Interfacing Oracle Human Resources Data to ADP PC Payroll

Susan E. Patterson
Enturia, Inc.

Introduction

The following information provides users of Oracle's HR module with a method for interfacing employee data to ADP for payroll processing. Using Standard Edition BI tools and a PL/SQL editor, we successfully accomplished this task. Today, we share our success and failure with the OAUG community. We hope this information will be beneficial to those considering an ADP interface for their own organization.

The Oracle HR Implementation and Pay Data

Organizations are often constrained from full implementation of the Oracle Human Resources module by issues such as labor, time, and money. Implementing HR and/or Payroll is no small task, and because of that, the project is often divided in to phases. In our first phase we had to be creative about which fields would remain solely in Oracle, and which would transfer to ADP through some type of interface. For the items that needed to be interfaced to ADP, the question was, 'Who will build the interface and how much will that cost?'

ADP provided a quote of $65,000.00, consisting of $54,000.00 to write the interface, and roughly $10,000.00 annually in additional monthly processing fees. Prior to transferring to IT, my function was to process payroll for our organization. Occasionally, I would process payroll outside of the system, in an Excel spreadsheet, and import it to ADP. When I realized that the interface ADP offered was nothing more than a very pricy version of the same, I volunteered to try and construct the interface myself.

We started with a document, provided by the implementation project manager, called 'The Field of Dreams.' This contained all the fields that were going to reside in Oracle and would need to be interfaced to ADP on a regular basis for payroll processing. There were two key pieces of documentation, and two key pieces of software utilized in the production of our interface:

Documentation
- *ADP Interface Implementation Guide*, provided by Oracle
- *ADP Integration Manual*, provided by ADP

Software
- *Discoverer Standard Edition 10g*, an Oracle reporting tool
- *TOAD*, a PL/SQL Editor for Oracle.

When I first began studying the HR tables, I noticed several with names preceded by 'ADP_.' I thought I had struck gold. I immediately recognized these tables as being

identical to the tables in ADP's Report Smith tool. While these tables did prove very helpful, we were not able to use those tables. There are restrictions and limitations to the ADP views that required us to join them with other tables, such as per_all_people_f, per_all_assignments_f, and per_all_periods_of_service, among others. We joined all of these tables to suit our needs, and then created a new 'Business Area' in the Discoverer application to begin assembling the interface. By creating a new business area specifically for these items, we increased data security. This business area is only accessible to those authorized to view payroll data.

The interface build progressed until we reached the topic of deductions. It became painstakingly obvious that we were not going to be able build the interface in Discoverer. The reason is, deduction data and earnings data are in the same fields in Oracle, but must be imported separately to ADP. Additionally, there was no way to pull all the types of deduction codes, amounts, and factors without a union. A major drawback to Discoverer is the inability to union data, or to apply a condition to only a single column. With three weeks left until the project deadline, we were faced with the bitter task of starting the entire interface over using our PL/SQL tool, TOAD.

Writing the Interface – Take Two

Our first concern was our new found knowledge of the benefits configuration. There is no field that says 'Deduction Code.' What we did see was a person, tied to an assignment, which was tied to a plan, which may or may not have options, and probably includes elements. The data needed for deductions, earnings, and other benefits can be found at any or all of these levels. We needed to decide where we were going to store the values that tell our system that, for example, '6' means Health PPO, or 'C' means Car Allowance. There were three options: Hard code these values in the interface, use the DFF fields in the Oracle forms, or set up validation tables within Oracle to translate the fields. The easiest method for our users, and for building the interface in Discoverer, was to use the Oracle provided DFFs.

Our developer began to construct views with the necessary unions for separating benefit data. In addition to the seeded ADP views in Oracle, our developer created some views that are the primary source for the interface: xxhr_adp_all_v, xxhr_adp_ben_v, and xxhr_adp_no_ben_v. All of these views contain the same fields, but are restricted by a union to serve a specific purpose. For example, the xxhr_adp_ben_v view contains all the data needed to transmit benefit data; xxhr_adp_no_ben_v includes data that is not related to benefits. The xxhr_adp_all_v is a combination of the xxhr_adp_ben_v and xxhr_adp_no_ben_v views. The views were put together this way because the data is tied to the person record. In the xxhr_adp_ben_v view, the query will return a line for every benefit a person has. So if Jane Doe has medical, dental, 401(k), and life insurance, Jane Doe will appear four times in the query. For this reason, benefit data and non benefit data are transmitted separately, and are compiled from separate views.

Once the views were arranged in this way, we imported them to a custom folder in the Discoverer Administrative Edition called 'ADP INTERFACE,' along with all the seeded ADP views in Oracle, as well as the tables per_all_assignments_f, per_all_people_f, and per_all_periods_of_service.

To meet the requirements of ADP's data standards, we aliased each field with the naming convention required for import to ADP. Because we chose to use DFFs for storing our benefit data, and since DFFs are stored under very generic names in the database such as attribute1, attribute2, etc… using an alias proved very helpful. For example 'ass_attribute1' was aliased as 'File No.' Skipping this step isn't detrimental, as column names can easily be edited with a double click in Discoverer; however, for ease of development and maintenance, it's helpful to find the fields correctly named and formatted. While I may know what the database name means, the interface is designed for our functional staff, and needs to be a user-friendly tool.

Once the views were imported to the Discoverer Administrative tool as a custom folder, or 'Business Area,' we began assembling the interface. Because of reporting requirements, and the behavior of Discoverer, we created seven worksheets in our interface workbook to present all our data in the correct format.

The ADP Interface Workbook

New Hires

The first sheet in our workbook is the New Hire sheet, which we have named 'nh.' Later, I will explain why the sheets are named this way. The new hire sheet is built off of the view we created called xxhr_adp_no_ben_v. This sheet is strictly for transmitting data on newly hired employees. Because subsequent sheets will contain data about these employees, it is imperative that the basic new hire information is transmitted first. For the ease of our end-users, I assembled the sheets in the order they need to be imported to ADP.

The requirements for new hire data are clearly mapped in the *ADP Integration Manual*. All sheets imported to ADP must start with 'Co Code' and 'File No.' By clicking on the 'Edit Sheet' icon in Discoverer, we can see which fields from our view are included for the New Hire worksheet; there are ten fields total, with eight fields listed in the selected fields list. This is because two of our fields are actually calculated fields. The hire status (active, leave, etc…) is a calculated field because it requires a decode statement. Oracle stores employees on leave-of-absence as an 'S,' but ADP needs to see them as an 'L.' We have a decode statement articulating that we wish to transmit 'S' as 'L' and items that are not 'S' as their default status code:

```
DECODE(Status,'S','L',Status)
```

The second calculated field is the Rate 1 Amount. Every employee needs to come in to ADP with a rate, but we wanted to put salaries in a separate worksheet. To accommodate this need, we have a calculated field that transmits all new hire rates as

$1.00. This satisfies ADP's requirement to successfully import the new employee to the system. We will see later that a subsequent worksheet overrides this with the correct rate amount.

For parameters, we are using a date range, as well as a company code, since we have two ADP company codes in use. To decide which date parameters are right for an organization, carefully think through the payroll processes, the timing, cut-off dates, etc…and decide which date best accommodates the organization. In our case, we are calling a function in the worksheet, and have tailored our date parameters to suit. The function is simply called 'New Hire,' and provides a 'Y' or 'N' flag for whether or not this person is a new hire:

```
SELECT 'Y'
  FROM   per_periods_of_service
  WHERE  person_id = p_person_id
         -- ================================================
         -- This is to make sure the person is not terminated.
         -- ================================================
  AND    actual_termination_date IS NULL
  AND    (date_start BETWEEN p_start_date AND p_end_date
         OR (date_start < p_start_date
             AND last_update_date BETWEEN p_start_date AND
p_end_date));
```

In this sheet, we call the function as a calculation, then create a condition stating 'New Hire = 'Y'.' If we had used the xxhr_adp_ben_v view here, rather than xxhr_adp_no_ben_v, we would see many records for each person, due to multiple deduction and earning elements tied to their person id. This would not pass ADP's New Hire import requirements; therefore, the view excluding benefit data was used.

Transfers

The next sheets are the File Hire Sheet (fh) and File Termination Sheet (ft). These may only be pertinent to organizations with more than one company code. It happens, on occasion, that we have an employee transfer from our AJM company to our AJK company, or vice versa. When this happens, it is necessary for ADP to 'terminate' the file in one company and 'hire' in the other. The 'File Hire' sheet looks exactly like the 'New Hire' sheet, and is even built on the same view; however, rather than calling the 'New Hire = 'Y'' function, we are calling another function, this time as a condition, to articulate 'Payroll Change = 'Y'. The Payroll Change function can be found in Appendix D.

We are using assignment effective start and end dates for our parameter. This is because the 'person id' of the employee remains the same through the transfer, and it is the *assignment* record that is changed by this transfer. The assignment effective start date distinguishes which assignment record we are referring to for a given person. Unlike the New Hire sheet, we don't want to know if they became hired, we want to know if their assignment record reflects a payroll change within our specified date

range. Our HR Coordinator designed her process around using these dates. For example, if she ran a payroll for the date parameters of January 1-15, then on the next payroll she knows to run it for January 16-31. She simply keeps track of the cut off dates used in the previous interface run, and starts on the following date for the next run. We are unique in that we have both semi-monthly and bi-weekly payrolls. Clever methods for tracking dates may not be an issue at other organizations. Correctly date-tracking in HR becomes a very sensitive matter when working around date parameters in the interface.

The file termination sheet is also pulled from xxhr_adp_no_ben_v, and pulls one standard field, File No, and four calculated fields to meet the requirements in ADP's guide for information necessary to terminate a record. The payroll change condition is set to 'Y' in this sheet, as well.

The Main Employee Data

The fourth sheet is responsible for the bulk of information transmitted to ADP. Because we ran New Hires, File Transfers In, and File Transfers Out in the previous sheets, everyone should be present and accounted for by the time the Main (mn) sheet is processed. As we can see in the Edit Sheet tool in Discoverer, this sheet is based off of the all encompassing xxhr_adp_all_v view. Because benefit data is included here, we will see multiple records for each person. This is exactly what we want. We have the option of only sending information over if it is new or different, but that takes more complex coding, and makes it more difficult to track the changes if a file is lost or corrupt. There is no harm in replacing information with the exact same information upon import. For example, if I have health, dental and 401(k) benefits, I will have three lines in the 'mn' sheet for my file number. In each line, my name, address, phone, etc…will remain the same, and the three lines will constantly overwrite their respective fields with repetitive information. This is not a problem. For each line, where the deduction factor is different (for health, dental, and 401(k)), ADP will intelligently recognize that information and include it in my deduction listing. It is possible to reverse a deduction to remove it, but that was not part of our phase one implementation. We handle this issue manually because it is a rare occurrence.

There are several pieces of basic information taken from the view, and there are several conditions and calculations, as well. Due to the way the data is presented in the database, we have made some of the address fields substrings, or padded them, to prevent rejection by ADP due to field size. We included calculated fields for separating Deduction Factor from Deduction Amount. For organizations planning to transmit 401(k) data, which I highly recommend for its census reporting value, ADP can store percentages as well as flat dollar rates. These have to be in separate columns in ADP; therefore, calculated fields to decode are necessary. The number of fields transmitted in the main sheet is determined by the organization's scope for the project. We only included the basics in phase one, but have already started adding additional items, such as ADP allowed and taken data. The *ADP Integration Manual* will clearly state which items can be on this sheet and the required field name formats. This sheet pulls *all*

active employees, *every* time; therefore, there is no date parameter linked to this query. The only parameter used in this sheet is the Company Code selection.

The Rate Sheet

The Rate Sheet (rt) is based off of the xxhr_adp_no_ben_v view. This sheet includes the rate of every employee where the 'Salary Last Update Date' falls within our date parameters, which are based on the same. Again, our HR associate takes measures to assure there is no gap between the run dates of this information from payroll to payroll. Oracle has been kind enough to provide a very wide variety of date options which remove the hassle of finding special ways to indicate we want the most recent record. It's worth the time to carefully examine all the date options and how they might accommodate or enhance existing processes.

Earnings Codes

In our organization, the only earnings code (ec) we are transmitting, other than rate which is in the rate sheet, is a benefit called 'Car Allowance.' Again, in Oracle, deductions and earnings are stored together, and the union in our code is what separates them for reporting purposes. This query is based on xxhr_adp_ben_v, and calls earnings where 'earnings code = C.' This is a very basic sheet. Like the main sheet, it transmits every pertinent employee, every time. There's no harm in overriding data with the exact same data. Also like the main sheet, the only parameter used here is Company Code, which is in every sheet.

Terminations

The last sheet in the interface transmits our terminations. This sheet is unique, as it is built off of the Oracle table per_all_periods_of_service, the seeded Discoverer 'Payroll Core' business area for payroll name, as well as per_all_people_f. The per_all_periods_of_service table is where Oracle stores key data such as 'Actual Termination Date.' We are using the data parameters of 'Actual Termination Date,' and the calculated field 'Hire Status' which is set to always transmit a 'T' for Terminated.

A Note About Conditions, Parameters, & Calculations

Each time we create a condition, parameter, or calculation, it needs to be one pertaining to the same view that is queried in the sheet. For example, if I create a parameter on a report using the xxhr_adp_no_ben_v view, which states that the company code should be AJK, I can't go to another sheet that is built off of xxhr_adp_ben_v view and use that condition. It will give me an error stating that this condition doesn't pertain to anything in the current sheet. It will be necessary to create another calculation for the current view which states the same thing. As an example, I called the company code parameters 'ben_v_company_code' and 'no_ben_company_code.' Like all fields, the calculated fields can easily be renamed in the actual sheet to meet import requirements.

Interfacing the Data

Now that we have all the sheets, it's important to note that the queries can be run in any order. Because the worksheets are set up in the correct order of import from left to right, they will always export in that sequence; regardless of the order they were processed. Once all the queries have been run, it's time to export. This is where the unique worksheet names come in to play. Per the *ADP Integration Manual*, the naming convention for these sheets needs to be 'xxAJKemp.csv.' This is two letters, followed by the company code, followed by 'emp' (for employee data, as opposed to pay-data), then the file type. When we go to 'File' and 'Export' in Discoverer, we see the export form. If we choose 'Export All Sheets,' choose file type '.csv', and a file name of 'AJK', they will automatically export concatenating all that to 'nhAJKemp.csv,' 'mnAJKemp.csv,' etc… This way, minimal editing is required to make sure the files are properly named for import.

Depending on whether or not an organization is hosted by ADP, the payroll processor will either save these files to 'C:' which is the hosted environment 'V:' or simply save them in the local folder of choice. For hosted customers, putting them on C:/V: works best. Then, go to the ADP file manager in the hosted environment and copy/paste these files from their current location on V: to the ADP DATA folder. Once in the ADP DATA folder, go to ADP and import them in the EXACT order they appear in the Discoverer interface.

When the files are received/imported to ADP, a message window will appear regarding the success or failure of the import. If a warning is encountered, simply double click on it to open the log file. Generally, it's something as benign as a department number or deduction code missing from the validation table in ADP, or a space in the column header name. Simply address the issue, then go back to the receive/import screen. There will be a 'restart' file available for importing the files that failed to import the first time. Once all the corrections have been made, simply load the restart file. If additional warnings are encountered, address them and reload as necessary until the entire file is transmitted successfully. If it is necessary to alter the .csv file, right click and open it with Notepad or Wordpad. Do not open the .csv in Excel, as it alters the formatting and makes uploading the data to ADP impossible.

Testing

To test the data transmitted by the interface, it is helpful to have a test database provided by ADP. We were able to run payroll reports from our live system and the test system concurrently, and compare the results. The Employee Changes Report in ADP's Employee Data Report section was our gold standard of comparison. We did not encounter a single instance where data in the interface was incorrect due the interface itself. The only data issues we encountered were due to incorrect entries in Oracle. The interface accurately pulled whatever was in Oracle, 100% of the time.

Friendly and Flexible

There are many ways we could have approached this task, but we needed something flexible that would allow growth while also providing a user friendly interface for our functional end-users. As time has passed, we have already been asked to make multiple additions and modifications due to new data or simply a change in preference. Because we used Discoverer to do this, change doesn't have to be a crisis. We simply alter the data in the Discoverer queries, if possible; if not, we change the code. Each time the code is changed, it is necessary to go to the Discoverer Administrative edition and refresh the business area. When we make a code change, we run it in a test environment before requesting the change in production. Because we are an Oracle OnDemand customer, code migration requests for production take anywhere from 24 to 48 hours. Whether a quick, in-house fix or an official production change, the time frames are generally acceptable to our end users.

From the restart of the interface, to the time we moved it to production for HR to begin parallel testing, was three weeks. If we can accomplish this with two people, zero dollars, and three weeks, we're confident anyone can.

Lessons Learned

There are two key pieces of advice that we offer:

> Make sure there is a *very* clear understanding of the benefits configuration before starting. This is what caused us to start over three weeks before the deadline. When we say configuration, we not only mean Oracle's standard configuration behavior with regard to plans, options, and elements, but also the organization's process configuration. Is it best to hardcode values? Utilitize DFF fields? Create look-up or validation tables? Iron these important details out prior to building the interface.
>
> The second piece of advice is to carefully plan the testing. There are many ways to test the integrity of the data without running the interface, such as parallel reporting. It is not necessary to wait until payroll processing time to run the interface and test the results. Any time new information has been entered in Oracle HR, the interface should present it. It's not necessary to restrict the project time-line by only testing semi-monthly or bi-weekly. Again, as previously stated, the interface *always* transmitted exactly what was in Oracle. Don't confuse testing the data with testing the interface. We wasted valuable time testing a flawless interface when we could have been testing data integrity, via other methods, all along.

Conclusion

As we grow, add international employees, expand the fields necessary for self-service, and set our sights on Oracle Payroll, the interface will meet our ever changing needs.

Eventually, we plan to conduct the entire payroll process on-site, and the interface will no longer be necessary. Until then, we are very pleased to say that Discoverer has proven to be a very flexible, user-friendly, and cost effective approach for handling our phasic HR implementation. It increased data security and data integrity, while infinitely expanding our reporting capabilities and saving us well over sixty-five thousand dollars in the first year alone.

About the Author

Susan Patterson is the Oracle System Administrator and Discoverer Administrator for Enturia, Inc. Prior to her current role, Susan spent over ten years in accounting, much of which involved processing payroll, primarily via ADP. At Enturia, Susan functioned as the Payroll Coordinator, specializing in PCPW and Report Smith.

Appendix A – SQL for xxhr_adp_all_v

```
CREATE OR REPLACE VIEW apps.xxhr_adp_all_v
AS
SELECT *
=====
FROM   xxhr_adp_ben_v
UNION
SELECT *
FROM   xxhr_adp_no_ben_v no_ben
WHERE  NOT EXISTS (SELECT 1
                   FROM   xxhr_adp_ben_v ben
                   WHERE  no_ben.person_id = ben.person_id
                   AND    no_ben.assignment_id = ben.assignment_id)
;

EXIT;
```

Appendix B – SQL for xxhr_adp_no_ben_v

```
CREATE OR REPLACE VIEW apps.xxhr_adp_no_ben_v
AS
SELECT
--
================================================================================
=====
-- Filename:  xxhr_adp_no_ben_v.sql
-- Author:    Mark J. Matson
-- Date:      30-aug-07
--
-- Desciption:  View for ADP Interface that contains everything except
Benefit
--              data.
--
-- Modification Log:
-- Date       Programmer        Description
-- ---------  ----------------  ---------------------------------------
------
-- 26-sep-07  Mark J. Matson    Updated for standards and new package
names
-- 14-jan-08  Mark J. Matson    Added first character of middle_names
to
--                              first_name field.  Added suffix to
last_name
--                              field.  Removed assignment attribute1.
Changed
--                              employee_number field to be aliased as
file_#.
-- 29-jan-08  Mark J. Matson    Removed comma between last_name and
suffix.
-- 19-feb-08  Mark J. Matson    Changed to pull entire middle_name, not
just
--                              first character.
--
================================================================================
=====
/*This select is for deductions*/
       ahaav.work_state worked_state_tax_code
     , ahaev.person_id person_id
     , ahaav.assignment_id assignment_id
     , ahaev.assignment_status status
     , ahaav.employee_number file_#
     , pcak.segment1 || ('10') home_department
     , ahaev.hire_date hire_date
     , pp.effective_start_date per_eff_start
     , pp.effective_end_date per_eff_end
     , perallassignf.effective_start_date assign_eff_start
     , perallassignf.effective_end_date assign_eff_end
     , ahaev.greatest_last_update_date emp_great_date
     , ahaav.greatest_last_update_date assign_great_date
     , pp.payroll_name co_code
     , ahaev.actual_termination_date termination_date
     , ahaev.address_line1 address_line_1
     , ahaev.address_line2 address_line_2
```

```
     , ahaev.birth_date birth_date
     , ahaev.city city
     , DECODE(ahaev.middle_names, NULL, ahaev.first_name,
ahaev.first_name ||
              ' ' || ahaev.middle_names) employee_first_name
     , ahaav.rate_effective_date primary_rate_effective_date
     , ahaav.sal_last_update_date sal_last_update_date
     , ahaev.sex gender
     , ahaev.social_security_number social_security_number
     , ahaev.state state_postal_code
     , ahaworkcompv.workers_comp_code workers_comp_code
     , DECODE(ahaev.suffix, NULL, ahaev.last_name, ahaev.last_name || '
' ||
             ahaev.suffix) employee_last_name
     , ahaev.zip_code zip_code
     , SUBSTR (hea.salary_basis, 1, 1) rate_type
     , SUBSTR (ahaev.phone_number, 1, 3) home_area_code
     , SUBSTR (ahaev.phone_number, 4, 8) home_phone_number
     , DECODE (ahaav.LOCATION, 'Remote', ffv_vl1.flex_value,
                ffv_vl2.flex_value) sui_sdi_tax_jurisdiction_code
     , ahaav.rate rate_1_amount
     , xxhr_adp_formulas.job_change (perallassignf.person_id,
                                     perallassignf.effective_start_date
                                    ) job_change
     , xxhr_adp_formulas.grade_change (perallassignf.person_id,
                                     perallassignf.effective_start_date
                                    ) grade_change
     , xxhr_adp_formulas.organization_change (perallassignf.person_id,
                                       perallassignf.effective_start_date
                                      ) org_change
     , xxhr_adp_formulas.adp_file_change (perallassignf.person_id,
                                      perallassignf.effective_start_date
                                      ) adp_file_change
     , xxhr_adp_formulas.payroll_change (perallassignf.person_id,
                                     perallassignf.effective_start_date
                                     ) payroll_change
     , NULL deduction_amount
     , NULL deduction_code
     , NULL earnings_amount
     , NULL earnings_code
     , NULL annum_amt
     , NULL allowed_taken_code
     , hou.organization_id
     , hea.business_group_id
     , hea.grade_id
     , hea.job_id
     , hea.location_id
     , hea.salary_basis_id
     , hea.supervisor_id
     , pp.payroll_id
     , NULL element_entry_id
     , NULL updating_action_id
     , NULL element_link_id
     , NULL original_entry_id
     , NULL target_entry_id
     , NULL source_id
     , NULL element_entry_value_id
```

```
      , NULL input_value_id
      , pptuf.person_type_usage_id
      , pptuf.person_type_id
      , '
' adp_attribute1
      , '
' adp_attribute2
      , '
' adp_attribute3
      , '
' adp_attribute4
      , '
' adp_attribute5
      , '
' adp_attribute6
      , '
' adp_attribute7
      , '
' adp_attribute8
      , '
' adp_attribute9
      , '
' adp_attribute10
      , '
' adp_attribute11
      , '
' adp_attribute12
      , '
' adp_attribute13
      , '
' adp_attribute14
      , '
' adp_attribute15
      , '
' adp_attribute16
      , '
' adp_attribute17
      , '
' adp_attribute18
      , '
' adp_attribute19
      , '
' adp_attribute20
FROM   hrfg_employee_assignments hea,
       paybg_cost_allocation_keyflex pcak,
       paybg_payroll pp,
       hrbg_organization_unit hou,
       apps.hr_adp_assignment_v ahaav,
       (SELECT DISTINCT person_id
             , assignment_status
             , hire_date
             , greatest_last_update_date
             , actual_termination_date
             , address_line1
             , address_line2
             , birth_date
             , city
```

```
              , first_name
              , sex
              , social_security_number
              , state
              , last_name
              , middle_names
              , suffix
              , zip_code
              , phone_number
       FROM   apps.hr_adp_employee_v) ahaev,
      apps.hr_adp_workers_comp_v ahaworkcompv,
      hr.per_all_assignments_f perallassignf,
      fnd_flex_values_vl ffv_vl1,
      fnd_flex_values_vl ffv_vl2,
      fnd_flex_value_sets ffvs1,
      hr.per_person_type_usages_f pptuf
WHERE perallassignf.person_id = ahaev.person_id
AND   ahaev.person_id = ahaav.person_id
AND   perallassignf.payroll_id = pp.payroll_id
AND   ahaev.person_id = ahaworkcompv.person_id(+)
AND   perallassignf.assignment_id = hea.assignment_id
AND   perallassignf.organization_id = hou.organization_id
AND   hou.cost_allocation_keyflex_id = pcak.cost_allocation_keyflex_id
AND   ffv_vl1.description = ahaev.state
AND   ffv_vl2.description = ahaav.work_state
AND   ffvs1.flex_value_set_id = ffv_vl1.flex_value_set_id
AND   ffvs1.flex_value_set_id = ffv_vl2.flex_value_set_id
AND   ffvs1.flex_value_set_name = 'XXHR_SUI_SDI_JURISDICTION_CODE'
AND   pptuf.person_type_id = 6
AND   pptuf.person_id = ahaev.person_id
AND   perallassignf.effective_start_date = ahaav.effective_start_date
AND   perallassignf.effective_end_date = ahaav.effective_end_date
;

EXIT;
```

Appendix C – SQL for xxhr_adp_ben_v

```
CREATE OR REPLACE VIEW apps.xxhr_adp_ben_v
AS
SELECT
--
================================================================================
=====
-- Filename:  xxhr_adp_ben_v.sql
-- Author:    Mark J. Matson
-- Date:      30-aug-07
--
-- Description:  View for ADP Interface that contains all benefit
related data.
--
-- Modification Log:
-- Date       Programmer        Description
-- ---------  ----------------  ------------------------------------
------
-- 26-sep-07  Mark J. Matson    Updated for standards and new package
names
-- 14-jan-08  Mark J. Matson    Added first character of middle_names
to
--                              first_name field.  Added suffix to
last_name
--                              field.  Removed assignment attribute1.
Changed
--                              employee_number field to be aliased as
file_#.
-- 29-jan-08  Mark J. Matson    Removed comma between last_name and
suffix.
-- 19-feb-08  Mark J. Matson    Changed to pull entire middle_name, not
just
--                              first character.
--
================================================================================
=====
/*This select is for deductions*/
       ahaav.work_state worked_state_tax_code
     , ahaev.person_id person_id
     , ahaav.assignment_id assignment_id
     , ahaev.assignment_status status
     , ahaav.employee_number file_#
     , pcak.segment1 || '10' home_department
     , ahaev.hire_date hire_date
     , pp.effective_start_date per_eff_start
     , pp.effective_end_date per_eff_end
     , perallassignf.effective_start_date assign_eff_start
     , perallassignf.effective_end_date assign_eff_end
     , ahaev.greatest_last_update_date emp_great_date
     , ahaav.greatest_last_update_date assign_great_date
     , pp.payroll_name co_code
     , ahaev.actual_termination_date termination_date
     , ahaev.address_line1 address_line_1
     , ahaev.address_line2 address_line_2
     , ahaev.birth_date birth_date
```

```
     , ahaev.city city
     , DECODE(ahaev.middle_names, NULL, ahaev.first_name,
ahaev.first_name ||
              ' ' || ahaev.middle_names) employee_first_name
     , ahaav.rate_effective_date primary_rate_effective_date
     , ahaav.sal_last_update_date sal_last_update_date
     , ahaev.sex gender
     , ahaev.social_security_number social_security_number
     , ahaev.state state_postal_code
     , ahaworkcompv.workers_comp_code workers_comp_code
     , DECODE(ahaev.suffix, NULL, ahaev.last_name, ahaev.last_name || '
' ||
             ahaev.suffix) employee_last_name
     , ahaev.zip_code zip_code
     , SUBSTR (hea.salary_basis, 1, 1) rate_type
     , SUBSTR (ahaev.phone_number, 1, 3) home_area_code
     , SUBSTR (ahaev.phone_number, 4, 8) home_phone_number
     , DECODE (ahaav.LOCATION, 'Remote', ffv_vl1.flex_value
     , ffv_vl2.flex_value) sui_sdi_tax_jurisdiction_code
     , ahaav.rate rate_1_amount
     , xxhr_adp_formulas.job_change (perallassignf.person_id,
                                     perallassignf.effective_start_date
                                    ) job_change
     , xxhr_adp_formulas.grade_change (perallassignf.person_id,
                                     perallassignf.effective_start_date
                                    ) grade_change
     , xxhr_adp_formulas.organization_change (perallassignf.person_id,
                                       perallassignf.effective_start_date
                                      ) org_change
     , xxhr_adp_formulas.adp_file_change (perallassignf.person_id,
                                      perallassignf.effective_start_date
                                      ) adp_file_change
     , xxhr_adp_formulas.payroll_change (perallassignf.person_id,
                                     perallassignf.effective_start_date
                                     ) payroll_change
     , SUM (bprv.cmcd_rt_val) deduction_amount
     , DECODE (bof.cop_attribute1, NULL, bpf.pln_attribute1,
                bof.cop_attribute1) deduction_code
     , NULL earnings_amount
     , NULL earnings_code
     , NULL annum_amt
     , NULL allowed_taken_code
     , hou.organization_id
     , hea.business_group_id
     , hea.grade_id
     , hea.job_id
     , hea.location_id
     , hea.salary_basis_id
     , hea.supervisor_id
     , pp.payroll_id
     , NULL element_entry_id
     , NULL updating_action_id
     , NULL element_link_id
     , NULL original_entry_id
     , NULL target_entry_id
     , NULL source_id
     , NULL element_entry_value_id
```

```
      , NULL input_value_id
      , pptuf.person_type_usage_id
      , pptuf.person_type_id
      , '
' adp_attribute1
      , '
' adp_attribute2
      , '
' adp_attribute3
      , '
' adp_attribute4
      , '
' adp_attribute5
      , '
' adp_attribute6
      , '
' adp_attribute7
      , '
' adp_attribute8
      , '
' adp_attribute9
      , '
' adp_attribute10
      , '
' adp_attribute11
      , '
' adp_attribute12
      , '
' adp_attribute13
      , '
' adp_attribute14
      , '
' adp_attribute15
      , '
' adp_attribute16
      , '
' adp_attribute17
      , '
' adp_attribute18
      , '
' adp_attribute19
      , '
' adp_attribute20
FROM   hrfg_employee_assignments hea,
       paybg_cost_allocation_keyflex pcak,
       paybg_payroll pp,
       hrbg_organization_unit hou,
       apps.hr_adp_assignment_v ahaav,
       (SELECT DISTINCT person_id
               , assignment_status
               , hire_date
               , greatest_last_update_date
               , actual_termination_date
               , address_line1
               , address_line2
               , birth_date
               , city
```

```
                , first_name
                , sex
                , social_security_number
                , state
                , last_name
                , middle_names
                , suffix
                , zip_code
                , phone_number
         FROM   apps.hr_adp_employee_v) ahaev,
        apps.hr_adp_workers_comp_v ahaworkcompv,
        hr.per_all_assignments_f perallassignf,
        ben.ben_prtt_enrt_rslt_f bperf,
        ben.ben_pl_f bpf,
        ben.ben_oipl_f bof,
        ben_prtt_rt_val bprv,
        fnd_flex_values_vl ffv_vl1,
        fnd_flex_values_vl ffv_vl2,
        fnd_flex_value_sets ffvs1,
        hr.per_person_type_usages_f pptuf
WHERE   perallassignf.person_id = ahaev.person_id
AND     ahaev.person_id = ahaav.person_id
AND     perallassignf.payroll_id = pp.payroll_id
AND     ahaev.person_id = ahaworkcompv.person_id(+)
AND     perallassignf.assignment_id = hea.assignment_id
AND     perallassignf.organization_id = hou.organization_id
AND     hou.cost_allocation_keyflex_id = pcak.cost_allocation_keyflex_id
AND     ffv_vl1.description = ahaev.state
AND     ffv_vl2.description = ahaav.work_state
AND     ffvs1.flex_value_set_id = ffv_vl1.flex_value_set_id
AND     ffvs1.flex_value_set_id = ffv_vl2.flex_value_set_id
AND     ffvs1.flex_value_set_name = 'XXHR_SUI_SDI_JURISDICTION_CODE'
AND     pptuf.person_type_id = 6
AND     pptuf.person_id = ahaev.person_id
AND     bperf.person_id = perallassignf.person_id
AND     bperf.pl_id = bpf.pl_id
AND     bperf.oipl_id = bof.oipl_id(+)
AND     perallassignf.effective_start_date = ahaav.effective_start_date
AND     perallassignf.effective_end_date = ahaav.effective_end_date
AND     bperf.prtt_enrt_rslt_stat_cd IS NULL
AND     bperf.enrt_cvg_thru_dt <= bperf.effective_end_date
AND     bperf.prtt_enrt_rslt_id = bprv.prtt_enrt_rslt_id
AND     SYSDATE BETWEEN bprv.rt_strt_dt AND bprv.rt_end_dt
GROUP BY ahaav.work_state,
        ahaev.person_id,
        ahaav.assignment_id,
        ahaev.assignment_status,
        ahaav.employee_number,
        pcak.segment1 || '10',
        ahaev.hire_date,
        pp.effective_start_date,
        pp.effective_end_date,
        perallassignf.effective_start_date,
        perallassignf.effective_end_date,
        ahaev.greatest_last_update_date,
        ahaav.greatest_last_update_date,
        pp.payroll_name,
```

```
       ahaev.actual_termination_date,
       ahaev.address_line1,
       ahaev.address_line2,
       ahaev.birth_date,
       ahaev.city,
       DECODE(ahaev.middle_names, NULL, ahaev.first_name,
ahaev.first_name ||
              ' ' || ahaev.middle_names),
       ahaav.rate_effective_date,
       ahaav.sal_last_update_date,
       ahaev.sex,
       ahaev.social_security_number,
       ahaev.state,
       ahaworkcompv.workers_comp_code,
       DECODE(ahaev.suffix, NULL, ahaev.last_name, ahaev.last_name || '
' ||
              ahaev.suffix),
       ahaev.zip_code,
       SUBSTR (hea.salary_basis, 1, 1),
       SUBSTR (ahaev.phone_number, 1, 3),
       SUBSTR (ahaev.phone_number, 4, 8),
       DECODE (ahaav.LOCATION, 'Remote', ffv_vl1.flex_value,
                 ffv_vl2.flex_value),
       ahaav.rate,
       xxhr_adp_formulas.job_change (perallassignf.person_id,
                                     perallassignf.effective_start_date),
       xxhr_adp_formulas.grade_change (perallassignf.person_id,
                                     perallassignf.effective_start_date),
       xxhr_adp_formulas.organization_change (perallassignf.person_id,

perallassignf.effective_start_date),
       xxhr_adp_formulas.adp_file_change (perallassignf.person_id,
                                  perallassignf.effective_start_date) ,
       xxhr_adp_formulas.payroll_change (perallassignf.person_id,
                                 perallassignf.effective_start_date) ,
       DECODE (bof.cop_attribute1, NULL, bpf.pln_attribute1,
                 bof.cop_attribute1),
       NULL,
       NULL,
       NULL,
       NULL
     , hou.organization_id
     , hea.business_group_id
     , hea.grade_id
     , hea.job_id
     , hea.location_id
     , hea.salary_basis_id
     , hea.supervisor_id
     , pp.payroll_id
     , NULL
     , NULL
     , NULL
     , NULL
     , NULL
     , NULL
     , NULL
     , NULL
```

```
      , pptuf.person_type_usage_id
      , pptuf.person_type_id
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
HAVING SUM (bprv.cmcd_rt_val) <> 0
UNION
SELECT
/*This select is for earnings*/
       ahaav.work_state
      , ahaev.person_id
      , ahaav.assignment_id
      , ahaev.assignment_status
      , ahaav.employee_number
      , pcak.segment1 || '10'
      , ahaev.hire_date
      , pp.effective_start_date
      , pp.effective_end_date
      , perallassignf.effective_start_date
      , perallassignf.effective_end_date
```

```
     , ahaev.greatest_last_update_date
     , ahaav.greatest_last_update_date
     , pp.payroll_name
     , ahaev.actual_termination_date
     , ahaev.address_line1
     , ahaev.address_line2
     , ahaev.birth_date
     , ahaev.city
     , DECODE(ahaev.middle_names, NULL, ahaev.first_name,
ahaev.first_name ||
              ' ' || ahaev.middle_names)
     , ahaav.rate_effective_date
     , ahaav.sal_last_update_date
     , ahaev.sex
     , ahaev.social_security_number
     , ahaev.state
     , ahaworkcompv.workers_comp_code
     , DECODE(ahaev.suffix, NULL, ahaev.last_name, ahaev.last_name || '
' ||
              ahaev.suffix)
     , ahaev.zip_code
     , SUBSTR (hea.salary_basis, 1, 1)
     , SUBSTR (ahaev.phone_number, 1, 3)
     , SUBSTR (ahaev.phone_number, 4, 8)
     , DECODE (ahaav.LOCATION, 'Remote', ffv_vl1.flex_value,
ffv_vl2.flex_value)
     , ahaav.rate
     , xxhr_adp_formulas.job_change (perallassignf.person_id,
                                     perallassignf.effective_start_date)
     , xxhr_adp_formulas.grade_change (perallassignf.person_id,
                                     perallassignf.effective_start_date)
     , xxhr_adp_formulas.organization_change (perallassignf.person_id,

perallassignf.effective_start_date)
     , xxhr_adp_formulas.adp_file_change (perallassignf.person_id,
                                    perallassignf.effective_start_date)
     , xxhr_adp_formulas.payroll_change (perallassignf.person_id,
                                   perallassignf.effective_start_date)
     , NULL
     , NULL
     , ppev.screen_entry_value
     , petf.attribute1
     , NULL
     , NULL
     , hou.organization_id
     , hea.business_group_id
     , hea.grade_id
     , hea.job_id
     , hea.location_id
     , hea.salary_basis_id
     , hea.supervisor_id
     , pp.payroll_id
     , peef.element_entry_id
     , peef.updating_action_id
     , peef.element_link_id
     , peef.original_entry_id
     , peef.target_entry_id
```

```
       , peef.source_id
       , ppev.element_entry_value_id
       , ppev.input_value_id
       , pptuf.person_type_usage_id
       , pptuf.person_type_id
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
       , '
'
FROM   hrfg_employee_assignments hea,
       paybg_cost_allocation_keyflex pcak,
       paybg_payroll pp,
       hrbg_organization_unit hou,
       apps.hr_adp_assignment_v ahaav,
       (SELECT DISTINCT person_id
               , assignment_status
               , hire_date
               , greatest_last_update_date
               , actual_termination_date
               , address_line1
               , address_line2
```

```
                , birth_date
                , city
                , first_name
                , sex
                , social_security_number
                , state
                , last_name
                , middle_names
                , suffix
                , zip_code
                , phone_number
         FROM   apps.hr_adp_employee_v) ahaev,
        apps.hr_adp_workers_comp_v ahaworkcompv,
        hr.per_all_assignments_f perallassignf,
        fnd_flex_values_vl ffv_vl1,
        fnd_flex_values_vl ffv_vl2,
        fnd_flex_value_sets ffvs1,
        hr.pay_element_entries_f peef,
        hr.pay_element_entry_values_f ppev,
        hr.pay_element_types_f petf,
        hr.per_person_type_usages_f pptuf
WHERE   perallassignf.person_id = ahaev.person_id
AND     ahaev.person_id = ahaav.person_id
AND     perallassignf.payroll_id = pp.payroll_id
AND     ahaev.person_id = ahaworkcompv.person_id(+)
AND     perallassignf.assignment_id = hea.assignment_id
AND     perallassignf.organization_id = hou.organization_id
AND     hou.cost_allocation_keyflex_id = pcak.cost_allocation_keyflex_id
AND     ffv_vl1.description = ahaev.state
AND     ffv_vl2.description = ahaav.work_state
AND     ffvs1.flex_value_set_id = ffv_vl1.flex_value_set_id
AND     ffvs1.flex_value_set_id = ffv_vl2.flex_value_set_id
AND     ffvs1.flex_value_set_name = 'XXHR_SUI_SDI_JURISDICTION_CODE'
AND     peef.element_entry_id = ppev.element_entry_id
AND     peef.element_type_id = petf.element_type_id
AND     peef.assignment_id = perallassignf.assignment_id
AND     pptuf.person_type_id = 6
AND     pptuf.person_id = ahaev.person_id
AND     perallassignf.effective_start_date = ahaav.effective_start_date
AND     perallassignf.effective_end_date = ahaav.effective_end_date
AND     petf.attribute2 = 'E'
AND     petf.attribute3 = 'Y'
UNION
SELECT
/*This select is for allowed*/
        ahaav.work_state
      , ahaev.person_id
      , ahaav.assignment_id
      , ahaev.assignment_status
      , ahaav.employee_number
      , pcak.segment1 || '10'
      , ahaev.hire_date
      , pp.effective_start_date
      , pp.effective_end_date
      , perallassignf.effective_start_date
      , perallassignf.effective_end_date
      , ahaev.greatest_last_update_date
```

```
     , ahaav.greatest_last_update_date
     , pp.payroll_name
     , ahaev.actual_termination_date
     , ahaev.address_line1
     , ahaev.address_line2
     , ahaev.birth_date
     , ahaev.city
     , DECODE(ahaev.middle_names, NULL, ahaev.first_name,
ahaev.first_name ||
              ' ' || ahaev.middle_names)
     , ahaav.rate_effective_date
     , ahaav.sal_last_update_date
     , ahaev.sex
     , ahaev.social_security_number
     , ahaev.state
     , ahaworkcompv.workers_comp_code
     , DECODE(ahaev.suffix, NULL, ahaev.last_name, ahaev.last_name || '
' ||
              ahaev.suffix)
     , ahaev.zip_code
     , SUBSTR (hea.salary_basis, 1, 1)
     , SUBSTR (ahaev.phone_number, 1, 3)
     , SUBSTR (ahaev.phone_number, 4, 8)
     , DECODE (ahaav.LOCATION, 'Remote', ffv_vl1.flex_value,
ffv_vl2.flex_value)
     , ahaav.rate
     , xxhr_adp_formulas.job_change (perallassignf.person_id,
                                     perallassignf.effective_start_date)
     , xxhr_adp_formulas.grade_change (perallassignf.person_id,
                                     perallassignf.effective_start_date)
     , xxhr_adp_formulas.organization_change (perallassignf.person_id,

perallassignf.effective_start_date)
     , xxhr_adp_formulas.adp_file_change (perallassignf.person_id,
                                    perallassignf.effective_start_date)
     , xxhr_adp_formulas.payroll_change (perallassignf.person_id,
                                   perallassignf.effective_start_date)
     , NULL
     , NULL
     , NULL
     , NULL
     , ppev.screen_entry_value
     , petf.attribute1
     , hou.organization_id
     , hea.business_group_id
     , hea.grade_id
     , hea.job_id
     , hea.location_id
     , hea.salary_basis_id
     , hea.supervisor_id
     , pp.payroll_id
     , peef.element_entry_id
     , peef.updating_action_id
     , peef.element_link_id
     , peef.original_entry_id
     , peef.target_entry_id
     , peef.source_id
```

```
      , ppev.element_entry_value_id
      , ppev.input_value_id
      , pptuf.person_type_usage_id
      , pptuf.person_type_id
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
      , '
'
FROM   hrfg_employee_assignments hea,
       paybg_cost_allocation_keyflex pcak,
       paybg_payroll pp,
       hrbg_organization_unit hou,
       apps.hr_adp_assignment_v ahaav,
       (SELECT DISTINCT person_id
               , assignment_status
               , hire_date
               , greatest_last_update_date
               , actual_termination_date
               , address_line1
               , address_line2
               , birth_date
```

```
                , city
                , first_name
                , sex
                , social_security_number
                , state
                , last_name
                , middle_names
                , suffix
                , zip_code
                , phone_number
         FROM     apps.hr_adp_employee_v) ahaev,
        apps.hr_adp_workers_comp_v ahaworkcompv,
        hr.per_all_assignments_f perallassignf,
        fnd_flex_values_vl ffv_vl1,
        fnd_flex_values_vl ffv_vl2,
        fnd_flex_value_sets ffvs1,
        hr.pay_element_entries_f peef,
        hr.pay_element_entry_values_f ppev,
        hr.pay_element_types_f petf,
        hr.per_person_type_usages_f pptuf
WHERE   perallassignf.person_id = ahaev.person_id
AND     ahaev.person_id = ahaav.person_id
AND     perallassignf.payroll_id = pp.payroll_id
AND     ahaev.person_id = ahaworkcompv.person_id(+)
AND     perallassignf.assignment_id = hea.assignment_id
AND     perallassignf.organization_id = hou.organization_id
AND     hou.cost_allocation_keyflex_id = pcak.cost_allocation_keyflex_id
AND     ffv_vl1.description = ahaev.state
AND     ffv_vl2.description = ahaav.work_state
AND     ffvs1.flex_value_set_id = ffv_vl1.flex_value_set_id
AND     ffvs1.flex_value_set_id = ffv_vl2.flex_value_set_id
AND     ffvs1.flex_value_set_name = 'XXHR_SUI_SDI_JURISDICTION_CODE'
AND     peef.element_entry_id = ppev.element_entry_id
AND     peef.element_type_id = petf.element_type_id
AND     peef.assignment_id = perallassignf.assignment_id
AND     pptuf.person_type_id = 6
AND     pptuf.person_id = ahaev.person_id
AND     perallassignf.effective_start_date = ahaav.effective_start_date
AND     perallassignf.effective_end_date = ahaav.effective_end_date
AND     petf.attribute2 = 'A'
AND     petf.attribute3 = 'Y'
;

EXIT;
```

Appendix D - Package xxhr_adp_formulas

```
CREATE OR REPLACE PACKAGE BODY apps.xxhr_adp_formulas AS

FUNCTION job_change ( p_person_id      NUMBER
                    , p_effective_date DATE )
RETURN VARCHAR
IS
  x_job_id      NUMBER;
  x_prev_job_id NUMBER;
  x_start_date  DATE;
  --
==========================================================================
==
  -- Determine job based of person based on p_effective_date parameter
  --
==========================================================================
==
  CURSOR current_job IS
  SELECT job_id, effective_start_date
  FROM   per_all_assignments_f
  WHERE  p_effective_date BETWEEN effective_start_date AND
effective_end_date
  AND    person_id = p_person_id;
  --
==========================================================================
==
  -- Determine previous job of person.
  --
==========================================================================
==
  CURSOR previous_job IS
  SELECT job_id
  FROM   per_all_assignments_f
  WHERE  effective_end_date = x_start_date - 1
  AND    person_id = p_person_id;
BEGIN
  -- ============================================================
  -- Get current job.
  -- ============================================================
  OPEN current_job;
  FETCH current_job INTO x_job_id, x_start_date;
  IF (current_job%NOTFOUND) THEN
    -- ============================================================
    -- If current job not found, return null.
    -- ============================================================
    CLOSE current_job;
    RETURN NULL;
  END IF;
  CLOSE current_job;
  -- ============================================================
  -- Get previous job.
  -- ============================================================
  OPEN previous_job;
  FETCH previous_job INTO x_prev_job_id;
  IF ( previous_job%NOTFOUND
```

```
       OR NVL(x_job_id, -10) != NVL(x_prev_job_id, -10) ) THEN
    --
================================================================
    -- If previous job not found or previous job does not equal current
job,
    -- return Y.
    --
================================================================
    CLOSE previous_job;
    RETURN 'Y';
  END IF;
  CLOSE previous_job;
  RETURN NULL;
END job_change;

FUNCTION grade_change ( p_person_id      NUMBER
                      , p_effective_date DATE )
RETURN VARCHAR
IS
  x_grade_id      NUMBER;
  x_prev_grade_id NUMBER;
  x_start_date    DATE;
  --
================================================================
==
  -- Determine grade based of person based on p_effective_date
parameter
  --
================================================================
==
  CURSOR current_grade IS
  SELECT grade_id, effective_start_date
  FROM   per_all_assignments_f
  WHERE  p_effective_date BETWEEN effective_start_date AND
effective_end_date
  AND    person_id = p_person_id;
  --
================================================================
==
  -- Determine previous grade of person.
  --
================================================================
==
  CURSOR previous_grade IS
  SELECT grade_id
  FROM   per_all_assignments_f
  WHERE  effective_end_date = x_start_date - 1
  AND    person_id = p_person_id;
BEGIN
  -- ============================================================
  -- Get current grade.
  -- ============================================================
  OPEN current_grade;
  FETCH current_grade INTO x_grade_id, x_start_date;
  IF (current_grade%NOTFOUND) THEN
    -- ============================================================
    -- If current grade not found, return null.
```

```
    -- ================================================================
    CLOSE current_grade;
    RETURN NULL;
  END IF;
  CLOSE current_grade;
  -- ================================================================
  -- Get previous grade.
  -- ================================================================
  OPEN previous_grade;
  FETCH previous_grade INTO x_prev_grade_id;
  IF ( previous_grade%NOTFOUND
       OR NVL(x_grade_id, -10) != NVL(x_prev_grade_id, -10) ) THEN
    --
================================================================================
    -- If previous grade not found or previous grade does not equal
current
    -- grade return Y.
    --
================================================================================
    CLOSE previous_grade;
    RETURN 'Y';
  END IF;
  CLOSE previous_grade;
  RETURN NULL;
END grade_change;

FUNCTION organization_change ( p_person_id      NUMBER
                             , p_effective_date DATE )
RETURN VARCHAR
IS
  x_organization_id      NUMBER;
  x_prev_organization_id NUMBER;
  x_start_date           DATE;
  --
================================================================================
===
  -- Determine organization based of person based on p_effective_date
parameter
  --
================================================================================
===
  CURSOR current_organization IS
  SELECT organization_id, effective_start_date
  FROM   per_all_assignments_f
  WHERE  p_effective_date BETWEEN effective_start_date AND
effective_end_date
  AND    person_id = p_person_id;
  --
================================================================================
==
  -- Determine previous organization of person.
  --
================================================================================
==
  CURSOR previous_organization IS
  SELECT organization_id
  FROM   per_all_assignments_f
```

```
  WHERE  effective_end_date = x_start_date - 1
  AND    person_id = p_person_id;
BEGIN
  -- ================================================================
  -- Get current organization.
  -- ================================================================
  OPEN current_organization;
  FETCH current_organization INTO x_organization_id, x_start_date;
  IF (current_organization%NOTFOUND) THEN
    -- ================================================================
    -- If current organization not found, return null.
    -- ================================================================
    CLOSE current_organization;
    RETURN NULL;
  END IF;
  CLOSE current_organization;
  -- ================================================================
  -- Get previous organization.
  -- ================================================================
  OPEN previous_organization;
  FETCH previous_organization INTO x_prev_organization_id;
  IF ( previous_organization%NOTFOUND
       OR NVL(x_organization_id, -10) != NVL(x_prev_organization_id, -
10) ) THEN
    --
================================================================================
    -- If previous organization not found or previous organization does
not
    -- equal current organization return Y.
    --
================================================================================
    CLOSE previous_organization;
    RETURN 'Y';
  END IF;
  CLOSE previous_organization;
  RETURN NULL;
END organization_change;

FUNCTION adp_file_change ( p_person_id      NUMBER
                         , p_effective_date DATE )
RETURN VARCHAR
IS
  x_adp_file       per_all_assignments_f.ass_attribute1%TYPE;
  x_prev_adp_file  per_all_assignments_f.ass_attribute1%TYPE;
  x_start_date     DATE;
  --
================================================================================
==
  -- Determine adp_file of person based on p_effective_date parameter
  --
================================================================================
==
  CURSOR current_adp_file IS
  SELECT ass_attribute1, effective_start_date
  FROM   per_all_assignments_f
  WHERE  p_effective_date BETWEEN effective_start_date AND
effective_end_date
```

```
  AND    person_id = p_person_id;
  --
================================================================================
==
  -- Determine previous adp_file of person.
  --
================================================================================
==
  CURSOR previous_adp_file IS
  SELECT ass_attribute1
  FROM   per_all_assignments_f
  WHERE  effective_end_date = x_start_date - 1
  AND    person_id = p_person_id;
BEGIN
  -- ============================================================
  -- Get current adp_file.
  -- ============================================================
  OPEN current_adp_file;
  FETCH current_adp_file INTO x_adp_file, x_start_date;
  IF (current_adp_file%NOTFOUND) THEN
    -- ============================================================
    -- If current adp_file not found, return null.
    -- ============================================================
    CLOSE current_adp_file;
    RETURN NULL;
  END IF;
  CLOSE current_adp_file;
  -- ============================================================
  -- Get previous adp_file.
  -- ============================================================
  OPEN previous_adp_file;
  FETCH previous_adp_file INTO x_prev_adp_file;
  IF ( previous_adp_file%NOTFOUND
       OR NVL(x_adp_file, -10) != NVL(x_prev_adp_file, -10) ) THEN
    --
================================================================================
    -- If previous adp_file not found or previous adp_file does not
equal
    -- current adp_file, return Y.
    --
================================================================================
    CLOSE previous_adp_file;
    RETURN 'Y';
  END IF;
  CLOSE previous_adp_file;
  RETURN NULL;
END adp_file_change;

FUNCTION payroll_change ( p_person_id      NUMBER
                        , p_effective_date DATE )
RETURN VARCHAR
IS
  x_payroll_id       per_all_assignments_f.payroll_id%TYPE;
  x_prev_payroll_id  per_all_assignments_f.payroll_id%TYPE;
  x_start_date       DATE;
```

```
  --
==============================================================================
==
  -- Determine payroll_id of person based on p_effective_date parameter
  --
==============================================================================
==
  CURSOR current_payroll_id IS
  SELECT payroll_id, effective_start_date
  FROM   per_all_assignments_f
  WHERE  p_effective_date BETWEEN effective_start_date AND
effective_end_date
  AND    person_id = p_person_id;
  --
==============================================================================
==
  -- Determine previous payroll_id of person.
  --
==============================================================================
==
  CURSOR previous_payroll_id IS
  SELECT payroll_id
  FROM   per_all_assignments_f
  WHERE  effective_end_date = x_start_date - 1
  AND    person_id = p_person_id;
BEGIN
  -- ============================================================
  -- Get current payroll_id.
  -- ============================================================
  OPEN current_payroll_id;
  FETCH current_payroll_id INTO x_payroll_id, x_start_date;
  IF (current_payroll_id%NOTFOUND) THEN
    -- ============================================================
    -- If current payroll_id not found, return null.
    -- ============================================================
    CLOSE current_payroll_id;
    RETURN NULL;
  END IF;
  CLOSE current_payroll_id;
  -- ============================================================
  -- Get previous payroll_id.
  -- ============================================================
  OPEN previous_payroll_id;
  FETCH previous_payroll_id INTO x_prev_payroll_id;
  IF ( previous_payroll_id%FOUND AND
       NVL(x_payroll_id, -10) != NVL(x_prev_payroll_id, -10) ) THEN
    --
==============================================================================
    -- If previous payroll_id not found or previous payroll_id does not
equal
    -- current payroll_id, return Y.
    --
==============================================================================
    CLOSE previous_payroll_id;
    RETURN 'Y';
  END IF;
  CLOSE previous_payroll_id;
```

```
  RETURN NULL;
END payroll_change;

FUNCTION check_dup_adp_file_number ( p_person_id       NUMBER
                                   , p_adp_file_number VARCHAR2 )
RETURN VARCHAR
IS
  x_dummy  NUMBER;
  --
================================================================================
==
  -- Determine if another person already has used this ADP file number.
  --
================================================================================
==
  CURSOR is_already_used IS
  SELECT 1
  FROM   per_all_assignments_f
  WHERE  person_id != p_person_id
  AND    ass_attribute1 = p_adp_file_number;
BEGIN
  -- ============================================================
  -- Get current payroll_id.
  -- ============================================================
  OPEN is_already_used;
  FETCH is_already_used INTO x_dummy;
  IF (is_already_used%FOUND) THEN
    -- ============================================================
    -- If current payroll_id not found, return null.
    -- ============================================================
    CLOSE is_already_used;
    RETURN 'Y';
  END IF;
  CLOSE is_already_used;
  RETURN 'N';
END check_dup_adp_file_number;

FUNCTION new_hire ( p_person_id   NUMBER
                  , p_start_date  DATE
                  , p_end_date    DATE )
RETURN VARCHAR
IS
  x_dummy  VARCHAR(1);
  --
================================================================================
==
  -- Determine job based of person based on p_effective_date parameter
  --
================================================================================
==
  CURSOR is_new_hire IS
  SELECT 'Y'
  FROM   per_periods_of_service
  WHERE  person_id = p_person_id
         -- ================================================
         -- This is to make sure the person is not terminated.
         -- ================================================
```

```
  AND    actual_termination_date IS NULL
  AND    (date_start BETWEEN p_start_date AND p_end_date
         OR (date_start < p_start_date
             AND last_update_date BETWEEN p_start_date AND
p_end_date));
BEGIN
  -- ================================================================
  -- Get current job.
  -- ================================================================
  OPEN is_new_hire;
  FETCH is_new_hire INTO x_dummy;
  IF (is_new_hire%FOUND) THEN
    -- ================================================================
    -- If current job not found, return null.
    -- ================================================================
    CLOSE is_new_hire;
    RETURN 'Y';
  END IF;
  CLOSE is_new_hire;
  RETURN 'N';
END new_hire;
END xxhr_adp_formulas;
/

SHOW ERRORS PACKAGE BODY apps.xxhr_adp_formulas;

SELECT TO_DATE('SQLERROR')
FROM user_errors
WHERE name = 'XXHR_ADP_FORMULAS'
AND type = 'PACKAGE BODY';

EXIT;
```

**To import a function to Discoverer, simply go to the Administrative Edition and 'Register' the function. Call the function as a calculation under the functions tab in Discoverer Desktop.*