

# SqlScheduler

- Co gdzie i jak
- Definicje jobów
- Zakresy czasu
  - Szablon skryptu
- Check typu [R] (automatyczny feedback)
  - Warunki startowe
  - Skadowe "odwróconego" checka
  - Przykładowy zestaw plików dla całego checka
  - Szablon skryptu [R]
  - Tabelka dla Jiry z wieloma wartociami
- Monitoring praw wykonywania

## Podstawowe informacje

- Lokalizacja: **scripts:/home/mpdevelop/DNIU/sqlScheduler**
- Uruchamiane z crona użytkownika **mpdevelop** na serwerze dniu (`*** ***/home/mpdevelop/DNIU/sqlScheduler/run.sh`)
- Jeżeli wyjde check **check\_sqlscheduler** albo **sqlsched\_is\_alive** należy sprawdzić czy wpis w crontabie nie został zakomentowany, sprawdzić zawartość logów (`/home/dyzur/svn/DNIU/sqlScheduler/log`) i spróbować poprawić sqlSchedulera. Priorytet pilny. Pierwszy z checków (**check\_sqlscheduler**) sprawdza timestamp ostatniego wpisu w logach, drugi (**sqlsched\_is\_alive**) sprawdza kiedy ostatnio sqlscheduler sprawdził czy może dobrać się do klastra oracle i czy udało mu się to zrobić (czyli szuka w logach "job: test: status: 0: result: X")

## Co gdzie i jak

```
./bin/* - skrypty wykorzystywane bezpośrednio przez główny skrypt
./cfg/* - dodatkowe konfiguracje dla danych jobów/skryptów
./disabled/jobs/* - wyczone joby
./disabled/scripts/* - wyczone skrypty
./disabled/sql/* - wyczone sql
./jobs/* - definicje jobów
./log/* - katalog do którego trafiają logi
./scripts/* - skrypty uruchamiane przez joby, ewentualnie skrypty pomocnicze.
./sql/* - zapytania sql wykorzystywane przez joby
```

Wszystkie ciekawe wzglednie do lokalizacji w której sqlScheduler został umieszczony

## Definicje jobów

Joby umieszczamy w katalogu **jobs** i tworzymy następująco

```
name = Expired Services
sql = is_expired
dbhost = sigma
timeperiods = 0:1440,1
resultscript = is_expired
empty = :
errorhandler = generic_ah
```

- name - nazwa joba
- sql - nazwa pliku z zapytaniem sql; dla przykładu powyższego **is\_expired** zostanie zamienione na **./sql/is\_expired.sql**
- dbhost - nazwa bazy na której zapytanie ma być wykonane; zgodnie z definicjami w dbproxy
- timeperiods - Zakresy czasu

- resultscript - skrypt do którego zostanie przekazany wynik w przypadku poprawnego wykonania zapytania i nie-pustego wyniku
- empty - skrypt który zostanie uruchomiony w przypadku gdy zapytanie zwróci pusty wynik; można użyć: (shellowy no-op) jeżeli nic nie ma zostać wykonane
- errorhandler - skrypt który zostanie uruchomiony w przypadku błąd w wykonaniu zapytania (np brak połączenia z dbproxy, błąd w zapytaniu, awaria bazy). Błąd zostanie przekazany do tego skryptu.

## Zakresy czasu

Każdy job może mieć podane kilka zakresów czasu w których jest uruchamiany. W przypadku gdy jest ich więcej niż 1, powinny być one rozdzielone spacją.

Póki co zakres czasu może wyglądać następująco:

- **OD:DO,CZSTOTLIWO|D1:D2** - OD i DO to kolejna minuta doby; aktualny czas w takim formacie można uzyskać przez

```
date '+%H*60+%M' | bc
```

**0:1440** oznacza całą dobę; częstotliwość oznacza co ile minut dany job się powinien odpalać

- **@CZAS|D1:D2** - kolejna minuta doby w której powinien się job odpalić, podobnie jak OD i DO  
Te |D1:D2 to część opcjonalnego zakresu czasu (wykorzystywana obecnie jedynie w orlen\_smscount).

**D1** to pierwszy dzień w tygodniu w którym dany zakres czasu obowiązuje, **D2** ostatni.

Dni tygodnia są symbolizowane przez kolejne liczby, poniedziałek to 1, niedziela to 7 (nie wincie mnie. "date +%w").

W przypadku kiedy chcemy wycofać 1 dzień tygodnia z działania, dopuszczalna jest konstrukcja w rodzaju:

**timeperiods \* 0:1440,1|1:3 0:1440,1|5:7**

## Standard tworzenia nowych reguł

Tworząc nowe reguły w SqlSchedulerze stosujemy się do poniższych wytycznych, pozwoli to uniknąć zbędnego bałaganu na Helpline oraz ułatwi nam pracę.

- Na początku tematu tworzonego joba (zmienna summary) powinna znajdować się fraza [sqlScheduler], przykład:

```
summary="[sqlScheduler] Monitoring usługi 101045 KonkursKasa dla Ciebie"
```

- Jeśli zapytanie jest większe niż 500 znaków należy na jego podstawie utworzyć widok w schemacie usera, z którego będziemy uruchamiać zapytanie oraz w sqlschedulerze wykorzystać zapytanie:

```
select * from nazwa_widoku
```

- Powinnyśmy stosować funkcję **get\_task\_id** pozwala to uniknąć generowania duplikatów tasków, przykład użycia:

```
[[ $(get_issue_id -p "ITSO" -s "${summary}") -lt 0 ]]
```

- Należy nadać odpowiednią kategorię i priorytet, czyli odpowiednie wartości polom **category** oraz **priority**

## Szablon skryptu

W 99% przypadków, poniższy skrypt spełnia wymagania stawiane przed monitoringiem (wysyłanie maili + tworzenie tasków):

### Przykładowy skrypt

```
#!/bin/bash
name="nazwa_skryptu"
LOG="/home/mpdevelop/DNIU/sqlScheduler/log/sqlScheduler_scriptsLog."`date
+ "%Y-%m-%d"`.log"
echo "---
$(date) $name started with: ${@} " >> $LOG
# Mail recipients and subject
RECIPIENTS="maile_oddzielone_przecinkami"
subject="Temat maila"
# Jira task information
project="HL" # helpline-it
category="Default"
summary="[sqlScheduler] $name"
category="Inne"
# opis taska / tre maila, rezultat zapytania standardowo zapisywany w tablicy
parametrów bashu: ${@} (lub zmienna ${context} w przypadku tabelki dla Jiry)
text=""
# *** Main scrip -> create task if not exist and send mail if necessary ***
echo "$(date) trying to create task " >> $LOG
echo "$(date) Response from JIRA API - task exist?"
$(/home/mpdevelop/DNIU/sqlScheduler/bin/get_issue_id -s "${summary}" -p ${project})"
>> $LOG

#Tworzenie taska
if [[ $(/home/mpdevelop/DNIU/mktask/mktask.pl -SE -P "${project}" -s "${summary}" )
-eq 0 ]]; then
    /home/mpdevelop/DNIU/mktask/mktask.pl -CR -P "${project}" -s "${summary}" -d
"${text}"
    echo "$(date) Task created" >> $LOG
# Wysyłanie maila
    echo "${text}" | mutt -s "${subject}" -- "${RECIPIENTS}"
    echo "$(date) Information sent to: $RECIPIENTS" >> $LOG
fi
```

### Check typu [R] (automatyczny feedback)

Istnieje moliwo stworzenia takiego checka, który będzie się zwraca (task przejdzie w status feedback) w momencie, gdy dana awaria się zakończy. W praktyce oznacza to sprytne skonfigurowanie drugiego lustrzanego checka.

### Warunki startowe

1. Check musi sprawdzać pojedyncze uwarunkowanie  
Oznacza to, że check sprawdza np. bilowania tylko u jednego operatora, lub rejestracje tylko do jednego serwisu. Dzieje się tak ze względu na ograniczenia samego Sqlschedulera, Jiry i możliwe komplikacje.  
Wyobramy sobie check sprawdzający 4 operatorów, wpada alert o awarii u dwóch, a następnie jeden jest już w stanie ok. Zwraca taska z informacją, czy jest ok, czy nie? Co się stanie jeżeli w tym czasie przestanie działać kolejny operator? Ze względu na możliwe komplikacje i zaciemnienie sytuacji, checki muszą być proste i konkretne.

Aby powstał check typu [R], należy zdefiniować normalnego checka, a następnie kolejnego, z sql, który spełnia odwrotny warunek (np. awaria to mniej niż 100 bilowa, zatem odwrotne sql to 100 lub więcej bilowa).

### Skadowe "odwrotnego" checka

1. Joba, wykonującego si co godzin
2. Sql z odwróconym warunkiem
3. Skryptu, którego przykład podany jest niej

Wszystkie pliki, posiada powinny nazwy identyczne jak sam check wykrywajcy awari, z dodatkowym sufixem `-rev` na kocu.

### Przykładowy zestaw plików dla caego checka

LP	Rodzaj	Nazwa
1	skrypt checka	brazil_transaction_oi.sh
2	job checka	brazil_transaction_oi
3	sql checka	brazil_transaction_oi.sql
4	skrypt [R]	brazil_transaction_oi-rev.sh
5	job [R]	brazil_transaction_oi-rev
6	sql [R]	brazil_transaction_oi-rev.sql

### Szablon skryptu [R]

## Skrypt zwracajcy taska

```
#!/bin/bash
name="nazwa"
LOG="/home/mpdevelop/DNIU/sqlScheduler/log/sqlScheduler_scriptsLog."`date
+ "%Y-%m-%d"`.log"
echo "---
$(date) $name started with: ${@} " >> $LOG
# Mail recipients and subject
#RECIPIENTS="maile_oddzielone_przecinkami"
#subject="Temat maila"
# Jira task information
project="HL" # helpline-it
category="Default"
summary="[sqlScheduler] [R] $name"
category="Inne"
# tre komentarza po feedbacku taska, rezultat zapytania standardowo zapisywany w
tablicy parametrów bashu: ${@} (lub zmienna ${context} w przypadku tabelki dla Jiry)
"
# *** Main scrip -> create task if not exist and send mail if necessary ***
echo "$(date) trying to feedback task " >> $LOG
echo "$(date) Response from JIRA API - task exist?"
$(/home/mpdevelop/DNIU/sqlScheduler/bin/get_issue_id -s "${summary}" -p ${project})"
>> $LOG
# Feedbackowanie taska
task=$(/home/mpdevelop/DNIU/mktask/mktask.pl -SE -P "${project}" -s "${summary}" |
head -n 1)
if [[ $task -ne 0 ]]; then
# echo $task
/home/mpdevelop/DNIU/mktask/mktask.pl -UP --text "${text}" -fb -i "${task}"
echo "$(date) Task feedbacked" >> $LOG
# Wysylanie maila
# echo "${text}" | mutt -s "${subject}" -- "${RECIPIENTS}"
# echo "$(date) Information sent to: $RECIPIENTS" >> $LOG
fi
```

Należy zwrócić uwagę na inn konstrukcj if pod koniec skryptu i inne uycie mktask ni w standardowym skrypcie checka

## Tips & Tricks

Kilka przydatnych rad, jak sobie poradzi z formatowaniem taska.

### Tabelka dla Jiry z wieloma wartociami

Jak wiadomo JIRA potrafi generowa adne tabelki. Wrzucając jednak wynik zapytania `${@}`, nawet umieszczając go między pionowe kreski `||`, uzyskamy dobry widok tylko dla pojedynczego wiersza.





Aby rozwiązać ten problem, w pierwszej kolejności należy dane wrzucić do tablicy, a następnie przypisać do zmiennej, która będzie już gotową tabelką:

```
# parsowanie danych do tabelki Jiry
s=0;
my_array=( );
IFS=$'\n'
for i in ${@}; do
    my_array[$s]="| $i | ";
    ((s++));
done;
# przypisanie tabelki Jiry do zmiennej
content=`printf '%s\n' "${my_array[@]}"`
```

Następnie, w trzecim zadaniu, zamiast zmiennej parametrów bash, należy użyć zmiennej content:

```
${content}
```

Różnicę przed i po zastosowaniu porady widać w poniższych zadaniach:

PRZED	PO
 <b>HL-94948</b> - [sqlScheduler] showmax_no_reg <b>CLOSED</b>	 <b>HL-96358</b> - [sqlScheduler] showmax_no_reg <b>CLOSED</b>
 <b>HL-96618</b> - [sqlScheduler] directbilling_all_registration <b>CLOSED</b>	 <b>HL-96633</b> - [sqlScheduler] directbilling_all_registration <b>CLOSED</b>

## Monitoring praw wykonywania

Wszystkie skrypty SQLScheduler, odpowiadające za interakcje muszą posiadać prawa wykonywania - przynajmniej dla użytkownika mpdevelop, z którego uruchamiany jest monitoring. Brak takich uprawnień powoduje, że w przypadku zwrócenia błędnej odpowiedzi przez zapytanie nie będzie żadnej reakcji monitoringu.

Aby temu zaradzić i ustrzec się przed poniesieniem nieuczciwej odpowiedzialności należy w SVN dodać stosowne uprawnienia do skryptu

```
svn propset svn:executable on nazwa_skryptu
svn commit nazwa_skryptu -m "Nadanie uprawnienia wykonywania"
```

Po wykonaniu tych dwóch kroków, wystarczy dokonać aktualizacji skryptu, co spowoduje ustawienie poprawnych uprawnień wykonywania. Każda następna osoba modyfikująca przygotowany skrypt z góry będzie miała ustawione poprawne prawa do wykonywania pliku.