Планировщик заданий в Oracle

Владимир Пржиялковский
Преподаватель технологий Oracle
prz@yandex.ru
http://www.ccas.ru/prz

Апрель 2008 г.

…Я старался дознаться у них, почему Нил, начиная от летнего солнцестояния, выходит из берегов и поднимается в течение приблизительно 100 дней; по истечении же этого срока вода снова спадает, ... и затем низкий уровень воды сохраняется целую зиму, вплоть до следующего летнего солнцестояния.

Геродот, История

Реферат

В статье рассматриваются некоторые свойства и примеры употребления планировщика заданий, появившегося в версии Oracle 10 на смену старому.

Введение

СУБД Oracle – большой и сложный механизм, требующий выполнения определенных плановых работ, таких как сбор статистики о хранимых объектах или сбор/чистка внутренней информации. Необходимость осуществлять плановый запуск работ могут испытывать и пользователи БД.

Первый механизм планового запуска появился в версии 7 для поддержки автоматических обновлений снимков (snapshots), как поначалу именовались нынешние материализованные виртуальные таблицы (materialized views). В версии 8 этот механизм был открыт для обычных пользователей через посредство некоторых параметров СУБД, таблиц словаря-справочника, а также пакета DBMS_JOB. Пакет DBMS_JOB позволял (и позволяет) запускать хранимую процедуру, или же неименованный блок PL/SQL в моменты времени, вычисляемые по указанной пользователем формуле.

К версии 10 такое устройство имевшегося планировщика заданий было сочтено слишком примитивным и к нему добавился новый, значительно более проработанный. Он использует следующие основные понятия:

- *Schedule* (расписание)
- *Program* (программа)
- *Job* (плановое задание = расписание + программа)

Кроме того, с ним связаны дополнительные, более специфичные понятия:

- *Job class* (класс заданий)
- *Window* и *window group* (ресурсное «окошко», интервал для автоматического включения ресурсного плана СУБД и группа окошек)
- *Chain* (цепочка заданий)
- *Event schedule* (возможность запустить задание по событию, зафиксированному по сообщению из очереди AQ)

В отличие от старого планировщика, в новом «программой» может быть не только блок PL/SQL, но и хранимая процедура на PL/SQL или на Java, внешняя процедура на C или даже команда ОС. Последнее означает, что Oracle отменяет необходимость использовать специфичные для разных платформ

планировщики заданий ОС (*cron*, *at*) при построении БД-центричного приложения. Вдобавок, сам запуск заданий получил возможность учета текущей вычислительной обстановки в СУБД, а также желаемой приоритетности среди прочих заданий.

Как и в случае со старым планировщиком, новый по сути представляет собой элемент ядра СУБД, доступ пользователя к которому предоставляется посредством программной логики и элементов схемы БД. Именно, в распоряжении пользователя имеется следующее:

- таблицы словаря-справочника LIKE '%SCHEDULER_%' (DBA_SCHEDULER_JOBS, DBA_SCHEDULER_JOB_LOG и прочие);
- несколько типов объектов хранения, как-то:
  JOB
  SCHEDULE
  PROGRAM
  JOB CLASS,
  ряд других;
- системные привилегии:
  CREATE SESSION
  CREATE JOB
  CREATE ANY JOB
  EXECUTE ANY PROGRAM
  EXECUTE ANY CLASS
  MANAGE SCHEDULER
  CREATE EXTERNAL JOB,
  и объединяющая их роль SCHEDULER_ADMIN;
- объектные привилегии:
  EXECUTE
  ALTER
  ALL,
  распространяющиеся на объекты типов JOB, SCHEDULE, PROGRAM и JOB CLASS;
- пакет DBMS_SCHEDULER.

Версия 11 дополнила планировщик возможностями:

- запуска «легковесных» заданий, делающая реальным их создание и удаление сотнями за секунду;
- запуска заданий на удаленных машинах посредством использования специального агента;
- запуска заданий только на основной БД физического горячего резерва или на страхующей логического.

Некоторые ключевые моменты использования планировщика в Oracle 10 рассматриваются ниже на примерах.

Простой запуск задания

Простой запуск задания очень напоминает запуск с помощью процедуры SUBMIT из пакета DBMS_JOB. Однако он возможен, в отличие от SUBMIT, только при наличии привилегии CREATE JOB. В последующих примерах созданием заданий и управлением ими для простоты будет заниматься пользователь SCOTT, хотя в жизни разумно подумать об отдельном администраторе для этой цели. Выдадим нужную привилегию пользователю SCOTT:

```
CONNECT / AS SYSDBA
GRANT CREATE JOB TO scott;
```

Кроме системных привилегий использование планировщика регулируется объектными привилегиями EXECUTE, ALTER и ALL, выдача которых применительно (GRANT … ON) к заданию, программе, расписанию или классу заданий позволяет работать с объектами БД соответственно типов JOB, PROGRAM, SCHEDULE и JOB CLASS, введенных в Oracle 10 вместе с новым планировщиком.

Ввиду того, что в дальнейшем предполагаются эксперименты с изменениями зарплаты сотрудников, будет удобно исходную зарплату сохранить:

```
CONNECT scott/tiger
```

```
ALTER TABLE emp ADD ( oldsal NUMBER );
UPDATE emp SET oldsal = sal;
COMMIT;
```

Внутреннее задание для СУБД

Пример внутреннего задания в виде неименованого блока PL/SQL:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB
( job_name   => 'simple_job'
, job_type   => 'PLSQL_BLOCK'
, job_action => 'UPDATE emp SET sal = sal + 1;'
, enabled    => TRUE
);
END;
/
```

Обратите внимание:

1) Обрамлять блок словами BEGIN и END не обязательно, так как код пакета DBMS_SCHEDULER это сделает самостоятельно (ради особой программной логики, добавляемой им к тексту пользователя).
2) Задание запускается в этом же сеансе и сопровождается неявной выдачей COMMIT. В этом легко удостовериться:

   ```
   COMMIT;
   UPDATE emp SET sal = sal + 1;
   BEGIN DBMS_SCHEDULER.CREATE_JOB (... как выше ...) END;
   ROLLBACK;
   ```

   Зарплата SAL увеличится на 2. Проверить это в качестве *упражнения*.

Для хранимой процедуры задание формируется аналогично:

```
CREATE PROCEDURE updatesal AS BEGIN UPDATE emp SET sal = sal - 1; END;
/

BEGIN
DBMS_SCHEDULER.CREATE_JOB
( job_name   => 'simple_job'
, job_type   => 'STORED_PROCEDURE'
, job_action => 'updatesal'
, enabled    => TRUE
);
END;
/
```

Обратите внимание, что нам не потребовалось удалять старое задание SIMPLE_JOB, так как при выбранных нами параметрах процедуры CREATE_JOB задания (и первое, и второе) прогонялись однократно, моментально и сразу же удалялись автоматически. Последнее как раз можно и отменить посредством не использованного в примере выше параметра AUTO_DROP.

В случае невозможности запустить задание СУБД, подобно как для старого планировщика (пакет DBMS_JOB), будет делать повторные попытка, но только по несколько иной схеме: через секунду, затем через 10 секунд, затем через 100 и далее – всего 6 раз, если только до этого не наступит очередной плановый момент.

Внешнее задание (для ОС)

Совсем новым в планировщике Oracle 10 является возможность запускать плановые задания в ОС. Однако, чтобы это было возможно, в ОС должна быть запущена программа *extjob* из ПО СУБД. На Windows она запускается службой *OracleJobScheduler<имя_СУБД>*. Для того, чтобы следующий пример проработал,

службу необходимо запустить. Вдобавок потребуется выдать пользователю SCOTT еще одну привилегию.

Пример запуска:

```
CONNECT / AS SYSDBA
GRANT CREATE EXTERNAL JOB TO scott;

CONNECT scott/tiger
BEGIN
DBMS_SCHEDULER.CREATE_JOB
( job_name   => 'simple_job'
, job_type   => 'EXECUTABLE'
, job_action => 'cmd.exe /C dir > \temp\out.txt'
, enabled    => TRUE
);
END;
/
```

Обратите внимание, что в Windows выдача команды ОС, или же запуск командного файла напрямую (без вызова *cmd.exe*), не проходит.

В Unix аналогичное действие можно записать как `'ls > /tmp/out.txt'`.

Возможности запуска, наблюдения, вмешательства

Так же, как для пакета DBMS_JOB, в новом планировщике предусмотрено именно плановое, а не одноразовое исполнение задания. Добавим к последнему вызову параметр:

```
, start_date => SYSTIMESTAMP + INTERVAL '10' SECOND
```

В результете корневой файл *out.txt* получим через 10 секунд после создания задания. Добавим еще параметр:

```
, repeat_interval => 'FREQ=MONTHLY; BYDAY=SUN, -1 SAT'
```

В результате задани будет исполняться ежемесячно по воскресениям и последним субботам месяца. В отличие от DBMS_JOB, DBMS_SCHEDULER, в дополнение к возможности употребить выражение на PL/SQL, имеет для формулирования графика запуска еще и специальный язык. Он позволяет указывать *частоту*, *интервал* и *уточнитель* запуска задания. Примеры:

`FREQ=HOURLY;INTERVAL=4`
– каждые 4 часа;
`FREQ=HOURLY;INTERVAL=4;BYMINUTE=10;BYSECOND=30`
– каждые 4 часа на 10-й минуте, 30-й секунде;
`FREQ=YEARLY;BYYEARDAY=-276`
– каждое 31-е марта;
`FREQ=YEARLY;BYMONTH=MAR;BYMONTHDAY=31`
– каждое 31-е марта;

Для проверки правильности составления выражения можно воспользоваться специальной процедурой:

```
DECLARE next_run_date TIMESTAMP;
BEGIN
DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING (
  'FREQ=HOURLY;INTERVAL=4;BYMINUTE=10;BYSECOND=30'
, SYSTIMESTAMP
, NULL
, next_run_date
);
DBMS_OUTPUT.PUT_LINE ( 'next_run_date: ' || next_run_date );
END;
/
```

Полное описание языка приводится в документации по Oracle.

Если указать план запуска, задание появится в системе уже надолго. Удалить его при необходимости можно будет так:

```
EXECUTE DBMS_SCHEDULER.DROP_JOB ( 'simple_job', TRUE )
```

Информация об имеющихся заданиях пользователь SCOTT может посмотреть запросом:

```
SELECT job_name, state, enabled FROM user_scheduler_jobs;
```

Более подробную информацию SCOTT обнаружит в таблицах USER_SCHEDULER_%, а более общую – в обычной таблице USER_OBJECTS.

Скомпонованное задание

Более развитая возможность DBMS_SCHEDULER позволяет скомпоновать задание из независимых элементов: программы и расписания. Характерная особенность в том, что оба эти элемента самостоятельны; их можно комбинировать в разных заданиях и изменять, не внося изменний в определения заданий.

Создание программы

Простой пример создания программы:

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM
( program_name   => 'simple_program'
, program_type   => 'STORED_PROCEDURE'
, program_action => 'updatesal'
, enabled        => TRUE
);
END;
/
```

Список сведений об имеющихся программах для планировщика имеется в таблицах DBA/ALL/USER_SCHEDULER_PROGRAMS.

Другими значениями параметра PROGRAM_TYPE могут быть 'PLSQL_BLOCK' и 'EXECUTABLE' (см. выше).

При наличии у процедуры параметров их количество потребуется указать особо:

```
CREATE PROCEDURE salary ( decr NUMBER ) AS
BEGIN UPDATE emp SET sal = sal - decr;
END;
/

BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM
( program_name        => 'simple_program1'
, program_type        => 'STORED_PROCEDURE'
, program_action      => 'salary'
, enabled             => FALSE
, number_of_arguments => 1
);
END;
/
```

Обратите внимание, что программа создана «отключенной». Дело в том, что указать фактические значения параметрам программе во «включенном» состоянии нельзя, так что последовательность действий будет следующая:

```
BEGIN
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT
( program_name      => 'simple_program1'
, argument_position => 1
, argument_name     => 'DELTA'
, argument_type     => 'NUMBER'
);
END;
/

EXECUTE DBMS_SCHEDULER.ENABLE ( 'simple_program1' )
```

Создание расписания

Пример создания расписания:

```
BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE
( schedule_name   => 'simple_schedule'
, start_date      => SYSTIMESTAMP
, repeat_interval => 'FREQ=WEEKLY; BYDAY=MON, TUE, WED, THU, FRI'
, end_date        => SYSTIMESTAMP + INTERVAL '1' MONTH
);
END;
/
```

В общем случае язык указания графика для расписания (параметр REPEAT_INTERVAL) допускает ссылаться на ранее созданные таким же образом расписания.

Список сведений об имеющихся расписаниях для планировщика имеется в таблицах DBA/ALL/USER_SCHEDULER_SCHEDULES.

Простой пример скомпонованого задания

Из самостоятельно существующих программы и расписания можно составить задание:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB
( job_name      => 'compound_job'
, program_name  => 'simple_program'
, schedule_name => 'simple_schedule'
, enabled       => TRUE
);
END;
/
```

При наличии параметра пример может выглядеть так:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB
( job_name      => 'compound_job1'
, program_name  => 'simple_program1'
, schedule_name => 'simple_schedule'
, enabled       => FALSE
);
END;
/

BEGIN
DBMS_SCHEDULER.SET_JOB_ANYDATA_VALUE
( job_name       => 'compound_job1'
, argument_name  => 'DELTA'
```

```
, argument_value => ANYDATA.CONVERTNUMBER ( 3 )
);
END;
/

EXECUTE DBMS_SCHEDULER.ENABLE ( 'compound_job1' )
```

Обратите внимание, что в этом случае задание сначала создается выключенным, и только после указания значения параметра программе оно может включаться.

Создание и использование ресурсного окошка СУБД для задания

Очередной запуск внутреннего задания может прийтись на время активного использования ресурсов СУБД другими процессами. Это может замедлить выполнение планового задания, обесценить его выполнение и даже поставить его выполнение под угрозу. Справиться с этой проблемой призвано ресурсное окошко. Суть его состоит в одновременном с очередным запуском задания автоматическом переключением СУБД на работу по требуемому ресурсному плану (план поведения для распределителя ресурсов СУБД. *resource manager*). Сами ресурсные окошки (windows) принадлежат схеме SYS, но создавать их разрешено и другим пользователям при наличии соответствующей привилегии:

```
CONNECT / AS SYSDBA
GRANT MANAGE SCHEDULER TO scott;
```

Ресурсный план построить несложно, но чтобы не отвлекаться, воспользуемся встроенным в любую БД планом SYSTEM_PLAN (см. таблицу DBA_RSRC_PLANS). Тогда создание окошка может выглядеть так:

```
CONNECT scott/tiger

BEGIN
DBMS_SCHEDULER.CREATE_WINDOW (
  window_name     => 'my_job_window'
, resource_plan   => 'SYSTEM_PLAN'
, start_date      => SYSTIMESTAMP
, repeat_interval => 'FREQ=MINUTELY; INTERVAL=3'
, duration        => INTERVAL '1' MINUTE
);
END;
/

EXECUTE DBMS_SCHEDULER.ENABLE ( 'sys.my_job_window' )
```

Теперь каждые три минуты на минуту будет включаться ресурсный план SYSTEM_PLAN. Это легко наблюдать, выдав несколько раз от имени SYS:

```
COL window_name FORMAT A20
COL log_date    FORMAT A40
SELECT * FROM (
   SELECT   log_date, window_name, operation
   FROM     dba_scheduler_window_log
   ORDER BY log_date DESC
)
WHERE ROWNUM <= 10
;
```

Если подгадать момент, когда значение OPERATION для окошка MY_JOB_WINDOW станет OPEN, от имени SYS можно будет удостовериться, что план включен:

```
SYS> SHOW PARAMETER resource

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
resource_limit                       boolean     FALSE
resource_manager_plan                string      SCHEDULER[0xD5A4]:SYSTEM_PLAN
```

Чтобы связать с этим периодически открывающимся ресурсным окошком задание, пользователю SCOTT достаточно указать его вместо расписания:

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
  job_name      => 'my_window_job'
, program_name  => 'simple_program'
, schedule_name => 'sys.my_job_window'
, enabled       => FALSE
);
END;
/

EXECUTE DBMS_SCHEDULER.ENABLE ( 'my_window_job' )
```

С этого момента можно наблюдать уменьшение зарплат сотрудников, осуществляемое каждые три минуты в условиях благоприятного ресурсного плана. В качестве варианта, в процедуре CREATE_WINDOW можно было сослаться и на какое-то существующее расписание.

Так как СУБД в каждый момент времени умеет работать только по одному ресурсному плану, окошки «не умеют» перекрываться, а могут только переопределять своим планом другой, ранее установленый и пришедшийся на то же время.

В качестве развития этой темы Oracle позволяет создавать именованый список существующих окошек и под маркой «группы» (window group) указывать его заданию значением параметра SCHEDULE_NAME, то есть там, где у нас было указано имя окошка.

Изменение свойств объектов планировщика

Хотя упомянутые JOB, SCHEDULE, PROGRAM, WINDOW и проч., причисляются к объектам хранения БД (и видны в таблицах DBA/ALL/USER_OBJECTS), не только их создание и удаление, но и изменение свойств выполняются так, как было удобно разработчику: через API. Для всех перечисленных видов объектов существует довольно много поведенческих свойств, указанию которых нет места в процедурах LIKE 'CREATE_%'. Устанавливать их явно следует единой для всех процедурой SET_ATTRIBUTE. Вот пример, как для задания MY_WINDOW_JOB (а) задать приоритет выполнения (по отношению к другим заданиям своего класса), если на одно время пришлось выполнение нескольких заданий одновременно, и (б) потребовать прекращения (процедурой STOP_JOB), если оно еще не выполнилось, а ресурсное окошко уже закрылось:

```
EXECUTE DBMS_SCHEDULER.DISABLE ( 'my_window_job', TRUE )

BEGIN
DBMS_SCHEDULER.SET_ATTRIBUTE (
  name      => 'my_window_job'
, attribute => 'JOB_PRIORITY'
, value     => 1
);
DBMS_SCHEDULER.SET_ATTRIBUTE (
  name      => 'my_window_job'
, attribute => 'STOP_ON_WINDOW_CLOSE'
, value     => TRUE
);
END;
/

EXECUTE DBMS_SCHEDULER.ENABLE ( 'my_window_job' )
```

Полный список атрибутов и объектов, к которым они применимы, имеется в документации по Oracle.

Заключение

Помимо использованного выше общения с планировщиком Oracle 10 средствами PL/SQL и SQL, общаться с

ним можно через графический интерфейс Oracle Enterprise Manager. По сути OEM ничего нового не дает, так как в конечном итоге отсылает к СУБД те же команды на PL/SQL и SQL, но выполнение разовых действий через OEM часто администратору быстрее и понятнее. Для автоматизации работ однако лучше может подойти работа со сценариями запросов.

После проведенных опытов с планировщиком не забудьте освободить БД от ненужных объектов. Например:

```
EXECUTE DBMS_SCHEDULER.DROP_JOB ( 'my_window_job', force => TRUE )
EXECUTE DBMS_SCHEDULER.DROP_WINDOW ( 'my_job_window', force => TRUE )
```

Удаление прочих созданных ранее объектов и изъятие выданых пользователю SCOTT привилегий предлагается сделать самостоятельно в виде *упражнения*, воспользовавшись таблицами словаря-справочника и, при надобности, документацией. Можно также использовать OEM.