**Verarbeitung der BEBU-Daten (NOTIZEN)**

**Bemerkungen**

Vorgehen gem. Hr. Plüss: Zuerst erfolgt eine Aufteilung auf Kostenstellen und erst in einem zweiten Schritt kommt die BEBU-Kontierung.

Für die korrekte Zuweisung der Kontierung pro MA gilt folgende Prio (höchste Prio zu oberst):

1. Explizite Angabe einer Kostenstelle beim Erfassen von variablen LOA
2. Die beim einzelnen MA konfigurierte (individuelle) Kontierung + Verteilung (payroll\_employee\_ID; wenn 0, dann ist die Konfiguration personen-neutral)
3. Auf MA festgelegte Stammkostenstelle
4. Kontierungen, die zu einer bestimmten Firma gehören (payroll\_company\_ID; wenn 0, dann ist die Konfiguration firmen-neutral)
5. Die im Personalstamm beim einzelnen MA hinterlegte Stammkostenstelle (wenn `cost\_center` leer ist, ist die Konfiguration kostenstellen-neutral)

**Lösungsansatz**

1. Pro MA und LOA werden die Beträge (+Mengen) in der Tabelle *payroll\_tmp\_mgmt\_acc\_split* auf Kostenstellen aufgeteilt.
   1. Existiert zur jeweiligen LOA in der Tabelle *payroll\_employee\_account* eine Kostenstellenangabe, so wird diese mit höchster Prio verwendet und der gesamte Betrag (od. Menge) wird zu 100% dieser Kostenstelle belastet. Allfällige %-Verteilungen werden ignoriert; die betroffenen Records in *payroll\_tmp\_mgmt\_acc\_split* sollen daher als „abgeschlossen“ markiert werden.
   2. Existiert zur jeweiligen Person eine %-Verteilung, so werden die Beträge (Mengen) entsprechend verteilt. Personen-bezogene %-Verteilung hat Vorrang gegenüber allgemein definierter %-Verteilungen.
   3. Dito wie (b) jedoch mit allgemeinen %-Verteilungen  
      **Bei (b) und (c) ist im Speziellen noch zu berücksichtigen, wie mit Resten aus der %-Verteilung verfahren werden muss. [Rest je nach Einstellung entweder auf Stammkostenstelle oder auf Kostenstelle, die nach %-Verteilung den höchsten Betrag aufweist. Wenn Rest auf Stammkostenstelle, dann muss geprüft werden, ob diese als Record in *payroll\_tmp\_mgmt\_acc\_split* existiert. Wenn nicht, ist diese noch einzufügen.]  
      RUNDUNG?**
   4. Es gibt keine %-Verteilung, also wird der Betrag (od. Menge) zu 100% der Stammkostenstelle zugewiesen und der Record als „abgeschlossen“ markiert.
2. Mit den Informationen pro MA, LOA und KST kann nun die BEBU-Kontierung vorgenommen werden. Diese kann (vermutlich) analog der FIBU-Kontierung laufen.

payroll\_mgmt\_acc\_assign (InnoDB Table)

Tabelle für BEBU-Kontierungs-Konfiguration

|  |  |  |
| --- | --- | --- |
| **Feldname** | **Typ** | **Bemerkung** |
| payroll\_account\_ID | INT |  |
| payroll\_employee\_ID | INT | wenn 0, dann ist die Kontierung Personen-neutral |
| payroll\_company\_ID | INT | wenn 0, dann ist die Kontierung Firmen-neutral |
| account\_no | VARCHAR(15) |  |
| counter\_account\_no | VARCHAR(15) |  |
| cost\_center | VARCHAR(15) | wenn leer, dann ist die Kontierung Kostenstellen-neutral.  wenn Wert vorhanden, wird versucht dieses Feld mit *payroll\_employee.CostCenter* zu linken |
| debitcredit | TINYINT(1) | 0=SOLL / 1=HABEN |
| entry\_text | VARCHAR(50) |  |
| invert\_value | TINYINT(1) | 0=nicht invertieren / 1=invertieren |
| processing\_order | TINYINT(4), **INDEX** | 1 = wenn payroll\_employee\_ID > 0  2 = wenn payroll\_company\_ID > 0 UND cost\_center != ‘’  3 = wenn payroll\_company\_ID > 0 UND cost\_center = ‘’  4 = wenn payroll\_company\_ID = 0 UND cost\_center != ‘’  9 = wenn payroll\_employee\_ID = 0 UND payroll\_company\_ID = 0 UND cost\_center = ‘’ |

payroll\_mgmt\_acc\_split

Diese Tabelle dient zur Speicherung der Informationen für die %-Verteilung (BEBU).

|  |  |  |
| --- | --- | --- |
| **Feldname** | **Typ** | **Bemerkung** |
| id | INT | PK, Auto-increment ; ist eventuell nötig, um Primary Keys miteinander zu verknüpfen… |
| payroll\_employee\_ID | INT |  |
| payroll\_company\_ID | INT |  |
| processing\_order | TINYINT | je höher dieser Wert, desto niedriger die Prio NÖTIG? |
| payroll\_account\_ID | VARCHAR(5) | Lohnart |
| cost\_center | VARCHAR(15) |  |
| amount | DECIMAL(5,2) | Betrag: wird als Prozentbetrag verwendet |
| invert\_value | TINYINT(1) | 0=nicht invertieren / 1=invertieren |
| remainder | TINYINT(1) | 0= Rest auf Stammkostenstelle  1= Rest auf Kostenstelle, die nach %-Verteilung den höchsten Betrag aufweist |
| processing\_order | TINYINT | 1 = wenn payroll\_employee\_ID > 0  2 = wenn payroll\_employee\_ID = 0 UND payroll\_company\_ID > 0  9 = wenn payroll\_employee\_ID = 0 UND payroll\_company\_ID = 0 |

payroll\_tmp\_mgmt\_acc\_split (Memory Table)

Temporäre Tabelle für die Aufteilung auf die Kostenstellen. Die ermittelten Daten dienen als Grundlage für die anschliessende BEBU-Kontierung.

|  |  |  |
| --- | --- | --- |
| **Feldname** | **Typ** | **Bemerkung** |
| payroll\_period\_ID | INT |  |
| payroll\_employee\_ID | INT |  |
| payroll\_company\_ID | INT |  |
| cost\_center | VARCHAR(15) |  |
| amount\_initial | DECIMAL(10,2) | Entspricht dem LOA-BETRAG |
| amount\_available | DECIMAL(10,2) | Entspricht dem LOA-BETRAG abzüglich allfälliger bereits abgezogener Beträge |
| amount | DECIMAL(10,2) | 99'999'999.99 |
| processing\_order | TINYINT | ***INDEX***  je höher dieser Wert, desto niedriger die Prio |
| invert\_value | TINYINT(1) | 0=nicht invertieren / 1=invertieren |
| payroll\_account\_ID | VARCHAR(5) |  |
| amount\_quantity | TINYINT(1) | 0=BETRAG / 1=MENGE |
| processing\_done | TINYINT(1) | default: 0 / sobald hier 1 steht, ist der Record abgeschlossen und darf nicht mehr verändert werden |
| having\_rounding | TINYINT(1) |  |
| rounding | DECIMAL(7,2) |  |
| remainder | TINYINT(1) |  |

payroll\_mgmt\_acc\_entry (InnoDB Table)

Tabelle für BEBU-Buchungen

|  |  |  |
| --- | --- | --- |
| **Feldname** | **Typ** | **Bemerkung** |
| payroll\_period\_ID | INT |  |
| payroll\_employee\_ID | INT |  |
| payroll\_account\_ID | VARCHAR(5) |  |
| entry\_date | DATE | nicht nötig; hier kann stattdessen payroll\_period\_employee.mgmt\_acc\_date verwendet werden |
| account\_no | VARCHAR(15) |  |
| counter\_account\_no | VARCHAR(15) |  |
| cost\_center | VARCHAR(15) |  |
| amount\_local | DECIMAL(11,2) |  |
| amount\_foreign | DECIMAL(11,2) | benötigt ? |
| debitcredit | TINYINT(1) | 0=SOLL / 1=HABEN |
| entry\_text | VARCHAR(50) |  |
| amount\_quantity | TINYINT(1) | 0=BETRAG / 1=MENGE |

**Queries für den Verbuchungsprozess:**

Zuerst nur die BEBU-Daten der betroffenen MA löschen…

|  |
| --- |
| DELETE accspl FROM `payroll\_tmp\_mgmt\_acc\_split` accspl  INNER JOIN `payroll\_tmp\_change\_mng` emplList ON accspl.`payroll\_employee\_ID`=emplList.`numID` AND emplList.`core\_user\_ID`=currentUser  WHERE accspl.`payroll\_period\_ID`=currentPeriod; |

…dann die Records in einer temporären MEMORY-Table neu anlegen. Zuerst LOA mit expliziter Übersteuerung der Kostenstelle…

|  |
| --- |
| INSERT INTO `payroll\_tmp\_mgmt\_acc\_split`(`payroll\_period\_ID`,`payroll\_company\_ID`,`payroll\_employee\_ID`,`cost\_center`,`payroll\_account\_ID`,`amount\_initial`,`amount\_available`,`amount`,`processing\_order`,`invert\_value`,`amount\_quantity`,`processing\_done`,`having\_rounding`,`rounding`)  SELECT currentPeriod,emp.`payroll\_company\_ID`,empacc.`payroll\_employee\_ID`,empacc.`CostCenter`,empacc.`payroll\_account\_ID`,0,0,calc.`amount`,0,0,0,1,0,0  FROM `payroll\_employee\_account` empacc  INNER JOIN `payroll\_tmp\_change\_mng` emplList ON empacc.`payroll\_employee\_ID`=emplList.`numID` AND emplList.`core\_user\_ID`=currentUser  INNER JOIN `payroll\_employee` emp ON emp.`id`=emplList.`numID`  INNER JOIN `payroll\_calculation\_current` calc ON calc.`payroll\_employee\_ID`=emplList.`numID` AND calc.`payroll\_account\_ID`=empacc.`payroll\_account\_ID` AND calc.`payroll\_period\_ID`= currentPeriod  WHERE empacc.`CostCenter`!=''; |

…als Nächstes ebenfalls Records in temporärer MEMORY-Table anlegen, aber jetzt die übrigen %-Verteilungen. Wichtig: Bereits verarbeitete LOA ausschliessen, 100%-Zuweisungen können direkt verarbeitet und die enspr. Records abgeschlossen/fixiert werden...

|  |
| --- |
| INSERT INTO `payroll\_tmp\_mgmt\_acc\_split`(`payroll\_period\_ID`,`payroll\_company\_ID`,`payroll\_employee\_ID`,`cost\_center`,`payroll\_account\_ID`,`amount\_initial`,`amount\_available`,`amount`,`processing\_order`,`invert\_value`,`amount\_quantity`,`processing\_done`,`having\_rounding`,`rounding`)  SELECT calc.`payroll\_period\_ID`, emp.`payroll\_company\_ID`, emp.`id`, IF(accasng.`payroll\_employee\_ID`=0 AND emp.`CostCenter`!='', emp.`CostCenter`,accasng.`cost\_center`), calc.`payroll\_account\_ID`, calc.`amount`,calc.`amount`, IF(accasng.`amount`=100,IF(accasng.`invert\_value`=1, calc.`amount`\*-1, calc.`amount`), accasng.`amount`) ,accasng.`processing\_order`,accasng.`invert\_value`,0,IF(accasng.`amount`=100,1,0),0,0  FROM `payroll\_mgmt\_acc\_split` accasng  INNER JOIN `payroll\_calculation\_current` calc ON calc.`payroll\_account\_ID`=accasng.`payroll\_account\_ID` AND calc.`payroll\_period\_ID`=currentPeriod  INNER JOIN `payroll\_tmp\_change\_mng` emplList ON calc.`payroll\_employee\_ID`=emplList.`numID` AND emplList.`core\_user\_ID`=currentUser  INNER JOIN `payroll\_employee` emp ON emp.`id`=emplList.`numID`  LEFT JOIN `payroll\_tmp\_mgmt\_acc\_split` tas ON tas.`payroll\_period\_ID`=calc.`payroll\_period\_ID` AND tas.`payroll\_employee\_ID`=calc.`payroll\_employee\_ID` AND tas.`payroll\_account\_ID`=calc.`payroll\_account\_ID` AND tas.`processing\_done`=1  INNER JOIN (SELECT ep.`id`, aa.`payroll\_account\_ID`, MIN(aa.`processing\_order`) as po  FROM `payroll\_mgmt\_acc\_split` aa  INNER JOIN `payroll\_tmp\_change\_mng` el ON el.`core\_user\_ID`=currentUser  INNER JOIN `payroll\_employee` ep ON ep.`id`=el.`numID`  WHERE (aa.`payroll\_employee\_ID`=0 OR aa.`payroll\_employee\_ID`=ep.`id`) AND (aa.`payroll\_company\_ID`=0 OR aa.`payroll\_company\_ID`=ep.`payroll\_company\_ID`)  GROUP BY ep.`id`, aa.`payroll\_account\_ID`) tx ON tx.`payroll\_account\_ID`=accasng.`payroll\_account\_ID` AND tx.`po`=accasng.`processing\_order` AND calc.`payroll\_employee\_ID`=tx.`id`  WHERE tas.`processing\_done` IS NULL; |

…Werte runden…

|  |
| --- |
| UPDATE `payroll\_tmp\_mgmt\_acc\_split` SET `amount`=ROUND(`amount`/`rounding`)\*`rounding` WHERE `having\_rounding`=1 AND `processing\_done`=0; |

…Falls es einen Restbetrag gibt, wird dieser nun noch entsprechend zugewiesen…

|  |
| --- |
| INSERT INTO `payroll\_tmp\_mgmt\_acc\_split`(`payroll\_period\_ID`,`payroll\_company\_ID`,`payroll\_employee\_ID`,`cost\_center`,`payroll\_account\_ID`,`amount\_initial`,`amount\_available`,`amount`,`processing\_order`,`invert\_value`,`amount\_quantity`,`processing\_done`,`having\_rounding`,`rounding`,`remainder`)  SELECT accsplt.`payroll\_period\_ID`,accsplt.`payroll\_company\_ID`,accsplt.`payroll\_employee\_ID`,IF(x1.`remainder`=1,accsplt.`cost\_center`,empl.`CostCenter`),accsplt.`payroll\_account\_ID`,accsplt.`amount\_initial`,accsplt.`amount\_available`, x1.`amount\_initial`-x1.`amount\_sum`, 0,0,accsplt.`amount\_quantity`,1,accsplt.`having\_rounding`,accsplt.`rounding`,accsplt.`remainder`  FROM payroll\_tmp\_mgmt\_acc\_split accsplt  INNER JOIN (SELECT `payroll\_employee\_ID`, `payroll\_account\_ID`, `amount\_initial`, SUM(`amount`) as amount\_sum, MAX(`amount`) as amount\_max, `remainder` FROM payroll\_tmp\_mgmt\_acc\_split WHERE processing\_done=0 GROUP BY payroll\_employee\_ID, payroll\_account\_ID) x1 ON accsplt.`payroll\_employee\_ID`=x1.`payroll\_employee\_ID` AND accsplt.`payroll\_account\_ID`=x1.`payroll\_account\_ID` AND accsplt.`amount`=x1.amount\_max  INNER JOIN `payroll\_employee` empl ON empl.`id`=accsplt.`payroll\_employee\_ID`  WHERE processing\_done=0; |

**Hinweis: Problem bei obigem Query könnte sein, dass wenn mehrere gleiche Maximalwerte existieren zu viele Records angelegt werden, da wir über** accsplt.`amount`=x1.amount\_max **joinen. Der Join an dieser Stelle sollte noch optimiert werden.**

…Records der temporären Tabelle in eine InnoDB Tabelle speichern…

|  |
| --- |
| INSERT INTO `payroll\_mgmt\_acc\_entry`(`payroll\_period\_ID`, `payroll\_employee\_ID`, `payroll\_account\_ID`, `account\_no`, `counter\_account\_no`, `cost\_center`, `amount\_local`, `debitcredit`, `entry\_text`, `amount\_quantity`)  SELECT calc.`payroll\_period\_ID`, calc.`payroll\_employee\_ID`, accasng.`payroll\_account\_ID`, accasng.`account\_no`, accasng.`counter\_account\_no`, calc.`cost\_center`, IF(accasng.`invert\_value`=1, calc.`amount`\*-1, calc.`amount`), accasng.`debitcredit`, accasng.`entry\_text`, 0  FROM payroll\_mgmt\_acc\_assign accasng  INNER JOIN (SELECT `payroll\_period\_ID`, `payroll\_employee\_ID`, `payroll\_account\_ID`, `cost\_center`, SUM(`amount`) as `amount` FROM payroll\_tmp\_mgmt\_acc\_split GROUP BY `payroll\_employee\_ID`,`payroll\_account\_ID`,`cost\_center`) calc ON calc.`payroll\_account\_ID`=accasng.`payroll\_account\_ID` AND calc.`payroll\_period\_ID`=currentPeriod  INNER JOIN `payroll\_tmp\_change\_mng` emplList ON calc.`payroll\_employee\_ID`=emplList.`numID` AND emplList.`core\_user\_ID`=currentUser  INNER JOIN (SELECT ep.`id`, aa.`payroll\_account\_ID`, MIN(aa.`processing\_order`) as po  FROM payroll\_mgmt\_acc\_assign aa  INNER JOIN `payroll\_tmp\_change\_mng` el ON el.`core\_user\_ID`=currentUser  INNER JOIN `payroll\_employee` ep ON ep.`id`=el.`numID`  WHERE (aa.`payroll\_employee\_ID`=0 OR aa.`payroll\_employee\_ID`=ep.`id`) AND (aa.`cost\_center`='' OR aa.`cost\_center`=ep.`CostCenter`) AND (aa.`payroll\_company\_ID`=0 OR aa.`payroll\_company\_ID`=ep.`payroll\_company\_ID`)  GROUP BY ep.`id`, aa.`payroll\_account\_ID`) tx ON tx.`payroll\_account\_ID`=accasng.`payroll\_account\_ID` AND tx.`po`=accasng.`processing\_order` AND calc.`payroll\_employee\_ID`=tx.`id`; |

…Records der temporären Tabelle löschen…

…und das Datum (Buchungsdatum) in payroll\_period\_employee anpassen plus die ID des Benutzers speichern, der die Verbuchung durchgeführt hat…

|  |
| --- |
| UPDATE payroll\_period\_employee prdemp  INNER JOIN `payroll\_tmp\_change\_mng` emplList ON prdemp.`payroll\_employee\_ID`=emplList.`numID` AND emplList.`core\_user\_ID`=currentUser  SET prdemp.`mgmt\_acc\_date`=’2013-01-01’, prdemp.`core\_user\_ID\_mgmt\_acc`=currentUser  WHERE prdemp.`payroll\_period\_ID`=currentPeriod |