DBMS_SCHEDULER

정현호
http://www.commit.co.kr
admin@commit.co.kr
cleanto@naver.com

Agenda

1. Introduction
2. JOB
3. Managing Scheduler Components
4. CLASS
5. CHAIN
6. The backup by using scheduler

1. Introduction

1. Introduction to Scheduler

2. DBMS_SCHEULER 특징

1. Introduction to Scheduler

왜 scheduler가 필요할까요?

일상적인 DB관리업무 혹은 APPLICATION LOGIC은 "특정 작업을 일정한 간격으로 실행함" 을 필요로 합니다.
예를 들어 어떤 BUSINESS행사기간에 평상시보다 많은 리소스지원을 가지고 최우선 처리되어야 할 작업이 존재할 수 있습니다.

이를 위해 DBA는 특정 시간대에 DB관리업무를 지원할 수 있어야 하는데, 이러한 작업들은 보통 복잡하고 오랜 작업시간을 요구합니다 이런 측면에서 DATABASE SCHEDULER를 통해서, 손쉽게 관리 및 해결이 가능합니다

데이타베이스 외부의 운영체제 파일 및 실행파일을 사용 할 수 있어 OS의 스케줄링 프로그램 이용시 발생할 수 있는 보안적 issue 또한 해결할 수 있습니다

사용의 예

- 하루에 두번씩 테이블과 인덱스의 통계 계산
- materialized view refresh를 통한 테이블 데이터 복제
- 데이터베이스의 일일 백업 수행
- 매달 마지막일 Report생성
- 10분마다 Event 체크
- 오전 4시 Batch를 실행
- 부적합한 서버 엑세스 시도에 대한 시간별 보고서 생성

2. DBMS_SCHEULER 특징

Scheduler를 이용하면 PL/SQL 프로그램뿐 아니라 OS 유틸리티와 프로그램 및 파일을 실행하는 것이 가능합니다

Scheduler는 자연 언어(natural language)를 사용하여 실행 주기를 정의할 수 있습니다
(스케줄을 매 30분 단위로 실행하고자 하는 경우, (PL/SQL 문법 대신) 자연 언어 형식의 표현을 사용하여 REPEAT_INTERVAL 매개변수를 사용할 수 있습니다)

DBMS_SCHEDULER는 이벤트 기반 스케줄링이 가능합니다
- DB 내부적인 특이사항(event)에 대해서 스케줄링이 가능하기에 OS 스케줄러, DBMS_JOB 에 비해서 활용 용도가 좋습니다

OS 파일을 사용할 수 있기에 기존에 OS 스케줄을 이용하면서 스크립트에 계정명/비밀번호 를 기술해야 했던 부분에 대해서 보완 할 수 있습니다

Job에 대한 naming 이나 comment 를 생성할 수 있습니다

2. JOB

1. SCHEDULER 구성

2. SCHEDULER의 개념과 구조

3.COMPONENT의 사용법

4. JOB 생성

5. JOB 조회

6. AUTO_DROP

7. JOB에서 작업 수행간격을 지정하는 방법

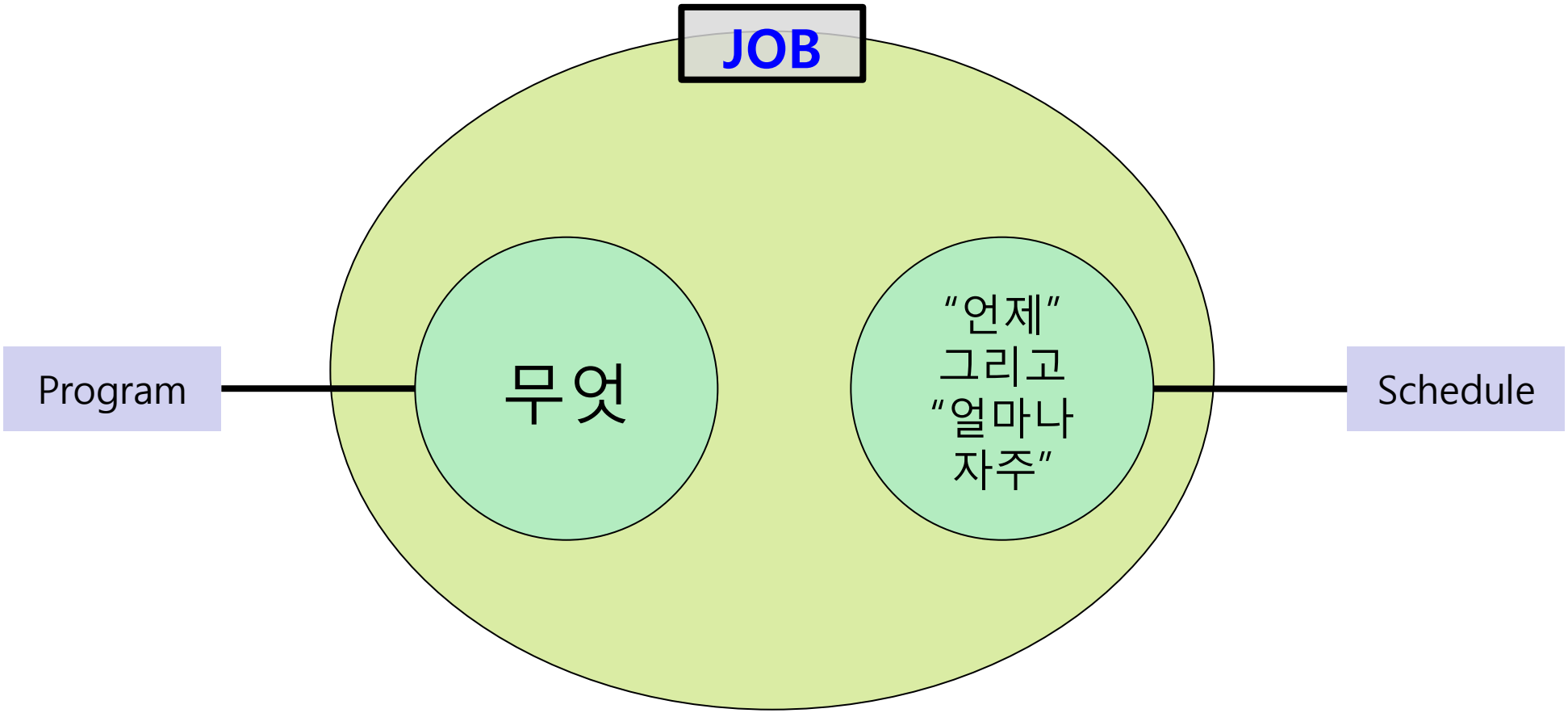
8. JOB – use program, schedule

9. Schedule caution

1. SCHEDULER 구성

Scheduler는 oracle환경에서 작업을 관리하기 위해 모듈화 된 접근방법을 제공합니다.

대표적으로 JOB이 스케줄을 수행하게 되고 그안에 '무엇' 이라는 것 과 '언제', '얼마나 자주' 라는 부분이 들어가게 되며 각각 program 과 schedule 로 매칭이 됩니다



2. SCHEDULER의 개념과 구조

PROGRAM

수행 가능한 파일, 스크립트나 프로시저 들에 대한 정의가 내려져 있습니다
즉, 이 객체를 통해서 사용자는 "무엇"을 수행할 것인지 결정합니다.

SCHEDULE

"언제" 그리고 "얼마나 자주" JOB을 수행할 것인가를 결정합니다.
사용자는 생성된 SCHEDULE을 하나 혹은 여러 개의 JOB에서 사용할 수 있습니다.

JOB

PROGRAM과 SCHEDULE을 통해 "무엇"을 "언제" 실행할 것인가를 정의합니다.
여기서 "무엇"은 PL/SQL 프로시저, BINARY 실행화일, JAVA APPLICATION, SHELL SCRPT등 이 될 수 있습니다.

JOB 생성시 미리 생성된 PROGRAM과 SCHEDULE을 사용할수 있으며
JOB 생성시 따로 기술하여 만들 수도 있습니다.

3. COMPONENT의 사용법

JOB은 크게 5가지 방법으로 생성할 수 있습니다

1. Program과 schedule 모두 job생성시 **직접** 입력해주는 방법.

2. "무엇"을 할 것인가를 직접 입력하고, 기존에 생성해둔 schedule을 사용하는 방법.

3. 생성해 둔 program을 사용하고, "언제"/ "얼마나 자주" 를 직접 입력하는 방법 등이 있습니다.

4. 생성해 둔 program과 schedule을 사용하여 job을 생성하는 방법.

5. 특정 조건을 만족하는 경우 실행되는 Event 형 JOB

4. JOB 생성

JOB 생성 시 '무엇' 에 관한 부분과 '언제', '얼마나 자주' 에 대해서
직접 입력 하여 생성하는 방법에 대한 예시 입니다

이와 같이 시간에 의한 JOB의 실행이 되므로 Time Base Scheduler 라고 할수 있습니다

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'test_self_contained_job',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats(''SCOTT''); END;',
start_date => SYSTIMESTAMP,
repeat_interval => 'freq=hourly; byminute=0',
comments => 'Job created using the CREATE JOB procedure.');
End;
/
```

JOB 이름을 지정하는 JOB_NAME

JOB 시작되는 시일을 알수있는 START_DATE

반복간격을 알수있는 REPEAT_INTERVAL

JOB_TYPE의 PARAMETER는 다음과 같이 5가지 값 중에서 하나를 가집니다.

4. JOB 생성

JOB의 JOB_TYPE 의 종류

1. PLSQL_BLOCK: 누구나 수행할 수 있는 PL/SQL블록을 의미합니다.
2. STORED_PROCEDURE: 유효한 이름으로 명명된 PL/SQL, JAVA, 혹은 외부 프로시저를 사용할 때 지정합니다.
3. EXECUTABLE: OS 에서 수행 가능한 COMMAND 나 스크립트 파일 등을 나타냅니다
4. PROGRAM_NAME : 생성해놓은 program 을 이용 합니다
5. Chain : 생성된 Chain 을 이용합니다

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'test_self_contained_job',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats(''SCOTT''); END;',
start_date => SYSTIMESTAMP,
repeat_interval => 'freq=hourly; byminute=0',
comments => 'Job created using the CREATE JOB procedure.');
End;
/
```

4. JOB 생성

START_TIME 파라미터의 값이 SYSTIMESTAMP이므로 JOB 실행시 즉시 수행되며, REPEAT_INTERVAL의 값에 의해 반복해서 작업을 수행합니다.
JOB의 시작을 즉시가 아닌 특정시간부터 시작할수도 있습니다

start_date => to_timestamp_tz('2010-07-07 09:00:00 ROK', 'YYYY-MM-DD HH24:MI:SS TZR')

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'test_self_contained_job',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats(''SCOTT''); END;',
start_date => SYSTIMESTAMP,
end_date => SYSTIMESTAMP + 30,
repeat_interval => 'freq=hourly; byminute=0',
comments => 'Job created using the CREATE JOB procedure.');
End;
/
```

COMMENTS
파라미터는 객체에 대한 설명을 기술하는 일종의 주석 입니다.

4. JOB 생성

END_DATE 파라미터가 설정되어 있지 않으면, 유효 기간 없이 최초 수행 후, freq=hourly 으로 설정 되어 있기 때문에 매 1시간마다 수행되게 됩니다.
END_DATE 는 아래와 같은 형식으로 작성이 가능합니다

```
end_date => SYSTIMESTAMP + 30

end_date => to_timestamp_tz('2010-07-08 10:50:00 ROK', 'YYYY-MM-DD HH24:MI:SS TZR  ')
```

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'test_self_contained_job',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats("SCOTT"); END;',
start_date => SYSTIMESTAMP,
end_date => SYSTIMESTAMP + 30,
repeat_interval => 'freq=hourly; byminute=0',
comments => 'Job created using the CREATE JOB procedure.');
End;
/
```

5. JOB 조회

JOB 목록 조회

```
Select P.JOB_NAME, P.JOB_TYPE, O.OBJECT_ID, P.ENABLED, CAST(P.NEXT_RUN_DATE AS DATE) next_run_date,
O.LAST_DDL_TIME, O.CREATED, P.STATE, P.JOB_CLASS ,O.OWNER ,schedule_type
FROM    SYS.DBA_OBJECTS O, SYS.DBA_SCHEDULER_JOBS P
WHERE   O.OWNER = P.OWNER
AND     O.OBJECT_NAME = P.JOB_NAME
AND     O.OBJECT_TYPE = 'JOB';
```

JOB 별 모니터링

```
select d.log_id, d.log_date, d.owner, l.operation, d.status
from dba_scheduler_job_run_details d, dba_scheduler_job_log l
where d.log_id=l.log_id and l.job_name= 'JOB이름입력'  and rownum between 1 and 50
order by d.log_date desc;
```

LOG ID 별 상세 조회

```
select d.log_id,d.log_date, d.job_name, l.operation, d.status, d.error#, d.req_start_date,
d.actual_start_date,d.run_duration, cpu_used, d.additional_info "Detail_Log"
from DBA_SCHEDULER_JOB_RUN_DETAILS d, dba_scheduler_job_log l
where d.log_id=l.log_id and l.job_name= 'JOB이름입력' and d.log_id=13925
order by d.log_date desc;
```

6. AUTO_DROP

JOB 에서 정해진 END_DATE 시간이 되었거나 REPEAT_INTEVAL 을 지정 하지않아 1회 실행 후 종료 되어있는 JOB 등, JOB 실행이 종료가 되면 JOB을 자동으로 삭제하는 속성 입니다

AUTO_DROP => TRUE 를 설정 함으로써 사용할 수 있습니다 .

AUTO_DROP

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'test_job',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats(''SCOTT''); END;',
start_date => systimestamp,
end_date => systimestamp +10,
auto_drop => true,
comments => 'attribute_null_test');
End;
/
```

end_date 가 10일 후 이기에 10일 후 JOB이 종료 되면 JOB은 자동 삭제가 됩니다

6. AUTO_DROP

MAX_RUNS 속성을 이용하여 해당 JOB 이 수행하는 횟수를 지정하고 해당 횟수에 도달하면 JOB 이 자동 삭제가 되게도 할수 있습니다

end_date 의 날짜가 되지 않았지만 **max_runs** 속성에 의해 5회 실행되고 JOB은 삭제 됩니다

AUTO_DROP

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'test_job',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats(''SCOTT''); END;',
start_date => systimestamp,
end_date => systimestamp + 10,
repeat_interval => 'FREQ=SECONDLY',
auto_drop => true,
enabled => true,
comments => 'attribute_null_test');
dbms_scheduler.set_attribute( name =>'test_job', attribute => 'max_runs',
value => 5);
End;
/
```

repeat_interval 이 매 1초 이기에 5초(5회 수행) 후 JOB은 삭제 가 됩니다

7. JOB에서 작업 수행간격을 지정하는 방법

JOB의 **repeat_interval** 에는 **CALEDNDARING EXPRESSION**과 **DATETIME EXPRESSION** 2가지 표현형식을 사용할 수 있습니다.

CALENDARING EXPRESSION

1) SCHEDULE 객체를 이용한다고 한다면
CALENDARING EXPRESSION 만 사용할 수 있습니다

2) CALENDARING EXPRESSION은 작업 사이의 시간간격 을 표기합니다

DATATIME EXPRESSION

1) JOB 에 직접 Repeat_interval 를 입력한다면
DATATIME EXPRESSION 과
CALENDARING EXPRESSION 둘다 가능합니다

2) DATATIME EXPRESSION 은 다음 실행시간을 표기 합니다

CALENDARING EXPRESSION 이 SCHEDULE 과 JOB에 직접 기술시 에도 사용 가능함으로 CALENDARING EXPRESSION 을 사용하는 것이 좋겠습니다

7. JOB에서 작업 수행간격을 지정하는 방법

CALENDARING EXPRESSION

repeat_interval=> 'FREQ=HOURLY; INTERVAL=4' (매 4시간 간격을 나타냅니다.)
repeat_interval=> 'FREQ=MINUTELY; INTERVAL=15' (매 15분의 간격을 나타냅니다.)

**** 작업간의 간격을 표기**

DATETIME EXPRESSION

repeat_interval=> 'SYSDATE + 1' (FREQ=DAILY와 같은 내용으로 다음날 수행됨을 지정합니다.)
repeat_interval=> 'SYSDATE + 15/1440'
(여기서 1440은 24시간을 다시 60분으로 곱한 값이므로, 결과적으로 15분 후 재실행을 의미합니다.)

**** 다음 작업까지의 시간 기술**

7. JOB에서 작업 수행간격을 지정하는 방법

Calendaring expression은 크게 3부분으로 구성되어 있습니다.

1. Frequency (mandatory)
2. Repeat Interval
3. Specifiers

Frequency를 기준으로 Repeat Interval과 Specifies항목을 조합하여, 주/일/시/분/초 단위로 job실행을 지정합니다.

Example
- 격 주 : FREQ=WEEKLY; INTERVAL=2
- 매5분 : FREQ=MINUTELY; INTERVAL=5
- 매 초 : FREQ=SECONDLY

좀 더 복잡한 설정 **BY 절**을 사용을 합니다 .
BY절을 사용하면 "4주 마다 화요일", "매주 화요일 6:24AM" 등의 설정이 가능해집니다.

"4주째 화요일"의 경우는 FREQ=YEARLY; BYWEEKNO=4,8,12,16,20,24,28,32,36,40,44,48,52;BYDAY=TUE;

"매주 화요일 아침 6:24"의 경우는 FREQ=WEEKLY;BYDAY=TUE; BYHOUR=6;BYMINUTE=24;

7. JOB에서 작업 수행간격을 지정하는 방법

1. Frequency (mandatory)

FREQ:XXXXX 에서 사용할수 있는 형식은 다음과 같습니다

YEARLY , MONTHLY , WEEKLY , DAILY , HOURLY , MINUTELY , SECONDLY

2. Repeat Interval

INTERVAL 은 1 부터 – 999 까지 지정 가능 합니다

7. JOB에서 작업 수행간격을 지정하는 방법

3. Specifiers

BYXXXX 에서 사용 할수 있는 형식은 다음과 같습니다

BY MONTH : JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC
BYWEEKNO : 1 ~ 53
BYYEARDAY : 1 ~ 366
BYMONTHDAY : 1 ~ 31
BYDAY : MON,TUE,WED,THU,FRI,SAT,SUN
BYHOUR : 0 ~ 23
BYMINUTE : 0 ~ 59
BYSECOND : 0 ~ 59

8. JOB – use program, schedule

미리 생성된 Program 과 Schedule 을 이용한 JOB 생성

PROGRAM

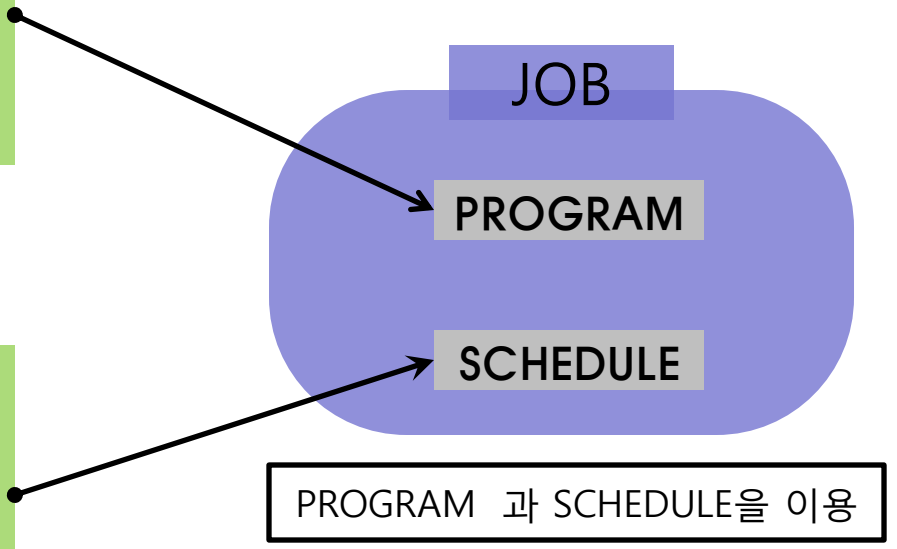
PROGRAM은 JOB을 구성하는 데 있어서 "무엇" 에 해당하는 COMPONENT입니다.

PROGRAM을 별도로 생성하여 사용할 경우 여러 JOB 에서 재 사용이 가능 합니다

SCHEDULE

"언제" 그리고 "얼마나 자주" JOB을 수행할 것인가를 결정합니다.

생성된 SCHEDULE을 하나 혹은 여러 개의 JOB에서 사용할 수 있습니다



8. JOB – use program, schedule

PROGRAM COMPONENT

PROGRAM을 생성할 때에는 **CREATE_PROGRAM()** 프로시저를 사용합니다.

프로그램은 기본적으로 **비활성 상태**로 생성됩니다
enable => true 는 프로그램이 활성화된 상태로 생성 될 수 있도록 지정 합니다

```
BEGIN
DBMS_SCHEDULER.create_program (
program_name => 'test_schema_program',
program_type => 'stored_procedure',
program_action => 'HR.UPDATE_SCHEMA_STATS',
Enabled => true,
comments => 'A program using a stored procedure');
END;
/
```

PROGRAM_NAME 에는 '**test_schema_program**' 이라는 program 이름을 지정하고 있습니다

PROGRAM_TYPE 은 **stored_procedure** 를 사용하고 있습니다

PROGRAM_ACTION는 수행될 UPDATE_SCHEMA_STATS 프로시저의 이름을 지정 하였습니다

8. JOB – use program, schedule

PROGRAM COMPONENT

```
BEGIN
DBMS_SCHEDULER.create_program (
program_name => 'test_schema_program',
program_type => 'stored_procedure',
program_action => 'HR.UPDATE_SCHEMA_STATS',
Enabled => true,
comments => ' A program using a stored procedure');
END;
/
```

PROGRAM_TYPE
은 3 가지 입니다

PROGRAM 의 PROGRAM_TYPE 종류

1) PLSQL_BLOCK : 누구나 수행할 수 있는 PL/SQL블록을 의미합니다

2) stored_procedure : 저장되어져 있는 procedure 를 이용합니다

3) Executable : OS Command 나 스크립트 파일을 실행을 합니다

8. JOB – use program, schedule

SCHEDULE COMPONENT

SCHEDULE 을 생성할 때에는 **CREATE_SCHEDULE()** 프로시저를 사용합니다.

repeat_interval 은 **CALENDARING EXPRESSION** 을 사용해야 합니다

```
BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE(
schedule_name => 'stats_schedule',
start_date => SYSTIMESTAMP,
end_date => SYSTIMESTAMP + 30,
repeat_interval => 'FREQ=HOURLY;INTERVAL=4',
comments => 'Every 4 hours');
END;
/
```

END_DATE에 명시한 바와 같이
오늘부터 30일간만 활성화됩니다

REPEAT_INTERVAL은
FREQ=HOURLY INTERVAL=4로
지정되어 있는데

이는 4시간 마다 해당 JOB을
수행함을 나타냅니다

8. JOB – use program, schedule

JOB COMPONENT

JOB은 JOB_NAME 파라미터에서 설정한 바와 같이 TEST_PROGRM_SCHEDULE_JOB이라는 이름을 가집니다.

```
BEGIN

DBMS_SCHEDULER.create_job (
job_name => 'test_program_schedule_job',
program_name => 'test_schema_program',
schedule_name => 'stats_schedule',
comments => 'Job created using an existing program and schedule');
End;
/
```

PROGRAM_NAME 에서는 이미 생성된 'test_schema_program' 이라는 program을 지칭하고 있습니다

SCHEDULE 에서는 'stats_schedule' 이라는 이미 생성된 SCHEDULE 을 사용하고있습니다

9. Schedule caution

Time based Schedule을 생성할 때에는 몇 가지 주의사항

1. Start time과 end time을 언급할때, 사용할 때 사용하는 시간은 TIMESTAMP WITH TIME ZONE 데이터타입을 사용합니다.

2. Repeat_interval설정에는 calendaring exression만 사용합니다.
(Job에서 직접 기술할때는 Calendaring expression 와 Datetime exression 모두 사용가능 합니다)

3. End Time을 설정하지 않는 경우라면 주어진 interval을 기준으로 계속해서 실행되지만 End Time을 설정하게 되면, End Time 이후에는 schedule 자체가 비활성화 됩니다.

4. REPEAT_INTERVAL이 언급되지 않았다면 이 **JOB은 1회성 JOB**임을 나타냅니다

3. Managing Scheduler Components

1. Management

2. COMPONENT ENABLE / DISABLE

3. ATTRIBUTE

4. JOB Management

5. PROGRAM Management

6. SCHEDULE Management

7. COMPONENT DROP TEST

1. Management

DBMS_SCHEDULER패키지의 다양한 프로시저를 사용하여 SCHEDULER COMPONENT관리

Enable 과 Disable

Schedule 을 제외한 program 이나 Job 등의 객체는 생성시 Default 로 비활성화 되어 생성 되기에 사용할 수 없는 상태 입니다(생성시 지정하지 않으면 기본이 Disable 상태)

그렇기에 해당 객체를 사용하기 위하여 활성화를 해줘야 합니다 생성시 활성화 하여 생성 할수도 있으며 차후에 비활성 에서 활성화 상태로 변경 할 수 있습니다

RUN 과 STOP

Program 이나 Chain 등은 JOB에 속해져서 실행되어 지기에 JOB 객체에 대해서만 RUN 과 STOP 프로시저를 이용하여 객체를 시작 하거나 중지 합니다

중지되어있는 JOB을 시작하기 위해서는 RUN_JOB 프로시저를,
구동되어있는 JOB을 멈추기 위한 STOP_JOB 프로시저를 이용합니다

**** job생성 후 구동은 run_job 프로시저를 사용해서 구동 하는 것이 아닌 job을 활성화가 시킨후 start_date 시간에 job이 구동 됩니다**

2. COMPONENT ENABLE / DISABLE

DBMS_SCHEDULER패키지의 다양한 프로시저를 사용하여 SCHEDULER COMPONENT관리

객체 활성화 / 비활성화

활성화는 ENABLE()프로시저를, 비활성화는 DISABLE()프로시저를 사용하여 설정합니다.

1) ENABLE()프로시저를 사용하여 HR스키마 내의 CALC_STATS program을 활성화하는 과정을,
2) DISABLE()프로시저를 사용하여 HR스키마 내의 GET_STATS라는 JOB을 비활성화하는 과정을 입니다

1)

CALC_STATS **프로그램 활성화**
EXEC DBMS_SCHEDULER.ENABLE('HR.CALC_STATS');

2)

GET_STATS JOB **비활성화**
EXEC DBMS_SCHEDULER.DISABLE ('HR.HR_STATS');

3. ATTRIBUTE

DBMS_SCHEDULER패키지의 다양한 프로시저를 사용하여 SCHEDULER COMPONENT관리

JOB의 속성 관리

SET_ATTRIBUTE() 프로시저를 사용할 때는 JOB의 이름을 **NAME** 파라미터에,
바꾸고자 하는 속성의 이름을 **ATTIRIBUTE** 라는 파라미터에, 변경 하려는 값을 **VALUE** 파라미터에 지정합니다.

변경되는 객체가 현재 ENABLE상태라면, SET_ATTRIBUTE()프로시저가 수행되는 순간 DISABLE 상태로 바뀐 뒤 PARAMETER를 변경합니다 . 그리고 변경작업이 완료되면 자동으로 ENABLE상태로 되돌아옵니다.

JOB의 속성 변경

```
EXEC DBMS_SCHEDULER.SET_ATTRIBUTE(name => 'hourly_sche', attribute=> 'repeat_interval', -
value => 'freq=hourly; byminute=0')
```

예제는 **hourly_sche** 이름의 스케줄의 속성 중 **repeat_interval** 의
값을 **freq=hourly;byminute=0** 으로 변경 하고 있습니다

3. ATTRIBUTE

DBMS_SCHEDULER패키지의 다양한 프로시저를 사용하여 SCHEDULER COMPONENT관리

JOB의 속성 삭제

SET_ATTRIBUTE_NULL 프로시저를 시용하여 부여된 속성을 지울수 가 있습니다

파라미터의 값을 NULL, 즉 UNSET 으로 설정하려고 할 때 SET_ATTRIBUTE_NULL 프로시저를 이용합니다

```
JOB 의 Comment 의 내용 삭제
 EXEC DBMS_SCHEDULER.SET_ATTRIBUTE_NULL('test_job','comments');
```

예제는 **test_job**이름의 JOB 속성 중 comment 의 값을 UNSET 시키는 과정입니다

4. JOB Management

DBMS_SCHEDULER패키지의 다양한 프로시저를 사용하여 SCHEDULER COMPONENT관리

JOB 관리

RUN_JOB()프로시저가 **JOB_NAME**만 옵션으로 가지는 것과 달리, DROP_JOB 와 STOP_JOB()프로시저는 JOB_NAME 이외에도 **FORCE** 라는 옵션 를 가집니다. FORCE의 경우 default 는 FALSE입니다.

JOB 실행

```
EXEC DBMS_SCHEDULER.RUN_JOB('HR.JOB1');
```

JOB 멈춤

```
EXEC DBMS_SCHEDULER.STOP_JOB(JOB_NAME=>'HR.JOB1', FORCE=> TRUE);
```

** STOP_JOB()프로지셔의 경우 **FORCE=TRUE**로 설정하게 되면, STOP을 시도할 경우 JOB이 실행 중이더라도 바로 JOB SLAVE를 KILL하게 됩니다.

JOB 삭제

```
EXEC DBMS_SCHEDULER.DROP_JOB(job_name =>'HR.JOB1', FORCE=> TRUE);
```

** DROP_JOB()프로시저의 경우도 FORCE=TRUE로 설정하게 되면, 현재 JOB이 실행 중이더라도 바로 STOP한 후 DROP 합니다

5. PROGRAM Management

DBMS_SCHEDULER패키지의 다양한 프로시저를 사용하여 SCHEDULER COMPONENT관리

Program 관리

프로그램 객체를 활성화 혹은 비활성화하는 방법은 동일하게 ENABLE()프로시저, 혹은 DISALBE()프로시저를 사용 합니다

DROP_PROGRAM()프로시저는 PROGRAM이 JOB에 서 사용중이라면 기본적으로 에러를 나타내며 DROP작업은 수행되지 않습니다. 그러나 **FORCE=> TRUE** 옵션을 사용하여 삭제 할수 있습니다

프로그램 활성화

```
EXEC DBMS_SCHEDULER.ENABLE('HR.PROG1');
```

프로그램 비활성화

```
EXEC DBMS_SCHEDULER.DISABLE('HR.PROG1');
```

** 사용되고 있는 program을 비활성화 시키기 위해서는 FORCE=TRUE 옵션을 사용해야 합니다

프로그램 삭제

```
EXEC DBMS_SCHEDULER.DROP_PROGRAM( PROGRAM_NAME =>'HR.PROG1', FORCE=> TRUE);
```

**** FORCE=> TRUE**설정한다면 DROP이 시도할 때 프로그램을 참조하고 있는 모든 JOB이 비활성 되고, PROGRM을 DROP시킵니다.

6. SCHEDULE Management

DBMS_SCHEDULER패키지의 다양한 프로시저를 사용하여 SCHEDULER COMPONENT관리

Schedule 관리

SCHEDULE은 시간에 대한 정보를 기록하고 있는 OBJECT일뿐이므로, 변경과 삭제 작업만 제공됩니다.

첫 번째 예제는 **SET_ATTRIBUTE()**프로시저를 사용하여 START_DATE PARAMETER값을 변경하는 작업을,
두 번째 예제는 **DROP_SCHEDULE()**프로시저를 사용하여 SCHEDULE 객체를 DROP하는 작업을 보여주고 있습니다.

스케줄 수정

```
EXEC DBMS_SCHEDULER.SET_ATTRIBUTE( name => 'HR.STATS_SCHEDULE', attribute =>'START_DATE', -
value => '01-JAN-2004 9:00:00 US/Pacific')
```

스케줄 삭제

```
EXEC DBMS_SCHEDULER.DROP_SCHEDULE( schedule_name => 'HR.STATS_SCHEDULE', -
force => TRUE);
```

** FORCE 값이 TRUE로 설정된 경우라면 이를 사용하고 잇는 JOB이나 WINDOW가 모두 비활성화 되고, SCHEDULE은 DROP됩니다.

7. COMPONENT DROP TEST

JOB 을 구성하는 객체인 SCHEDULE 과 PROGRAM 을 삭제시 구동중인 JOB이 어떻게 되는지 확인 TEST

TEST

SCHDULE 생성

```
BEGIN
DBMS_SCHEDULER.CREATE_SCHEDULE(
schedule_name => 'test_schedule',
start_date => SYSTIMESTAMP,
end_date => SYSTIMESTAMP + 30,
repeat_interval => 'freq=minutely; bysecond=0',
comments => '1 minute ');
END;
/
```

start_date는 **SYSTIMESTAMP** 로 바로 시작 하게 되며
end_date 가 **SYSTIMESTAMP +30** 임으로 30일 뒤 종료 되는 스케줄을 생성

PROGRAM 생성

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
program_name => 'schedule_test',
program_action => '/oracle/testbackup/test.sh',
program_type => 'EXECUTABLE',
enabled => TRUE,
comments => 'schedule_test');
END;
/
```

test.sh 를 실행시키는 EXCUTABLE 형태의 Program 을 생성

7. COMPONENT DROP TEST

SCHDULE 생성

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name => 'schedule_1',
program_name => 'schedule_test',
schedule_name => 'test_schedule',
enabled => TRUE,
comments => 'scheduler_drop_test');

DBMS_SCHEDULER.CREATE_JOB (
job_name => 'schedule_2',
program_name => 'schedule_test',
schedule_name => 'test_schedule',
enabled => TRUE,
comments => 'scheduler_drop_test');

DBMS_SCHEDULER.CREATE_JOB (
job_name => 'schedule_3',
program_name => 'schedule_test',
schedule_name => 'test_schedule',
enabled => TRUE,
comments => 'scheduler_drop_test');

DBMS_SCHEDULER.CREATE_JOB (
job_name => 'schedule_4',
program_name => 'schedule_test',
schedule_name => 'test_schedule',
enabled => TRUE,
comments => 'scheduler_drop_test');
END;
/
```

생성한 스케줄인 test_schedule 과
프로그램인 schedule_test 를 이용한
JOB 4개를 생성

7. COMPONENT DROP TEST

JOB 상태 확인

```
set linesize 600
col job_name format a30

select job_name,operation,status
from DBA_SCHEDULER_JOB_LOG
where job_name='SCHEDULE_1'
order by log_date desc;

select job_name,operation,status
from DBA_SCHEDULER_JOB_LOG
where job_name='SCHEDULE_2'
order by log_date desc;

select job_name,operation,status
from DBA_SCHEDULER_JOB_LOG
where job_name='SCHEDULE_3'
order by log_date desc;

select job_name,operation,status
from DBA_SCHEDULER_JOB_LOG
where job_name='SCHEDULE_4'
order by log_date desc;
```

생성된 JOB 이 정상적으로 동작하고 있음을 확인 합니다

```
SQL> select job_name,operation,status
  2  from DBA_SCHEDULER_JOB_LOG
  3  where job_name='SCHEDULE_1'
  4  order by log_date desc
  5  /

JOB_NAME                       OPERATION                      STATUS
------------------------------ ------------------------------ ----------
SCHEDULE_1                     RUN                            SUCCEEDED
```

```
SQL> select job_name,operation,status
  2  from DBA_SCHEDULER_JOB_LOG
  3  where job_name='SCHEDULE_2'
  4  order by log_date desc
  5  /

JOB_NAME                       OPERATION                      STATUS
------------------------------ ------------------------------ ----------
SCHEDULE_2                     RUN                            SUCCEEDED
```

```
SQL> select job_name,operation,status
  2  from DBA_SCHEDULER_JOB_LOG
  3  where job_name='SCHEDULE_3'
  4  order by log_date desc
  5  /

JOB_NAME                       OPERATION                      STATUS
------------------------------ ------------------------------ ----------
SCHEDULE_3                     RUN                            SUCCEEDED
```

```
SQL> select job_name,operation,status
  2  from DBA_SCHEDULER_JOB_LOG
  3  where job_name='SCHEDULE_4'
  4  order by log_date desc
  5  /

JOB_NAME                       OPERATION                      STATUS
------------------------------ ------------------------------ ----------
SCHEDULE_4                     RUN                            SUCCEEDED
```

7. COMPONENT DROP TEST

1) SCHEDULE 삭제

SCHDULE 삭제

```
exec dbms_scheduler.drop_schedule('test_schedule', force=>true);
```

사용중인 SCHEDULE 을
FORCE => TRUE 로 삭제

아래와 같이 JOB이 **비활성화** 로
변경 되는 것을 확인 할 수 있습니다

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_1'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_1                     FALSE
```

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_3'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_3                     FALSE
```

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_2'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_2                     FALSE
```

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_4'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_4                     FALSE
```

7. COMPONENT DROP TEST

2) PROGRAM 삭제

PROGRAM 삭제

```
exec dbms_scheduler.drop_schedule('test_schedule', force=>true);
```

전과 동일하게 구성 후

사용중인PROGRAM 을
FORCE => TRUE 로 삭제 하게 되면

SCHEDULE 과 마찬가지로 바로
JOB 이 **비활성화** 됩니다

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_1'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_1                     FALSE
```

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_3'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_3                     FALSE
```

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_2'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_2                     FALSE
```

```
SQL> select job_name , enabled from dba_scheduler_jobs
  2  where job_name='SCHEDULE_4'
  3  /

JOB_NAME                       ENABL
------------------------------ -----
SCHEDULE_4                     FALSE
```

4. CLASS

1. JOB CLASS - Resource

2. JOB CLASS - Service

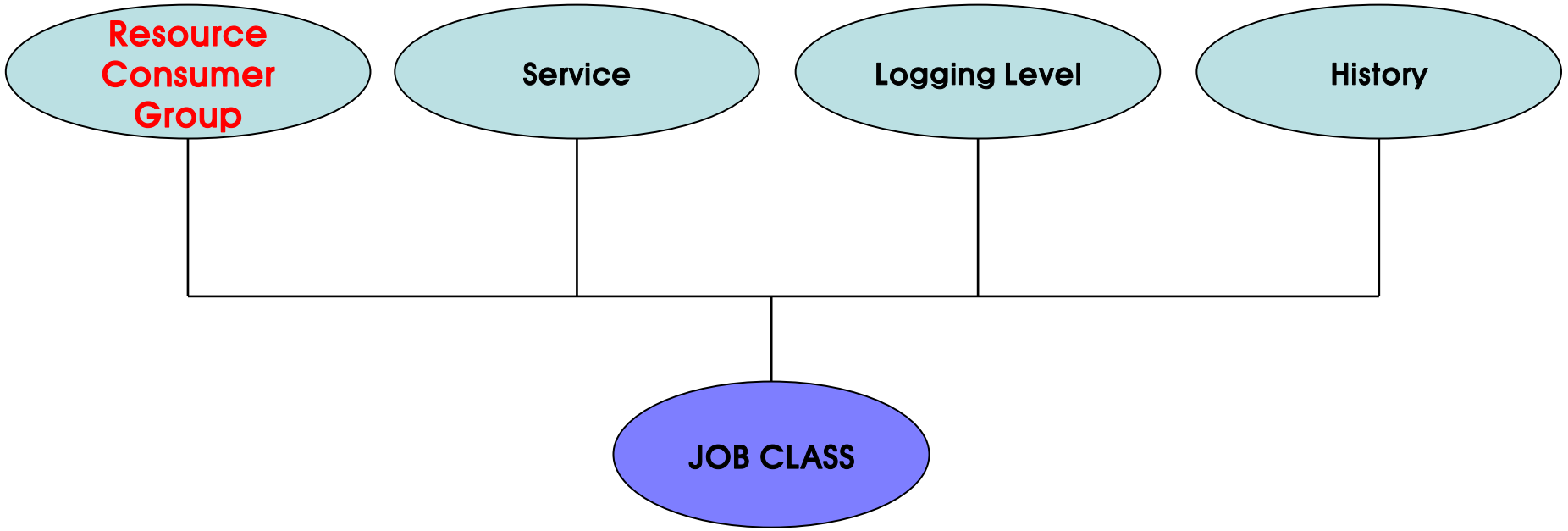
3. JOB CLASS - LOG

1. JOB CLASS - Resource

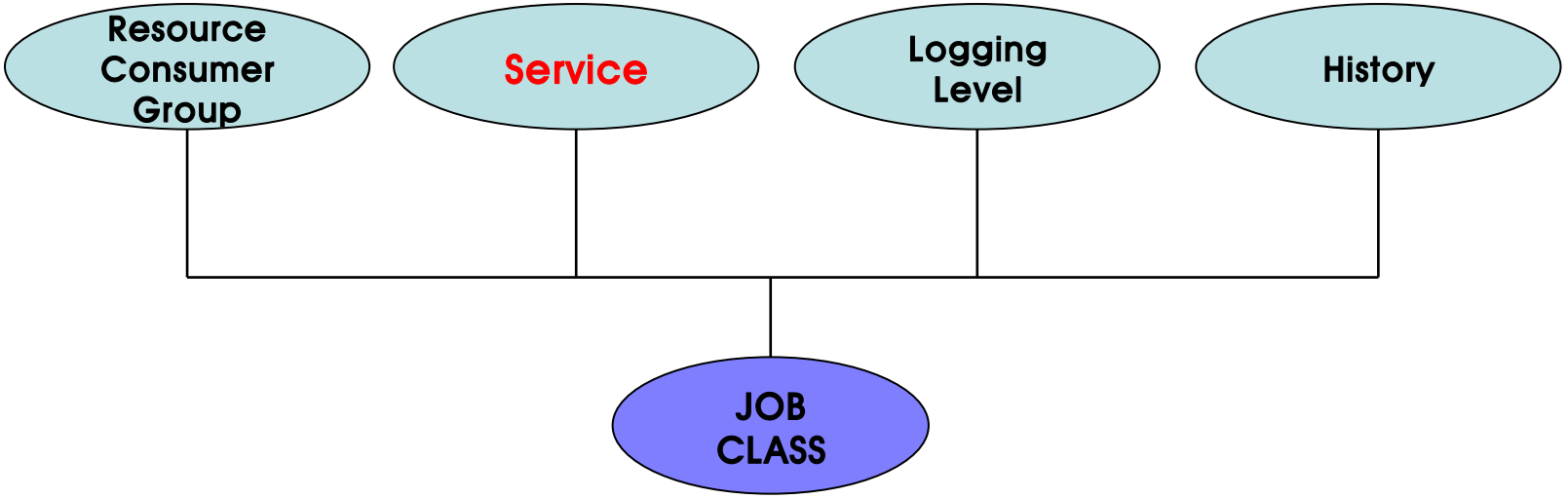
JOB을 생성한 후 명시적으로 Resource 할당을 할 필요가 있을 때, 일일이 모든 JOB에 대해서 설정해야 한다면 상당히 비효율적일 것입니다.

이를 위해서 scheduler는 job들간의 resource할당을 관리하게 위해서 job class개념을 제공합니다.

하나의 job class에 Resource Consumer Group을 매핑하여 이러한 관리를 가능하게 하는 원리인데, 이 resource_consumer_group 파라미터는 job class생성시 언급해주거나, 추후에 SET_ATTRIBUTE() 프로시저를 사용하여 지정할 수 있습니다.



2. JOB CLASS - Service



10g 부터는 Service라는 개념을 좀 더 다양하게 활용할 수 있는데, 그 중 활용방안 중 하나로서 특정 JOB을 '지정된 instance'에서만 수행되도록 지정할 수 있습니다.

즉, 대용량의 batch 가 job으로 등록되어 있는 경우, 한정된 instance에서 수행되게 함으로서 효율적인 resource 관리를 도모할 수 있습니다.

2. JOB CLASS - Service

SERVICE 예제 1/4

ORA101 이라는 TNS ALIAS를 통해 '1번 INSTANCE'에서 수행될 SERVICE인 FIRST, 그리고 ORA102 이라는 TNS ALIAS를 통해 '2번 INSTANCE'에서 수행될 SERVICE인 SECOND를 생성합니다.

(1) 생성

```
exec DBMS_SERVICE.CREATE_SERVICE('FIRST','ORA101') ;
exec DBMS_SERVICE.CREATE_SERVICE('SECOND','ORA102') ;
```

(2) 시작

```
exec DBMS_SERVICE.START_SERVICE('FIRST','ORA101') ;
exec DBMS_SERVICE.START_SERVICE('SECOND','ORA102') ;
```

(3) 확인

```
SQL> select NAME, NETWORK_NAME
 from dba_services where name in ('FIRST','SECOND');

NAME                                                             NETWORK_NAME
----------------------------------------
 FIRST                                                             ORA101
 SECOND                                                            ORA102
```

2. JOB CLASS - Service

SERVICE 예제 2/4

생성된 SERVICE를 참조할 JOB CLASS 생성합니다.

(1) 생성

```
BEGIN
DBMS_SCHEDULER.create_job_class(
job_class_name => 'JOB_CLASS_FIRST',
service => 'FIRST');
END;
/

BEGIN
DBMS_SCHEDULER.create_job_class(
job_class_name => 'JOB_CLASS_SECOND',
service => 'SECOND');
END;
/
```

2. JOB CLASS - Service

SERVICE 예제 3/4

생성된 CLASS 를 이용하여 JOB을 생성 및 활성화

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'job_class_test1',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats(''SCOTT''); END;',
start_date => SYSTIMESTAMP,
job_class => 'JOB_CLASS_FIRST'
repeat_interval => 'freq=hourly; byminute=0',
comments => 'Job class test1');
End;
/


exec dbms_shceduler.enable('job_class_test1');
```

2. JOB CLASS - Service

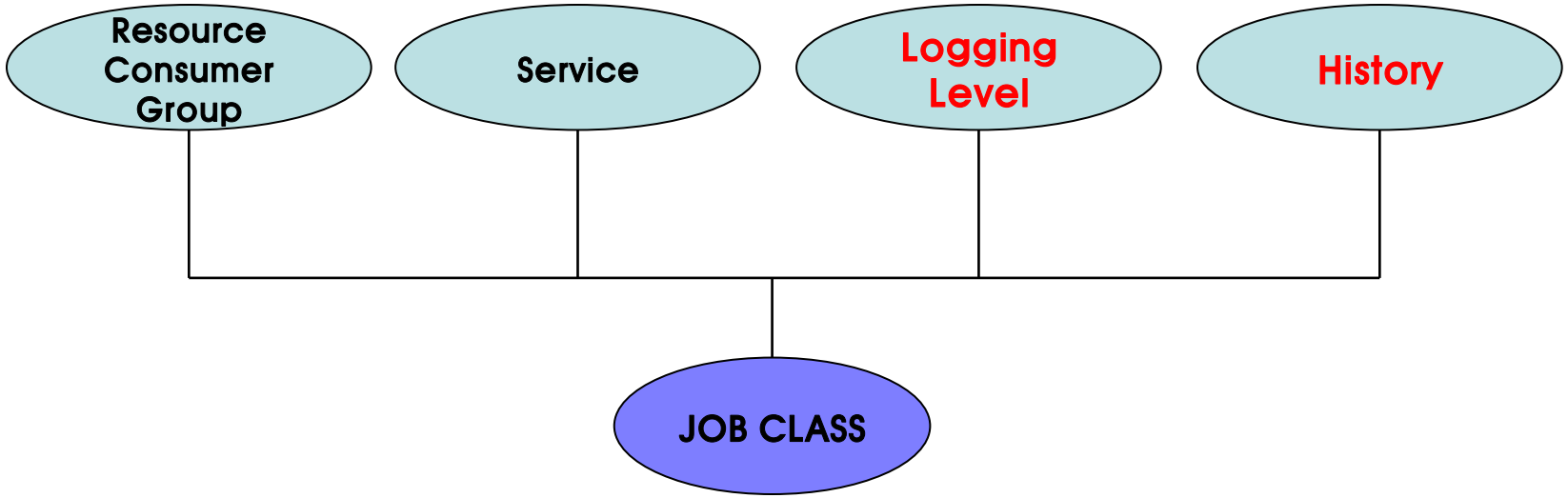
SERVICE 예제 4/4

생성된 CLASS 를 이용하여 JOB을 생성 및 활성화

```
BEGIN
DBMS_SCHEDULER.create_job (
job_name => 'job_class_test2',
job_type => 'PLSQL_BLOCK',
job_action => 'BEGIN DBMS_STATS.gather_schema_stats(''SCOTT''); END;',
start_date => SYSTIMESTAMP,
job_class => 'JOB_CLASS_SECOND'
repeat_interval => 'freq=hourly; byminute=0',
comments => 'Job class test2');
End;
/


exec dbms_shceduler.enable('job_class_test2');
```

3. JOB CLASS - LOG

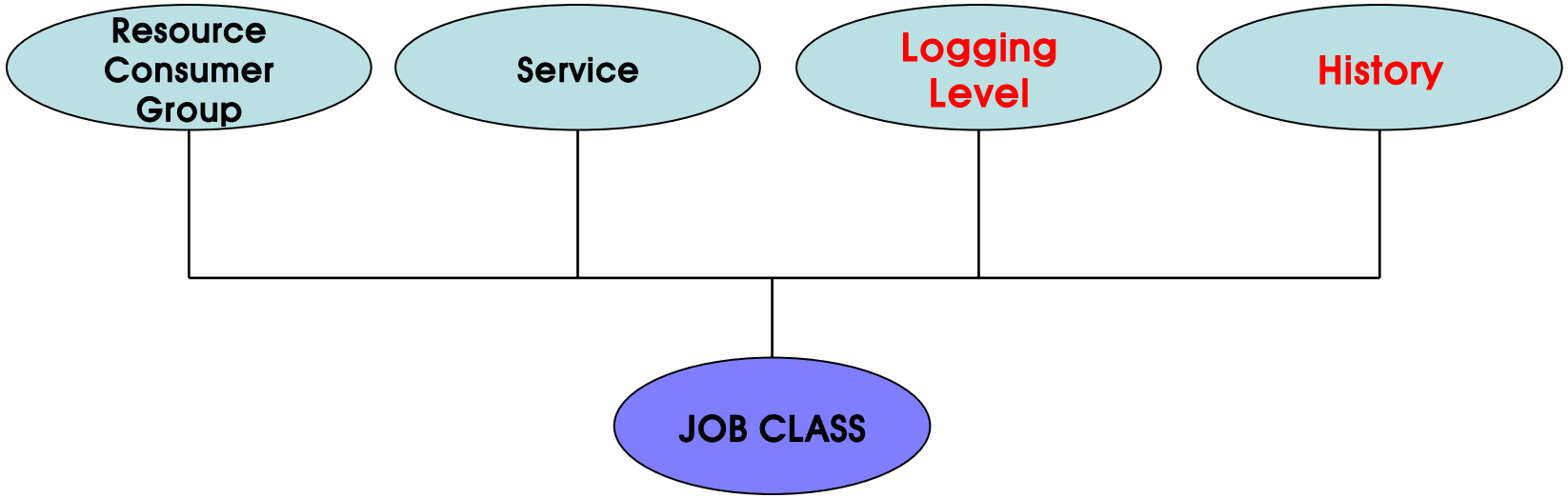


먼저 Logging Level은 3가지 type으로 설정할 수 있습니다

1. LOGGING_OFF : 이 옵션은 아무런 LOG도 남기지 않게 합니다
2. LOGGING_RUNS : JOB CLASS안에서 각기 수행되는 JOB에 대해서만 LOG를 기록합니다
3. LOGGING_FULL : JOB이 실행되는 순간뿐만 아니라, JOB에 관련된 모든 작업에 대해서 기록합니다
   예를 들어 JOB의 생성, 변경, 활성화 혹은 비활성화 등의 내용에 대해서도 기록합니다.

History 는 로그 보존 기간으로 써 LOG가 얼마나 오랫동안 기록 되어있을지를 결정 합니다

3. JOB CLASS - LOG

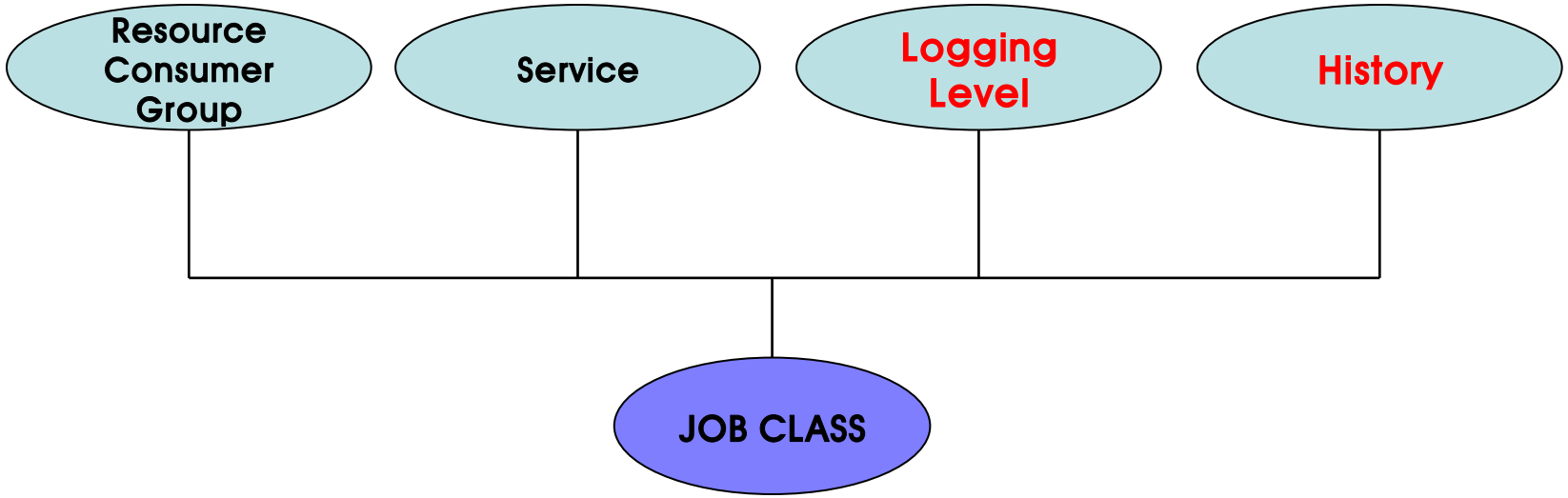


class 생성시

FULL 은 logging_level => DBMS_SCHEDULER.LOGGING_FULL
RUNS 은 logging_level => DBMS_SCHEDULER.LOGGING_RUNS
OFF 은 logging_level => DBMS_SCHEDULER.LOGGING_OFF

위와같은 형태로 기술하며 **class 를 이용하지 않고 job을 생성하게 되면 Default 로 loging_level 이 LOGGING_RUNS 로 job이 생성이 됩니다**

3. JOB CLASS - LOG



JOB에 대한 로그는 아래 dictionary 에서 확인할수 있습니다
ALL_SCHEDULER_JOB_LOG
DBA_SCHEDULER_JOB_LOG
USER_SCHEDULER_JOB_LOG

History(보존기간) 은 default 로 30 일이 지정 됩니다

5. CHAIN

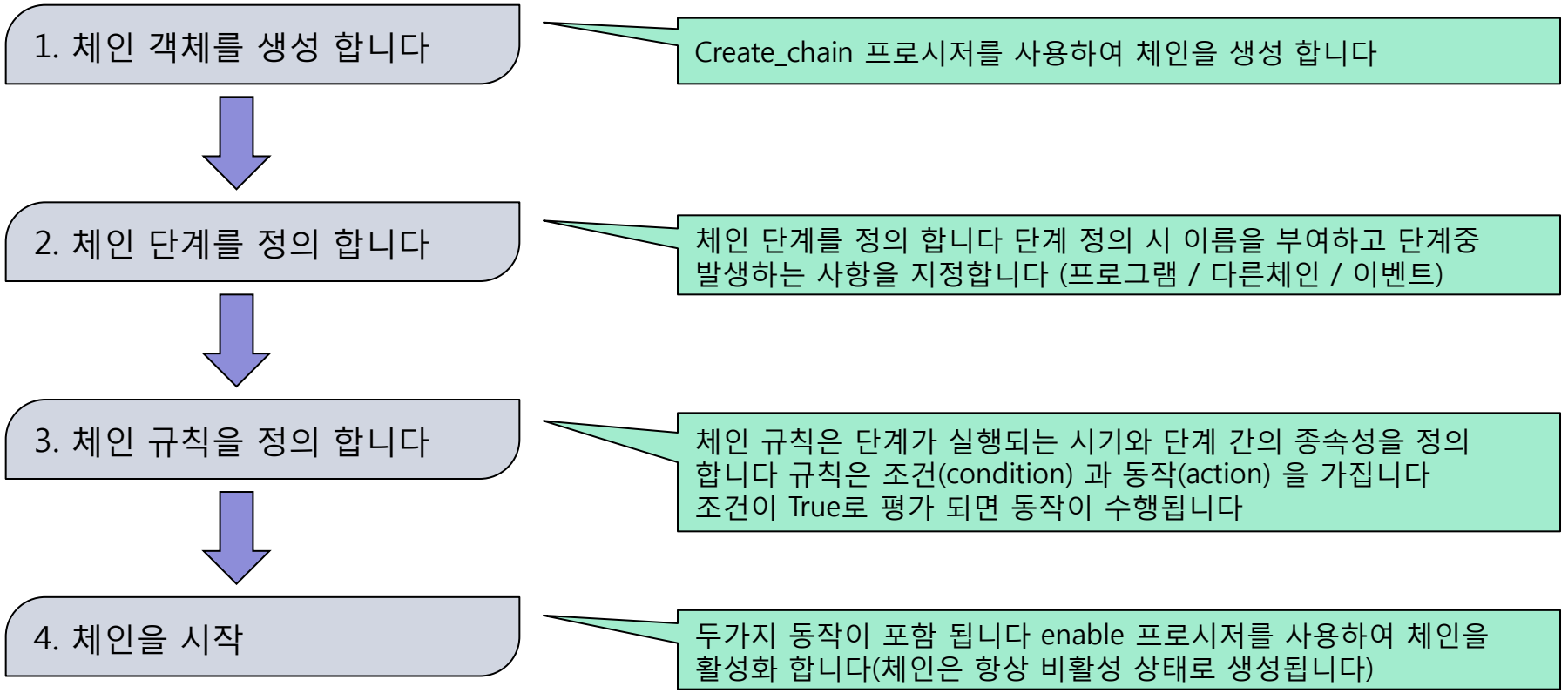
1. Introduction CHAIN
2. CHAIN 객체생성
3. CHAIN STEP
4. CHAIN RULE
5. CHAIN ENABLE
6. JOB(Using Chain)
7. CHAIN MORNITOR
8. CHAIN 객체 삭제

1. Introduction CHAIN

Chain 은 특정 작업을 위해 연결된 일련의 명명된 프로그램 입니다
이를 "종속적 스케줄링" 이라고 합니다 다음은 체인의 한 예입니다

" 프로그램 A를 실행 한 다음 프로그램 B를 실행하고 프로그램 A와 B가 성공적으로 완료 된 경우에만
프로그램 C를 실행한다"

상호 의존적인 프로그램 체인 내의 각 위치를 단계(step) 이라고 합니다



2. CHAIN 객체생성

chain

create_chain 프로시저를 사용하여 체인을 생성 합니다
chain 명은 **test_chain_1** 으로 체인을 생성 합니다

```
BEGIN
DBMS_SCHEDULER.create_chain (
chain_name => 'test_chain_1',
comments => 'A test chain');
END;
/
```

3. CHAIN STEP

Chain 단계 정의

생성한 chain 에 대해서 **단계** 정의 를 합니다.

step_name 은 유일한 이름으로 명명 합니다 program 에서는 생성되어 있는 program 이나 stored_procedure 를 이용합니다

****이 예제에서는 이미 생성되어 있는 program 을 이용하는 예제 입니다**

```
BEGIN

DBMS_SCHEDULER.define_chain_step (
 chain_name => 'test_chain_1', step_name => 'chain_step_1',
 program_name => 'test_program_1');

DBMS_SCHEDULER.define_chain_step (
chain_name => 'test_chain_1', step_name => 'chain_step_2',
program_name => 'test_program_2');

DBMS_SCHEDULER.define_chain_step (
 chain_name => 'test_chain_1', step_name => 'chain_step_3',
program_name => 'test_program_3');

END;
/
```

chain_name 은 전단계 에서 생성했던 chain 명 으로 기술되어 있습니다

생성되어 있는 Program 을 이용

4. CHAIN RULE

```
BEGIN
DBMS_SCHEDULER.define_chain_rule (
chain_name => 'test_chain_1', condition => 'TRUE',
action => 'START chain_step_1',
rule_name => 'rule_1',
comments => 'First link in the chain.');

 DBMS_SCHEDULER.define_chain_rule (
chain_name => 'test_chain_1',
condition => 'chain_step_1 completed',
action => 'START chain_step_2',
rule_name => 'rule_2',
comments => 'Second link in the chain.');

DBMS_SCHEDULER.define_chain_rule (
chain_name => 'test_chain_1',
condition => 'chain_step_2 completed',
action => 'START chain_step_3',
rule_name => 'rule_3',
comments => 'Third link in the chain.');

DBMS_SCHEDULER.define_chain_rule (
chain_name => 'test_chain_1',
condition => 'chain_step_3 completed',
action => 'END',
rule_name => 'rule_4',
comments => 'End of the chain.');
END;
/
```

Chain 규칙 정의

체인 단계를 지정한 후에 체인에 대한 규칙을 생성 할 수 있습니다

체인 규칙은 단계가 실행되는 **조건** 와 **단계 간의 종속성**을 정의 합니다

각 규칙은 조건과 동작을 가집니다
조건이 True로 평가 되면 동작이 수행 됩니다

첫 번째 체인 룰의 condition 값은 TRUE로 해야 하며 ,

마지막 체인룰의 action 은 END 이여야 합니다

그래야만 체인이 시작되며 종료가 됩니다

5. CHAIN ENABLE

Chain 활성화

체인 생성 및 수정을 완료한 후에 ENABLE 프로시저를 호출하여 체인을 활성화 합니다
CHAIN 은 생성시 Enable => true 와 같은 Enable 파라미터가 존재 하지 않음으로

생성후 반드시 chain을 사용하기 전에 활성화 해줘야 사용할수 있습니다

```
BEGIN
DBMS_SCHEDULER.enable ('test_chain_1');
END;
/
```

6. JOB(Using Chain)

JOB생성

체인을 시작하기 위한 JOB을 생성 합니다
job_type 은 CHAIN으로 설정하고 **job_action 은 위에서 생성한 chain 명을 설정**합니다
나머지 인자는 다른 유형의 작업을 구성할 때와 마찬가지로 구성됩니다.

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name => 'test_chain_1_job',
job_type => 'CHAIN',
job_action => 'test_chain_1',
repeat_interval => 'freq=minutely; bysecond=0',
start_date => SYSTIMESTAMP,
end_date => SYSTIMESTAMP + (1/48);
END;
/
```

JOB_ENABLE

```
EXEC DBMS_SCHEDULER.enable('test_chain_1_job');
```

7. CHAIN MORNITOR

생성된 체인 조회

```
SELECT owner, chain_name, rule_set_owner,
          rule_set_name, number_of_rules, number_of_steps,
          enabled, comments
FROM dba_scheduler_chains;
```

정의된 단계 조회

```
SELECT owner, chain_name, step_name,
       program_owner, program_name,
       step_type
FROM dba_scheduler_chain_steps
ORDER BY owner, chain_name, step_name;
```

정의된 규칙 조회

```
SELECT owner, chain_name, rule_owner,
 rule_name, condition, action,
comments
FROM dba_scheduler_chain_rules
ORDER BY owner, chain_name, rule_owner, rule_name;
```

[DBA | ALL | USER]_SCHEDULER_CHAINS
[DBA | ALL | USER]_SCHEDULER_CHAIN_RULES
[DBA | ALL | USER]_SCHEDULER_CHAIN_STEPS
[DBA | ALL | USER]_SCHEDULER_RUNNING_CHAINS

8. CHAIN 객체 삭제

STEP 삭제

DBMS_SCHEDULER.DROP_CHAIN_STEP(chain_name => 'test_chain_1',- step_name=>'chain_step_1', force => true);

** chain_name 에는 해당 chain 이름을 , step_name 에는 삭제하려는 step 이름을 기술합니다
해당 step 이 사용중이라면 force => true 를 사용하여 삭제 합니다

RULE 삭제

DBMS_SCHEDULER.DROP_CHAIN_RULE(chain_name => 'test_chain_1',- rule_name=>'rule_2', force => true);

** chain_name 에는 해당 chain 이름을 , rule_name 에는 삭제하려는 rule 이름을 기술합니다
해당 rule 이 사용중이라면 force => true 를 사용하여 삭제 합니다

CHAIN 삭제

DBMS_SCHEDULER.DROP_CHAIN_RULE(chain_name => 'test_chain_1', force => true);

** chain_name 에는 삭제하려는 chain 이름을 기술 합니다, 사용중인 chain을 삭제 하려면
force => true 를 사용하여 삭제 합니다

**** CHAIN 객체 삭제시 CHAIN STEP 과 CHAIN RULE 도 같이 종속적으로 삭제 됩니다**

6. The backup by using scheduler

1. Scenario

2. Creation Chain & Program

3. Define Chain Step

4. Define Chain Rule

5. Creation Job & Execute Chain

6. NOTICE

1. Scenario

DBMS_SCHEDULER 를 이용하여 백업 정책을 수립하려 합니다

Chain 진행 순서

1. chain_step_1이 매일 새벽 2시 10분 START한다.
2. chain_step_1 성공하면 10분 후에
3. chain_step_2이 실행된다.
4. Chain_step_2이 completed 되면 종료한다.

백업단위	Job_name	비 교
RMAN 백업	RMAN_BACKUP_ZERO	매주 월요일 am 2:00
DataPump 백업	PUMP_BK_DB_FULL	DB FULL EXPORT

2. Creation Chain & Program

체인 생성

```
BEGIN
DBMS_SCHEDULER.create_chain (
chain_name => 'sche_backup',
comments => 'backup_using_to_scheduler');
END;
/
```

체인 객체를 생성 합니다
chain 이름은 **sche_backup** 으로 합니다

chaning 할 program생성

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
program_name => 'Backup_Incr_zero_Chain1',
program_action => '/oracle/testbackup/rman/backup_zero.sh',
program_type => 'EXECUTABLE',
enabled => TRUE,
comments => 'full backup on monday am two');
END;
/
```

chain에 사용할 program을 생성합니다
program이름은
Backup_inre_zero_Chain1
입니다

3. Define Chain Step

chaning 할 program생성

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM (
program_name => 'Backup_full_export',
program_action => '/oracle/testbackup/exp/exp_db_full.sh',
program_type => 'EXECUTABLE',
enabled => TRUE,
comments => 'incremental backup monday on week am two');
END;
/
```

chain에 사용할 program을 생성합니다
program이름은
Backup_full_export 입니다

STEP 정의

```
BEGIN
DBMS_SCHEDULER.define_chain_step (
chain_name => 'sche_backup',
step_name => 'chain_step_1',
program_name => 'Backup_Incr_zero_Chain1');

DBMS_SCHEDULER.define_chain_step (
chain_name => 'sche_backup',
step_name => 'chain_step_2',
program_name => 'Backup_full_export');
END;
/
```

STEP을 정의 합니다
step 이름은 **chain_step_1** 이며
step에서 사용하는 program이름은
Backup_incr_zero_Chain1입니다

STEP을 정의 합니다
step 이름은 **chain_step_12**이며
step에서 사용하는 program이름은
Backup_full_export입니다

4. Define Chain Rule

RULE 정의

```
BEGIN
DBMS_SCHEDULER.define_chain_rule (
chain_name => 'sche_backup',
condition => 'TRUE',
action => 'START chain_step_1',
rule_name => 'chain_rule_1',
comments => '1st step in this chain on monday');

DBMS_SCHEDULER.define_chain_rule (
chain_name => 'sche_backup',
condition => 'chain_step_1 SUCCEEDED',
action => 'AFTER 00:10:00 START chain_step_2',
rule_name => 'chain_rule_2',
comments => '2st step in this chain');

DBMS_SCHEDULER.define_chain_rule (
chain_name => 'sche_backup',
Condition=>'chain_step_2 completed'
rule_name => 'end_rule',
action=>'end');
END;
/
```

chain 규칙 정의

각 단계에 대해서 RULE을 정의 합니다

CONDITION => TRUE 로써 chain_rule_1 을 실행하며
action으로는 **chain_step_1** 을 시작합니다

CONDITION => chain_step_1 succeeded로써
chain_step_1 이 성공했다면 이라는 조건이 이며
해당 조건이 만족하게 되면
action으로는 **AFTER 00:10:00 START chain_step2**
로 지정되어 있으며 10분 후 chain_step2 을 시작하라는
의미 입니다

CONDITION => chain_step_2 completed 로써
chain_step_2 가 완료 되었다면 이라는 조건이며
action으로는 **end** 로 체인을 종료 한다 는 의미 입니다

5. Creation Job & Execute Chain

CHAIN이 정의된 JOB생성

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name => 'sche_backup_job',
job_type => 'CHAIN',
job_action => 'sche_backup',
start_date => SYSTIMESTAMP ,
repeat_interval => 'freq=daily;byhour=2;byminute=10');
END;
/
```

job_type 을 CHAIN 으로 하는 JOB을 생성 합니다

job_action 은 생성해둔 chain이름인 sche_backup 으로 기술하고

JOB 이 수행될 시간과 주기를 지정합니다

CHAIN ENABLE

```
EXEC DBMS_SCHEDULER.ENABLE('sche_backup');
```

JOB ENABLE

```
EXEC DBMS_SCHEDULER.ENABLE('sche_backup_job');
```

마지막으로 생성된
CHAIN 과 JOB 을 활성화 시킴으로써

JOB이 수행될것입니다

6. NOTICE

CHAIN 사용시 주의 사항

CHAIN 을 생성하고 STEP 과 RULE 을 생성하고 ENABLE 하기 전에 문제가 발생하여도 활성화가 됩니다

해당 CHAIN으로 JOB을 생성하고 활성화 역시 됩니다 그말은 문법만 맞다면 생성 및 활성화가 된다는 것입니다

그렇기에 해당 SCHEDULER 에 문제가 있다고 알수있는건 JOB이 실행되면서 남는 로그를 확인해봐야 정상적인 SCHEDULER인지 CHAIN상에 문제가 없었는지를 알 수 있습니다

TEST

CHAIN 객체 생성

```
BEGIN
DBMS_SCHEDULER.create_chain (
chain_name => 'test_chain_1',
comments => 'A test chain');
END;
/
```

6. NOTICE

CHAIN STEP 정의

```
BEGIN

DBMS_SCHEDULER.define_chain_step (
 chain_name => 'test_chain_1', step_name => 'chain_step_1',
 program_name => 'Backup_Incr_zero_Chain1');

DBMS_SCHEDULER.define_chain_step (
chain_name => 'test_chain_1', step_name => 'chain_step_2',
program_name => 'Backup_full_export');

END;
/
```

미리 생성해놓은
PROGRAM 을 이용

6. NOTICE

CHAIN RULE 정의

```
BEGIN
DBMS_SCHEDULER.define_chain_rule (
chain_name =>'test_chain_1', condition => 'TRUE',
action => 'START chain_step_1',
rule_name => 'rule_1',
comments => 'First link in the chain');


DBMS_SCHEDULER.define_chain_rule (
chain_name => 'test_chain_1',
condition => 'chain_step_1 completed',
action =>'START chain_step_2',
rule_name => 'rule_2',
comments => 'Second link in the chain.');

DBMS_SCHEDULER.define_chain_rule (
chain_name => 'test_chain_1',
condition => 'chain_step_2 completed',
action => 'end',
rule_name => 'rule_3',
comments => 'Third link in the chain.');

end;
/
```

6. NOTICE

CHAIN 을 활성화 하기전에 chain_step_2 의 STEP 을 삭제후 활성화를 시켜 봅니다

CHAIN STEP 삭제

```
exec dbms_scheduler.drop_chain_step(chain_name => 'test_chain_1', - step_name =>
'chain_step_2', force => true);
```

STEP 삭제

CHAIN 활성화

```
exec dbms_scheduler.enable('test_chain_1');
```

CHAIN 을 활성화 하게 되면 정상적으로 활성화가 된 것을 아래와 같이 확인 할수 있습니다

```
SQL> select chain_name,enabled from dba_scheduler_chains
  2  where chain_name ='TEST_CHAIN_1';

CHAIN_NAME                     ENABL
------------------------------ -----
TEST_CHAIN_1                   TRUE
```

6. NOTICE

JOB을 생성하고 활성화 하게 되면 문제가 발생

JOB 생성

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
job_name => 'test_chain_1_job',
job_type => 'CHAIN',
job_action => 'test_chain_1',
repeat_interval => 'freq=minutely; bysecond=0',
start_date => SYSTIMESTAMP,
end_date => SYSTIMESTAMP + (1/48));
END;
/
```

JOB 의 활성화

```
EXEC DBMS_SCHEDULER.enable('test_chain_1_job');
```

JOB의 생성과 활성화 역시 정상적으로 됩니다
하지만 로그를 보게 되면 결과는 실패 라는 것을 확인할 수 있습니다

6. NOTICE

JOB을 생성하고 활성화 하게 되면 문제가 발생

LOG 조회

```
set linesize 600
col job_name format a30

select job_name,operation,status
from DBA_SCHEDULER_JOB_LOG
where job_name='TEST_CHAIN_1_JOB'
order by log_date desc
/
```

```
JOB_NAME                         OPERATION                        STATUS
-------------------------------- -------------------------------- --------
TEST_CHAIN_1_JOB                 CHAIN_RUN                        FAILED
TEST_CHAIN_1_JOB                 CHAIN_START                      RUNNING
TEST_CHAIN_1_JOB                 CHAIN_RUN                        FAILED
TEST_CHAIN_1_JOB                 CHAIN_START                      RUNNING
TEST_CHAIN_1_JOB                 CHAIN_RUN                        FAILED
TEST_CHAIN_1_JOB                 CHAIN_START                      RUNNING
TEST_CHAIN_1_JOB                 CHAIN_RUN                        FAILED
TEST_CHAIN_1_JOB                 CHAIN_START                      RUNNING
TEST_CHAIN_1_JOB                 CHAIN_RUN                        FAILED
TEST_CHAIN_1_JOB                 CHAIN_START                      RUNNING
```

CHAIN_RUN부분에서 FAILED 된 것을 확인할 수 있습니다
그렇기 때문에 CHAIN 을 이용한 SCHEDULER 를 생성시 각 단계마다 이상이 있는 여부를 체크해봐야 하고
문제 발생시 에는 **CHAIN의 각단계 및 JOB 등을 하나씩 모두 살펴 봐야 합니다**

Thanks