November 2005

Click to return to Database Documentation

County-Based Data Validation Procedures - Program Flow

This document gives a brief description of each trigger that fires as records from agencies that place traps on a county-based system are loaded into the database. Currently, Illinois and Wisconsin are the only states in which this county-based system is used. Data is submitted to the database at Michigan State University, and these validations are run at that location. The triggers are listed in the order that they fire.

County-Based Trap Placement Data

Trigger Name: **TPLACE_CTY_UPD_BEF_ROW**

Table: TEST_PLACEMENT_CTY

Procedures Called: CtyDataPkg.Get_Cty_GM_Sequence_ID

Script File:

Sceolaun -> C:\oracle\trap_triggers\tplace_cty_upd_bef_row.trg

Mothsbane ->D:\trap_files\trap_triggers\tplace_cty_upd_bef_row.trg

ACTION: This trigger fires BEFORE INSERT, UPDATE, OR DELETE on the TEST_PLACEMENT_CTY table for each row that is affected. Variables in the CtyDataPkg package are assigned values from the current row in the TEST_PLACEMENT_CTY table. There are no changes made to the table data in order to avoid the dreaded 'mutating table' error. The variables will be referenced by the next trigger, TPLACE_CTY_INS_UPD_AFT_ROW.

Inserting:

The trigger fires and proceeds through this section when a row is inserted into the TEST_PLACEMENT_CTY table. The function CtyDataPkg.GET_CTY_GM_SEQUENCE_ID assigns the next value from the GM_CTYSEQ sequence to the ID column. Values are assigned to variables from the row being inserted.

The next two 'IF' statements are here because we needed some creativity in loading data from the GPS records. There is limited space in the data entry fields, so we have had to make several of the character positions serve multiple purposes. The first 'IF' statement was used with Trimble Scoutmaster GPS receivers and is no longer used with the Magellan units. I left it in here in case the manufacturers of the Magellan units decide to add a similar feature to their units. The second 'IF' statement is still used to indicate that a trap was placed beyond the edge of the target circle.

First IF statement – (IF v_gpsdata = 'L' THEN…) The column GPS_DATA was added to the TEST_PLACEMENT_CTY table when STS project personnel began to replace opscan forms with GPS units. A 'Y' is inserted into this column to indicate that the record was received from a GPS unit. The Trimble Scoutmasters included a feature that indicated the accuracy of the UTM coordinate. The valid entry types were 'ACU-LOCK', 'UNDER 30', and 'MANUAL'. (The Magellan GPS receivers do not have this feature.)

With the Scoutmaster, the database assumed that 30 'points' were recorded with every set of UTM coordinates. If the trapper stopped the GPS unit before reaching 30 points, he/she could enter the letter 'L' in the 14th position of the GPS record to indicate fewer than 30 points were recorded. The letter 'L' was loaded into the GPS_DATA column. This trigger converts the letter 'L' to the letter 'Y' in the GPS_DATA column and inserts 'UNDER 30' into the ENTRY_TYPE column.

Second IF statement – (IF m_outside_target IS NULL AND v_extravar = '8' THEN…) The column OUTSIDE_TARGET in the TEST_PLACEMENT_CTY table is used to indicate that a trapper placed a trap further than the desired distance from the target location. The trapper enters the letter 'B' in the position after the trap_type code. (This position is also used to report the reason for omitting a site.) The letter 'B' is converted to the number 8 and is inserted into the EXTRA_VAR column of the TEST_PLACEMENT_CTY table. This 'IF' statement sets the OUTSIDE_TARGET column to 'Y' and sets the EXTRA_VAR column to NULL.

Updating:

If a row is being updated, the trigger deletes the rows from the ERRORS table with a value in the ROW_ID column that matches the internal ROWID value of the row being updated in the TEST_PLACEMENT_CTY table. The trigger also updates rows in the ERROR_SUMMARY_CTY table and sets the DATE_OUT column to the SYSDATE where the ID value matches the ID of the row being updated in the TEST_PLACEMENT_CTY table.

Deleting:

If a row is being deleted from the TEST_PLACEMENT_CTY table, the trigger deletes the appropriate rows from the ERRORS table based on the ROW_ID value. It also updates the ERROR_SUMMARY_CTY table and sets the DATE_OUT value to the SYSDATE where the ID matches the ID of the row being updated in the TEST_PLACEMENT_CTY table.

Trigger Name: **TPLACE_CTY_INS_UPD_AFT_ROW**

Table: TEST_PLACEMENT_CTY

Procedures Called: CTYDATAPKG (package variables only)

Script File:

Sceolaun -> C:\oracle\trap_triggers\tplace_cty_ins_upd_aft_row.trg

Mothsbane -> D:\trap_files\trap_triggers\tplace_cty_ins_upd_aft_row.trg

ACTION:

This trigger fires AFTER INSERT OR UPDATE on the TEST_PLACEMENT_CTY table for each row that is affected. Its only action is to assign the internal row id value of the current row in the TEST_PLACEMENT_CTY table to the row_id variable in the CtyDataPkg package.

Trigger Name: **TPLACE_CTY_INS_UPD_AFT**

Table: TEST_PLACEMENT_CTY

Procedures Called: CTYDATAPKG (package variables only)
COUNTY_VALID_CTY
SITE_VALID_CTY
DAY_CHECK_CTY
FORMDATE_LATER_THAN_SCAN_CTY
TRAPPER_CONVERT_CTY
TRAPTYPE_VALID_CTY
COUNTY_BLOCK_SITE_UNIQUE
GRIDUTMS_UNIQUE_CTY
UTMS_UNIQUE_CTY
OMIT_WHY_VALID_CTY
TRAP_DISTANCE_CTY
TRAPS_TOO_CLOSE_CTY

Script File:

Sceolaun -> C:\oracle\trap_triggers\tplace_cty_ins_upd_aft.trg

Mothsbane -> D:\trap_files\trap_triggers\tplace_cty_ins_upd_aft.trg

ACTION:

This trigger fires AFTER INSERT OR UPDATE on the TEST_PLACEMENT_CTY table. A boolean variable, NO_ERRORS, is set to true at the beginning of the trigger. If an error is encountered during the processing of the trigger NO_ERRORS will be set to false and control continues with the next command. If at the end of the trigger NO_ERRORS is still true, a row will be inserted into either the PLACEMENT_CTY or OMITTED_SITES_CTY table based on the value of V_TRAPTYPE. Otherwise, the row will remain in TEST_PLACEMENT_CTY with the appropriate error codes entered into the ERRORS and ERROR_SUMMARY_CTY tables.

The procedural flow of the trigger is as follows:

First, all variables are declared and assigned a data type. Two cursors are defined. The first (SITES_CUR) selects a row from the SITE_LOCATIONS_CTY table based on the values of the variables v_state, m_county, v_block_id, and v_site. It also selects the UTM zone from the QUAD table based on the value in the QUAD column in the SITE_LOCATIONS_CTY table. The second cursor (FIRST_COORDS_ID) searches for previous entries in the FIRST_COORDINATES_CTY table based on the value of the ID column (V_ID).

Following the first BEGIN statement, data values are assigned to the variables from the currently selected record.

The procedure COUNTY_VALID_CTY checks the value in the COUNTY column (M_COUNTY). It must not be null, and it must contain a valid FIPS code from the COUNTY table. Errors are logged in the ERRORS and ERROR_SUMMARY_CTY tables.

If the site number either equals zero or is missing, the procedure SITE_VALID_CTY is called to manage the error.

If there was an error with either the county code or site number, the trigger raises the exception CTY_SITE_ERROR and exits without performing any additional validations.

If both the county code and site number are valid, the trigger continues by opening and fetching from the cursor SITES_CUR. A record will be returned from the SITE_LOCATIONS_CTY table if a matching state, county, block_id, and site are found for the values in the variables v_state, m_county, v_block_id, and v_site. If no matching record is found, the exception NO_SITE_DATA is raised, records are inserted into the ERRORS and ERROR_SUMMARY_CTY tables, and processing ends.

If a record is found in the SITE_LOCATIONS_CTY table with a matching state, county, block_id, and site value processing continues with the following validations.

The variable M_SCAN is filled with the current SYSTEM date that is loaded with the GPS record. The date will be formatted to be compatible with Oracle's default date format (DD-MON-YYYY).

(Note: The term SCAN is used because data was submitted using optical scan forms when the database was created in 1994. GPS units were first introduced into the project in 1996.)

The procedure DAY_CHECK_CTY is called to check for a valid trap placement date. If there are no errors in the placement date, the procedure FORMDATE_LATER_THAN_SCAN_CTY is called. This procedure checks for trap placement dates that are later than the data entry dates (m_scan).

The procedure TRAPPER_CONVERT_CTY is called to search for the trapper's initials in the PEOPLE table. Errors are logged if the trapper's initials are missing. The procedure also compares the agency value in the record from the SITE_LOCATIONS_CTY table to the agency value in the record that is found in the PEOPLE table. if these values do not match an error is logged in the ERRORS and ERROR_SUMMARY_CTY tables with constraint –20002.

The procedure TRAPTYPE_VALID_CTY checks for an entry in the M_TRAPTYPE field. If it is missing, errors are logged in the ERRORS and ERROR_SUMMARY_CTY tables through the NULL_TRAPTYPE exception. The code from the GPS record (D, M, or O) is converted as the record is inserted into the TEST_PLACEMENT_CTY table. The valid values are DELTA, MILK CARTON, and OMIT. If any other code is entered in the GPS record, it is inserted without any conversion into the trap_type field in the TEST_PLACEMENT_CTY table. If invalid values are found, the TRAPTYPE_VALID_CTY procedure will insert records into the ERRORS and ERROR_SUMMARY_CTY tables with the constraint 'CHK_TESTPLACE_TRAPTYPE'.

The procedure COUNTY_BLOCK_SITE_UNIQUE looks for an existing record with the same state, county code, block_id, and site number. If a match is located in either the PLACEMENT_CTY or the OMITTED_SITES_CTY table, a 'duplicate site' error is logged in the ERRORS and ERROR_SUMMARY_CTY tables.

The procedure GRIDUTMS_UNIQUE_CTY looks for an existing grid node utm coordinate pair within the zone where the current record is located. If a match is located in either the PLACEMENT_CTY or the OMITTED_SITES_CTY table, a 'duplicate grid node utms' error is logged in the ERRORS and ERROR_SUMMARY_CTY tables.

The procedure UTMS_UNIQUE_CTY looks for an existing utm coordinate pair within the zone where the current row is located. If a match is located in either the PLACEMENT_CTY or the OMITTED_SITES_CTY table, a 'duplicate utms' error is logged in the ERRORS and ERROR_SUMMARY_CTY tables.

If the value in v_traptype is 'OMIT', the procedure OMIT_WHY_VALID_CTY checks for a valid value from the OMIT_REASONS table. If the variable V_WHY is null then the exception NULL_OMIT_REASON is raised. If the value in the variable V_WHY is not found in the OMIT_REASONS table, errors are logged with constraint –20018 through the exception INVALID_OMIT_REASON.

The procedure TRAP_DISTANCE_CTY reports the distance of the trap from the grid node location. If the trap is located outside of the target circle, the procedure raises the exception OUTSIDE_TARGET. The distance between the target grid node location and the actual trap location is calculated and stored in the variable 'trap_dist' (meters). The distance between the trap location and the edge of the target circle is calculated and stored in the variable 'v_outside_target' (meters). This value will be inserted into the DISTANCE_OUTSIDE column in the PLACEMENT_CTY table.

The procedure TRAPS_TOO_CLOSE_CTY checks the PLACEMENT_CTY table for any traps that have been placed within 100 meters of the current trap. The OMITTED_SITES_CTY table is also checked for any records within 100 meters of the current trap. If any traps or omitted sites are located, a 'traps too close' error is logged in the ERRORS and ERROR_SUMMARY_CTY tables.

A row is inserted into the FIRST_COORDINATES_CTY table in order to store the original utm coordinates that are reported for each site. Since each record is to be stored the first time it enters the database, the cursor FIRST_COORDS_ID is used to search for an existing record. A row is inserted into the FIRST_COORDINATES_CTY table if the ID is not found; otherwise, control continues to the next step.

The error flag NO_ERRORS is checked at this point. If one or more errors were encountered the NO_ERRORS variable is now FALSE. No further action will be taken. However, if NO_ERRORS has remained TRUE, the trigger will perform two additional actions. First, the SITE_LOCATIONS_CTY table will be updated. A 'Y' will be entered into the PLACED column to indicate that the site has been addressed.

Second, a row is inserted into either the PLACEMENT_CTY or OMITTED_SITES_CTY table. The trigger checks the variable V_TRAPTYPE. If the value is 'OMIT', a row is inserted into the OMITTED_SITES_CTY table. If the value is either 'DELTA' or 'MILK CARTON', a row is inserted into the PLACEMENT_CTY table.

Trigger Name: **OMIT_CTY_INSERT_AFT**

Table: OMITTED_SITES_CTY

Procedures Called: No procedures

Script File:

Sceolaun -> C:\oracle\trap_triggers\omit_cty_insert_aft.trg

Mothsbane -> D:\trap_files\trap_triggers\omit_cty_insert_aft.trg

ACTION:

This trigger fires when a row is INSERTED into the OMITTED_SITES_CTY table. It deletes a row from the TEST_PLACEMENT_CTY table if there is an ID value that matches the ID of the row being inserted into OMITTED_SITES_CTY. By coding this action into a trigger, it ensures that the row is inserted into the OMITTED_SITES_CTY table before the row is deleted from TEST_PLACEMENT_CTY.

Trigger Name: **PLACE_CTY_INSERT_AFT**

Table: PLACEMENT_CTY

Procedures Called: No procedures

Script File:

Sceolaun -> C:\oracle\trap_triggers\place_cty_insert_aft.trg

Mothsbane -> D:\trap_files\trap_triggers\place_cty_insert_aft.trg

Action:

This trigger fires when a row is INSERTED into the PLACEMENT_CTY table. It deletes a row from the TEST_PLACEMENT_CTY table if there is an ID value that matches the ID of the row being inserted into PLACEMENT_CTY. By coding this action into a trigger, it ensures that the row is inserted into the PLACEMENT_CTY table before the row is deleted from TEST_PLACEMENT_CTY.

County-Based Trap Inspection Data

Trigger Name: **TINSPECT_CTY_UPD_BEF_ROW**

Table: TEST_INSPECTION_CTY

Procedures Called: CtyDataPkg.Get_Cty_GM_Sequence_ID

Script File:

Sceolaun -> C:\oracle\trap_triggers\tinspect_cty_upd_bef_row.trg

Mothsbane -> D:\trap_files\trap_triggers\tinspect_cty_upd_bef_row.trg

ACTION:

This trigger fires BEFORE INSERT, UPDATE, or DELETE on the TEST_INSPECTION_CTY table. This trigger fires for each row that is affected. Variables in the CtyDataPkg package are assigned values from the current row in the TEST_INSPECTION_CTY table. There are no changes made to the table data in order to avoid the dreaded 'mutating table' error. The variables will be referenced by the next trigger, TINSPECT_CTY_INS_UPD_AFT_ROW.

Inserting:

The trigger fires and proceeds through this section when a row is inserted into the TEST_INSPECTION_CTY table. The function CtyDataPkg.GET_CTY_GM_SEQUENCE_ID assigns the next value from the GM_CTYSEQ sequence to the ID column. Values are assigned to variables from the row being inserted.

The next three 'IF' statements are here because we needed some creativity in loading data from the GPS records. There is limited space in the data entry fields, so we have had to make several of the character positions serve multiple purposes. The first 'IF' statement was used with Trimble Scoutmaster GPS receivers and is no longer used with the Magellan units. I left it in here in case the manufacturers of the Magellan units decide to add a similar feature to their units.

First IF statement – (IF v_gpsdata = 'L' THEN…) The column GPS_DATA was added to the TEST_PLACEMENT_CTY table when STS project personnel began to replace opscan forms with GPS units. A 'Y' is inserted into this column to indicate that the record was received from a GPS unit. The Trimble Scoutmasters included a feature that indicated the accuracy of the UTM coordinate. The valid entry types were 'ACU-LOCK', 'UNDER 30', and 'MANUAL'. (The Magellan GPS receivers do not have this feature.) With the Scoutmaster, the database assumed that 30 'points' were recorded with every set of UTM coordinates. If the trapper stopped the GPS unit before reaching 30 points, he/she could enter the letter 'L' in the 14th position of the GPS data record to indicate fewer than 30 points were recorded. The letter 'L' was loaded into the GPS_DATA column. This trigger converts the letter 'L' to the letter 'Y' in the GPS_DATA column and inserts 'UNDER 30' into the ENTRY_TYPE column.

The second 'IF' statement (ELSIF v_gpsdata IS NULL THEN…) simply ensures that the GPS_DATA column contains a 'Y'.

The third 'IF' statement (ELSIF TO_NUMBER(v_gpsdata) >= 0 THEN…) allows us to use the last two characters in the GPS record (positions 19 and 20) for two purposes. A trapper can record a catch value during a routine trap inspection, or a supervisor can enter a QC failure code during a 'less-than-perfect' quality control inspection. SQL*Loader will not allow one position in the input file to be designated as character data in one case and numeric data in another. So, we assign that position (20) to a character data type and convert to a numeric data type when necessary. There are three possible scenarios:

1. A trapper enters a routine trap catch. The trapper is required to enter three digits for the catch value. The first two digits (positions 18 and 19) are inserted into the CATCH column, and the third digit (position 20) is inserted into the GPS_DATA column. This trigger multiplies the CATCH value by 10, then converts the value in GPS_DATA to a numeric value and adds it to the CATCH value. This final value will be inserted into the CATCH column in the INSPECTION_CTY table.
2. A supervisor enters a passing QC inspection. He/she enters one digit into the CATCH column (position 18) and the letter 'P' into the FIELD_CHECK column (position 19). Position 20 is left blank, and a null value is assigned to the GPS_DATA column.
3. A supervisor enters a failing QC inspection. He/she enters one digit into the CATCH column (position 18), the letter 'F' into the FIELD_CHECK column (position 19), and a QC failure code into the QC_FAIL column (position 20).

Updating:

If a row is being updated, the trigger deletes the rows from the ERRORS table with a value in the ROW_ID column that matches the internal ROWID value of the row being updated in the TEST_INSPECTION_CTY table. The trigger also updates rows in the ERROR_SUMMARY_CTY table and sets the DATE_OUT column to the SYSDATE where the ID value matches the ID of the row being updated in the TEST_INSPECTION_CTY table.

Deleting:

If a row is being deleted from the TEST_INSPECTION_CTY table, the trigger deletes the appropriate rows from the ERRORS table based on the ROW_ID value. It also updates the ERROR_SUMMARY_CTY table and sets the DATE_OUT value to the SYSDATE where the ID matches the ID of the row being updated in the TEST_INSPECTION_CTY table.

Trigger Name: **TINSPECT_CTY_INS_UPD_AFT_ROW**

Table: TEST_INSPECTION_CTY

Procedures Called: CTYDATAPKG (variables only)

Script File:

Sceolaun -> C:\oracle\trap_triggers\tinspect_cty_ins_upd_aft_row.trg

Mothsbane -> D:\trap_files\trap_triggers\tinspect_cty_ins_upd_aft_row.trg

ACTION:

This trigger fires AFTER INSERT OR UPDATE on the TEST_INSPECTION_CTY table for each row that is affected. Its only action is to assign the internal row id value of the current row in the TEST_INSPECTION_CTY table to the row_id variable in the CtyDataPkg package.

Trigger Name: **TINSPECT_CTY_INS_UPD_AFT**

Table: TEST_INSPECTION_CTY

Procedures Called: CTYDATAPKG
DAY_CHECK_CTY
FORMDATE_LATER_THAN_SCAN_CTY
SITE_VALID_CTY
OMITTED_SITE_CTY
PLACED_SITE_CTY
TRAPPER_VALID_CTY
INSPECT_UNIQUE_CTY
UTMS_UNIQUE_CTY
TRAP_DISTANCE_CTY
TRAPS_TOO_CLOSE_CTY

QC_VALID_CTY
VISIT_FINAL_CTY
VISIT_VALID_CTY
TRAP_CONDITION_CTY
CATCH_VALID_CTY
PLACEMENT_UPD_CTY

Script File:

Sceolaun -> C:\oracle\trap_triggers\tinspect_cty_ins_upd_aft.trg

Mothsbane -> D:\trap_files\trap_triggers\tinspect_cty_ins_upd_aft.trg

ACTION:

This trigger fires AFTER INSERT OR UPDATE on the TEST_INSPECTION_CTY table. A boolean variable, NO_ERRORS, is set to true at the beginning of the trigger. If an error is encountered during the processing of the trigger NO_ERRORS will be set to false and control continues with the next command. If at the end of the trigger NO_ERRORS is still true, a row will be inserted into the INSPECTION_CTY table. Otherwise, the row will remain in the TEST_INSPECTION_CTY table with the appropriate error codes inserted into the ERRORS and ERROR_SUMMARY_CTY tables.

The procedural flow of the trigger is as follows:

First, all variables are declared and assigned a data type. Two cursors are defined. The first (SITES_CTY_CUR) selects the ID and AGENCY from the SITE_LOCATIONS_CTY table and the UTM ZONE from the QUAD table for records matching the values of the variables m_state, m_county, m_blockid, and m_site. The second cursor (TOT_CATCH) selects the total_catch value from the PLACEMENT_CTY table where the state, county, block_id, and site match the values of the m_state, m_county, m_blockid, and m_site variables.

Following the first BEGIN statement, data values are assigned to the variables from the currently selected record. If the table is being updated, the ID value is assigned from the o_id variable in the CtyDataPkg package.

The variable M_SCAN is filled with the current SYSTEM date that is loaded with the GPS record. The date will be formatted to be compatible with Oracle's default date format (DD-MON-YYYY).

(Note: The term SCAN is used because data was submitted using optical scan forms when the database was created in 1994. GPS units were first introduced into the project in 1996.)

The procedure DAY_CHECK_CTY is called to check for a valid trap inspection date. If there are no errors in the inspection date, the procedure FORMDATE_LATER_THAN_SCAN_CTY is called. This procedure checks for trap inspection dates that are later than the data entry dates (m_scan).

If the site number either equals zero or is missing, the procedure SITE_VALID_CTY is called to manage the error.

Next open the SITES_CTY_CUR cursor and select the following values into variables: SITE_LOCATIONS.ID, SITE_LOCATIONS.AGENCY, QUAD.ZONE, SITE_LOCATIONS.QUAD. The cursor is closed if no record is returned; otherwise, the PREDETERMINED_SITE variable is set to true.

If there were no errors with either the quad code or site number, then the OMITTED_SITE_CTY procedure is called. This procedure searches the OMITTED_SITES_CTY table for a record matching the state, county, block_id and site values in the variables m_state, m_county, m_blockid, and v_site. There should be no inspections since there are no traps placed at omitted sites. An error will be generated if a row is found in the OMITTED_SITES_CTY table. If there are no matches in the OMITTED_SITE_CTY table, the PLACED_SITE_CTY procedure is called. This procedure searches for a row in the PLACEMENT_CTY table based on the values in the m_state, m_county, m_blockid, and v_site variables. A row with a matching state, county, block_id, and site value must exist in the PLACEMENT_CTY table before a row can be inserted into the INSPECTION_CTY table. An error will be generated if no match is found in the PLACEMENT_CTY table.

The procedure TRAPPER_VALID_CTY is called to search for the trapper's initials in the PEOPLE table. Errors are logged if the trapper's initials are missing. The procedure also compares the agency value in the record from the SITE_LOCATIONS_CTY table to the agency value in the record that is found in the PEOPLE table. If these values do not match an error is logged in the ERRORS and ERROR_SUMMARY_CTY tables with constraint –20002.

The procedure INSPECT_UNIQUE_CTY checks for an existing record in the INSPECTION_CTY table. Only one record with the same state, county, block_id, site, inspection date, and field_check value may be entered. Errors are logged in the ERRORS and ERROR_SUMMARY_CTY tables if a record exists with matching values for all six columns.

The next section was developed for use with the Trimble Scoutmaster GPS receivers. The Scoutmaster units enabled the trapper to record UTM coordinates with various levels of accuracy. The most accurate data entry type is ACU-LOCK, followed by UNDER 30, MANUAL, and OPSCAN. The variable M_REPLACE is originally set to FALSE. The variable M_ENTRYTYPE is assigned the value of the current ENTRY_TYPE column from the TEST_INSPECTION_CTY table. The variable N_ENTRYTYPE is assigned the value of the ENTRY_TYPE column from the row with a matching quad and site value in the PLACEMENT_CTY table. The variable M_REPLACE will be set to TRUE if the value of M_ENTRYTPE is considered to be more accurate than the value of N_ENTRYTYPE. If M_REPLACE is TRUE, the UTM coordinates in the current TEST_INSPECTION_CTY record are checked more closely. The following procedures are called: utms_unique_cty, trap_distance_cty, and traps_too_close_cty. If there are no errors, the UTM coordinates in the PLACEMENT_CTY table are replaced with the UTM coordinates from the current TEST_INSPECTION_CTY record. The ENTRY_TYPE column in the PLACEMENT_CTY record will be updated with the value from the TEST_INSPECTION_CTY record (M_ENTRYTYPE). The valid value for ENTRY_TYPE when entering data from the Magellan units is MAGELLAN. The variable M_REPLACE will remain FALSE, so processing will continue with the call to the QC_VALID_CTY procedure.

The QC_VALID_CTY procedure checks for valid values in the FIELD_CHECK and QC_FAIL columns. The valid values for the FIELD_CHECK column are F, N, P, or NULL. If the value of the FIELD_CHECK column is F, then the QC_FAIL column must contain a valid value from the QC_FAIL_REASONS table. Errors are logged in the ERRORS and ERROR_SUMMARY_CTY tables if these conditions are not met.

If the value of the M_VISIT variable is 'FINAL', the VISIT_FINAL_CTY procedure is called to check for two conditions. The VISIT column must not be NULL, and there must exist only one record for each quad and site with a value of 'FINAL' in the VISIT column and a value of 'N' in the FIELD_CHECK column. A trapper may remove a trap only one time; however, a supervisor may check the site multiple times to be sure the trapper removed the trap. There may exist multiple records with the same state, county, block_id, and site with a value of 'FINAL' in the VISIT column and a value of 'P' or 'F' in the FIELD_CHECK column to indicate a QC inspection.

The procedure VISIT_VALID_CTY checks for a valid value in the VISIT column (V_VISIT). Valid entries are 'MIDSEASON' or 'FINAL', and the column may not be null. The procedure also checks the value in the DAY column (V_DAY) to ensure that all midseason inspections occur prior to a final inspection. It is assumed that the trap is removed from the field at the earliest final visit. Errors are logged in the ERRORS and ERROR_SUMMARY_CTY tables.

The next section regarding DEPTH and EXTRA VARIABLES has been commented out. This section was used for a short time to help trappers record moth catches in areas where there were very high gypsy moth populations. The trapper would install milk carton traps and measure the depth of the moths in the traps instead of counting individual moths. The formula could then be applied to convert the depth to a catch value. This method of recording trap catch values is no longer used. I kept it in the trigger in case it needs to be revived at some point.

The procedure TRAP_CONDITION_CTY checks for a valid value in the CONDITION column (M_TRAP_COND). The column many not be null, and the valid values are: GOOD, DAMAGED, INACCESSIBLE, MISSING. The database assumes that a trapper can physically touch a trap and look inside the trap for moths when the condition is reported as 'GOOD' or 'DAMAGED'. If a trapper returns to a trap site for an inspection and is unable to either get access to the trap location or is unable to find the trap, the condition should be entered as 'MISSING' or 'INACCESSIBLE'. The condition should also be entered as 'MISSING' if the trap is so severely damaged that moths cannot be counted. Errors are logged in the ERRORS and ERROR_SUMMARY_CTY tables if the CONDITION column is NULL or if any other values are entered.

If there are no errors in the CONDITION column, then the procedure CATCH_VALID_CTY is called. This procedure checks for the following situations:

1. If the value of CONDITION (V_TRAP_COND) is 'MISSING' or 'INACCESSIBLE', then CATCH (M_CATCH) must be NULL. (The trapper cannot count moths if he/she cannot look inside of the trap.)
2. If the value of CONDITION (V_TRAP_COND) is 'GOOD' or 'DAMAGED', then CATCH (M_CATCH) must be greater than or equal to zero. (The trapper must report a moth catch of zero or greater.)

Errors are logged in the ERRORS and ERROR_SUMMARY_CTY tables.

If at this point the error flag (NO_ERRORS) remains true, then each of the column values are assigned to a table record. This record is inserted into the INSPECTION_CTY table.

The procedure PLACEMENT_UPD_CTY is called to update the parent row in the PLACEMENT_CTY table. The first IF statement will be ignored since M_REPLACE will be FALSE while the Magellan GPS units are in use. If M_REPLACE is TRUE, the UPDATE statement will set the UTM coordinates to the values from the inspection record (M_UTME and M_UTMN). It will also set the ENTRY_TYPE column to the value in M_ENTRYTYPE from the inspection record, and it will update the DISTANCE_OUTSIDE column with the figure that was calculated using the UTM coordinates in the inspection record (V_OUTSIDE_TARGET).

The TOTAL_CATCH value is updated in the PLACEMENT_CTY table based on the previous TOTAL_CATCH value (S_CATCH) in the PLACEMENT_CTY record and the CATCH value in the new inspection record (V_CATCH). The TOTAL_CATCH value in the PLACEMENT_CTY table is indicated as follows:

-2 = no inspection (default value when row inserted into PLACEMENT table)

-1 = site inspected by trap was missing/inaccessible at all inspections

0+ = total moth catch for the site

The TOTAL_CATCH column value will be updated as follows:

1. Change TOTAL_CATCH from –2 to –1 only if this is the first inspection and the trap is missing/inaccessible.
2. Recalculate TOTAL_CATCH only if the catch value is > 0. If TOTAL_CATCH = -1 or –2, set TOTAL_CATCH to 0 before adding new catch value. (A null catch value indicates a missing/inaccessible trap.)

Trigger Name: **INSPECT_CTY_INSERT_AFT**

Table: INSPECTION_CTY

Procedures Called: None

Script File:

Sceolaun-> C:\oracle\trap_triggers\inspect_cty_insert_aft.trg

Mothsbane -> D:\trap_files\trap_triggers\inspect_cty_insert_aft.trg

ACTION:

This trigger fires AFTER a row is INSERTED into the INSPECTION_CTY table. It deletes a row from the TEST_INSPECTION_CTY table if there is an ID value that matches the ID of the row being inserted into the INSPECTION_CTY table.