alphacarina
professional software architectures

SqlSurfer

Version 0.96.2

www.sqlsurfer.de

AlphaCarina Software GmbH
Auweg 13
94541 Grattersdorf

Inhaltsverzeichnis

Markenrechtlicher Hinweis

Die in diesem Dokument wiedergegebenen Firmen-, Marken- und Warenzeichen können auch ohne besondere Kennzeichnung geschützte Namen oder Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

Sämtliche in diesem Dokument abgedruckten Bildschirmabzüge unterliegen dem Urheberrecht © des jeweiligen Herstellers.

Microsoft, Excel, Powerpoint, Word und Windows sind Marken oder eingetragene Marken der Microsoft Corporation, Redmond (USA).

Oracle und MySql sind Marken oder eingetragene Marken der Oracle Corporation, Redwood Shores (USA).

Sybase ist eine Marke oder eingetragene Marke der Sybase Inc., Dublin (USA).

Alle anderen Namen von Produkten und Dienstleistungen sind Marken der jeweiligen Firmen.

Vorwort

Dieses Dokument richtet sich an die Anwender des SqlSurfers.

Bei den Anweisungen in diesem Dokument wird davon ausgegangen, dass der Anwender der Software sowohl mit den Grundlagen der Bedienung von Microsoft Windows und einer Betriebssystem-Shell als auch mit der Datenbankabfragesprache SQL vertraut ist.

Das vorliegende Dokument besteht aus den vier Abschnitten Schnelleinstieg, Installation, Benutzerhandbuch und Referenzteil. Es wurde so konzipiert, dass die einzelnen Abschnitte unabhängig voneinander gelesen werden können.
Das bedeutet, dass der Anwender nach dem Durcharbeiten des Abschnitts Schnelleinstieg bereits in der Lage ist, den SqlSurfer zur Erstellung eines Reports mit einfachen Anforderungen zu verwenden.
Der zweite Abschnitt beinhaltet Hinweise zur Installation des SqlSurfers sowie zu den Anforderungen an Hardware und Software.
Der dritte Abschnitt bildet den Hauptteil des Dokuments. Im Benutzerhandbuch werden der Aufbau und sämtliche Funktionen des SqlSurfers detailliert dargestellt. Zwangsläufig enthält dieser Abschnitt Wiederholungen aus dem Schnelleinstieg.
Der Referenzteil, der letzte Abschnitt dieses Dokuments, dient der Vervollständigung der Ausführungen der vorausgehenden Abschnitte.

Bei den aufgeführten Code-Beispielen im Benutzerhandbuch handelt es sich in den meisten Fällen lediglich um Ausschnitte aus einem SqlJob bzw. aus den Preferences. Es werden meist nur diejenigen Tags dargestellt, die für das gerade behandelte Thema bzw. die zu erklärende Funktion im jeweiligen Kapitel maßgeblich sind. Um die vollständige Syntax eines SqlJobs bzw. der Preferences darzustellen, dienen die Beispiele im Schnelleinstieg.

1 Schnelleinstieg

Im ersten Kapitel soll dem Anwender ein Schnelleinstieg in die Funktionsweise des SqlSurfers vermittelt werden. Es wird kurz beschrieben, was der SqlSurfer ist und was seine wichtigsten Funktionen sind. Anschließend wird für ein Minimalbeispiel die Syntax der Preferences und des SqlJobs sowie das resultierende Excel-Sheet dargestellt.

1.1 Was ist der SqlSurfer

Der SqlSurfer ist ein Programm, das automatisiert verschiedene Arbeitsschritte (z.B. SQL-Befehle, Programmausführungen oder Scriptverarbeitungen) ausführt um Daten aus Datenbanken zu extrahieren, importieren oder zu übertragen. Ggf werden die Ergebnisse in ein Microsoft Excel Workbook oder eine andere Datenbank übertragen.
Dabei benötigt der SqlSurfer immer zwei getrennte XML-Dateien: die Preferences und den SqlJob. In den Preferences werden Einstellungen u.a. zur Datenbankverbindung vorgenommen. Der SqlJob beinhaltet u.a. die auszuführenden Befehle und Informationen zum Export der Ergebnisse.
Die Bearbeitung und das Ausführen von SqlJobs können über einen Shell-Aufruf oder mit Hilfe einer grafischen Benutzeroberfläche (GUI) erfolgen. Die Ergebnisse werden in Microsoft Excel Worksheets an der gewünschten Stelle mit beliebigen Formatierungen eingefügt oder direkt in festgelegte Tabellen verschiedener Datenbanken exportiert.
Mit dem SqlSurfer können verschiedene Aufgaben erledigt: Datenbanken auslesen, Daten importieren, aufbereiten, kopieren, manipulieren und exportieren, Scripts starten und Berechnungen durchführen sowie Reports erstellen.

1.2 Minimalbeispiel helloWorld

Im Folgenden werden die wesentlichen Komponenten eines Minimalbeispiels dargestellt, bei dem ein einfaches SQL-Statement an die Datenbank Derby geschickt und das Ergebnis in ein Excel-Sheet exportiert wird.

Die Datei preferences.xml im Minimalbeispiel sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Preferences >
   <SqlConnection
      driver="org.apache.derby.jdbc.EmbeddedDriver"
      url="jdbc:derby:db;create=true"/>
</Preferences>
```

Die wichtigste Einstellung in den Preferences ist die Datenbankverbindung (SqlConnection). Dabei werden die Java-Klasse, die den notwendigen JDBC-Treiber enthält, sowie die Verbindungs-URL für die Datenbank angegeben.

Der SqlJob helloWorld.sqj hat folgende Syntax:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <SqlJob
    <Command
      cmd="select 'Hello World!' as testcolumn from SYSIBM.SYSDUMMY1">
      <ExportData outputRef="excel1"/>
```

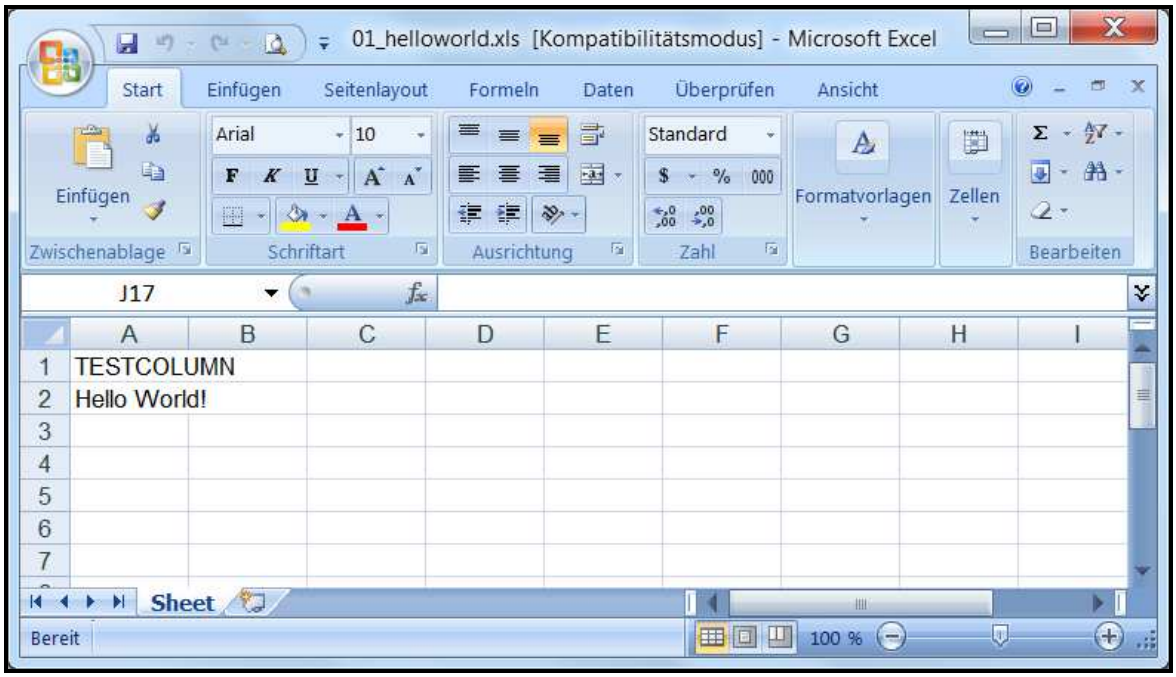
```
    </Command>
    <Output id="excel1" mode="poi" outFile="01_helloworld.xls"/>
  </SqlJob>
```

Den Kern des SqlJobs bildet ein SQL-Statement im Kommando (Command). Dieses besitzt eine Referenz auf die Output-Anweisung „excel1“, die festlegt, dass das Ergebnis über den Befüllungsmodus „poi“ in das Excel-Workbook „01_helloworld.xls“ exportiert wird.

Ausgeführt wird der Job mit

sq run helloworld.sqj

Das Ergebnis des SqlJobs helloworld.sqj zeigt folgender Screenshot aus Microsoft Excel:



2 Installation

In diesem Kapitel werden die Systemanforderungen an Hardware und Software, sowie die empfohlene Vorgehensweise bei der Installation des SqlSurfers dargestellt.

2.1 Systemanforderungen

Um den SqlSurfer verwenden zu können, müssen folgende Minimal-Anforderungen an die Hardware und Software erfüllt sein:

Anforderungen	
Hardware	**Software**
▪ Intel Pentium® oder AMD Athlon® - 600 MHz oder mehr ▪ 128 MB RAM (für SqlSurfer) ▪ Ca. 170 MB (für SqlSurfer)	▪ Microsoft Office (Excel, Word Powerpoint) ▪ Sun Java 1.6 (optional)

2.2 Betriebssysteme

Anforderungen
Unterstütze Betriebssysteme
▪ Microsoft Windows 2000/XP/Vista/7/8 ▪ Linux (Ubuntu, Red Hat u.a.) ▪ Solaris ▪ Prinzipiell alle BS auf den Java verfügbar ist.

2.3 Vorgehensweise

Bei der Installation des SqlSurfers sollte folgendermaßen vorgegangen werden:

a) Prüfen, ob Java 1.6 installiert ist und ggf. installieren (kostenloser Download von Java unter http://www.java.com/de/download/manual.jsp)
b) Herunterladen des SqlSurfers (unter http://www.sqlsurfer.de/ inzwischen nur alte Version)
c) Entpacken der heruntergeladenen Zip-Datei in ein beliebiges Installationsverzeichnis
d) Eintragen des Installationsverzeichnisses in die PATH-Umgebungsvariable (unter Systemsteuerung -> System -> Erweiterte Systemeinstellungen -> Registerkarte Erweitert -> Umgebungsvariablen)
e) Optional kann unter Windows mit Administrations-Rechten in der gestarteten Applikation (sq.exe) mit Optionen->Installationseinstellungen … eine Verknüpfung für die Dateierweiterung *.sqj ein Icon auf dem Desktop und ein Eintrag im Startmenü erstellt bzw. gelöscht werden.
f) Möchte man Excelsheet mit VBA Code befüllen, oder den Zoom oder zoomfast Befüller verwenden, muss man den Zugriff auf das VBA-Projektobjektmodell einschalten (siehe Kapitel 4.8.1)

2.4 Bemerkungen

Es wird eine java Version mit installiert. Löscht man das Unterverzeichnis ./jre aus dem Installationsverzeichnis wird eine Java VM aus dem System verwendet.

In der Datei sq.exe.vmoptions können Parameter der Java VM einstellt werden. Damit kann z.B. der Initial und maximale Speicherbedarf der VM unter Windows einstellt werden:

```
-Xms500m
-Xmx700m
```

3 Benutzerhandbuch

Das Benutzerhandbuch stellt den Hauptteil dieses Dokuments dar. Darin werden sämtliche Funktionen des SqlSurfers erläutert und mit Beispielen verdeutlicht.

3.1 Executive Summary / Einleitung

Der SqlSurfer ist ein Programm, das automatisiert SQL-Befehle oder andere Anweisungen ausführt und die Ergebnisse in ein Microsoft Excel Workbook oder eine Datenbank einträgt.
Er wurde komplett in der objektorientierten Programmiersprache Java entwickelt und ist dabei so aufgebaut, dass er immer zwei getrennte XML-Dateien benötigt:

- die Preferences und
- den SqlJob.

In den Preferences werden Einstellungen u.a. zur Datenbankverbindung vorgenommen. Der SqlJob beinhaltet u.a. die auszuführenden Befehle und Informationen zum Export der Ergebnisse.

Die Erstellung und die Bearbeitung von SqlJobs müssen in XML-Notation erfolgen, dafür kann ein beliebiger Texteditor verwendet werden. Außerdem kann ein SqlJob mit Hilfe einer grafischen Benutzeroberfläche (GUI) angepasst werden. Das Starten von SqlJobs kann über einen Shell-Aufruf oder die GUI geschehen.
Für die Weiterverarbeitung der Ergebnisse stehen verschiedene Alternativen zur Verfügung. Sie können in Microsoft Excel – in leere Worksheets oder vorgegebenen Templates – sowie in Microsoft Powerpoint an der gewünschten Stelle mit beliebigen Formatierungen eingefügt werden. Die Ergebnisse können aber auch in eine CSV-Datei oder direkt in festgelegte Tabellen verschiedener Datenbanken exportiert werden. Des Weiteren können SQL-Script-Dateien, die zu einem späteren Zeitpunkt oder zur Ausführung auf anderen Datenbanken verwendet werden, und ausführbare bat-Dateien erstellt werden.
Mit dem SqlSurfer können verschiedene Aufgaben erledigt werden: Datenbanken auslesen, Daten importieren, aufbereiten, kopieren, manipulieren und exportieren, Scripts starten und Berechnungen durchführen sowie Reports erstellen.

3.2 Vorteile

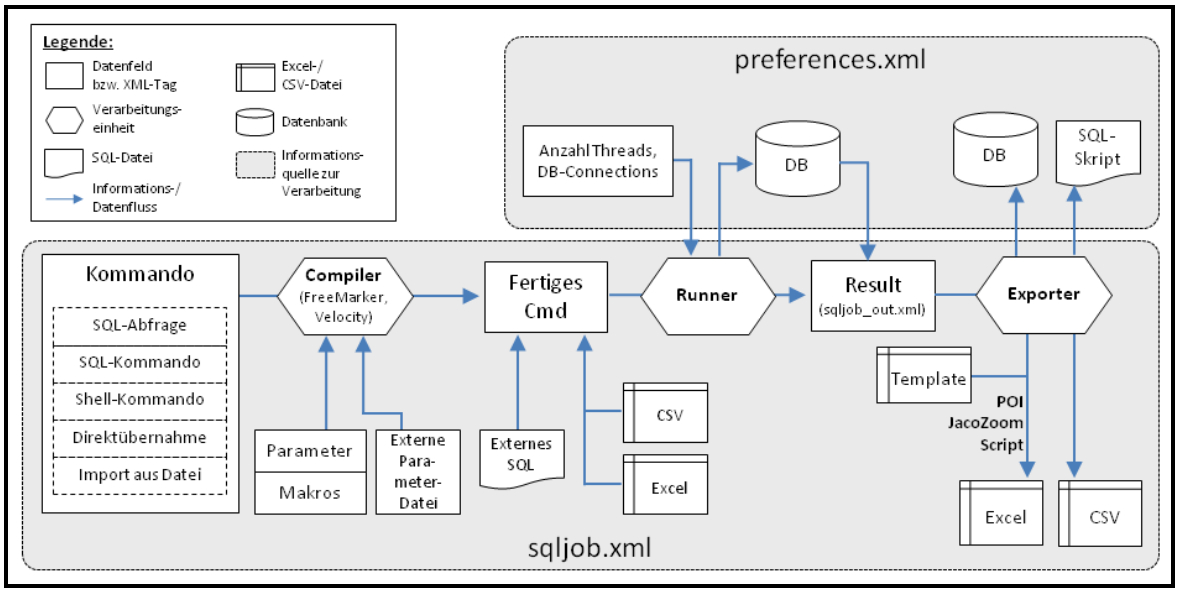
Durch seinen Aufbau und die verfügbaren Funktionen bietet der SqlSurfer folgende Vorteile:

- Automatisierung von Arbeitsschritten (Datenbank-Zugriff, Kopieren, Verarbeiten und Exportieren von Daten)
- Erstellung von komplexen und flexiblen Reports als Programm
- Einfache und dennoch mächtige Strukturierung von Reporting-Aufgaben
- Zusammenfassung von diversen Aufgaben in einem Verarbeitungstool
- Einfache Einbindung in eigene Programme
- Modellierung und Berücksichtigung von Abhängigkeiten in den Programmabläufen
- Verbindung verschiedener Datenbanken in einem SqlJob

- Flexibilisierung von Reports durch Parameter und Makros

3.3 Architektur

Die folgende Grafik gibt einen Überblick über die Architektur des SqlSurfers:



Die verwendeten Begriffe sowie die Zusammenhänge der einzelnen Komponenten werden in den nachfolgenden Kapiteln erläutert.

3.4 Begriffe

Zum besseren Verständnis der Architektur des SqlSurfers und der Ausführungen in den nachfolgenden Kapiteln werden an dieser Stelle einige Begriffe erklärt.

Begriff	Erklärung
SqlJob	Ein SqlJob ist eine XML-Datei mit der Endung .sqj, die alle Befehle (Kommandos) enthält, die der SqlSurfer für die Erstellung eines Reports oder zum Durchführen anderer Aufgaben ausführt sowie Informationen darüber gibt, wohin die Ergebnisdaten exportiert werden sollen.
Preferences	In der Datei preferences.xml werden grundlegende Einstellungen vorgenommen, die für den zugehörigen SqlJob gelten, wie z.B. Datenbankverbindungen, die Anzahl der parallel zu verarbeitenden Threads oder die Log-Levels.
Scheduler	Der Scheduler ist ein Steuerprogramm, das die zeitliche Ausführung mehrerer Prozesse im SqlSurfer regelt. Abhängig von den Einstellungen in den Preferences zur maximalen Anzahl von gleichzeitigen Prozessen und der Anzahl der Datenbankverbindungen steuert der Scheduler, ob die Berechnungen eines SqlJobs sequenziell oder parallel ausgeführt werden.
Compiler	Der Compiler transformiert den Code einer Skriptsprache (z.B. FreeMarker oder Velocity) mit beliebigen Variablenaufrufen (Parameter) oder Funktionsaufrufen (Makros) in einen von einer Datenbank ausführbaren Code, er erstellt demnach ein fertiges

	SQL-Statement.
Runner	Der Runner erhält vom Compiler fertige Kommandos und schickt diese an eine Datenbank bzw. eine sonstige Auswerteeinheit (z.B. Shell).
Exporter	Der Exporter führt den letzten Schritt eines SqlJobs aus. Er exportiert die berechneten Ergebnisse in ein vorgegebenes Ausgabeformat (z.B. Excel- oder CSV-Datei, Datenbank-Tabelle) mit den gewünschten Formatierungen.
Parameter	Parameter sind Variablen, die zur Flexibilisierung des SqlJobs beitragen. Sie werden zentral an einer Stelle definiert und können an beliebigen Positionen des SqlJobs aufgerufen werden. Ein Parameter kann dabei immer nur einen bestimmten Wert (Text, Zahl, Datum usw.) annehmen. Parameter werden an der entsprechenden Stelle am Anfang des SqlJobs festgelegt, über eine externe Parameter-Datei aufgerufen oder beim Aufruf eines Batch-Jobs an diesen weitergegeben.
Makro	Ein Makro ist ein kleiner Teil eines Programm-Codes, der zur Vereinfachung vom Kommando-Bereich des SqlJobs ausgelagert wird, da der Code komplexe oder sich wiederholende SQLs enthält. Ein Makro kann von einem Kommando an beliebigen Stellen aufgerufen werden und in Makros können wiederum Parameter eingebaut werden.
Kommando	Ein Kommando ist der wesentliche Teil eines SqlJobs, der die auszuführenden Befehle enthält. Er kann aus verschiedenen Komponenten bestehen, z.B.: ▪ ein gewöhnliches SQL-Statement ▪ Aufruf von Makros und Parametern ▪ Aufruf von anderen SqlJobs über den Batch-Modus (optional mit Übergabe von Parametern) ▪ Import von SQL-Statements, CSV-Dateien, Excel Workbooks Scripts oder Batchprogramme Um das Ergebnis eines Kommandos zu exportieren, muss das Kommando eine Referenz auf die festgelegten Ausgabe-Varianten enthalten.
Template	Als Template wird hier eine Vorlage für ein Excel-Workbook bezeichnet, das aus mehreren Excel-Sheets bestehen kann. Dieses Workbook wird vor der Ausführung eines SqlJobs im Datenverzeichnis mit den gewünschten Formatierungen gespeichert und vom Exporter des SqlSurfers zur Befüllung der Excel-Sheets verwendet. Ein Template wird jedoch nie überschrieben, sondern der SqlSurfer kopiert es und speichert das befüllte Excel-Workbook unter einem anderen Namen ab.

3.5 Ausführen der Applikation

3.5.1 Starten aus einer Shell

Die prinzipielle Architektur des SqlSurfer ist shell-basiert. Entsprechend wird sie in einem Verzeichnis ausgeführt.

In diesem Verzeichnis muss eine Datei preferences.xml und eine XML-Datei mit dem SqlJob (z.B. sqljob.sqj) existieren.

Das Starten erfolgt mit dem Befehl sq. Es können SqlJobs gestartet werden,

- ohne dass eine GUI die Parameter noch mal zur Eingabe anbietet (run-Mode),
- die Parameter des Jobs vorher noch zum Eingeben angeboten werden (gui-Mode) und
- die XML-Datei mit dem SqlJob in einem Editor editiert wird (edit-Mode).

Es sind folgende Befehle möglich:

Sq	Editiert sqljob.xml.
sq edit	Editiert sqljob.xml.
sq run	Startet sqljob.xml an.
sq export	Startet nur den Export von sqljob_out.xml an.
sq gui	Startet sqljob.xml über eine gui an.
sq <inFile>	Editiert das File <inFile>
sq edit <inFile>	Editiert das File <inFile>
sq run <inFile>	Startet das File <inFile> an
sq export <inFile>	Startet nur den Export des Files <inFile>_out.xml an.
sq gui <inFile>	Startet das File <inFile> über eine gui an
sq run <inFile> -param „name1=wert1“ „name2=wert2“ usw.	Startet das File <inFile> an und belegt die Parameter gemäß Eingabe
sq gui <inFile> -param „name1=wert1“ „name2=wert2“ usw.	Startet das File <inFile> über eine gui an und belegt die Parameter gemäß Eingabe

Die Ergebnisse der Verarbeitung werden standardmäßig im aktuellen Verzeichnis abgelegt. Dazu gehören:

- eine XML-Datei mit dem Status nach Ausführung der Kommandos (z.B.sqljob_out.xml),
- alle Dateien, die durch Output-Definitionen erstellt wurden,
- ggf. eine Datei out.log mit allen Logs
- ggf. out_<Id>.<ext> Dateien mit den fertig kompilierten Kommandos und
- ggf. einige Zwischendateien, falls kein anderes Tmp-Verzeichnis in den Preferences definiert wurde.

3.5.2 Starten per Doppelklick

Es gibt auch die Möglichkeit, den SqlSurfer per Doppelklick zu starten. Dazu muss eine Dateiverknüpfung mit der Dateiendung .sqj auf die Datei „sq.exe“ angelegt werden. Dabei erstellt der SqlSurfer automatisch einen entsprechenden Eintrag in der Windows-Registry, daher sind je nach Windows Version Administrator-Berechtigungen erforderlich.

Außerdem ist die Ausführung des Menüpunktes Optionen->Installationseinstellungen erforderlich.

3.6 Bestandteile der System-Architektur

In diesem Kapitel werden die wichtigsten Bestandteile der System-Architektur und deren Zusammenspiel erläutert.

3.6.1 Preferences

In den Preferences werden allgemeine und datenbankspezifische Einstellungen festgelegt. Die entsprechende Datei preferences.xml muss immer in dem Verzeichnis liegen, aus dem der SqlSurfer aufgerufen wird.

3.6.1.1 Datenbankverbindungen

Damit der SqlSurfer auf Daten einer Datenbank zugreifen kann, muss für jede Datenbank in den Preferences eine Datenbankverbindung eingerichtet werden. Grundsätzlich ist dies für alle Datenbanken möglich, für die es einen JDBC-Treiber gibt.
Die Java Database Connectivity (JDBC) dient dabei als Datenbankschnittstelle zur Java-Applikation SqlSurfer. Sie leitet SQL-Abfragen an die Datenbank weiter und wandelt die Ergebnisse so um, dass sie von Java weiterverarbeitet werden können.

Zum Einrichten einer neuen Datenbankverbindung muss zunächst die Jar-Datei des JDBC-Treibers in das Installationsverzeichnis kopiert werden. Anschließend werden in den Preferences folgende Angaben benötigt:

- Eindeutiger Name der Verbindung
- Name der Java-Klasse, die den JDBC-Treiber enthält
- Verbindungs-URL für die jeweilige Datenbank
- Username für die Datenbankverbindung (optional)
- Passwort als Klartext oder verschlüsselt (optional)

Ein SqlJob kann beliebig viele unterschiedliche Datenbankverbindungen verwenden.

Der SqlSurfer übernimmt eigenständig das Management der Datenbankverbindungen. Er regelt, wie viele Verbindungen zu welcher Datenbank gleichzeitig benutzt werden können. Dabei wird Connection Pooling verwendet. Dieses dient zur Wiederverwendung physischer Datenbankverbindungen und steigert die Geschwindigkeit. Bei Bedarf nimmt der SqlSurfer eine Verbindung aus dem Connection Pool und verwendet sie zur Ausführung von Kommandos. Wenn die Verbindung nicht mehr benötigt wird, wird sie nach einer bestimmten Idle-Time (Leerlaufzeit) automatisch wieder abgebaut.

Über zusätzliche Properties können Eigenschaften der Verbindungen eingestellt werden. Ggf. erstellt man zwei SqlConnection Einträge zur gleichen Datenbank, wenn man unterschiedliche Properties verwenden möchte.

3.6.1.2 Dateiverknüpfungen

Mit einer Dateiverknüpfung wird festgelegt, mit welchem Programm Dateien eines bestimmten Typs, d.h. mit einer bestimmten Dateiendung, geöffnet werden sollen. Diese verknüpfungen werden all allen Stellen im SqlSurfer verwendet, wenn man Dateien anzeigen oder auch Scripte (u.ä.) anstarten möchte.

In Microsoft Windows gibt es in der Windows-Registry standardmäßig eine Liste von Programmen, die zum Ausführen der gängigsten Dateitypen verwendet werden. Diese kann jedoch manuell verändert werden.
Unabhängig davon können in den Preferences des SqlSurfers auch Dateiverknüpfungen definiert werden, die von der Windows-Standard-Belegung abweichen.
Über den Modus der Zuordnungslogik von Dateitypen gibt es nun folgende Auswahlmöglichkeiten:

- System: Verwendung der Zuordnung des Betriebssystems
- Explicit: Explizite Angabe der Dateiverknüpfung:
  - File: Pfad und Dateiname des zu verwendenden Programms
  - Cmd: Struktur eines Shell-Befehls (Pfad/Dateiname des Programms und zu öffnender Dateiname, mit einem Leerzeichen getrennt)

3.6.1.3 Session Properties

Über Session Properties können bestimmte Einstellungen für jeweils eine Session vorgenommen werden.
Es handelt sich dabei um eine Liste von Paaren aus den Attributen Name und Wert, deren Ausprägung bei jeder Datenbankverbindung anders ist. Einzelheiten dazu müssen beim JDBC-Treiber oder bei der Datenbank nachgelesen werden.

Bei der Datenbank Derby kann z.B. über die Property „create=true“ geregelt werden, dass die Datenbank neu angelegt wird, falls diese noch nicht existiert. Das kann wie folgt spezifiziert werden:

```
<Preferences>
  <SqlConnection
    driver="org.apache.derby.jdbc.EmbeddedDriver"
    url="jdbc:derby:db"
    passwdCrypt="C193E6B7B99FDD48">
    <SessionProperty id="create" value="true"/>
  </SqlConnection>
…
```

3.6.1.4 Loglevels

Der SqlSurfer wurde so konzipiert, dass für jeden einzelnen Prozessschritt ein Logeintrag erstellt wird. Dazu wurden verschiedene Loglevels definiert, um das Logging gemäß den Anforderungen bzw. Wünschen des Anwenders ausführlicher oder knapper zu gestalten.
Folgende Loglevels stehen zur Verfügung:

- Trace
- Debugging
- Informationen
- Warnungen
- Fehler
- fatale Fehler

Damit die Logeinträge übersichtlicher sind und der jeweiligen Funktionalität des SqlSurfers zugeordnet werden können, werden sie in nachfolgende Kategorien eingeteilt:

- Informationen über SQL-Befehle, die an Datenbanken gesendet werden
- Informationen über Datenbankverbindungen mit Treibern sowie deren Auf und Abbau
- Informationen über den Scheduler
- Programmfortschritt
- Applikationsmeldungen
- Informationen über erstellte Dateien die gespeichert werden.
- Statistiken über die Ausführungszeiten
- Informationen des Exporter
- Informationen des Compilers
- Automation
- Shell-Befehle

Die gesamten Logeinträge können automatisch in ein Textdokument (mit der Dateiendung .log) gespeichert werden. Der für diese Einstellung nötige XML-Tag lautet:

```
<SqlJob
    exportLog="true">
…
```

Eine Übersicht über alle möglichen Logeinträge der verschiedenen Loglevels und Kategorien befindet sich im Referenzteil dieses Dokuments.

3.6.2 SqlJob

Der SqlJob ist das Kernstück für die Verarbeitung von Befehlen im SqlSurfer. Er enthält alle Informationen zu Kommandos, Makros, Parametern sowie Output- und Export-Anweisungen.
Zunächst werden aber einige allgemeine Einstellungen festgelegt, wie die Namen des Reports, des Laufes, des Laufverzeichnisses oder Bemerkungen.
Für jede Funktion gibt es einen XML-Tag, der über einen Texteditor oder über die GUI editiert werden kann.

3.6.2.1 Report-Name und Bemerkungen

Für jeden SqlJob besteht die Möglichkeit, einen Report-Namen festzulegen und beliebige Bemerkungen anzufügen.
Folgende XML-Tags sind dafür vorgesehen:

```
<SqlJob
    id="Bsp: Monatlicher Report 01/2011"
    desc="Bsp: Das ist eine Bemerkung.">
…
```

3.6.2.2 Laufverzeichnis und Laufname

Um unterschiedliche Läufe desselben SqlJobs besser organisieren zu können, kann für jeden Lauf ein Laufverzeichnis und ein Laufname bestimmt werden.
Wird ein Laufverzeichnis (rundir) angegeben, so wird ein Verzeichnis mit diesem Namen erstellt und alle gewünschten Ausgabe-Dateien nach der Berechnung durch den SqlSurfer in dieses Verzeichnis exportiert. Das Laufverzeichnis ist dabei ein Unterverzeichnis des Datenverzeichnisses, in dem sich der SqlJob befindet.

Der Name des Laufverzeichnisses kann jedoch auch aus dem Laufnamen oder anderen Referenzen gebaut werden (Zur Vereinfachung wird hier als Referenz nur der Laufname verwendet; für weitere Möglichkeiten zum Bau von Namen: siehe Kapitel *3.7.2.4 Verwendung von Makros und Parametern zum Bau von Feldern*):

```
<SqlJob
     runname="Final_01"
     rundir="Lauf_${SQ_JOB.getName()}">
…
```

Durch diese Anweisung werden die Ausgabedateien in einem Verzeichnis mit dem Namen „Lauf_Final_01“ gespeichert.

Wird nun wie im obigen Beispiel mit dem Laufverzeichnis auf den Laufnamen referenziert, jedoch ist der XML-Tag „runname“ leer oder nicht vorhanden, so wird als Default-Wert für eine Referenz auf einen leeren Laufnamen ein Timestamp des Erstellungszeitpunkts verwendet.

Der entsprechende Teil des SqlJobs könnte folgendermaßen aussehen:

```
<SqlJob
     runname=""
     rundir="Lauf_${SQ_JOB.getName()}">
…
```

In diesem Fall wird ein Verzeichnis mit dem Namen bzw. Format „Lauf_YYYY_MMDD_HHMM_SS“ erstellt.

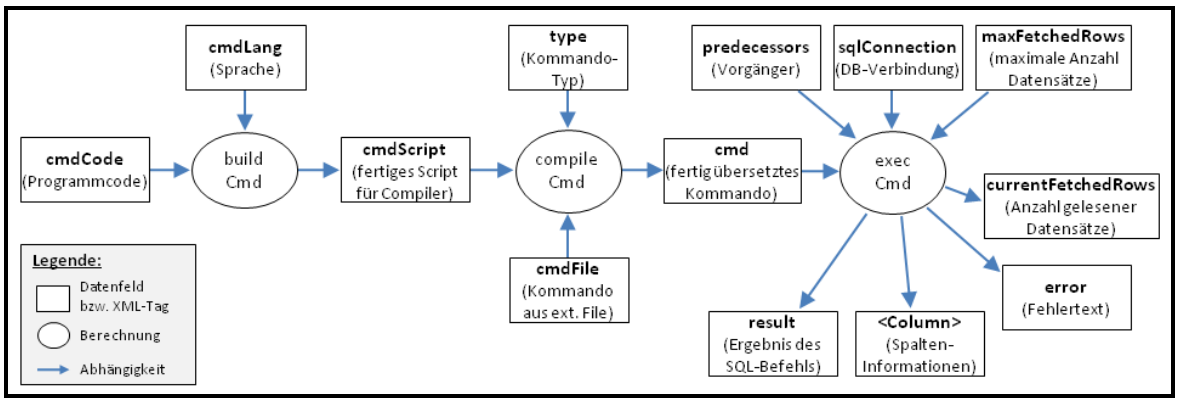
Um Probleme beim Öffnen der Ausgabe-Dateien zu vermeiden, wird empfohlen, darauf zu achten, dass das letzte Zeichen des Laufverzeichnisses kein Leerzeichen ist.

3.6.3 Kommando

Das Kommando bildet den wesentlichen Teil eines SqlJobs. Es beinhaltet die auszuführenden Befehle und besteht abhängig vom Kommando-Typ aus verschiedenen Bestandteilen.
Zum Exportieren von Ergebnissen muss im Kommando eine Referenz auf die gewünschten Ausgabe-Varianten festgelegt werden.

3.6.3.1 Aufbau

Das eigentliche Kommando wird Schritt für Schritt über verschiedene Wege erstellt.
In folgender Grafik ist der Ablauf der Erstellung eines Kommandos dargestellt:



In der Grafik sind alle XML-Tags aufgeführt, die bei der Erstellung eines Kommandos beteiligt sind. Zu jedem XML-Tag ist die entsprechende Bezeichnung des Feldes in der GUI bzw. eine kurze Beschreibung beigefügt.

3.6.3.2 Kommando-Typen

Für jedes Kommando muss zunächst der Typ festgelegt werden. Davon hängen die weiteren erforderlichen Einstellungen ab.

Kommando-Typ	Erklärung
SQL-Abfrage	Abfrage an eine Datenbank, bei der ein Ergebnis erwartet wird, in der Regel SELECT (Default Kommando-Typ)
SQL-Kommando	Liste von Kommandos an eine Datenbank ohne Rückmeldung von Ergebnissen, z.B. CREATE, INSERT INTO
Shell-Kommando	Kommando an Shell, Windows Scripting Host oder anderes Programm, das an eine Dateierweiterung gebunden ist. Das zusammengebaute Kommando wird in eine Datei mit Endung gespeichert. Dann das an die Endung gebundenen Programm aufgerufen und das Ergebnis auf Standard.Out als Kommando genommen.
Direktübernahme	Direktübernahme des Kommandos in das Ergebnis
Import aus Datei	Import von Daten aus einer Datei (.csv oder .xls)

3.6.3.3 Import von CSV-/ SQL-Dateien

Der SqlSurfer bietet neben dem Schreiben von SQL-Statements oder anderen Befehlen im Kommando auch eine Funktionalität, um komplette Dateien in ein Kommando zu importieren. Dies ist möglich für CSV- und SQL-Dateien.

- Import von CSV-Dateien
  Zum Importieren von Daten aus einer CSV-Datei muss der Kommando-Typ „Nichts“ eingestellt und die CSV-Datei im entsprechenden XML-Tag angegeben werden:

```
<SqlJob
  <Command
    type="none"
```

```
        cmdFile="csv_datafile.csv">
…
```

- Import von SQL-Dateien
  Um externe SQL-Dateien in ein Kommando zu importieren, ist der Kommando-Typ „SQL-Abfrage“ erforderlich. Da dies die Default-Einstellung bei jedem Kommando ist, muss der Typ hier nicht angegeben werden:

```
<SqlJob
  <Command
    cmdFile="sql_extern.sql">
…
```

Die zu importierende Datei kann dabei über die Angabe des absoluten oder relativen Pfades geladen werden.

Alternativ können CSV und Excel Dateien auch Strukturiert über den Reiter Cmd geladen werden.

3.6.4 Parameter

Parameter sind Variablen, die einen bestimmten Wert annehmen und beliebig oft im SqlJob aufgerufen werden können. Sie können entweder direkt im SqlJob oder über eine externe Parameter-Datei festgelegt werden.
Ähnlich wie bei einem SqlJob kann auch zu einem Parameter eine Bemerkung über den XML-Tag „desc“ hinzugefügt werden.

3.6.4.1 Parameter-Definition im SqlJob

Bei der Definition von Parametern werden ein eindeutiger Name, ein Datentyp für den Parameter und der Parameter-Wert bestimmt. Als Datentypen stehen String, Nummer, Zeitstempel und ein String als Auswahl aus einer festen Liste zur Verfügung.

Beispiel für Parameter-Definition:

```
<SqlJob
  <Parameter
    id="Param1"
    val="12345"
    label="Beschriftung01"
    desc="Bemerkung01"/>
…
```

Bei Verwendung der GUI können die Parameter zusätzlich noch mit einer Beschriftung versehen werden, um die Anzeige der Parameter übersichtlicher zu gestalten.

3.6.4.2 Externe Parameter-Datei

Die Belegung der Parameter kann auch in einer externen Datei erfolgen. Die Datei muss im Datenverzeichnis des SqlJobs liegen und wird über den Dateinamen im Feld paramFile des SqlJobs aufgerufen.

Beispiel-Aufruf einer Parameter-Datei im SqlJob:

```
<SqlJob
    paramFile="params_extern.xml">
…
```

Beispiel für eine Parameter-Datei:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ParameterValues>
       <ParameterValue id="Param1" val="Neuwert" />
       <ParameterValue id="Param2" val="1234" />
</ParameterValues>
```

Für eine externe Parameter-Datei gilt ebenfalls die XML-Notation.
Durch den Aufruf von Parametern aus einer externen Datei werden lediglich die Werte der Parameter des SqlJobs überschrieben, d.h. im SqlJob müssen zuvor gleichnamige Parameter mit den erforderlichen Datentypen definiert werden.

3.6.4.3 Parameter-Eingabe über Shell

Es besteht auch die Möglichkeit, Parameter direkt über einen Shell-Befehl zu setzen. Dazu werden an den Aufruf des SqlJobs wie folgt entsprechende Argumente angehängt:

sq run <inFile> -param „name1=wert1“ „name2=wert2“ usw.

Bei dieser Methode werden sowohl die im SqlJob als auch die in einer Parameter-Datei gesetzten Parameter-Werte überschrieben.

3.6.5 Makros

Mit einem Makro kann ein Teil eines Codes zur Vereinfachung von Kommandos ausgelagert werden. Häufige Anwendung finden Makros, wenn der Code komplexe oder sich wiederholende SQL-Statements enthält.
Ein Makro kann von einem Kommando an beliebigen Stellen aufgerufen werden und in Makros können wiederum Parameter eingebaut werden.

3.6.5.1 Aufruf von Makros

Die Syntax für den Aufruf von Makros hängt immer von der verwendeten Template Engine ab. Diese wird in Kapitel *3.7.2 Compiler* detailliert dargestellt.
In den folgenden Beispielen wird jeweils ein Makro mit der Template Engine FreeMarker aufgerufen (für die Verwendung anderer Template Engines: siehe Referenzteil):

```
<SqlJob
  <Macro
    id="macro01"
    cmdLang="freemarker"
    body="Test_Macro_Body"/>
  <Command
    cmdLang="freemarker"
    cmdCode="[@macro01/]">
…
```

An allen Stellen von *[@macro01/]* wird der Text *Test_Macro_Body* eingesetzt.

3.6.5.2 Verwendung von Argumenten

Mit dem SqlSurfer können Makros nicht nur aufgerufen werden. Es besteht die Möglichkeit, im Kommando beim Makro-Aufruf einen lokalen Parameter mit einem beliebigen Wert an das Makro zu übergeben. Der lokale Parameter wird im Makro als Argument angegeben und kann an gewünschter Stelle in das Makro eingebaut werden. Dies soll folgendes Beispiel verdeutlichen:

```
<SqlJob
  <Macro
    id="macro02"
    cmdLang="freemarker"
    args="param01"
    body="select '${param01}' as testcolumn from SYSIBM.SYSDUMMY1"/>
  <Command
    cmdLang="freemarker"
    cmdCode="[@macro02 param01="Test_Parameter_local"/]"/>
…
```

Es ist zu beachten, dass der Parameter “param01” vorher nicht gesondert definiert wurde. Das Ergebnis dieses SqlJobs wäre ein ResultSet mit einer Spalte namens “TESTCOLUMN“ und einer Ergebnis-Zeile mit dem Wert „Test_Parameter_local“.

3.6.6 Fertiges Cmd

An dieser Stelle der Prozesskette liegt das fertig gebaute Cmd zur Weiterverarbeitung vor. Das Kommando wurde vom Compiler gemäß dem gewählten Kommando-Typ bearbeitet, es wurden alle Parameter und Makros integriert sowie die zu importierenden CSV- oder SQL-Dateien geladen und eingebaut. Nun kann das fertige Cmd ausgeführt werden.

3.6.7 Result

Das vom Compiler fertiggestellte Cmd wurde ausgeführt und die Ergebnisse werden in die Datei sqljob_out.xml gespeichert. Sie hat den gleichen Aufbau wie der SqlJob selbst, enthält jedoch zusätzliche Informationen wie zum Beispiel:

- Verarbeitungsdauer, Start- und Endzeit des SqlJobs und der einzelnen Kommandos
- das cmdScript sowie das fertige Cmd für jedes Kommando
- das Ergebnis jedes Kommandos

- die Anzahl der gelesenen Datensätze

Sie ist eine Eingabe- und Ergebnisdatei in Einem. Im Regelfall wird sie nur intern verwendet, bietet aber gute Möglichkeiten um die Berechnungen und Ersetzungen nachzuvollziehen.

3.6.8 Output

3.6.8.1 Benennung von Ergebnis-Dateien

Die vom Runner erzeugten Ergebnisse werden wiederum in einer XML-Datei gespeichert. Diese Datei beinhaltet neben den Ergebnissen auch den kompletten SqlJob und wird, wenn möglich, in der Form „[SqlJob-Name]_out.xml“ benannt. Sie bildet die Grundlage für den Export von Output-Dateien.
Falls die zu erzeugende Ergebnis-Datei im Datenverzeichnis bzw. Laufverzeichnis bereits existiert, wird der Name der bereits vorhandenen Datei mit einem Timestamp ihres Erstellungszeitpunkts versehen und entsprechend in die Form „[SqlJob-Name]_out_YYYY_MMDD_HHMM.xml“ unbenannt. Ist der Dateiname der bereits vorhandenen Datei gesperrt, weil sie beispielsweise gerade geöffnet ist, kann diese Datei vom SqlSurfer nicht umbenannt werden. In diesem Fall wird die zu erzeugende Datei unter dem Namen „[SqlJob-Name]_out_2.xml“ gespeichert.
Dieses System für die Benennung von Ergebnis-Dateien gilt für alle Ausgabe-Dateien.

3.6.8.2 Betrachten von Ergebnissen eines Kommandos

Die erstelle Datei „[SqlJob-Name]_out.xml“ kann ihrerseits wieder in den SqlSufer geladen werden. Dann werden neben den Eingabedaten auch die Zwischenergebnisse vollständig mit angezeigt.

3.6.8.3 Behandlung erstellter Dateien

Nach der Durchführung der Berechnungen und dem Export können die erstellten Ausgabe-Dateien auf verschiedene Weise behandelt werden. Sie können entweder nur im Filesystem abgelegt oder direkt vom SqlSurfer geöffnet werden. Diese Funktionalität wird durch folgendes Feld geregelt:

```
<SqlJob
  <Output
    viewResult="true"/>
…
```

Durch diese Einstellung wird die Ausgabe-Datei sofort geöffnet. Die Ablage im Filesystem ist als Default-Wert vorgegeben und muss daher nicht extra eingestellt werden.

3.6.8.4 Ausgabe von Kommandos und Logs

Es besteht die Möglichkeit, dass der SqlSurfer automatisch alle gerechneten Kommandos als einzelne SQL-Dateien und alle Logeinträge als Log-Dateien speichert. Diese Dateien

werden jeweils im Datenverzeichnis des SqlJobs oder, falls angegeben, im Laufverzeichnis abgelegt.
Hierfür sind folgende XML-Tags vorhanden:

```
<SqlJob
    exportCmds="true"
    exportLog="true"
…
```

3.6.8.5 Statistiken

Der SqlSurfer merkt sich die Anfangs- und Endzeit jedes Kommandos und des gesamten SqlJobs und ermittelt daraus jeweils die Dauer. Diese Zeiten werden in der Datei sqljob_stat.xml im Datenverzeichnis bzw. Laufverzeichnis gespeichert.
Wird ein SqlJob erneut gestartet und es liegt bereits eine Statistik-Datei vor, wird diese vom SqlSurfer verwendet, um die geplante Berechnungsdauer zu ermitteln.

3.6.8.6 Ausgabearten

Abhängig vom gewählten Befüllungsmodus können folgende Arten von Ausgabe-Dateien erstellt werden:

Befüllungsmodus	Ausgabe-Datei
Poi	Excel-Datei (Erstellt mit dem Toolkit Poi, das auch unter Linux und Solaris funktioniert)
Zoom	Excel-Datei
Script	Excel-Datei
Csv	CSV-Datei
Db	Tabelle in DB
Dbsql	SQL-Datei
powerpoint	Powerpoint-Datei
Word	Word Dokument

Der genaue Ablauf der Erstellung der Ausgabe-Dateien wird in Kapitel *3.7.5 Exporter* beschrieben.

3.7 Prozessablauf

Nach den Bestandteilen der System-Architektur im vorherigen Kapitel wird an dieser Stelle auf den Prozessablauf innerhalb des SqlSurfers eingegangen.

3.7.1 Konzept eines SqlJobs

Zunächst sollen einige konzeptionelle Aspekte eines SqlJobs dargestellt werden, bevor anschließend die einzelnen Prozessschritte gezeigt werden.

3.7.1.1 Prüfungen vor der Ausführung

Vor der Ausführung eines SqlJobs prüft der SqlSurfer die vorliegende Infrastruktur:

- Prüfung, ob eine Datenbank-Verbindung benötigt wird:
  Wenn ja: jeweils eine DB-Verbindung zum Laden der Daten sowie für den Exporter
- Zugriff auf Verzeichnisse
- Verwendung schreibbarer Dateinamen der Ausgabe-Dateien
- Abhängigkeiten (auf Kommando-Ebene)

3.7.1.2 Status eines SqlJobs

Während der Ausführung durchläuft ein SqlJob verschiedene Status. Diese haben folgende Bedeutung:

Status	Bedeutung
Initialisiert	SqlJob wurde erstellt und kann ausgeführt werden
in Vorbereitung	Es werden Prüfungen bzw. vorbereitende Checks gemacht, um augenscheinliche Fehler sofort zu finden
Rechnend	SQLs werden gebaut und ausgeführt (einzelne Kommandos); wenn direkter Export eingestellt, werden Daten ohne Zwischenspeicherung direkt exportiert
Exportieren	Alle zwischengespeicherten Daten werden exportiert
Fertig	Der SqlJob wurde komplett ausgeführt

3.7.1.3 Fehlerbehandlung

Alle Fehler, die beim Ausführen eines SqlJobs auftreten, werden im Log angezeigt und protokolliert. Sie können auch durch Öffnen der Datei sqljob_out.xml über die GUI im jeweiligen Kommando betrachtet werden.
Der SqlSurfer versucht aber immer, auftretende Fehler zu umgehen, indem andere Dateinamen verwendet oder Teilschritte weglassen werden.
Eine ausführliche Liste an möglichen Fehlermeldungen befindet sich im Referenzteil dieses Dokuments.

3.7.2 Compiler

Nachdem der eingegebene Programmcode (cmdCode) mit Hilfe einer Skriptsprache wie zum Beispiel FreeMarker oder Velocity mit beliebigen Variablenaufrufen (Parameter) oder Funktionsaufrufen (Makros) zu einem fertigen Script (cmdScript) aufgelöst wurde, wird er vom Compiler ausgelesen. Abhängig vom gewählten Kommando-Typ transformiert ihn der Compiler in einen von einer Datenbank ausführbaren Code, also ein fertiges SQL-Statement, oder in einen ausführbaren Befehl für eine Betriebssystem-Shell.

Der SqlSurfer ist so konzipiert, dass an verschiedenen Stellen sog. Template Engines verwendet werden können. Dabei handelt es sich um eine Software, die Platzhalter in Dateien mit aktuellem Inhalt füllt.
Im SqlSurfer sind standardmäßig die FreeMarker Template Language (FTL) sowie die Velocity Template Language (VTL) von Apache integriert. Diese beiden Template Engines sind Java-basiert und frei erhältlich. Über ein API können weitere Scriptsprachen mit eingebunden werden.
Innerhalb eines SqlJobs besteht die Möglichkeit, mehrere verschiedene Template Engines zu verwenden, da diese für jedes einzelne Kommando und Makro explizit angegeben werden können.

3.7.2.1 FreeMarker

Die FreeMarker Template Language (FTL) wird verwendet, um in einer Java-Anwendung ein Template einzulesen, dieses mit Daten aus einem Datenmodell zu ergänzen und es dann in einen Outputstream zu schreiben.
Das Datenmodell in FreeMarker ist vergleichbar mit einem Baum. Ausgehend von einem Wurzelelement können die Äste mit der aus Java bekannten Punktnotation erreicht werden. Ist zum Beispiel die Variable *name* ein Element von *person*, kann der Zugriff über *person.name* erfolgen.

3.7.2.1.1 Codedarstellung

In FreeMarker gelten folgende Grundregeln für den Programmcode:

Code:	Bedeutung:
${…}	Aufruf einer Variablen
[#...]	Verwendung von Standard-FTL-Tags (z.B. assign, if, list)
[@...]	Verwendung von selbstdefinierten FTL-Tags (z.B. Makros)
[#-- Kommentar --]	Einfügen von Kommentaren

3.7.2.1.2 Variablen

Es gibt folgende Arten von Variablen in FreeMarker:

Typen	Datentypen	Bemerkung
Scalar		Scalare sind die Blätter in der Baumstruktur.
	String	Text
	Number	Zahlen
	Boolean	Boolsche Variable
	Date	Datum (oder Zeit)
Hash		Ein Hash ist ein Container, dessen Elemente über Namen (Strings) angesprochen werden können.
Sequence		Die Elemente in einer Sequence können nicht direkt über einen Namen angesprochen werden, sondern nur über ihre Position.

3.7.2.1.3 Built-ins

Mit Built-ins können verschiedene Arten von Operationen und Formatierungen vorgenommen werden. Diese werden mit einem „?“ an den Variablennamen angehängt.
Eine Übersicht über alle zur Verfügung stehenden Built-ins bietet das FreeMarker Manual unter folgendem Link: http://freemarker.sourceforge.net/docs/ref_builtins.html
Einen kleinen Einblick in die Funktionsweise von Built-ins soll folgendes Beispiel liefern:

```
<SqlJob
  <Command
    cmdCode="[#assign str=”abc”]
             ${str?is_number?string}
             ${str?is_string?string(“ja”,”nein”)}>
  </Command>
…
```

Zunächst wird ein String „str“ definiert. Anschließend erfolgt eine Prüfung, ob „str“ das Format „Number“ hat. Der Rückgabewert (Boolean) dieser Prüfung wird in einen String konvertiert.
In der letzten Zeile wird geprüft, ob „str“ ein String ist. Der Rückgabewert wird wieder in einen String konvertiert, wobei bei Boolean-Prüfungen alternative Rückgabewerte anstelle von „true / false“ definiert werden können.

Das Ergebnis dieses Kommandos lautet entsprechend:

false
ja

3.7.2.1.4 Kontrollstrukturen

Im Folgenden wird für FreeMarker die Syntax der wichtigsten Aufrufe dargestellt:

a) Aufruf von Variablen und Makros

Code:	Bedeutung:
${Param1}	Verwendung der Variablen "Param1"
[@Macro1/]	Verwendung des Makros "Macro1"

b) Parameter in Makros

In FreeMarker gibt es auch die Möglichkeit, Parameter an ein Makro zu übergeben. Dieser Mechanismus wird vom SqlSurfer vereinfacht, indem im Makro die gewünschten Argumente angegeben werden können, die im Funktionsrumpf verwendet werden sollen. Das Makro wird vom SqlSurfer automatisch in die korrekte FreeMarker-Syntax übersetzt. Im Kommando können dann Makros aufgerufen und die Parameter mit Inhalt gefüllt werden, wie im nachfolgenden Beispiel dargestellt:

```
<SqlJob
  <Macro
    id="mac1"
    args="param1 param2"
    body="select ‘${param1}’ || ‘{param2}’ as testcolumn from
```

```
        SYSIBM.SYSDUMMY1"/>
  <Command
    cmdCode="[@mac1 param1=“Ersetzung” para2=“klappt”/]">
  </Command>
…
```

Außerdem können Default-Werte für Parameter vergeben werden. Dazu wird bei der Angabe der Argumente hinter den Parameter-Namen der gewünschte Wert geschrieben, z.B. param3=“super“. Im Kommando wird dieser Parameter beim Makroaufruf einfach weggelassen. Im obigen Beispiel müsste also nur folgende Zeile angepasst werden:

```
args="param1 param2 param3=”super”"
```

c) Bedingungen

Code:	Bedeutung:
[#if a<b]*thenbody1* [#elseif a>b]*thenbody2* [#else]*elsebody* [/#if]	Verwendung einer If-Else-Bedingung

d) Variablendefinition

Code:	Bedeutung:
[#assign a = "Var1"]	Definition einer String-Variablen
[#assign x = 1]	Definition einer Number-Variablen
[#assign is = true]	Definition einer Boolean-Variablen
[#assign dat = "21/12/2012"?date("dd/MM/yyyy")]	Definition einer Date-Variablen
[#assign seq=["Tick", "Trick", "Track"]]	Definition einer Sequence

e) Zugriff auf Listenelemente und Teile eines Strings

Code:	Bedeutung:
[#assign seq=["A", "B", "C"]] ${seq[1]}	Zugriff auf das Listenelement an der Indexposition 1 („B“)
[#assign word="abcd"] ${word[2]}	Zugriff auf das Zeichen an der Indexposition 2 („c“)

f) Schleifen

Code:	Bedeutung:
[#assign x=a] [#list 1..x as i]${i} [#if i=a][#break][/#if] , [/#list]	Verwendung der Auflistungsfunktion "list" zur Darstellung einer Zahlenreihe der Zahlen von 1 bis a, mit dem Separator ", "; Nach dem letzten Element wird die Auflistung über „break“ abgebrochen und kein Separator mehr angefügt.
[#assign seq=["A", "B", "C"]] [#list seq as item] ${item} [/#list]	Verwendung der Auflistungsfunktion "list" zur Darstellung von Elementen einer Liste

g) Vorhandensein von Variablen

Code:	Bedeutung:
[#if param1?exists]*thenbody* [#else]*elsebody* [/#if]	Prüfung, ob param1 existiert. Falls ja, wird *thenbody* ausgeführt. Falls nein, wird *elsebody* ausgeführt.
[#if param1?has_content]*thenbody* [#else]*elsebody* [/#if]	Prüfung, ob param1 mit Inhalt gefüllt ist. Falls ja, wird *thenbody* ausgeführt. Falls nein, wird *elsebody* ausgeführt.

h) Funktionen

Code:	Bedeutung:
[#function calc a b c] [#return a*a+b-c] [/#function] ${calc(5,6,7)}	Definition einer Funktion „calc“ mit Parametern a, b, c Bestimmung des Rückgabewertes der Funktion Funktionsaufruf mit Übergabe von Parametern

3.7.2.1.5 Konvertierung von Datentypen

Konvertierungen von Datentypen werden in FreeMarker mit Built-ins vorgenommen.

a) Konvertierung in einen String

Prinzipiell können die Datentypen Number, Boolean und Date mit dem Built-in „?string“ in einen String konvertiert werden. Abhängig vom Datentyp gibt es jedoch noch verschiedene Formatierungsmöglichkeiten:

Konvertierung:	Format:	Ergebnis:
Number -> String	[#assign x=12345.678] ${x?string} ${x?string("0.##")} ${x?string(",##0 €")} ${x?string("0 %")}	 12.345,678 12345,68 12.346 € 1234568 %
Boolean -> String	[#assign bool=true] ${bool?string} ${bool?string("yes","no")}	 true yes
Date -> String	[#assign date3 = "21/12/2012"?date("dd/MM/yyyy")] ${dat?string.short} ${dat?string.medium} ${dat?string("dd.MM.yyyy")}	 21.12.12 21.12.2012 21.12.2012

b) Konvertierung eines Strings in eine Number

Konvertierung:	Format:	Ergebnis:
String -> Number	[#assign num=”3”] ${num?string} ${num?number}	 3 (als String) 3 (als Number)

	${num+num} ${num?number+num?number}	33 (Ergebnis einer Konkatenation) 6 (Ergebnis einer Summe)

3.7.2.1.6 Quoting

Grundsätzlich können Strings mit einfachen (´Text´) und doppelten (“Text“) Anführungszeichen versehen werden.
Falls im hervorgehobenen Text selbst die verwendeten Anführungszeichen vorkommen, müssen diese durch sogenannte Escape-Sequenzen markiert werden.

Beispiel:

Code:	Ausgabe:
${"It's \"quoted\" and this is a backslash: \\"}	It's "quoted" and this is a backslash: \

Bei Numbers und Booleans sind keine Anführungszeichen zu verwenden.

Weitere Informationen zu Quoting und Escape-Sequenzen sind unter folgendem Link zu finden: http://freemarker.sourceforge.net/docs/dgui_template_exp.html

3.7.2.1.7 Unterdrückung von Zeilenumbrüchen im Ausgabe-SQL

Beim Schreiben von SQL-Statements oder anderen Kommandos werden gerne Zeilenumbrüche und Leerzeichen bzw. Tabstopps eingefügt, um den Code übersichtlicher zu machen. Falls nun die vom SqlSurfer ausgeführten Kommandos als Datei gespeichert werden sollen und darin Zeilenumbrüche und Leerzeichen unerwünscht sind, lassen sich diese einfach unterdrücken, indem an die entsprechende Zeile der Trim-Befehl [#t] angehängt wird, wie im folgendem Beispiel:

```
<SqlJob
  <Command
    cmdCode="select [#t]
               ‘NewLine unterdrücken klappt’ [#t]
               as testcolumn [#t]
             from SYSIBM.SYSDUMMY1 [#t]">
  </Command>
…
```

Die exportierte SQL-Datei besteht damit aus nur einer Zeile.

3.7.2.1.8 Aufruf Java-interner Methoden

Aus Freemaker können Java-Klassen und die gesamte Java-Klassenbibliothek verwendet werden. Die steht genauer unter
http://freemarker.sourceforge.net/docs/dgui_template_exp.html.

3.7.2.2 Velocity

Im Folgenden wird die Syntax der wichtigsten Aufrufe in Velocity dargestellt:

a) Funktionsaufrufe

Code:	Bedeutung
${Param1}	Verwendung der Variablen "Param1"
#Macro1	Verwendung des Makros "Macro1"

b) Bedingungen

Code:	Bedeutung:
#if (a<b)*thenbody1* #elseif (a>b)*thenbody2* #else *elsebody* #end	Verwendung einer If-Else-Bedingung

c) Variablendefinition

Code:	Bedeutung:
#set($a = "Var1") #set($x = 1)	Definition einer Variablen

d) Schleifen

Code:	Bedeutung:
#set($i = a) #foreach($x in [1..$i])$x #if($x==$i)#break#end , #end	Verwendung der "foreach"-Schleife zur Darstellung einer Zahlenreihe der Zahlen von 1 bis a, mit dem Separator ", "; Nach dem letzten Element wird die Auflistung über „break“ abgebrochen und kein Separator mehr angefügt.
#set($list = ["A", "B", "C"]) #foreach($item in $list) $item #end	Verwendung der "foreach"-Schleife zur Darstellung von Elementen einer Liste

3.7.2.3 SqlSurfer API

Innerhalb von FreeMarker oder Velocity erhält man Zugriff auf Attribute oder Funktionen eines Jobs und auf Kommandos für den Scheduler. Dazu wird das Objekt SQ_JOB für den aktuellen Job und SQ_SCH für den Scheduler im Kontext von FreeMarker oder Velocity bereitgestellt. Dafür werden im Folgenden einige Beispiele dargestellt:

SQ_JOB FreeMarker:	Bedeutung:
${SQ_JOB.getVersion()}	liefert die aktuelle Version des SqlSurfers

${SQ_JOB.result('sql')}	liefert das Ergebnis des Kommandos mit dem Namen „sql“ zurück
Velocity:	
$SQ_JOB.version	liefert die aktuelle Version des SqlSurfers
$SQ_JOB.result('sql')	liefert das Ergebnis des Kommandos mit dem Namen „sql“ zurück
$SQ_JOB.directsql('con','select * from table1')	Es wird der SQL Befehl an die mit dem namen db geschickt und das Ergebnis Zeilenweise (; als Spaltenseparator) in das Script aungesetzt
SQ_SCH FreeMarker:	
${SQ_SCH.param('name','value')}	Setzt den Wert des Parameters „name“ auf „value“
${SQ_SCH.run('filename','.')}	Startet die Verarbeitung des SqlJobs in der Datei „filename“.
Velocity:	
$SQ_SCH.param('name','value')	Setzt den Wert des Parameters „name“ auf „value“
$SQ_SCH.run('filename','.')	Startet die Verarbeitung des SqlJobs in der Datei „filename“.

Eine vollständige Liste mit allen möglichen Ersetzungen ist im Referenzteil ersichtlich.

In den SqlSurfer können neben den bereits integrierten Template Engines weitere Scriptsprachen eingebunden werden. Die Voraussetzungen und die Vorgehensweise dafür wird in Kapitel *3.9.3 Einbinden eigener Skriptsprachen* detailliert dargestellt.

3.7.2.4 Verwendung von Makros und Parametern zum Bau von Feldern

Zur Benennung von Feldern können nicht nur normaler Fließtext, sondern auch Makros und Parameter herangezogen werden. Diese werden dabei, wie in den Kapiteln *3.7.2.1 FreeMarker* und *3.7.2.2 Velocity* beschrieben, aufgerufen.
Im folgenden Beispiel wird ein Parameter in den Namen des Laufverzeichnisses eingebaut:

```
<SqlJob
     rundir="hello_${Param1}"
  <Parameter
     id="Param1"
     val="World!"/>
…
```

Der Einsatz von Makros und Parametern ist u.a. zum Bau von folgenden Feldern möglich:

- Report-Name und Bemerkungen
- Laufname und Laufverzeichnis
- Parameter-Datei
- Zip-File
- Ausgabe-Dateinamen
- Template-Dateiname und Template-Sheet-Name
- Excel-Sheet-Name
- Startzelle
- DB-Tabellenname

- VBA-Modul-Name und VBA-ModulCall
- Datenbankverbindung
- maximale Anzahl an auszulesenden Datensätzen
- Kommando- oder SQL-Datei

3.7.3 Importer

Der SqlSurfer besitzt die Funktionalität, Daten aus CSV- oder Microsoft Excel-Dateien zur Weiterverarbeitung einzulesen. Über den GUI-Modus kann vor dem eigentlichen Importieren der Datensätze die Importdatei gescannt werden, um automatisch für jede Spalte der Importdatei den Datentyp sowie Precision und Scale bestimmen zu lassen. Nach dem Scannen kann der User, falls nötig, bei einzelnen Spalten nachträglich Name, Datentyp, Precision, Scale und Spaltenposition editieren.

Vor dem Importieren muss als erster Schritt der Pfad der zu importierenden Datei angegeben werden. Außerdem wird die Anzahl der Scanzeilen festgelegt. Soll die gesamte Importdatei bzw. das gesamte Excel-Worksheet gescannt werden, ist hier 0 einzutragen. Anschließend kann das Dateiformat (csv oder xls) eingestellt werden. Davon hängen die nachfolgenden Schritte und Möglichkeiten ab. Spätestens beim Scannen wird das Dateiformat jedoch automatisch vom Importer anhand des Dateityps der Importdatei erkannt und richtig gesetzt.

3.7.3.1 Import von Excel-Dateien

Beispielhafter SqlJob für den Import von Excel-Dateien:

```
<SqlJob
  <Command
    type="import"
    file="C:\ImportDataExcel\Tab03.xls"
    maxScanLines="0"
    fileformat="xls"
    sheet="Tab03_breit"
    startCell="a2"
    readHeader="true"
    readTypes="no"
    importencoding="ISO-8859-1"
    lineSeparator="\n"
    columnMode="var"
    columncountcheck="exact"
    columnSeparator=";"
    columnMarker="&quot;"
    columnMarkerQuoting="&quot;&quot;">
    <ScanColumn
      id="Datum1"
      datatype="date"
      precision="10"
      scale="0"
      numberFormat="dd.MM.yyyy"
      columnPosition="1"/>
...
```

3.7.3.2 Import von CSV-Dateien

Beispielhafter SqlJob für den Import von CSV-Dateien:

```
<SqlJob
  <Command
    type="import"
    file="C:\ImportDataCSV\Tab01.csv"
    maxScanLines="0"
    fileformat="csv"
    readHeader="true"
    readTypes="no"
    importencoding="ISO-8859-1"
    lineSeparator="\n"
    columnMode="var"
    columncountcheck="exact"
    columnSeparator=";"
    columnMarker="&quot;"
    columnMarkerQuoting="&quot;&quot;">
…
```

3.7.4 Runner

Der Runner erhält das fertige Cmd und führt es aus. Abhängig vom Kommando-Typ wird dabei ein SQL an eine Datenbank gesendet, oder ein ausführbarer Befehl an die Betriebssystem-Shell geschickt.

3.7.4.1 Ausführung von Kommandos

In diesem Kapitel soll erläutert werden, wie Kommandos, abhängig vom gewählten Kommando-Typ, ausgeführt werden.

3.7.4.1.1 SQL-Abfrage

Bei der SQL-Abfrage wird vom Compiler als Cmd ein SQL-Statement zusammengesetzt, indem alle Parameter und Makros von der verwendeten Template Engine übersetzt werden. Es handelt sich hier üblicherweise um eine SELECT-Anweisung.
Anschließend schickt der Runner das SQL-Statement über die festgelegte Datenbankverbindung an eine Datenbank und lässt es darauf ausführen. Die zurückkommenden Ergebnisse werden dann vom Exporter verwertet.

3.7.4.1.2 SQL-Kommando

Falls von der angebundenen Datenbank kein Ergebnis erwartet wird, das vom Exporter weiterverarbeitet werden soll, eignet sich der Kommando-Typ SQL-Kommando. Dabei wird ein SQL-Statement nur an die Datenbank geschickt und ausgeführt. Als Anwendungsfälle kommen hier beispielsweise in Betracht:

- Erstellen einer Tabelle über „Create Table“
- Löschen einer Tabelle über „Drop Table“
- Erstellung eines Index

3.7.4.1.3 Shell-Kommando

Bei der Einstellung Shell-Kommando wird ein auszuführender Befehl an die Betriebssystem-Shell geschickt und gestartet.

3.7.4.1.4 Direktübernahme

Bei diesem Kommando-Typ findet keine Übersetzung von Template Engines bzw. Scriptsprachen statt. Die Anweisungen im Kommando werden ohne Änderung direkt in das fertige Cmd übernommen.

3.7.4.1.5 Import aus Datei

Wenn dieser Kommando-Typ gewählt wird, werden Daten aus einer „.csv“- oder „.xls“-Datei importiert. In der GUI im Kommando-Reiter „Import“ können dazu die nötigen Einstellungen vorgenommen werden.

Dazu wird der Dateiname angegeben und ein Scanvorgang angestartet, um automatisch die Struktur zu erkennen. Die Strukturinformationen mit Name und Typ angezeigt und können kontrolliert und im Bedarfsfall nachgebessert werden.

Beim Durchlauf des Jobs wird dann anhand der Strukturinformationen ein Import der Daten durchgeführt.

3.7.4.2 Ausführungsbedingung für Kommandos

Unabhängig vom Kommando-Typ kann für jedes Kommando festgelegt werden, ob es ausgeführt werden soll oder einfach übersprungen wird. Hierfür dient die Ausführungsbedingung „execCond“. Ist das Feld leer, wird das Kommando übersprungen. Wenn das Feld gefüllt ist, wird das Kommando ausgeführt und zusätzlich der eingetragene Text für das Logging verwendet.

```
<SqlJob
 <Command
   id="CMD_1"
   execCond="">
 <ExportData
  …/>
 </Command>
 <Command
   id="CMD_2"
   execCond="Jetzt CMD_2 ausführen!">
 <ExportData
```

```
  …/>
</Command>
…
```

In diesem Beispiel wird das Kommando CMD_1 einfach übersprungen. Das zweite Kommando CMD_2 wird ausgeführt und erzeugt im Log die Meldung „Berechnung: Jetzt CMD_2 ausführen!“

3.7.4.3 Abhängigkeiten zwischen Kommandos

Wenn in einem SqlJob mehrere Kommandos ausgeführt werden, kommt es vor, dass verschiedene Kommandos voneinander abhängen bzw. aufeinander aufbauen. Um Abhängigkeiten abzubilden, können jedem Kommando beliebige Vorgänger zugeordnet werden.
Im folgenden Beispiel soll das Kommando „B“ erst ausgeführt werden, nachdem Kommando „A“ beendet wurde.

```
<SqlJob
  <Command
    id="A">
  </Command>
  <Command
    id="B"
    predecessors="A">
  </Command>
…
```

Diese Funktionalität ist natürlich eine zwingende Voraussetzung für den Aufruf des Ergebnisses eines Kommandos in einem anderen Kommando über das Objekt „SQ_JOB.result()“.

3.7.4.4 Parallelisierung von Kommandos

Im SqlSurfer besteht die Möglichkeit, mehrere Kommandos parallel ausführen zu lassen. Dazu sind folgende Voraussetzungen erforderlich:

- Ausreichende Anzahl an Threads und Datenbankverbindungen in den Preferences
- Kein Konflikt mit den Abhängigkeiten zwischen einzelnen Kommandos

3.7.5 Exporter

Zum Exportieren von Ergebnissen aus der Datei sqljob_out.xml stehen im SqlSurfer verschiedene Exporter zur Verfügung.
An dieser Stelle soll auf eine Beschränkung bei Dateinamen von Microsoft Office-Dateien hingewiesen werden. Bei den vom SqlSurfer erstellbaren Microsoft Office-Dateien dürfen die Anzahl der Zeichen im Dateinamen bestimmte Grenzen nicht überschreiten. Andernfalls tritt ein Fehler auf und die Datei kann nicht mehr gelesen werden.
Dabei ist die Gesamtlänge von Pfad und Dateiname einschließlich Dateinamen-erweiterung zu beachten.

Beispiel: Der Pfad „C:\myFolder\myFile.xls“ beinhaltet 22 relevante Zeichen, denn es werden alle Zeichen mitgezählt.

Folgende Beschränkungen liegen z.B. bei Microsoft Office 2007 vor:

Programm	Maximale Anzahl Zeichen
Microsoft Excel	218
Microsoft Powerpoint	258
Microsoft Word	254

3.7.5.1 Export nach Excel

Für alle Exporter nach Excel oder in eine CSV-Datei kann der Name der Ausgabe-Datei festgelegt werden, wobei auch die Angabe der richtigen Dateiendung notwendig ist:

```
<SqlJob
  <Output
    mode="poi"
    outFile="output_file.xls"/>
```

Im Folgenden werden die Vorteile der einzelnen Excel-Exporter dargestellt.

3.7.5.1.1 POI

Der Exporter POI hat den Vorteil, dass er unter jedem Betriebssystem ohne zusätzliche Betriebssystem-abhängige Java-Bibliothek läuft. Er kann somit neben Windows auch problemlos unter Linux oder Unix verwendet werden.

3.7.5.1.2 JacoZoom

Der Exporter JacoZoom zeichnet sich dadurch aus, dass er Kommando für Kommando nacheinander per COM-Kommunikation an Excel schickt. Somit sind die Ursachen auftretender Fehler leichter identifizierbar.

3.7.5.1.3 Script

Der Vorteil des Exporters Script (zoomFast) liegt in der höheren Geschwindigkeit. Durch die schnellere Ausführung der Kommandos stehen aber wesentlich weniger Fehlerausgaben zur Verfügung. Es wird eine Art Batch Job erstellt, der dann komplett innerhalb von Excel gestartet wird.

3.7.5.1.4 Formatierungen

Für jeden Typ der oben genannten Exporter gibt es im SqlSurfer eine Vielzahl an Formatierungsmöglichkeiten.

a) Überschrift

Standardmäßig ist der SqlSurfer so eingestellt, dass die Überschrift der Datenspalten immer nach Excel exportiert wird. Dabei handelt es sich um die Namen der Datenbankspalten, die von einem SQL-ResultSet übernommen werden oder vom Benutzer selbst (über das Feld „ColumnData“) definiert wurden.
Diese Funktion lässt sich aber auch mit folgendem Feld ausschalten:

```
<SqlJob
  <Command
  <CommandJob/>
  <ExportData
    viewHeader="false"/>
…
```

b) Startzelle

Die zu exportierenden Daten können in einem Excel-Worksheet in eine beliebige Startzelle geschrieben werden. Dabei kommt die Startzelle die Überschrift der ersten Ergebnisspalte bzw., falls keine Überschrift exportiert wird, der Ergebniswert der ersten Zeile und Spalte.
Im folgenden Beispiel soll die Befüllung des Excel-Worksheets bei Zelle D4 beginnen:

```
<SqlJob
  <Command
  <CommandJob/>
  <ExportData
    startCell="D4"/>
…
```

c) Zeilensortierung

Es besteht die Möglichkeit, die Sortierung der Zeilen im ResultSet abzuändern. Dies geschieht über die kommaseparierte Angabe der Zeilenreihenfolge im Feld „rowResort“.
Im nachfolgenden Beispiel werden die ersten drei Ergebniszeilen in die erste, vierte und sechste Zeile des vorgesehenen Bereichs des Excel-Worksheets geschrieben.

```
<SqlJob
  <Command
  <CommandJob/>
  <ExportData
    rowResort="1,4,6"/>
…
```

Es müssen nur diejenigen Zeilen des ResultSets, von oben beginnend, angegeben werden, die umsortiert werden sollen. Alle weiteren Zeilen, die nicht explizit angeführt wurden, werden jeweils in die nachfolgende Zeile der letzten angegebenen Zeile platziert.

d) Transponieren

Neben der Umsortierung kann der SqlSurfer die Daten des ResultSets auch transponieren.
Dabei werden nach dem Umsortieren der Zeilen die Zeilen- und Spaltenpositionen vertauscht, d.h. die Daten werden an der Winkelhalbierenden gespiegelt.

```
<SqlJob
  <Command
  <CommandJob/>
  <ExportData
    transpose="true"/>
…
```

e) Zellformate und Leerwerte

Standardmäßig werden die im SqlSurfer definierten Zellformate beim Export nach Excel übernommen und auch bei Leerwerten die Zellen des Templates überschrieben. Diese beiden Funktionen können folgendermaßen ausgeschaltet werden:

```
<SqlJob
  <Command
  <CommandJob/>
  <ExportData
    fillFormat="false"
    fillEmpty="false"/>
…
```

f) Spaltenbeschreibung in Excel

Im SqlSurfer können außerdem für jede Spalte des ResultSets explizite Spaltenbeschreibungen für Zahlenformate in Excel definiert werden.
Dazu müssen zunächst im Kommando Datenbankspalten mit JDBC-Datentypen angelegt werden.
Anschließend kann für jede angelegte Datenbankspalte die entsprechende Ergebnisspalte in Excel formatiert werden. Als Spaltenname wird derselbe Name wie bei der Definition der Datenbankspalte verwendet. Als „numberFormat“ können jetzt die aus Microsoft Excel bekannten Zahlenformate gesetzt werden. Optional kann auch die Spaltenposition angegeben werden (ausgehend von der oben festgelegten Startzelle). Falls die Spaltenposition nicht verändert werden soll, wird der Default-Wert „-1“ verwendet. Das folgende Beispiel zeigt die Funktionsweise der Spaltenbeschreibungen und einige mögliche Zahlenformate

```
<SqlJob
  <Command
    type="none"
    cmdCode="1;2;3;4;5;06/19/2011;hello">
    <Column id="Column_A"    typeNum="4"    precision="1"/>
    <Column id="Column_B"    typeNum="3"    precision="3"     scale="2"/>
    <Column id="Column_C"    typeNum="3"    precision="1"/>
    <Column id="Column_D"    typeNum="3"    precision="7"    scale="2"/>
    <Column id="Column_E"    typeNum="2"    precision="7"    scale="2"/>
    <Column id="Column_F"    typeNum="91"/>
    <Column id="Column_G"    typeNum="12"   precision="40"/>
  <ExportData
    outputRef="excel"
    viewHeader="false">
    <ExcelColumn id="Column_A" numberFormat="Standard"/>
    <ExcelColumn id="Column_B" numberFormat="#,##0.00" columnPosition="2"/>
    <ExcelColumn id="Column_C" numberFormat="#,##0 ?" columnPosition="3"/>
    <ExcelColumn id="Column_D" numberFormat="#,##0.00 ?" columnPosition="5"/>
    <ExcelColumn id="Column_E" numberFormat="0.00%" columnPosition="6"/>
    <ExcelColumn id="Column_F" numberFormat="TT.MM.JJJJ" columnPosition="7"/>
```

```
    <ExcelColumn id="Column_G" numberFormat="@"    columnPosition="9"/>
  </ExportData>
</Command>
  …
```

Das exportierte Excel-Sheet für diesen SqlJob sieht nun folgendermaßen aus:

3.7.5.1.5 Templates

Beim Exportieren von Daten nach Excel kann auch ein Template herangezogen werden. Dieses Vorlage-Excel-Workbook kann aus mehreren Excel-Sheets bestehen kann. Das Workbook wird vor der Ausführung eines SqlJobs im Datenverzeichnis mit den gewünschten Formatierungen gespeichert und vom Exporter des SqlSurfers zur Befüllung der Excel-Sheets verwendet. Dazu sind folgende Felder auszufüllen:

```
<SqlJob
  <Command
  <CommandJob/>
  <ExportData
    sheet="templatesheet"/>
  </Command>
  <Output
    templateFile="template.xls"/>
…
```

Im Feld „sheet“ muss der Name des entsprechenden Worksheets angegeben werden, das als Vorlage verwendet wird. Ein Worksheet kann dabei von mehreren Kommandos befüllt werden.
Ein Template wird nie überschrieben, sondern der SqlSurfer kopiert es und speichert das befüllte Excel-Workbook unter einem anderen Namen ab.

3.7.5.1.6 VBA-Module

Mit dem SqlSurfer können nicht nur Zellen in Excel-Worksheets befüllt werden, sondern auch komplette VBA-Module erstellt und aufgerufen werden. In das Kommando des

SqlJobs wird dabei der Code eines beliebigen VBA-Moduls geschrieben. Über das Feld „fillVBAModul“ wird gesteuert, dass das Kommando als VBA-Modul exportiert wird.

```
<SqlJob
  <Command
    type="none"
      cmd="PUBLIC SUB vbaCode &amp;nl;  Range(&quot;A1&quot;).Value =
       &quot;Hello World!&quot;&amp;nl;END SUB">
  <CommandJob />
  <ExportData
    fillVBAModul="true"
    vbaModulName="testname"
    vbaModulCall="vbaCode"/>
</Command>
…
```

3.7.5.2 Export nach Powerpoint

Neben dem Export nach Excel unterstützt der SqlSurfer auch das Exportieren von Daten in Powerpoint-Dateien. Dabei erfolgt die Adressierung der Einfügestellen über Platzhalter im Template, da es keine Zellen wie in Excel gibt. In PowerPoint 2003 gibt es auch keine Möglichkeit die intern existierenden Namen der Objekte wie Slides, Charts oder ähnliches zu beschriften.

3.7.5.2.1 Platzhalter-Konzept

Platzhalter können im Template über einen globalen Namen in der Form „@name1@“ angesprochen werden. Global bedeutet dabei, dass dieser Name nur einmal in der gesamten PowerPoint-Präsentation enthalten sein darf.

Der Platzhalter „name1“ wird mit der Umklammerung von „@“ direkt in den Text geschrieben. Diese Ersetzung kann an folgenden Stellen und für folgende Objekttypen einer Präsentation erfolgen:

- Freitext
- Aufzählungen
- Powerpoint-Tabellen
- in den Zellen von eingebetteten OLE-Objekten wie Excel.Sheet.8, Excel.Chart.8 sowie im Datenblatt eines MSGraph.Chart.8

Dabei kann der Platzhalter an einer beliebigen Stelle im Text stehen, z.B. „Präsentation vom @PresentationDate@“. Der Platzhalter „@PresentationDate@“ wird mit dem Ergebnis eines SqlSurfer-Kommandos ersetzt. Soll ein „@“ in der fertig befüllten Präsentation stehen, kann dies mit „@@“ erreicht werden.

3.7.5.2.2 Output

Hat das Ergebnis des Kommandos mehrere Spalten und mehrere Zeilen, werden die Spalten und Zeilen auch entsprechend ausgegeben.

Bei Freitext und Aufzählungen werden die Spalten eines ResultSets durch Leerzeichen voneinander getrennt, und die Zeilen eines ResultSets durch jeweils neue Zeilen in Powerpoint abgebildet.
Bei allen tabellenartigen Powerpoint-Objekten wie Powerpoint-Tabellen und eingebetteten OLE-Objekten genügt es, die linke obere Ecke der gewünschten Stelle mit einem Platzhalter zu versehen. Die Ergebnis-Daten werden wie beim Export nach Excel in die benachbarten Zellen geschrieben.
Der Powerpoint-Exporter bietet bei einem ResultSet ebenfalls die Funktionen Transponieren, Zeilenumsortierung, Formatierungen durch Spaltenbeschreibung sowie Ausgabe einer Überschrift. Im Feld „startCell“ wird der Name des zu suchenden Platzhalters im Format „@name1@“angegeben.

```
<SqlJob
  <Command
    id="cmd1"
    type="none"
    cmd="a1;b2">
  <ExportData
      outputRef="ppt"
      startCell="@name1@"/>
  </Command>
  <Output
    id="ppt"
    mode="powerpoint"
</SqlJob>
```

3.7.5.3 Export in eine CSV-Datei

Neben dem Importieren von CSV-Dateien gibt es im SqlSurfer auch einen Exporter für CSV-Dateien. Als Default-Spaltentrenner wird dabei ein Semikolon verwendet, dieser kann jedoch auch geändert werden. Außerdem kann ein Ersetzungszeichen für den Spaltentrenner festgelegt werden, falls der Spaltentrenner selbst in den Daten vorkommt.
Es ist zu beachten, dass auch das „outFile“ entsprechend als CSV-Datei benannt wird.

```
<SqlJob
  <Output
    mode="csv"
    outFile="csvfile.csv"/>
…
```

3.7.5.4 Export in eine Datenbank

Neben dem Export nach Excel oder in eine CSV-Datei, kann der SqlSurfer Ergebnisdaten auch in eine Datenbank schreiben.

3.7.5.4.1 Unterstützte Datenbankdatentypen

In folgender Aufstellung ist eine Auswahl an Datenbankdatentypen aufgeführt, die vom SqlSurfer unterstützt werden (JDBC Data Types):

Datentyp-Nummer	Datentyp	Beschreibung
2	Numeric	Numerischer Datentyp mit fester Genauigkeit und fester Anzahl von Dezimalstellen
3	Decimal	Numerischer Datentyp mit fester Genauigkeit und fester Anzahl von Dezimalstellen
4	Integer	Ganzzahliger Datentyp
12	Varchar	Zeichendatentyp für Daten mit variabler Länge
91	Date	Datumsdatentyp

Eine vollständige Liste aller JDBC Data Types findet sich in der Dokumentation zu JDBC von Oracle. Jeder Hersteller einer Datenbank dokumentiert auch die Liste der unterstützten Typen.

3.7.5.4.2 Definition von Datenbankspalten

Bei der Ausführung eines SQL-Statements erhält die Java-Applikation SqlSurfer nicht nur ein ResultSet mit Ergebnissen zurück, sondern darüber hinaus auch ergänzende Informationen, sog. Metadaten. Diese geben beispielsweise Auskunft über Anzahl und Namen der Spalten oder die maximale Anzahl an Zeichen pro Spalte. Die Metadaten werden vom SqlSurfer automatisch u.a. für die Erstellung der Ergebnisspalten verwendet.

Falls der SqlSurfer diese Metadaten nicht für die Definition der Ergebnis-Datenstruktur heranziehen soll, gibt es die Möglichkeit, die zur Ausgabe benötigten Datenbankspalten eigenhändig zu definieren. Hierzu erhält jede Spalte einen Namen, einen Datentyp in numerischer Angabe (JDBC Data Type) sowie (abhängig vom jeweiligen Datentyp) die Anzahl der gesamten Stellen und Nachkommastellen. Dies soll folgendes Beispiel mit drei verschiedenen Spalten verdeutlichen:

```
<SqlJob
  <Command
    type="none"
  <Column
    id="Spalte_A"
    typeNum="12"
    precision="20"/>
  <Column
    id="Spalte_B"
    typeNum="4"
    precision="7"/>
  <Column
    id="Spalte_C"
    typeNum="3"
    precision="9"
    scale="2"/>
  <CommandJob />
  </Command>
…
```

3.7.5.4.3 Datentypkonvertierungen zwischen verschiedenen Datenbanken

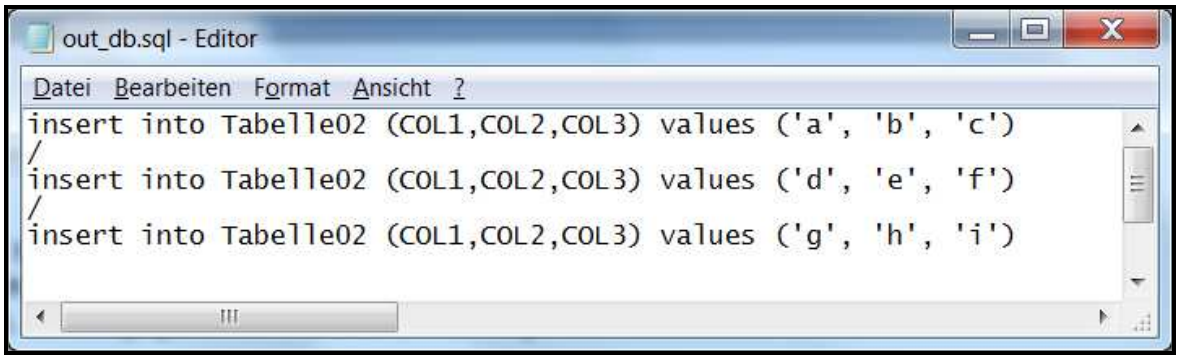
Der SqlSufer konvertiert bei Bedarf die Typen der Datenbankspalten automatisch. Dabei wird immer der Typ verwendet, der den Datentyp der Ausgangsdatenbank vollständig abbildet, aber noch möglichst speziell ist.

3.7.5.5 Export nach SQL-Scripts

Im vorherigen Kapitel wurde beschrieben, wie Daten in eine Datenbank exportiert werden können. Dabei legt der SqlSurfer die erforderlichen Tabellen eigenständig an und befüllt sie durch entsprechende SQL-Statements.
Daneben gibt es die Möglichkeit, die erzeugten „INSERT INTO“-Befehle nicht direkt ausführen zu lassen, sondern über den Ausgabemodus „dbsql“ als SQL-Datei zu exportieren. Dies zeigt folgendes Beispiel:

```
<SqlJob
  <Command
    cmdCode="select Col1, Col2, Col3 from Tabelle01"
    sqlConnection="derby">
  <CommandJob/>
  <ExportData
    outputRef="db"
    tableName="Tabelle02"/>
  </Command>
  <Output
    id="db"
    mode="dbsql"
    exportConnection="derby"/>
…
```

Es wird hier vorausgesetzt, dass die aufgeführten Spalten in der Tabelle01 existieren und gefüllt sind (in diesem Fall mit den Buchstaben von „a“ bis „i“, in drei Datensätzen). Das Ergebnis, die ausgegebene SQL-Datei, sieht folgendermaßen aus:



Dieses SQL-Statement kann nun problemlos zu einem späteren Zeitpunkt oder auf einer anderen Datenbank ausgeführt werden.

3.7.5.6 Erstellung von Zip-Dateien

In den SqlSurfer wurde eine Funktion integriert, die die Ausgabe-Dateien in eine Zip-Datei packt. In dieser Datei befinden sich nach der Durchführung des SqlJobs die Excel-Ausgabe-Datei sowie, falls diese exportiert werden sollen, die Kommando- und Log-Dateien in komprimierter Größe.
Falls ein Laufverzeichnis verwendet wird, wird die Zip-Datei darin abgelegt, ansonsten wird sie im Datenverzeichnis gespeichert.
Die Zip-Funktion wird über folgendes Feld aktiviert, indem ein Name für die zu erstellende Zip-Datei eingegeben wird:

```
<SqlJob
    zipFile="job_zipping_test.zip">
…
```

3.7.5.7 Erneutes Exportieren

Wenn ein SqlJob fertig gerechnet und die Datei sqljob_out.xml erstellt wurde, kann ein erneuter Export der Ergebnisse über den Aufruf „sq export sqljob.xml“ gestartet werden.
Diese Funktion findet Anwendung, wenn zum Beispiel die erzeugte Excel-Datei verloren geht oder versehentlich geändert wurde. Somit kann die Excel-Datei wiederhergestellt werden, ohne dass der gesamte SqlJob nochmal gerechnet werden muss.

Dieser Modus ist nur verwendbar, falls alle Ergebnisse auch in der [Name]_out.xml Datei zwischengespeichert wurden.

3.8 Anwendungsmöglichkeiten

Nachdem eine Vielzahl an wichtigen Funktionen des SqlSurfers erläutert wurde, werden in diesem Kapitel verschiedene Anwendungsmöglichkeiten des SqlSurfers aufgezeigt.

3.8.1 Erstellen von Reports

Eine der grundlegenden Möglichkeiten zur Verwendung des SqlSurfers ist das Erstellen von Reports. Es können dabei alle oben dargestellten Funktionen genutzt werden, wie zum Beispiel der Zugriff auf verschiedene Datenbanken, die Mechanismen zur Benennung der Ausgabe-Dateien, die Verwendung von Makros und Parametern oder die Formatierungsmöglichkeiten beim Export nach Excel. Der Funktionsumfang sowie die hohe Flexibilität von Excel bleiben dabei stets erhalten.

3.8.2 Übertragen von Daten zwischen Datenbanken

Mit dem SqlSurfer lassen sich auf einfache Weise Daten von einer Datenbank in eine andere kopieren. Dazu werden lediglich zwei entsprechende Datenbankverbindungen in den Preferences benötigt.
Im folgenden Beispiel werden die Daten dreier Spalten aus der Tabelle „h2_tab“ über die Datenbankverbindung „h2_con“ ausgelesen und über die Datenbankverbindung „derby_con“ in die automatisch erzeugte Tabelle „derby_tab“ geschrieben:

```
<SqlJob
  <Command
    cmd="select col_1, col_2, col_3 from h2_tab"
    sqlConnection="h2_con"
  <ExportData
    outputRef="derby"
    tableName="derby_tab"
    tableCreate="true"
    tableClear="true"/>
  </Command>
  <Output
```

```
    id="derby"
    mode="db"
    exportConnection="derby_con"/>
  <Output
    id="h2"
    mode="db"
    exportConnection="h2_con"/>
…
```

3.8.3 Starten von SqlJobs aus einem anderen SqlJob

Der SqlSurfer bietet die Möglichkeit, aus einem SqlJob heraus andere SqlJobs aufzurufen und zu starten. Die aufgerufenen SqlJobs werden dabei auf die übliche Art und Weise ausgeführt. Ein solcher Aufruf wird im folgenden Beispiel dargestellt:

```
<SqlJob>
  <Command
    type="none"
    cmdCode="Begin RUN
            ${SQ_SCH.run(&apos;job_1.xml&apos;,&apos;.&apos;)}
            ${SQ_SCH.run(&apos;job_2.xml&apos;,&apos;.&apos;)}
            End RUN"
    sqlConnection="derby">
  <CommandJob />
  </Command>
</SqlJob>
```

Durch diesen Aufruf werden die SqlJobs “job_1.xml” und “job_2.xml” nacheinander ausgeführt.

Außerdem ist es durch diese Funktion möglich, Parameter an aufgerufene SqlJobs weiter zu geben. Mit folgendem Aufruf wird der Parameter „param1“ auf den Wert „MyParameter“ gesetzt. Dieser wird an den SqlJob „batchjob_1.xml“ weitergegeben und der aufgerufene SqlJob wird ausgeführt.

```
<SqlJob>
  <Command
    type="none"
    cmdCode="${SQ_SCH.param(&apos;param1&apos;,&apos;MyParameter&apos;)}
       ${SQ_SCH.run(&apos;batchjob_1.xml&apos;,&apos;.&apos;)}"
    sqlConnection="derby">
    <CommandJob />
  </Command>
</SqlJob>
```

3.8.4 Einbindung in Java-Applikationen

Der SqlSurfer kann auf einfache Weise in andere Java-Applikationen eingebunden werden.

3.8.5 Erstellen von Datenbanksnapshots

Eine weitere sehr interessante Anwendungsmöglichkeit ist das Erstellen von Datenbanksnapshots. Dabei handelt es sich um eine Art der Datensicherung einer Datenbank zu einem bestimmten Zeitpunkt.
Der SqlSurfer kann von einer kompletten Datenbank ein textbasiertes Backup erstellen, indem ein SqlJob den Inhalt der Datenbank selektiert und als XML-Datei (sqljob_out.xml) speichert. Wenn nun eine Datenbank wiederhergestellt werden soll, muss nur der Export von der Datei sqljob_out.xml erneut gestartet werden. Mit den entsprechenden Einstellungen befüllt der SqlSurfer automatisch alle zuvor selektierten Tabellen und stellt den Datenbestand zum Zeitpunkt des Datenbanksnapshots wieder her.

3.9 Entwicklerspezifische Themen

3.9.1 Behandlung verschiedener Betriebssysteme

Der SqlSurfer wurde so konzipiert, dass er auf allen gängigen Architekturen gleichermaßen lauffähig ist. Auch das Handling verhält sich auf verschiedenen Betriebssystemen analog.
Der einzige Unterschied unter Linux und Solaris besteht darin, dass statt den Excel-Exportern „zoom“ und „zoomFast“ (script) automatisch der Exporter „poi“ verwendet wird.

3.9.2 Speicher-Management

Wenn ein speicheraufwändiger SqlJob ausgeführt wird und der standardmäßig von Java zur Verfügung gestellte Speicher nicht ausreicht, kann eine Datei mit dem Namen „sq.exe.vmoptions“ anlegt werden und über diese Argumente an die Java-VM übergeben werden:

- Xms<size> set initial Java heap size
- Xmx<size> set maximum Java heap size
- Xss<size> set java thread stack size

Beispiel: Xmx800M: setzt den maximalen Speicher auf 800 MB.

3.9.3 Einbinden eigener Skriptsprachen

Der SqlSurfer bietet die Möglichkeit, in Java implementierte Engines für beliebige Skriptsprachen zum Verarbeiten von Daten einzubinden. Der Einsatz eigener Skriptsprachen fügt sich dabei nahtlos in das Konzept des SqlSurfers ein.
So können vorhandene Skripte verwendet werden, die in einer externen Datei vorliegen, aber auch dynamisch zur Laufzeit erzeugt werden. Hierbei steht dem Anwender die gesamte Flexibilität des SqlSurfers zum Beschreiben von Eingangsdaten, zum Erzeugen von Kommandos sowie zum Exportieren der entstehenden Ausgangsdaten zur Verfügung.

3.9.3.1 Arbeitsweise prinzipiell

Das Evaluieren eines Skriptes S stellt im Prinzip eine Transformation von Eingangsdaten E zu ggf. geänderten Ergebnisdaten E' dar.

$$E \times S \rightarrow E'$$

Das Skript wird auf jeden Datensatz der Eingangsdaten E angewendet, modifiziert ggf. Daten und erzeugt auf diese Weise die Ergebnisdaten E'.

3.9.3.2 Voraussetzungen

Um eine eigene ScriptEngine im SqlSurfer nutzen zu können, müssen folgende Voraussetzungen erfüllt sein.

3.9.3.2.1 Konformität zum Java-Scripting-API

Die eigene ScriptEngine muss dem API des JDK-Package „javax.script“ [http://download.oracle.com/javase/6/docs/api/javax/script/package-summary.html] genügen, wobei hiervon nicht alle Interfaces implementiert sein müssen. Folgende Methoden stellen jedoch das Minimalset dar und müssen zwingend implementiert sein.

a) ScriptEngineFactory
   - getExtensions
   - getScriptEngine

b) ScriptEngine
   - createBindings()
   - get(String key)
   - getBindings(int scope)
   - getContext()
   - put(String key, Object value)
   - setBindings(Bindings bindings, int scope)
   - setContext(ScriptContext context)

c) Compilable
   - compile(String script)

d) CompiledScript
   - getEngine()
   - eval(Bindings bindings)

e) ScriptContext

3.9.3.2.2 Einbinden der eigenen ScriptEngine

Die eigene ScriptEngine muss in ein JAR kompiliert werden und das StdScritiong API von Java unterstützen. Das Kopieren in das autoload Verzeichnis macht die Scriptsprache verwendbar.

3.9.3.2.3 Registrierung der eigenen ScriptEngine im ScriptEngineManager

Die Registrierung im ScriptEngineManager stellt die Verbindung zwischen dem Bezeichner der Skriptsprache und der zugehörigen ScriptEngine her. Der ScriptEngineManager entscheidet anhand dieser Registrierung, welche ScriptEngine für eine gegebene Skriptsprache zur Laufzeit geladen wird.
Die Registrierung erfolgt über den Service-Provider-Mechanismus von Java [http://download.oracle.com/javase/6/docs/api/javax/script/package-summary.html#package_description]
[http://download.oracle.com/javase/1.4.2/docs/guide/jar/jar.html#Service%20Provider].

3.9.3.2.4 Implementierungshinweise

Rückgabe des eval-Aufrufs: Bei der Verwendung eigener ScriptEngines im SqlSurfer ist der Benutzer in der Regel an der Transformation der Eingangsdaten E und damit letztlich an den Ergebnisdaten E' interessiert. Neben diesen Daten liefert der Aufruf der eval-Methode eine Rückgabe vom Typ Object. Der SqlSurfer kann diese Rückgabe zu Informationszwecken ausgeben, weshalb die eval-Methode so implementiert sein sollte, dass sie einen Java-String liefert.

3.9.3.3 Script übergeben oder erzeugen

Ein auszuführendes Skript wird als eigenes Kommando gekapselt. Damit stehen verschiedene Möglichkeiten zur Verfügung, ein Skript bereitzustellen:

a) In Datei vorhandenes Skript benutzen: Im Reiter ‚Cmd' des Kommandos kann die Datei angegeben werden, die als Skript geladen werden soll.
b) Fertiges Skript als Text: Im Reiter ‚Cmd' des Kommandos kann das fertige Skript angegeben werden.
c) Generieren des Skripts zur Laufzeit: Im Reiter ‚Makro' des Kommandos kann ein Makro angegeben werden, welches mit den bekannten Mechanismen (Variablen, Kontrollstrukturen usw.) zur Laufzeit das auszuführende Skript erzeugt.

3.9.3.4 Script einbinden

Das Skript arbeitet als Transformation auf Eingangsdaten. Folglich wird es in einem eigenen Kommando eingebunden.
Dieses Kommando muss eine Abhängigkeit auf das Kommando erhalten, welches das Skript in einem früheren Skript erzeugen (oder laden) soll.

- **Eingangsdaten**
  Die Eingangsdaten, auf denen das Skript operiert, werden mit den bekannten Mechanismen des Kommandos beschrieben (SQL-Abfrage, Datenbankverbindung, Makro-Reiter zum Erzeugen zur Laufzeit, Cmd-Reiter zum Vorgeben usw.).

- **Skript**
  Das Skript wird im Reiter ‚Datenbankspalten' referenziert. Im Feld ‚Language Prg' muss die Skriptsprache angegeben werden, in der das Skript vorliegt. Hierrüber

wird die ScriptingEngine festgelegt, die das Skript evaluieren wird. Das Skript-erzeugende Kommando wird im Feld ‚Kommando Prg' angegeben.

- **Ergebnisdaten**
  Das Skript wird zur Laufzeit auf jeden Datensatz der Eingangsdaten angewendet. Dabei werden grundsätzlich alle Spalten der Eingangsdaten in die Ergebnisdaten übernommen. Ein Filtern ist jedoch möglich. Dazu werden die Spalten, die in die Ergebnisdaten übernommen werden sollen, in der Aufzählung ‚Spalte in der DB' explizit mit Name und Typ und ggf. Wertebereich angegeben.

  Die so erzeugten Ergebnisdaten können mit den bekannten Mechanismen eines Kommandos exportiert werden.

4 Referenzteil

Der Referenzteil dient der Vervollständigung der Ausführungen im Benutzerhandbuch.

4.1 Beschreibung der XML-Datenformate

In diesem Abschnitt werden alle möglichen Felder der Preferences, des SqlJobs und der externen Parameter-Datei aufgeführt.

4.1.1 Preferences

Preferences	
tmpDir	In diesem Verzeichnis werden Zwischenfiles abgelegt (keine Ergebnisse).
maxExecThreads	Dieser Parameter bestimmt die maximale Anzahl von Prozessen, die im Scheduler bei der Berechnung der Jobs verwendet werden
log_sql	Dieser Parameter bestimmt den Detailierungsgrad für Log-Meldungen von SQL-Befehlen. Es gibt die Belegungen: ▪ f: Fatale Fehler ▪ e: Fehler ▪ w: Warnungen ▪ i: Informationsmeldungen ▪ d: Debug-Meldungen ▪ t: Trace-Meldungen
log_con	Dieser Parameter bestimmt den Detailierungsgrad für Log-Meldungen von Datenbankverbindungen (Belegungen siehe log_sql).
<SqlConnection/>*	In diesem Tag wird die Beschreibung einer Datenbankverbindung zusammengefasst. Er kann mehrfach eingetragen werden.
id	Eindeutiger Name für eine Datenbankverbindung
driver	Name der Javaklasse, die den JDBC Treiber enthält, z.B. ▪ Derby: driver="org.apache.derby.jdbc.EmbeddedDriver" ▪ Oracle: driver="oracle.jdbc.driver.OracleDriver" ▪ Sybase: driver="com.sybase.jdbc3.jdbc.SybDriver" ▪ MySql: driver="com.mysql.jdbc.Driver" Um einen neuen Treiber einzubinden, muss die Treiber-jar-Datei mit in das Installationsverzeichnis kopiert werden. Das Jar wird in die .bat-Datei eingetragen, und darin wird der Klassennamen des Treibers angegeben.
url	Die Zugriffsreferenz/Verbindungsdaten für die Datenbank, z.B. ▪ Derby: url="jdbc:derby:dbname;create=true" ▪ Oracle: url="jdbc:oracle:thin:@hostname:1521:SID" ▪ Sybase: url="jdbc:sybase:Tds:hostname:8000/dbname" ▪ MySql: url="jdbc:mysql://hostname:3306/dbname"
user	Der Username für die Datenbankverbindung
passwd	Das Passwort der Datenbankverbindung im Klartext.

	Dieses Feld kann alternativ zu passwdCrypt angegeben werden.
passwdCrypt	Das verschlüsselte Passwort der Datenbankverbindung. Dieses Feld kann alternativ zu passwd angegeben werden. Es wird nur verwendet falls das Feld passwd leer ist. Das Setzen des verschlüsselten Passwortes erfolgt am besten über die GUI.
<SessionProperty/>*	Parameter auf Session-Ebene für diese Datenbankverbindung
id	Name des Parameters
value	Wert des Parameters
<FileAssociation/>*	Zuordnung von Dateitypen auf Viewer/bzw. Editoren
id	Pattern für die Extension in der Form *.txt
type	Modus bzw. Quelle der Zuordnungslogik: ▪ system: Zuordnung des Betriebssystems ▪ explicit: Explizite Angabe über file und cmd
file	Pfad und Dateiname des Editors oder Viewers (nur bei type=explicit)
cmd	Struktur des Shell-Befehls. %0 wird mit dem Pfad und Dateiname des Editors ersetzt und %1 wird mit dem Dateinamen, der geöffnet werden soll vorbelegt. Beispiel und Vorbelegung: “%0“ “%1“.
maxConnections	Dieser Parameter gibt die Anzahl der maximal gleichzeitig offenen Verbindungen mit dieser JDBC Datenbank an.

4.1.2 SqlJob

<SqlJob/>	Ein SqlJob, dessen Kommandoliste übersetzt, ausgeführt und ggf. exportiert werden kann.
id	ID bzw. Name des SqlJobs
runname	Name des Laufes. Dieser Name wird zur Bezeichnung und Unterscheidung verschiedener Berechnungen des SqlJobs verwendet.
rundir	Unterverzeichnis (relativ zu dataDir), in dem die Ergebnisdaten abgelegt werden. Bei einem leeren rundir werden die Daten im Verzeichnis dataDir abgelegt.
exportCmds	true oder false. Bei true wird jedes fertig übersetzte Kommando zusätzlich unter dem Dateinamen out_<id>.<ext> im Filesystem abgelegt.
exportLog	true oder false. Bei true wird die Datei out.log erzeugt, in der alle Logeinträge gespeichert werden.
zipFile	Falls dieses Feld einen Wert ungleich leer enthält, wird eine Zip-Datei mit allen Ausgabedateien, ggf. den exportCmds und dem exportLog erstellt. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden. Sollte das Kommado einen Fehler enthalten wird die Verarbeitung fortgesetzt und out.zip als Default verwendet.
status	Der Status des SqlJobs. Dieser kann folgende Ausprägungen haben: ▪ init: Initialisiert ▪ run: Rechnend ▪ ready: Fertig berechnet

	Das Feld wird automatisch bei der Berechnung eines SqlJobs gesetzt.
duration	Berechnungsdauer des Reports. Dieses Feld wird automatisch bei der Berechnung gesetzt.
planduration	Geplante Berechnungsdauer des Reports. Dieses Feld wird automatisch vom letzten Lauf des Reports übernommen.
startTime	Startzeitpunkt des SqlJobs
endTime	Zeitpunkt der Fertigstellung des Reports
<Parameter/>*	Eine beliebige Anzahl von Parametern, die bei jedem Rechenlauf neu belegt werden können.
paramFile	Externe Datei mit Parameterlegungen. Ist das Feld leer, werden die Parameterbelegungen aus dem SqlJob-Tag „Parameter“ verwendet. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
<Macro/>*	Eine bel. Anzahl von Makros, mit Argumenten und einem Makro-Body
<Command/>*	Eine bel. Anzahl von Kommandos
<Output/>*	Eine bel. Anzahl von Ausgabe-Dateien, die beim Ausführen oder Exportieren in das Ausgabeverzeichnis geschrieben werden.
cmdformat	Datenformat der cmd-Attribute: ▪ attr: Speicherung als Attrubut ▪ elem: Speicherung als Element (ohne CDATA Umgebung) ▪ cdata: Speicherung als Element (mit CDATA Umgebung) Die Attribute cmd, cmdCode und cmdScript von Command und body von Makro können als attribute, Emenete oder Elemente mit CDATA umgebung geseichert werden. Beim Laden werden alle der drei Formate unterstützt. Es darf aber nur eines verwendet werden.
descformat	Datenformat alle descr Felder. Die desc Felder des SQLJob, Parameter, ParameterSelection, Makro und Command können in drei verschiedenen Formaten gespeichert werden. ▪ attr: Speicherung als Attribut ▪ elem: Speicherung als Element (ohne CDATA Umgebung) ▪ cdata: Speicherung als Element (mit CDATA Umgebung) Beim Laden wird jedes der drei Formate erkannt. Es darf aber nur eines verwendet werden.
attrformat	Datenformat der Attribute. Attribute können in einer Zeile oder mehrzeilig dargestellt werden. ▪ oneline: Alle Attribute alle Felder werden in einer Zeile dargestellt ▪ mixline: Kleine Elemente werden in einer Zeile dargestellt ▪ moreline: Jedes Attribut wird in einer eigenen Zeile dargestellt
defaultformat	Beim Speichern können entweder alle Felder in die Datei gespeichert werden, oder nur diejenigen die sich von Voreinstellungswert unterscheiden. ▪ diff: Es werden nur die veränderten Attribute gespeichert ▪ all: Es werden immer alle Attribut gespeichert
encoding	Encoding

<ImportData/>	
file	Pfad und Name der einzulesenden Datei
maxScanLines	Anzahl der maximal zu lesenden Zeilen (zur Identifizierung der Spalteninformationen)
fileformat	Format der zu lesenden Datei (.csv oder .xls)
sheet	Name eines Excel-Worksheets, das gelesen wird
startCell	Startzelle, an der das Einlesen beginnen soll
readHeader	Soll Überschrift gelesen werden?
readTypes	Sollen Datentypen explizit eingelesen werden? ▪ Datentypen-Angaben werden aus der 1. oder 2. Zeile der Datei gelesen ▪ Zusätzliche Datei mit Datentypen wird gelesen
importencoding	Encoding für CSV-Import
lineSeparator	Zeilentrenner
columnMode	▪ Variable Zeichenanzahl ▪ Gleiche Zeichenanzahl
columncountcheck	▪ Genau gemäß Datenstruktur ▪ Weniger erlaubt ▪ Weniger erlaubt, mehr falls leer ▪ Weniger erlaubt, mehr werden ignoriert
columnSeparator	Spaltentrenner
columnMarker	Textbegrenzungszeichen
columnMarkerQuoting	Ersetzung Textbegrenzungszeichen
ScanColumn	Zusatzbeschreibung für gelesene Spalten
id	Name der Spalte
datatype	Numerische Angabe des Datentyps (JDBC Data Type), z.B.: ▪ varchar: 12 ▪ numeric: 2 ▪ decimal: 3 ▪ integer: 4 ▪ date: 91
precision	Anzahl der gesamten Stellen
scale	Anzahl der Nachkommastellen
numberFormat	Format der Zelle aus Excel, z.B.: ▪ String: @ ▪ Währung: #,##0.00 € ▪ Prozent: 0.00% ▪ Datum: dd/mm/yyyy ▪ Nummer: #,##0.0000
columnPosition	Spaltenposition

<Output/>	Eine Ausgabedatei kann ein Excelsheet, eine CVS Datei, ein Script oder der Export in eine Datenbank sein.
id	ID bzw. eindeutiger Name des Outputs
mode	Modus bzw. Art der Ausgabe ▪ poi: Es wird ein Excelsheet mit der Bibliothek poi erstellt ▪ zoom: Es wird ein Excelsheet mit der Bibliothek jacoZoom erstellt ▪ zoomfast: Es wird ein Excelsheet in einer Kombination von Script und jacoZoom befüllt. ▪ csv: Es wird eine CSV-Datei erstellt

	▪ db: Die Ergebnisse werden in eine Datenbank übertragen ▪ dbsql: Es wird ein SQL-Script zur direkten Ausführung in der Datenbank erstellt.
directExecute	true oder false. Bei true werden die Ergebnisse sofort in das Ausgabemedium übertragen. Es erfolgt keine Zwischenspeicherung. Bei false werden die Ergebnisse zwischengespeichert und werden erst beim Export übertragen.
viewResult	true oder false. Bei true wird die Ausgabedatei in einem passenden Editor geöffnet, nachdem sie erstellt wurde.
outFile	Dateiname der zu erstellenden Datei. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden. Sollte das Kommando einen Fehler enthalten wird die Verarbeitung fortgesetzt und out_${id}.<Extension> als Default verwendet.
templateFile	Dateiname der Template-Datei, die als Vorlage zur Befüllung verwendet wird. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden. Sollte das Kommando einen Fehler enthalten, wird kein Template verwendet.
exportConnection	Name der Datenbankverbindung aus den Preferences, die zum Speichern der Daten verwendet wird. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
columnSeparator	Wird der Export in eine CSV-Datei geschrieben, wird dieses Feld als Trenner für die einzelnen Spalten verwendet.
replaceSeparator	Sollte die Zeichenfolge des Trenners in einem Feld vorkommen, wird es durch diese Zeichenfolge ersetzt.

<ExportData/>	
outputRef	Referenz auf ein OutputFile. Die ID des Output-Files wird hier eingetragen. Dieses Feld muß gefüllt sein.
sheet	Name des Excel-WorkSheets, in das die Daten exportiert werden. Existiert ein Sheet mit diesem Namen nicht, wird ein neues Sheet angelegt und ggf. mit dem Template-Sheet vorbelegt. Ist das Feld leer oder tritt ein Fehler bei der Berechnung auf, wird der Name des Kommandos verwendet. Ist auch dieser leer, wird der Name ‚Sheet' verwendet. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
sheetTemplate	Name eines Excel-Worksheets, das als Vorlage verwendet wird. Das Sheet wird kopiert und unter dem Namen des Feldes <sheet> abgelegt. Wird kein Template-Sheet angelegt, wird dieser Kopierprozess übersprungen. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
viewHeader	true oder false. Bei true wird eine Header-Zeile über dem Datenfeld angezeigt. Die Namen werden aus dem SQL-Kommando übernommen. Bei CSV-Dateien wird nach Belegung

	die Header-Zeile mit in die Datei geschrieben.
fillFormat	true oder false. Bei True wird auch das Format der Zelle im Excelsheet gesetzt. Die Voreinstellung ist true.
fillEmpty	true oder false. Bei True werden auch leere Zellinhalte in das Excelsheet geschrieben. Dadurch wird der vorherige Inhalt überschrieben. Die Voreinstellung ist true.
columnSeparator	Wird der Export in eine CSV-Datei geschrieben, wird dieses Feld als Trenner für die einzelnen Spalten verwendet.
replaceSeparator	Sollte die Zeichenfolge des Trenners in einem Feld vorkommen, wird es durch diese Zeichenfolge ersetzt.
startCell	Diese Zelle enthält die Excelzelle im Format <Spaltenbuchstabe><Zeilennummer> (z.B. A3), ab der die Daten eingefügt werden sollen. Ist das Feld leer, werden die Daten ab A1 (ohne Header) bzw. ab A2 mit Header eingefügt. Bei transponierten Tabellen mit header wird B1 verwendet. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
rowResort	Eine kommaseparierte Liste von Zeilennummern, die beschreibt, in welche Zeile die i-te Zeile geschrieben wird. Ist das Feld leer oder sind nicht genügend Einträge in der Liste, wird die Position nicht geändert. (Bsp.: Bei „1,3,4“ werden die ersten 3 Ergebniszeilen in die erste, dritte und vierte Zeile des Excel-Worksheets geschrieben.)
transpose	true oder false. Bei true werden nach dem Umsortieren der Zeilen die Zeilen- und Spaltenpositionen vertauscht. D.h. die Daten werden an der Winkelhalbierenden gespiegelt.
tableName	Beim Export in eine Datenbank enthält dieses Feld den Tabellennamen. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
commitAfterRows	Dieser Parameter wird nur beim Exportieren in eine Datenbank verwendet. Er gibt die Anzahl der Zeilen an, nach denen jeweils ein Commit durchgeführt wird. Wird eine 0 angegeben, wird erst nach dem Einfügen aller Datensätze ein Commit durchgeführt. Ein Wert von 1 setzt nach jedem Datensatz ein Commit ab. Alle Werte größer 1 fügen jeweils die gewünschte Anzahl der Zeilen als batch in die Datenbank ein und setzen ein Commit ab.
fillVBAModul	true oder false. Bei true werden keine Zellen befüllt, sondern ein VBA-Modul importiert bzw. aufgerufen und ggf. wieder gelöscht.
vbaModulName	Falls dieses Feld befüllt ist, wird das Kommando als VBA-Modul unter diesem Namen in das Excel-Workbook exportiert (nur bei ZOOM). Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
vbaModulCall	Beim Exportieren wird die VBA-Procedure oder VBA-Funktion mit dem Namen dieses Feldes aufgerufen. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
tableCreate	true oder false. Bei true wird vor dem Export eine Tabelle angelegt. Dieses Feld wird nur beim Export in eine Datenbank verwendet.
tableClear	true oder false. Bei true wird die gewählte Tabelle vorher geleert. Dieses Feld wird nur beim Export in eine Datenbank verwendet.
<ExcelColumn>*	In einer Zusatzbeschreibung können Zusatzinformationen wie

	Formate oder Spaltenpositionen angegeben werden.
id	Name der Spalte
numberFormat	Format der Zelle in Excel: ▪ String: @ ▪ Währung: #,##0.00 € ▪ Prozent: 0.00% ▪ Datum: dd/mm/yyyy ▪ Nummer: #,##0.0000
columnPosition	Hier kann angegeben werden, in welcher Excel-Spalte die jeweilige Spalte des ResultSet platziert werden soll. Bsp.: bei columnPosition "3" werden die Daten in Excel in die dritte Spalte von links (ausgehend von der festgelegten Startzelle) geschrieben. Falls die Reihenfolge der Spalten nicht geändert werden soll, ist hier (der Defaultwert) "-1" einzutragen.

<Parameter/>	
id	Name der Spalte
type	Datentyp des Parameters: ▪ String: ▪ Nummer: ▪ DateTime: ▪ Einzelauswahl:
val	Wert des Parameters
label	Beschriftung des Parameters
<ParameterSelection/>*	Ein bel. Anzahl von Auswahlkombinationen
id	ID der Auswahlkombination
label	Beschriftung der Auswahlkombination
descr	Beschreibung der Auswahlkombination
selectionCmd	Makro zur Berechnung der Auswahlkombinationen. Dies kann über $SQ_JOB.directsql(…), als bedingte Liste oder beliebige Makro erfolgen. Jeder Zeile ist eine Kombination. In der Zeile wird id, label und descr mit ; getrennt. Falls dieses Feld befüllt ist wird <ParamterSelection/> ignoriert.
descr	Beschreibung des Parameters

<Macro/>	
id	ID bzw. Name des Makros
cmdLang	Verwendete Template Engine (freemarker, velocity)
args	Kommaseparierte Liste von Argumenten
body	Rumpf des Makros in der unter cmdLang angegebenen Programmiersprache
descr	Beschreibung

<Command/>	
id	ID bzw. Name des Kommandos
type	Typ des Kommandos. Es gibt folgende Belegungsmöglichkeiten: ▪ sqlquery ▪ sqlexec ▪ script

	▪ none ▪ import
cmdLang	Name der Makrosprache. Standardmäßig sind freemarker und velocity verwendbar.
cmdCode	Programmcode in der unter cmdLang angegebenen Programmiersprache
cmdScript	Fertig aufgelöstes Script, wie es an den Compiler übergeben wird.
cmdFile	Kommando aus externer Datei. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
cmd	Fertig übersetztes Kommando
envVariables	Für den type script: Komma-separierte Liste von Umgebungsvariablen, die im Script zur Verfügung stehen
scriptCall	Aufruf des Script mit Argumenten
scriptName	Dateiendung bzw. kompletter Dateiname
envOS	Einschränkung der Script-Ausführung auf das angegebene Betriebssystem. Folgende Ausprägungen sind möglich: ▪ Windows NT
sqlConnection	Name der Datenbankverbindung, die zum Ausführen von SQL-Befehlen verwendet wird. Das Feld muss bei den Typen sqlexec und sqlquery gesetzt sein. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden. Tritt beim Compilieren eine Fehlermeldung auf, wird die Connection auf default gesetzt.
maxFetchedRows	Die maximale Anzahl von Datensätzen, die aus der Ergebnismenge eines SQL-Befehls ausgelesen wird. "-1" bedeutet unbegrenzt. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden. Tritt beim Compilieren eine Fehlermeldung auf, wird der Wert auf 100 gesetzt.
predecessors	Komma-separierte Liste von IDs anderer Kommandos, die abgearbeitet sein müssen, bevor dieses Kommando übersetzt wird. (Vorgänger-Kommandos)
execCond	Ist dieses Feld leer, wird das Kommando bei der Ausführung übersprungen. Enthält es einen Text, wird es ausgeführt. Der Text wird als Ausführungsanzeige verwendet. Es können alle Parameter oder Makros des SqlJobs zum Compilieren verwendet werden.
error	Fehlertext, falls ein Fehler bei der Ausführung auftritt.
<ImportData/>	Ein bel. Anzahl von Informationen über den Import von Daten
<Column/>	Ein bel. Anzahl von Informationen über die Spalten von Ergebnissen aus SQL-Befehlen

Id	Name der Spalte
typeNum	Numerische Angabe des Datentyps (JDBC Data Type), z.B.: ▪ varchar: 12 ▪ numeric: 2 ▪ decimal: 3 ▪ integer: 4 ▪ date: 91
Precision	Anzahl der gesamten Stellen

Scale	Anzahl der Nachkommastellen
result	Ergebnis des SQL-Befehls, falls das Ergebnis zwischengespeichert wird. Die Spalten sind immer mit ";" separiert. Die Zeilen sind mit einem \n getrennt.
currentFetchedRows	Die Anzahl der tatsächlich gelesenen Datensätze.
<CommandJob/>	Statistische Informationen über die Ausführung eines Kommandos
<ExportData/>	Ein bel. Anzahl von Beschreibungen über den Export der Daten
desc	Beschreibung

<CommandJob/>	
…..	Diese Struktur liefert Logdaten beim Lauf, die für den nächsten Lauf als Statistik verwendet werden.

4.1.3 Externe Parameter-Datei

< ParameterValues/>	
<ParameterValue/>	Eine beliebige Anzahl von Paaren mit Parametername und Parameterwert
id	ID bzw. eindeutiger Name des Parameters
val	Wert des Parameters

4.2 APIs SQ_JOB und SQ_SCH

Bei einigen Aufrufen der APIs SQ_JOB und SQ_SCH können Argumente in runden Klammern angegeben werden. Diese Argumente sollten zwischen einfachen Apostrophen stehen, wie z.B. SQ_JOB.result('sql').

Compiler API (SQ_JOB):

Aufruf	Beschreibung
getVersion()	Version des SqlSurfers
getFullVersion()	Lite-Version oder Vollversion
result('sql')	liefert das Ergebnis des Kommandos mit dem Namen „sql“ zurück
resultList('command')	liefert das Ergebnis des Kommandos „command“ als Vektor zurück
getName()	Name des aktuellen Reports
getTmpFileName ('prefix', 'suffix')	liefert einen temporären Dateinamen zurück. Dieser beginnt mit „prefix“ und endet mit „suffix“.
getDBConnectionString ('id')	liefert den Connection Url inkl. user und password (falls unverschlüsselt angegeben) sowie alle Properties zurück. Id ist der Name der SQLConnection.
getDBUrl('id')	liefert die URL zur Verbindung an eine Datenbank zurück. Id ist der Name der SQLConnection.
getDBDriver('id')	Liefert den Treiber einer Datenbankverbindung zurück. Id ist der Name der SQLConnection.
getDBConnectionName()	Name der Datenbankverbindung

getDBConnectionInfo()	alle Informationen der Datenbankverbindung
--- getParams()	Parameter als Liste mit Namen und Wert
--- getParamValue ('paramId')	Wert des Parameters “paramId”
--- getParamList()	ResultSet mit allen Parametern in der Form: name1=wert1, name2=wert2, usw.
--- getFullParamList()	ResultSet mit allen Parametern und allen Attributen
getRunName()	neue Version von getName
getDir()	Verzeichnis des Sqljobs
getCommandName()	Name des aktuellen Kommandos
getExcelVersion()	Aktuelle Excel-Version
getExcelExtension()	Aktuelle Excel-Erweiterung
getBSName()	Betriebssystemname
getCalcUser()	User, unter dem die Berechnung durchgeführt wurde
getExportUser()	User, unter dem der Export durchgeführt wurde
getStartTimeStamp()	Zeitpunkt zu dem die Berechnung gestartet wurde
getFinnishTimeStamp()	Zeitpunkt zu dem die Berechnung beendet wurde
getStatus()	Status der Berechung
getErrors()	alle aufgesammelten Fehler der Berechnung
getCalcEngine()	Verwendete scripting engine, z.B. velocity, freemarker
getErrorFlag()	Kennzeichnungen ob ein fehler aufgetreten ist
--- getExternalParamFlag()	Kennzeichen, ob eine externe Parameter-Datei geladen wurde
getParamResultIds()	Alle Parameter in datenorientierter Form, z.B. zum Kopieren in das Result
getParamResultLabels()	Alle Parameter in darstellungsorientierter Form, z.B. zum Kopieren in das Result

Scheduler API (SQ_SCH):

FreeMarker:	
param('name', 'value')}	Setzt den Wert des Parameters „name“ auf „value“
run('filename')}	Startet die Verarbeitung des SqlJobs in der Datei „filename“. Der Scheduler verwendet alle gesetzten Parameter.
run('filename', 'viewParameters', 'onlyExport')	Startet die Verarbeitung des SqlJob in der Datei filename. Alle Parameter werden zuvor in einem Dialogfenster zur Bearbeitung angezeigt, falls viewParameters auf true gesetzt wird. Mit onlyExport kann gesteuert werden, ob der Report gerechnet und exportiert wird, oder nur exportiert wird. Der Scheduler verwendet alle gesetzten Parameter.
export('filename')	Exportiert die Ergebnisse des SqlJob in der Datei filename. Es wird die Eingabedatei, nicht die Out-Datei, angegeben.
close()	Diese Funktion wartet bis alle Jobs fertig abgeschlossen sind.
closeAllConnections()	Schließt alle Datenbankverbindungen die idle sind
closeConnections('name')	Schließt alle Connections der Datenbankverbindung mit der Bezeichnung name. Es werden nur idle Verbindungen geschlossen.
getSystemInfo('key')	Liest Systemeigenschaften über Java aus. Key kann mit folgende Namen belegt werden:

	file.encoding,file.encoding.pkg,file.separator,line.separator,path.separator, java.class.path,java.class.version, java.endorsed.dirs,java.ext.dirs,java.home, java.io.tmpdir,java.library.path, java.runtime.name,java.runtime.version, java.specification.name, java.specification.vendor, java.specification.version, java.vendor, java.vendor.url, java.vendor.url.bug, java.version, java.vm.info, java.vm.name, java.vm.specification.name, java.vm.specification.vendor, java.vm.specification.version, java.vm.vendor, java.vm.version, os.arch, os.name, os.version, sun.arch.data.model, sun.boot.class.path, sun.boot.library.path, sun.cpu.endian,sun.cpu.isalist, sun.desktop, sun.io.unicode.encoding, sun.java.launcher, sun.jnu.encoding, sun.management.compiler, sun.os.patch.level, user.country, user.dir, user.home, user.language, user.name, user.timezone, user.variant

4.3 FreeMarker Built-ins

In folgender Tabelle steht eine Auswahl an Funktionen, die in FreeMarker als Built-ins verwendet werden können:

Datentyp	Built-ins			
String	Substring chop_linebreak html js_string left_pad number split word_list	cap_first date, time groups last_index_of right_pad replace starts_with xhtml	uncap_first datetime index_of length contains rtf trim xml	capitalize ends_with j_string lower_case matches url upper_case
Number	String	round	floor	ceiling
Date	String	date	time	datetime
Boolean	string			
Sequence	First seq_last_index_of sort	last sort_by	seq_contains reverse chunk	seq_index_of size
Hash	keys	values		
Seldom used / expert built-ins	byte, double, float, int, long, short number_to_date, number_to_time, number_to_datetime eval	has_content	interpret	is_...

	namespace new

Für genauere Informationen zu den einzelnen Built-ins wird an dieser Stelle auf den entsprechenden Beitrag im FreeMarker Manual verwiesen:
http://freemarker.org/docs/ref_builtins.html

4.4 Logging Meldungen

Der gesamte Ablauf des SqlSurfers wird über Logmeldungen protokolliert. Diese sind in verschiedene Kategorien eingeteilt. Für jede Kategorie lässt sich der Detailierungsgrad separat in den Preferences einstellen.

Folgende Kategorien und Levels sind verfügbar:

Kategorien	Level	Beschreibung
SQL		Informationen über SQL-Befehle, die an Datenbanken gesendet werden.
	ERROR	Alle Fehler, die beim Ausführen von SQL-Befehlen entstehen.
	WARN	Ggf. sind nicht alle Datentypen der Datenbanken originär unterstützt. Diese werden automatisch konvertiert und eine Warnung ausgegeben.
	INFO	Bei Batch-Jobs wird die Anzahl der eingefügten Datensätze angezeigt.
	DEBUG	Alle SQL-Kommandos werden, so wie sie an die Datenbank geschickt werden, ausgegeben.
	TRACE	Bei Batch Inserts werden alle inserts mit geloggt.
CON		Informationen über Datenbankverbindungen mit Treibern sowie deren Auf und Abbau
	ERROR	Fehler mit Datenbanktreibern, Fehler beim Aufbau / Schließen von Verbindungen oder bei den Einstellungen der Verbindungen in den Preferences
	WARN	Warnung bei automatischer Umstellung des Exporters auf POI, weil JacoZoom auf dem Betriebssystem nicht läuft
	INFO	Verbindungsauf- und Abbau
	DEBUG	Informationen über Resourcenprüfung, Infomationen über Treiberversionen
	TRACE	Informationen über den Status von Verbindungen
SCH		Informationen über den Scheduler
	ERROR	Fehlermeldungen des Schedulers, wenn zum Beispiel ▪ ein Job abgebrochen wird, ▪ in den Abhängigkeiten der Kommandos eines Jobs eine Schleife ist oder ▪ ein Kommando sich selbst als Vorgänger enthält
	INFO	Informationen über die Einstellungen in den Preferences (Anzahl Threads, Anzahl Connections der DB Verbindungen), Anzeige der aktuell laufenden Jobs und der noch nicht abgeschlossenen Jobs
	DEBUG	Anzeigen des Ausführungsplanes gemäß Abhängigkeiten
	TRACE	Anzeigen der Ressourcenallokationsversuche

APL		Applikationsmeldungen
	ERROR	Informationen über nicht implementierte Features, Hilfe kann nicht angezeigt werden
	INFO	Programmstart und Programmende, Infozeile mit Version und Betriebssystem
	DEBUG	Lizenzangabe, Lizenzregistrierungen
PRG		Programmfortschritt
	ERROR	Job abgebrochen, Fehler bei Dateiverknüpfungen
	WARN	Warnung, wenn Parameter-Datei fehlerhaft oder unauffindbar, leere Output-Referenzen
	INFO	Beginn und Ende der Jobs, Anzahl Kommandos im Job, Liste der Parameter mit den aktuellen Werten eines Jobs
	DEBUG	Beginn und Ende der einzelnen Kommandos der Jobs
	TRACE	Meldung über leere SQL-Kommandos
FLE		Informationen über erstellte Dateien, die gespeichert werden.
	ERROR	Fehler beim laden oder Speichern von Dateien die auf Dateisystemprobleme zurückzuführen sind.
	WARN	Warnungen, falls Dateien nicht mit den gewünschten Dateinamen angelegt werden konnten (z.B. existierende Datei ist gelockt).
	INFO	Erstellte Datei gespeichert, Datei existiert bereits (alte Version wird umkopiert)
STA		Statistiken über die Ausführungszeiten
	INFO	Ausführungszeiten der einzelnen Jobs
	DEBUG	Prognostizierte Ausführungszeiten eines Jobs
	TRACE	Ausführungszeiten der einzelnen Kommandos eines Jobs
EXP		Exporter
	ERROR	Fehler beim Laden / Speichern von Workbooks, Excel Template-Sheet nicht gefunden, JacoZoom-Fehler, Zelle nicht im Zellbereich, Fehler bei VBA-Modulen
CPL		Informationen des Compilers
	ERROR	Fehler beim Compilieren, Setzen von Defaults aufgrund falscher Formate, fehlende Scripting Engines
	WARN	Parameter nicht gefunden
SHL		Shellbefehle
	ERROR	Abbruch von Shellkommandos
	INFO	Shell-Ausgaben
	DEBUG	Ausgabe von geladenen Shell-Parametern
API		Automation
	ERROR	Kommando nicht gefunden oder noch nicht gerechnet
	INFO	Ergebnis von Kommando kopiert

4.5 Fehlermeldungen

Code	Beschreibung
SQ-00100	Das Feld sqlconnection eines Kommandos verweist auf eine SqlConnection, die in den preferences nicht existiert
SQ-00101	Das Kommando ist vom Typ sqlquery oder sqlexec. Entsprechend muss die SQLConnection gesetzt werden.
SQ-00102	Das Session Property kann nicht gesetzt werden. Ggf. unterstützt der JDBC-Treiber dieses Feature nicht.
SQ-00600	Beim Speichern eines Excelsheet unter dem angegebenen Namen ist ein Fehler aufgetreten. Ggf. ist die Datei durch einen anderen Prozess gesperrt.
SQ-00601	Für die Dateiendung der angegebenen Datei wurde keine Programm-Verknüpfung in Windows angelegt. Entweder muss ein entsprechender Viewer installiert werden, oder in den Einstellungen von Windows eine entsprechende Verknüpfung erstellt werden. Entsprechend wurde sie nicht geöffnet.
SQ-00602	Beim Starten der Datei-Verknüpfung ist ein Fehler aufgetreten.
SQ-00603	Der Job {0} ({1}) wurde mit folgendem Fehler abgebrochen:{2}
SQ-00700	Beim Ausführen des SQL Kommandos ist folgender Fehler aufgetreten: {0}
SQ-00701	Beim Abschließen des SQL-Batch Befehles ist folgender Fehler aufgetreten:{0}
SQ-00702	Beim Anlegen der Tabelle {0} ist folgender Fehler aufgetreten:{1}
SQ-00703	Beim Löschen der Tabelle {0} ist folgender Fehler aufgetreten:{1}
SQ-00704	Beim Leeren der Tabelle{0} ist folgender Fehler aufgetreten:{1}
SQ-00705	Beim Einfügen von Daten in die Tabelle {0} ist folgender Fehler aufgetreten: {1}
SQ-00706	Bein Aufbauen der Datenbankverbindung ist folgender Fehler aufgetreten:{0}
SQ-00707	SQ-00707: Fehler beim Absetzen eines Commit Befehles:\n{1}
SQ-00708	SQ-00708: Die Konvertierung {0} für die Spalte {1} mit Wert {2} war fehlerhaft.\{3}
SQ-00800	Die Datenbankverbindung {0} wurde nicht in den Preferences gefunden.
SQ-00801	Der JDBC-Treiber der Datenbankverbindung {0} wurde nicht gefunden.
SQ-00802	SQ-00802: Fehler beim Schließen der Datenbankverbindungen:{0}
SQ-00803	SQ-00803: Beim Prüfen der Datenbankverbindung {1} ist folgender SQL-Fehler aufgetreten:{2}
SQ-00900	Der Job wurde abgebrochen.
SQ-00901	Der Job {0} wurde abgebrochen.
SQ-00902	Die Abhängigkeiten enthalten eine Scheife. Die Abhängigkeit zu {1} wird entfernt.
SQ-00903	Das Kommando enthält sich selbst in den Abhängigkeiten. Die Abhängigkeit wurde entfernt.

4.6 Tutorial

Zur praktischen Einführung der Anwender in die Bedienung des SqlSurfers wurde ein Tutorial entwickelt, das zu den wichtigsten Funktionen des SqlSurfers möglichst einfache kleine Beispiele bietet die genau einen Teil exemplarisch darstellen.
Das Tutorial ist in die Installationsdatei integriert und befindet sich nach der Installation im gewählten Installationsverzeichnis.

4.7 Versions-Änderungsnachweis

Im Änderungsnachweis werden für jede Version die vorgenommenen Änderungen protokolliert.

Version	Beschreibung
0.63	• Erstellung einer Dokumentation • Beispiel mit gleichen Spaltennamen in einem gejointen select eingebaut • Zusätzliche Checks, ob der SQL-Connection-Name gesetzt ist bzw. in den Preferences existiert
0.64	• SessionProperties für die Datenbankverbindungen neu eingebaut • BugFix: Auslesen des Resultset verwendet jetzt die Aliasnamen und nicht mehr die blanken Namen der Tabellenspalten
0.65	• Fast Excel Exporter als Testversion
0.66	• FastFill,fillFormat,fillEmpty neu eingebaut
0.67	• Die Datentypen der Exporter sind komplett überarbeitet • Der zoom Fast Exporter ist verwendbar
0.68	• dbExporter unterstützt BatchInsert • Create Table speziell für oracle überarbeitet
0.69	• Hack um für sybase varchar die Länge auszulesen
0.70	• File Associations über explizite Liste in den Preferences möglich
0.71	• File Logging auf Log4J umgestellt. Dies ist eine noch halb fertige Version des Loggings
0.72	• Alle Logkategorien sind auf log4j umgestellt
0.73	• Die Logs weiter überarbeitet, EXE-Version, Bugfixes bzgl. Logkategorien und Losing-References
0.74	• Fehlerbehebung beim Zeilenumsortieren der Excel-Exporter • Überarbeiten der Dokumentation
0.75	• Bereichschecks des Excel-Zoom fast Exporter verbessert • Der Exporter Zoom fast bricht bei leeren Time und Timestamp-Feldern nicht mehr mit einer Fehlermeldung ab. • Das Number-Format von NUMERIC, DECIMAL, DOUBLE und FLOAT berücksichtig die Nachkommastellen. • Die Batchaufruf-Option sq export ermöglicht es, einen fertig gerechneten SqlJob noch einmal zu exportieren. • In der GUI gehen die Referenzen auf andere Objekte nicht mehr verloren (bei Referenzen auf Output und bei Referenzen auf Vorgänger und Nachfolger)
0.76	• Das Feld sqlconnection kann auch ein Makro sein. • Aktualisierter Oracle Jdbc Treiber. Es ist die Version 11g Release 2 integriert.
0.77	• Build Zip File nicht mehr als Ant Job sondern in Java Klasse • Log der Parameter • V-Scroller bei diversen Widgets eingebaut • Verwendung des Sybase Treibers des Aqua-Studios

0.78	• GUI für Run SQL Job eingebaut • Verschiedene Elemente für Statistiken eingebaut • Verwendung des _out.xml Laufes zur Statistik verwenden
0.79	• Fehlerbehebung im ZOOM-Exporter beim Ausgeben einer Fehlermeldung • CSV-Exporter auf Numberformat US umgestellt
0.80	• Die Option fastfill bei Output-Elementen wurde durch zwei zusätzliche Auswahlmöglichkeiten fastfill und dbsql in dem Attribut mode bei Command ersetzt. • Defaultregelung bei Attribut sheet aus ExportData überarbeitet. • Fehler im ZOOM_FAST Exporter behoben. Es werden auch mehr als 255 Zeilen angezeigt. Die range-Checks wurden bei den Excel-Exportern überarbeitet.
0.81	• Schreibfehler im Language-Support-Deutsch behoben • Fehler im Scriptbasierten Zoom Exporter für Datum behoben • Zusätzliche Logs in der Kategorie FLE Level TRACE, um File-Umbenennungen oder Locks zu prüfen.
0.82	• SWT auf SWT 3.6 umgestellt • PDF Viewer eingebaut
0.83	• Build obf deutlich verbessert
0.84	• XML4J durch Xerces ersetzt • SQ_OBF.exe klappt jetzt
0.85	• Logging von Velocity in die Datei velocity.log durch einen Parameter ausgeschaltet • Fehlermeldungen für Schleifen in de Scheduler eingebaut
0.86	• Lite Version als test
0.87	• Lite Version mit sq_run
0.88	• Default des Kommandotypen ist jetzt sqlquery • Der Default für die sqlconnection eines Kommandos wurde auf „“ geändert.
0.89	• Builds für Linux und Solaris eingebaut • Neue Fehlermeldung in CompilerAPI falls aList leer • Neue Sequentialisierung für die light Version (gemäß Dependencies) • Nur tags die ungleich den Default-Werten sind werden geschrieben • Java Xmx Option auf 1,2 GB gesetzt (bei allen aufrufen) • Logmeldung bei LogJob Status verbessert
	• Wait Problem bei der parallelen Verabeitung klappt
0.90	• Zwei neue Attribute eingebaut: runname und rundir • Umstellung auf dataDir mit relative Pfadangabe für alle Berechnungen • Gui für Rundialog überarbeitet • Die Stattistiken werden in <infile>_stat.xml abgelegt und auch wieder von dort verwendet. • Fehlerfix bei Type-Converter bei Übergabe von Datentyp Date mit Wert null. • Leere SQL-Kommandos erzeugen keine Fehlermeldung mehr • Fehlerfix im Logger bei decide
0.91	• Scripting API für ResultSet Ersetzungen • Fehlerbehebung: Die Parameterdatei ist wieder ladbar, das Zip wird wieder erstellt und die Templates werden richtig gezogen. • Die Vorbelegung für den Laufnamen enthält jetzt eine lesbare Nummer als Timestamp • Die Umbenennungsdateien (wenn Ausgabedateien schon bestehen) enthalten jetzt mehr Underscores und sind besser lesbar. • Der Call VBA Teil ist überarbeitet, es gibt ein neues Attribut fillVBAModul. • Der Dialog ExportData enthält jetzt funktionierende Abhängigkeiten

	• MaxFetchedRows auch als kompilierbarer Ausdruck verwendbar • Alle printStacktrace an die Konsole durch richtige Fehlermeldungen ersetzt • Die Meldung SQ-0707 eingebaut • Die Java-Klasse CommandList in SqlJob umbenannt
0.92	• Integer Fehler im Excel Befüller gefixed (0000044) • Schreibweise SqlSurfer (0000042) • Fehlermeldung im Batch-Aufruf gefixed (0000035) • VBA Modulname (0000036) getestet • \n Behandlung bei Result Tag des Commandos geändert. Es werden nur mehr zwischen den einzelnen Zeilen \n eingefügt und nicht mehr am Ende. • Erweiterte Fehlermeldungen SQ-00708,SQ-00803 • Fehlermeldung bei nicht verfügbaren Tmp-Directory • Zykluserkennung komplett überarbeitet • Abhängigkeiten werden nur mehr über Vorgänger erkannt. Dies ist eine nicht aufwärtskompatible Syntaxänderung der XML-Spezifikation eines XMLJobs. Alle Nachfolgebedingungen müssen als Vorgängerbedingungen kodiert werden.
0.92.1	• Sqlsurfer_lib.jar wird mit geladen • Icon überarbeitet • Felhermeldungen für vba Codemodule laden verbessert
0.92.2	• Excel Export in einem ‚remotesicheren' Prozess • RunDir wird bei jedem Export in der GUI neu berechnet
0.92.3	• CVSExporter gibt keine Fehlermeldung mehr bei leeren Datums aus • SimpleDateFormat ist nicht threadsave, Dies wird jetzt richtig unterstützt
0.92.4	• TinyInt werden in derby jetzt als smallint abgebildet • Resultsets werden ohne Fehlermeldung ‚ResultSet is closed' geschlossen
0.92.5	• CloseAllConnections und CloseConnections sind am Scheduler API implementiert
0.92.6	• Ausgabe von Shellausgaben step by step ausgeben und auch Errors loggen • Beim Export in einer Datenbank wird das drop table, create table und delete from table ggf auch in eine Datei exportiert, wenn exportCmds angewählt ist.
0.92.7	• Die isClosed Methode zum Schließen der DB Connection wird nicht mehr verwendet, da offensichtlich einige Datenbank-Treiber sie nicht richtig implementieren bzw. jeder Treiber eine andere Exception wirft. • Die ExportConnection in einem OutputFIle kann auch als Ausdruck formuliert werden. • GetMesage bei JNIExceptions gibt „“ zurück. Entsprechend ist zur Analyse eine fortlaufende Nummer eingebaut. • GetSystemInfo in Scheduler API eingebaut
0.92.8	• Fehlermeldung für jacoZoom Fehler verbessert • Das Compiler API um Meldungen für DB-Informationen erweitert.
0.92.9	• Umleiten des errorStreams bei Shellaufrufen auf den Outputstream. Damit sollten Shellaufrufe mit kombinierten error und outstreams klappen. • Die Funktionen getParamResultLabels, getParamResultIds implementiert
0.92.10	• Das Loglevel der FLE Logging Kategorie kann jetzt richtig einstellt werden • Das Beschreibungfeld in im sq gui Dialog kann mit cut&paste ausgelesen werden • Das Feld cmdFile kann auch als freemarker Expression beschrieben werden. • BugFix im RunWindow (Division by Zero)
0.92.11	• BugFix: Falls die Spalten bei Berechenmodus nichts nicht vollständig gefüllt sind, gib der DB Exporter keine Fehlermeldung mehr aus. • Bugfix: Der DB-Exporter gibt jetzt keine Fehlermeldung mehr aus, wenn eine

	Tabelle, die gelöscht werden soll, nicht existiert. • Das neue Attribut execCond eines Kommandos ermöglicht über eine Funktion zu steuern, ob das Kommando ausgeführt wird, oder übersprungen wird. • Die Logmeldung ‚Display Parameters' der Kategorie PRG ist von debug auf Info Level angehoben. • Einige Log Meldungen verbessert.
0.92.12	• Logmeldungen für offene Connections und übersprungene Kommandos verbessert. • BugFix Kommandos überspringen verwaltet die Connections jetzt richtig
0.92.13	• Export Von Flisskommazahlen die null sind klappt auch mit dem Scriptbefüller • Der Exporter für Word und PowerPoint als Dummyklassen implementiert • XML Format als CDATA noch nicht aktiv
0.93	• XML Format mit CDATA • XML Pretty printing des XML Formates
0.93.1	• XML Loader mit Exceptionsprüfung • XML Loader mit Prüfung, dass ein Feld nicht als Attribut und gleichzeitig als Element geladen wird.
0.93.2	• Vorbelegung für attrformat ist jedes Attribut in neue Zeile
0.93.3	• Encoding für die WIndows-Console unterstützt jetzt cp850 encoding für windows. D.h. Umlaute werden richtig angezeigt. • Icon für ActionHelp • ArgumentHandling –param auf Kommandozeile • Ansatz den Import von CSV Datei zu bauen • Genswt als anttask • ImportDialog mit rudimentärem CSV und Excel Importer • .DLL und .LIC für jacoZoom aus resource laden • Bugfix freemarker cache gefixed, bugfix parameterübergabe an batch jobs
0.93.4	• Konsolidieren der Version
0.93.5	• Viele Bugfixes in GIU mit DataDir • Kommand-Template Mechanismus eingebaut um Kommandos aus einem Template zu generieren
0.93.6	• Neue Kategorie IMP • Einige Bugfixes im Excel Importer • Vorschäge bei den SQLConnections • Allocate Resources für ExportConnections eingebaut. • Close Excel bei ExcelImport und Runner.import einbebaut
0.93.7	• Attribut file in import file geändert. Importfile kann jetzt auch freemarker ersetzungen. • deleteTemplateFile eingebaut um templates automatisch zu löschen. • Rehenfolge beim add und move von Excelsheets verbessert (bei add noch noch nicht ok) • Delete statt truncate bei derby und h2 • Dependancies werden auch bei Komand-Templates mit generiert • Change preferences klappt jetzt in der Gui klappt jetzt • Disable menu items klappt • Der Joblogger loggt nur in das richtige Fenster
0.94	• SQ_JOB.directsql() eingebaut. Es wird ganz normal über connections gestartet • selectionCmd in den Parametertyp Auswahl eingebaut • SQlJob New Bugfix klappt, jetzt auch ein save danach
0.95	• BugFix ScriptExporter für VBAmakros

	• Tutorial DerbyServermode eingebaut
0.95.4	• Diverse Bugfixes im Type Mapper für TinyInt und Byte
0.95.5	• PowerPoint @name@ Ersetzungen auch auf substrings von textranges • Fehlermeldungen verbessert
0.95.6	• Fehlermeldung für Zoom Exporter verbessert
	• Fehlermeldungen Script-Exporter verbessert
0.95.7	• Shell.bat für windows 10 angepasst • Felermeldung beim Öffnen Exel Exporter ermöglicht weiterlaufen der Applikation
0.95.8	• BugFix BigDecimal und TinyInt TypeKonverter
0.95.9	• Lizenzmechanismus eingebaut
0.95.10	• Lizentmechanismus auf loadfrom resource umgestellt
0.95.11	• Hilfe darstellen klappt
0.95.12	• Darstellung des Hilfemechanismus funktioniert
0.96	• Tests und kleine Änderungen im Typemapper
0.96.1	• Lizenzmechanismus mit aktuellen Keys
0.96.2	• Bugfix im Lizenzmechanismus

4.8 Fehlerdiagnose

4.8.1 Marko-Sicherheit in Excel

Für das externe Befüllen von Excel-Workbooks über den sqlSurfer sind ggf. die Zugriffsrechte auf das VBA-Projektobjektmodell einzustellen, damit der sqlSurfer die Erlaubnis erhält auf Excel zuzugreifen und ohne ständig wiederkehrende Sicherheitsfrage befüllen darf. Diese Einstellung ist je nach Windows Excel Version unterschiedlich einzustellen. Die Dokumentation von Excel gibt ggf. darüber genauere Auskunft wie man die Excel-Workbook-Befüllung durch Markos einschalten kann.

Je nach Office Version sind folgende Änderungen einmalig durchzuführen:

Excel 2010

In einer beliebigen geöffneten Excel Applikation unter

```
Datei->Optionen->Sicherheitscenter->Einstellungen für das
Sicherheitscenter->Einstellungen für Markos
```

Das Kontrollkästchen

```
Zugriff auf das VBA-Projektobjektmodell vertrauen
```

einschalten.

Excel 2013

In einer beliebigen geöffneten Excel Applikation unter

```
Datei->Optionen-> Einstellungen für das Trust Center->Makroeinstellungen
```

Das Kontrollkästchen

`Zugriff auf das VBA-Projektobjektmodell vertrauen`

einschalten.