ABSTRACT

One of the tasks of the analysis division at the bank is to keep track of new account referrals made by bank representatives based upon location, sales territory and manager. This is made more difficult because over time representatives may change their sales territory, no longer be active, report to different managers or more than one manager, or their reporting entities may change over time. The reporting requirements include being able to report on all activity based upon current sales territory and manager as well as to report on activity based upon sales territory and manager at the time the referral was made. Referrals by sales territory need to be able to report all referrals in that territory over a period of time by all representatives as well as only those sales made by currently active representatives. To handle these requirements, a slowly changing dimension table is created as part of a data warehouse. This table is maintained using the SAS hash object to reduce processing time. Reports are produced using SAS BI tools including OLAP cubes and web report studio. This paper demonstrates the use of the SAS hash object to maintain the table and provides examples of reporting techniques.

INTRODUCTION

Farm Bureau Bank was founded in 1999 to serve the membership of Farm Bureau Associations throughout the country providing retail banking services. Marketing to its customer base of Farm Bureau members is carried out predominantly through direct mail, referrals by participating states’ Farm Bureau Insurance agents and internet marketing. Farm Bureau Insurance agents of participating states throughout the United States provide bank product referrals to their customers. Currently agent referrals provide approximately 90% of the new account applications with the remainder coming from direct mail or the internet. Agents are compensated for their referrals and there are incentives for the agents based on their account production. One of the responsibilities of the Finance/analytics area is to track the referral production of these insurance agents and provide reports to the agents, their managers and sales directors. The area is also responsible for calculating, reporting and paying commissions. Both referral and commission reports are distributed to members throughout the country using SAS BI allowing flexibility in terms of method and type of report distribution.

Accurate and timely reporting of account referrals is critical to maintaining the agents’ willingness to refer business to the bank. This is made more difficult due to the structure of the agent relationships within the state. The state Farm Bureau Associations are independent entities with each Association having its own organizational structure. Some states structure their agents based upon counties, with each county having one or more agencies within the county, with each agency manager reporting to a county agency manager, and then going up to a district manager and state manager. Other states may have more than one county reporting to a manager and then having no district. As well as the different reporting structures, there is the usual churn among the agencies and managers. Over time agents may change counties, become agency managers, or become inactive. The reporting relationship may change as states redraw their districts, combine reporting entities or revamp their entire reporting structure.

Referral and commission reports must be able to deal with the continuing changes in the agent reporting relationship. In addition, because the number of referrals may influence the compensation of agency and district managers, accurate, timely and flexible reporting is necessary. For example, at times the total number of referrals by agent may be requested, independent of the agency or county associated with the agent. Other report requests may be based on the number of referrals by county, with the number of referrals made by agents during the assignment to the county, irrespective of whether they are currently active or currently assigned to the county. These same issues arise for referral reporting related to agency and district managers.

To handle these requirements a data warehouse was developed with an application fact table and a number of different dimensions including an agent dimension. The agent dimension is a type 2 slowly changing dimension that allows for tracking any changes in the agent status and reporting relationship. This allows us the flexibility necessary for providing the various reports that may be requested. To reduce processing time and provide the functionality needed to maintain the agent dimension a SAS hash object was used. Reports are produced using SAS BI tools including OLAP cubes and Web Report Studio.
DATA WAREHOUSE

In building our data warehouse we use the basic ETL process. We extract application data from the various databases and input channels, transform and clean the data to conform to bank and regulatory standards then load the information into our Application fact table and assorted dimension tables.

AGENT ROSTER – A SLOWLY CHANGING DIMENSION

The key to being able to produce the desired reports is the Agent Roster table. This table is structured as a slowly changing dimension. This allows us to track changes in the agent location or reporting structure. Below is an example of the type of information stored in our agent dimension table. This is only a subset of the demographic and internal/external account data we track. For demonstration purposes we will use this small dataset to walk you through the Agent Dimension updating process.

<table>
<thead>
<tr>
<th>Target</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column Name</td>
<td>Type</td>
</tr>
<tr>
<td>Agent_Dim_Key</td>
<td>Int</td>
</tr>
<tr>
<td>RSC</td>
<td>Char</td>
</tr>
<tr>
<td>RIM_NO</td>
<td>Num</td>
</tr>
<tr>
<td>DIST_NAME</td>
<td>Char</td>
</tr>
<tr>
<td>COUNTY_NAME</td>
<td>Char</td>
</tr>
<tr>
<td>DEP_ACCT</td>
<td>Char</td>
</tr>
<tr>
<td>FULL_NAME</td>
<td>Char</td>
</tr>
<tr>
<td>Row_Eff_Date</td>
<td>Num</td>
</tr>
<tr>
<td>Row_Exp_Date</td>
<td>Num</td>
</tr>
<tr>
<td>Current_ind</td>
<td>Int</td>
</tr>
</tbody>
</table>

Figure 1. Agent Dimension Table Structure

The starting point in the creation of all of our dimension and fact tables is good documentation. Above is a Target/Source table for our Agent Roster dimension table. It gives some of the basic column properties of the dataset as well as details specific to the ETL process. For example, SCD Type provides information on the type of slowly changing dimension for each variable. We also store information on the database and table(s) our source data is coming from as well as rules about how each variable is updated or replaced. Having these tables as a starting point in our data warehouse design has saved countless hours during the creation and debugging of our ETL programs. (RSC is the referral source code which is a unique identifier assigned by the bank to referring agents).

STEP 1: CREATING WORK TABLES WITH AGENT_ROSTER AND UPDATE TABLES

The first steps in the program copy the Agent_Roster from the permanent library into the work library and pull data from the banking system to create the roster update table (Agent_Roster_UPD) in the work library. You will notice that in the Agent_Roster_UPD program we rename some of the variables by adding a T_ to the beginning. This is done in order to facilitate comparison of fields in the current Agent_Roster to the Agent_Roster update table (step 3). We have highlighted the differences between the current and update tables.

We track any changes or new agents with a change log table. Reports are created using this table to allow other bank departments to verify that any changes they have made to the agent record in the banking system have been
entered correctly. Any changes to the records in the agent table, either a new record or a changed record, are tracked in the change log file.

### Agent_Roster Table Before Updating

#### proc sql;
```sql
create table Agent_Roster_UPD as
select rsc,
      rim_no,
      full_name as T_full_name,
      dep_acct as T_dep_acct,
      dist_name as T_dist_name,
      county_name as T_county_name
from (Banking System);
quit;
```

#### data Agent_Roster_UPD;
```sql
set Agent_Roster_UPD;
length rim_rsc $15.;
rim_rsc=cats(rim_no,rsc,'0');
run;
```

1. Pull the fields from the banking system and change the names as needed.
2. Create a unique variable rim_rsc based upon the RIM and RSC fields; a ‘0’ is appended to the field which indicates a current record.

### Agent_Roster_UPD Table with Differences Highlighted

**STEP 2: ADDING A NEW RECORD INTO THE AGENT DIMENSION USING A HASH TABLE**

The first task in keeping the Agent dimension up to date is to check for new agents. A table (Agent_Roster_UPD) containing all of the current agents and their information is pulled from the banking system. The Agent_Roster table is then loaded as a hash table and the Agent_Roster_UPD table is checked against this table using the Rim_RSC as the key. If the Rim_RSC is not found in the Agent_Roster then a record is created in the table New_Agents and a record is created in the change log with the information about the new record. The record is then added into the Agent_Roster table.
data work.Changelog_New (keep= full_name affected_table source_table change_date change_reason rim_rsc change_date)
work.New_Agents (keep=rim_rsc);
length _change_reason table_name $40. affected_table source_table action $15.;
if 0 then set work.Agent_Roster;
format change_date mmddyy8.;
if _n_ =1 then do;
declare hash rimhash (DATASET: 'WORK.Agent_Roster', ORDERED: 'A', HASHEXP: 16
rc=rimhash.defineKey ( 'rim_rsc');
rc=rimhash.defineData ( 'rim_rsc', 'full_name');
rc=rimhash.defineDone();
end;
affected_table 'Agent Dimension';
change_date=date();
change_date= 'Create Record';
source_table= 'RSC_Acct';
full_name= '';
do while (not done);
set WORK.Agent_Roster_UPD end=done;
rc=rimhash.find();
if rc ne 0 then do;
change_reason= 'New Record';
output work.Changelog_New;
output work.New_Agents;
end;
end;
stop;
run;

3. Define the hash table, the key and the data desired to be returned from the table
4. Check to see if the records in the Agent_Roster_UPD are in the agent_roster table. If not, create a new record and put a notation in the change log.

<table>
<thead>
<tr>
<th>CHANGE_REASON</th>
<th>AFFECTED_TABLE</th>
<th>SOURCE_TABLE</th>
<th>ACTION</th>
<th>RIM_RSC</th>
<th>FULL_NAME</th>
<th>CHANGE_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Record</td>
<td>Agent Dimension</td>
<td>RSC_Acct</td>
<td>Create Record</td>
<td>1000087000080</td>
<td>Fritz Pareto</td>
<td>10/25/2011</td>
</tr>
</tbody>
</table>

Figure 4. Records Created in Changelog_New and New_Agent tables

**STEP 3: CHECKING FOR CHANGED INFORMATION USING A HASH TABLE - OVERWRITE**

The next step is to check if any of the fields in the agent table have been changed since the last update of the agent record. Some of the fields result in new records being created (Step 4) if they are different, and other fields are overwritten. Based on the Target/Source table we know that the full_name and dep_acct variables are Type 1. This means that if a change is detected in either of these variables, the fields are overwritten with the new data. Shown below is the code that is used to update the records for fields that are overwritten. (For brevity we have included only the Dep_Acct variable)

data work.Changelog_Overwrite (keep=rim_no rsc full_name affected_table source_table change_date orig_char new_char change_rsn state rim_rsc action);
length orig_char new_char change_rsn $40. affected_table source_table action $15.;
format change_date mmddyy8.;
if 0 then set work.Agent_Roster;
if _n_ =1 then do;
```
declare hash rimhash(DATASET:'work.Agent_Roster', ORDERED:'A');
rc=rimhash.defineKey('rim_rsc');
rc=rimhash.defineData('rim_rsc','rsc','rim_no','full_name','dist_name','county_name','
dep_acct','current_ind');
rc=rimhash.defineDone();
end;
affected_table='Agent Dimension';
Change_date=date();
Source_table='RM_ACCT';
action='Update Record';
do while (not done1);
set work.Agent_Roster_UPD  end=done1;
rc=rimhash.find() ;
if rc = 0 then do;
  if T_dep_acct ne dep_acct then do;
    orig_char=strip(dep_acct);
    new_char=strip(T_dep_acct);
    change_rsn='change in dep_acct';
    output work.Changelog_Overwrite;
    dep_acct=T_dep_acct;
  end;
  RC=rimhash.replace();
end;
rc=rimhash.output(dataset:'Agent_Roster2'); /* need to output hash table*/
stop;
run;
```

1. This data is used to track the change information that is entered into the change log
2. This will replace the any changed fields with the updated fields.

<table>
<thead>
<tr>
<th>ORIG_CHAR</th>
<th>NEW_CHAR</th>
<th>CHANGE_RSN</th>
<th>AFFECTED_TABLE</th>
<th>SOURCE_TABLE</th>
<th>ACTION</th>
<th>FULL_NAME</th>
<th>RIM_NO</th>
<th>RSC</th>
<th>RIM_RSC</th>
<th>CHANGE_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>500005</td>
<td>600001</td>
<td>change in dep_acct</td>
<td>Agent Dimension</td>
<td>RM_ACCT</td>
<td>Update Record</td>
<td>Bill Phillips</td>
<td>100005</td>
<td>700005</td>
<td>1000057000050</td>
<td>10/25/2011</td>
</tr>
</tbody>
</table>

**Figure 5. Changelog_Overwrite and Agent_Roster2 tables**

**STEP 4: CHECKING FOR CHANGED INFORMATION USING A HASH TABLE – PARTITION**

The next step is to check if any of the fields that would result in a new agent record being created have been changed since the last update of the agent record. This step shows the code used to partition Type 2 variables. Whenever there is a change in these variables a new record is created in a temporary table and will be added to the dimension table in Step 6.

```
data WORK.Changelog_Partition (keep=rim_no rsc full_name affected_table source_table
change_date orig_char new_char change_rsn state rim_rsc action)
WORK.Partition_Rec (keep=rsc rim_no full_name city dist_name county_name dep_acct
row_eff_date row_exp_Date current_ind rim_rsc agent_dim_key);
  if 0 then set work.Agent_Roster2;
```
length orig_char orig_char new_char change_rsn table_name $40.;
format change_date mmdy8.;
if _n_ =1 then do;
   declare hash rimhash(DATASET:'work.Agent_Roster2', ORDERED:'A');
   rc=rimhash.defineKey('rim_rsc');
   rc=rimhash.defineData('rim_rsc','rsc','rim_no','full_name','dist_name','county_name', 'dep_acct','agent_aim_key');
   rc=rimhash.defineDone();
end;
affected_Table='Agent Dimension';
change_date=date();
action='Replace Record';
source_table='RSC_ACCT';
do while (not done);
set WORK.Agent_Roster_UPD end=done;
rc=rimhash.find() ;
replace=0;
if rc = 0 then do;
   if T_county_name ne county_name then do;
      orig_char= county_name;
      New_char=T_county_name;
      change_rsn='change in county_name';
      output work.changelog_Partition;
      replace=1;
      county_name=T_COUNTY_NAME;
   end;
   if replace =1 then do;
      row_eff_date=Today();
      current_ind=0;
      output work.Partition_rec;
   end;
end;
end;
stop;
run;

<table>
<thead>
<tr>
<th>RIM_NO</th>
<th>RSC</th>
<th>RIM_RSC</th>
<th>FULL_NAME</th>
<th>ORIG_CHAR</th>
<th>NEW_CHAR</th>
<th>CHANGE_RSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>700001</td>
<td>1000017</td>
<td>Ken Arrow</td>
<td>Friedman</td>
<td>Friedman</td>
<td>change in COUNTY_NAME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AFFECTED_TABLE</th>
<th>SOURCE_TABLE</th>
<th>ACTION</th>
<th>CHANGE_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent Dimension</td>
<td>RSC_ACCT</td>
<td>Replace Record</td>
<td>10/25/2011</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RIM_NO</th>
<th>RSC</th>
<th>RIM_RSC</th>
<th>FULL_NAME</th>
<th>DEP_ACCT</th>
<th>DIST_NAME</th>
<th>COUNTY_NAME</th>
<th>CURRENT_IND</th>
<th>AGENT_DIM_KEY</th>
<th>ROW_EFF_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>100001</td>
<td>700001</td>
<td>1000017</td>
<td>Ken Arrow</td>
<td>500001</td>
<td>District 1</td>
<td>Friedman</td>
<td>0</td>
<td>100</td>
<td>10/25/2011</td>
</tr>
</tbody>
</table>

Figure 6. Change Log and Partition Record

**STEP 5: ADVANCING CURRENT INDICATOR FOR PARTITIONED RECORDS**

In this step the current indicator is incremented for any agent records for which a changed field causes a new record to be created. The variable current_ind is used to track changes in an Agent’s record. If current_ind = 0 then this is the most recent record. Therefore it is necessary to advance the current indicator for all entries related to the partitioned records. We also add a row expiration date variable in order to have a record of when the information in a partitioned entry was changed.

data work.partition;
set work.changelog_partition (keep= rim_no rsc);
run;
proc sort data=WORK.Agent_Roster2; by rim_no rsc; run;
proc sort data=work.partition nodupkey; by rim_no rsc; run;

data work. Agent_Roster3;
merge work.partition (in=a) WORK.Agent_Roster2 (in = b);
by rim_no rsc;
if a and b then do;
if current_ind=0 then row_exp_date=today()-1;
Current_ind=Current_ind+1;
rim_rsc=cats(rim_no,rsc,current_ind);
end;
run;

1. This will update the current indicator by 1 for any record that will be replaced. It also adds the current indicator to the Rim_RSC, keeping that field as a unique field.

<table>
<thead>
<tr>
<th>RIM_NO</th>
<th>RIM_RSC</th>
<th>COUNTY_NAME</th>
<th>CURRENT_IND</th>
<th>AGENT_DIM_KEY</th>
<th>ROW_EXP_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>700001</td>
<td>100001</td>
<td>Keynes</td>
<td>0</td>
<td>100</td>
<td>10/25/2011</td>
</tr>
<tr>
<td>700002</td>
<td>100002</td>
<td>Keynes</td>
<td>0</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>700003</td>
<td>100003</td>
<td>Keynes</td>
<td>0</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>700004</td>
<td>100004</td>
<td>Keynes</td>
<td>0</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>700005</td>
<td>100005</td>
<td>Friedman</td>
<td>0</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>700006</td>
<td>100006</td>
<td>Friedman</td>
<td>0</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>700007</td>
<td>100007</td>
<td>Friedman</td>
<td>0</td>
<td>106</td>
<td></td>
</tr>
</tbody>
</table>

Figure 7. Agent_Roster3 Table

STEP 6: CREATING NEW AGENT KEY FOR NEW RECORDS

In this next to last step of the Agent Dimension update process we are adding the Agent Keys to any new records in the dimension table. The highlighted section shows the hash method used to find the last key in the dataset. Once this is located the Keycount is advanced by 1 for all new and partitioned entries.

data work.New_Partitioned (drop = Keycount rc);
Length Agent_Dim_Key 8. ;
/******************** Hash Object to Get Last foreign Key ********************/
retain Keycount;
if _n_ =1 then do;
Declare hash for_key(DATASET:'WORK.Agent_Roster3', ORDERED:'A');
rc=for_key.defineKey('Agent_Dim_Key');
rc=for_key.defineData('Agent_Dim_Key');
Declare hiter hi_for_key('for_key');
rc=for_key.defineDone();
rc=hi_for_key.last(); /* Get last key in the table */
Keycount=Agent_Dim_Key;
end;
/******************** End of Hash Object to Get Last foreign Key *****************/
set work.partition_rec work.New_Agents;
Keycount=keycount+1;
Agent_Dim_Key=keycount;
run;
1. Set up the iterative hash table to determine the last key in the current table.

<table>
<thead>
<tr>
<th>AGENT_DIM_KEY</th>
<th>RIM_RSC</th>
<th>FULL_NAME</th>
<th>DIST_NAME</th>
<th>COUNTY_NAME</th>
<th>DEP_ACCT</th>
<th>CURRENT_IND</th>
<th>ROW_EFF_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>1000017000010</td>
<td>Ken Arrow</td>
<td>District 1</td>
<td>Friedman</td>
<td>500001</td>
<td>0</td>
<td>10/25/2011</td>
</tr>
<tr>
<td>108</td>
<td>1000087000080</td>
<td>Fischer Black</td>
<td>District 1</td>
<td>Friedman</td>
<td>600000</td>
<td>0</td>
<td>10/25/2011</td>
</tr>
</tbody>
</table>

**Figure 8. Agent_Roster3 Table**

**STEP 7: CREATING NEW AGENT_ROSTER TABLE**

The final step adds the new and partitioned records with the updated key and the rim_rsc to the full dataset and our Agent Roster dimension table has been updated. The highlighted cells show the data and/or records that have been overwritten or partitioned.

```sql
data dataware.Agent_Roster;
set work.Agent_Roster3 work.New_Partitioned;
format row_eff_date row_exp_date mmddyy8.;
by rsc  current_ind;
run;
```

<table>
<thead>
<tr>
<th>RSC</th>
<th>FULL_NAME</th>
<th>DEP_ACCT</th>
<th>COUNTY_NAME</th>
<th>CURRENT_IND</th>
<th>AGENT_DIM_KEY</th>
<th>ROW_EXP_DATE</th>
<th>ROW_EFF_DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>700001</td>
<td>Ken Arrow</td>
<td>500001</td>
<td>Friedman</td>
<td>0</td>
<td>107</td>
<td>10/24/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700001</td>
<td>Ken Arrow</td>
<td>500001</td>
<td>Keynes</td>
<td>1</td>
<td>100</td>
<td>10/24/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700002</td>
<td>Anna Schwartz</td>
<td>500002</td>
<td>Keynes</td>
<td>0</td>
<td>101</td>
<td>09/01/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700003</td>
<td>Frank Edgeworth</td>
<td>500003</td>
<td>Keynes</td>
<td>0</td>
<td>102</td>
<td>09/01/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700004</td>
<td>Irving Fisher</td>
<td>500004</td>
<td>Keynes</td>
<td>0</td>
<td>103</td>
<td>09/01/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700005</td>
<td>Bill Phillips</td>
<td>600001</td>
<td>Friedman</td>
<td>0</td>
<td>104</td>
<td>09/01/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700006</td>
<td>Gene Slutsky</td>
<td>500006</td>
<td>Friedman</td>
<td>0</td>
<td>105</td>
<td>09/01/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700007</td>
<td>Fritz Pareto</td>
<td>500007</td>
<td>Friedman</td>
<td>0</td>
<td>106</td>
<td>09/01/2011</td>
<td>09/01/2011</td>
</tr>
<tr>
<td>700008</td>
<td>Fischer Black</td>
<td>600000</td>
<td>Friedman</td>
<td>0</td>
<td>108</td>
<td>10/25/2011</td>
<td>10/25/2011</td>
</tr>
</tbody>
</table>

**Figure 9. Agent roster table with changed and new records**

**STEP 8: BI REPORTING**

The agent dimension is used with the application fact table to create an OLAP cube for reporting. A referred application will have an RSC associated with it and this is used to establish the link with the application dimension and the agent dimension key. Once the link between the application fact table and the agent dimension, it is simple to create the reports desired. An OLAP cube is created with dimension that allows drilling down from state to county to agent. With the link between the fact table and the agent based upon the location at the time of the application, reports will show multiple locations for the agents based upon their location at time of the application. An example of this report is shown below. In addition any new agents and their applications will be shown.

<table>
<thead>
<tr>
<th>Month</th>
<th>Applications</th>
<th>Approved</th>
<th>Declined</th>
<th>Applications</th>
<th>Approved</th>
<th>Declined</th>
<th>Applications</th>
<th>Approved</th>
<th>Declined</th>
<th>Applications</th>
<th>Approved</th>
<th>Declined</th>
<th>Applications</th>
<th>Approved</th>
<th>Declined</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KEYNES</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ken Arrow</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anna Schwartz</td>
<td>13</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Irving Fisher</td>
<td>9</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FRIEDMAN</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ken Arrow</td>
<td>5</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bill Phillips</td>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gene Slutsky</td>
<td>2</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fritz Pareto</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fischer Black</td>
<td>2</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 10. Application Report based upon location of agent at time of application**
A report can also be created based upon the current location of the agent irrespective of the location of the agent at the time of the application. Selecting all of the applications of the agent and then basing the agent location on the current location of the agent results in the creation of this report.

<table>
<thead>
<tr>
<th>Month</th>
<th>Application_Status</th>
<th>September</th>
<th></th>
<th>October</th>
<th></th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Approved</td>
<td></td>
<td>Approved</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KEYNES</td>
<td>Anna Schwartz</td>
<td>13</td>
<td>15</td>
<td>13</td>
<td>15</td>
<td>58</td>
</tr>
<tr>
<td></td>
<td>Irving Fisher</td>
<td>9</td>
<td>11</td>
<td>8</td>
<td>3</td>
<td>31</td>
</tr>
<tr>
<td>FRIEDMAN</td>
<td>Kon Armstrong</td>
<td>5</td>
<td>6</td>
<td>10</td>
<td>4</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Bill Phillips</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td>Gene Slutsky</td>
<td></td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Fritz Pereto</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Fischer Black</td>
<td></td>
<td></td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>0</td>
<td>13</td>
<td>49</td>
<td>36</td>
<td>98</td>
</tr>
</tbody>
</table>

Figure 11. Application Report based upon current location of agent

CONCLUSION

The hash tables make it possible to easily check for any changes in fields and determine the last key in a current table. Because the hash tables are in memory, the hash tables also result in decreased processing time. Structuring the agent dimension table with the type 1 and type 2 variables provides the ability to easily create the desired reports. Reports created from the change log table provide other departments of the bank with information concerning new and changed records resulting in easy validation of any changes.

REFERENCES


CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: James Beaver
Enterprise: Farm Bureau Bank
Address: 17300 Henderson Pass
City, State ZIP: San Antonio, TX
Work Phone: 210-637-4809
E-mail: jbeaver@farmbureaubank.com

Name: Tobin Scroggins
Enterprise: Farm Bureau Bank
Address: 17300 Henderson Pass
City, State ZIP: San Antonio, TX
Work Phone: 210-637-4809
E-mail: tscroggins@farmbureaubank.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.

Other brand and product names are trademarks of their respective companies.