Advanced Monitoring von ETL-Prozessen

Sven Bosinger
its-people

Schlüsselworte

Data Warehouse, ETL, Ladeprozesse, Datenbewirtschaftung, Oracle Scheduler, Execution Plan, V$-Views, Alerting, Profiles

Einleitung

Die Datenbewirtschaftung stellt im Rahmen einer Data Warehouse Lösung immer eine Herausforderung dar. Dabei werden viele einzelne Verarbeitungsschritte, Mappings genannt, zu mehr oder weniger großen Prozessketten zusammengestellt. Der Start eines einzelnen Mappings ist dabei in der Regel abhängig vom Status der Verarbeitung der zuvor gestarteten Mappings. Um derartige Ladeszenarien automatisiert zu betreiben kann der Scheduler-Mechanismus der Oracle Datenbank eingesetzt werden. Dazu wird hier eine Lösung basierend auf einer Oracle 11g Rel.2 Datenbank und Apex 4.0 vorgestellt.

Besonderes Augenmerk wurde bei der Lösung auf den regelmäßigen Betrieb und die damit einhergehenden Herausforderungen an das Monitoring gelegt. So kann ein Logging auf unterschiedlicher Detailebene dynamisch zur Laufzeit ein- und ausgeschaltet werden. Ebenfalls ist es möglich, relevante Informationen über die Abarbeitung einzelner SQL-Statements zu protokollieren, bis hin zum durch den Cost Base Optmizer gewählten Ausführungsplan. Darüber hinaus ist es möglich auf Statement-Ebene Alert-Events zu formulieren, welche bei Über- oder Unterschreitung vorher festgelegter Schwellwerte eine Benachrichtigung der Betriebsführung auslösen.

Der Oracle Scheduler

Die von its-people entwickelte Job-Steuerung setzt als Basistechnologie den Oracle Scheduler ein. Der Oracle Scheduler wurde mit der Datenbank- Version 10g Rel.1 eingeführt und löst die Funktionalitäten des DBMS_JOB Packages ab. In der Datenbank-Version 11g Rel.2 wurde die Funktionalität nochmals erweitert.

Der Oracle Scheduler dient dazu, PL/SQL- oder Java- Programme in einer lokalen oder einer Remote Datenbank zu planen, zu starten und zu kontrollieren. Darüber hinaus können auch Betriebssystem-Skripte außerhalb der Datenbank gestartet werden. Das eigentliche Scheduling kann Zeitbasiert, Ereignisbasiert oder in Abhängigkeit eines anderen Schedulings erfolgen. Im Rahmen des Oracle Schedulers können einzelne Jobs zu Job- Klassen zusammengefasst werden, die mit Prioritäten versehen werden können. So kann gewährleistet werden, dass kritische, hoch priorisierte Jobs mit ausreichenden Ressourcen versehen werden, so dass deren Abarbeitung unabhängig von der aktuellen Auslastung des Systems

gewährleistet wird. Darüber hinaus stellt der Oracle Scheduler Funktionalitäten zur Verfügung, um Jobs auch in einem Cluster Environment zu managen und zu monitoren.

Komponenten

Der Oracle Scheduler besteht aus mehreren Komponenten von denen hier einige ausgewählte näher beschrieben werden sollen. Angelegt, modifiziert und gelöscht werden die Komponenten des Oracle Schedulers über das PL/SQL-Package DBMS_SCHEDULER.

Program

Ein *Program* beschreibt, was durch den Oracle Scheduler gestartet und betrieben werden soll. Ein *Program* besteht aus:

- *Action*: Name der zu startenden PL/SQL-Procedure, zu startendes Betriebssystem-Skript oder ein anonymer PL/SQL-Block
- *Type*: Typ der *Action*, 'STORED_PROCEDURE', 'PLSQL_BLOCK' oder 'EXTERNAL'
- Anzahl der Argumente (Parameter), die an die *Action* übergeben werden

Chains

Eine *Chain* ist eine Kette von einzelnen Verarbeitungsschritten, die ein *Program* oder wiederum eine andere *Chain* sein können.

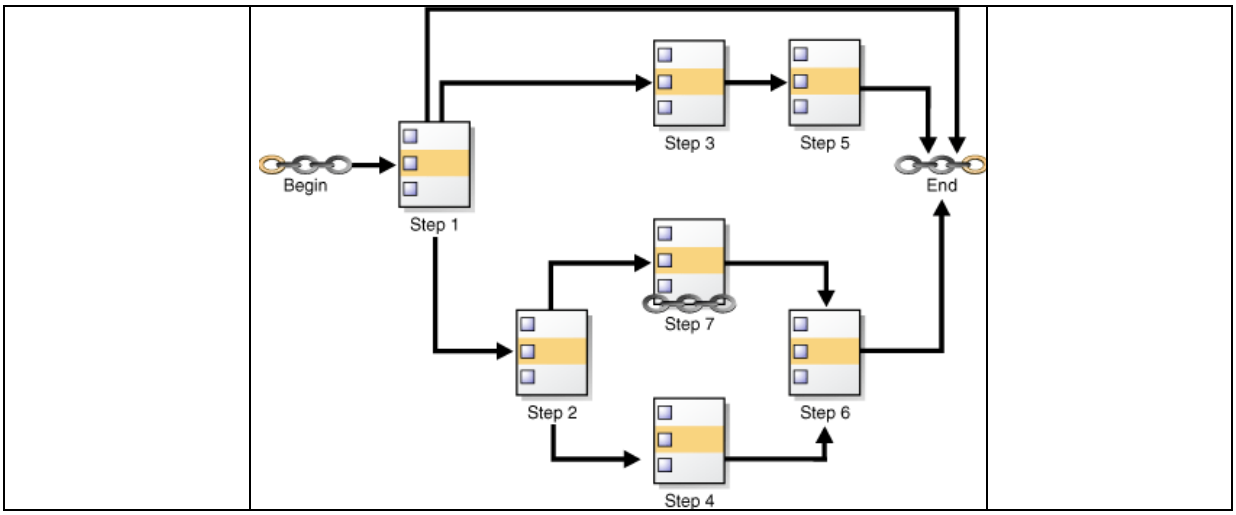

Abb. 1: Chain mit mehrfacher Verzweigung

Dabei können zwischen den einzelnen Verarbeitungsschritten Regeln implementiert werden, die steuern, ob und wann ein nachfolgender Schritt gestartet wird. Dies erfolgt über sogenannte *Chain Rules*.

Chain Rule

Eine *Chain Rule* beschreibt, unter welcher Bedingung ein einzelner Schritt einer *Chain* gestartet werden soll. Dabei kann der Start am Zustand eines oder mehrer anderer Schritte festgemacht werden. Die Zustände, die dabei eingesetzt werden können sind SUCCEEDED, FAILED, STOPPED oder COMPLETED. Die jeweiligen Zustände der einzelnen Schritte können dabei mit booleschen Operatoren verknüpft werden, z.B.: "STEP1 SUCCEEDED AND STEP2 NOT FAILED". Des Weiteren kann man sich auf einen ERROR_CODE eines Schrittes beziehen. Ein Schritt einer *Chain* wird nur dann gestartet, wenn der boolesche Ausdruck der *Chaine Rule* den Wert "TRUE" ergibt.

Job

Ein *Job* ist eine Sammlung von Metadaten, die eine benutzerdefinierte Aufgabe beschreiben. Es gibt verschiedene Arten von *Jobs*, hier die wichtigsten:

- *Database Jobs*: Diese starten eine PL/SQL-Procedure oder einen PL/SQL-Block direkt. Es wird dabei unterschieden zwischen *local database jobs* (die PL/SQL-Source liegt in derselben Datenbank) und *remote database jobs* (die PL/SQL-Source liegt in einer anderen Datenbank).

- *External Jobs*: Diese starten ein Betriebssystem-Skript. Auch hier gibt es die Unterscheidung zwischen *local* und *remote*.

- *Chain Jobs*: Diese starten eine *Chain*.

Ein *Job* kann grundsätzlich synchron, d.h. in der *Session* aus der heraus er aufgerufen wurde, oder asynchron, d.h. in einer neuen *Session* laufen. Der zweite Fall ist vergleichbar mit einem Hintergrundprozess, der weiterläuft, auch wenn die aktuelle *Session* beendet wird.

Schedule

Ein *Schedule* bestimmt wann und wie oft ein *Job* gestartet wird. Dabei kann ein *Schedule* von mehreren *Jobs* genutzt werden. Es gibt dabei zwei Arten von *Schedules*:

- *Time Schedules*: In diesen wird festgelegt, wann ein *Job* das erste Mal gestartet wird. Darüber hinaus kann festgelegt werden in welchem Zeitabstand (Intervall) er jedes weitere Mal gestartet werden soll und ab wann er ggf. nicht mehr zu starten ist.

- *Event Schedules*: Ein *Job* wird gestartet wenn ein definiertes *Event* eingetreten ist, z.B. wenn eine bestimmte Datei im Betriebssystem abgelegt wurde.

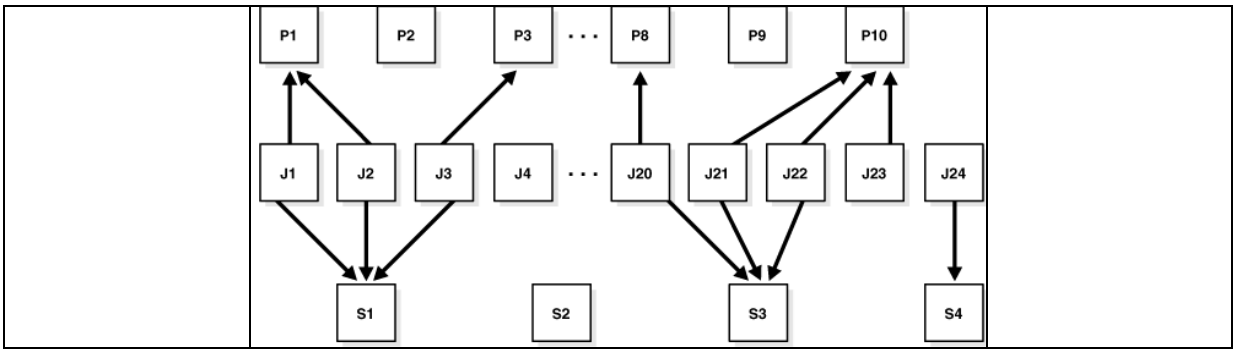

Abb. 2: Beziehung zwischen Programs, Jobs und Schedules

Beispiel

Im Folgenden Beispiel wird eine Prozesskette aus drei Prozessschritten angelegt, wobei Schritt1 zuerst startet und Schritt2 und Schritt3 parallel gestartet werden, sobald Schritt1 erfolgreich beendet wurde.

Anlage von *Program1*

Program1 startet die PL/SQL-Procedure PCK_LOAD_STAGE.SETZE_NEUE_LADE_NR im Schema DWH, die drei Parameter benötigt:

```
begin
  dbms_scheduler.create_program(program_name        => 'SETZE_NEUE_LADE_NR'
                               ,program_type        => 'STORED_PROCEDURE'
                               ,program_action      => 'DWH.PCK_LOAD_STAGE.SETZE_NEUE_LADE_NR'
                               ,number_of_arguments => 3
                               ,enabled             => false
                               ,comments            => 'Setzen der neuen Ladenummer'
                               );
  dbms_scheduler.define_program_argument(program_name      => 'SETZE_NEUE_LADE_NR'
                                        ,argument_position => 1
                                        ,argument_name     => 'P_ID'
                                        ,argument_type     => 'NUMBER'
                                        ,default_value     => 0
                                        ,out_argument      => false
                                        );
  dbms_scheduler.define_program_argument(program_name      => 'SETZE_NEUE_LADE_NR'
                                        ,argument_position => 2
                                        ,argument_name     => 'P_LOG'
                                        ,argument_type     => 'NUMBER'
                                        ,default_value     => 5
                                        ,out_argument      => false
                                        );
  dbms_scheduler.define_program_argument(program_name      => 'SETZE_NEUE_LADE_NR'
                                        ,argument_position => 3
                                        ,argument_name     => 'P_DATUM'
                                        ,argument_type     => 'DATE'
                                        ,default_value     => '29.07.2010'
                                        ,out_argument      => false
                                        );
  dbms_scheduler.enable('SETZE_NEUE_LADE_NR');
end;
/
```

Anlage von *Program2*

Program2 startet die PL/SQL-Procedure PCK_LOAD_STAGE.LOAD_STG_KUNDEN im Schema DWH, die zwei Parameter benötigt:

```
begin
  dbms_scheduler.create_program(program_name        => 'STG_KUNDEN_LADEN'
                               ,program_type        => 'STORED_PROCEDURE'
                               ,program_action      => 'DWH.PCK_LOAD_STAGE.LOAD_STG_KUNDEN'
                               ,number_of_arguments => 2
                               ,enabled             => false
                               ,comments            => 'Laden der Kunden nach STAGE'
                               );
  dbms_scheduler.define_program_argument(program_name      => 'STG_KUNDEN_LADEN'
                                        ,argument_position => 1
                                        ,argument_name     => 'P_ID'
                                        ,argument_type     => 'NUMBER'
                                        ,default_value     => 0
                                        ,out_argument      => false
                                        );
  dbms_scheduler.define_program_argument(program_name      => 'STG_KUNDEN_LADEN'
                                        ,argument_position => 2
                                        ,argument_name     => 'P_LOG'
                                        ,argument_type     => 'NUMBER'
                                        ,default_value     => 5
                                        ,out_argument      => false
                                        );
  dbms_scheduler.enable('STG_KUNDEN_LADEN');
end;
/
```

Anlage von *Program3*

Program3 startet die PL/SQL-Procedure PCK_LOAD_STAGE.LOAD_STG_AUFTRAEGE im Schema DWH, die zwei Parameter benötigt:

```
begin
  dbms_scheduler.create_program(program_name        => 'STG_AUFTRAEGE_LADEN'
                               ,program_type        => 'STORED_PROCEDURE'
                               ,program_action      => 'DWH.PCK_LOAD_STAGE.LOAD_STG_AUFTRAEGE'
                               ,number_of_arguments => 2
                               ,enabled             => false
                               ,comments            => 'Laden der Kunden nach STAGE'
                               );
  dbms_scheduler.define_program_argument(program_name      => 'STG_AUFTRAEGE_LADEN'
                                        ,argument_position => 1
                                        ,argument_name     => 'P_ID'
                                        ,argument_type     => 'NUMBER'
                                        ,default_value     => 0
                                        ,out_argument      => false
                                        );
  dbms_scheduler.define_program_argument(program_name      => 'STG_AUFTRAEGE_LADEN'
                                        ,argument_position => 2
                                        ,argument_name     => 'P_LOG'
                                        ,argument_type     => 'NUMBER'
                                        ,default_value     => 5
                                        ,out_argument      => false
                                        );
  dbms_scheduler.enable('STG_AUFTRAEGE_LADEN');
end;
/
```

Anlage der *Chain*

Die *Chain* besteht aus drei Schritten (*Program1* bis *3*) und den zugehörigen *Chain Rules*, die sicherstellen, das *Program1* augenblicklich und *Program2* und *Program3* erst nach erfolgreichem Abschluss von *Program1* gestartet werden:

```
begin
  dbms_scheduler.create_chain
    (chain_name              => 'LOAD_DWH'
    ,rule_set_name           => NULL
    ,evaluation_interval     => NULL
    ,comments                => 'Laden des DWH'
    );
  dbms_scheduler.define_chain_step
    (chain_name              => 'LOAD_DWH'
    ,step_name               => 'STEP1'
    ,program_name            => 'SETZE_NEUE_LADE_NR'
    );
  dbms_scheduler.define_chain_step
    (chain_name              => 'LOAD_DWH'
    ,step_name               => 'STEP2'
    ,program_name            => 'STG_KUNDEN_LADEN'
    );
  dbms_scheduler.define_chain_step
    (chain_name              => 'LOAD_DWH'
    ,step_name               => 'STEP3'
    ,program_name            => 'STG_AUFTRAEGE_LADEN'
    );
  dbms_scheduler.define_chain_rule
    (chain_name   =>   'LOAD_DWH'
    ,condition    =>   'TRUE'
    ,action       =>   'START STEP1'
    ,rule_name    =>   'RULE_FOR_STEP1'
    ,comments     =>   'Starten der Chain'
    );
  dbms_scheduler.define_chain_rule
    (chain_name   =>   'LOAD_DWH'
    ,condition    =>   'STEP1 SUCCEEDED'
    ,action       =>   'START STEP2'
    ,rule_name    =>   'RULE_FOR_STEP2'
    ,comments     =>   'Starten von Step2, wenn Step1 erfolgreich beendet'
    );
  dbms_scheduler.define_chain_rule
    (chain_name   =>   'LOAD_DWH'
    ,condition    =>   'STEP1 SUCCEEDED'
    ,action       =>   'START STEP3'
    ,rule_name    =>   'RULE_FOR_STEP3'
    ,comments     =>   'Starten von Step3, wenn Step1 erfolgreich beendet'
    );
  dbms_scheduler.define_chain_rule
    (chain_name   =>   'LOAD_DWH'
    ,condition    =>   'STEP2 COMPLETED AND STEP3 COMPLETED'
    ,action       =>   'END'
    ,rule_name    =>   'RULE_FOR_END'
    ,comments     =>   'Beenden der Chain'
    );
  dbms_scheduler.enable ('LOAD_DWH');
end;
/
```

Anlage und einmaliger Start des *Jobs*

Die *Chain* wird einem *Job* zugeordnet, der einmalig asynchron gestartet wird:

```
begin
  dbms_scheduler.create_job
    (job_name            => 'JOB_LOAD_DWH'
    ,job_type            => 'CHAIN'
    ,job_action          => 'LOAD_DWH'
    ,number_of_arguments => 0
    ,enabled             => false
    ,comments            => 'Job zum Laden des DWH'
    ,auto_drop           => false
    );
end;
/

begin
  dbms_scheduler.enable ('JOB_LOAD_DWH');
end;
/
```

Die Job-Steuerung

In einer Data Warehouse Lösung ist es üblich in festen Intervallen, z.B. jede Nacht, über einen Batch-Job Daten zu aktualisieren. Diese Datenbewirtschaftung erfolgt in der Regel über eine große Anzahl in einer Abhängigkeit zueinander stehender Mappings. Diese werden zum Teil seriell und zum Teil parallel abgearbeitet und können zu einer Prozesskette zusammengefasst werden. Klassischerweise gibt es in einer Data Warehouse Lösung nicht nur eine derartige Prozesskette, sondern eine Vielzahl, die unterschiedliche Aufgaben übernehmen können, z.B. nächtliches Laden der Daten, Monats- und Jahresabschlüsse und Wöchentliche Wartungsaktivitäten.

Um eine derartige Datenbewirtschaftung möglichst effizient durch die Betriebsführung durchzuführen, wurde von its-people eine Applikation mit der Bezeichnung Job-Steuerung entwickelt. Diese bedient sich der vorher dargelegten Komponenten und Funktionen des Oracle Schedulers und kombiniert diese zusammen mit einer GUI zu einer flexibel Einsetzbaren Lösung. Ergänzt wird diese um ein ebenfalls von its-people entwickeltes Statistik-Modul, dass eine feingranulare Protokollierung der einzelnen Prozessschritte und Mappings ermöglicht.

Bei der hier im Folgenden dargestellten Job-Steuerung handelt es sich um eine Applikation, die in der Lage ist:

- Prozessketten zu definieren und zu verwalten, die aus einzelnen Prozessschritten bestehen
- Prozessketten zu schedulen und zu monitoren
- Prozessprotokolle zu erstellen und zu visualisieren

Die Applikation besteht aus mehreren Komponenten:

- Eine API, die in PL/SQL entwickelt wurde und einerseits dazu dient Prozesse und Prozessketten zu erstellen, zu verändern und zu löschen. Zum Anderen kann mittels dieser API ein Prozess gestartet und kontrolliert werden.

- Ein Datenmodell, in dem die Metadaten der einzelnen Objekte (Prozesse, Prozessschritte, Parameter, …) gespeichert werden. Die Tabellen werden über die API gefüllt.

- Ein Repository, dass sämtliche erzeugten Protokolle und Statistiken aufnimmt

- Eine GUI, entwickelt unter Apex 4.0, über welche die API angesteuert und die Protokoll- und Statistikinformationen visualisiert werden können

Objekte

Folgende Objekte werden im Rahmen der Job-Steuerung erzeugt und verwaltet:

Prozesse

Ein Prozess oder eine Prozesskette ist eine Abfolge von einzelnen Prozessschritten, die in festgelegter Reihenfolge gestartet werden. Ein Prozessschritt wird durch den Prozess dann gestartet, wenn alle vorherigen Prozessschritte erfolgreich beendet wurden. Beim Start eines Prozesses kann der erste und der letzte Prozessschritt vorgegeben werden, dadurch ist eine Wiederaufsetzbarkeit gewährleistet. Technisch wird ein Prozess durch eine *Chain* abgebildet.

Prozessschritte

Prozessschritte sind aufsteigend numerisch innerhalb eines Prozesses sortiert und werden in dieser Reihenfolge abgearbeitet. In einem Prozessschritt können mehrer Programme parallel gestartet werden. Der Prozessschritt gilt erst dann als erfolgreich beendet, wenn alle gestarteten Programme erfolgreich beendet wurden. Bricht nur eins dieser Programme mit einem Fehler ab, so gilt der gesamte Prozessschritt als nicht erfolgreich und die Verarbeitung der gesamten Prozesskette wird mit einem Fehler beendet. Die Abhängigkeiten der Prozessschritte untereinander werden mit *Chain Rules* abgebildet.

Programme

Ein Programm ist entweder eine PL/SQL Stored-Procedure (standalone oder im Package) oder ein Betriebssystem-Skript. Ein Programm kann beliebig viele Parameter besitzen. Ein Programm kann dabei unter dem Job-Steuerungsbenutzer oder einem beliebigen anderen Schema-Benutzer gestartet werden, sofern dieser Execute-Rechte auf das Programm besitzt. Die Programme werden durch *Programs* realisiert.

Parameter

Jedes Programm, auch ein Betriebssystem-Skript, kann mit beliebig vielen Parametern versehen werden. Allerdings sind die Datentypen auf NUMBER, VARCHAR2 und DATE beschränkt. Die Parameterwerte werden in den Metatabellen gespeichert und zum Start des Prozesses ausgelesen.

Funktionen

Die Job-Steuerung besitzt eine Reihe von Funktionen, die sämtlich über eine API abgebildet werden.

Verwaltung von Objekten

Die Objekte Prozess, Prozessschritt, Programm und Parameter können angelegt, gelöscht und modifiziert werden. Dabei kann eine Änderung an einem Objekt nur dann erfolgen, wenn der zugehörige Prozess aktuell nicht gestartet ist. Während ein Prozess läuft, ist dieser gegen Änderungen gesperrt.

Scheduling von Prozessen

Ein Prozess kann einmalig gestartet werden oder auch in einem regelmäßigen Intervall. Wird ein Prozess einmalig gestartet, kann festgelegt werden, mit welchem Prozessschritt er beginnt und mit welchem er enden soll. Als Default wird immer der kleinste respektive größte Prozessschritt gewählt. Der einmalige Start eines Prozesses wird über einen *Job* realisiert.

Soll der Prozess mehrfach gestartet werden, so kann ein Intervall (Stunde, Tag, Monat, Jahr) und eine Intervallgröße angegeben werden. Soll nur eine bestimmte Anzahl an Starts ausgeführt werden, so kann diese begrenzt werden. Der mehrfache Start eines Prozesses wird über einen *Job* zusammen mit einem *Schedule* realisiert.

Monitoring

Jeder Prozess kann während seines Laufs und auch danach detailliert betrachtet werden. Dazu werden Monitoring-Informationen in das Repository der Job-Steuerung geschrieben, die mittels der API ausgelesen und aufbereitet werden können. Dabei sind alle Protokolleinträge eindeutig einem Prozessschritt, einem Prozess und einer Prozessausführung zugeordnet.

Protokollierung

Die in einem Prozessschritt aufgerufenen Programme können Protokollinformationen erzeugen, die in einer Protokoll-Tabelle gespeichert werden. Dabei sind die Protokolleinträge in vier verschiedene Klassen aufgeteilt:

- Fehler: Fehler werden immer protokolliert
- Log: Log-Informationen werden immer protokolliert
- Debug: Debug-Informationen werden nur dann protokolliert, wenn die zugehörigen Programme im Debug- oder Trace-Modus laufen
- Trace: Trace-Informationen werden nur dann protokolliert, wenn die zugehörigen Programme im Trace-Modus laufen

Der Modus, in welchem ein Prozess läuft, legt den Modus für die zugehörigen Programme fest und kann dynamisch zur Laufzeit geändert werden. Diese Änderung wirkt sich nur auf die noch nicht gestartete Programme aus. Die Protokollierung erfolgt ebenfalls in der zentralen Protokoll-Tabelle und ist damit immer einem Prozess, Prozessschritt und einer Prozessausführung eindeutig zugeordnet.

Das Statistik-Modul

Bei Datenwirtschaftungen, die regelmäßig zu einem festen Zeitpunkt gestartet werden, handelt es sich in der Regel um automatisch ablaufende Prozesse, die Bedienerfrei keine weiteren manuellen Aktionen benötigen. Sie laufen oft nachts zu lastarmen Zeiten. Es wird häufig kein Online-Monitoring durchgeführt, sondern nach Ende der Verarbeitung geprüft, ob Fehler aufgetreten sind. Kommt es dabei zu auffälligen Laufzeitveränderungen, z.B. ein Prozessschritt braucht die 15fache Zeit, wird dies meistens erst sehr spät festgestellt. Oft sind dann im Datenbanksystem keine Informationen mehr über die SQL-Laufzeitstatistiken, z.B. CPU-Verbrauch, oder Anzahl gelesener Blöcke, mehr vorhanden. Diese erschwert die Analyse.

Das Statitik-Modul stellt drei Funktionalitäten zu Verfügung, die im Folgenden näher beschrieben werden.

Statistiken

Über eine klassische Protokollierung der einzelnen Programme in einer Prozesskette hinaus, kann es daher sinnvoll sein weitergehende Detail-Informationen auf SQL-Statementebene zu erzeugen. Durch das von its-people entwickelte Statistik-Modul kann eine Protokollierung bis hin zum einzelnen SQL-Statement und dessen Laufzeitstatistiken durchgeführt werden. Erfasst werden hierbei:

- Laufzeitinformationen zur Abarbeitung eines einzelnen SQL-Statements, wie z.B. CPU-, und Elapsed-Time, Disk-Reads und Buffer-Gets.

- Execution-Pläne der gestarteten SQL-Statements

Hierzu werden die Informationen der V$-Views insbesondere V$SQL und V$SQL_PLAN herangezogen.

Jedes Programm oder auch der Prozess als ganzes kann mit einem Flag versehen werden, das festlegt in welcher Detaillierung die Informationen festgehalten werden. Es werden folgende Optionen angeboten

- N: keine Protokollierung

- S: Laufzeitinformationen zum SQL-Statement

- P: Laufzeitinformationen und Execution- Pläne zum SQL-Statement

Die erzeugten Informationen werden in eigenen Repository-Tabellen gespeichert und können so zu späteren Auswertung herangezogen werden. Die Informationen sind analog den klassischen Protokoll-Einträgen ebenfalls dem Prozess, Prozessschritt und der Prozessausführung zugeordnet.

Der Vorteil dieser Vorgehensweise liegt darin, dass bei regelmäßig laufenden Prozessen eine Zeitreihenbetrachtung durchgeführt werden kann, um möglichst frühzeitig Probleme oder Engpässe zu erkennen. Darüber hinaus stellt sich immer wieder die Frage, warum ein Programm, das normalerweise in sehr kurzer Zeit durchläuft auf einmal wesentlich länger braucht. Häufig kommt dies bei nächtlichen Ladeläufen vor und wird erst festgestellt, wenn keine Informationen mehr in den V$-Views zu finden sind. Über diese Form der Protokollierung kann eine wesentlich bessere und fundierte Analyse durchgeführt werden.

Profile

Neben der reinen Protokollierung der SQL-Statements gibt es noch die Möglichkeit sogenannte Profile zu erzeugen. Dabei handelt es sich um Laufzeitinformationen zu SQL-Statements, die über einen definierten Zeitbereich gemittelt wurden. Es wird dabei über relevante Kenngrößen, z.B. Buffer-Gets oder CPU-Time, ein Mittelwert und eine Standardabweichung gebildet. Diese Profile werden dann pro Statement in einer Tabelle hinterlegt. Bei einer Analyse können diese Profile als Richtwert herangezogen werden.

Alerting

Auf Basis der Statistiken und Profile kann eine Alerting-Funktion aktiviert werden. Diese vergleicht nach Erhebung der Laufzeitinformationen zu SQL-Statements diese Werte mit dem hinterlegten Profil und erzeugt einen Alert, wenn die neuen Werte nicht in dem Korridor liegen, der durch den Durchschnittswert und die Standardabweichung des Profils gebildet wird. Der Alert wird dann in der Protokoll-Tabelle gespeichert und kann zusätzlich noch als Email versendet werden.

Zusammenfassung

Durch den Einsatz der von its-people entwickelten Job-Steuerung ist es möglich, eine dauerhafte und nachhaltige Datenbewirtschaftung in einer Data Warehouse Lösung zu etablieren, die den Anforderungen eines Regelbetriebs genügt. Die hierzu eingesetzten Produkte Oracle Scheduler und Apex sind Standardprodukte des Herstellers Oracle und von den meisten Kunden allgemein akzeptiert. Zusätzliche Lizenzkosten zu einer Oracle Datenbank (mind. Standard Edition)) fallen nicht an. Durch den Einsatz einer in PL/SQL entwickelten API ist die Integration in andere Oberflächen, wie z.B. einer eigenen Java-Entwicklung, leicht möglich.

Das Statistik-Modul als Erweiterung zur Job-Steuerung eröffnet einen detaillierten Einblick in die Laufzeitinformationen der verwendeten SQL-Statements und ermöglich in Zusammenspiel mit den hinterlegten Profilen eine effiziente Problemanalyse bei laufzeitkritischen Batch-Jobs. Die Alerting-Funktionalität bietet darüber hinaus die

Möglichkeit ein Frühwarnsystem zu etablieren, das aufkommende Probleme schon frühzeitig erfasst und alle relevanten Daten zu Analyse und Behebung vorrätig hält.

Kontaktadresse:

Sven Bosinger
its-people Hochtaunus GmbH
Lyoner Str. 44-48
60528 Frankfurt am Main

Telefon: +49(0)69-24 75 210-0
Fax: +49(0)69-24 75 210-21
E-Mail sven.bosinger@its-people.de
Internet: www.its-people.de