



## Common Performance Strategies

- **Maximize parallel processing**
  - Great strategy for both custom and standard programs
  - “Divide and Conquer”
- **Save complex data to custom tables**
  - Some of the data stored in SAP is not easy to retrieve without complex calculations. This includes both payroll and payroll posting data.
  - Save this data to targeted custom tables for fast retrieval with the primary key or index of the table
  - Custom reports can read the table and output to ALV with no additional calculations. Users will love the performance!

3

## Common Performance Strategies (cont.)

- **Minimize redundant processing**
  - Why redo your work?
  - Read once, write many?
- **Optimize programs with effective coding techniques**
  - Use the database effectively
  - Use internal tables effectively
  - Use authorization checking effectively

4

## What We'll Cover ...

- **Common Performance Strategies**
- **Overview of Performance Problem Areas**
- **Performance Enhancements for Custom Programs**
- **Performance Enhancements for Payroll Schema**
- **Enhancing Standard Payroll Programs for Parallel Processing**
- **Wrap-up**

5

## Overview of Performance Problem Areas

- **Standard SAP Payroll programs**
  - Payroll Driver (RPCALCU0)
  - Pre-program DME (RPCDTU0)
  - Third-Party Remittance Evaluation (RPCALCU0)
  - Payroll Posting (RPCIPE00)
- **Custom programs**
  - Reports
  - Interfaces

6

## Overview of Performance Problem Areas (cont.)

- **Standard SAP Payroll programs**
  - Minimum ability to change how program runs
    - ▶ We must work with the program as delivered
    - ▶ We can make small modifications to standard to help us with our performance techniques, but we cannot interfere with the “functional” code within the program
- **Custom programs**
  - Maximum ability to improve performance
    - ▶ We can take a “maximum performance” strategy and write the program accordingly

7

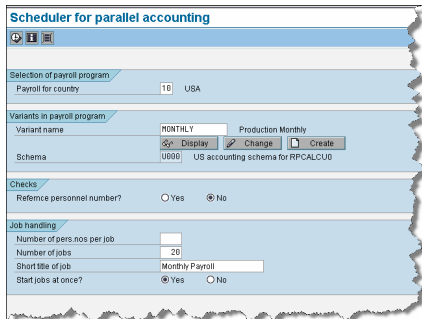
## Payroll Driver

- **There are several delivered techniques for improving the performance of the payroll driver**
  - Parallel Processing with Payroll Scheduler (RPCSC000)
    - ▶ Takes advantage of parallel processing
  - Matchcode W
    - ▶ Minimizes performing the same calculation twice
  - Authorization Object P\_ABAP
    - ▶ Avoids unnecessary authorization checks
  - Schema
    - ▶ We have a reasonable amount of control over this

8

## Parallel Processing with Payroll Scheduler (RPCSC000)

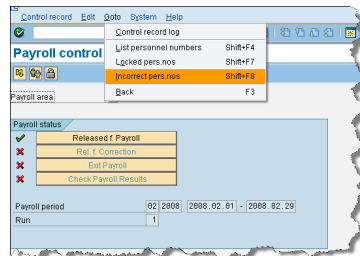
- Standard program to submit payroll run into many parallel processes by personnel number
  - Create variant for RPCALCU0
  - Enter Variant, Schema
  - Request either number of employees per job or number of jobs



9

## Matchcode W

- Eliminates need for multiple full payroll runs during the payroll cycle
  - Used after "Release for Payroll"
  - Contains employee numbers that have changed since previous run of current payroll
  - Contains employee numbers that had errors in current payroll
  - View via transaction PA03



10

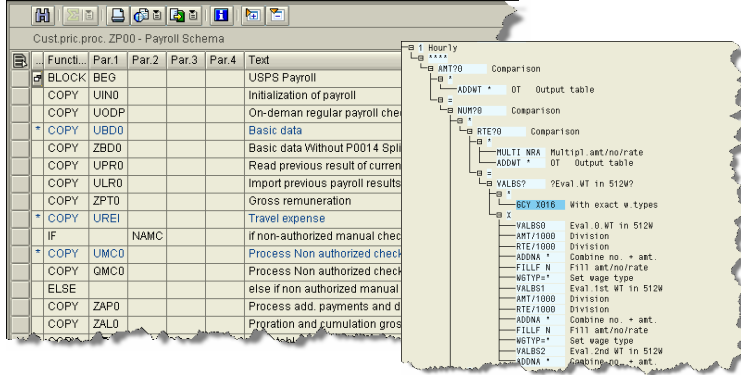
## Authorization Object P\_ABAP

- PNP logical database programs will check this object first and if COARS = 2 for the program that is being run, it will turn off all subsequent authorization checks
- Set COARS = 2 and REPID = RPCALCU0 for user ID that will be used to run payroll

11

## Schema

- Effective use of Personnel Calculation Rules will reduce the processing time of the payroll driver (RPCALCU0)



12

## What We'll Cover ...

- Common Performance Strategies
- Overview of Performance Problem Areas
- Performance Enhancements for Custom Programs
- Performance Enhancements for Payroll Schema
- Enhancing Standard Payroll Programs for Parallel Processing
- Wrap-up

13

## Performance Enhancements for Custom Programs

- Logical database or all custom programs
- Interfaces versus reports
- Reading Payroll Clusters
- Effective use of ABAP

14

## Logical Database or Custom

- Logical database PNP with GET PAYROLL is effective for small to medium employee populations, but inefficient for reports or interfaces that must retrieve payroll results for a large number of employees (more than 20,000)
- When current reports are taking too long to run and users are getting frustrated, they may need to be re-written with performance as a main goal
- Reports can still “appear” to the users as logical database reports by using the selection screen of the standard logical database

15

## Logical Database or Custom (cont.)

- Hybrid reports that use some functionality of the logical database but not all may be sufficient and the best compromise where performance is an issue
  - Example 1 – create a report that uses the logical database for the selection screen, but then use that information to retrieve desired infotype data via direct selects
    - ▶ Be careful with authorization when using this technique!
  - Example 2 – create a report that uses both selection screen and several key infotypes before using custom selects to get additional data
    - ▶ Also be aware of authorization implications!

16

## Interfaces vs. Reports

- Reports
  - Require a common look and feel
    - ▶ PNP logical database provides standard interface
    - ▶ PNP logical database may not provide adequate performance
  - Require authorization checks
    - ▶ PNP logical database provides authorization checks that need to be replicated in high performance reports
  - Require fast response for online ALV output
    - ▶ More than 1 minute response will frustrate users
    - ▶ Excessive runtime will cause timeouts and force users to run in background resulting in loss of online ALV

17

## Interfaces vs. Reports (cont.)

### • Reports

- Write custom code using high performance strategies
- Tailor authorization checks to the requirements and use of the report
- “Pre-process” the report in batch and store the data in custom tables. The report can then retrieve the data with a single select statement and display in ALV format. This strategy is especially useful for reporting payroll posting data.

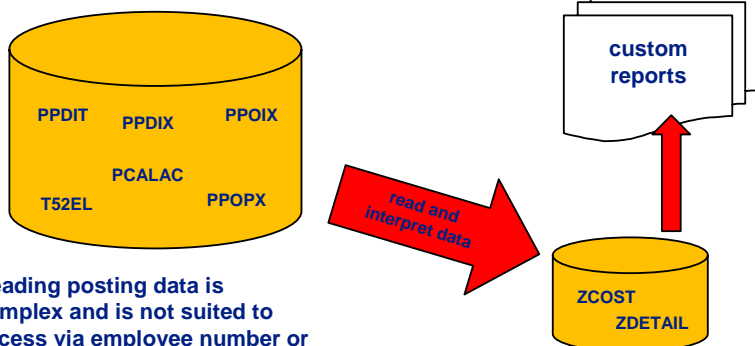
18

## Reports – Case Study 1 – Pre-Process and Store

- An SAP customer with over 65,000 employees had a requirement for an online ALV report that displayed the posting results for up to 10,000 employees in a single run
- A program was written to interpret the posting data and store in a custom table. The table was indexed appropriately to efficiently retrieve data based on the selection options of the report.
- The report read the custom tables to quickly retrieve and display in ALV format
- Custom authorization checks were coded in the report to prevent users from accessing the data of other business areas

19

## Reports – Case Study 1 – Pre-Process and Store (cont.)



20

## Reports – Case Study 1 – Pre-Process and Store (cont.)

- Sample custom authorization check for a report that only needs to restrict a users access by structural authorization

```
FORM authorisation_check .
IF super_user NE 'X' AND p_ess NE 'X'.
  LOOP AT itab_pernr ASSIGNING <pernr>
    CALL FUNCTION 'RH_STRU_AUTHORITY_CHECK'
      EXPORTING
        p_lvar = '01'
        o_type = '0'
        obj_id = <pernr>-orgh
      EXCEPTIONS
        OTHERS = 04.
    IF sy-subrc NE 0.
      WRITE: / 'No Authorization for:',
              <pernr>-pernr,
              <pernr>-sname,
              <pernr>-orgh.
      DELETE itab_pernr.
    ENDIF.
  ENDOLOOP.
ENDIF.
ENDFORM.                    * authorization_check
```

User is already authorized to run report based on transactional authorization

In this example, a table of employees is checked based on the employees' organizational unit.

If the user running the report is not allowed to view an employee based on the org unit they are in, the employee is deleted from the table and an error is logged.

21

## Interfaces vs. Reports

- Interfaces
  - A requirements-driven selection screen is sufficient
  - Authorization checks are unnecessary
    - ▶ Program is restricted to batch user ID
  - Use parallel processing for large employee populations
  - Design interfaces to minimize multiple runs of the same program retrieving the same data to create multiple files
    - ▶ Use a custom table to configure file output
    - ▶ Read data once and output many files
  - Simplify subsequent interfaces by saving results of key data (payroll or payroll posting) in a custom table that can be read by other interfaces

22

## Interfaces – Case Study 2 – Read Once, Output Many

- An SAP customer with 65,000 employees in a single payroll area had an interface that created 30 different output files for 30 different recipients
- A single interface report was run 30 times per pay period with 30 different variants. The total runtime was 35,000 seconds when all jobs were taken into account.
- The program was written with a custom table that controlled the specific requirements of each output file. The table was configured in the customizing client and transported through the landscape.
- The program would read the required data once and use the configuration table to filter the data for each required file

23

## Interfaces – Case Study 2 – Read Once, Output Many (cont.)

- The result was that the new interface ran in 1,000 seconds
  - Additional performance techniques were added in addition to the “run once, output many” technique (see Case Study 3)
- The number of variants was reduced from 30 to 1
- The number of scheduled jobs was reduced from 30 to 1
- New output file requirements need to be configured, tested, and moved through the landscape resulting in better change management for this interface

24

## Interfaces – Case Study 2 – Read Once, Output Many (cont.)

Interface specific selection parameters for payroll period

Run Key is used to identify a configured set of selection criteria for each file. The 7 character run key is also used in the file name to identify the interface variation

A PC or Server directory is added and all files are placed in that directory

Program appends a standard name to front of file to identify interface

25

## Interfaces – Case Study 2 – Read Once, Output Many (cont.)

Field	Key	Initi...	Data element	Data Ty...	Length	Decl...	Short Description
MANDI	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	MANDI	CLNT	3	0	Client
KEY1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZZ634_RUNKEY	CHAR	7	0	Unique Run Key
FIELD	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	ZZ634_FIELD	CHAR	6	0	Selection Field
SEQNO	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	SEQNO	CHAR	3	0	Sequence Number
SIGN	<input type="checkbox"/>	<input type="checkbox"/>	ZZ634_SIGN	CHAR	1	0	Sign of Select Option
SOPTION	<input type="checkbox"/>	<input type="checkbox"/>	ZZ634_OPTION	CHAR	2	0	Selection Option
LOW	<input type="checkbox"/>	<input type="checkbox"/>	ZZ634_LOW	CHAR	10	0	Select-Option Low
HIGH	<input type="checkbox"/>	<input type="checkbox"/>	ZZ634_HIGH	CHAR	10	0	Select Option High

MANDT	KEY1	FIELD	SEQNO	SIGN	SOPTION	LOW	HIGH
201	FSE_DED	VENDOR		I	EQ	0002170900	
201	GCU_DED	VENDOR		I	EQ	0002179200	
201	GEN_DED	VENDOR		I	EQ	0000366032	
201	GEN_DED	VENDOR	01	I	EQ	0000366033	
201	HCA_INS	LGART	01	I	BT	2525	2532
201	HCA_INS	LGART	02	I	BT	2500	2510
201	HCA_INS	LGART	03	I	EQ	2143	
201	HCA_INS	LGART	04	I	EQ	2124	
201	HCA_INS	LGART	05	I	BT	2983	2984
201	HCA_INS	LGART	06	I	BT	2538	2541
201	HCA_INS	LGART	07	I	EQ	2514	

Run configuration table contains selection criteria for each output file. The file is in the same format as a select-option table.

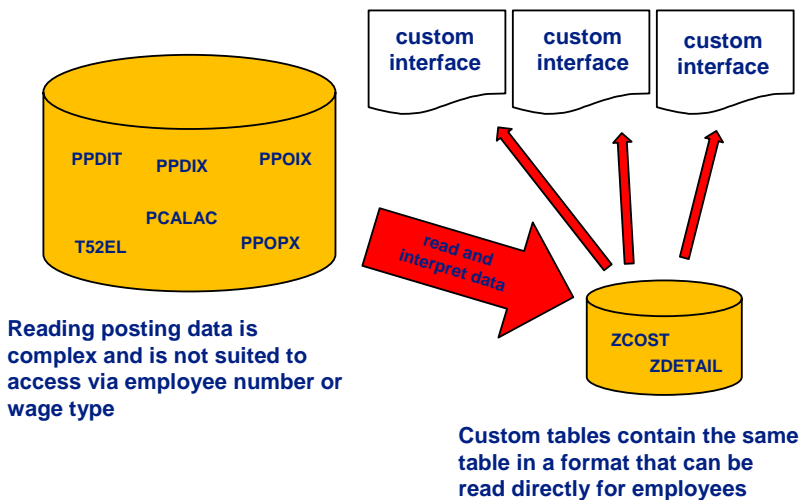
26

## Interfaces – Case Study 3 – Store Results

- An SAP customer with 65,000 employees in a single payroll area had an interface that read payroll posting data and performed additional calculations on this data before sending it to a legacy accounting system
- The data was saved in custom tables after each payroll posting run
- Five subsequent interfaces read this data to produce output for other systems
- Performance of the subsequent interfaces improved by a factor of 10 as they were no longer required to repeat calculations performed by the main payroll posting interface

27

## Interfaces – Case Study 3 – Store Results (cont.)



28

## Interfaces – Case Study 4 – Parallel Process

- The SAP customer described in the previous sections had a requirement for a custom earnings statement report
  - This required reading payroll results and analyzing retro, determining earnings by position in addition to many other requirements
- A custom program was written using all available performance strategies
  - Despite this, the program still ran for several hours
- A “submit” program was written to divide the processing into 20 discrete jobs. This allowed the program to run in an acceptable 30 minutes.
  - The data was stored and retrieved by a subsequent program

29

## Interfaces – Case Study 4 – Parallel Process (cont.)

Payroll Period	
Payroll Area	11
Pay Period	02 2008
Personnel Number	<input type="checkbox"/> Use Current Period to <input type="text"/>
Run Type	
Type	P
Description	Production Payroll
Other Parameters	
Number of Jobs	20
Job Name	Earnings Statement
Variant	Monthly

Interface specific selection screen for payroll period

Run Type is used to determine how data will be stored

Selection Parameter for number of jobs and the job name. Custom splitter program will append a sequence number to each job name when creating job.

30

## Reading Payroll Clusters

- **Developers have many options for reading cluster data**
  - PNP with screen 900 (GET PAYROLL)
    - ▶ Recommended for smaller employee populations
    - ▶ Not high performance
  - Standard SAP functions
    - ▶ Easy to use
  - Macros
    - ▶ Traditional method
  - Methods
  - Import statement
    - ▶ The "lowest level" of ABAP code for reading cluster tables

31

## Reading Payroll Clusters (cont.)

- **Import Statement**
  - The preferable method for reading payroll cluster data when performance is a requirement
  - Retrieve only the required tables from the required payroll result
  - Only use if the results are not going to be returned to the cluster
- **Export Statement**
  - NEVER use the export statement to return payroll results to the cluster. This may result in lost data.



32

## Reading Payroll Clusters (cont.)

```
REPORT z_ph11_taylor_300
TABLES: hrpy_rgdtr.

DATA: BEGIN OF rx-key.
  INCLUDE STRUCTURE pc200.
DATA: END   OF rx-key.

DATA: rt TYPE TABLE OF pc207.
      wpbp TYPE TABLE OF pc205.

field-symbols: <rt> TYPE pc207.
               <wpbp> TYPE pc205.

* get sequence number from HRPY_RGDIR

rx-key-pernr = hrpy_rgdtr-pernr.
rx-key-seqno = hrpy_rgdtr-seqnr.

IMPORT
  rt
  wpbp
FROM DATABASE pc12(ru) ID rx-key.
IF sy-subrc = 0.

loop at rt assigning <rt>.
  * read wage type data from payroll cluster here
endloop.

ENDIF.
```

Only declare the structures that you need. Use tables without header lines and field symbols to read the tables.

A convenient location for the payroll sequence number is the HRPY\_RGDIR table which can be read with a select statement

Import only the data objects that you need from the appropriate payroll cluster

33

## Effective Use of ABAP Statements

- Performance must always be on the mind when developing custom reports and interfaces
- Minimize use of the database server and maximize the use of the application servers
  - There is usually only one database server and this becomes the weakest link in the performance chain
  - Most SAP implementations have many application servers
- The key problem areas are:
  - Retrieving data from the database
  - The use of internal tables

34

## Effective Use of ABAP Statements – Data Retrieval

- Minimize the number of calls to the database
  - Look for opportunities to load data into lookup tables that will be used later in the program. This would include employee names, text of org units, positions, business areas, etc.
  - Never read the same data twice – what a waste of resources!
  - Design your database reads around the use of the report, not just on its functionality
    - ▶ For example, if the report is used to display the records of a single employee, you would probably not read in all the names of every employee in the system into an internal table

35

## Effective Use of ABAP Statements – Data Retrieval (cont.)

- Read the data effectively
  - Always think of what the database is doing when it reads your data
    - ▶ If you are reading data from a large table without supplying all the fields in the primary index (the key fields of the table), then the database could be reading many more records than required
  - Look for opportunities to add an additional index to your table
    - ▶ A posting table may save data by document number, but you need it by Personnel ID (PERNR). This may be an index opportunity.

36

## Effective Use of ABAP Statements – Internal Tables

- Always determine the optimum way of reading an internal table. If you are using a table to look up position names, then they should be stored in a hashed or sorted table.
  - Tip! Hashed tables are faster for reading many records!
- If you use “Loop at where,” then make sure your table is a sorted table with the key fields the same as those in the “Where” clause
- Use field symbols when reading tables
- Use work areas when building tables
- Take advantage of the “collect” statement
  - There is an internal hash algorithm used when this is processing!

37

## Effective Use of ABAP Statements – Internal Tables (cont.)

```
DATA: itab_0000 TYPE SORTED TABLE OF zhr_g009_0000 WITH NON-UNIQUE KEY pernr.  
FIELD-SYMBOLS: <0000> TYPE zhr_g009_0000.  
LOOP AT itab_0000 ASSIGNING <0000>  
  WHERE pernr = <0001>-pernr.  
ENDLOOP.
```

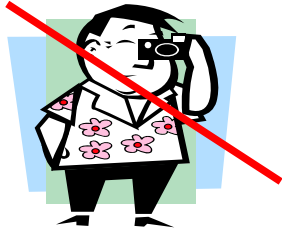
An internal table is declared as a sorted table using a custom dictionary structure. The field PERNR is the first field in the table.

The loop at where clause will now use a binary search algorithm to loop at only the rows of the table where PERNR = <0001>-pernr. If we did not take this approach a full table scan would be performed.

38

## Effective Use of ABAP Statements

- **Don't be an ABAP "Tourist"**
  - Many developers speak "tourist" ABAP. They can order a meal, find room and lodging, but they can't really have an in-depth conversation with the locals. This will result in programs that "work" but not very effectively.
  - Buy the two-volume ABAP reference manual and learn it all!



39

## What We'll Cover ...

- Common Performance Strategies
- Overview of Performance Problem Areas
- Performance Enhancements for Custom Programs
- Performance Enhancements for Payroll Schema
- Enhancing Standard Payroll Programs for Parallel Processing
- Wrap-up

40

## Performance Enhancements for Payroll Schema

- **There are three main areas where you can make the payroll schema more efficient:**
  - Avoid unnecessary rules and wage types
  - Make PCRs (Personnel Calculation Rules) as specific as possible
  - The location within the schema of store points that move wage types from the IT table to the RT table

41

## Avoid Unnecessary Rules and Wage Types

- There is often a temptation to put “reporting” wage types into the schema to make the development of custom reports easier. These may be wage types that track arrears or track retroactivity.
- This type of wage type should be avoided for the following reasons:
  - The running of the payroll driver is usually on a critical path of the payroll cycle so adding additional wage types will slow down the processing of payroll
  - If the desired result can be achieved with a calculation in a report, then that is the preferred technique

42

## Make PCRs as Specific as Possible

- Whenever a PCR is defined as generic it must cycle through all wage types in the internal table
- By making PCRs as specific as possible, you will minimize the number of passes of the internal tables used in payroll (IT, OT, etc.)
  - These passes add up and cause performance degradation

43

## Storing Wage Types to RT Table

- Find suitable locations to process given wage types and store from IT to RT
- In many cases, a group of wage types can be processed and stored early in the schema
  - This removes them from the IT table and reduces overall processing of this table

44

## What We'll Cover ...

- Common Performance Strategies
- Overview of Performance Problem Areas
- Performance Enhancements for Custom Programs
- Performance Enhancements for Payroll Schema
- **Enhancing Standard Payroll Programs for Parallel Processing**
- Wrap-up

45

## Enhancing SAP Standard Payroll Programs

- **Parallel Processing**
  - The SAP payroll driver is designed to be run as a set of parallel jobs. SAP provides a scheduling program to do this.
  - The pre-program DME, Third-Party Remittance Evaluation, and the Payroll Posting report can also benefit from parallel processing. These programs need some help to enable this type of processing.
- **Wrapper Programs**
  - When enhancing the Third-Party Remittance Evaluation program, we will not have to make any changes to the standard. We do need to write some "wrapper" programs to control the process.

46

## Registering Changes to SAP Standard Programs

- **Enhancing standard SAP programs may be perceived as a dangerous undertaking. Changing the functionality would require significant testing and introduce risk.**
- **The proposed techniques do not change the functionality of the standard**
  - You just need to add some code to the standard programs to provide additional information you would not normally have access to
- **Never make copies of standard reports for these small changes. Use the modification assistant and register the changes.**
  - This will prevent the change from being overlooked during an upgrade or support pack

47

## Pre-Program DME – Create a Parallel Process

- The pre-program DME program is designed to be run as a single job. This job will run for hours if the number of employees in the payroll run exceeds 10,000.
  - A current SAP customer had a pre-program DME run that contained 65,000 employees. The program ran for 11 hours, which greatly impacted the payroll cycle.
- The pre-program DME reads and modifies payroll results. It serves as a precursor program to the payment medium workbench in addition to the print check and DME program.
- If the pre-program DME is split into many jobs, it will appear as many payment runs and this is not desirable

48

## Pre-Program DME – Create a Parallel Process (cont.)

- The program will run quickly when divided into smaller jobs, but the customer needed it to appear as one job when all was said and done
- The following strategy was taken:
  - Create a “wrapper” program to submit the pre-program DME
  - Add code to the standard pre-program DME to allow for the tracking of the run
  - Change the run identifiers in the payroll cluster BT table and several other tables to make the run look like a single pre-program DME run. This was done with a custom program.

49

## Pre-Program DME – Create a Parallel Process (cont.)

- **Submit Program**
  - A custom submit program was written to pass parameters to the standard pre-program DME program and submit in background
  - This type of program is very useful and once written can be used as a template for custom reports and interfaces that need to be run in parallel
  - The foundation of the submit program is the following:
    - ▶ Call Function Job\_Open ...
    - ▶ Submit RPCDTCU0 with ...
    - ▶ Call Function Job\_Close ...

50

## Pre-Program DME – Create a Parallel Process (cont.)

### • Submit Program

- A simple programming technique is used to split a table of employee numbers into equal groups, which are then passed to the program and submitted in background

### • Control Table

- A custom table was created to track the date and time stamp used by the pre-program DME as a key to the run
- A job sequence number is passed to the program during the submit process and a modification to the pre-program DME writes the time stamp to the custom table with the job sequence number

51

## Pre-Program DME – Create a Parallel Process (cont.)

Data Browser: Table ZHR\_PRE\_DME Select Entries

MANDT	JOBNAME	SEQNO	LAUFD	LAUFI
700	PRE_DME_20061103	1	11/03/2006	07411P
700	PRE_DME_20061103	2	11/03/2006	07413P
700	PRE_DME_20061103	3	11/03/2006	07415P
700	PRE_DME_20061103	4	11/03/2006	07420P
700	PRE_DME_20061103	5	11/03/2006	07422P
700	PRE_DME_20061103	6		
700	PRE_DME_20061103	7	Table BT - Payment Information	
700	PRE_DME_20061103	8		
700	PRE_DME_20061103	9	TA WType Max.amount in payroll currency PyatMeth Recipient	
700	PRE_DME_20061103	10	Location Ctry Bank number Bank account	Re
700	PRE_DME_20061103	11	InvType Date Time Transfer ID Trans. Post bank	
700	PRE_DME_20061103	12	Amt. in Payment Currency Currency Region POR number POR ref.	
700	PRE_DME_20061103	13	01 /559 1,489.07 USD D	
700	PRE_DME_20061103	14	US 125000024	
700	PRE_DME_20061103	15	11/03/2006 07:41:18	00
700	PRE_DME_20061103	16	USD	

52

## Pre-Program DME – Create a Parallel Process (cont.)

### • Consolidate Program

- Once all the pre-program DME jobs have finished executing, another submit program is executed to run a consolidation process in parallel
- The consolidation process uses standard SAP functions to read and change the payroll cluster for each employee. The "safe" functions to do this are:
  - ▶ CALL FUNCTION 'PYXX\_READ\_PAYROLL\_RESULT'
  - ▶ CALL FUNCTION 'PYXX\_WRITE\_PAYROLL\_RESULT'
- These functions read the entire payroll result. The time stamp is changed in the BT table and then it is written back.

53

## Pre-Program DME – Create a Parallel Process (cont.)

- The entire process involves three programs:
  - The pre-program DME submit program
  - The consolidation program submit program
  - The consolidation program
- The process went from 11 hours to 20 minutes by running 16 jobs in parallel
  - The pre-program DME gets progressively slower when more employees are added due to inefficient internal table reads
- The risk of this enhancement was mitigated by extensive testing, running ST05 (SQL trace) to determine changed tables
- The reward was a streamlined payroll process

54

## Pre-Program DME – Create a Parallel Process (cont.)

Program Z\_PRE\_DME\_SUBMIT runs to submit the standard Pre-DME program many times

In this example 17 Pre-DME jobs are created – program RPCDTCUO

Job	Job	Job Crossref	Status	Start Date	Start Time	Duration(sec)	Delay (sec)
0030_DPREXC_PRE_DME_SPLIT1	TIDALSAP		Complete	01/07/2008	05:08:02	14	0
PRE_DME_20080107001	TIDALSAP		Complete	01/07/2008	05:08:08	514	0
PRE_DME_20080107002	TIDALSAP		Complete	01/07/2008	05:08:08	1,051	0
PRE_DME_20080107003	TIDALSAP		Complete	01/07/2008	05:08:09	885	0
PRE_DME_20080107004	TIDALSAP		Complete	01/07/2008	05:08:09	1,291	0
PRE_DME_20080107005	TIDALSAP		Complete	01/07/2008	05:08:09	1,593	0
PRE_DME_20080107006	TIDALSAP		Complete	01/07/2008	05:08:09	972	0
PRE_DME_20080107007	TIDALSAP		Complete	01/07/2008	05:08:13	266	1
PRE_DME_20080107008	TIDALSAP		Complete	01/07/2008	05:08:13	916	0
PRE_DME_20080107009	TIDALSAP		Complete	01/07/2008	05:08:13	950	0
PRE_DME_20080107010	TIDALSAP		Complete	01/07/2008	05:08:13	993	0
PRE_DME_20080107011	TIDALSAP		Complete	01/07/2008	05:08:14	895	0
PRE_DME_20080107012	TIDALSAP		Complete	01/07/2008	05:08:14	1,442	0
PRE_DME_20080107013	TIDALSAP		Complete	01/07/2008	05:08:15	857	1
PRE_DME_20080107014	TIDALSAP		Complete	01/07/2008	05:08:15	668	1
PRE_DME_20080107015	TIDALSAP		Complete	01/07/2008	05:08:15	954	0
PRE_DME_20080107016	TIDALSAP		Complete	01/07/2008	05:08:15	674	0
PRE_DME_20080107017	TIDALSAP		Complete	01/07/2008	05:08:16	353	1
0030_DPREXC_PRE_DME2_CONS_002	TIDALSAP		Complete	01/07/2008	05:38:12	38	0
DME_CONS_002	TIDALSAP		Complete	01/07/2008	05:38:18	669	0
DME_CONS_003	TIDALSAP		Complete	01/07/2008	05:38:21	296	0
DME_CONS_004	TIDALSAP		Complete	01/07/2008	05:38:21	544	0
DME_CONS_005	TIDALSAP		Complete	01/07/2008	05:38:21	792	0
DME_CONS_006	TIDALSAP		Complete	01/07/2008	05:38:23	424	0
DME_CONS_007	TIDALSAP		Complete	01/07/2008	05:38:25	450	0
DME_CONS_008	TIDALSAP		Complete	01/07/2008	05:38:27	403	0
DME_CONS_009	TIDALSAP		Complete	01/07/2008	05:38:29	410	0
DME_CONS_010	TIDALSAP		Complete	01/07/2008	05:38:29	442	0
DME_CONS_011	TIDALSAP		Complete	01/07/2008	05:38:31	408	0
DME_CONS_012	TIDALSAP		Complete	01/07/2008	05:38:32	749	0
DME_CONS_013	TIDALSAP		Complete	01/07/2008	05:38:33	304	0
DME_CONS_014	TIDALSAP		Complete	01/07/2008	05:38:35	244	0
DME_CONS_015	TIDALSAP		Complete	01/07/2008	05:38:40	214	0
DME_CONS_016	TIDALSAP		Complete	01/07/2008	05:38:40	228	0
DME_CONS_017	TIDALSAP		Complete	01/07/2008	05:38:49	35	0
*Summary						22,226	4

Program Z\_PRE\_DME\_SUBMIT\_CONS runs to submit the consolidation jobs

17 Consolidation jobs are created – program Z\_PRE\_DME\_CONSOLIDATE

55

## Third-Party Remittance Evaluation (4.7)

- The Third-Party Remittance Evaluation is the first in a series of steps required to remit third-party payments in payroll
- SAP ERP 6.0 has improved this process and this enhancement may not be necessary in that environment
- Third-Party Remittance Evaluation is essentially a payroll run with a special schema (U500) and therefore it could be greatly helped by parallel processing
- This enhancement can be done without any modifications to SAP code or tables

56

## Third-Party Remittance Evaluation (4.7) (cont.)

- Third-Party Remittance Evaluation creates pairs of TEMSE files that need to be stored. Once stored, the posting run will pick up all unprocessed runs in a single posting run.
- Splitting Third-Party Remittance Evaluation can be done manually without any custom code, but it would be very tedious storing the correct TEMSE file pairs of 20 or so parallel runs

57

## Third-Party Remittance Evaluation (4.7) (cont.)

### • Performance Strategy

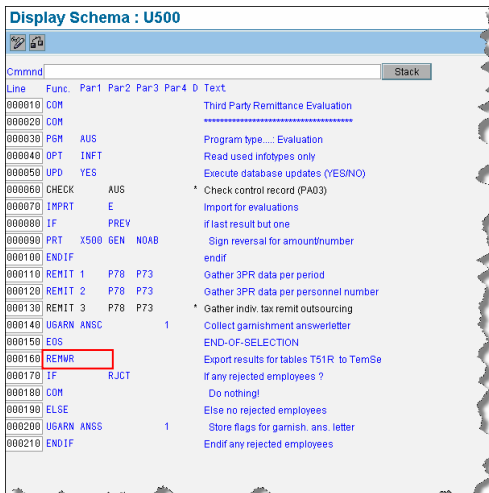
- We need to create many parallel runs of the evaluation run
- We need to find the best way to find and store the correct TEMSE file pairs for all the runs
  - ▶ One option is to read the spool file. The TEMSE file pairs are listed in an easy-to-identify format.
  - ▶ Another option was discovered by looking at the source code of the U500 schema functions in PE04. Function REMWR had the following statements:

```
EXPORT rem_tsobj_2 TO MEMORY ID 'REM_TSOBJ_2'.  
EXPORT rem_tsobj_1 TO MEMORY ID 'REM_TSOBJ_1'.
```

which will allow us to retrieve these values

58

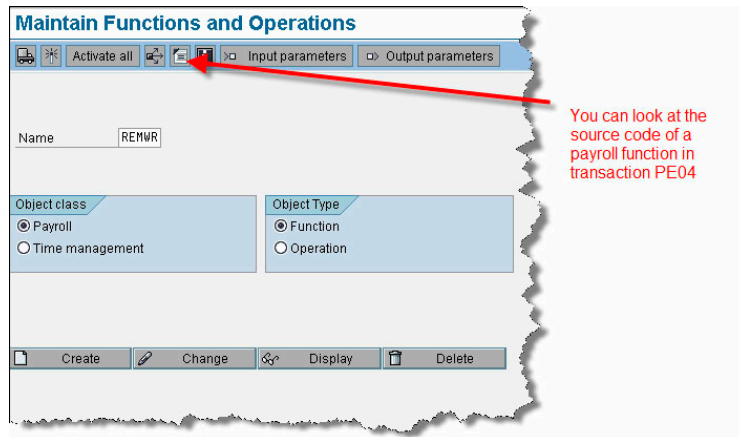
## Third-Party Remittance Evaluation (4.7) (cont.)



The U500 schema has a function REMWR that exports the TEMSE file. We can look at the source code of this function with transaction PE04.

59

## Third-Party Remittance Evaluation (4.7) (cont.)

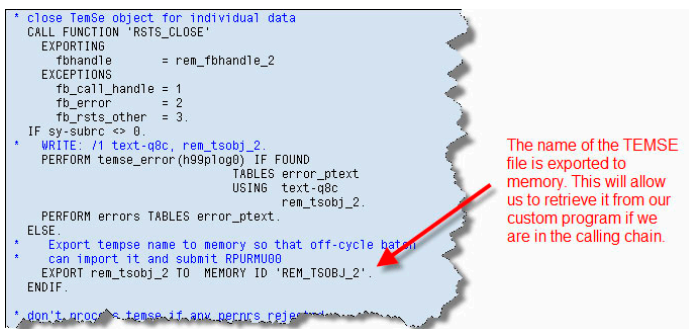


You can look at the source code of a payroll function in transaction PE04

Transaction – PE04

60

## Third-Party Remittance Evaluation (4.7) (cont.)



The name of the TEMSE file is exported to memory. This will allow us to retrieve it from our custom program if we are in the calling chain.

61

## Third-Party Remittance Evaluation (4.7) (cont.)

- Performance Strategy
  - Create two custom programs to complete the evaluation and store process
    - ▶ Program 1
      - Submit program 2 passing employee number ranges to each job
    - ▶ Program 2
      - Submit RPURMU00 for the passed employee number range
      - Retrieve the names of the TEMSE files via the import statement
      - Submit RPURMP00 to store the TEMSE files

62

## Third-Party Remittance Evaluation (4.7) (cont.)

```
DESCRIBE TABLE itab_pernr LINES itab_all_rows.  
IF sy-subrc = 0 AND itab_all_rows GT numjobs.  
  itab_all_chunk = itab_all_rows / numjobs.  
ENDIF.
```

A table of all in scope employees is evaluated for the approximate size of each job

This sample shows how to divide a table of employee numbers into multiple jobs. All employees in a given personnel area are contained in a single job.

```
LOOP AT itab_pernr.  
  ADD 1 TO count.  
  AT END OF werks.  
    IF NOT itab_pernr-werks IS INITIAL.  
      IF count GT itab_all_chunk.  
        MOVE itab_pernr-werks TO s_werks-low.  
        MOVE 'I' TO s_werks-sign.  
        MOVE 'EQ' TO s_werks-option.  
        APPEND s_werks.  
        PERFORM submit.  
        REFRESH s_werks.  
        CLEAR s_werks.  
        CLEAR count.  
      ELSE.  
        MOVE itab_pernr-werks TO s_werks-low.  
        MOVE 'I' TO s_werks-sign.  
        MOVE 'EQ' TO s_werks-option.  
        APPEND s_werks.  
      ENDIF.  
    ENDIF.  
  ENDAT.  
ENDLOOP.  
  
LOOP AT s_werks.  
  
  ENDOLOOP.  
  IF NOT s_werks-low IS INITIAL.  
    PERFORM submit.  
  ENDIF.
```

63

## What We'll Cover ...

- Common Performance Strategies
- Overview of Performance Problem Areas
- Performance Enhancements for Custom Programs
- Performance Enhancements for Payroll Schema
- Enhancing Standard Payroll Programs for Parallel Processing
- Wrap-up

64

## Wrap-Up

- SAP customers with very large employee populations have unique performance issues that need to be addressed
- Standard Payroll Processes can be made to run quickly using innovative techniques
  - Parallel processing is your best bet for improving production throughput on standard programs
  - Created methods can be used to allow the pre-DME and Third-Party Remittance Evaluation to be run in a parallel mode
  - Payroll schema modifications need to be preformed with performance in mind
  - Use Matchcode W to prevent unnecessary recalculation of payroll results

65

## Wrap-Up (cont.)

---

- Custom Reports and Interfaces can be written with high performance in mind using some of the following techniques:
  - Effective use of logical databases when appropriate
  - Effective use of data stores for fast report retrieval
  - Effective use of database
  - Effective use of internal tables with custom programs
  - Effective interface design – read once, write many

66

## Resources

---

- Horst Keller, *The Official ABAP Reference* (SAP Press, 2005).
- Thomas Schneider, *SAP Performance Optimization Guide* (SAP Press, 2007).
- SAP online help (<http://help.sap.com>)

67

## 7 Key Points to Take Home

---

- Use Parallel Processing to improve runtimes of SAP standard and custom programs
- Enhance the pre-program DME program to significantly improve performance
- Use a “Read Once, Output Many” to avoid multiple runs of the same interface
- Take advantage of high performance ABAP development techniques to improve custom programs
- Use stored data to greatly improve the performance of online reports and interfaces

68

## 7 Key Points to Take Home (cont.)

- Use wrapper programs to enhance the performance of standard SAP programs
- Think before you code. Just because the program works does not mean it works effectively.

69

---

---

---

---

---

---

---

---

---

---

## Your Turn!



How to contact me:  
Phil Taylor  
philt@rowsix.net

70

---

---

---

---

---

---

---

---

---

---









**HR2008**  
MARCH 10-13 • ORLANDO

Wellesley Information Services, 990 Washington Street, Suite 308, Dedham, MA 02026  
*Copyright © 2008 Wellesley Information Services. All rights reserved.*