Inhaltsverzeichnis

Erweiterte Maskenprogrammierung: Freemarker Templates

Die OpenSource-Bibliothek FreeMarker (www.freemarker.org) wird als Template-Engine eingesetzt.

Damit Sie in einer Abfrage die Freemarker Funktionalität benutzen können, muss im Kopf des Select-Statements eine Hinweiszeile (--freemarker template) enthalten sein.

Klassische Verarbeitung ohne Freemarker

Die einzelnen Abfragen (auch synonym Masken genannt) enthalten Platzhalter, wie beispielsweise:

```
select  monat,sum(betrag)  from  cob_busa
where      monat=<>;
```

Auf der Maske gibt es ein Feld Monat. Vorm Abschicken des SQL wird <> durch den gewählten Wert ersetzt.

Falls eine Maske Felder enthält, welche optional gefüllt werden können, so wird der Ausdruck zwischen /* und */ gesetzt. Das hat zur Folge, dass dieser entfernt wird, falls kein Wert ausgewählt wurde.

Aus beispielsweise

```
select  monat,sum(betrag)  from  cob_busa
where      monat=<>

/*  and  gege=<>  */;
```

wird, falls kein Geldgeber ausgewählt wurde z. B.

```
select  monat,sum(betrag)  from  cob_busa
where      monat=1;
```

, aber falls ein Geldgeber ausgewählt wurde z. B.

```
select  monat,sum(betrag)  from  cob_busa
where      monat=1
and  gege=3;
```

Achtung:

Der Ausdruck in <> darf nur einmal in dem optionalen Block vorkommen. Falls er zweimal benötigt wird, müssen diese auf zwei Blöcke aufgeteilt werden. Z. B.:

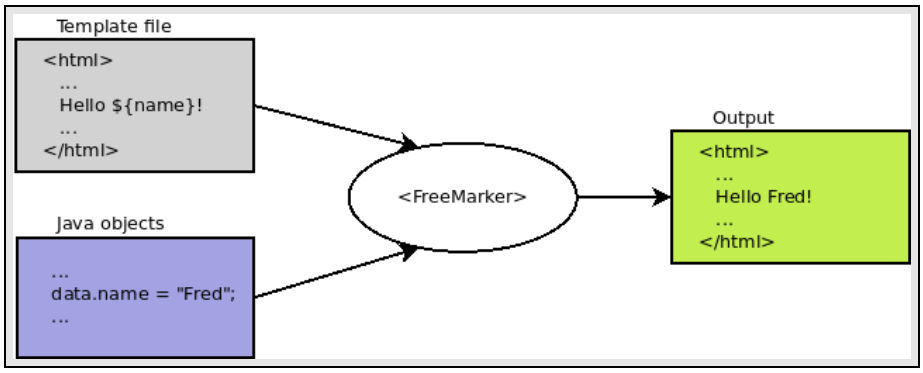
```
/*  and  (dr  in  (<>)  */

/*  or  dr2  in  (<>))*/
```

FreeMarker-Transformation

Übersicht

Nach der klassischen Transformation mit generateSql folgt ggfs. die FreeMarker Transformation.

FreeMarker transformiert eine Vorlage (template) mit Hilfe eines Datenmodells (mit Java Objekten) zu einem Ausgabetext.



Sehr oft wird es zur Erzeugung von HTML benutzt, wir produzieren stattdessen SQL.

Die Java-Objekte im Datenmodell sind die Felder, die auf der Maske zur Auswahl stehen.

Als einfachsten Anwendungsfall könnten wir also für eine Maske mit einem Monatsfeld statt des klassischen SuperX-Tags

```
select monat,sum(betrag) from tmp_busa where monat=<>
```

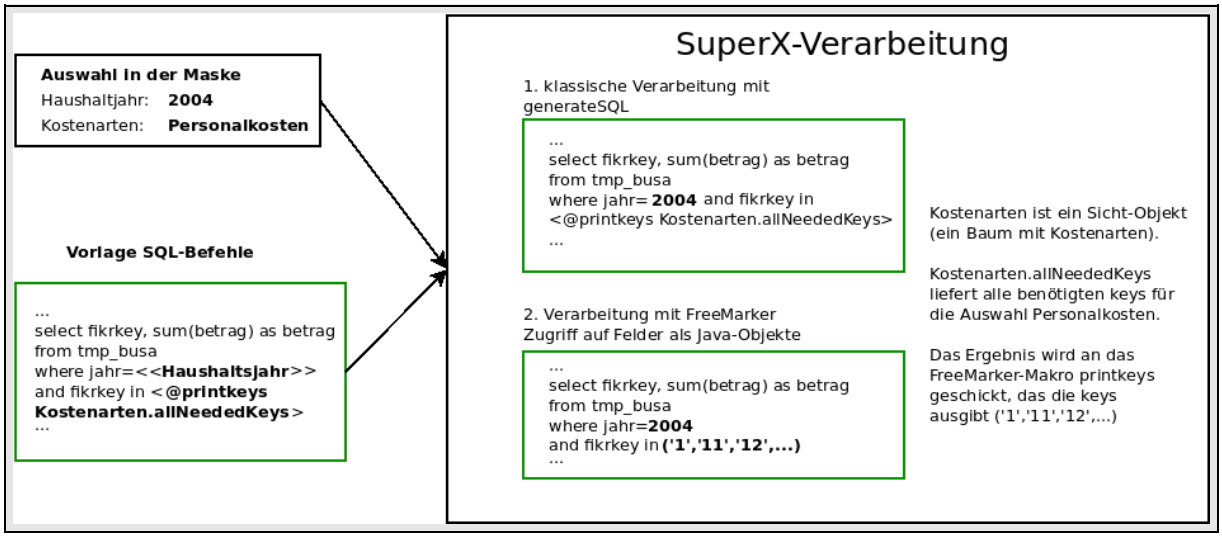
auch die FreeMarker Notation nehmen.

```
select monat,sum(betrag) from tmp_busa where monat=${Monat}
```

Innerhalb von Freemarker kann ${} entfallen:

```
<#if Monat=1<#if Monat=1> ...
```

Ein komplexes Beispiel:



Interaktion Freemarker mit Maskenfeldern

Das Servlet verarbeitet bei der Maskenausführung zuerst die Maskenfelder, die mit den Steuerungszeichen "<<" und ">>" in das

select_stmt eingefügt werden.

So wird z.B. bei der Anweisung

```
<#if (einFach.strukturStr = "Fach (intern)" && <>=10 )>
```

im Servlet zunächst die Variable "Aggregierung Fach " aufgelöst und dann erst startet der Freemarker-Parser. Für das o.g. Statement würde also zunächst:

```
<#if (einFach.strukturStr = "Fach (intern)" && 10=10 )>
```

ersetzt und dann die Freemarker-IF-Bedingung ausgewertet.

So können Sie z.B. auch Checkboxen auswerten, wenn sie nicht angekreuzt sind: nehmen wir an das Feld heißt "nur aktuelle Stg." und hat den Typ "char" und die art=10, dann wäre der Code:

```
<#if "<>" ="https://superxhosting.de/wiki/index.php/null">

select 'Beispiel: nur aktuelle Stg. nicht angekreuzt' from xdummy;
```

Wenn das Feld nicht angekreuzt ist, macht der SuperX-Parser daraus zunächst

```
<#if "https://superxhosting.de/wiki/index.php/null" ="https://superxhosting.de/wiki/index.php/null">
```

und Freemarker würde diese if-Bedingung mit "wahr" beantworten und den SQL "select 'Beispiel: nur aktuelle Stg. nicht angekreuzt' from xdummy;" ausführen.

Wenn das Feld angekreuzt wird, macht der SuperX-Parser daraus zunächst

```
<#if "'true'" ="https://superxhosting.de/wiki/index.php/null">
```

und Freemarker würde diese if-Bedingung mit "falsch" beantworten und den SQL "select 'Beispiel: nur aktuelle Stg. nicht angekreuzt' from xdummy;" nicht ausführen.

Programmieren mit FreeMarker

FreeMarker unterstützt praktisch alle Konzepte klassischer Programmiersprachen.

Die Tags sind HTML-ähnlich.

if-Abfragen

Mit Freemarker können Sie z.B. if-then-Abfragen in normales SQL einbauen, z.B. um je nach gewünschter Aggregierungsstufe einen unterschiedlichen insert zu benutzen:

```
<#if "<>"="stark">

insert into .. select ...
<#elseif "<>="mittel">

insert into .. select ...
<#else>

insert into .. select ...
```

Der klassische SuperX-Tag <<Aggregierung>> wird vor der FreeMarker Transformation ersetzt, sodass FreeMarker effektiv zwei Strings vergleicht (if "stark"="stark").

Alternativ könnte den ausgewählten Wert des Felds Aggregierung im Java-Objekt direkt ansprechen.

```
<#if  Aggregierung="stark">
```

Hier braucht kein ${} um Aggregierung, da wir ja schon innerhalb einer FreeMarker-Anweisung sind.

Variablen: assign

Mit assign kann man eigene Variabeln definieren. Z. B.

```
<#assign  sortnr=0<#assign  sortnr=0>
<#assign  sortnr=sortnr+1<#assign  sortnr=sortnr+1>
```

oder

```
<#if  "<>"="NEIN">


<#assign  quelltabelle  =  "sos_statistik">


<#else>


<#assign  quelltabelle  =  "sos_statistik">
```

<<Stichtagsbezogen>> wird von der SuperX-Transformation ersetzt, sodass Freemarker vergleicht:

<#if "JA"="NEIN"> oder <#if "NEIN"="NEIN">

Die Variablen werden wie folgt abgerufen:

```
insert  into  ...  values  (${sortnr},...)
```

oder

```
select  ....  from  ${quelltabelle}  where  ...
```

Folgender Effekt ist schon mal aufgetreten:

Wenn man ein einer Abfrage z.B. schreibt

```
<#assign  sortnr=sortnr+1<#assign  sortnr=sortnr+1>
insert  into  tmp_rs_base
(struktur,text,  ...
```

dann klappt das nicht, man muss unter dem assign eine Leerzeile einfügen.

Variablen für Fortgeschrittene: sqlvars

Manchmal wünscht man sich mit FreeMarker auf Variablen zugreifen zu können, die aus der Datenbank gefüllt werden müssten, weil sie nicht als Felder auf der Maske vorkommen. Allerdings muss die Freemarker-Transformation ja schon laufen, bevor der fertige SQL an die Datenbank geschickt wird, weil Postgres/Informix ja mit FreeMarker-Befehlen nichts anfangen können.

Man kann also nicht so etwas

```
<#assign  gegename="select  name  from  fin_geldgeber  G  where  G.xyz=<>">
```

machen, weil die Variable gegename dann einfach nur select... als String enthält, Freemarker hat mit der Datenbank keinen direkten Kontakt.

Dies lässt sich umgehen, indem man einen Block anlegt:

```
<!--  einfache  Variable--<!--  einfache  Variable-->
```

```
select sp_table_exists('fin_geldgeber') from xdummy


<!-- Listenvariable -- 1. Spalte key (darf nicht null sein!), 2. Spalte name--<!-- Listenvariable -- 1. Spalte key (darf nicht null sein!), 2. Spalte name-->


select ggnr,ggname1 from fin_geldgeber
```

Anschließend kann man in der Abfrage mit FreeMarker auf diese aus der Datenbank gefüllten Variablen zugreifen:

```
<#if fin_geldgeber_exists=1<#if fin_geldgeber_exists=1>
insert into .. select * from fin_geldgeber;
<#else>


insert into .. select * from fin_geldgeber;
```

Die Variable wird als einfacher Wert erkannt, weil nur eine Spalte im SQL selektiert wurde. Sofern der Select mehrere Zeilen liefert, wird nur der letzte gefundene Wert hinterlegt.

Die zweite Art von Variable (geldgeber) wird als Liste von Items geführt und kann in der Abfrage benutzt werden z.B. mit

```
where ggnr in (
<#foreach gg in geldgeber<#foreach gg in geldgeber>
${gg.key},
```

Die erste per SQL eingelesene Spalte ist das Attribut key, die zweite name.

Optional kann auch noch dritte/vierte Spalte mit Strukturinfo eingelesen werden:

```
select ggnr,ggname1,klr_geldgeber,strukturint from fin_geldgeber
```

Auf die dritte Spalte kann man zugreifen über das Attribut ".strukturStr", und die vierte Spalte mit ".strukturInt", also für das orbige Beispiel mit:

```
<#foreach gg in geldgeber <#foreach gg in geldgeber >
-- Hier steht der Inhalt der Spalte klr_geldgeber: ${gg.strukturStr}
...
```

Analog für strukturInt.

In den sqlvars kann auch freemarker-syntax und repository/konstanten benutzt werden, obstruses Beispiel:

```
<#if K_FIN_Quellsystem=1<#if K_FIN_Quellsystem=1>
select ggnr,ggname1 from fin_geldgeber where ggnr in ${FIN_Drittmittel}
<#else>


select ggnr,ggname2 from fin_geldgeber where ggnr in ${FIN_Drittmittel}
```

Achtung : bei den SQL-Statements innerhalb von SQLVAR-Abschnitten dürfen Sie keine "<" oder ">"-Zeichen nutzen, sondern mit der html-Notation < und > umschreiben, also statt:

```
select distinct tid, eintrag
from semester
where tid < =20101
order by 1;
```

schreiben Sie besser

```
select distinct tid, eintrag
from semester
where tid &lt; =20101
order by 1;
```

Bei Problemen mit sqlvars, deren SQL dynamisch generiert werden, ist es schwierig, sich den produzierten SQL anzusehen (insb. wenn nur Browserzugriff besteht).

Beispiel:

```
<!sum(hhans)">from fin_konto_aggr
where rechnungsjahr= <>

and ch110_institut in ${Kostenstelle.allNeededElements}>
```

Trick: Einen SQL-Syntaxfehler einbauen:

```
<!XXXXX">sum(hhans) from fin_konto_aggr
where rechnungsjahr= <>

and ch110_institut in ${Kostenstelle.allNeededElements}>
```

Dadurch wird ein SQL-Fehler erzeugt, der im Browser angezeigt wird und man kann sehen, wie <> ersetzt wurde und Freemarker arbeitet.

Man kann auch Inhalt von Freemarker-Variablen bzw. vorhergehenden SQLVars anzeigen lassen.

Beispiel Konstante FIN_Quellsystem und vorhergehende sqlvar instname:

```
select drucktext from fin_inst where key_apnr=<>
```

```
<!'${K_FIN_Quellsystem}">- ${instname}' from xdummy;
select XX sum(hhans) from fin_konto_aggr where
rechnungsjahr= <>

and ch110_institut in ${Kostenstelle.allNeededElements}>
```

In sqlvars kann man auch auf die vorherigen sqlvars zugreifen:

```
select ..


<!..
where
<#if">v1=1<#if  v1=1>
feld=1
<#else>


feld=2
>
```

geht bisher nur innerhalb von if-Anweisungen o.ä.

CDATA ist wichtig für wohlgeformtes xml

```
select  apnr  from  konstanten  where  beschreibung='  COB_FIN_STARTJAHR  '
```

```
<!distinct">klrjahr*12+klrmonat,'monat'
from  fin_buch_akt
where  klrjahr>0  and  klrjahr>=${COB_FIN_TO_BUSA_STARTJAHR}>
```

hier kommt Fehlermeldung smallint Operator >= existiert nicht.

Hintergrund:

${COB_FIN_TO_BUSA_STARTJAHR} wird als $-Kommentar vom klassischen general-sql gelöscht und

```
select  distinct  klrjahr*12+klrmonat,'monat'
from  fin_buch_akt
where  klrjahr>0  and  klrjahr>=
```

abgeschickt.

Sichten und Freemarker

Bei Sicht-Feldern (Feldart 12) gibt es besondere Möglichkeiten. Mittels Java-Reflection kann FreeMarker auf Methoden der Objekte im Datenmodell zugreifen. Bei Sichtfeldern wie "Institution" oder "Kostenart" sind folgende Methoden interessant.

allNeededKeys für temporäre Datentabellen

Diese Methode liefert alle benötigten Schlüssel.

Wenn bei dme Sichtenfeld nichts ausgewählt wurde, werden einfach alle im Baum vorhandenen Schlüssel geliefert.

Wenn z.B. Personalkosten ausgewählt wurde, wird nur der Schlüssel von Personalkosten ('1') und dessen Unterknoten (z.B. '11','12') geliefert.

Dafür wird noch das allgemeine Makro printkeys benutzt.

```
<@printkeys
Kostenarten.allNeededKeys
```

Beispiel für Erstellung einer temporären Datentabelle

```
execute  procedure
sp_user_orga_child
```

(Statt Benutzung der Prozedur sp_user_orga_child könnte man analog verwenden:

```
where  B.110_institut  in
<@printkeys
```

Ggfs. versteckte Knoten werden hier mit ausgegeben.

Bei Kostenstellen-Feldern werden nur erlaubte Einträge ausgegeben.

Neu ab kern4.5:
Diese Methode kann man auch für Feldart 1-Felder benutzen, z.B. mit

```
${Haushaltsprogramm.allNe
ededKeys}.
```

ContainsElements

Neu ab Kern4.5:
Für Sichten und auch Feldart 1 kann man ermitteln, ob eine Auswahl möglich ist.

z.B.

```
<#if
Kostenstelle.containsEleme
```

Oder bei Feldart 1 :

```
 <#if
HaushaltsprogrammObject.
```

keysToRoot

Bei dem Maskenfeld Verteilschritte wird die Sicht von unten nach oben durchlaufen. sind ein ungewöhnliches Konzept.

1
2
3

Schritt 3 ist die Summe aus 1-3.

In Abfragen wird

```
<@printkeys
Verteilschritt.keysToRoot/>
```

benutzt

elements: Schleifen über ausgewählte Knoten

Die Methode elements liefert eine Collection, entweder über alle Knoten im Sichtbaum oder nur über einen ausgewählten Knoten und deren Kinder.

Beispiel:

```
<#foreach eineKostenart in Kostenarten.elements>
```

elements liefert die Knoten genau in der Reihenfolge, in der sie auch im Baum sind. Alternativ kann man *breadthFirstElements* oder *depthFirstElements* angeben. Dann wird beim Baum zunächst in die Breite/Tiefe gegangen.

Wichtig:

Für eine foreach-Schleife werden auch bei Kostenstellen-Feldern bei eingeschränkten Usern immer alle Knoten ausgegeben, z.B. nur Rechte auf 11 und 13

root-Hochschule (Auswahl)
fak-Fakultäten (Auswahl)
1-fak1 (Auswahl)
11-Institut 1

Es werden alle Knoten durchlaufen, weil (Teil)summenzeilen interessant sein können. Anders ist es bei Berechnung (Methode subkeys) da werden nur die tatsächlich erlaubten Schlüssel ausgegeben.

Zugriff auf einzelne Knoten im Baum

Im Rahmen einer forEach Schleife bekommt man Zugriff auf einzelne Elemente eines Sichtenbaums. Für die einzelnen Knoten kann FreeMarker wiederum mittels Java-Reflection auf bestimmte Methoden zugreifen.

Knotenelement id, name

Die Attribute id und name liefern Zugriff auf den Schlüssel und den Namen des Knotens, z.B.

```
Insert into tmp_erg (fikr , betrag )
```

Knoten-Attribut subkeys

Die Methode 'subkeys' liefert eine Liste mit dem Schlüssel des aktuellen Knotens (z.B. Personalkosten '1') und aller seiner Unterkn* oten (z.B. '11','12') (auch von versteckten Knoten!)

```
Insert into tmp_erg (fikr , betrag )
```

Bei Kostenstellen-Sichten werden nur die erlaubten Knoten ausgegeben, z.B.

root-Hochschule (Auswahl)
fak-Fakultäten (Auswahl)
1-fak1 (Auswahl)
11-Institut 1
13 – Institut 3

wenn nichts ausgewählt wurde, root, fak, oder fak1 wird trotzdem nur 11,13 als subkeys ausgegeben, weil nur die selbst erlaubt sind.

Bei der foreach-Methode elements ist es anders: root, fak, fak1 werden auch mit durchlaufen, weil (Teil)summenzeilen interessant sein können

Knoten-Strukturattribut mit strukturInt,strukturStr

Ein Element im Baum kann ein Strikturattribut haben, dies beschreibt Art oder Struktur des Knotens. Beispiel: Das Feld "orgstruktur" im Organigramm beschreibt, ob ein Knoten eine Lehreinheit oder ein Fachbereich ist (20 bzw. 30). Diese Strukturinformation ist im Knoten hinterlegt, sofern beim Einlesen der Sicht an Postition 4 und/oder 5 etwas angegeben wurde (z.B. select name,key_apnr,parent,orgstruktur from organigramm).

Sie kann z.B. für if-Abfragen zur Aggregierung benutzt werden.

```
<#foreach eineInstitution in Institutionen>
```

nodeattrib - ein Knotenattribut

Das Attribut nodeattib steuert:

- wenn null oder 0: Knoten wird ganz normal angezeigt
- wenn 1, dann wird es in der Anzeige versteckt
- wenn 2, dann wird es angezeigt, ist aber nicht selektierbar im Baum
- wenn 3, dann ist es eingerückt bei Feldart 1

Beispiel für relations-SQL für Feldart 1

```
select key,name,0 as nodeattrib from tabelle
```

getSubKeys

Eine besonderer Trick ist, wenn man einen bestimmten Knoten aus dem Baum braucht, der nicht ausgewählt sein muss: Man nimmt den Feldnamen der Sicht und schreibt z.B.

```
Kostenarten("getSubkeys","21")
```

Dann erhält man eine Schlüsselliste für alle Schlüssel 21 und untergeordnete.

Mehrfachauswahl bei Sicht-Feldern mit Schleifenfunktion

Ab Kern4.5 kann man eine Mehrfachauswahl bei Sichtfeldern aktivieren, die als Schleife für die Ergebnisdarstellung benutzt werden.

Angenommen wir haben folgenden Finanzstellenbaum

- Hochschule
- A1
- A11
- A12
- B2
- B21
- B22

Bisher gab es ein Problem wenn ein User bei der Mehrfachauswahl A11 und B2 auswählte, weil dann die Ebenendarstellung

durcheinander kam, die Ergebniszeile für A11 wurde gar nicht angezeigt, weil Ebene 2 kleiner als Ebene 1 der zweiten Selektion B2.

Dies kann man jetzt umgehen, in dem man das Feld Finanzstelle obligatorisch macht, es muss also immer eine Selektion geben und dann für jede Finanzstelle nicht *level* benutzt, sondern *levelFromSelection+1*

```
<#foreach eineFistl in Finanzstelle.elements>
```

Eine Summenberechnung bei mehr als einer Zeile geht dann so

```
 <#if Finanzstelle.selectionCount
```

Ein Beispiel ist die Maske GXSTAGE Budget nach Finanzstellen 33060.

has_content

Wenn man wissen möchte, ob eine Variable mit Inhalt gefüllt ist, kann man dies mit has_content Abfragen, z.B.

```
<#if lehr_abg?has_content <#if lehr_abg?has_content >
```

ForEach

FreeMarker kann nicht nur primitive Datentypen wie Strings oder Zahlen verarbeiten, sondern auch Collections. Wenn im Datenmodell eine Collection hinterlegt ist, kann man forEach benutzten.

Das sieht ungefähr so aus:

```
<#foreach eineKostenart in Kostenarten.elements<#foreach eineKostenart in Kostenarten.elements>
-- Auswertung für ${eineKostenart}
```

Details siehe bei Sichtfeldern-Schleifen.

For-Next-Schleifen: List

FreeMarker kann auch eine For-Next-Schleife mit 1er-Schritten erzeugen:

```
create temp table tmp_aggre
(struktur char(50),text char(200), ch30_fach char(3),sortnr int,
<#list 0..30 as i<#list 0..30 as i>
m_a${18+(i*2)} decimal(7,2),
w_a${18+(i*2)} decimal(7,2),



gesamt decimal(7,2));
```

Nach der Freemarker-Transformation:

```
create temp table tmp_aggre
(struktur char(50),text char(200), ch30_fach char(3),sortnr int,
m_a18 decimal(7,2),
w_a18 decimal(7,2),
m_a20 decimal(7,2),
w_a20 decimal(7,2),
m_a22 decimal(7,2),
w_a22 decimal(7,2),
[…]
m_a78 decimal(7,2),
w_a78 decimal(7,2),
gesamt decimal(7,2));
```

Freemarker in dynamischen Felderrelationen

Damit ein Feld Freemarker Variablen auslesen kann wie Kostenstelle, muss es als dynamisch markiert sein. SuperX erkennt ein Feld als dynamisch wenn << darin vorkommt:< darin vorkommt:

```
<>

--Freemarker  Template
<#include  "SQL_lingua_franca"/>

<#include  "SuperX_general"/>\

select  distinct  buchungsab_fb,trim(buchungsab_fb)||'-'||max(ba_name)  from  fin_used_inst\
where  kostenstelle  in  <@printkeys  Kostenstelle.allNeededKeysList  />
```

klappt nicht, aber

```
<>

--Freemarker  Template
<#include  "SQL_lingua_franca"/>

<#include  "SuperX_general"/>\

select  distinct  buchungsab_fb,trim(buchungsab_fb)||'-'||max(ba_name)  from  fin_used_inst\
where  kostenstelle  in  <@printkeys  Kostenstelle.allNeededKeysList  />    /*    --quatsch  <>  */
```

Makros und Funktionen

Es könnten auch Makros und Funktionen (die Werte zurückliefern) definiert werden.

```
<#macro  macroname  param1  param2<#macro  macroname  param1  param2>
```

Aufgerufen werden sie mit

```
<@makroname  />
```

bzw.

```
<@makroname  param1  param2  ../<@makroname  param1  param2  ../>
```

oder

```
<@makroname  param1=wert1  param2=wert2  ../<@makroname  param1=wert1  param2=wert2  ../>
```

Die Reihenfolge in der Makros definiert werden spielt keine Rolle.

Einige Makros für Datenunabhängigkeit (SQL Lingua franca) und allgemeine Makros sind in der Tabelle fm_templates hinterlegt.

Diese können mit
<#include "xx"/> eingebunden werden, wobei "xx" durch die jew. fm_templates.id ersetzt wird.

Exportdateinamen

Ab Kern5.2 /HISinOne-BI 2025.06 kann man per Freemarker sqlvar bestimmen, wie ein Exportdateiname beim Export nach Excel/PDF heißen soll, wenn es nicht der Maskenname bzw. der Reportname bei JasperReports sein soll.

Dazu muss man im select_stmt der maske eine sqlvar namens "exportfilename" definieren. Dieser kann mit allem arbeiten, was SQL zu bieten hat und bei Bedarf auch auf vorherige sqlvars zugreifen.

Beispiel:

```
<sqlvars>
```

Der resultierende Dateiname könnte damit z.B. sein "Bilanz-2024-1241.xlsx"