Retek® Merchandising System 10.1.3

Addendum to Operations Guide

Retek

WWW.RETEK.COM | HELPING THE RETAIL INDUSTRY CREATE, MANAGE AND FULFILL CONSUMER DEMAND™

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403

888.61.RETEK (toll free US)
+1 612 587 5000

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46
Fax: +44 (0)20 7563 46 10

Customer Support

Customer Support hours:

Customer Support is available 7x24x365 via e-mail, phone, and Web access.

Depending on the Support option chosen by a particular client (Standard, Plus, or Premium), the times that certain services are delivered may be restricted. Severity 1 (Critical) issues are addressed on a 7x24 basis and receive continuous attention until resolved, for all clients on active maintenance.

Contact Method	**Contact Information**
Internet (ROCS)	www.retek.com/support Retek's secure client Web site to update and view issues
E-mail	support@retek.com
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5800 EMEA: 011 44 1223 703 444 Asia Pacific: 61 425 792 927
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step by step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction

This addendum to the Retek Merchandising System (RMS) 10.1.2 Operations Guide contains updates to the following batch designs:

- Daily record deletion (dlyprg)
- Contract replenishment (cntrordb)
- Time hierarchy download (ftmednld)
- Upload stock count results (stkupld)

Chapter 2 – ftmednld.pc

Functional Area

RDF Interfaces

Module Affected

ftmednld.pc (new) – Time Hierarchy Download

Design Overview

Currently, no extracts exist for the time dimension. So as not to have to maintain the calendar in multiple places (i.e. RMS, RDF and RPP), a time dimension extract is required that will download the RMS calendar, including the following fields: year, half, quarter, month, week, day and date (in a yyyymmdd format). The downloaded information would only use the 454 calendar format. The download would include the entire calendar in the RMS. The extract must account for a fiscal year that could be different than the standard year in the calendar table. A field on the System Options table indicates the month in which the fiscal year begins. For example, if the fiscal year begins on the 3rd month, the following chart highlights how this impacts the extract (the following is a subset of the data on the Calendar table):

First Day Month	Year	Month	# of Weeks in Month
25-OCT-99	1999	11	4
22-NOV-99	1999	12	5
27-DEC-99	**2000**	**1**	**4**
24-JAN-00	2000	2	4
21-FEB-00	**2000**	**3**	**5**
27-MAR-00	2000	4	4

If the fiscal year followed standard calendar, the first day of the year 2000 would be 27-Dec-99. However, for the fiscal year which starts on the 3rd month, the first day of the year 2000 would be 21-FEB-00. Therefore, 20-FEB-00 would be extracted as 1999 (year), 2 (half), 4 (quarter), 12 (month), 4 (week), 7 (day), 20000220 (date). If the year followed the regular calendar, 20-FEB-00 would be extracted as 2000 (year), 1 (half), 1 (quarter), 2 (month), 4 (week), 7 (day), 20000220 (date).

Input Specifications

'Table-To-File'

This program fetches the earliest and latest dates from the calendar also incorporating the start_of_half_month from the system_options table when fetching the earliest date.

Driving Cursor:

NA

Output Specifications

Output Files

The file outputted will be named rmse_clndmstr.dat.

Output File Format:

Record Name	Field Name	Field Type	Default Value	Description
	Year	Char(4)		The 454 year
	Half	Char(1)		The 454 half of the year, valid values are 1 or 2
	Quarter	Char(1)		The 454 quarter of the year, valid values 1-4
	Month	Char(2)		The 454 month of the year, valid values 1-12
	Week	Char(2)		The 454 week of the year, valid values 1-53
	Day	Char(1)		The 454 day of the current week, valid values 1-7
	Date	Char(8)		The date from which the 454 data was derived, in YYYYMMDD format

Function Level Description

main():

The standard Retek main() function. Calls init(), process(), and final().

init():

Initialize restart recovery by calling retek_init() and set up the output file.

format_buffer():

Formats the string that will be used to write to the output file.

get_dates():

Mutates output arguments with first and last calendar date fetched from the calendar.first_day field. First calendar date is determined by taking the first record from the calendar table ordered in chronological order with respect to first_date. The fetched record's month_454 field must match the absolute value of the system_options.start_of_half_month field. Last calendar date is determined by fetching the first record while the calendar table records are ordered in a descending order with respect to the first_day field.

increment_date()

Mutates input/output arguments to hold incremented date.

increment_454()

Mutates input/output arguments to hold incremented 454 date. Also fetches the calendar.no_of_weeks field from the calendar table's row whose first day field corresponds to the present date.

To determine the 454 day, month, weeks, yearly weeks, quarter, half and year, simply increment their current values by one if the corresponding date counters justify the incrementation. If any one of them turns over reset them to 1. Note that weeks turn over when the no_of_weeks value fetched from the calendar table is no longer greater than or equal to the current week value.

init_454()

Initializes argument 454 date instance's date fields.

write_fdetl()

Writes data from argument to output file.

process():

This function first makes a call to format_buffer(). Then allocates a date struct to hold the 454 date. It then calls the init_454() function with a pointer to the 454 date struct as argument. It then fetches the earliest and latest values of calendar.first_day into local variables by calling get_dates(). For each day in the date range (including the earliest and latest dates), the current calendar and 454 dates are calculated by calling increment_date() and increment_454().

A record containing all of the 454 values for the date, in addition to the date itself (in YYYYMMDD format) will then be written to the file by calling write_fdetl().

final():

Take care of file clean up and complete the restart recovery process by calling retek_close().

Scheduling Considerations

This program can be run ad hoc.

Restart/Recovery

Due to the relatively small amount of processing this program performs, restart recovery will not be used. The calls to retek_init() and retek_close() are used in the program only for logging purposes (to prevent double-runs).

Chapter 3 – POS Upload [posupld]

Design Overview

The purpose of this batch module is to process sales and return details from an external point of sale system. The sales/return transactions will be validated against Retek item/store relations to ensure the sale is valid, but this validation process can be eliminated if the sales being passed in have already been screened by sales auditing. The following common functions will be performed on each sales/return record read from the input file:

- read sales/return transaction record
- lock associated record in RMS
- validate item sale
- check if VAT maintenance is required, if so determine the VAT amount for the sale
- write all financial transactions for the sale and any relevant markdowns to the stock ledger.
- post item/location/week sales to the relevant sales history tables
- if a late posting occurs in a previous week (i.e. not in the current week), if the item for which the late posting occurred is forecastable, the last_hist_export_date on the item_loc_soh table has to be updated to the end of week date previous to the week of the late posting. This will result in the sales download interface programs extracting the week(s) for which the late transactions were posted to maintain accurate sales information in the external forecasting system.

Scheduling Constraints

Processing Cycle: PHASE 2 (daily)

Scheduling Diagram: This program will likely be run at the beginning of the batch run during the POS polling cycle. It can be scheduled to run multiple times throughout the day, as POS data becomes available.

Pre-Processing: N/A

Post-Processing: N/A

Threading Scheme: N/A

Restart Recovery

The logical unit of work for the sales/returns upload module will be a valid item sales transaction at a given store location. The location type will be inferred as a store type and the item can be passed as an item or reference item type. The logical unit of work will be defined as a number of these transaction records. The commit_max_ctr field on the restart_control table will determine the number of transactions that equal a logical unit of work.

The file records will be read in groups of numbers equal to the commit_max_ctr. After all records in a given read are processed (or rejected either as a reject record or a lock error record), the restart commit logic and restart file writing logic will be called, and then the next group of file records will be read and processed. The commit logic will save the current file pointer position in the input file and any application image information (e.g. record and reject counters) and commit all database transactions. The file writing logic will append the temporary holding files to the final output files.

The commit_max_ctr field should be set to prevent excessive rollback space usage, and to reduce the overhead of file I/O. The recommended commit counter setting is 10000 records (subject to change based on experimentation).

Error handling will recognize three levels of record processing: process success, non-fatal errors, and fatal errors. Item level validation will occur on all fields before table processes are initiated. If all field-level validations return successfully, inserts and updates will be allowed. If a non-fatal error is produced, the remaining fields will be validated, but the record will be rejected and written to the reject file or written to the lock file depending on the reject reason. If a fatal error is returned, then file processing will end immediately. A restart will be initiated from the file pointer position saved in the restart_bookmark string at the time of the last commit point that was reached during file processing.

Program Flow

N/A

Shared Modules

validate_all_numeric: intrface library function.

validate_all_numeric_signed: intrface library function.

valid_date: intrface library function.

ORDER_ATTRIB_SQL.DELIVERY_MONTH: called from consignment_data(), returns order delivery month into the :invoices variable.

VAT_SQL.GET_VAT_RATE: called from pack_check(), returns the composite vat rate for a packitem.

CURRENCY_SQL.CONVERT: returns the converted monetary amount from

Currency to currency.

NEW_ITEM_LOC: called from item_check() and pack_check(), creates a new item if one doesn’t already exist for the item/location passed in.

UPDATE_SNAPSHOT_SQL.EXECUTE: called from update_snapshot(), updates the stake_sku_loc and edi_daily_sales tables for late transactions. If the item is a return, edi_daily_sales will not be updated.

NEXT_ORDER_NO: called from consignment_data(), returns the next available generated order number.

STKLDGR_SQL.TRAN_DATA_INSERT: called from consignment_data(), performs tran_data inserts (tran_type 20) for a consignment transaction.

Posupld and VAT:

There are three different data sources in POSUPLD.

1 the input file

2 RMS stock ledger tables (tran_data in this context)

3 RMS base tables (other that stock ledger)

Each of these data sources can be vat inclusive or vat exclusive.

There are five different system variables that are used to determine whether of not the different inputs are vat inclusive or vat exclusive.

1 system_options.vat_ind (assume Y for this document)

2 system_options.class_level_vat_ind

3 system_options.stkldgr_vat_incl_retl_ind

4 class.class_vat_ind

5 store.vat_include_ind (this is retrieved from the table when RESA is on and read from the input file when RESA is off)

Given the three different data source and all combinations of vat inclusive or vat exclusive, we are left with the 8 potential combinations of inputs to POSUPLD.

Possible POSUPLD inputs			
SCENARIO	**FILE**	**RMS**	**STOCK LEDGER**
1	Y	Y	Y
2	Y	Y	N
3*	Y	N	Y
4*	Y	N	N
5	N	Y	Y
6	N	Y	N
7	N	N	Y
8	N	N	N

* Scenarios 3 and 4 are not possible – the file will never have vat when RMS does not.

The combinations of system variables and the resulting scenarios:

System_options Class_level_vat_ind	System_options Stkldgr vat ind	Class Class_vat_ind	Store Vat_include_ind	Resulting Scenario
Y	Y	Y	Y - Ignored	1
Y	Y	Y	N - Ignored	1
Y	Y	N	Y - Ignored	7
Y	Y	N	N - Ignored	7
Y	N	Y	Y - Ignored	2
Y	N	Y	N - Ignored	2
Y	N	N	Y - Ignored	8
Y	N	N	N - Ignored	8
N	Y	Y – Ignored	Y	1
N	Y	Y – Ignored	N	5
N	Y	N – Ignored	Y	1
N	Y	N – Ignored	N	5
N	N	Y – Ignored	Y	2
N	N	Y – Ignored	N	6

System_options Class_level_vat_ind	System_options Stkldgr vat ind	Class Class_vat_ind	Store Vat_include_ind	Resulting Scenario
N	N	N – Ignored	Y	2
N	N	N – Ignored	N	6

POSUPLD table writes

Scenario 1:

tran code 1 from file retail.

tran code 2 from file retail with vat removed.

retail from file is compared directly with price_hist for off retail check.

Scenario 2:

tran code 1 from file retail with vat removed.

tran code 2 not written.

retail from file is compared directly with price_hist for off retail check.

Scenario 5:

tran code 1 from file retail with vat added.

tran code 2 from file retail.

retail from file has vat added for compare with price_hist for off retail check.

Scenario 7:

tran code 1 from file retail with vat added.

tran code 2 from file retail.

retail from file is compared directly with price_hist for off retail check.

Scenario 8:

tran code 1 from file retail.

tran code 2 not written.

retail from file is compared directly with price_hist for off retail check.

Function Level Description

Declarations:

declare input structures: file header (only date and type) & detail (all fields)

init()

initialize restart recovery

open input file (posupld)

    - file should be specified as input parameter to program

fetch system variables, including the SYSTEM_OPTIONS.CLASS_LEVEL_VAT_IND.

Retrieve all valid promotion types

declare final output filename (used in restart_write_file logic)

open reject file (as a temporary file for restart)

file should be specified as input parameter to program

open lock reject file (as a temporary file for restart)

- file should be specified as input parameter to program

call restart_file_init logic

assign application image array variables- line counter (g_l_rec_cnt), reject counter (g_l_rej_cnt), lock reject file counters (pl_lock_cnt, pl_lock_dtl_cnt), store, transaction_date

if fresh start (l_file_start = 0)

read file header record (get_record)

write FHEAD to lock reject file

if (record type <> 'FHEAD') Fatal Error

validate file type = 'POSU'

else fseek to l_file_start location

validate location and date are valid

set restart variables to ones from restart image file_process()

This function will perform the primary processing for transaction records retrieved from the input file. It will first perform validation on the THEAD record that was fetched. If the transaction was found to be invalid, a record will be written to the reject file, a non-fatal error will be returned, and the next transaction will be fetched.

Next, the unit retail from price_hist will be fetched by calling the get_unit_retail() function. The retail retrieved from this function will be compared with the actual retail sent in from the input file to determine any discrepencies in sale amounts.

Fetch all of the TDETL records that exist for the transaction currently being processed until a TTAIL record is encountered. Perform validation on the transaction detail records. If a detail record is found to be invalid, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. If a valid promotion type (code for mix & match, threshold promotions, etc.) was included in the detail record and it is not an employee disc record, write a record to the daily_sales_discount table. If it is an employee discount record write an employee discount record to tran_data. Finally, accumulate the discount amounts for all transaction detail records for the current transaction, unless the record was an employee discount.

Call the item_process() function to perform item specific processing. Once all records have been processed, write FTAIL record to lock reject file and call posting_and_restart to commit the final records processed since the last commit and exit the function.

item_process()

Check to see if any validation failed for the item before this function was called. If a lock error was found, call write_lock_rej() then return. If an other error was found, call write_rej() and process_detail_error() then return.

Set the item sales type for the current transaction. Valid sales types are 'R'egular sales, 'C'learance sales, and 'P'romotional sales. These will be used when populating the sales types for the item-location history tables. If an item is both on promotion and clearance, the transaction will be written as a clearance transaction.

If the system's VAT indicator is turned to on, VAT processing will be performed. The function vat_calc() will retrieve the vat rate and vat code for the current item-location. The total sales including and excluding VAT will be calculated for use in writing transaction data records. If any VAT errors occur, the entire transaction will be written to the reject file, a non-fatal error will be returned, and the next record will be fetched. A record will be written to vat_history for the item, location, transaction date.

Calculate the item sales totals (i.e. total retail sold, total quantity sold, total cost sold, etc.). If VAT is turned on in the system, calculate exclusive and inclusive VAT sales totals.

Calculate any promotional markdowns that may exist by calling the calc_prom_totals() function. The markdown information calculated here will be used when writing tran_data (tran_type 15) records for promotional markdowns.

Calculate the over/under amount the item was sold at compared to it's price_hist record. Since we do not create price_hist records of type 9 (promotional retail change) when the system_options.multi_prom_ind = 'Y', we do not know what the promotional retail for this item is. Therefore, we will take the total sales reported from the header record plus the total of sales discounts reported in the TDETL records, divided by the total sales quantity for the item to calculate its unit retail. If the system_options.multi_prom_ind = 'N', we can do a comparison of the price_hist record and the unit retail (total retail / total sales) inputted from the POS file. Any difference using either method will write to the daily_sales_discount table with a promotion type of 'in store' and tran_data (tran_type 15) If the transaction is a return, no daily_sales_discount record will be written, and tran_data records will be written as opposite of what they were sold as (i.e. if the sale was written as a markup, which would be written as a negative retail with a tran_data 15, the return would be written as a 15 with a positive retail).

If the item is a packitem and the transaction is a Sale, the process_pack() function will update the last_hist_export_date field on the item_loc_soh table to the transaction date and the item_loc_hist table will be updated with the transaction information.

If the item currently being processed is a packitem, calculate the retail markdown the item takes for being included in the pack and write a transaction data record as a promotional markdown. This markdown is calculated by comparing the retail contribution of the packitem's component item to the packitem to the component item's regular retail found on the price_hist table. The retail contribution for a component item is calculated by taking the component item's unit retail from price_hist, divided by the total retail of all component items in the packitem, and multiplying the packitem's unit retail. So if the retail contribution of a component item within packitem A is $10, and the same component item's price_hist record has a retail of $14, and there is only one packitem sold, and this component item has a quantity of one, a tran_data

Record (tran_type 15) will be written for $4 (assume no vat is used).

Write transaction data records for sales and returns. If the transaction is a sale, write a tran_data record with a transaction code of 1 with the total sales. If the system VAT indicator is on and the system_options.stkldgr_vat_incl_retl_ind is on, write a tran_data record with a transaction code of 2 for VAT exclusive sales. If the transaction is a return, write a tran_data record (tran_type 1) with negative quantities and retails for the amount of the return. If the system VAT indicator is on and the system_options.stkldgr_vat_incl_retl_ind is on, write a tran_data record (tran_type 2) and negative quantities and retails for the VAT exclusive return. Also, write a tran_data record with a transaction code of 4 for the total return. Any tran_data record that is written should be either VAT exclusive or VAT inclusive, depending on the system_options.stkldgr_vat_incl_retl_ind. If it is set to 'Y', all tran_data retails should be VAT inclusive. If it is set to 'N', all tran_data retails should be VAT exclusive. When writing tran_data records for packitems, always break them down to the packitem level, writing the retail as the packitem multiplied by the component item's price ratio. The packitem itself should never be inserted into the tran_data table.

If the transaction is late (transaction date is before the current date) and it is not a drop shipment, call update_snapshot() to update the stake_sku_loc and edi_daily_sales tables. If the transaction is current, update the edi_daily_sales table only (stake_sku_loc will be updated in a batch program later down the stream). The edi_daily_sales table should only be updated if the items supplier edi sales report frequency = 'D'.

If VAT is turned on in the system, write a record to the vat_history table to record the vat amount applied to the transaction. The VAT amount is calculated by taking the sales including VAT minus the sales excluding VAT.

Update the sales history tables for non-consignment items that are Sale transactions. Do not update for returns. Also, update stock count on the item-location table for Sales and Returns unless the item is on consignment or is drop shipped.

If the dropship indicator is set to 'Y', then the sale is drop shipped and there is no update for stock on hand. Drop shipments are used for sales at a virtual or physical location where an order is taken from a customer, but the goods are shipped directly from the vendor to the customer (not via any store or warehouse owned by the retailer). If an item is used only for drop shipments and there is no stock on hand before or after the cost price is changed, the weighted average cost is never updated when average cost accounting method is used. The average cost will be the initial cost price at the time the item is set up. Over a period of time, under average cost accounting method, the cost price used to charge these items will drift away from the actual supplier cost. See SYSTEM_OPTIONS.STD_AV_IND for further details on cost accounting method.

If an off_retail amount was identified for the item/location, call the write_off_retail_markdowns() function to write tran_data records (tran_type 15) to record the difference. If the system_options.multi_prom_ind = ‘N’ and the item is on promotion, or if the system_options.multi_prom_ind = ‘Y’ and the TDETL total discount amount is greater than zero, write a promotional markdown. Note: this will also record a tran_data record (tran_type 15) for a TDETL record that has a promotional transaction type with no promotion number in order to record the markdown.

If an employee discount TDETL record has been encountered, a tran_data record with tran_code 60 will be written.

If the item is a wastage item, a tran_data record with tran_code 13 will be written. This record is used to balance the stock ledger, it accounts for the amount of the item that was wasted in processing.

process_detail_error()

This function writes a record to the load_err table for every non-fatal error that occurs.

set_counters()

Depending on the action passed into this function, it will either set a savepoint and store the values of counters or rollback a savepoint and reset the values of certain counters back to where they were originally set. This function is called when a non-fatal error occurs in the item_process() function to rollback and changes that may have been made.

calc_item_totals()

This function will set total retail and discount values including and excluding VAT, depending upon the store.vat_include_ind, system_options.vat_ind, system_options.multi_prom_ind, and the system_options.stkldgr_vat_incl_retl_ind.

calc_prom_totals()

This function will set promotional markdown values including and excluding VAT, depending upon the system_options.multi_prom_ind and the system_options.stkldgr_vat_incl_retl_ind. If the multi_prom_ind is on, the promotional markdown is the sum of the TDETL discount amounts. If the multi_prom_ind is off, the promotional markdown is the difference between the price_hist record with a tran_code of 0,4,8,11 and the price_hist record with a tran_code of 9 multiplied by the total sales quantity. Also, the tran_data old and new retail fields are only written if the multi_prom_ind is off.

process_sales_and_returns()

If the item is on consignment and not a packitem, the consignment_data() function will be called to perform consignment processing. The function write_tran will be called to write a tran_data record with a tran_type 1 (always written), a tran_type 2 (if the system_options.stkldgr_vat_incl_retl_ind = Y), and a tran_type 4 (if the transaction was a return). If the transaction is a return, any tran_data records with tran_types of 1 and 2 will be written with negative retails. Also the update_price_hist() function will be called to update the most recent price_hist record.

posting_and_restart()

Post all array records to their respective tables and call restart_file_commit to perform a commit the records to the database and restart_file_write to append temporary files to output files.

validate_FHEAD()

Do standard string validations on input fields. This includes null padding fields, checking that numeric fields are all numeric, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the store location exists.

If the sales audit indicator is on currency and vat information will be provided in the file that has already been validated.

validate_THEAD()

Do standard string validations on input fields. This includes null padding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true. This function will also validate the reference item exists.

If a reference item is passed in from the input file, retrieve the item for the reference item. Once the item is an item, retrieve the tranasaction and item level values, pack indicator, department, class, subclass, waste_type, waste_pct. Once this information is retrieved, check that the item/location relationship exists for the appropriate item type and call check_item_lock() and/or check_pack_lock depending on item type to lock this item’s ITEM_LOC record.

If the sale audit indicator is ‘Y’ on system_options, the item will be a item and the dept, class, subclass, item level, transaction level and pack_ind will be included in the file. The UOM is assumed to already by have been converted to the standard UOM by Sales Audit.

If the Sales Audit indicator is 'N' on system_options, the UOM at which the item was sold will be compared with the items standard UOM value. If they are different, the quantity will be converted to the standard UOM amount. The ratio of the difference will also be computed and saved for use by validate_TDETL().

If an item is a wastage item set the wastage qty. The qty sent in the file shows the weight of the item sold. The wastage qty is the qty that was processed to come up with the qty sold. So if .99 of an item was sold, and item wastage percent is 10. The wastage qty is .99 / (1-.10) = 1.1 The wastage qty will be used through out the program except when writing tran_data records(see write_wastage_markdown) and daily_sales_discount records which will uses the processed qty from the file.

Class-level vat functionality is addressed here. The c_ get_class_vat cursor is fetched into the pi_vat_store_include_ind variable if vat is tracked at the class level in RMS (SYSTEM_OPTIONS.VAT_IND = 'Y' and SYSTEM_OPTIONS.CLASS_LEVEL_VAT_IND = 'Y'). The vat inclusion indicator passed in the input file is overwritten with the vat indicator for the class passed in the THEAD record of the input file.

Check_item_lock

This function will lock this item/location's record in the RMS item_loc table. Returns a lock error if lock failed due to contention, otherwise returns 0 if no errors occurred, or fatal if other errors occurred.

Check_pack_lock

This function will call check_item_lock for every component item of the current pack item.

validate_TDETL

This function will perform validation on the TDETL records passed into the program. The standard string validation on these fields includes null padding fields, left shifting fields, checking that numeric fields are all numeric, placing decimal in all quantity and value fields, and validating the date field. If any errors arise out of these validation checks, return non-fatal error then set non-fatal error flag to true.

The quantity is multiplied by the UOM ratio determined in validate_THEAD().

If a promotional transaction type is passed in, verify it is valid. If a promotional transaction type is passed in, but it is not valid, return non-fatal error then set non-fatal error flag to true. If a promotion number is passed in, validate it by checking the promhead table and set the promotional indicator to True.

If the item is a wastage item set the tdetl wastage qty. This is done the same way as setting the THEAD wastage qty.

New_item_loc

This function creates a new store item relationship for items. It is called by item_check.

item_store_cursors

This function checks the item_loc for the item / store combination. It is called by the item_check function.

item_check

This function verifies the fashion item/location relationship exists. It is only called when the item being processed is a fashion item. If the item/location relationship does not exist, it is created and a record is written to the Invalid item/location output file.

New_pack_loc

This function creates a new store item relationship for pack items. It is called by pack_check.

pack_check

This function verifies the pack item/location relationship exists and retrieves the component items for the packitem. It is only called when the item being processed is a packitem. The component item, system indicator, department, class, subclass, cost, retail, price_hist retail, and component item quantity are fetched. If the packitem/location relationship does not exist, it is created for the Packitem and all of its components and a record is written to the Invalid item/location output file for the packitem.

The component items price ratios are also calculated. This indicates the retail contribution the component item gives towards the unit retail of the packitem. This ratio is calculated by taking the price_hist unit retail of the component divided by the total price_hist retail of all the component items for the packitem. Below is an example of how this ratio is calculated:

	Unit Retail	Qty	Retail	Calculation	Ratio
packitem A	$60				
item 1	$15	2	$30	($30/$90) * $60	.3333
item 2	$10	6	$60	($60/$90) * $60	.6667

get_unit_retail

This function retrieves the current unit retail and the retail price of the item at the time of the sale from price_hist for the item/location being processed. If a tran_code of 8 is returned, the item is on clearance. The function will always return retail that are vat inclusive. If retail is stored in RMS with out vat (system_options.class_level_vat_ind = Y and class.class_vat_ind = Y) it will add vat to the retails.

process_packitems

This function performs processing for the component items of the packitems. This would include updates/inserts into stake_item_loc, edi_daily_sales, item_loc, item_loc_hist, vat_history_data, and tran_data. All of these tables do not write records at the packitem level, but at the component item level. When figuring retails to write to these tables, the component items price ratio should always be applied against the packitems retail to come up with the correct retail for each component item. If an employee discount TDETL record has been encountered, an tran_data record with tran_code 60 will be written for each component item.

process_daily_sales_discount()

This function will insert/update a record to daily_sales_discount for each TDETL record that has a promotional transaction type except employee discounts. Employee discount records are not written to daily_sales_discount, they are put on tran_data with a tran_code of 60. When employee discount records are encountered, values are set for the tran_data insert and the discount amount is added to the total sales value. This is done so employee discounts do figure into the promotional and in store calculations. When the multi_prom_ind is on all promotion types except employee discount will be ignored.

write_in_store()

This function will handle record sent in as ‘is store’ discounts amounts. It will call check_daily_exist and daily_sales_insert_update.

Remove_stklgdr_vat()

This fuction will remove vat from 3 fields after the dailiy_sales_discount processing is complete. The variables od_off_retail_amt, od_new_retail, and od_old_retail are stripped of vat by calling vat_convert if the stock ledger does not contain vat.

Write_off_retail()

This function will calculate discrepancies between the amount sold for an item, and the amount it should have sold for (price_hist record). If these amounts are not in balance, a record is written to the daily_sales_discount table with a prom_type of ‘in store’ for reporting.

Daily_sales_exist()

This function will check the daily_sales_discount for the existence of a record matching the input parameters

Daily_sales_insert_update()

This function is called by write_off_retail, write_in_store, and process_daily_sales_discount. It performs the actual insert or fills a update array for the daily_sales_discount table.

write_off_retail_markdown()

The write_tran_data() function will be called to write the off_retail markdown unless the item is on consignment or the off_retail amount is zero.

write_promotional_markdown()

The write_tran_data() function will be called to write the promotional markdown unless the item multi_prom_ind is off and the transaction is a return, the item is on consignment, or the promotional markdown amount is zero. The tran_data new and old retails are only written if the multi_prom_ind is off.

Write_wastage_markdown()

This function will call to the write_tran_data() function if the item is a wastage item. A wastage item is an item that loses some of its weight (value) in processing. For example, a 1 pound chicken is broiled and loses 10% of its weight. The item is sold at .9 pounds, but in reality selling that .9 pounds of chicken removes 1 pound of chicken from the inventory. This function writes a tran_code 13 tran_data record to account for the amount of the chicken that was lost due to wastage in processing.

vat_convert()

This function will either add or remove vat from a retail value.

process_items()

Update the stock on hand on the item_loc_soh table for Sales and Returns unless the item is on consignment or is drop shipped. Also, update the item_loc_hist table for Sale transactions. Do not update for returns.

process_pack()

Update the stock on hand on the item_loc_soh table for Sales and Returns. Also, update the item_loc_hist table for Sale transactions. Do not update for returns.

write_tran_data()

Writes a record to the tran_data insert array.

Write_edi_daily_sales()

Writes a record to edi_daily_sales.

update_snapshot()

Calls the UPDATE_SNAPSHOT_SQL.EXECUTE function to update the stake_sku_loc and edi_daily_sales tables for late transactions.

write_vat_err_message()

This function will create and write to the VAT output file when an item does not have VAT infomation setup when it is expected.

vat_history_data()

Writes a record to the vat_history table.

consignment_data()

This function will perform processing for consignment items. Consignment items are such when the item_supplier table has a consignment rate applied to it. Consignment is when a retailer will allow a third party to operate under its umbrella and be paid for what it sells. An example of consignment may be a mass-merchant who consigns the magazine section of their store to a magazine vendor. The magazine vendor would have control over keeping the product stocked within the store. When a magazine is sold, the retailer would get paid for the magazine, then the retailer would essentially buy the magazine from the vendor. The consignment cost paid by the retailer to the vendor is the VAT-inclusive retail multiplied by the consignment rate divided by 100. So if the VAT-inclusive retail price of a magazine was $10 and the consignment rate was 50, the consignment cost would be $5.

Also a completed order to the vendor should be found/created for the supplier with an orig_ind = 4 (consignment). Consignment type invoices will be created for all PO's created for consignments

Also a tran_data record (tran_type 20) will be written to record the consignment transaction to the stock ledger. The retails should be VAT inclusive or exclusive, depending on the system_options.stkldgr_vat_incl_retl_ind.

This function uses support functions: check_order(), order_head(), invc_data(), to handle the order creation-update and the invoice creation-update.

get_prom_type_info()

This function will retrieve all valid promotional transaction types from the code_detail table. Valid promotional transaction types are those where the code_type = 'PRMT'.

fill_packitem_array()

This function will retrieve the component items for a packitem with the appropriate item level information into an array.

Write_lock_rej

This function will write the current record set from the input file (THEAD-{TDETL}-TTAIL) that was rejected due to lock error to the lock file.

write_item_store_report()

This function will create and write to the Invalid item/location output file when an item does not exist at a location it was sold/returned at.

ON Fatal Error

- Exit Function with -1 return code

ON Non-Fatal Error

- write out rejected record to the reject file using write_to_rej_file functionby passing pointer to detail record structure, number of bytes in structure, and reject file pointer, or use the write_lock_rej() function to write to the lock reject file in case the non-fatal error was a lock error,

Input File

The input file should be accepted as a runtime parameter at the command line. All number fields with the number(x,4) format assume 4 implied decimal included in the total length of 'x'.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: Vat include indicator, Vat region, Currency code, and Currency retail decimals. When the sa_ind is 'N' these values will not be used and retrieved from the system.

When the system_options field sa_ind is 'Y' the following FHEAD fields will be populated and already validated: Item Level, Transaction Level, Pack_ind, Dept, Class, and Subclass. When the sa_ind is 'N' these values will not be used and retrieved from the system. Also, the UOM at which the item was sold will been converted to the standard UOM for the item. When the sa_ind is on, all items are assumed to be items.

Record Name	Field Name	Field Type	Default Value	Description
File Header	File Type Record Descriptor	Char(5)	FHEAD	Identifies file record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	File Type Definition	Char(4)	POSU	Identifies file as 'POS Upload'
	File Create Date	Char(14)	create date	date file was written by external system
	Location Number	Number(10)	specified by external	Store identifier

Record Name	Field Name	Field Type	Default Value	Description
			system	
	Vat include indicator	Char(1)		Determines whether or not the store stores values including vat. Not required but populated by Retek sales audit
	Vat region	Number(4)		Vat region the given location is in. Not required but populated by Retek sales audit
	Currency code	Char(3)		Currency of the given location. Not required but populated by Retek sales audit
	Currency retail decimals	Number(1)		Number of decimals supported by given currency for retails. Not required but populated by Retek sales audit
Transaction Header	File Type Record Descriptor	Char(5)	THEAD	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Date	Char(14)	transaction date	date sale/return transaction was processed at the POS
	Item Type	Char(3)	REF ITM	item type will be represented as a REF or ITM
	Item Value	Char(25)	item identifier	the id number of an ITM or REF
	Dept	Number(4)	Item's dept	Dept of item sold or returned. Not

Record Name	Field Name	Field Type	Default Value	Description
				required but populated by Retek sales audit
	Class	Number(4)	Item’s class	Class of item sold or returned. Not required but populated by Retek sales audit
	Subclass	Number(4)	Item’s subclass	Subclass of item sold or returned. Not required but populated by Retek sales audit
	Pack Indicator	Char(1)	Item's pack indicator	Pack indicator of item sold or returned. Not required but populated by Retek sales audit
	Item level	Number(1)	Item's item level	Item level of item sold or returned. Not required but populated by Retek sales audit
	Tran level	Number(1)	Item's tran level	Tran level of item sold or returned. Not required but populated by Retek sales audit
	Wastage Type	Char(6)	Item’s wastage type	Wastage type of item sold or returned. Not required but populated by Retek sales audit
	Wastage Percent	Number(12)	Item’s wastage percent	Wastage percent of item sold or returned. Not required but populated by Retek sales audit
	Transaction Type	Char(1)	‘S’ – sales ‘R’ - return	Transaction type code to specify whether

Record Name	Field Name	Field Type	Default Value	Description
				transaction is a sale or a return
	Drop Shipment Indicator	Char(1)	'Y' 'N'	Indicates whether the transaction is a drop shipment or not. If it is a drop shipment, indicator will be 'Y'. This field is not required, but will be defaulted to 'N' if blank.
	Total Sales Quantity	Number(12)		Number of units sold at a particular location with 4 implied decimal places.
	Selling UOM	Char(4)		UOM at which this item was sold.
	Sales Sign	Char(1)	'P' - positive 'N' - negative	Determines if the Total Sales Quantity and Total Sales Value are positive or negative.
	Total Sales Value	Number(20)		Sales value, net sales value of goods sold/returned with 4 implied decimal places.
	Last Modified Date	Char(14)		For VBO future use
Transaction Detail	File Type Record Descriptor	Char(5)	TDETL	Identifies transaction record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Promotional Tran Type	Char(6)	promotion type – valid values see	code for promotional type from code_detail,

Record Name	Field Name	Field Type	Default Value	Description
			code_detail table.	code_type = 'PRMT'
	Promotion Number	Number(10)	promotion number	promotion number from the RMS
	Sales Quantity	Number(12)		number of units sold in this prom type with 4 implied decimal places.
	Sales Value	Number(20)		value of units sold in this prom type with 4 implied decimal places.
	Discount Value	Number(20)		Value of discount given in this prom type with 4 implied decimal places.
Transaction Trailer	File Type Record Descriptor	Char(5)	TTAIL	Identifies file record type
	File Line Identifier	Char(10)	specified by external system	ID of current line being processed by input file.
	Transaction Count	Number(6)	specified by external system	Number of TDETL records in this transaction set
File Trailer	File Type Record Descriptor	Char(5)	FTAIL	Identifies file record type
	File Line Identifier	Number(10)	specified by external system	ID of current line being processed by input file.
	File Record Counter	Number(10)		Number of records/transactions processed in current file (only records between head & tail)

Invalid Item/Store File:

The Invalid Item/Store File will only be written when a transaction holds an item that does not exist at the processed location. In the event this happens, the relationship will be created during the program execution and processing will continue with the item and store number being written to this file for reporting.

VAT File:

The VAT file will only be written if a particular item cannot retrieve a VAT rate when one is expected (e.g. the system_options.vat_ind is on). In this event, a non-fatal error will occur against the transaction and a record will be written to this file and the Reject file.

Reject File:

The reject file should be able to be re-processed directly. The file format will therefore be identical to the input file layout. The file header and trailer records will be created by the interface library routines and the detail records will be created using the write_to_rej_file function. A reject line counter will be kept in the program and is required to ensure that the file line count in the trailer record matches the number of rejected records. A reject file will be created in all cases. If no errors occur, the reject file will consist only of a file header and trailer record and the file line count will be equal to 0.

A final reject file name, a temporary reject file name, and a reject file pointer should be declared. The reject file pointer will identify the temporary reject file. This is for the purposes of restart recovery. When a commit event takes place, the restart_write_function should be called (passing the file pointer, the temporary name and the final name). This will append all of the information that has been written to the temp file since the last commit to the final file. Therefore, in the event of a restart, the reject file will be in synch with the input file.

Error File:

Standard Retek batch error handling modules will be used and all errors (fatal & non-fatal) will be written to an error log for the program execution instance. These errors can be viewed on-line with the batch error handling report.

Technical Issues

Assumption: Variable weight UPCs are expected to already be converted to a VPLU with the appropriate quantity.

Chapter 4 – saexprdw.pc

Introduction

Purpose

The Batch Detailed Design is a thorough definition of a single batch program / module within one functional area. The documented information is derived from this functional area's Technical Design.

Objectives

This Batch Detailed Design must:

- Document specific functions for a single batch program,
- Enable project team review, validation and consensus regarding the individual batch program's scope,
- Document the batch program in preparation for and in response to prototyping, and
- Prepare for and provide a defined and documented framework in which to perform Development Phase activities.

Functional Area

Design Overview

The purpose of this batch module is to fetch all corrected sale and return transactions that do not have RDW errors from the Retek Sales Audit (ReSA) database tables for transmission to the Retek Data Warehouse (RDW). The data will be sent at the store day level. If the transaction has a status of Deleted and it has previously been transmitted, a reversal of the transaction will be sent.

Four files of type RDWT, RDWF, RDWS and RDWC will be created for each store_day. See the file Interface File – SA to RDW.doc for more information.

RDW requires that the employee id be sent. saexprdw is expected to do this by mapping a cashier ID to an employee ID using the sa_store_emp table. However, the latter may not always be populated and thus, we send a blank field to RDW in this case.

Table	Operations Performed			
	Select	Insert	Update	Delete
sa_store_day	Yes	No	No	No
sa_export_log	Yes	No	Yes	No
sa_error	Yes	No	No	No
sa_error_impact	Yes	No	No	No
sa_tran_head	Yes	No	No	No
sa_tran_item	Yes	No	No	No
sa_tran_disc	Yes	No	No	No
sa_tran_tender	Yes	No	No	No
sa_customer	Yes	No	No	No
sa_tran_head_rev	Yes	No	No	No
sa_tran_item_rev	Yes	No	No	No
sa_tran_disc_rev	Yes	No	No	No
sa_tran_tender_rev	Yes	No	No	No
sa_store_emp	Yes	No	No	No
sa_total	Yes	No	No	No
sa_exported	Yes	Yes	No	No
sa_exported_rev	Yes	No	No	No

Program Flow

This is a detailed diagram (structure chart type or function level text) that will define all functions performed in the module. This section will require the designer to specify the program flow and activities performed in each code segment without detailing SQL.

Thoroughly analyze the module with the sole objective of achieving the most efficient implementation. Uncover parallelism inherent in the application. Identify functional modules executed concurrently and the dependencies between these modules. In addition, identify standard computational modules, scope, performance, scheduling constraints, common functions, maintainability, and overall module integrity.

This should provide an overall picture of the LUW performed by the batch module, derived from the Technical Design.

Global Variable Descriptions

Gobal Variable	Description
pl_commit_max_ctr	Commit max counter used for array fetches.
ps_sysdate	Current sysdate value from the database.
ps_store	Store ID from store/day driving cursor.
ps_business_date	Business date from store/day driving cursor.
ps_temp_rdwtfile	Temporary file name to be used for the RDWT file.
ps_temp_rdwffile	Temporary file name to be used for the RDWF file.
ps_temp_rdwsfile	Temporary file name to be used for the RDWS file.
ps_temp_rdwcfile	Temporary file name to be used for the RDWC file.
pi_curtrat	Current transactions transaction type converted to an enum.
pi_tdetl_count	TDETL record count for TTAIL record in the RDWT file.
ps_total_sales_value	Total sales value of a TITEM record minus any discounts from associated IDISC records.
pl_rdwc_line_ctr	Line counter for the RDWC file.
pl_rdwf_line_ctr	Line counter for the RDWF file.
pl_rdws_line_ctr	Line counter for the RDWS file.
pl_rdwt_line_ctr	Line counter for the RDWT file.
RDWFFile	File pointer for the RDWF file.
RDWTFile	File pointer for the RDWT file.
RDWSFile	File pointer for the RDWS file.
RDWCFile	File pointer for the RDWC file.
pi_num_locks_not_released	Counter for the number of store/day locks that could not be released.
pi_num_non_fatal_errors	Counter for the number of non-fatal errors encountered: Store/day lock could not be release. An unexpected total was encountered. Could not translate a cashier POS ID to an employee ID. Could not translate a salesperson POS ID to an employee ID.

Function Level Description

main()

int argc

char *argv[]

Check command line for required arguments.

Call LOGON to connect to the database.

Call Init to initialize the program.

Call process to export the available RDW data.

Report unlocking errors.

Report non-fatal errors.

Call final to cleanup.

init()

No arguments

This function initializes Restart recovery.

Get the value of sa_system_options.unit_of_work by calling the library function fetchSaSystemOptions.

Initialize Oracle Number functions by calling OraNumInit.

Get temporary filenames to use for generating the output files. Store these names in ps_temp_rdwtfile, ps_temp_rdwffile, ps_temp_rdwsfile, and ps_temp_rdwcfile.

process()

No arguments

Picks a store/day to be processed by fetching using the first driving cursor. Save the store ID in ps_store and the date in ps_business_date.

Attempt to lock the store/day with a call to get_lock. If this fails, go on to the next store/day.

Open RDWTFile, RDWSFile, RDWCFile and RDWFile, using temporary names generated in init.

Set pl_rdwc_line_ctr, pl_rdwf_line_ctr, pl_rdws_line_ctr and pl_rdwt_line_ctr to 0.

Call fetchSysDate to get the current date/time. Store it in ps_sysdate.

Call WrRDWFHead to write a RDWT FHEAD record to the RDWT file.

Call WrRDWFHead to write a RDWF FHEAD record to the RDWF file.

Call processStoreDay to process the store/days transactions.

Call WrOutputData to write the data in memory to the appropriate file.

Increment pl_rdwt_line_ctr.

Call WrRDWFTail to write a RDWT FTAIL record to the RDWT file.

Call WrRDWFTail to write a RDWF FTAIL record to the RDWF file.

Call processStoreDayTotals to process all totals for a given store day.

Update the status in sa_export_log to Complete by calling the library function markStoreDayExported.

Close the RDWTFile, RDWFFile, RDWSFile and RDWCFile and rename them appropriately (file-type_store_business-date_current-datetime).

Call to release_lock and go on to the next store/day. This function commits as a side effect, thus committing the changes to the database.

final()

int ii_process_ret

Remove the temporary file, if we failed to finish (ii_proces_ret is not OK).

Call retek_close.

Call retek_refresh_thread.

processStoreDay()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

For each transaction from the store/day being processed, get the following information from the second driving cursor and call processTransHead with the information.

Table	Column	Description
Sa_tran_head	Tran_seq_no	
Sa_tran_head	Rev_no	
Sa_tran_head	Tran_datetime	Format YYYYMMDDHH24MISS
Sa_tran_head	Tran_no	
Sa_tran_head	Register	
Sa_store_emp	Emp_id	Pos_id = cashier via an outer join separate from salesperson
Sa_store_emp	Emp_id	Pos_id = salesperson via an outer join separate from cashier
Sa_customer	Cust_id_type	via an outer join
Sa_customer	Cust_id	via an outer join
Sa_tran_head	Reason_code	
Sa_tran_head	Tran_type	
Sa_tran_head	Sub_tran_type	
Sa_tran_head	Orig_tran_no	
Sa_tran_head	Orig_reg_no	
Sa_tran_head	Ref_no1	
Sa_tran_head	Ref_no2	
Sa_tran_head	Ref_no3	
Sa_tran_head	Ref_no4	
Sa_tran_head	Vendor_no	
Sa_tran_head	Status	
Sa_tran_head	Value	'SIGN_N' or 'SIGN_P' depending on the sign of value.
Sa_tran_head	Value	Absolute value multiplied by 10000.
	Transaction Sign	'SAFD_P' if the transaction has not been deleted (status != 'SAST_D') and there are no errors and it has not been exported. 'SAFD_N' if the transaction has been deleted (status = 'SAST_D') and it has been exported after being exported.
Sa_exported	Exp_datetime	Only for transactions with a Transaction Sign of 'SAFD_N'. Format YYYYMMDDHH24MISS

Calls the library function **markTransactionExported** to insert a record into sa_exported for each transaction.

processTransHead()

char is_store_day_seq_no[NULL_BIG_SEQ_NO] struct pt_sa_tran_head ir_sa_tran_head

If the transaction status is deleted (SAST_D) and it has been previously exported, then call retrieveTransHeadRev. Also, if the revision number of the transaction is not 1, then a previous revision may have been exported; call retrieveTransHeadRev to get the exported revision (for full disclosure purposes).

Call retrieveTransItem, retrieveTransDisc and retrieveTransTender to obtain the items, discounts and tenders for the transaction, both Positive transactions and Negative ones.

Call saveData for both the Positive and Negative transactions to write the information into the RDW files.

retrieveTransHeadRev()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

struct pt_sa_tran_head *or_sa_tran_head_rev

This function gets the sa_tran_head_rev record that needs to be processed. A record needs to be processed if it has been previously exported.

Table	Column	Description
Sa_tran_head_rev	Tran_seq_no	
Sa_tran_head_rev	Rev_no	
Sa_tran_head_rev	Tran_datetime	Format YYYYMMDDHH24MISS
Sa_tran_head_rev	Tran_no	
Sa_tran_head_rev	Register	
Sa_store_emp	Emp_id	Pos_id = cashier via an outer join separate from salesperson
Sa_store_emp	Emp_id	Pos_id = salesperson via an outer join separate from cashier
Sa_customer	Cust_id_type	via an outer join
Sa_customer	Cust_id	via an outer join
Sa_tran_head_rev	Reason_code	
Sa_tran_head_rev	Tran_type	
Sa_tran_head_rev	Sub_tran_type	
Sa_tran_head_rev	Orig_tran_no	
Sa_tran_head_rev	Orig_reg_no	
Sa_tran_head_rev	Ref_no1	
Sa_tran_head_rev	Ref_no2	
Sa_tran_head_rev	Ref_no3	

Table	Column	Description
Sa_tran_head_rev	Ref_no4	
Sa_tran_head_rev	Vendor_no	
Sa_tran_head_rev	Status	
Sa_tran_head_rev	Value	'SIGN_N' or 'SIGN_P' depending on the sign of value.
Sa_tran_head_rev	Value	Absolute value multiplied by 10000.
	Transaction Sign	'SAFD_N'
Sa_exported_rev	Exp_datetime	Only for transactions with a Transaction Sign of 'SAFD_N'. Format YYYYMMDDHH24MISS

If no data is found, than set or_sa_tran_head_rev->s_rev_no to –1.

retrieveTransItem()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_rev_no[NULL_SA_REV_NO]

long *ol_num_sa_tran_item

struct pt_sa_tran_item **or_sa_tran_item

This function gets all sa_tran_item records or sa_tran_item_rev (if is_rev_no is not –1) that need to be processed for a tran_seq_no.

Table	Column	Description
Sa_tran_item	Tran_seq_no	
Sa_tran_item	Item_seq_no	
Sa_tran_item	Item_status	
Sa_tran_item	Item	
Sa_tran_item	Ref_item	
Sa_tran_item	Non_merch_item	
Sa_tran_item	Voucher_no	
Sa_tran_item	Dept	
Sa_tran_item	Class	
Sa_tran_item	Subclass	
Sa_tran_item	Standard_qty	'SIGN_N' or 'SIGN_P' depending on the sign of qty.
Sa_tran_item	Standard_qty	Absolute value multiplied by 10000.

Table	Column	Description
Sa_tran_item	Standard_unit_retail	'SIGN_N' or 'SIGN_P' depending on the sign of unit_retail.
Sa_tran_item	Standard_unit_retail	Absolute value multiplied by 10000.
Sa_tran_item	Tax_ind	
Sa_tran_item	Item_swiped_ind	
Sa_tran_item	Standard_orig_unit_retail	'SIGN_N' or 'SIGN_P' depending on the sign of orig_unit_retail.
Sa_tran_item	Standard_orig_unit_retail	Absolute value multiplied by 10000.
Sa_tran_item	Item_type	
Sa_tran_item	Override_reason	
Sa_store_emp	Emp_id	
Sa_tran_item	Return_reason_code	
Sa_tran_item	Drop_ship_ind	

The same columns as above are select from the sa_tran_item_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_item to the total number of records fetched.

retrieveTransDisc()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_rev_no[NULL_SA_REV_NO]

long *ol_num_sa_tran_disc

struct pt_sa_tran_disc **or_sa_tran_disc

This function gets all sa_tran_disc or sa_tran_disc_rev records (if is_rev_no is not –1) for a tran_seq_no that needs to be processed.

Table	Column	Description
Sa_tran_disc	Tran_seq_no	
Sa_tran_disc	Item_seq_no	
Sa_tran_disc	Discount_seq_no	
Sa_tran_disc	Rms_promo_type	
Sa_tran_disc	Promotion	
Sa_tran_disc	Discount_type	
Sa_tran_disc	Coupon_no	

Table	Column	Description
Sa_tran_disc	Coupon_ref_no	
Sa_tran_disc	Standard_qty	'SIGN_N' or 'SIGN_P' depending on the sign of qty.
Sa_tran_disc	Standard_qty	Absolute value multiplied by 10000.
Sa_tran_disc Sa_tran_item	(Unit_retail * standard_qty) – (unit_discount_amt * qty)	Absolute value multiplied by 10000.
Sa_tran_disc Sa_tran_item	(Unit_retail * standard_qty) – (unit_discount_amt * qty)	'SIGN_N' or 'SIGN_P' depending on the sign of the expression.
Sa_tran_disc	Standard_unit_discount_amt	'SIGN_N' or 'SIGN_P' depending on the sign of unit_discount_amt.
Sa_tran_disc	Standard_unit_discount_amt	Absolute value multiplied by 10000.
Sa_tran_disc		

The same columns as above are select from the sa_tran_disc_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_disc to the total number of records fetched.

retrieveTransTender()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

char is_rev_no[NULL_SA_REV_NO]

long *ol_num_sa_tran_tender

struct pt_sa_tran_tender **or_sa_tran_tender

This function gets all sa_tran_tender or sa_tran_tender_rev records (if is_rev_no is not –1) for a tran_seq_no that needs to be processed.

Table	Column	Description
Sa_tran_tender	Tran_seq_no	
Sa_tran_tender	Tender_seq_no	
Sa_tran_tender	Tender_type_group	
Sa_tran_tender	Tender_type_id	
Sa_tran_tender	Tender_amt	'SIGN_N' or 'SIGN_P' depending on the sign of tender_amt.

Table	Column	Description
Sa_tran_tender	Tender_amt	Absolute value multiplied by 10000.
Sa_tran_tender	Cc_no	
Sa_tran_tender	Cc_auth_no	
Sa_tran_tender	Cc_auth_src	
Sa_tran_tender	Cc_cardholder_verf	
Sa_tran_tender	Cc_exp_date	Format YYYYMMDD
Sa_tran_tender	Cc_entry_mode	
Sa_tran_tender	Cc_term_id	
Sa_tran_tender	Cc_spec_cond	
Sa_tran_tender	Voucher_no	
Sa_tran_head Sa_voucher	Business_date – iss_date	Voucher age
Sa_voucher	Escheat_date	
Sa_tran_tender	Coupon_no	
Sa_tran_tender	Coupon_ref_no	

The same columns as above are select from the sa_tran_tender_rev table if the rev_no passed in is not –1.

Set *ol_num_sa_tran_tender to the total number of records fetched.

saveData()

struct pt_sa_tran_head ir_sa_tran_head

long il_num_sa_tran_item

struct pt_sa_tran_item *ia_sa_tran_item

long il_num_sa_tran_disc

struct pt_sa_tran_disc *ia_sa_tran_disc

long il_num_sa_tran_tender

struct pt_sa_tran_tender *ia_sa_tran_tender

Set pi_curtrat to the current transaction type by calling trat_lookup.

Call WrRDWTHead to process the current ia_sa_tran_head record if the transaction type (pi_curtrat) is TRATTT_COND, TRATTT_PAIDIN or TRATTT_PAIDOU.

For each item record:

Call tsv_lookahead to calculate the total sales value for later use.

Call WrRDWTHead to process the current ia_sa_tran_item record.

For each item's discount record:

Call WrRDWTDetl to process the current ia_sa_tran_disc record.

For each tender record:

Call WrRDWFDetl to process the current ia_sa_tran_tender.

Call WrRDWTTail to create a TTAIL record for the RDWT file.

ProcessStoreDayTotals()

char is_store_day_seq_no[NULL_BIG_SEQ_NO]

const char is_usage_type[NULL_CODE]

This function will loop through the library function getBalTotals for the current store day.

Call WrRDWFHead to write this header to the RDWS file.

Call WrRDWFHead to write this header to the RDWC file.

For each total returned:

1 If the total_id is "OVRSHT_B" then write the data to the RDWC file.

2 Else, if the cashier_id and the register_id are both nulls, then write to the RDWS file.

3 Else, mark this as an error, since the RDWS file can only handle store level totals.

4 If the total is not a 'N'egative total, mark the total exported by calling the library function **markTotalExported**.

Call WrRDWFTail to write this header to the RDWS file.

Call WrRDWFTail to write this header to the RDWC file.

tsv_lookahead()

int i

This function calculates the total sales value (ps_total_sales_value) by "looking ahead" and summing up the item values and discounts for the current item record (i).

WrRDWFHead()

char *is_file_type

FILE *is_file

long *iol_line_ctr

Set *iol_line_ctr to 1. This is the appropriate global line counter variable for the file type.

Writes an RDW_FHEAD record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
frecdesc	char	RDW_FRECDESC_SIZE	RDW_FHEAD_FRECDESC
flineid	char	LEN_FILE_LINE_NO	*iol_line_ctr
file_type_definition	char	LEN_FILE_TYPE_DEF	is_file_type
file_create_date	char	LEN_DATETIME	ps_sysdate

Call **putrec** to write the record out to the RDWT or RDWF file.

WrRDWTHead()

pt_sa_tran_head *ir_head

Pt_sa_tran_item *ir_item

Increment pl_rdwt_line_ctr.

Set pi_tdetl_count to 0.

This function writes a RDW_THEAD record (as defined in salib.h) to the output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
fredesc	char	RDW_FRECDESC_SIZE	RDW_THEAD_FRECDESC
flineid	char	LEN_FILE_LINE_NO	pl_rdwt_line_ctr
tran_datetime	char	LEN_DATETIME	ir_head->s_tran_datetime

Field	Type	Size	Source
Location	char	LEN_LOC	ps_store
register_id	char	LEN_REGISTER	ir_head->s_register
cashier_id	char	LEN_EMP_ID	ir_head-> s_cashier
Salesperson_id	char	LEN_EMP_ID	ir_item-> s_sales_person if NULL than use ir_head-> s_salesperson
cust_id_type	char	CIDT_SIZE	ir_head-> s_cust_id_type
cust_id_number	char	LEN_CUST_ID	ir_head-> s_cust_id
tran_no	char	LEN_TRAN_NO	ir_head-> s_tran_no
Orig_register	Char	LEN_REGISTER	Ir_head-> s_orig_register
Orig_tran_no	Char	LEN_TRAN_NO	Ir_head-> s_orig_tran_no
tran_seq_no	char	LEN_BIG_SEQ_NO	ir_head-> s_tran_seq_no
rev_no	char	LEN_SA_REV_NO	ir_head-> s_rev_no
tran_sign	char	LEN_IND	ir_head-> s_tran_sign
tran_type	char	TRAT_SIZE	ir_head-> s_tran_type
sub_tran_type	char	TRAS_SIZE	ir_head-> s_tran_sub_type
emp_cashier_no	char	LEN_EMP_ID	ir_head-> s_ref_no1 if sub_tran_type = TRAS_EMP
receipt_ind	char	LEN_IND	ir_head-> s_ref_no1 if tran_type = TRAT_RETURN
reason_code	char	REAC_SIZE	ir_head->s_reason_code
vendor_no	char	LEN_VENDOR_NO	ir_head->s_ref_no1 if tran_type = TRAT_PAIDOU
item_type	char	SAIT_SIZE	SAIT_ITEM if ir_item->s_item_type is either SAIT_ITEM or SAIT_REF. SAIT_GCN if ir_item->s_item_type is SAIT_GCN.
item_no	char	LEN_ITEM_NO	Ir_item->s_item if ir_item->s_item_type is SAIT_ITEM. Ir_item->s_voucher_no if ir_item->s_item_type is SAIT_GCN.
tax_ind	char	LEN_IND	ir_item->s_tax_ind
item_swiped_ind	char	LEN_IND	ir_item->s_item_swiped_ind
Dept	char	LEN_DEPT	ir_item->s _dept
Class	char	LEN_CLASS	ir_item->s _class

Field	Type	Size	Source
Subclass	char	LEN_SUBCLASS	ir_item->s _subclass
total_sales_qty	char	LEN_QTY	ir_item->s_qty
total_sales_value	char	LEN_AMT	ps_total_sales_value if tran_type is not TRAT_COND, TRAT_PADIN or TRAT_PAIDOU. ir_head->value if tran_type is TRAT_PAIDIN or TRAT_PAIDOU.
override_reason	char	ORRC_SIZE	ir_item->s_override_reason
Return_reason_code	Char	SARR_SIZE	Ir_item->s_return_reason_code
total_orig_sign	char	LEN_SALES_SIGN	ir_item->s_qty_sign
total_orig_value	char	LEN_AMT	ir_item->s_qty * ir_item->s_orig_unit_retail / 10000
Weather	char	LEN_CODE	ir_head->s_ref_no1 if tran_type is TRAT_COND
Temperature	char	LEN_CODE	ir_head->s_ref_no2 if tran_type is TRAT_COND
Traffic	char	LEN_CODE	ir_head->s_ref_no3 if tran_type is TRAT_COND
Construction	char	LEN_CODE	ir_head->s_ref_no4 if tran_type is TRAT_COND

Call putrec to write the record out to the RDWT file.

WrRDWTDetl()

pt_sa_tran_head *ir_head

ps_sa_tran_disc *ir_disc

Increment both pl_rdwt_line_ctr and pl_tdetl_count.

Writes an RDW_TDETL record (as defined in salib.h) to the RDWT output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
frecdesc	char	RDW_FRECDESC_SIZE	RDW_TDETL_FRECDESC
flineid	char	LEN_FILE_LINE_NO	pl_rdwt_line_ctr
Discount_type	Char	SADT_SIZE	Ir_disc-> s_discount_type
promo_tran_type	char	PRMT_SIZE	ir_disc-> s_rms_promo_type
promo_no	char	LEN_PROMOTION	ir_disc-> s_disc_ref_no

Field	Type	Size	Source
tran_sign	char	LEN_IND	ir_head-> s_tran_sign
Coupon_no	Char	LEN_COUPON_NO	Ir_disc-> s_coupon_no
Coupon_ref_no	Char	LEN_COUPON_REF_NO	Ir_disc-> s_coupon_ref_no
sales_qty	char	LEN_QTY	ir_disc-> s_qty
sales_sign	char	LEN_SALES_SIGN	ir_disc->s_qty_sign
sales_value	char	LEN_AMT	ps_total_sales_value
disc_value	char	LEN_AMT	ir_disc-> s_unit_disc_amt

Call putrec to write the record out to the RDWT file.

WrRDWTTail()

No arguments

Increment pl_rdwt_line_ctr.

Writes an RDW_TTAIL record (as defined in salib.h) to the RDWT output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
frecdesc	char	RDW_FRECDESC_SIZE	RDW_TTAIL_FRECDESC
flineid	char	LEN_FILE_LINE_NO	pl_rdwt_line_ctr
tran_rec_counter	char	LEN_DTL_LINE_CNT	pi_tdetl_count

Call putrec to write the record out to the RDWT file.

WrRDWFTail()

FILE *is_file

long *iol_line_ctr

Increments *iol_line_ctr. This is the appropriate global line counter variable for the file type.

Writes an RDW_FTAIL record (as defined in salib.h) to the specified output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
frecdesc	char	RDW_FRECDESC_SIZE	RDW_FTAIL_FRECDESC
Flineid	char	LEN_FILE_LINE_NO	*iol_line_ctr
file_rec_counter	char	LEN_DTL_LINE_CNT	*iol_line_ctr – 2

Call putrec to write the record out to the RDWT or RDWF file.

WrRDWSTDetl()

char *is_status

char *is_total_id

char *is_ref_no1

char *is_ref_no2

char *is_ref_no3

char *is_total_value

Increment pl_rdws_line_ctr.

Writes an RDWS_TDETL record (as defined in salib.h) to the RDWS output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
frecdesc	char	RDW_FRECDESC_SIZE	RDW_FDETL_FRECDESC
flineid	char	LEN_FILE_LINE_NO	pl_rdws_line_ctr
tran_date	char	LEN_DATEONLY	ps_business_date
location	char	LEN_LOC	ps_store
sales_sign	char	LEN_SALES_SIGN	is_status
total_id	char	LEN_TOTAL_ID	is_total_id
Ref_no1	char	LEN_REF_NO	Is_ref_no1
Ref_no2	char	LEN_REF_NO	Is_ref_no2
Ref_no3	char	LEN_REF_NO	Is_ref_no3
total_sign	char	LEN_SALES_SIGN	SIGN_N or SIGN_P depending on whether or not is_total_value is negative.

Field	Type	Size	Source
total_amount	char	LEN_AMT	Absolute value of is_total_value.

Call putrec to write the record out to the RDWT file.

WrRDWCTDetl()

char *is_cashier_id

char *is_register_id

char *is_status

char *is_total_id

char *is_ref_no1

char *is_ref_no2

char *is_ref_no3

char *is_total_value

Increment pl_rdwc_line_ctr.

Writes an RDWC_FDETL record (as defined in salib.h) to the RDWC output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
frecdesc	char	RDW_FRECDESC_SIZE	RDW_FDETL_FRECDESC
flineid	char	LEN_FILE_LINE_NO	pl_rdwc_line_ctr
tran_date	char	LEN_DATEONLY	ps_business_date
location	char	LEN_LOC	ps_store
cashier_id	char	LEN_EMP_ID	is_cashier_id
register_id	char	LEN_REGISTER	is_register_id
sales_sign	char	LEN_SALES_SIGN	is_status
total_id	char	LEN_TOTAL_ID	is_total_id
Ref_no1	char	LEN_REF_NO	Is_ref_no1
Ref_no2	char	LEN_REF_NO	Is_ref_no1
Ref_no3	char	LEN_REF_NO	Is_ref_no1
total_sign	char	LEN_SALES_SIGN	SIGN_N or SIGN_P depending on whether or not is_total_value is negative.

Field	Type	Size	Source
total_amount	char	LEN_AMT	Absolute value of is_total_value.

Call putrec to write the record out to the RDWC file.

WrRDWFDetl()

pt_sa_tran_head *ir_head

pt_sa_tran_tender *ir_tend

Increment pl_rdwf_line_ctr.

Writes an RDWF_FDETL record (as defined in salib.h) to the RDWF output file. This must match the definition of the record in Interface File – SA to RDW.doc.

Field	Type	Size	Source
frecdesc	char	RDW_FRECDESC_SIZE	RDW_FDETL_FRECDESC
flineid	char	LEN_FILE_LINE_NO	pl_rdwf_line_ctr
business_date	char	LEN_DATEONLY	ps_business_date
tran_datetime	char	LEN_DATETIME	ir_head->s_tran_datetime
location	char	LEN_LOC	ps_store
casher_id	char	LEN_EMP_ID	ir_head->s_cashier
register_id	char	LEN_REGISTER	ir_head->s_register
tran_sign	char	LEN_SALES_SIGN	ir_head ->s_tran_sign
tran_seq_no	char	LEN_BIG_SEQ_NO	ir_head->s_tran_seq_no
rev_no	char	LEN_SA_REV_NO	ir_head ->s_rev_no
tran_type	char	TRAT_SIZE	ir_head ->s_tran_type
tender_type_group	char	TENT_SIZE	ir_tend->s_tender_type_group
tender_type_id	char	TENS_SIZE	ir_tend->s_tender_type_id
tender_amt	char	LEN_AMT	ir_tend->s_tender_amt
cc_no	char	LEN_CC_NO	ir_tend->s_cc_no
cc_exp_date	char	LEN_DATEONLY	ir_tend->s_cc_exp_date
cc_auth_no	char	LEN_CC_AUTH_NO	ir_tend->cc_auth_no
cc_auth_src	char	CCAS_SIZE	ir_tend->s_cc_auth_src
cc_entry_mode	char	CCEM_SIZE	ir_tend->s_cc_entry_mode
cc_cardholder_verf	char	CCVF_SIZE	ir_tend->s_cc_cardholder_verf
cc_terminal_id	char	LEN_TERM_ID	ir_tend->s_cc_terminal_id
cc_special_cond	char	CCSC_SIZE	ir_tend->s_cc_special_cond

Field	Type	Size	Source
voucher_no	char	LEN_VOUCHER_NO	ir_tend->s_voucher_no
Voucher_age	Char	LEN_VOUCHER_AGE	Ir_tend->s_voucher_age
Escheat_date	Char	LEN_DATEONLY	Ir_tend->s_escheat_date
Coupon_no	Char	LEN_COUPON_NO	ir_tend->s_coupon_no
Coupon_ref_no	char	LEN_COUPON_REF_NO	ir_tend->s_coupon_ref_no

Call **putrec** to write the record out to the RDWF file.

Stored Procedures / Shared Modules (Maintainability)

Shared Module	Module Description
libretek.a functions	Refer to Library Design – retek.doc for details.
Retek_init	Initialize restart recovery.
Retek_close	Close restart recovery functions.
Retek_refresh_thread	Refresh the current thread so that it may be used again.
Libresa.a functions:	Refer to Library Design – ReSA.doc for details.
get_lock	used to establish a read lock on a store/day.
release_lock	used to release a store/day lock.
fetchSaSystemOptions	Fetch the values from the sa_system_options table.
fetchSysDate	Fetch the current SYSDATE value.
fetchStoreDayErrorCount	Fetch the number of errors that corresponds to a particular store/day and system.
markStoreDayExported	Mark a particular store/day and system as exported
markTransactionExported	Mark a particular transaction and system as exported.
OraNum functions (Add, Sub, Mul, Div)	Used to perform arithmetic operations on strings containing large numbers.
getBalTotal	Get the specified balance totals.
putrec	Writes a record to a file.

Output Specifications

Output Files

Data is output in the RDW file format. This is described in the file Interface File – SA to RDW.doc.

The filename convention for these valid RDWT, SIF Tender, RDWS and RDWC files will be rdwt_store_businessdate_curdatetime, rdwf_store_businessdate_curdatetime, rdws_store_businessdate_curdatetime and rdwt_store_businessdate_curdatetime. The files should start out with a temporary name generated by the Unix tempnam (3S) call and then be renamed with Unix rename (2) call when the files are complete.

Scheduling Considerations

Processing Cycle: Anytime – Sales Audit 3.0 is a 24/7 system.

Scheduling Diagram: This program will be run after auditors have made corrections to the data.

Pre-Processing: sagetref.pc to get waste data, and saimptlog.pc and saimptlogfin.pc to get post-void data.

Post-Processing:

- resa2rdw should be run on all output files created by saexprdw.pc. This will reformat the files for RIB-ETL loads by RDW.

Threading Scheme: saexprdw can be threaded for up to 6 concurrent threads. The threading scheme is based on the cursor c_store_day in the process() function. Since the thread values are used within the ORDER BY clause, the maximum number of concurrent threads equals the number of columns in this cursor.

Locking Strategy

In conjunction with the Performance and the Scheduling Considerations section, this section should describe the locking (and release) strategy required beyond the preset Retek standards. It should describe how the module accesses data and the ‘hold’ or ‘lock’ it has on a database and / or its records, during processing. It should also describe the ‘lock’ release.

Restart / Recovery

The logical unit of work for this module is defined as a unique store/day combination. Records will be fetched, updated and inserted in batches of pl_commit_max_ctr. Only two commits will be done, one to establish the store/day lock and another at the end, to release the lock after a store/day has been completely processed. The RDWT, RDWF, RDWS and RDWC formatted output files will be created with temporary names and renamed just before the end of store/day commit.

In case of failure, we rollback all work done to the point right after the call to get_lock and then we release the lock. Thus, we assume that the rollback segment is large enough to hold all inserts into sa_exported for one store_day. If this is not the case, we need to increase the size of the rollback segment. The EXEC SQL SAVEPOINT statement is used to save the state of the database after the call to get_lock.

There are 3 driving cursors in this module. The first picks a store/day to work on:

```
c_store_day CURSOR FOR
SELECT   /*+ rule */
         sd.store_day_seq_no,
         el.seq_no,
         sd.store,
         TO_CHAR(sd.business_date, 'YYYYMMDD'),
         ROWIDTOCHAR(el.rowid)
FROM     sa_store_day sd, sa_export_log el
WHERE    sd.store_day_seq_no = el.store_day_seq_no
AND      sd.store_status = :SASS_C    /* Closed
*/
AND      sd.data_status  = :SADS_F    /* Fully loaded
*/
AND      sd.audit_status = :SAAS_A    /* Audited, but no
Errors */
AND      el.system_code  = :SYSE_RDW
AND      el.status       = :SAES_R    /* 'R'eady to be
exported */
ORDER BY MOD(TRUNC(sd.store_day_seq_no / :pi_num_threads)
             + :pi_thread_val, :pi_num_threads),
             sd.store, sd.business_date;
```

Since RDW cannot accept data from a store_day with errors pending, we select store_days that have audit_status ‘A’ only. The library function fetchStoreDayToBeExported cannot be used here because it fetches store_days with an audit_status of ‘E’ (Errors pending).

The second driving cursor fetches the store/day transaction data to be output:

```
      SELECT h.tran_seq_no,
             h.rev_no,
             TO_CHAR( h.tran_datetime,
'YYYYMMDDHH24MISS'),
             NVL( h.register, ' '),
             NVL( TO_CHAR( h.tran_no), ' '),
             NVL( em.emp_id, ' '),
             NVL( em2.emp_id, ' '),
             NVL( c.cust_id_type, ' '),
             NVL( c.cust_id, ' '),
             NVL( h.reason_code, ' '),
             h.tran_type,
             NVL( h.sub_tran_type, ' '),
             NVL( TO_CHAR( h.orig_tran_no), ' '),
             NVL( h.orig_reg_no, ' '),
             NVL( h.ref_no1, ' '),
             NVL( h.ref_no2, ' '),
             NVL( h.ref_no3, ' '),
             NVL( h.ref_no4, ' '),
             NVL( h.vendor_no, ' '),
             h.status,
             DECODE( SIGN( h.value), -1, :SIGN_N,
:SIGN_P),
             NVL( TO_CHAR( ABS(h.value) *
:pl_multiplier), '0'),
             :SAFD_P,
             ' '
        FROM sa_tran_head h,
             sa_customer c,
             sa_store_emp em,
             sa_store_emp em2,
          /* This temporary view selects all cashiers for
the given store */
          (SELECT DISTINCT th.cashier,
                  sd.store
             FROM sa_tran_head th,
                  sa_store_day sd
            WHERE sd.store_day_seq_no =
th.store_day_seq_no
              AND sd.store_day_seq_no =
TO_NUMBER(:is_store_day_seq_no)) temp_view1,
          /* This temporary view selects all salespersons
for the given store */
```

```
          (SELECT DISTINCT th.salesperson,
                  sd.store
             FROM sa_tran_head th,
                  sa_store_day sd
            WHERE sd.store_day_seq_no =
th.store_day_seq_no
              AND sd.store_day_seq_no =
TO_NUMBER(:is_store_day_seq_no)) temp_view2
       WHERE h.store_day_seq_no =
TO_NUMBER(:is_store_day_seq_no)
         AND em.pos_id(+) = temp_view1.cashier
         AND em.store(+)  = temp_view1.store
         AND (   temp_view1.cashier = h.cashier
              OR (    temp_view1.cashier IS NULL
                  AND h.cashier IS NULL))
         AND em2.pos_id(+) = temp_view2.salesperson
         AND em2.store(+)  = temp_view2.store
         AND (   temp_view2.salesperson = h.salesperson
              OR (    temp_view2.salesperson IS NULL
                  AND h.salesperson IS NULL))
         AND h.tran_seq_no = c.tran_seq_no(+)
         AND h.tran_type IN (:TRAT_SALE,   :TRAT_RETURN,
:TRAT_EEXCH,
                             :TRAT_PAIDIN, :TRAT_PAIDOU,
:TRAT_NOSALE,
                             :TRAT_VOID,   :TRAT_PVOID,
:TRAT_COND)
         AND (h.status = :SAST_P
              AND NOT EXISTS                  /* and no
errors for the transaction. */
                 (SELECT er.tran_seq_no
                    FROM sa_error er, sa_error_impact ei
                   WHERE h.tran_seq_no = er.tran_seq_no
                     AND er.error_code = ei.error_code
                     AND ei.system_code = :SYSE_RDW
                     AND er.hq_override_ind != :YSNO_Y))
         AND NOT EXISTS
            (SELECT e.store_day_seq_no
               FROM sa_exported e
              WHERE h.store_day_seq_no =
e.store_day_seq_no
                AND h.tran_seq_no = e.tran_seq_no
                AND e.system_code = :SYSE_RDW)
```

```
   UNION ALL
      SELECT h.tran_seq_no,
             h.rev_no,
             TO_CHAR( h.tran_datetime,
'YYYYMMDDHH24MISS'),
             NVL( h.register, ' '),
             NVL( TO_CHAR( h.tran_no), ' '),
             NVL( em.emp_id, ' '),
             NVL( em2.emp_id, ' '),
             NVL( c.cust_id_type, ' '),
             NVL( c.cust_id, ' '),
             NVL( h.reason_code, ' '),
             h.tran_type,
             NVL( h.sub_tran_type, ' '),
             NVL( TO_CHAR( h.orig_tran_no), ' '),
             NVL( h.orig_reg_no, ' '),
             NVL( h.ref_no1, ' '),
             NVL( h.ref_no2, ' '),
             NVL( h.ref_no3, ' '),
             NVL( h.ref_no4, ' '),
             NVL( h.vendor_no, ' '),
             h.status,
             DECODE( SIGN( h.value), -1, :SIGN_N,
:SIGN_P),
             NVL( TO_CHAR( ABS(h.value) *
:pl_multiplier), '0'),
             :SAFD_N,
             NVL( TO_CHAR( e.exp_datetime,
'YYYYMMDDHH24MISS'), ' ')
        FROM sa_tran_head h,
             sa_exported e,
             sa_customer c,
             sa_store_emp em,
             sa_store_emp em2,
          /* This temporary view selects all cashiers for
the given store */
          (SELECT DISTINCT th.cashier,
                  sd.store
             FROM sa_tran_head th,
                  sa_store_day sd
            WHERE sd.store_day_seq_no =
th.store_day_seq_no
```

```
              AND sd.store_day_seq_no =
TO_NUMBER(:is_store_day_seq_no)) temp_view1,
          /* This temporary view selects all salespersons
for the given store */
          (SELECT DISTINCT th.salesperson,
                  sd.store
             FROM sa_tran_head th,
                  sa_store_day sd
            WHERE sd.store_day_seq_no =
th.store_day_seq_no
              AND sd.store_day_seq_no =
TO_NUMBER(:is_store_day_seq_no)) temp_view2
       WHERE h.store_day_seq_no =
TO_NUMBER(:is_store_day_seq_no)
         AND em.pos_id(+) = temp_view1.cashier
         AND em.store(+)  = temp_view1.store
         AND (   temp_view1.cashier = h.cashier
              OR (    temp_view1.cashier IS NULL
                  AND h.cashier IS NULL))
         AND em2.pos_id(+) = temp_view2.salesperson
         AND em2.store(+)  = temp_view2.store
         AND (   temp_view2.salesperson = h.salesperson
              OR (    temp_view2.salesperson IS NULL
                  AND h.salesperson IS NULL))
         AND h.tran_seq_no = c.tran_seq_no(+)
         AND h.tran_type IN (:TRAT_SALE,   :TRAT_RETURN,
:TRAT_EEXCH,
                             :TRAT_PAIDIN, :TRAT_PAIDOU,
:TRAT_NOSALE,
                             :TRAT_VOID,   :TRAT_PVOID,
:TRAT_COND)
         AND h.status in (:SAST_V, :SAST_D)
         AND h.tran_seq_no = e.tran_seq_no(+)
         AND e.status = :SAST_P
         AND e.system_code = :SYSE_RDW
    ORDER BY 3;
```

The third driving cursor is encapsulated in the getBalTotal function, which fetches all totals with a usage_type of ‘RDW’. It returns, among other things, the total_id, the cashier id and the register id. These are then used to determine whether to write a record to the RDWS file or the RDWC file. Only totals with a total_id of “OVRSHT_B” (over/short balance level) are exported to the RDWC file. The other totals are exported to the RDWS file only if both their register and their cashier ids are empty, i.e. the total is at the store level. If the total cannot be written to neither the RDWC nor the RDWS file, then we write an error to the log and continue.

Performance

In conjunction with the Scheduling Considerations and Locking Strategy sections, the optimization considerations of a batch module must adhere to Retek standards. This section should call out special performance considerations that may exceed current documented Retek practices. Such considerations should be the basis for update to Retek standards. Each database operation should be optimized based on quantity and quality of the database transactions. Batch modules are executed on the database or dedicated batch server and thus there are no additional performance gains to forcing database interaction logic onto the server.

Security Considerations

Credit card numbers and other customer information are present in the output files. Access to these files is controlled only by the Unix permissions that these files have.