Inhaltsverzeichnis

Kernmodul Shellscripte

SuperX-spezifische Scripte: Übersicht

Für die Administration des DataWarehouse sind Shellscripte vorbereitet, die flexible Werkzeuge zur Datenbankadministration bereitstellen. Die Shellscripte werden in den Update-Scripten aufgerufen, können aber auch zur manuellen Administration benutzt werden. Die wichtigsten Bereiche sind die Masken-Verwaltung und die Ladescripte im Umgang mit Tabellen sowie allgemeine Scripte.

Alle Scripte befinden sich unter $SUPERX_DIR/db/bin, deshalb muss dieser Pfad in der Umgebungsvariable PATH enthalten sein. Die Scripte wurden unter UNIX entwickelt (ohne Endung oder Endung .x), einige davon sind auch nach DOS portiert worden (erkennbar an der Endung .bat).

Einige Scripte lauten "sx_auto_"..., dies bedeutet, dass die Scripte ohne Sicherheitsabfrage ausgeführt werden.

Voraussetzung für den Ablauf der Scripte ist die Eintragung der korrekten Umgebungsvariablen in $SUPERX_DIR/db/bin/SQL_ENV bzw. $SUPERX_DIR\db\bin\sql_env.bat. Wenn der Client jdbc verwendet wird, muss ausserdem die korrekte DB_PROPERTIES gesetzt sein.

Die Umgebungssteuerung SQL_ENV

Das Script $SUPERX_DIR/db/bin/SQL_ENV steuert die Umgebung und ist für den Betrieb der Scripte unverzichtbar. Einige Variablen sind vorbelegt, Beispiele sind auf Kommentar gesetzt. Da die Umgebung von dem System abhängt, muss jeder Anwender die Werte manuell pflegen. Bei einem Update des SuperX-Kernmoduls wird diese Datei nicht überschrieben, lediglich sein SQL_ENV.sam im gleichen Verzeichnis. Von dort müssen relevante Änderungen dann in die "richtige" SQL_ENV manuell übernommen werden. Informix- und Postgres-spezifische Variablen sind in dem Kapitel zur Installation und *Konfiguration* der Datenbankserver beschrieben.

Folgende Variablen sind auf jeden Fall zubelegen:

Variable	Erläuterung
SUPERX_DIR	Der Installationspfad von SuperX
DATABASE	Das Datenbanksystem ("POSTGRES" / "INFORMIX")
DBNAME	Der Name der Datenbank (standardmäßig "superx").
SX_CLIENT	Die Clientanwendung (bei Postgres "psql", bei Informix "dbaccess"). Ein client namens "jdbc" ist generisch und dient dem Zugriff auf beliebige DB-Systeme, für die jdbc-Treiber existieren. Der jdbc-Client wurde bisher mit Informix, Postgres und hsqldb getestet – die jdbc-Treiber für Informix und Postgres werden mitgeliefert und dürfen auf keinen Fall durch andere ersetzt werden.
MAILPROG	Das Mailprogramm unter UNIX, z.B. mutt oder mailx; dies muss sich im PATH des users superx befinden.
LOGMAIL/ERRORMAIL	Die superx-weite Mailadresse, an die Logdateien von ETL-Scripten geschickt werden.
JAVA_HOME	Installationspfad der Java-Runtime. Das Unterverzeichnis bin muss in den PATH aufgenommen werden. Standardmäßig wird für Kernmodul 5.0 Java 8 oder besser Java 11 vorausgesetzt.
JAVA_OPTS	Java-Runtime-Optionen, z.B. RAM JAVA_OPTS="-Xmx200M -Djava.awt.headless=true"
CATALINA_OPTS	Tomcat-Runtime-Optionen, z.B. RAM JAVA_OPTS="-Xmx400M -Djava.awt.headless=true -DSuperX-DB-PROPERTIES-SET=true"
DB_PROPERTIES	Der Pfad zur *DB_PROPERTIES*, standardmäßig $SUPERX_DIR/webserver/tomcat/webapps/superx/WEB-INF/db.properties
MANDANTID	Mandantennummer (Hochschulnummer) bei mandantenfähigen Installationen
Die folgenden Umgebungsvariablen sind nur für den JDBC-Client sowie für Postgres relevant:	
LOGGING_PROPERTIES	Logging-Parameter für den jdbc-Client. Voreingestellt ist "WARNING", mehr Ausgaben erhält man mit "FINE"
PGHOST	Name des Postgres-Servers (für Postgres unter Windows)
Die folgenden Umgebungsvariablen werden wahrscheinlich nicht geändert (sollten sie auch nicht):	
DBDELIMITER	Standardmäßig "^"
PATH	Der PATH wird erweitert um das Verzeichnis ./:$SUPERX_DIR/db/bin
JDBC_CLASSPATH	Der Pfad zu den relevanten jdbc-Treibern und Hilfsprogrammen.
XML_CLASSPATH	Der Pfad zu den XML-Tools (Xalan, Xerces & co).
FM_DEBUG	Wenn FM_DEBUG = true gesetzt wird, werden bei Freemarker Scripten von DOSQL die *.tmp.sql-Dateien nicht gelöscht.

Die Datei sollte unter UNIX in jedem Aufruf der shell "gesourced" werden, z.B. durch den Befehl:

```
. ~/db/bin/SQL_ENV
```

(Leerzeichen zwischen Punkt und Tilde!) in der Datei ~/.bashrc.

Nutzung der SQL_ENV unter HISinOne-BI

Die SuperX Shellscripte lassen sich auch in der HISinOne-BI unter Linux nutzen. In der Distribution befindet sich eine Beispiel-SQL_ENV für die Nutzung in HISinOne:

```
webapps/superx/WEB-INF/conf/edustore/db/bin/SQL_ENV_his1.sam
```

Der Unterschied ist in der Verzeichnisstruktur: Unter SuperX gibt es den Ordner $SUPERX_DIR, der normalerweise ganz oben liegt, z.B. in /home/superx. Unter HISinOne liegt der Ordner unterhalb der Webanwendung, z.B. in

```
/var/lib/tomcat9/webapps/superx/WEB-INF/conf/edustore
```

Weiterhin muss man die Umgebungsvariable WEBAPP umsetzen, z.B.:

SuperX: /home/superx/webserver/tomcat/webapps/superx

HISinOne: /var/lib/tomcat9/webapps/superx

Achtung:
Welche Java-Version?

Ab HISinOne-BI 2023.12 wird in der Webanwendung / im Tomcat für den qisserver die JRE Java 17 vorausgesetzt. Die Shellscripte für SuperX laufen aber aus Gründen der Abwärtskompatibilität nur mit Java 11. Sie müssen also beide Runtimes installieren, und die folgende Konfiguration nutzen, hier z.B. unter Ubuntu Linux 2022.04:

```
apt-get  install  -y  openjdk-11-jdk
apt-get  install  -y  openjdk-17-jdk
```

Dann setzen Sie in der Datei

```
/etc/default/tomcat9
```

die Zeilen

```
JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
export  JAVA_HOME
```

und in der SQL_ENV die Zeilen

```
JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export  JAVA_HOME
```

Das Beispiel läßt sich leicht auf andere Distributionen übertragen.

Normalerweise wird für HISinOne-BI kein Zugriff via psql benötigt, daher setzen Sie die Variable

```
SX_CLIENT=jdbc
```

Außerdem muss die Variable JDBC_CLASSPATH angepaßt werden um den Eintrag $WEBAPP/WEB-INF/classes, weil es unter HISinOne-BI keine superx-jar mehr gibt:

```
JDBC_CLASSPATH="$WEBAPP/WEB-INF/lib_ext/servlet-api.jar:$WEBAPP/WEB-INF/classes";  for  i  in  `ls  $LIB_PATH/*.jar`  ;  do  JDBC_CLASSPATH=$JDBC_CLASSPATH:$i  ;  done  ;  XML_CLASSPATH=$JDBC_CLASSPATH
```

Die Datei muss nicht "SQL_ENV" heißen, Sie können auch das Spezialmodul ergänzen, z.B. "SQL_ENV_FHDO".

Datenbank- und SQL-Scripte

Propadmin

Der PropAdmin ist ein kleines Werkzeug, um den Zugriff auf jdbc-Datenbanken zu testen und die *Verbindungsparameter* in einer übergebenen properties-Datei zu sichern. Wenn keine graphische Umgebung eingerichtet ist, müssen Sie die alle Verbindungsparameter manuell in die db.properties eintragen. Nur das Passwort kann mit dem propadmin bearbeitet werden.

(Musterdateien für Postgres und Inofrmix liegen vor in

$SUPERX_DIR/webserver/tomcat/webapps/superx/WEB-INF/db-postgres.properties bzw. db-informix.properties). Wenn als weiterer Parameter kein Dateiname übergeben wird, dann wird die Umgebungsvariable DB_PROPERTIES ausgewertet.

Syntax UNIX	propadmin.x -nogui(optional)
Syntax DOS	propadmin.bat

Wenn die Default-Dateiencodierung der aktiven Locale für die Passwort-Verschlüsselung nicht ausreicht, wird eine Fehlermeldung ausgegeben. Unter Windows / DOS ist die Vorbelegung Cp1252 bei deutscher Codepage ausreichend, unter Unix wird die deutsche Locale benötigt.

DOSQL

Zum Absetzen beliebiger SQL-Kommandos werden die Befehle DOSQL und DOQUERY genutzt.

DOSQL kann SQL-Anweisung(en) in der der Datei (als Parameter) in der SuperX-Datenbank ausfuehren.

Syntax	DOSQL "Dateiname mit sql-Anweisung(en)" header (true,false)(optional) Ausgabedatei(optional)
Beispiel	DOSQL "/home/superx/db/isql/test.sql" false "^" output.txt

Das Ergebnis kann mit Feldüberschriften (header=true) in eine Datei Ausgabedatei ausgegeben werden.

Wenn

```
FM_DEBUG  =  true
export  FM_DEBUG
```

gesetzt wird, werden bei Freemarker Scripten von DOSQL die *.tmp.sql-Dateien nicht gelöscht.

DOQUERY

Shellvariablen setzen und eingegebene SQL-Anweisung (als Parameter) in der SuperX-Datenbank ausfuehren.

Syntax	DOQUERY "sql-Anweisung" header (true,false)(optional) Delimiter(optional) Ausgabedatei(optional)
Beispiel	DOQUERY "select name from userinfo" false "^" output.txt

Das Ergebnis kann mit Feldüberschriften (header=true) in eine Datei Ausgabedatei ausgegeben werden.

XML-Scripte

sx_transform.x

Transformiert eine xml-Datei mit einer übergebenen XSL-Datei und gibt das Ergebnis in einen Ausgabekanal aus (stdout oder Datei). Dabei wird der in SuperX integrierte XML-Parser Xerces und der XSL-Prozessor Saxon benutzt.

Syntax	sx_transform.x -IN:xml-Datei -XSL:xsl-Datei -OUT:Ausgabedatei -method:Ausgabeformat (text, xml,html,rtf,pdf) (optional) -params:Parameter(optional)
Beispiel	sx_transform.x -IN:myxml.xml -XSL:myxsl.xsl -OUT:output.htm -method:html

Als Parameter "method" kann ein spezielles Ausgabeformat gewählt werden, z.B. text (siehe Xalan-Doku). Bei rtf wird der RTF-Construktor Jfor aufgerufen, bei pdf wird FOP aufgerufen. Die *.fo-Datei wird nach tmp.fo geschrieben und dann nach pdf transformiert. Wir gehen also nicht davon aus, dass .fo-Dateien die Eingabequelle darstellen.

Stylesheet Parameter werden mit "," getrennt optional angefügt, z.B.

```
sx_transform.x  -IN:myxml.xml  -XSL:myxsl.xsl  -OUT:output.htm  -method:html    "-params:Jahr=2020,Monat=12"
```

saxon_transform.x

Wie sx_transform.x transformiert das Script XML via XSLT, hier werden allerdings Saxon-spezifische Parameter ermöglicht.

Syntax	saxon_transform.x xml-Datei xsl-Datei Ausgabeformat (text, xml,html) (optional) Ausgabedatei (optional) Parameter (optional)
Beispiel	sx_transform.x myxml.xml myxsl.xsl output.htm html

Bei pdf wird FOP aufgerufen. Die *.fo-Datei wird nach tmp.fo geschrieben und dann nach pdf transformiert.

Wenn ein XSL-Script mehrere Ausgabedateien erzeugen soll, kann dieses Script ggü. sx_transform.x Vorteile haben.

show_fop_fontlist.x

Zeigt die installierten Fonts für PDF Erzeugung mit XSL-FO an.

Das Script hat keine Argumente.

sx_validate.x

sx_validate.x validiert eine XML-Datei
- ob sie wohlgeform ist
- ob sie zu einem optional übergebenen Schema (XSD-Datei) valide ist.

```
sx_validate.x  XML-Datei  SCHEMA-Datei  (optional)
```

sx_xmlfileappender.x

Das Script *sx_xmlfileappender.x* faßt mehrere XML-Dateien in einem übergebenen Pfad zu eine rDatei mit einem neuen root-Knoten "artificalroot" zusammen.

Aufruf:

```
sx_xmlfileappender.x  Path  file(regex  oder  Dateiname  mehrere  mit  ,  getrennt)  Outputfile
```

Codierung in ISO und UTF-8

Ältere Systeme arbeiten in der Regel mit der Zeichencodierung ISO-8859-1 bis ISO-8859-9. Dieser Zeichensatz wird auch LATIN-1 genannt. Die UNIX-Locale de_DE@euro entspricht z.B. ISO-8859-9 und enthält das EUR-Symbol.

Aktuelle SuperX-Installationen arbeiten in der Regel mit der UTF-8 Codierung, dies wird bei der Installation gesetzt.

Mit dem Wechsel von ISO-Codierung zu UTF8 bleibt oft der Bedarf bestehen, Dateien hin- und herzucodieren. Sei es, weil beim Entladen aus einer entfernten Datenbank noch das ISO-System genutzt wird, oder bei der Migration eines Systems. Nach unserer Erfahrung sollten Umlaute in Dateinamen unbedingt vermieden werden.

SuperX bietet unter UNIX Shellscripte zur Erfassung und Änderung der Zeichencodierung (Verzeichnis $SUPERX_DIR/db/bin). Im Wesentlichen werden dabei die Unix-Kommandos file und recode genutzt, die Shellscripte machen den Umgang mit umfangreichen Dateilisten komfortabler. Bei der Verarbeitung von Dateilisten sollte man die Scripte sehr vorsichtig einsetzen, es finden keine Sicherheitsüberprüfungen statt.

Die Scripte wurden bisher nur unter Linux mit bash getestet, andere UNIXe wie BSD verhalten sich ggf. anders als erwartet. Daher bitte mit Vorsicht benutzen.

sx_show_encoding.x

Das Script zeigt die Encodierung einer Datei an.

Syntax	sx_show_encoding.x
Beispiel	sx_show_encoding.x $SUPERX_DIR/webserver/tomcat/webapps/superx/WEB-INF/web.xml
Ausgabe	/hopme/superx/webserver/tomcat/webapps/superx/WEB-INF/web.xml: XML ISO

Das Script nutzt verschiedene UNIX-Tools, je nach System kann die Ausgabe variieren. Bei XML-Dateien wird auch der Dateiinhalt (XML-Header) ausgewertet.

sx_recode_iso2utf.x

Das Script ändert die Encodierung einer Datei von ISO nach UTF-8:

Syntax	sx_recode_iso2utf.x

Syntax	sx_recode_iso2utf.x
Beispiel	sx_recode_iso2utf.x $SUPERX_DIR/webserver/tomcat/webapps/superx/WEB-INF/web.xml
Ausgabe	--keine--

Das Script nutzt das UNIX-Kommando recode. Darüberhinaus werden bei XML-Dateien auch die XML-Header "encoding=..." geändert, so wird z.B. aus

der Header

Andere Inhalte der Datei unterhalb der ersten Zeile werden keinesfalls geändert.

sx_recode_utf2iso.x

Das Script ändert die Encodierung einer Datei von ISO nach UTF-8:

Syntax	sx_recode_utf2iso.x
Beispiel	sx_recode_utf2iso.x $SUPERX_DIR/webserver/tomcat/webapps/superx/WEB-INF/web.xml
Ausgabe	--keine--

Das Script nutzt das UNIX-Kommando recode. Darüberhinaus werden bei XML-Dateien auch die XML-Header "encoding=..." geändert, so wird z.B. aus

der Header

Andere Inhalte der Datei unterhalb der ersten Zeile werden keinesfalls geändert.

sx_list_isofiles.x

Das Script listet alle ISO-Dateien im übergebenen Verzeichnis auf (inkl. Unterverzeichnisse).

Syntax	sx_list_isofiles.x
Beispiel	sx_list_isofiles.x webserver/tomcat/webapps/superx/WEB-INF
Ausgabe	webserver/tomcat/webapps/superx/WEB-INF/lib/LocalStrings_de.properties webserver/tomcat/webapps/superx/WEB-INF/lib/hierhin_den_informix_treiber_kopieren.txt [...] webserver/tomcat/webapps/superx/WEB-INF/db.properties

Die Ausgabe kann in eine Datei umgeleitet werden, welche wiederum für das Script *sx_recode_isofiles.x* als Eingabedatei genutzt werden.

sx_list_isofiles.x webserver/tomcat/webapps/superx/WEB-INF >iso.txt

sx_recode_isofiles.x

Das Script konvertiert alle Dateien in der übergebenen Dateiliste von ISO nach UTF-8:

Syntax	sx_recode_isofiles.x
Beispiel	sx_list_isofiles.x iso.txt
Ausgabe	--Keine--

Die Eingabedatei ist in der Regel die Ausgabe des Scriptes **'sx_list_isofiles.x'**.

sx_list_utf8files.x

Das Script listet alle UTF-8-Dateien im übergebenen Verzeichnis auf (inkl. Unterverzeichnisse).

Syntax	sx_list_utf8files.x
Beispiel	sx_list_utf8files.x webserver/tomcat/webapps/superx/WEB-INF
Ausgabe	webserver/tomcat/webapps/superx/WEB-INF/lib/LocalStrings_de.properties webserver/tomcat/webapps/superx/WEB-INF/lib/hierhin_den_informix_treiber_kopieren.txt [...] webserver/tomcat/webapps/superx/WEB-INF/db.properties

Die Ausgabe kann in eine Datei umgeleitet werden, welche wiederum für das Script *sx_recode_utf8files.x* als Eingabedatei genutzt werden.

sx_list_isofiles.x webserver/tomcat/webapps/superx/WEB-INF >utf.txt

sx_recode_utf8files.x

Das Script konvertiert alle UTF-8 Dateien in der übergebenen Dateiliste von UTF-8 nach ISO:

Syntax	sx_recode_utf8files.x
Beispiel	sx_recode_utf8files.x utf.txt
Ausgabe	--Keine--

Die Eingabedatei ist in der Regel die Ausgabe des Scriptes sx_list_utf8files.x.

Umgang mit Tabellen

In SuperX werden ständig Tabellen erstellt / geladen / entladen. Zu diesem Zweck wurden Shellscripte entwickelt.

sx_unload_table.x

Entlädt die Inhalte der Tabelle nach <(optional) oder <>.unl

Syntax	sx_unload_table.x < <>(optional)
Beispiel	sx_unload_table.x userinfo

sx_upload_table.x

Löscht die Inhalte der Tabelle <, und lädt die Inhalte einer Datei in die Tabelle mit *sx_upload_records*. Wenn kein Dateiname übergeben wurde, wird als Name <>.unl angenommen.

Syntax	sx_upload_table.x < <>(optional)
Beispiel	sx_upload_table.x userinfo

sx_upload_records.x

Lädt die Inhalte einer Datei in die Tabelle, ohne vorherige Inhalte zu löschen. Wenn kein Dateiname übergeben wurde, wird als Name <>.unl angenommen.

Syntax	sx_upload_records.x < <>(optional)
Beispiel	sx_upload_records.x userinfo

Bei Postgres als DB-System wird eine Java-Klasse (de.superx.bin.UnlFileConverter) aufgerufen, die die Unload-Datei entsprechend einer Spezifikation aufbereitet (siehe $SUPERX_DIR/db/conf/unldescr*).

Wenn der jdbc-Client benutzt wird, können umfangreiche Parameter übergeben werden (Import mit Spaltenüberschriften, Ausgabe von Fehlerprotokollen). Vergleichen Sie die Kommentare im Script.

sx_schema.x

Entlädt das Schema einer Tabelle in einem vorgegebenen Format.

Syntax	sx_schema.x Tabelle format (pg|ids|ansi|xml|HIS)) (optional) Ausgabedatei (optional)
Beispiel	sx_schema.x userinfo ids myschema.sql
Die Formate	Die Formate sind entweder sql-Scripte für die jeweiligen Datenbanktypen (Postgres, Informix, ANSI), die aus der Umgebungsvariable DATABASE ausgelesen werden, oder xml bzw. ein xml-Format in Anlehnung an die Datenbank-DTD der GX-Software der HIS e.G.

Modulverwaltung

Mit SuperX Version 2.1 wurde die Verwaltung der Module (Installieren / Aktualisieren / sichern und die zugehörogen Logdateien) in zentrale Shellscripte verlagert, die sich ebenfalls in $SUPERX_DIR/db/bin befinden. Die Shellscripte sind dabei nur die operativen "Hüllen" um die eigentlichen SQL-Scripte. Diese wiederum werden zum Teil "von Hand" erzeugt (um z.B. hochschulspezifische Erweiterungen oder Anpassungen vorzunehmen), und zum Teil automatisch aus einer zentralen Steuerdatei

$SUPERX_DIR/db/module/</conf/<>.xml

jeweils für Postgres und Informix erzeugt.

module_scripts_create.x

Das Script erzeugt die Installationsdateien für ein Modul, jeweils für Postgres und Informix, nach dem Schema

<_<_<Z.B. wird für das BAU-Modul aus der Datei $BAU_PFAD/conf/bau.xml das Script bau_load_pg.sql erzeugt, das die Rohdaten unter Postgres lädt, oder die Datei bau_trans_ids.sql für das Script, das die Bau-Tabellen unter Informix transformiert

Syntax	module_scripts_create.x < < < <>
Beispiel	module_scripts_create.x BAU $BAU_PFAD INFORMIX 1.0

Im Grunde handelt es sich um XML-Transformationen. Die Stylesheets für dieses Script befinden sich im Verzeichnis $SUPERX_DIR/db/conf, und die XML-Datei für das Module in $SUPERX_DIR/db/module/<>/conf. Wenn die Datei nicht gefunden wird, bricht das Script ab.

Die folgende Abbildung zeigt die Arbeitsweise:

Das Script module_scripts_create.x erzeugt eine Reihe von Scripten, die das Modul installieren / aktualisieren / deinstallieren. Außerdem werden html- bzw. rtf-Dokumentationen erzeugt sowie Administrationsformulare für **dbforms**.

module_install.x

Installiert ein Modul < in der Datenbank, wobei die Installationsdateien sich im <> befinden.

Syntax	module_install.x < <>
Beispiel	module_install.x BAU $BAU_PFAD

module_drop.x

Löscht die Komponenten eines Moduls < in der Datenbank, wobei die Installationsdateien sich im <> befinden.

Syntax	module_drop.x < <>
Beispiel	module_drop.x BAU $BAU_PFAD

Entladen

Das Entladescript lautet $SUPERX_DIR/db/module/</rohdaten/<_unload.x. Die Entladeparameter werden in folgender Datei festgelegt:

$SUPERX_DIR/db/module/</rohdaten/<>_ENV

Entladeroutine bei mandantenfähigen Installationen:

$SUPERX_DIR/db/module/</rohdaten</<>_ENV

Dokumentation zu den jew. Parametern finden Sie in den jeweiligen Administrationshandbüchern der Module. Meist kann man Start-Semester oder -Jahre für das Entladen festlegen. Immer muß man auch das Datenbank-Vorsystem festlegen (Hostname, Kennung etc) sowie, bei HIS-Systemen, die Versionsnummer.

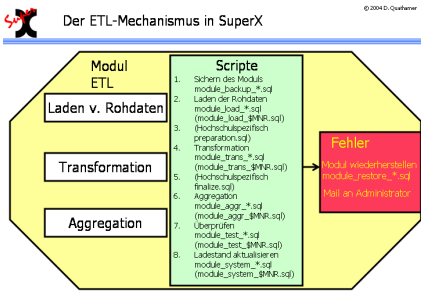
module_etl.x

Aktualisiert ein Modul < in der Datenbank, wobei die Installationsdateien sich im <> befinden.

Syntax	module_etl.x < <>
Beispiel	module_etl.x bau $BAU_PFAD

Die folgende Abbildung zeigt, wie die Komponenten zusammenhängen (klicken Sie auf die Grafik, um sie zu vergrößern):

Zunächst werden die Rohdaten geladen, die Daten vorbereitet, transformiert, und nachbereitet. Danach werden die Hilfstabellen erzeugt und, bei erfolgreichem durchlaufen, das Standdatum aktualisiert.



Bei Fehlern im ETL-Prozeß wird die Sicherung wiederhergestellt, und eine Mail an die Administrator_in verschickt. Außerdem werden die übernommenen Daten überprüft; wenn z.B. Schlüssel fehlen oder Rohdaten falsch zu sein scheinen, wird dies als Attachment an die Log- oder Fehlermail angehängt.

In der Praxis wird dieses Script nicht direkt von Cronjobs ausgeführt, sondern von einem Shellscript, das vorher die Umgebung einrichtet. Das folgende Beispiel zeigt das Update-Script für Bau unter Informix:

bau_update.x

```
#!/bin/sh
. /home/superx/db/bin/SQL_ENV
ERRORMAIL="bau-admin@hochschule.de"
export ERRORMAIL
LOGMAIL=$ERRORMAIL
export LOGMAIL
module_etl.x bau $BAU_PFAD >$BAU_ERRORDAT 2>&1
```

Wie Sie sehen ist es möglich, für jedes Modul unterschiedliche Mailadressen zuzuweisen. Die Mailadressen werden systemweit in der SQL_ENV] eingetragen.

Im allgemeinen ETL-Prozeß wird standardmäßig auch die Tabelle protokoll in einem festzulegendem Rhythmus (Konstante Löschung Protokoll (Tage)) gelöscht. Beim Vorgabewert 90 werden bei jeder ETL-Routine Einträge, die älter als 90 Tage sind, gelöscht.

Hochschulspezifische Transformationen im ETL-Prozess

Zusätzlich lassen sich im ETL-Prozeß hochschulspezifische Scripte ausführen (und überwachen). Dazu müssen Dateien mit einem gewissen Namensschema im Stammverzeichnis des Moduls vorhanden sein. Es gibt einen vereinfachten und einen erweiterten Modus für hochschulspezifische Transformationen.

- Einfacher Modus
  - Wenn im Modulpfad die Datei "preparation.sql" existiert, wird sie nach dem LOAD-Schritt ausgeführt.
  - Wenn im Modulpfad die Datei "finalize.sql" existiert, wird sie nach dem TRANS-Schritt ausgeführt.
- Erweiterter Modus / mandantenspezifische Scripte

- Wenn im Modulpfad Dateien nach dem Schema <_<_<>.sql exisitieren, werden diese jeweils nach dem "normalen" ETL-Schritt ausgeführt.

Wenn also z.B. die Datei

```
$COB_PFAD/cob_trans_70.sql
```

existiert und in der SQL_ENV die Umgebungsvariable $MANDANTID auf "70" steht, dann wird das Script nach dem normalen Trans-Schritt ausgeführt und nach

```
L_cob_trans_mandant_70.log
```

geloggt.

Der erweiterte Modus erlaubt die beliebige Anpassung eines Modus an eigene Bedürfnisse, z.B. Schlüsselumsetzung o.ä. Gleichzeitig erlaubt er einen echten mandentenfähigen Betrieb der ETL-Scripte.

Logging der Shellscripte

Hinweis: bei mandantenfähigen Installationen steht vor der Endung .log immer die MandantID.

Installation / Upgrade

Fürs Kernmodul lauten die Dateien bei der Installation:

```
$SUPERX_DIR/db/install/L_*.log
```

Beim Upgrade:

```
$SUPERX_DIR/db/install/upgrade/upgrade.log
```

Für alle anderen Komponenten:

```
$SUPERX_DIR/db/module/</L_<_<>.log
```

Laderoutinen

Für alle Module sind die Dateien wie folgt benannt:

Entladeroutine:

```
$SUPERX_DIR/db/module/</rohdaten/<>_unload.err
```

Entladeroutine bei mandantenfähigen Installationen:

```
$SUPERX_DIR/db/module/</rohdaten</<>_unload.err
```

Laderoutine:

```
$SUPERX_DIR/db/module/</L_<_<>.log
```

wobei <> wie folgt aufgebaut ist:

1. Unload (Entladen aus Vorsystem)
2. Load (CSV-Upload)
3. Transformation (Schlüsselharmonisierung, Prüfroutinen)
4. Aggregation (Aufbau der Hilfstabellen)
5. System (Ladedatum aktualisieren, Datenbank-Wartung)

Wenn die Laderoutine erfolgreich ist, werden alle Schritte hintereinander ausgeführt und geloggt. Wenn nicht, dann wird der jew. Schritt zuende geführt, und dann die Laderoutine gestoppt. Wenn also z.B. beim LOAD ein Fehler auftritt, dann wird der Schritt "Transformation" gar nicht erst begonnen. So ist sichergestellt daß die Auswertungen trotz Fehler laufen.

Debugging von Freemarker Scripten

Da die Laderoutinen oft mit Freemarker Scripten arbeiten, werden diverse SQL-Scripte nur zur Laufzeit generiert und ausgeführt, und danach wieder gelöscht. Um die Laderoutinen transparenter "beobachten" zu können, können Sie die die Löschung der Scripte mit dem Parameter

```
FM_DEBUG="true"
export  FM_DEBUG
```

Wenn FM_DEBUG = true gesetzt wird, werden bei Freemarker Scripten von DOSQL die *.tmp.sql-Dateien nicht gelöscht. Sie können den Parameter in der aktuellen Shell setzen, oder permanent in der SQL_ENV.

Kettle-Jobs starten

Da Kettle in der Webanwendung integriert ist, können Sie Kettle Jobs auch ohne die PDI-Standalone-Installation per Kommandozeile ausführen. Wenn die SQL_ENV eingerichtet und geladen ist, geben Sie z.B. einfach ein:

```
sx_kitchen.x  -file  "sospos_res1.kjb"  -norep
```

Die Parameter werden direkt an die Java Anwendung "kitchen" weitergereicht, d.h. weitere Dokumentation finden Sie in unserem Kettle-Kurs.

Installation und Wartung

Patch einspielen

Für SuperX können spezielle Patch-Dateien per Kommandozeile installiert werden. Vorbedingung ist, dass die bash und das Paket unzip installiert sind.

```
patch_apply.x
```

Das Script entpackt eine übergebene Patch-Datei (zip-File), und installiert die Dateien und DB-Operationen für die jeweils installierten und zum Patch passenden Module. Sie starten es direkt aus dem Patchordner, in dem der zu installierende Patch liegt:

```
patch_apply.x  -PatchFile-
```

Ein Beispiel:

```
patch_apply.x  patch_2011-06-01_superx_iso.zip
```

Dabei wird in dem Verzeichnis der Patch entpackt und ausgeführt. Je nach Patch muss auch ein Tomcat Neustart erfolgen, ein entsprechender Hinweis befindet sich in der Patch-Dokumentation.

Wenn ein Patch auch Dateien unterhalb von webapps/superx enthält, müssen Sie im Falle von Tomcat Clusterbetrieb den Patch auf alle Tomcat Hosts übertragen, entweder direkt oder mit git / rsync.

Weitere Hinweise zum Script finden Sie im Installationshandbuch.

Bei HISinOne-BI funktioniert das patch-apply Script nicht, der Patch mit dem Kürzel "webapps" im Dateinamen wird einfach in webapps/superx entpackt, und in der Regel sollten die jew. Module geupgradet werden. Weitere Hinweise finden Sie in der jew. Patch-Readme.

JasperReports kompilieren

Wenn Sie mit Unterberichten arbeiten, müssen Sie kompilierte .jasper-Dateien einbinden und ausliefern. Da dieser Dateityp normalerweise (und richtigerweise) im ".gitignore" ist, d.h. nicht über git ausgeliefert werden kann, können Sie die Dateien auch direkt auf dem Applikationsserver mit der exakt passenden JR-Version erzeugen. Typischerweise führen Sie das Script im Ordner WEB-INF/reports aus. Der Aufruf:

```
cd  $WEBAPP/WEB-INF/reports
java  $JAVA_OPTS  -cp  "$JDBC_CLASSPATH"  de.superx.bin.SxJasperCompiler
```

Der Workflow ist wie folgt:

- Die Quelldatei.jrxml ist die XML-Beschreibungsdatei des Berichts
- Der compile erzeugt aus der Quelldatei.jrxml eine Datei Quelldatei.jasper im gleichen Verzeichnis
- Die Quelldatei.jasper ist kompilierte Beschreibungsdatei für den Bericht, und wird in Unterberichten und Booklets genutzt. Weitere Details siehe unser JR-Handbuch.

LDAP User deaktivieren

Ausschließlich für SuperX-Standalone-Anwender können wir im System eine Gültigkeit von UserAccounts und automatisiertes Sperren von LDAP-Usern mit Sperr-Merkmal einbauen.

Die Tabelle userinfo und das entsprechende Formular zum Userbearbeiten hat die Merkmale Beginn Gültigkeit (gueltig_von) und Ende Gültigkeit (gueltig_bis), so dass SuperX-Standalone Hochschulen z.B. für befristete Mitarbeiter_innen in der Userverwaltung direkt eintragen können, dass ein Mitarbeiter sich z.B. nach dem 31.12.2024 nicht mehr anmelden können soll.

Wenn die Hochschule eine Passwortkontrolle via LDAP macht, kann diese per Script User mit einem entsprechenden Merkmal wie userLocked=true automatisiert zu sperren.

Hierfür setzt das Script den Eintrag userinfo.maxVersuch (also die maximal erlaubten Anmeldeversuche) auf 0 gesetzt, als "Ende Gültigkeit" wird das Vortagsdatum eingetragen sowie die Spalte *password_sha* (verschlüsseltes SuperX-Passwort) geleert.

Aufruf:

```
sx_ldap_lockout.x  Pfad/zu/db.properties  pfad/zu/superx_standalone_ldap.properties
```

Das Script kann nächtlich ausgeführt werden.

Masken-Verwaltung

Die Masken-Verwaltung ist detailliert im Entwicklerhandbuch SuperX beschrieben. Hier nur ein paar Hinweise zur Verwaltung der Masken. Zum Erzeugen und Verändern von Masken gibt es unter UNIX eine Kommandoschnittstelle, die auf dem Gebrauch folgender Skripte beruht. Die Skripte stehen unter dem Verzeichnis

```
$SUPERX_DIR/db/bin
```

und erzeugen oder verwenden Dateien in dem gegenwärtigen Arbeitsverzeichnis. Nach dem Einspielen der Datenbank sollten Sie darauf achten, den Dateien Ausführungsberechtigung (chmod 750 *) zu geben.

Eine Maske suchen

Wenn Sie eine Maske suchen, sollten die die Felder tid oder name in der Tabelle maskeninfo durchsuchen. Das folgende Script macht dies automatisch:

sx_search_mask

Aufruf:	sx_search_mask
Aktion:	sx_search_mask sucht die Masken, deren Name enthält
Ausgabe: .	tid, name der gefundenen Masken

Eine Maske sichern und entladen

Um eine Maske zu sichern, müssen Sie die entsprechenden Einträge in den Tabellen

1. felderinfo,
2. masken_felder_bez,
3. maskeninfo,
4. sachgeb_maske_bez,
5. maske_system_bez

selektieren und sichern. Für dies gibt es das Script sx_select_mask.

sx_select_mask

Aufruf:	sx_select_mask
Aktion:	sx_select_mask entlädt alle Metadaten aus den Tabellen maskeninfo, felderinfo, masken_felder_bez, sachgeb_maske_bez, maske_system_bez zur Maske mit tid = .

Ausgabe:	Fünf Dateien: 1. _felderinfo.unl, 1. _masken_felder_bez.unl, 1. _maskeninfo.unl, 1. _sachgeb_maske_bez.unl, 1. _maske_system_bez.unl

Eine Maske neu einfügen

Um eine Maske neu einzufügen, müssen Sie die entsprechenden Einträge in den Tabellen

1. felderinfo,
2. masken_felder_bez,
3. maskeninfo,
4. sachgeb_maske_bez,
5. maske_system_bez

einfügen. Dafür gibt es das Script sx_insert_mask.

sx_insert_mask

Aufruf:	sx_insert_mask []
Aktion:	sx_insert_mask lädt den Inhalt der fünf Dateien 1. _felderinfo.unl, 2. _masken_felder_bez.unl, 3. _maskeninfo.unl, 4. _sachgeb_maske_bez.unl, 5. _maske_system_bez.unl in die jeweiligen Tabellen der SuperX-Datenbank. Mit "j" wird die Sicherheitsabfrage umgangen.

Falls

Falls

select max(tid) from maskeninfo;

!	Um den Austausch von Abfragen innerhalb der Hochschulen zu erleichtern ("Abfragen-Pooling" über die SuperX-Website), sollten die Masken immer im Nummernkreis xxxx0000 bis xxxx9990 liegen, wobei xxxx der von der HIS verwandten Hochschulnummer entspricht. Die Zehnerschritte ergeben sich daraus, dass die dazwischen liegenden Nummern für die Maskenfelder (Tabelle felderinfo) reserviert sind Aus historischen Gründen liegen die Nummern aus Karlsruhe im Bereich 0-9990, aus Duisburg im Bereich 10000-19990. .

Wie im Abschnitt Userverwaltung beschrieben, kann die neue Maske Benutzern oder Gruppen zugänglich gemacht werden.

Eine Maske löschen

Um eine Maske zu löschen, müssen Sie die Einträge in den oben genannten Tabellen entfernen. Dafür gibt es das Script sx_delete_mask

sx_delete_mask

Aufruf:	sx_delete_mask
Aktion:	sx_delete_mask löscht alle Metadaten aus den Tabellen maskeninfo, felderinfo, masken_felder_bez, sachgeb_maske_bez und maske_system_bez zur Maske mit tid = .

Änderungen an einer Maske vornehmen

1. Selektieren der Metadaten der betreffenden Maske: sx_select_mask

2. Editieren der fünf Metadaten-Dateien ,,_..."

3. Abspeichern der neuen Metadaten: sx_insert_mask

Masken per Kommandozeile

In diversen Szenarien kann es sinnvoll sein, Masken nicht nur im Browser über die Webanwendung auszuführen, sondern über Kommandozeile, also ohne laufenden Tomcat:
- Für Entwicklungszwecke kann es praktisch sein, wenn man keinen laufenden Tomcat braucht, und Masken z.B. in Eclipse ausführt und debuggen kann
- Große Ergebnisdateien lassen sich leichter erzeugen, weil man von der Webanwendung unabhängig ist und z.B. dem Kommandozeilenaufruf mehr Arbeitsspeicher zuteilt.
- Für die Verteilung von Downloads kann es sinnvoll sein, Berichtsergebnisse in Dateiform zu generieren und im Download-Bereich zu verlinken.

Seit dem Kernmodul 4.2.1 bzw. HISinOne-BI 5.1 ist diese Funktionalität vorhanden, wenn die Kommandozeile eingerichtet ist:

- Unter SuperX: SQL_ENV
- Unter HISinOne-BI: Nutzung der SQL_ENV unter HISinOne-BI, oder Einrichtung einer HISinOne-Umgebung unter Eclipse: https://wiki.his.de/mediawiki/index.php/Einrichtung_einer_HISinOne-Arbeitsumgebung

Damit der Aufruf klappt, muss man in die Umgebungsvariable JDBC_CLASSPATH noch die Tomcat Library servlet-api.jar aufnehmen, diese liegt in ..webapps/superx/lib_ext. Der Aufruf muss bis Kernmodul 5.0 oder HISinOne-BI 2022.12 in dem Verzeichnis $WEBAPP/WEB-INF ausgeführt werden.

Der Kommandozeilenaufruf sieht unter Linux wie folgt aus:

```
java -cp "$JDBC_CLASSPATH" $JAVA_OPTS de.superx.bin.ExecuteMask -tid:Maskennummer -out:usgabedatei -user:Benutzerkennung "-params:Parameter..." -logger:$SUPERX_DIR/db/conf/logging.properties
```

Hier ein Beispiel für den Aufruf der Maske "Studierende und Studienanfänger (Zeitreihe)" (tid=16000) als Admin-User und deren Ausgabe in die Datei "test.htm":

```
java -cp "$JDBC_CLASSPATH" $JAVA_OPTS de.superx.bin.ExecuteMask -tid:16000 -out:test.htm -user:admin "-params:Köpfe oder Fälle ?=1=1&Stichtag=1" -logger:$SUPERX_DIR/db/conf/logging.properties
```

Das Beispiel läßt sich fortführen für andere Ausgabeformate, hier z.B. für PDF:

```
java -cp "$JDBC_CLASSPATH" $JAVA_OPTS de.superx.bin.ExecuteMask -tid:16000 -out:test.pdf -user:admin "-params:Köpfe oder Fälle ?=1=1&Stichtag=1&stylesheet=tabelle_fo_pdf.xsl&contenttype=application/pdf" -logger:$SUPERX_
```

Das Beispiel zeigt daß über das Params-Argument beliebige Ausgabeformate übergeben werden können, z.B. auch Excel (xlsx):

```
java -cp "$JDBC_CLASSPATH" $JAVA_OPTS de.superx.bin.ExecuteMask -tid:16000 -out:test.xlsx -user:admin "-params:Köpfe oder Fälle ?=1=1&Stichtag=1&stylesheet=tabelle_xls.xsl&contenttype=application/vnd.openxmlformats-office
```

oder als CSV Export:

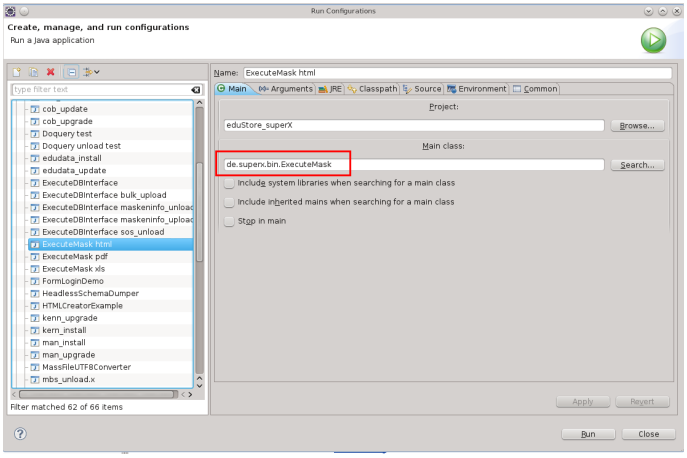
```
java -cp "$JDBC_CLASSPATH" $JAVA_OPTS de.superx.bin.ExecuteMask -tid:16000 -out:test.csv -user:admin "-params:Köpfe oder Fälle ?=1=1&Stichtag=1&contenttype=text/csv" -logger:$SUPERX_DIR/db/conf/logging.properties
```

Über den "Deep-Link"-Button lassen sich beliebige Parameter-Zeichenketten erzeugen und nutzen. Auch Aufrufe von JasperReports sind damit möglich, z.B. ein Studierendenbericht als Excel-Report:

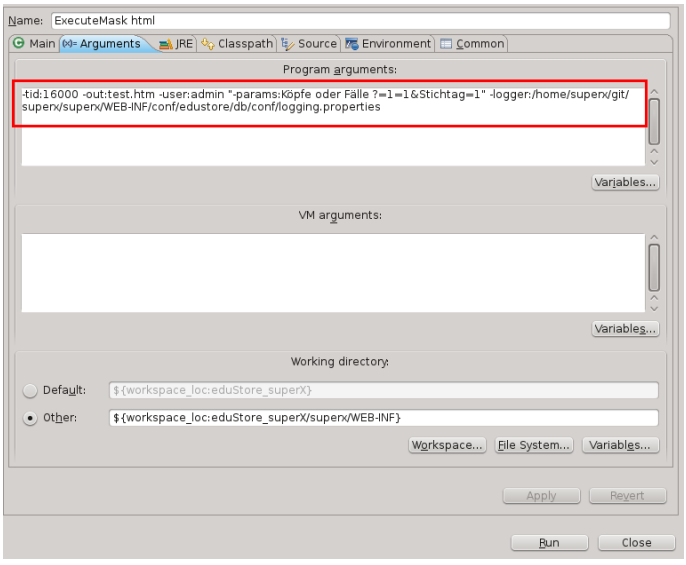
```
java -cp $JDBC_CLASSPATH $JAVA_OPTS -Xmx2048m de.superx.bin.ExecuteMask -tid:160440 -out:$OUTFILE "-params:maxoffset=1000000&stylesheet=tabelle_160440_semester_geschlecht.jrxml&tablestylesheet=tabelle_160440_sem
```

Bei Datenblattberichten sollte auch der Parameter tablestylesheet übergeben werden.

Unter Eclipse sieht der Aufruf für das erste Beispiel (Ausgabe nach html) so aus:



Und hier die Parameter:



Das Beispiel lässt sich leicht auf andere Plattformen (DOS, Netbeans) übertragen.

Sie können auch Parameter bewußt leer übergeben, indem Sie den Parameter ohne Wert übergeben, hier z.B. den Parameter „Sprache“:

```
java -cp "$JDBC_CLASSPATH" $JAVA_OPTS de.superx.bin.ExecuteMask -tid:70170 -out:test.htm -user:admin "-params:Stichwort=REPORT&Sprache=" -logger:$SUPERX_DIR/db/conf/logging.properties
```

Ausführen von JasperReports

Neben der Ausführung im Browser gibt es eine "Kommandozeilenversion" des Aufrufs: sx_jasper.x

Aufruf:	sx_jasper.x -JRXML: -XML: -JRPRINT: -OUT:
Aktion:	sx_jasper.x führt einen JasperReports-Task aus. Die Datenquelle kann entweder xml sein (Parameter -XML), oder eine Datenbankverbindung in der Datei db.properties. Das Ergebnis wird in eine Datei ausgegeben. Wenn keine Ausgabedatei angegeben wird, wird der jrxml-Dateiname verwendet, und eine PDF-Ausgabe erzeugt.

Sie können es mit einer Maskenausführung kombinieren, oder zum Entwickeln. Bei der Maskenausführung können Sie aber im Script ExecuteMask direkt JasperReports aufrufen.

Mailversand

Für den Mailversand empfehlen wir die Unix Anwendung s-nail. Installation unter Ubuntu mit

```
apt-get install -y s-nail
```

Unter RedHat

```
yum install s-nail
```

Einrichtung .mailrc

Das s-nail in Ubuntu 22.04 (Version 14.9.23) bietet die Möglichkeit, in der Shell mails für verschiedene Accounts zu verschicken. Zur Konfiguration wird die Datei $HOME/.mailrc genutzt.

alt:

```
set smtp="smtp://smtp.variomedia.de:587"
set from="mail@memtext.de"
set smtp-auth=login
set smtp-auth-user="noreply@superx-projekt.de"
set smtp-auth-password=securepw1234
```

neu:

```
set v15-compat
account test {
```

Mail verschicken über die bash

Zum Verschicken einer Testmail schreiben Sie z.B.

```
echo "test" | s-nail --
account=test -s "test"
```

Das CLI von s-nail verhält sich etwas UNIX-untypisch: Beachten Sie dass mehrere Mailadressat_innen mit Leerzeichen getrennt hinten angefügt werden - ohne Anführungszeichen um alle Mailadressen.

SQL_ENV mit s-nail

In der SQL_ENV für die neue Konfig-Syntax am besten die Variable MAILPROG mit z.B.

```
MAILPROG="s-nail --account=test"
```

angeben.