



Managed User Jobs

JOB SCHEDULER

Dokumentation

Juli 2005

Impressum

Software- und Organisations-Service GmbH
Giesebrechtstr. 15
D-10629 Berlin

Telefon (030) 86 47 90-0
Telefax (030) 8 61 33 35
Mail info@sos-berlin.com
Web www.sos-berlin.com

Letzte Aktualisierung: Juni 2005

Inhaltsverzeichnis

1 Einführung	4
2 Installation	5
2.1 Job Scheduler	5
2.2 MySQL Prozeduren	5
2.3 MySQL UDF für UDP Datagramme	5
2.4 Konfiguration der Rechte	5
3 SQL Schnittstelle	7
3.1 JOB_SUBMIT	7
3.2 JOB_RUN	7
3.3 JOB_DELETE	8
3.4 JOB_SET_SCHEMA	8
3.5 JOB_SET_ACTION	8
3.6 JOB_SET_TITLE	8
3.7 JOB_SET_PRIORITY	9
3.8 JOB_SET_SUSPENDED	9
3.9 JOB_SET_NEXT_START	9
4 SQL Schnittstelle für Database Reports	10
4.1 REPORT_JOB_SUBMIT	10
4.2 REPORT_JOB_SET_MAILTO	10
4.3 REPORT_JOB_SET_MAILCC	10
4.4 REPORT_JOB_SET_MAILBCC	10
4.5 REPORT_JOB_SET_ASBODY	11
4.6 REPORT_JOB_SET_BODY	11
4.7 REPORT_JOB_SET_SUBJECT	11
4.8 REPORT_JOB_SET_STYLESHEET	11
4.9 REPORT_JOB_SET_PATH	11
4.10 REPORT_JOB_SET_FILENAME	12
4.11 SEND_MAIL	12

1 Einführung

Für Oracle gibt es das Paket `dbms_scheduler` bzw. `dbms_job` um Datenbankstatements automatisiert zu vorher geplanten Zeitpunkten ausführen zu lassen. MySQL bietet eine solche Möglichkeit nicht. An dieser Stelle setzen die Managed User Jobs an. Sie bieten die Möglichkeit, über eine SQL Schnittstelle (ähnlich der `dbms_scheduler` Schnittstelle von Oracle) Jobs zu definieren. Mit Hilfe eines externen Job Schedulers werden diese dann unter den Rechten des Nutzers, der sie eingestellt hat, ausgeführt.

Zusätzlich bieten die Managed User Jobs die Möglichkeit, eingetragene Empfänger per eMail über Erfolg oder Misserfolg ausgeführter Statements zu informieren. Für die Verwendung der Managed User Jobs gibt es folgende Voraussetzungen:

- Eine MySQL 5 Datenbank (mit Einschränkungen auch ab MySQL 4.1.1)
- Ein Benutzer für die Datenbank mit superuser Rechten
- Installation des `udf_sos_send_udp` shared object bzw. dll für den MySQL Server
- Installation des Job Schedulers mit Zugriff auf die Datenbank über den o.g. Nutzer
- Das MySQL Maintenance Jobs Paket für den Scheduler
- Der Scheduler muss so konfiguriert werden, dass der Job `JobSchedulerManagedUserJob` regelmässig (z.B. jede Minute) startet.
- Damit Datenbanknutzer eigene Jobs einstellen können, benötigen sie weitere Rechte. Siehe Installation.

2 Installation

Sollte noch keine MySQL Datenbank (am besten ab Version 5) vorhanden sein, so muss zunächst ein MySQL Server installiert werden. Siehe dazu die offizielle MySQL Dokumentation.

2.1 Job Scheduler

Der Job Scheduler muss nicht zwangsläufig auf demselben Rechner installiert werden, wie die Datenbank. Zur Installation kann das Installationsprogramm des Job Schedulers verwendet werden. Die Pakete "Database Support" und "MySQL Maintenance Jobs" müssen dabei aktiviert werden. Die benötigten Tabellen können vom Job Scheduler Installationsprogramm angelegt werden. Dafür müssen vorher ein Benutzer und ein Schema für den Scheduler eingerichtet sein.

2.2 MySQL Prozeduren

Nach der Installation des Job Schedulers müssen noch MySQL Prozeduren per Hand eingespielt werden. Das Script dafür liegt in `<scheduler_installationsverzeichnis>/db/mysql/procedures`. Durch folgende Schritte wird dieses eingepielt:

- Mit einem MySQL Client als Scheduler Benutzer mit dem MySQL Server verbinden
- Das Datenbankschema des Job Schedulers auswählen (z.B. `use scheduler`)
- Den Delimiter auf andere Zeichen umstellen: `delimiter //`
- Die Prozeduren einlesen: `source <pfad>/scheduler_job_procedure.sql`
- Dem Delimiter zurücksetzen mit: `delimiter ;`

2.3 MySQL UDF für UDP Datagramme

Im Verzeichnis `<scheduler_installationsverzeichnis>/lib/` wird eine User Defined Function zum Versenden von UDP Befehlen mitgeliefert. Die Installation ist nicht zwingend erforderlich, wird aber benötigt, wenn man die Prozedur `JOB_RUN` verwenden möchte.

Linux: Im Startscript des MySQL Servers muss die Variable `LD_LIBRARY_PATH` um den Pfad, in dem die Datei `udf_sos_send_udp.so` liegt, erweitert werden, z.B.

```
export LD_LIBRARY_PATH=/pfad/pfad:$LD_LIBRARY_PATH;
```

Danach ist ein Neustart des MySQL Servers erforderlich. Nach dem Neustart wird dem MySQL Server die Funktion bekanntgegeben mit `create function sos_send_udp returns string soname 'udf_sos_send_udp.so';`

Windows: Die Datei `udf_sos_send_udp.dll` muss ins bin Verzeichnis des MySQL-Servers kopiert werden. Danach ist ein Neustart des MySQL Servers erforderlich. Nach dem Neustart wird dem MySQL Server die Funktion bekanntgegeben mit `create function sos_send_udp returns string soname 'udf_sos_send_udp.dll';`

2.4 Konfiguration der Rechte

Das Ausführen der geplanten Statements wird vom Job Scheduler erledigt. Dazu legt dieser zur Laufzeit selbst neue User an. Zu diesem Zweck benötigt der Job Scheduler Benutzer superuser Rechte:

```
grant all on *.* to <scheduler_user>@<scheduler_host>
identified by <scheduler_password> with grant option
grant all on mysql.* to <scheduler_user>@<scheduler_host>
identified by <scheduler_password> with grant option
```

Jeder normale Benutzer der Datenbank, dem es erlaubt werden soll, eigene Jobs einzustellen, benötigt Zugriffsrechte für die Tabellen, die von seinen Jobs verwendet werden, sowie das Recht, die Prozeduren des Scheduler Users ausführen zu dürfen:

```
grant execute on scheduler.* to <user_name>@<user_host>
```

Damit darf der Benutzer eigene Jobs einstellen und auch nur seine eigenen modifizieren.

Für MySQL ab Version 4.1 und vor Version 5 entfällt das letzte Statement, da es keine Prozeduren gibt. Stattdessen müssen die Benutzer Zugriff auf die Tabell scheduler_managed_user_jobs bekommen:

```
grant all on scheduler.scheduler_managed_user_jobs to <user_name>@<user_host>
```

Dies setzt natürlich das Sicherheitskonzept ausser Kraft.

3 SQL Schnittstelle

Die Administration der Job funktioniert (ähnlich wie bei Oracle) direkt über eine SQL-Schnittstelle. Die hier Dokumentierten Prozeduren befinden sich im Schema des Scheduler Benutzers. Um sie zu verwenden muss man entweder vorher das entsprechende Schema auswählen (z.B. `use scheduler`) oder die Prozeduren qualifiziert aufrufen (z.B. `call scheduler.JOB_SUBMIT(...)`).

Um das Ergebnis des Prozeduraufrufs zu erfahren, haben alle Prozeduren einen ersten Parameter `result`, der einen Ergebnisstring aufnimmt, den man später auslesen kann:

```
call scheduler.JOB_SET_SCHEMA(@result, 1, 'test');
SELECT @result;
```

Möchte man einen Job bearbeiten und benötigt dafür mehrere API Aufrufe, so sollten diese in einer Transaktion durchgeführt werden. Ansonsten kann es passieren, dass ein Job bereits ausgeführt wird, bevor man ihn fertig bearbeitet hat.

3.1 JOB_SUBMIT

Paramter:

- OUT `job_result` VARCHAR(250)
- OUT `job_id` INT
- IN `job_action` TEXT
- IN `job_start_time` DATETIME
- IN `job_next_start` TEXT
- IN `job_schema` VARCHAR(250)

`JOB_SUBMIT` legt einen neuen Job an. Die zurückgegebene `job_id` identifiziert diesen Job und wird von anderen Prozeduren als Eingangsparameter benötigt. In `job_action` steht der auszuführende SQL-Befehl. Es können auch mehrere Befehle angegeben werden. Diese müssen durch Semikolon getrennt werden (siehe `JOB_SET_ACTION`). `job_start_time` gibt die Startzeit für den ersten Start des Jobs an. Ist sie null, wird der Job so bald wie möglich gestartet (aber erst nach dem commit). `job_next_start` gibt eine Regel an, um den nächsten Startzeitpunkt zu errechnen, wenn der Job gelaufen ist. Bei null läuft der Job nur einmal. Siehe dazu `JOB_SET_NEXT_START`. Mit `job_schema` wird das zu verwendende Datenbankschema angegeben. Es darf nicht null sein und der Benutzer muss die entsprechenden Rechte für diese Schema haben.

Beispiel: Anlegen eines Jobs

```
call scheduler.JOB_SUBMIT(@result, @jobid, 'UPDATE persondata SET age=age+1;', null, 'ADDTIME(NOW(), '1:0:0)'), 'test');
```

Der String '1:0:0' (eine Stunde) muss doppelt quotiert werden, da er sich in einem weiteren String befindet.

3.2 JOB_RUN

Paramter:

- OUT `job_result` VARCHAR(250)
- IN `job_id` INT
- IN `job_start_time` DATETIME

Übergibt einem Job sofort eine neue Startzeit oder startet ihn sofort, wenn `job_start_time` null ist. Dazu wird per UDP ein Befehl an den Job Scheduler, wofür die Installation der entsprechenden User Defined Function notwendig ist. Dieser Befehl sollte nicht innerhalb einer Transaktion mit anderen Befehlen verwendet werden, sondern nur einzeln, um zum Beispiel einen Job zu Testzwecken sofort zu starten.

Beispiel: Job 25 sofort starten

```
call scheduler.JOB_RUN(@result, 25, null);
```

3.3 JOB_DELETE

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT

Löscht den angegebenen Job.

Beispiel: Job 20 löschen

```
call scheduler.JOB_DELETE(@result, 20);
```

3.4 JOB_SET_SCHEMA

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN job_schema VARCHAR(250)

Setzt das Datenbankschema, in dem der Job ausgeführt werden soll.

Beispiel: Schema von Job 19 auf 'test' setzen.

```
call scheduler.JOB_SET_SCHEMA(@result, 19, 'test');
```

3.5 JOB_SET_ACTION

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN job_action TEXT

Setzt den auszuführenden Befehl eines Jobs. Es können auch mehrere Befehle angegeben werden. Diese müssen durch Semikolon getrennt werden.

Beispiel: mehrere Befehle für einen Job

```
call scheduler.JOB_SET_ACTION(@result, 19, 'UPDATE persondata SET age=age+1; UPDATE persondata SET weight=weight-1;');
```

3.6 JOB_SET_TITLE

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN job_title VARCHAR(250)

Setzt einen Titel für den Job.

Beispiel: Titel setzen

```
call scheduler.JOB_SET_TITLE(@result, 15, 'Testjob');
```

3.7 JOB_SET_PRIORITY

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN job_priority INT

Setzt die Priorität des Jobs. Jobs mit höherer Priorität werden bevorzugt ausgeführt, wenn nicht genügend Tasks verfügbar sind, um alle anstehenden Jobs gleichzeitig auszuführen. Default ist 1.

Beispiel: Priorität des Jobs 13 auf 5 setzen

```
call scheduler.JOB_SET_PRIORITY(@result, 13, 5);
```

3.8 JOB_SET_SUSPENDED

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN job_suspended BOOL

Aktiviert(3. Argument false) oder deaktiviert(3. Argument true) einen Job.

Beispiel: Job 16 deaktivieren

```
call scheduler.JOB_SET_SUSPENDED(@result, 16, true);
```

3.9 JOB_SET_NEXT_START

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN job_next_start TEXT

Setzt die Regel, um die nächste Startzeit zu errechnen. Diese muss ein SQL-Ausdruck sein, der ein ISO-Datum zurückliefert. Der Ausdruck wird immer dann evaluiert, wenn der Job ausgeführt wurde.

Beispiel: Job 10 jeden Tag zur selben Uhrzeit(alle 24h) nach Ablauf des ersten Jobstarts starten

```
call scheduler.JOB_SET_NEXT_START(@result, 10, 'ADDDATE(NOW(), 1)');
```

Beispiel: Job 10 einmal pro Stunde starten

```
call scheduler.JOB_SET_NEXT_START(@result, 10, 'ADDDATE(NOW(), "01:00:00")');
```

Beispiel: Job 10 alle 5 min starten

```
call scheduler.JOB_SET_NEXT_START(@result, 10, 'ADDTIME(NOW(), "0:5:0")');
```

Beispiel: Job 10 zu jeder vollen Stunde starten

```
call scheduler.JOB_SET_NEXT_START(@result, 10, 'ADDDATE(CURRENT_DATE(),  
INTERVAL EXTRACT(HOUR FROM CURRENT_TIME()+1 HOUR)');
```

Beispiel: Job 10 zum Ultimo jedes Monats um 18 Uhr starten

```
call scheduler.JOB_SET_NEXT_START(@result, 10, 'adddate(last_day(current_date()), interval 18 hour)');
```

Beispiel: Job 10 täglich um 5 Uhr starten

```
call scheduler.JOB_SET_NEXT_START(@result, 10, 'IF(HOUR(NOW())>=5,  
ADDDATE(CURRENT_DATE(), INTERVAL 24+5 HOUR),  
ADDDATE(CURRENT_DATE(), INTERVAL 5 HOUR));');
```

4 SQL Schnittstelle für Database Reports

Ist das Managed Jobs Paket installiert, können zusätzlich Datenbank Report Jobs erstellt werden. Diese funktionieren ähnlich wie die normalen Datenbank Jobs. Nach der Ausführung wird ein Report mit dem Ergebnis des letzten Select Statements per Email verschickt. Dies ist genauer in der Managed Jobs Dokumentation erklärt. Für die Report Jobs gilt die oben Beschriebene SQL Schnittstelle, sowie die folgenden Funktionen:

4.1 REPORT_JOB_SUBMIT

Parameter:

- OUT job_result VARCHAR(250)
- OUT job_id INT
- IN job_action TEXT
- IN job_start_time DATETIME
- IN job_next_start TEXT
- IN job_schema VARCHAR(250)
- IN report_recipient VARCHAR(250)

REPORT_JOB_SUBMIT legt äquivalent zu JOB_SUBMIT einen neuen Report-Job an. Als neuen Parameter gibt es `report_recipient`. Hier wird die Email Adresse des Report Empfängers festgelegt.

Beispiel: Anlegen eines Report Jobs

```
call scheduler.JOB_SUBMIT(@result, @jobid, 'SELECT * FROM persondata;', null, 'ADDTIME(NOW(), "1:0:0")', 'test', 'info@sos-berlin.com');
```

4.2 REPORT_JOB_SET_MAILTO

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN report_recipient VARCHAR(250)

Setzt die Adresse des Email Empfängers.

Beispiel: Report Adresse von Job 30 auf 'info@sos-berlin.com' setzen.

```
call scheduler.REPORT_JOB_SET_MAILTO(@result, 30, 'info@sos-berlin.com');
```

4.3 REPORT_JOB_SET_MAILCC

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN report_recipient VARCHAR(250)

Setzt die Adresse des Email-CC Empfängers.

4.4 REPORT_JOB_SET_MAILBCC

Parameter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN report_recipient VARCHAR(250)

Setzt die Adresse des Email-BCC Empfängers.

4.5 REPORT_JOB_SET_ASBODY

Paramter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN asbody BOOL

Steht `asbody` auf `true(1)`, so wird der generierte Report als Body der Email und nicht als Attachment verschickt.

4.6 REPORT_JOB_SET_BODY

Paramter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN body TEXT

Der Parameter `body` setzt den body der Report email. Dieser body wird nur verwendet, wenn `asbody` nicht auf `true` gesetzt wurde.

4.7 REPORT_JOB_SET_SUBJECT

Paramter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN subject VARCHAR(250)

Der Parameter `subject` setzt das Subject der Report email.

4.8 REPORT_JOB_SET_STYLESHEET

Paramter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN stylesheet VARCHAR(250)

Der Parameter `stylesheet` setzt das Stylesheet für die Transformation des Reports.

4.9 REPORT_JOB_SET_PATH

Paramter:

- OUT job_result VARCHAR(250)

- IN job_id INT
- IN path VARCHAR(250)

Wird hier ein Pfad gesetzt, so wird der Report nicht nur per email verschickt, sondern auch hier abgelegt.

4.10 REPORT_JOB_SET_FILENAME

Paramter:

- OUT job_result VARCHAR(250)
- IN job_id INT
- IN filename VARCHAR(250)

Der Parameter `filename` setzt die Dateinamenmaske für die Reportdatei.

4.11 SEND_MAIL

Paramter:

- OUT job_result VARCHAR(250)
- IN mailto VARCHAR(250)
- IN subject VARCHAR(250)
- IN body TEXT

Diese Funktion verschickt eine Email mit dem Betreff `subject` an den Empfänger `mailto`. `body` gibt den Text der Email an. Hier darf auch Html verwendet werden.