confid_sw(i), t_fumigant_sw(i),
                            t_rodent_sw(i), t_agriccom_sw(i))
            THEN
               p_prodno := t_prodno(i);
               p_new_revision_no := t_revision_no(i);
               p_replace_type := 'estimate';
               p_error_type := 'possible';
               v_found_product := TRUE;
               --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has valid site; prodno = '||p_prodno);
               EXIT;
            END IF;
         END LOOP;
      END IF;

      /* If still have not found valid product, just look for product which has
         valid specific gravity.
       */
      IF NOT v_found_product THEN
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'Third try...');
         FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
            IF HasSpecificGravity(t_spec_gravity(i), t_formula_cd(i), p_unit_of_meas)
            THEN
               p_prodno := t_prodno(i);
               p_new_revision_no := t_revision_no(i);
               p_replace_type := 'estimate';

               IF HasValidSiteOther(p_mfg_firmno, p_label_seq_no, p_site_code, p_other_product)
               THEN
                  p_error_type := 'possible';
               ELSE
                  p_error_type := 'invalid';
               END IF;

               v_found_product := TRUE;
               --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has valid s.g.; prodno = '||p_prodno);
               EXIT;
            END IF;
         END LOOP;
      END IF;

      /* If still not found prodno with valid site and specific gravity,
         pick the first one you find.
       */
      IF NOT v_found_product THEN
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'Fourth try...');
         p_prodno := t_prodno(1);
         p_new_revision_no := t_revision_no(1);
         p_replace_type := 'estimate';

         IF HasValidSiteOther(p_mfg_firmno, p_label_seq_no, p_site_code, p_other_product)
         THEN
            p_error_type := 'possible';
         ELSE
            p_error_type := 'invalid';
         END IF;

         v_found_product := TRUE;
         --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has prod; prodno = '||p_prodno);
      END IF;

   /* prodno = -1 means that no product was found with the same
      3-part registration number.  For this situation, loader has
      already return invalid error code 37.
    */
   ELSE
      --DBMS_OUTPUT.PUT_LINE(chr(9)||'prodno = -1');
      /* Get all products with this 2-part registration number.
       */
      SELECT   prodno, formula_cd,
               DECODE(spec_gravity, 0, NULL, spec_gravity),
               UPPER(NVL(confid_sw,'0')),
               UPPER(NVL(fumigant_sw,'0')),
               UPPER(NVL(rodent_sw,'0')),
               UPPER(NVL(agriccom_sw,'0')),
               revision_no, reg_firmno
```

```
bulk collect INTO
         t_prodno, t_formula_cd, t_spec_gravity, t_confid_sw,
         t_fumigant_sw, t_rodent_sw, t_agriccom_sw,
         t_revision_no, t_reg_firmno
FROM     product
WHERE    mfg_firmno = p_mfg_firmno AND
         label_seq_no = p_label_seq_no;

/* If there are no products with this mfg_firmno and label_seq_no, then
   return prodno = -1 (NULL value) with invalid site code error.
 */
IF t_prodno.COUNT = 0 THEN
   p_replace_type := 'same';
   p_error_type := 'invalid';
   --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'No prodno');
   RETURN;
END IF;

/*
DBMS_OUTPUT.PUT_LINE(chr(9)||'Found these products:');
FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
   DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||t_prodno(i));
END LOOP;
*/

/* If there are some products, choose one that has valid specific gravity,
   application site on its label, and was registered at
   the time of application.
 */
--DBMS_OUTPUT.PUT_LINE(chr(9)||'First try...');
FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
   IF HasSpecificGravity(t_spec_gravity(i), t_formula_cd(i), p_unit_of_meas) AND
      HasValidSite(t_prodno(i), p_site_code, t_confid_sw(i), t_fumigant_sw(i),
                   t_rodent_sw(i), t_agriccom_sw(i)) AND
      HasValidRegDate(t_prodno(i), p_applic_dt)
   THEN
      p_prodno := t_prodno(i);
      p_new_revision_no := t_revision_no(i);
      p_new_reg_firmno := t_reg_firmno(i);
      p_replace_type := 'estimate';
      p_error_type := 'possible';
      v_found_product := TRUE;
      --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has valid site +; prodno = '||p_prodno);
      EXIT;
   END IF;
END LOOP;

/* If have not found valid product, just look for product which has
   the site on its label.
 */
IF NOT v_found_product THEN
   --DBMS_OUTPUT.PUT_LINE(chr(9)||'Second try...');
   FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
      IF HasValidSite(t_prodno(i), p_site_code, t_confid_sw(i), t_fumigant_sw(i),
                      t_rodent_sw(i), t_agriccom_sw(i))
      THEN
         p_prodno := t_prodno(i);
         p_new_revision_no := t_revision_no(i);
         p_new_reg_firmno := t_reg_firmno(i);
         p_replace_type := 'estimate';
         p_error_type := 'possible';
         v_found_product := TRUE;
         --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has valid site; prodno = '||p_prodno);
         EXIT;
      END IF;
   END LOOP;
END IF;

/* If still have not found valid product, just look for product which has
   valid specific gravity.
 */
```

```
      IF NOT v_found_product THEN
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'Third try...');
         FOR i IN t_prodno.FIRST..t_prodno.LAST LOOP
            IF HasSpecificGravity(t_spec_gravity(i), t_formula_cd(i), p_unit_of_meas)
            THEN
               p_prodno := t_prodno(i);
               p_new_revision_no := t_revision_no(i);
               p_new_reg_firmno := t_reg_firmno(i);
               p_replace_type := 'estimate';
               p_error_type := 'invalid';
               v_found_product := TRUE;
               --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has valid s.g.; prodno = '||p_prodno);
               EXIT;
            END IF;
         END LOOP;
      END IF;

      /* If still not found prodno with valid site and specific gravity,
         pick the first one you find.
       */
      IF NOT v_found_product THEN
         --DBMS_OUTPUT.PUT_LINE(chr(9)||'Fourth try...');
         p_prodno := t_prodno(1);
         p_new_revision_no := t_revision_no(1);
         p_new_reg_firmno := t_reg_firmno(1);
         p_replace_type := 'estimate';
         p_error_type := 'invalid';
         v_found_product := TRUE;
         --DBMS_OUTPUT.PUT_LINE(chr(9)||chr(9)||'Has prod; prodno = '||p_prodno);
      END IF;

   END IF;

EXCEPTION
   WHEN OTHERS THEN
      RAISE;
END Check_site;


/* Is this site on the label for this product?
 */
FUNCTION HasValidSite
   (p_prodno IN NUMBER, p_site_code IN NUMBER,
    p_confid_sw IN CHAR, p_fumigant_sw IN CHAR,
    p_rodent_sw IN CHAR, p_agriccom_sw IN CHAR)
RETURN BOOLEAN IS
   v_num_sites   NUMBER(5) := 0;
BEGIN
   IF p_prodno IS NOT NULL AND p_site_code >= 1000 AND
      p_confid_sw != 'X' AND p_fumigant_sw != 'X'  AND
      p_rodent_sw != 'X' AND p_agriccom_sw != 'X'
   THEN
      SELECT COUNT(*)
      INTO  v_num_sites
      FROM  prod_site
      WHERE prodno = p_prodno AND
            site_code = p_site_code;
   ELSE
      RETURN TRUE;
   END IF;

   IF v_num_sites >= 1 THEN
      RETURN TRUE;
   ELSE
      RETURN FALSE;
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      RAISE;
END HasValidSite;
```

```
/* Is this site on a label for any products with this 2-part registration number?
 */
FUNCTION HasValidSiteOther
   (p_mfg_firmno IN NUMBER, p_label_seq_no IN NUMBER, p_site_code IN NUMBER,
    p_other_product OUT VARCHAR2)
RETURN BOOLEAN IS
   v_found_product   BOOLEAN := FALSE;

   CURSOR prodno_cur IS
      SELECT   prodno,
               UPPER(NVL(confid_sw,'0')) confid_flag,
               UPPER(NVL(fumigant_sw,'0')) fumigant_flag,
               UPPER(NVL(rodent_sw,'0')) rodent_flag,
               UPPER(NVL(agriccom_sw,'0')) agriccom_flag,
               revision_no, reg_firmno
      FROM     product
      WHERE    mfg_firmno = p_mfg_firmno AND
               label_seq_no = p_label_seq_no;

   prodno_rec prodno_cur%ROWTYPE;

BEGIN
   p_other_product := NULL;

   OPEN prodno_cur;
   LOOP
      FETCH prodno_cur INTO prodno_rec;
      EXIT WHEN prodno_cur%NOTFOUND OR v_found_product;

      IF HasValidSite(prodno_rec.prodno, p_site_code,
                      prodno_rec.confid_flag, prodno_rec.fumigant_flag,
                      prodno_rec.rodent_flag, prodno_rec.agriccom_flag)
      THEN
         v_found_product := TRUE;
         IF p_mfg_firmno = prodno_rec.reg_firmno THEN
            p_other_product := p_mfg_firmno||'-'||p_label_seq_no||'-'||
               prodno_rec.revision_no;
         ELSE
            p_other_product := p_mfg_firmno||'-'||p_label_seq_no||'-'||
               prodno_rec.revision_no||'-'||prodno_rec.reg_firmno;
         END IF;
      END IF;
   END LOOP;
   CLOSE prodno_cur;

   RETURN v_found_product;

EXCEPTION
   WHEN OTHERS THEN
      RAISE;
END HasValidSiteOther;


/* Does this product have a valid specific gravity?
 */
FUNCTION HasSpecificGravity
   (p_specific_gravity IN NUMBER, p_formula_cd IN CHAR,
    p_unit_of_meas IN CHAR)
RETURN BOOLEAN
IS
BEGIN
   IF (p_specific_gravity < 0.0001 OR p_specific_gravity IS NULL) AND
      (p_unit_of_meas IN ('GA','QT','PT','LI','ML') OR
         (p_unit_of_meas = 'OZ' AND
          p_formula_cd IN ('B0','C0','D0','G0','H0','I0','M0','O0','Q0','S0','T0')))
   THEN
      RETURN FALSE;
   ELSE
      RETURN TRUE;
   END IF;
EXCEPTION
```

```
      WHEN OTHERS THEN
         RAISE;
   END HasSpecificGravity;

   /* Was this product registered at the time of application?
    */
   FUNCTION HasValidRegDate(p_prodno IN NUMBER, p_applic_dt IN DATE)
   RETURN BOOLEAN
   IS
      v_reg_dt DATE;
      v_prod_inac_dt DATE;
   BEGIN
         -- product is a copy of product@empmdb2_wilhoit
      SELECT   reg_dt, prod_inac_dt
      INTO     v_reg_dt, v_prod_inac_dt
      FROM     product
      WHERE    prodno = p_prodno;

      IF v_reg_dt IS NULL AND p_applic_dt < v_prod_inac_dt THEN
         RETURN TRUE;
      ELSIF v_prod_inac_dt is NULL AND p_applic_dt > v_reg_dt THEN
         RETURN TRUE;
      ELSIF p_applic_dt BETWEEN v_reg_dt and v_prod_inac_dt THEN
         RETURN TRUE;
      ELSE
         RETURN FALSE;
      END IF;
   EXCEPTION
      WHEN OTHERS THEN
         RAISE;
   END HasValidRegDate;

END Get_prodno;
/
show errors

EXIT 0
```

Intermediate_check.sql

```
/* Check the number of rows that now exist in raw2001i,
   errors2001i, changes2001i.
   These tables should have no rows before loading data
   from a new file.  Thus, if this returns any value > 0,
   there is some kind of error.
 */

SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET document off
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

VARIABLE v_error NUMBER

DECLARE
   v_number_raw NUMBER;
   v_number_errs NUMBER;
   v_number_chs NUMBER;
   v_number_recs NUMBER;
BEGIN
   SELECT   COUNT(*)
   INTO     v_number_raw
   FROM     raw2001i;

   SELECT   COUNT(*)
   INTO     v_number_errs
   FROM     errors2001i;

   SELECT   COUNT(*)
   INTO     v_number_chs
   FROM     changes2001i;

   v_number_recs := v_number_raw + v_number_errs + v_number_chs;

   IF v_number_recs = 0 THEN
      :v_error := 0;
   ELSE
      :v_error := 1;
   END IF;
EXCEPTION
   WHEN OTHERS THEN
      :v_error := 1;
END;
/

EXIT :v_error
```

Load_log.sql

```
/* Record information about the loading of
   the county file into RAWI.
 */

SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET serveroutput ON
WHENEVER SQLERROR EXIT :v_error ROLLBACK
WHENEVER OSERROR EXIT :v_error ROLLBACK

VARIABLE v_error NUMBER

DECLARE
   p_file_name   VARCHAR2(20);
   p_num_rows    INTEGER;
   v_file_name   VARCHAR2(20);
   v_num_rows    INTEGER;
BEGIN
   p_file_name := '&1';
   p_num_rows := &2;

   /* Check that the current file name matches the value in raw2001i.
    */
   BEGIN
      SELECT   DISTINCT file_name
      INTO     V_file_name
      FROM     raw2001i;

      :v_error := 0;
   EXCEPTION
      WHEN TOO_MANY_ROWS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! raw2001i has data from more than one county file.');

      WHEN NO_DATA_FOUND THEN
         /* For some reason this exception is not raised if there are no
            values for file_name and file_date
           */
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! raw2001i has no county file names.');
   END;

   IF :v_error = 0 THEN
      IF V_file_name != p_file_name THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! The file name in raw2001i ('||v_file_name||')');
         DBMS_OUTPUT.PUT_LINE('**** does not match current county file ('||p_file_name||').');
      END IF;
   END IF;


   /* Check that the number of rows in raw2001i equals the number of lines
      in the county file.
    */
   IF :v_error = 0 THEN
      SELECT   COUNT(*)
      INTO     v_num_rows
      FROM     raw2001i;

      IF v_num_rows < p_num_rows THEN
         :v_error := 1;
               DBMS_OUTPUT.PUT_LINE('**** Error! Not all rows from county file were loaded into raw2001i.');
      ELSIF v_num_rows > p_num_rows THEN
         :v_error := 1;
               DBMS_OUTPUT.PUT_LINE('**** Error! There were too many rows loaded into raw2001i.');
      END IF;
   END IF;

   /* Create a log of the loaded data.
    */
   IF :v_error = 0 THEN
      BEGIN
         INSERT INTO log2001
         (file_name, file_date, load_date, num_of_records, start_use_no, end_use_no)
            SELECT   file_name, file_date, SYSDATE, v_num_rows, min(use_no), max(use_no)
            FROM     raw2001i
            GROUP BY file_name, file_date;

         COMMIT;
      EXCEPTION
         WHEN OTHERS THEN
            :v_error := 1;
                  DBMS_OUTPUT.PUT_LINE('**** Error! There was error making a log of the county file 
loading.');
      END;
   END IF;

EXCEPTION
   WHEN OTHERS THEN
      :v_error := 1;
      DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
END;
/

EXIT :v_error
```

Load_pur_c_old.ctl

```
options (direct=true, errors=999999)
load data
into table raw2001i
append
(record_id      position(1:1) CHAR,
process_mt     position(2:3) CHAR,
process_yr     position(4:5) CHAR,
batch_no       position(6:9) CHAR,
nursery_ind    position(14:14) CHAR,
county_cd      position(15:16) CHAR,
section        position(17:18) CHAR,
township       position(19:20) CHAR,
tship_dir      position(21:21) CHAR,
range          position(22:23) CHAR,
range_dir      position(24:24) CHAR,
base_ln_mer    position(25:25) CHAR,
aer_gnd_ind    position(26:26) CHAR,
grower_id      position(27:37) CHAR,
cedts_ind      position(38:38) CHAR,
site_loc_id    position(39:46) CHAR,
acre_planted   position(47:54) CHAR,
unit_planted   position(55:55) CHAR,
applic_dt      position(56:61) CHAR,
site_code      position(62:67) CHAR,
qualify_cd     position(68:69) CHAR,
planting_seq   position(70:70) CHAR,
acre_treated   position(71:78) CHAR,
unit_treated   position(79:79) CHAR,
mfg_firmno     position(80:86) CHAR,
label_seq_no   position(87:91) CHAR,
revision_no    position(92:93) CHAR,
reg_firmno     position(94:100) CHAR,
amt_prd_used   position(101:110) CHAR,
unit_of_meas   position(111:112) CHAR,
document_no    position(113:120) CHAR,
summary_cd     position(121:124) CHAR,
applic_cnt     position(125:130) CHAR,
file_date      position(131:150) DATE "DD-MON-YYYY HH24:MI:SS",
file_name      position(*) CHAR TERMINATED BY X'0A'
)
```

Load_pur_c_new.ctl

```
options (direct=true, errors=999999)
load data
into table raw2001i
append
(record_id      position(1:1) CHAR,
process_mt     position(2:3) CHAR,
process_yr     position(4:5) CHAR,
batch_no       position(6:9) CHAR,
nursery_ind    position(14:14) CHAR,
county_cd      position(15:16) CHAR,
section        position(17:18) CHAR,
township       position(19:20) CHAR,
tship_dir      position(21:21) CHAR,
range          position(22:23) CHAR,
range_dir      position(24:24) CHAR,
base_ln_mer    position(25:25) CHAR,
aer_gnd_ind    position(26:26) CHAR,
grower_id      position(27:37) CHAR,
cedts_ind      position(38:38) CHAR,
site_loc_id    position(39:46) CHAR,
acre_planted   position(47:54) CHAR,
unit_planted   position(55:55) CHAR,
applic_dt      position(56:61) CHAR,
site_code      position(62:67) CHAR,
qualify_cd     position(68:69) CHAR,
planting_seq   position(70:70) CHAR,
acre_treated   position(71:78) CHAR,
unit_treated   position(79:79) CHAR,
mfg_firmno     position(80:86) CHAR,
label_seq_no   position(87:91) CHAR,
revision_no    position(92:93) CHAR,
reg_firmno     position(94:100) CHAR,
amt_prd_used   position(101:110) CHAR,
unit_of_meas   position(111:112) CHAR,
document_no    position(113:120) CHAR,
summary_cd     position(121:124) CHAR,
applic_cnt     position(125:130) CHAR,
license_no     position(131:143) CHAR,
file_date      position(144:163) DATE "DD-MON-YYYY HH24:MI:SS",
file_name      position(*) CHAR TERMINATED BY X'0A'
)
```

Load_pur_f_old.ctl

```
options (direct=true, errors=999999)
load data
into table raw2001i
append
(record_id       position(1:1) CHAR,
process_mt      position(2:3) CHAR,
process_yr      position(4:5) CHAR,
batch_no        position(6:9) CHAR,
nursery_ind     position(14:14) CHAR,
county_cd       position(15:16) CHAR,
section         position(17:18) CHAR,
township        position(19:20) CHAR,
tship_dir       position(21:21) CHAR,
range           position(22:23) CHAR,
range_dir       position(24:24) CHAR,
base_ln_mer     position(25:25) CHAR,
aer_gnd_ind     position(26:26) CHAR,
grower_id       position(27:37) CHAR,
cedts_ind       position(38:38) CHAR,
site_loc_id     position(39:46) CHAR,
acre_planted    position(47:54) CHAR,
unit_planted    position(55:55) CHAR,
applic_dt       position(56:61) CHAR,
site_code       position(62:67) CHAR,
qualify_cd      position(68:69) CHAR,
planting_seq    position(70:70) CHAR,
acre_treated    position(71:78) CHAR,
unit_treated    position(79:79) CHAR,
mfg_firmno      position(80:86) CHAR,
label_seq_no    position(87:91) CHAR,
revision_no     position(92:93) CHAR,
reg_firmno      position(94:100) CHAR,
amt_prd_used    position(101:110) CHAR,
unit_of_meas    position(111:112) CHAR,
document_no     position(113:120) CHAR,
summary_cd      position(121:124) CHAR,
file_date       position(125:144) DATE "DD-MON-YYYY HH24:MI:SS",
file_name       position(*) CHAR TERMINATED BY X'0A'
)
```

Load_pur_f_new.ctl

```
options (direct=true, errors=999999)
load data
into table raw2001i
append
(record_id      position(1:1) CHAR,
process_mt     position(2:3) CHAR,
process_yr     position(4:5) CHAR,
batch_no       position(6:9) CHAR,
nursery_ind    position(14:14) CHAR,
county_cd      position(15:16) CHAR,
section        position(17:18) CHAR,
township       position(19:20) CHAR,
tship_dir      position(21:21) CHAR,
range          position(22:23) CHAR,
range_dir      position(24:24) CHAR,
base_ln_mer    position(25:25) CHAR,
aer_gnd_ind    position(26:26) CHAR,
grower_id      position(27:37) CHAR,
cedts_ind      position(38:38) CHAR,
site_loc_id    position(39:46) CHAR,
acre_planted   position(47:54) CHAR,
unit_planted   position(55:55) CHAR,
applic_dt      position(56:61) CHAR,
site_code      position(62:67) CHAR,
qualify_cd     position(68:69) CHAR,
planting_seq   position(70:70) CHAR,
acre_treated   position(71:78) CHAR,
unit_treated   position(79:79) CHAR,
mfg_firmno     position(80:86) CHAR,
label_seq_no   position(87:91) CHAR,
revision_no    position(92:93) CHAR,
reg_firmno     position(94:100) CHAR,
amt_prd_used   position(101:110) CHAR,
unit_of_meas   position(111:112) CHAR,
document_no    position(113:120) CHAR,
summary_cd     position(121:124) CHAR,
applic_time    position(125:128) CHAR,
file_date      position(129:148) DATE "DD-MON-YYYY HH24:MI:SS",
file_name      position(*) CHAR TERMINATED BY X'0A'
)
```

Load_pur_caltrans.ctl

```
options (direct=true, errors=999999)
load data
into table raw2001i
append
(record_id      position(1:1) CHAR,
process_mt     position(2:3) CHAR,
process_yr     position(4:5) CHAR,
batch_no       position(6:9) CHAR,
county_cd      position(15:16) CHAR,
applic_dt      position(56:61) CHAR,
site_code      position(62:67) CHAR,
mfg_firmno     position(80:86) CHAR,
label_seq_no   position(87:91) CHAR,
revision_no    position(92:93) CHAR,
reg_firmno     position(94:100) CHAR,
amt_prd_used   position(101:110) CHAR,
unit_of_meas   position(111:112) CHAR,
document_no    position(113:120) CHAR,
summary_cd     position(121:124) CHAR,
file_date      position(135:154) DATE "DD-MON-YYYY HH24:MI:SS",
file_name      position(*) CHAR TERMINATED BY X'0A'
)
```

loaded_files.sql

```
/* Create file listing all county PUR files that have
   been loaded so far this year.
 */

SET pause off
SET pagesize 0
SET linesize 3000
SET termout off
SET feedback off
SET document off
SET trimspool on
SET serveroutput ON
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

spo /Scalos_ora05/data/2001data/loaded_files_2001.txt
SELECT   'File Name'||chr(9)||'File Date'||chr(9)||'Load Date'||chr(9)||
         'Num. Rec.'||chr(9)||'Start use_no'||chr(9)||'End use_no'
FROM     dual;

SELECT   file_name||chr(9)||file_date||chr(9)||load_date||chr(9)||
         num_of_records||chr(9)||start_use_no||chr(9)||end_use_no
FROM     log2001
ORDER BY load_date;
spo off

EXIT 0
```

Move_intermediates.sql

```
/* Move all the records from the intermediate tables
   to the main tables:
   raw2001i to raw2001,
   errors2001i to errors2001,
   changes2001i to changes2001.

   If there is an error during any stage this procedure ends
   with return code of 0.  If all succeed it returns 1--
 */

SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET serveroutput ON
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

VARIABLE v_error NUMBER

BEGIN
   :v_error := 0;

   /* First move data from raw2001i to raw2001.
    */
   BEGIN
      INSERT INTO raw2001
         SELECT * FROM  raw2001i;
   EXCEPTION
      WHEN OTHERS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! Could not insert records into raw2001.');
         DBMS_OUTPUT.PUT_LINE('**** ' || SQLERRM);
         RAISE;
   END;

   -- No need to test for :v_error because calling RAISE in exception will
   -- propogate to this block's exception section.
   BEGIN
      DELETE FROM raw2001i;
      /* Note we cannot use TRUNCATE, for example, by calling:

         EXECUTE IMMEDIATE 'TRUNCATE TABLE raw2001i';

         because this is a DDL statement and, therefore, will
         cause a commit to be called.  We don't want to commit
         at this point because if we failed to delete records from
         raw2001i we do not want to insert the records into raw_test.
        */

      COMMIT;
   EXCEPTION
      WHEN OTHERS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! Could not delete records from raw2001i.');
         DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
         RAISE;
   END;


   /* Next, move data from errors2001i to errors2001.
    */
   BEGIN
      INSERT INTO errors2001
         SELECT * FROM  errors2001i;
   EXCEPTION
      WHEN OTHERS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! Could not insert records into errors2001.');
         DBMS_OUTPUT.PUT_LINE('**** ' || SQLERRM);
   END;

   BEGIN
      DELETE FROM errors2001i;
      COMMIT;
   EXCEPTION
      WHEN OTHERS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! Could not delete records from errors2001i.');
         DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
   END;

   /* Next, move data from changes2001i to changes2001.
    */
   BEGIN
      INSERT INTO changes2001
         SELECT * FROM  changes2001i;
   EXCEPTION
      WHEN OTHERS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! Could not insert records into changes2001.');
         DBMS_OUTPUT.PUT_LINE('**** ' || SQLERRM);
   END;

   BEGIN
      DELETE FROM changes2001i;
      COMMIT;
   EXCEPTION
      WHEN OTHERS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! Could not delete records from changes2001i.');
         DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
   END;

   /* Next, delete data from use_error_report2001i.
      There is no reason to keep this data since it is just used to create error reports.
    */
   BEGIN
      DELETE FROM use_error_report2001i;
      COMMIT;
   EXCEPTION
      WHEN OTHERS THEN
         :v_error := 1;
         DBMS_OUTPUT.PUT_LINE('**** Error! Could not delete records from use_error_report2001i.');
         DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
   END;



EXCEPTION
   WHEN OTHERS THEN
      :v_error := 1;
      DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
END;
/

EXIT :v_error
```

Outlier.sql

```
/* Package to determine what if any values to replace when
   outlier in rate of use if found.

 */

set termout on
SET document off
SET FEEDBACK OFF
SET verify off
SET pause off
set serveroutput on size 1000000 format word_wrapped
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

CREATE OR REPLACE PACKAGE Outlier AS

   FUNCTION Wrong_unit
      (p_record_id IN VARCHAR2, p_site_code IN NUMBER, p_prodno IN NUMBER,
       p_rate IN NUMBER, p_acre_planted IN NUMBER, p_unit_planted IN VARCHAR2,
       p_acre_treated IN NUMBER, p_unit_treated IN OUT VARCHAR2)
   RETURN BOOLEAN;

   PROCEDURE Estimate_unit
      (p_estimated_field OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   FUNCTION Wrong_acres
      (p_record_id IN VARCHAR2, p_site_code IN NUMBER,
       p_lbs_prd_used IN NUMBER, p_median IN NUMBER,
       p_acre_planted IN VARCHAR2, p_unit_planted IN VARCHAR2,
       p_acre_treated IN OUT VARCHAR2, p_unit_treated IN VARCHAR2)
   RETURN BOOLEAN;

   PROCEDURE Estimate_acres
      (p_estimated_field OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

   PROCEDURE Estimate_lbs
      (p_estimated_field OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2);

END Outlier;
/
show errors

CREATE OR REPLACE PACKAGE BODY Outlier AS

   /* First see if unit_treated is unusual for this type of application--
      if so, and using the usual unit makes the rate reasonable,
      change unit.
      Nearly all prod ag records use unit = A, except for site_codes in
         (90, 91, 16003, 40010 - 65999).
         In many cases, other unit is S.  If S is changed to A this
         will reduce rate of use.

      If change unit_treated to A makes acres_treated > acres_planted,
            then we should not change unit_treated.
     */
   FUNCTION Wrong_unit
      (p_record_id IN VARCHAR2, p_site_code IN NUMBER, p_prodno IN NUMBER,
       p_rate IN NUMBER, p_acre_planted IN NUMBER, p_unit_planted IN VARCHAR2,
       p_acre_treated IN NUMBER, p_unit_treated IN OUT VARCHAR2)
   RETURN BOOLEAN IS
      v_ai_a_1000_200   NUMBER;
      v_prd_u_50m       NUMBER;
      v_nn4             NUMBER;
      v_acres_planted   NUMBER(10, 2) := NULL;
   BEGIN
      IF p_record_id IN ('1','4','A','B') AND
         p_unit_planted IN ('A', 'S') AND
         (p_site_code NOT IN (90, 91, 16003) AND
          p_site_code NOT BETWEEN 40010 AND 65999) AND
         p_unit_treated <> 'A'
      THEN
         /* If change unit_treated to A makes acres_treated > acres_planted,
            then we should not change unit_treated.
          */
         IF p_unit_planted = 'A' THEN
            v_acres_planted := p_acre_planted;
         ELSIF p_unit_planted = 'S' THEN
            v_acres_planted := p_acre_planted/43560;
         END IF;

         IF p_acre_treated > v_acres_planted THEN
            RETURN FALSE;
         END IF;


         SELECT   ai_a_1000_200, prd_u_50m, nn4
         INTO     v_ai_a_1000_200, v_prd_u_50m, v_nn4
         FROM     usetype99stats
         WHERE    prodno = p_prodno AND
                  site_code = p_site_code AND
                  unit_treated = 'A' AND
                  record_id_type =
                     TRANSLATE(p_record_id, 'C2G9DHAB14EF', 'NNNNNNAAAAAA');

         IF p_rate < LEAST(v_ai_a_1000_200, v_prd_u_50m, V_nn4)  THEN
            p_unit_treated := 'A';
            RETURN TRUE;
         END IF;
      END IF;

      RETURN FALSE;

   EXCEPTION
      WHEN OTHERS THEN
         RETURN FALSE;
   END Wrong_unit;

   PROCEDURE Estimate_unit
      (p_estimated_field OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
   IS
   BEGIN
      p_estimated_field := 'unit_treated';
      p_error_type := 'probable';
      p_replace_type := 'estimate';
   END Estimate_unit;


   /* For situations with low value for acre_treated (nursery, greenhouse,
      and a few other uses can have low areas treated so ignore those),
      calculate a new value for acre_treated from the lbs_prd_used
      and the median rate for this product on this site.
      If the new value for acre_treated is less than acre_planted then
      use the new value as an estimate of acre_treated.
    */
   FUNCTION Wrong_acres
      (p_record_id IN VARCHAR2, p_site_code IN NUMBER,
       p_lbs_prd_used IN NUMBER, p_median IN NUMBER,
       p_acre_planted IN VARCHAR2, p_unit_planted IN VARCHAR2,
       p_acre_treated IN OUT VARCHAR2, p_unit_treated IN VARCHAR2)
   RETURN BOOLEAN IS
      v_estimated_acre_treated   NUMBER(10, 2) := NULL;
      v_acres_planted            NUMBER(10, 2) := NULL;
      v_median_rate_acres        NUMBER(10, 2) := NULL;

      v_estimated_cu_ft_treated  NUMBER(10, 2) := NULL;
      v_cubic_feet_planted       NUMBER(10, 2) := NULL;
      v_median_rate_cu_ft        NUMBER(10, 2) := NULL;
   BEGIN
      IF p_record_id IN ('1','4','A','B') AND
         p_acre_treated < 1.0 AND
         p_acre_planted IS NOT NULL AND
         p_unit_treated IN ('A','S','C','K') AND
         p_unit_planted IN ('A','S','C','K') AND
         (p_site_code NOT BETWEEN 151 AND 156 AND
          p_site_code NOT BETWEEN 8000 AND 8049 AND
          p_site_code NOT IN (13501, 28012, 29123))
      THEN
         IF p_unit_treated = 'A' THEN
            v_median_rate_acres := p_median;
         ELSIF p_unit_treated = 'S' THEN
            v_median_rate_acres := p_median * 43560;
         ELSIF p_unit_treated = 'C' THEN
            v_median_rate_cu_ft := p_median;
         ELSIF p_unit_treated = 'K' THEN
            v_median_rate_cu_ft := p_median/1000;
         END IF;

         IF p_unit_planted = 'A' THEN
            v_acres_planted := p_acre_planted;
         ELSIF p_unit_planted = 'S' THEN
            v_acres_planted := p_acre_planted/43560;
         ELSIF p_unit_planted = 'C' THEN
            v_cubic_feet_planted := p_acre_planted;
         ELSIF p_unit_planted = 'K' THEN
            v_cubic_feet_planted := p_acre_planted * 1000;
         END IF;

         IF p_unit_treated IN ('A', 'S') AND
            p_unit_planted IN ('A', 'S')
         THEN
            IF v_median_rate_acres < 0.000001 THEN
               v_median_rate_acres := 0.000001;
            END IF;

            v_estimated_acre_treated := p_lbs_prd_used/v_median_rate_acres;

            IF v_estimated_acre_treated <= v_acres_planted THEN
               IF p_unit_treated = 'A' THEN
                  p_acre_treated := v_estimated_acre_treated;
               ELSE
                  p_acre_treated := v_estimated_acre_treated * 43560;
               END IF;

               RETURN TRUE;
            END IF;
         ELSIF p_unit_treated IN ('C', 'K') AND
               p_unit_planted IN ('C', 'K')
         THEN
            IF v_median_rate_cu_ft < 0.000001 THEN
               v_median_rate_cu_ft := 0.000001;
            END IF;

            v_estimated_cu_ft_treated := p_lbs_prd_used/v_median_rate_cu_ft;

            IF v_estimated_cu_ft_treated <= v_cubic_feet_planted THEN
               IF p_unit_treated = 'C' THEN
                  p_acre_treated := v_estimated_cu_ft_treated;
               ELSE
                  p_acre_treated := v_estimated_cu_ft_treated/1000;
               END IF;

               RETURN TRUE;
            END IF;

         END IF;
      END IF;

      RETURN FALSE;

   EXCEPTION
      WHEN OTHERS THEN
         RETURN FALSE;
   END Wrong_acres;

   PROCEDURE Estimate_acres
      (p_estimated_field OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
   IS
   BEGIN
      p_estimated_field := 'acre_treated';
      p_error_type := 'probable';
      p_replace_type := 'estimate';
   END Estimate_acres;


   /* Estimate lbs_prd_used using the median rate and acre_treated.
      This function gets called for all cases where
      the rate of use was higher than at least one outlier criteria and
      the unit_treated and acre_treated could not be estimated.
    */

   PROCEDURE Estimate_lbs
      (p_estimated_field OUT VARCHAR2,
       p_error_type OUT VARCHAR2, p_replace_type OUT VARCHAR2)
   IS
   BEGIN
      p_estimated_field := 'lbs_prd_used';
      p_error_type := 'probable';
      p_replace_type := 'estimate';
   END Estimate_lbs;



END Outlier;
/
show errors

EXIT 0
```

Procedure_validity.sql

```
SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET serveroutput ON
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

VARIABLE v_error NUMBER

DECLARE
   v_status          VARCHAR2(7);
   v_status_body     VARCHAR2(7);
   v_file_name       VARCHAR2(30);
BEGIN
   v_file_name := RTRIM(LOWER('&1'), '.sql');

   SELECT   status
   INTO     v_status
   FROM     user_objects
   WHERE    object_name = UPPER(v_file_name) AND
            object_type = 'PACKAGE';

   SELECT   status
   INTO     v_status_body
   FROM     user_objects
   WHERE    object_name = UPPER(v_file_name) AND
            object_type = 'PACKAGE BODY';

   IF v_status = 'VALID' AND v_status_body = 'VALID' THEN
      :v_error := 0;
   ELSE
      :v_error := 1;
   END IF;

EXCEPTION
   WHEN OTHERS THEN
      :v_error := 1;
      DBMS_OUTPUT.PUT_LINE('**** Error! ' || SQLERRM);
END;
/

EXIT :v_error
```

Update_log.sql

```
/* Update table log2001 by adding the min and max use_no.

   If there is an error during any stage this procedure ends
   with return code of 0.  If all succeed it returns 1--
 */

SET pause off
SET verify off
SET TERMOUT ON
SET FEEDBACK OFF
SET serveroutput ON
WHENEVER SQLERROR EXIT 1 ROLLBACK
WHENEVER OSERROR EXIT 1 ROLLBACK

UPDATE log2001
SET    (start_use_no, end_use_no) =
         (SELECT min(use_no), max(use_no)
          FROM   raw2001i)
WHERE  file_name = '&1';
COMMIT;

EXIT 0
```

Setup.sql

```
#!/usr/local/bin/perl

# Procedures to create the directories and files for
# the next year's PUR loading.

mkdir /home/purload/pur2001
mkdir /home/purload/pur2001/2001data
mkdir /home/purload/pur2001/continue
mkdir /home/purload/pur2001/data
mkdir /home/purload/pur2001/load_logs
mkdir /home/purload/pur2001/loaded
mkdir /home/purload/pur2001/loading
mkdir /home/purload/pur2001/not_loaded
mkdir /home/purload/pur2001/old
mkdir /home/purload/pur2001/reports
mkdir /home/purload/pur2001/reports/printed
mkdir /home/purload/pur2001/reports/to_print

chmod 770 /home/purload/pur2001
chmod 770 /home/purload/pur2001/2001data
chmod 770 /home/purload/pur2001/continue
chmod 770 /home/purload/pur2001/data
chmod 770 /home/purload/pur2001/load_logs
chmod 770 /home/purload/pur2001/loaded
chmod 770 /home/purload/pur2001/loading
chmod 770 /home/purload/pur2001/not_loaded
chmod 770 /home/purload/pur2001/old
chmod 770 /home/purload/pur2001/reports
chmod 770 /home/purload/pur2001/reports/printed
chmod 770 /home/purload/pur2001/reports/to_print

mkdir /home/purload/sql/pur2001
chmod 770 /home/purload/sql/pur2001

cp -p /home/purload/sql/pur2000/check_errors.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/check_value2000.sql /home/purload/sql/pur2001/check_value2001.sql
cp -p /home/purload/sql/pur2000/clear_intermediates.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/co_error2000.sql /home/purload/sql/pur2001/co_error2001.sql
cp -p /home/purload/sql/pur2000/dates.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/error_report.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/file_check.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/get_prodno.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/intermediate_check.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/load_log.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/load_pur_c_new.ctl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/load_pur_c_old.ctl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/load_pur_caltrans.ctl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/load_pur_f_new.ctl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/load_pur_f_old.ctl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/move_intermediates.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/outlier.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/procedure_validity.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/update_log.sql /home/purload/sql/pur2001

cp -p /home/purload/sql/pur2000/loader.pl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/batch_run.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/check.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/check_load.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/continue.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/continue2.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/continue_template.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/create_tables.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/create_index.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/create_use_error_report.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/delete_file.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/delete_file_short.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/delete_file_batch.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/error_report_by_county.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/error_report_by_file.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/error_report_caltrans.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/error_report_continue.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/error_reports.pl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/loaded_files.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/pur2_add_2000.sql /home/purload/sql/pur2001/pur2_add_2001.sql
cp -p /home/purload/sql/pur2000/results.sql /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/run_err_by_county.pl /home/purload/sql/pur2001
cp -p /home/purload/sql/pur2000/setup /home/purload/sql/pur2001

#  This reads all files in the current directory
#  (except for this script and files with file name extensions "vpj" and "vtg")
#  and replaces "2000" with "2001"
#  unless "(" or "=" preceeds "2000",
#  or "2000" is at beginning of a line.
#

while ($v_nextfile = <*>) {
   $v_extension = $v_nextfile;
   $v_extension =~ s/.*\.//; # remove part before last ".".

   # If the next file is not this script file ($0 is name of this perl script)
   # and is a plain file and file name extension not "vpj" and "vtg", then proceed
   if ($v_nextfile ne $0 and -f $v_nextfile and
       $v_extension ne "vpj" and $v_extension ne "vtg") {
      print $v_nextfile."\n";

      open(INFILE, "$v_nextfile") ||
         die "**** Error! Couldn't read file $v_nextfile, stopped";

      open(TMPFILE, ">$v_nextfile.t") ||
         die "**** Error! Couldn't write to file $v_nextfile.t, stopped";

      # Replace "2000" with "2001" unless "(" or "=" preceeds "2000"
      while(<INFILE>) {
         s/([^(=])2000/${1}2001/g;
         print TMPFILE $_;
      }

      close(INFILE);
      close(TMPFILE);

      # Delete temporary file
      unlink($v_nextfile);
      rename("$v_nextfile.t", $v_nextfile);

      # Change permissions
      chmod(0770, $v_nextfile);
   }
}
```