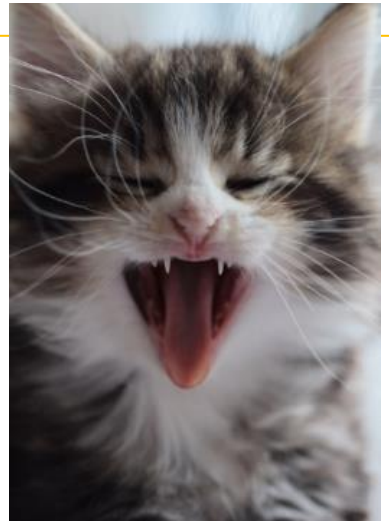


APEX connect

APEX Connect- Online Konferenz vom 04.Mai bis 05.Mai 2021

Job Steuerung in der Oracle Datenbank mit Oracle APEX

APEX 20.2 AUTOMATIONS FEATURE / DER ORACLE SCHEDULER IM EINSATZ

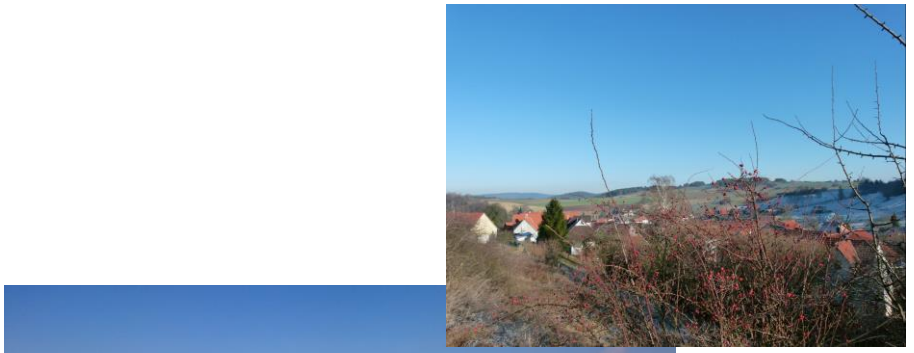
GPI Consult Gunther Pippèrr

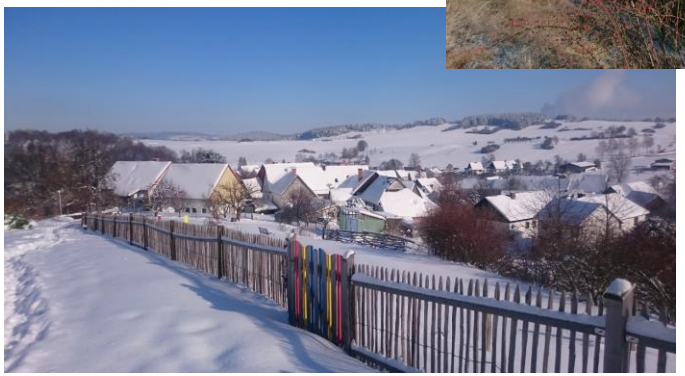
ORACLE ACE Associate

gunther@pipperr.de

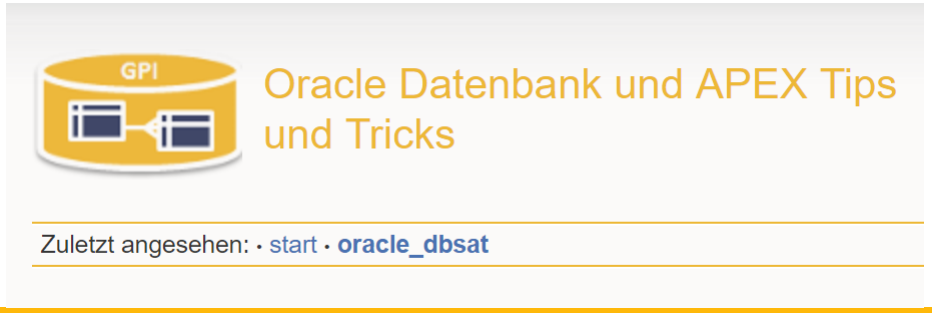
Mein Blog

https://www.pipperr.de/dokuwiki/





Bergweg 14 - 37216 Witzenhausen/Roßbach



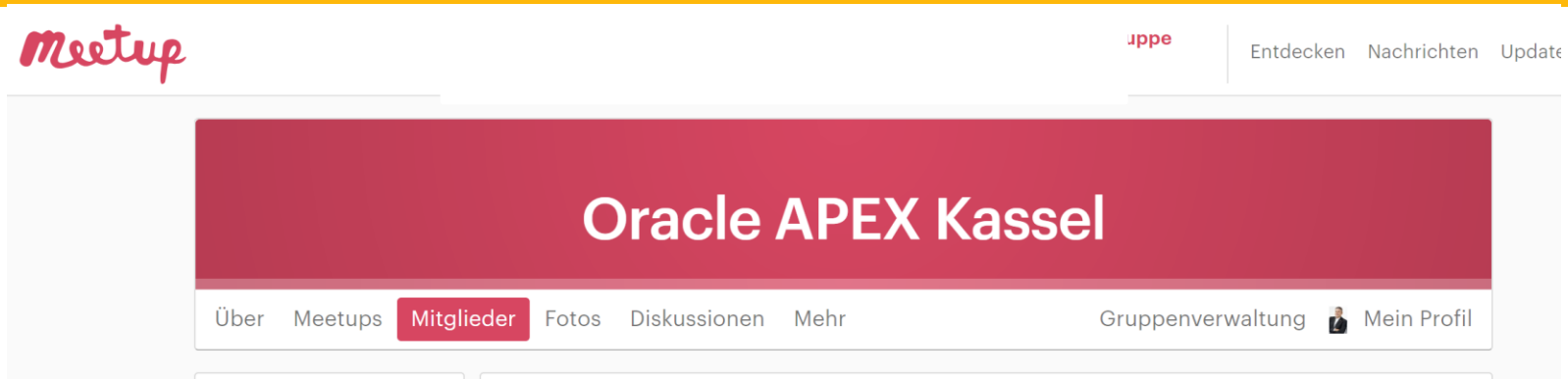
Freiberuflicher Oracle Datenbank Experte - Ich unterstütze Sie gerne in ihren Projekten.

APEX Meetup Gruppe Kassel-Göttingen

Raum für Veranstaltung in Kassel oder Göttingen gesucht, können Sie uns unterstützen?

Mitglieder gesucht!

https://www.meetup.com/de-DE/Oracle-APEX-Kassel/

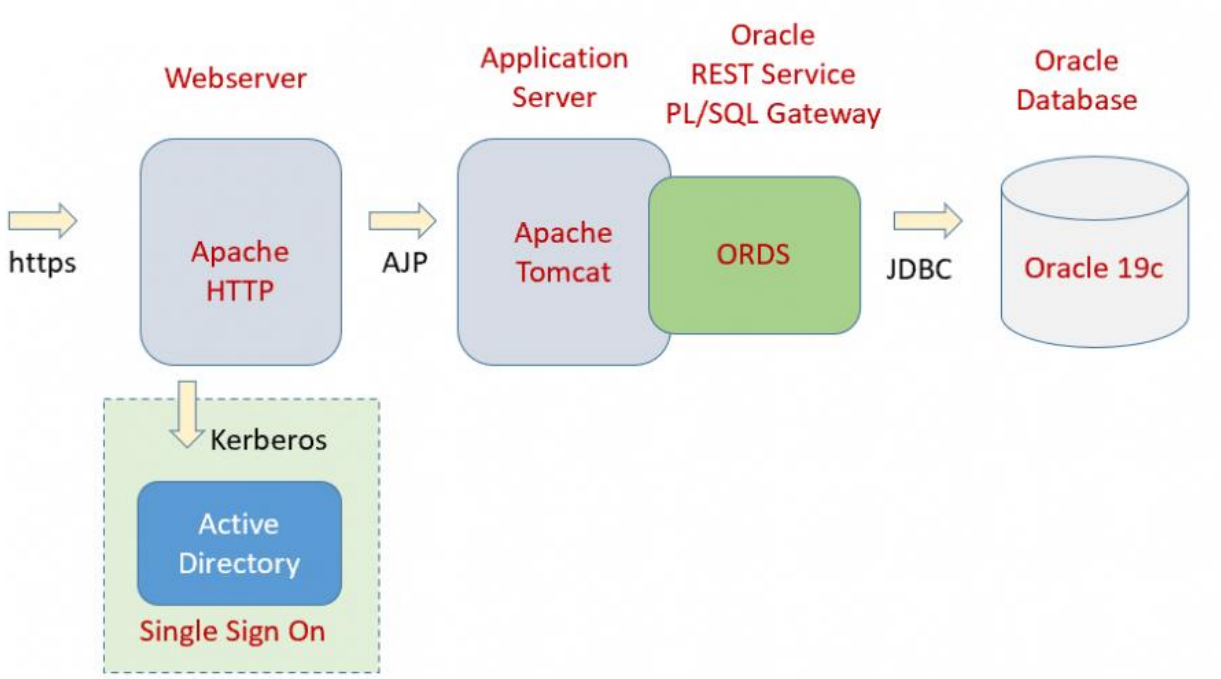


Agenda

1	**Die Ausgangslage / Intention / Szenarien**
2	APEX Automation
3	Job Steuerung Architektur
4	Das Feature Set des Oracle Schedulers
5	Fazit

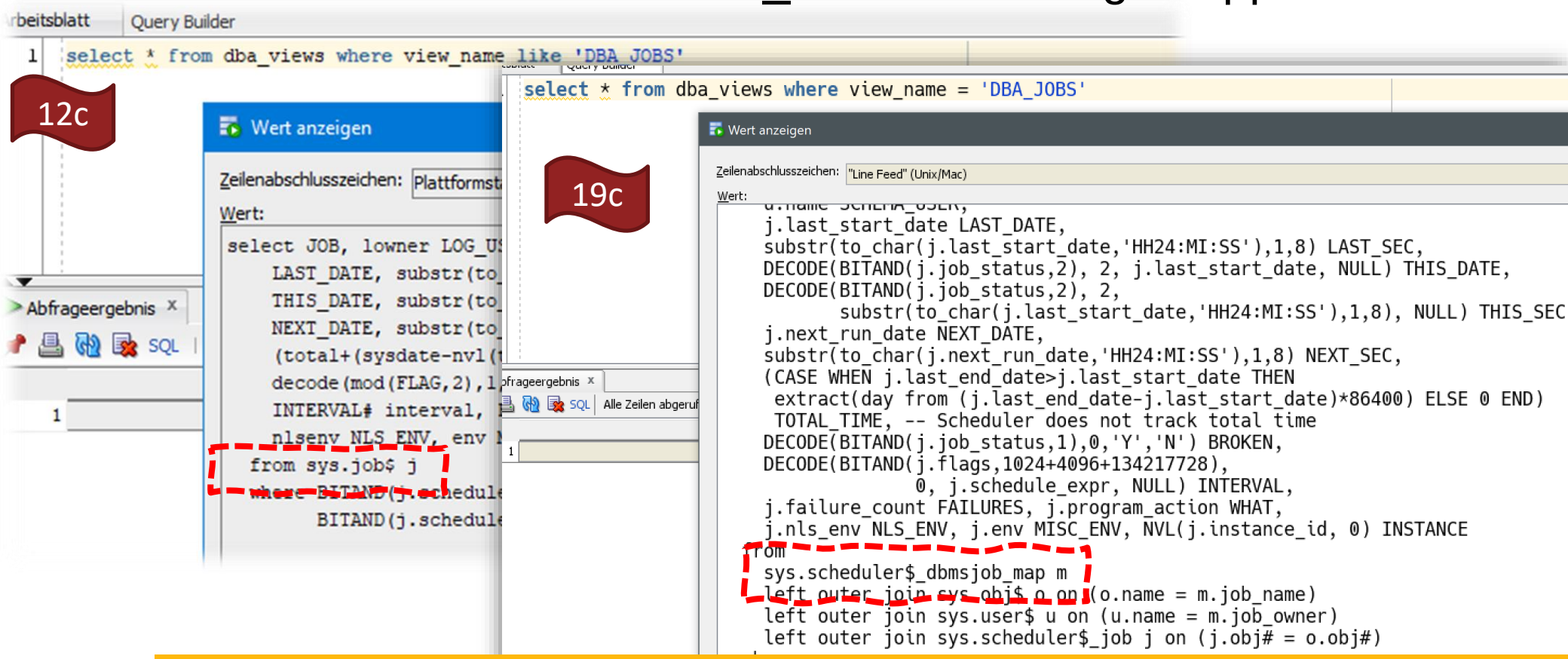
Das Demo Setup

- Siehe => https://www.pipperr.de/dokuwiki/doku.php?id=prog:oracle_apex_20_2_install_windows_19c_linux_8



Die Ausgangslage – 19c und DBMS_JOB (1)

- Ab Oracle 19c keine DBMS_JOBS mehr!
  - Alles wird intern auf DBMS_SCHEDULER gemappt



Mapping ALT auf NEU über => **sys.scheduler$_dbmsjob_map**

Die Ausgangslage – 19c und DBMS_JOB (2)

- DBMS_JOB kann noch verwendet werden, führt aber zu einem Scheduler Job mit allen damit verbunden Abhängigkeiten und Funktionen!

19c

```
1  BEGIN
2
3  DBMS_JOB.isubmit (
4      job       => 66
5      ,what     => 'begin null; /*doNix Job */ end;'
6      ,next_date => SYSDATE
7      ,interval  => 'SYSDATE + 1/24 /* 1 Hour */'
8  );
9  COMMIT;
10
11 END;
12 /
```

Abfrageergebnis | Skriptausgabe

Aufgabe abgeschlossen in 0,241 Sekunden

```
PL/SQL-Prozedur erfolgreich abgeschlossen.
```

```
1  select * from dba_scheduler_jobs where job_name like 'DBMS%'
```

Single Record View

OWNER	SYS
JOB_NAME	DBMS_JOB$_66
JOB_SUBNAME	
JOB_STYLE	REGULAR
JOB_CREATOR	SYS
CLIENT_ID	
GLOBAL_UID	
PROGRAM_OWNER	
PROGRAM_NAME	
JOB_TYPE	PLSQL_BLOCK
JOB_ACTION	begin null; /*doNix Job */ end;
NUMBER_OF_ARGUMENTS	0
SCHEDULE_OWNER	
SCHEDULE_NAME	
SCHEDULE_TYPE	PLSQL
START_DATE	13.04.21 12:31:59,000000000 EUROPE/BERLIN
REPEAT_INTERVAL	SYSDATE + 1/24 /* 1 Hour */
EVENT_QUEUE_OWNER	

Im Vergleich

DBMS_JOB	DBMS_SCHEDULER
▪ Kein automatisches Commit	▪ Automatisches Commit
▪ Einfacher in der Definition	▪ Komplexer in der Definition
▪ Für komplizierte “Next Job” Zeiten oft umständlich	▪ Log vergangener Job Läufe
▪ Nur Aufruf on PL/SQL	▪ Kalender für Schedules
	▪ Event Basierender Aufruf
	▪ OS Jobs / Remote Jobs und mehr

Lizenz?

- In der Datenbank
  - In allen DB Editionen enthalten
  - Oracle Scheduler ab Oracle 10g

- Oracle Remote Scheduler Agent
  - „Oracle Database installed on Remote Host“ => enthalten
  - Keine Oracle Installation vorhanden => **Unklar ….**

ETL Szenarien umsetzen

- Für welche Szenarien können wir die erweiterten Möglichkeiten von Oracle Scheduler Jobs und APEX Automation verwenden?

- **Ziel :**
  Alles über die Datenbank/APEX zentral steuern und konfigurieren

- Wie:
  - Dateien bei Bedarf verarbeiten und Einlesen
  - Auf einem Event in der Datenbank reagieren und Business Prozesse anstoßen
  - Auf Fehler reagieren und Job Abhängigkeiten beachten

ETL Szenario mit Scheduler Job Feature (1)

- Quelle liefert “gelegentlich” auf Server in die DMZ, diese müssen zuerst in Richtung Datenbank “versandt” werden
  - Wenn die Daten eintreffen müssen die Daten gleich verarbeitet werden

Filewatcher Funktionalität / External Job Fuktionalität

- Prüfe alle x Minuten ob die Datei da ist
- Falls ja, starte den Lade Job

ETL Szenario mit Scheduler Job Feature (2)

- Daten werden als CSV geliefert und müssen angepasst werden
  - Betriebssystem Programm muss aber die Daten zuvor formatieren

External Skript Funktionalität

Scheduler ruft im OS das Programm auf und formatiert die Daten

https://www.pipperr.de/dokuwiki/doku.php?id=dba:oracle_scheduler_12c_external_scripts

ETL Szenario mit Scheduler Job Feature (3)

- Daten müssen in die Datenbank eingelesen werden
  - Stage Tabelle muss gefüllt und Daten verteilt werden

Job Chain

War der Schritt zuvor erfolgreich, kann über “External Table” die Datei eingelesen werden

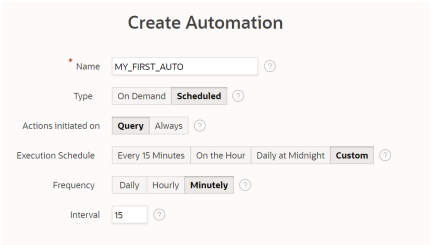
ETL Szenario mit Scheduler Job Feature (4)

- Nach dem Einlesen der Daten kann die Tagesverarbeitung starten
  - Liegen Daten vor, kann der Job A für die Tagesverarbeitung gestartet werden

Event Basierend

Liegen die Daten korrekt vor, startet der Tagesverarbeitungsjob

APEX Automations



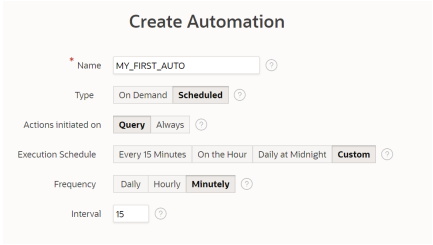
ETL Szenario mit Scheduler Job Feature (5)

- Fachabteilung soll per Mail informiert werden

Job Mail Feature

Information über die eigentliche Verarbeitung versenden

APEX Automations



Rechte - Wer darf Scheduler Jobs definieren?

- Welche Rollen sind notwendig

  – Für mein Schema einen Job anlegen
    - Create Job

  – Job für andere Schemas aufrufen
    - Create any Job
    - Execute any program
    - Execute any class

  – Scheduler verwalten
    - Manager Scheduler

Rechte für Externe Ressourcen

- Betriebssystem Zugriff auf Filesystem / Scripts
  - Wird über ein “Credential Object” vergeben

```
-- Credentials anlegen
BEGIN
  DBMS_CREDENTIAL.CREATE_CREDENTIAL('OS_USER_JOB_CONTROL','job_control','my_secret_pwd');
END;    -- DD abfragen
/
        SELECT  credential_name
               ,username
           FROM   user_credentials
        ORDER BY credential_name
        /
```

- Mail Versandt
  - ACL zum Zugriff auf dem Mail Server
  - Wallet bei Verschlüsselung

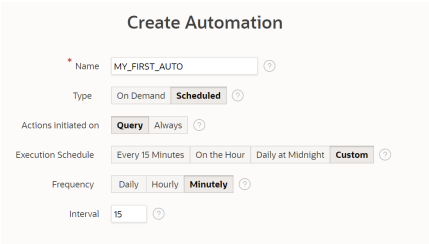
Siehe im Detail => https://www.pipperr.de/dokuwiki/doku.php?id=prog:plsql_send_mail_tls

APEX Automations Feature

APEX's eigener "einfacher" Job Scheduler

Declarative asynchronous processes

APEX Automations



https://www.pipperr.de/dokuwiki/doku.php?id=prog:apex_automations

APEX Automations Feature - Idee

- Logik im Hintergrund der Applikation bei Bedarf ausführen lassen

- Typische Business Cases
  - Workflows an starten, wenn bestimmte Daten in der Datenbank vorliegen
    - **Wie:** Sachbearbeiter A hat Vorgang bearbeitet und abgeschlossen, Kunde B erhält automatisch eine Mail, dass der Vorgang bearbeitet wurde
    - **Wie**: Bei Fehlern im Log Admin per Mail benachrichtigen
  - Regelmäßig im Hintergrund etwas starten
    - **Wie**: Logs aufräumen

APEX Automations Feature

- Mit APEX Automations werden PL/SQL Aktionen definiert

- Diese Aktion kann in Abhängigkeit von den Ergebnissen einer Query ausgeführt werden
  - Die Ergebnisse der Query können mit in der Aktion verarbeitet werden
  - SQL Query (Local Database or REST Enabled SQL)
    - Query kann auch mit einer „PL/SQL function returning SQL Query“ definiert werden
  - REST Data Source

- Eine vorab definierte Automation kann manuell über das APEX_AUTOMATION Package aufgerufen werden

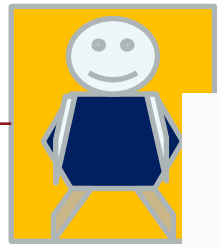
API zum Aufruf einer Automation

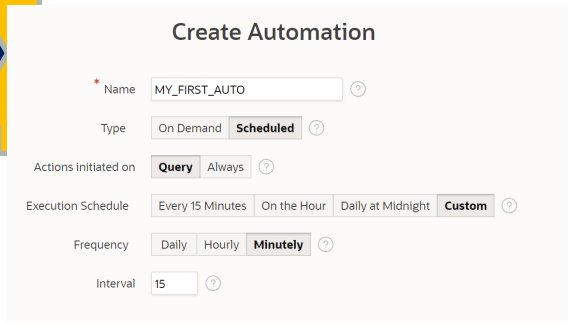
- DISABLE Procedure
- ENABLE Procedure
- EXECUTE Procedure
- EXECUTE for Query Context Procedure
- EXIT Procedure
- GET_LAST_RUN Return Function
- GET_LAST_RUN_TIMESTAMP Function
- LOG_ERROR Procedure
- LOG_INFO Procedure
- LOG_WARN Procedure
- RESCHEDULE Procedure
- SKIP_CURRENT_ROW Procedure

https://docs.oracle.com/en/database/oracle/application-express/20.2/aeapi/APEX_AUTOMATION.html

Übersicht APEX Automations

Meta Daten über Oberfläche eintragen





WWV_FLOW_AUTOMATIONS

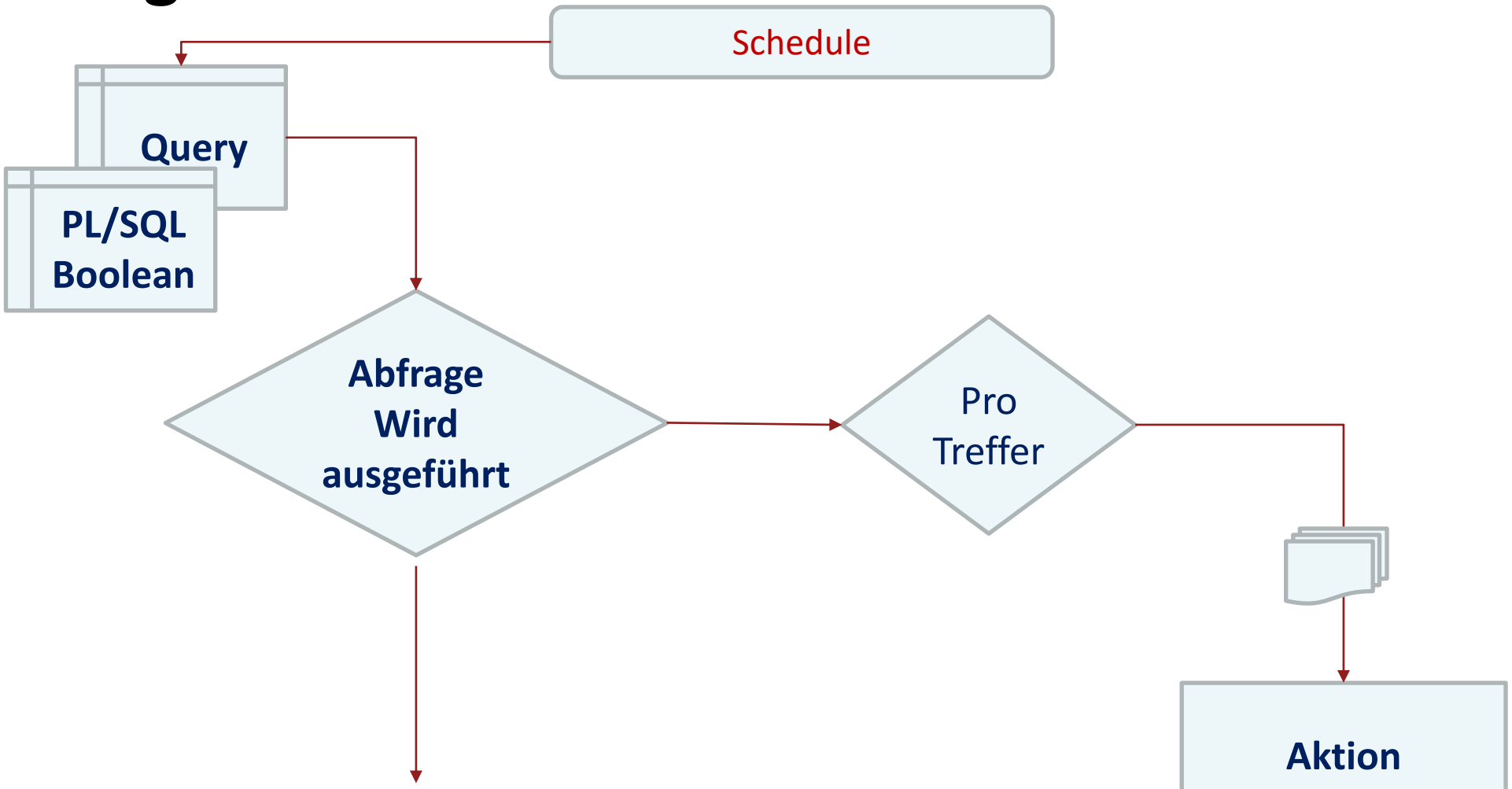
Prüft alle 2 Minuten auf zu startende Jobs

APEXxxxx DB Job
ORACLE_APEX_AUTOMATIONS

Startet über einen „One Time DB Scheduler Job“ im APP Schema
dieser führt die eigentliche Aufgabe im Scope des Schemas durch

DB JOB

Die eigentliche Job Action



Falls kein Treffer – je nach Einstellung

Die Query “parametrisieren”

- Datum des letzten Aufrufes verwenden mit **GET_LAST_RUN**

- Über die API das Datum des letzten Run der Automation in der Query verwenden
  - APEX_AUTOMATION.GET_LAST_RUN
    - return TIMESTAMP WITH TIME ZONE;

```
select *
  from {table}
 where created_at > apex_automation.get_last_run;
```

In der Aktion “reagieren”

- : Bind Variable aus der Event Query

- APEX_AUTOMATION.SKIP_CURRENT_ROW
  - Diese Ziele ignorieren
- APEX_AUTOMATION.EXIT
  - Aktion beenden und verlassen

- APEX_AUTOMATION.LOG_INFO/ERROR/WARN
  - In das APEX Automations Log schreiben

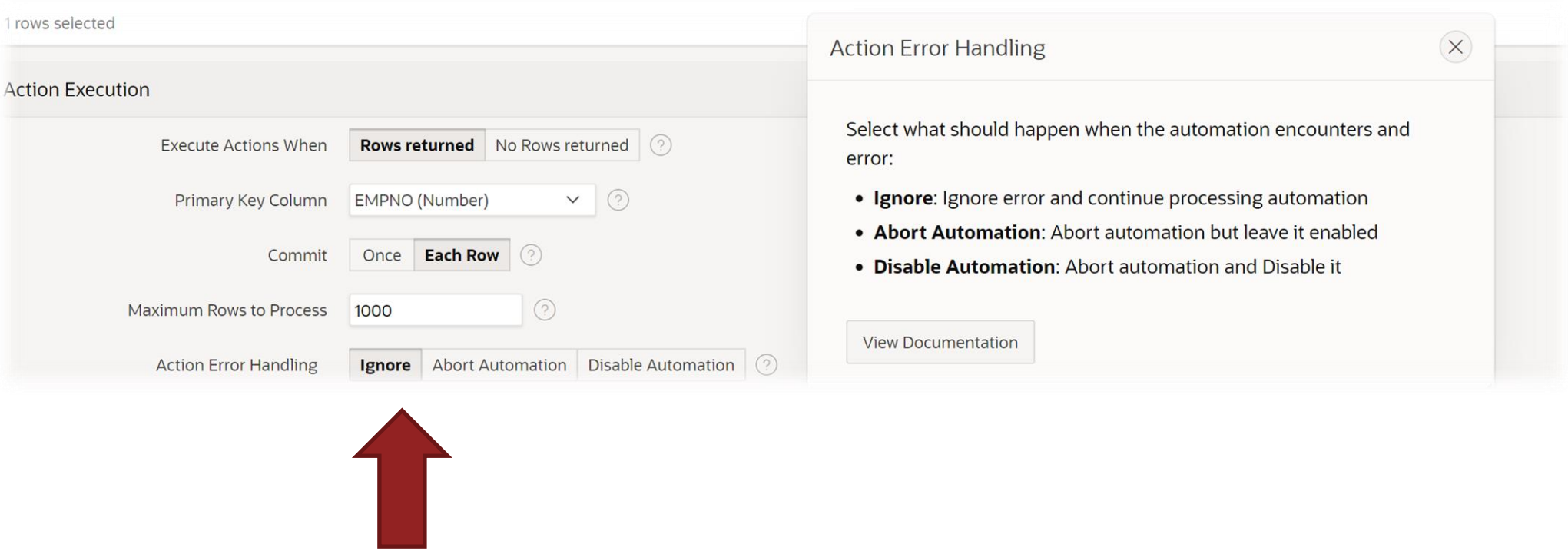
Aktion : Beispiel

```
BEGIN

  if :ENAME = 'Gunther' then
     apex_automation.skip_current_row( p_log_message => 'Gunther is found' );
     apex_automation.log_warn(         p_message     => 'Gunther is found' );
  end if;

  if :ENAME is null then
     apex_automation.exit(p_log_message => 'No Member Name found' );
  end if ;

END;
```

Auf die Reihenfolge der Aktionen achten! Exit verhindert evlt. die Ausführung der nächsten Aktion!

Fehler Behandlung

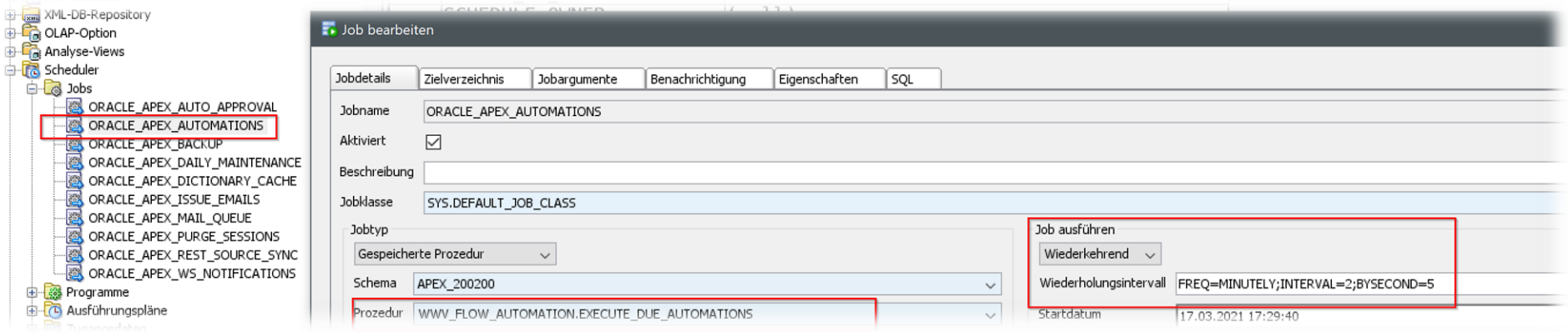
- Jede Aktion wird pro Trefferzeile aufgerufen
  - Wie nun bei Fehlern reagieren?



Was wird erzeugt? – Wer ruft das dann auf? (1)

- Kein DB Job pro Automation!

- Ein zentraler APEX Job, per Default alle 2 Minuten



Was wird erzeugt? – Wer ruft das dann auf? (2)

- Der Job “EXECUTE_DUE_AUTOMATIONS” prüft die “WWV_FLOW_AUTOMATIONS” View

- Falls der eigentliche Task laufen soll, wird ein einmaliger Job über den Scheduler angelegt

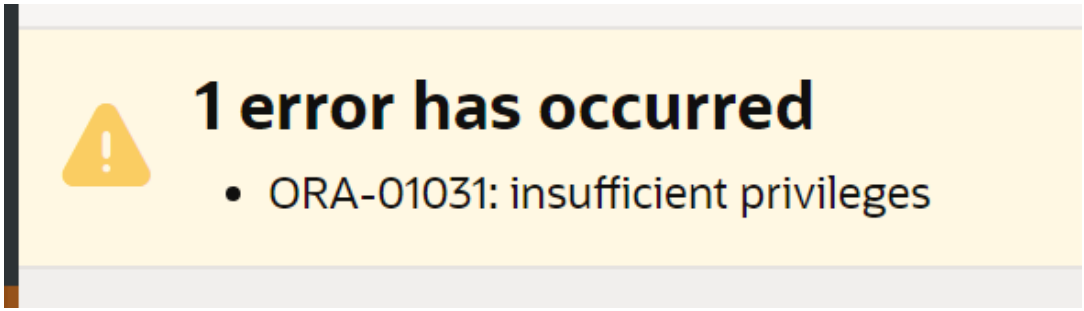
  – Mit der Methode “EXECUTE_AUTOMATION_ACTIONS”

  – Diese startet den eigentliche Task Code im richtigen Scope der jeweiligen APEX Applikation über WWV_FLOW_PLUGIN.EXECUTE_PROCESS

Vorraussetzung – Create Job Recht!

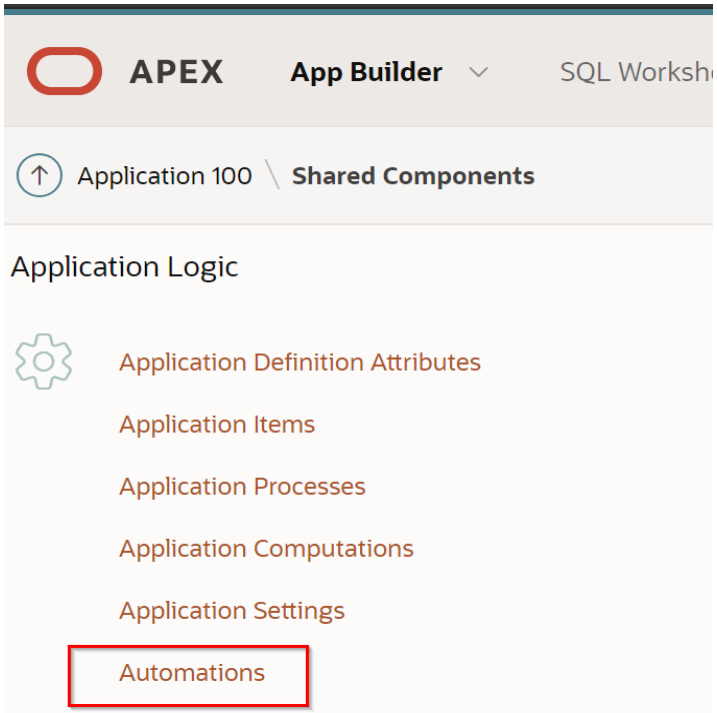
- **Das APP Schema muss das „create job“ Rechte besitzen**

- **Automation lässt sich anlegen, aber nicht aktivieren!**



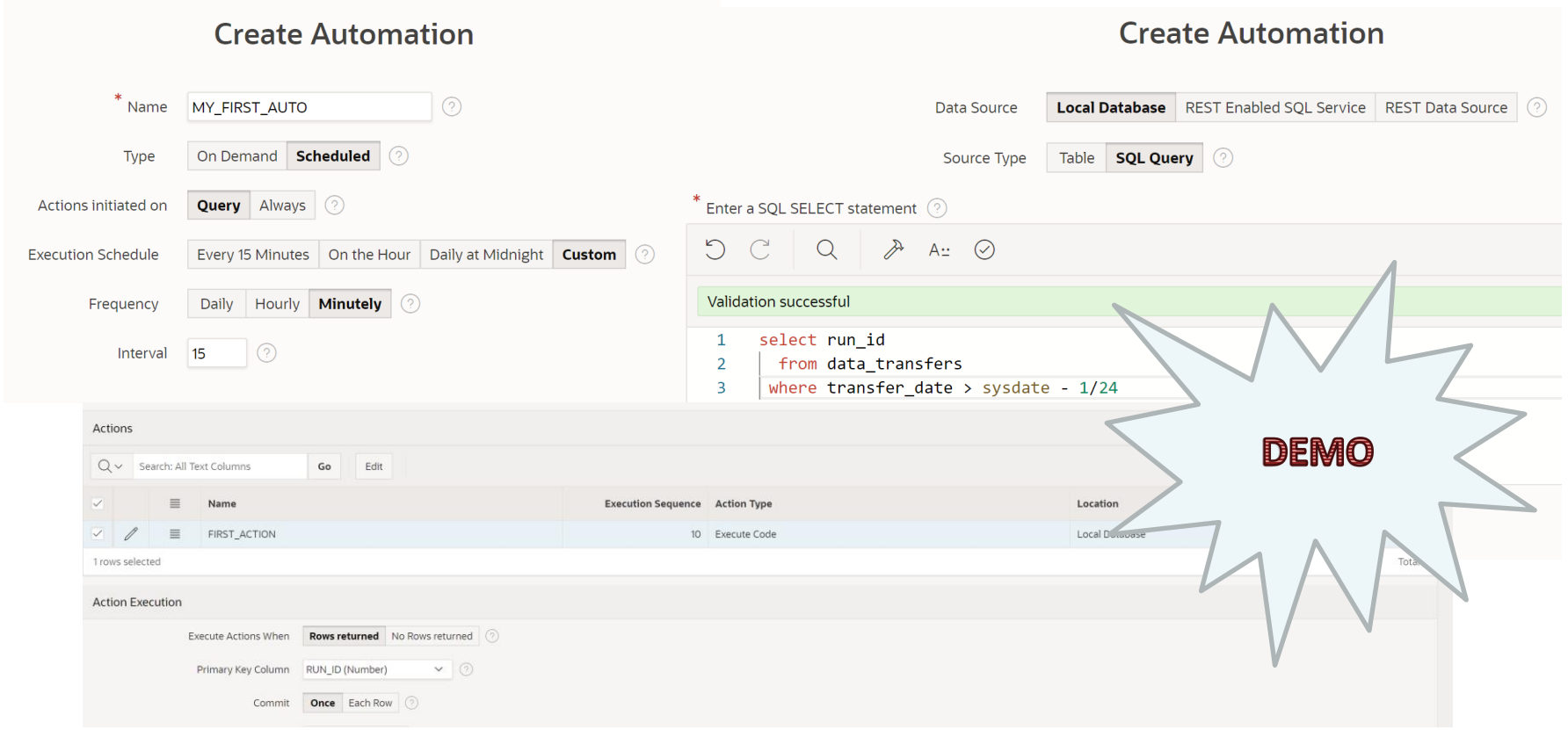
Anlegen (1)

- Unter “**Shared Components**”
  - ” Application Logic - **Automations**“



Anlegen (2)

- DEMO



Auf die **STATIC ID** achten, über diese wird der Automation Task später in der API angesprochen!

Automation Job per API aufrufen (1)

- Über den Scheduler sofort aufrufen lassen
  - Als APEX App Schema Owner

```
BEGIN
   apex_session.create_session(p_app_id        => 2021
                              , p_page_id      => 1
                              , p_username => 'ADMIN' );

   apex_automation.reschedule(
         p_static_id       => 'new-emp-message'
   );

END;
```

Automation Job per API aufrufen (2)

- Mit Filter im Session Scope aufrufen:

```
DECLARE
   l_filters apex_exec.t_filters;
BEGIN
   apex_session.create_session(p_app_id        => 2021
                              , p_page_id      => 1
                              , p_username => 'ADMIN' );
   apex_exec.add_filter(
      p_filters       => l_filters,
      p_column_name    => 'DEPTNO',
      p_filter_type    => apex_exec.c_filter_eq,
      p_value         => 10 );

   apex_automation.execute(
      p_static_id      => 'my_emp_table_automation',
      p_filters       => l_filters );
END;
```

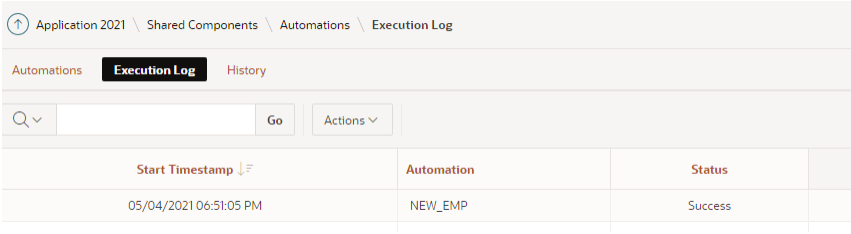
(Aus API Dokumentation)

Warum funktioniert es nicht?

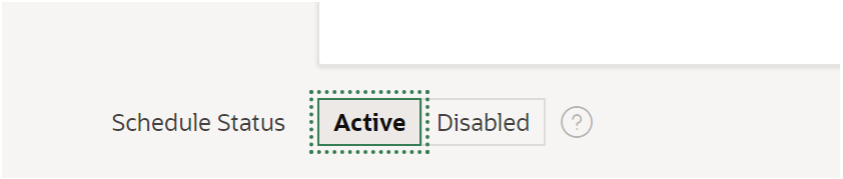
- Bei ersten Fehler wird der Job disabled!

Name	Type	Execute Actions When	Schedule Status	Last Run
NEW_EMP	Scheduled	Rows Returned	Disabled	05/04/2021 03:50:22 PM

- Wo finde ich den Fehler?
  - APEX_AUTOMATION_LOG
  - APEX_AUTOMATION_MSG_LOG



- Nach Behebung des Fehlers wieder aktiveren!



Meta Daten dazu abfragen

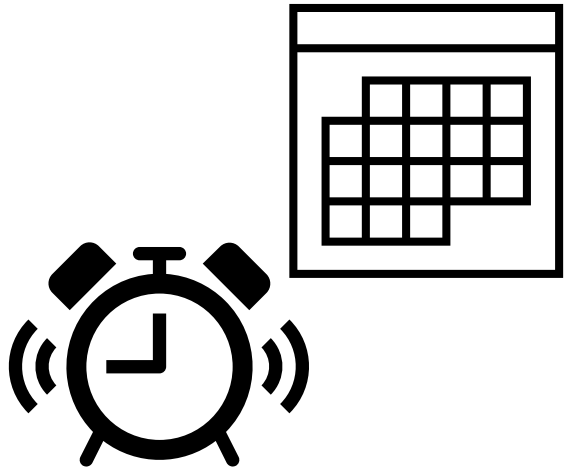
- Was wurde angelegt:
  - APEX_APPL_AUTOMATION_ACTIONS
  - APEX_APPL_AUTOMATIONS
- Monitoring
  - APEX_AUTOMATION_LOG
  - APEX_AUTOMATION_MSG_LOG

Vorteil

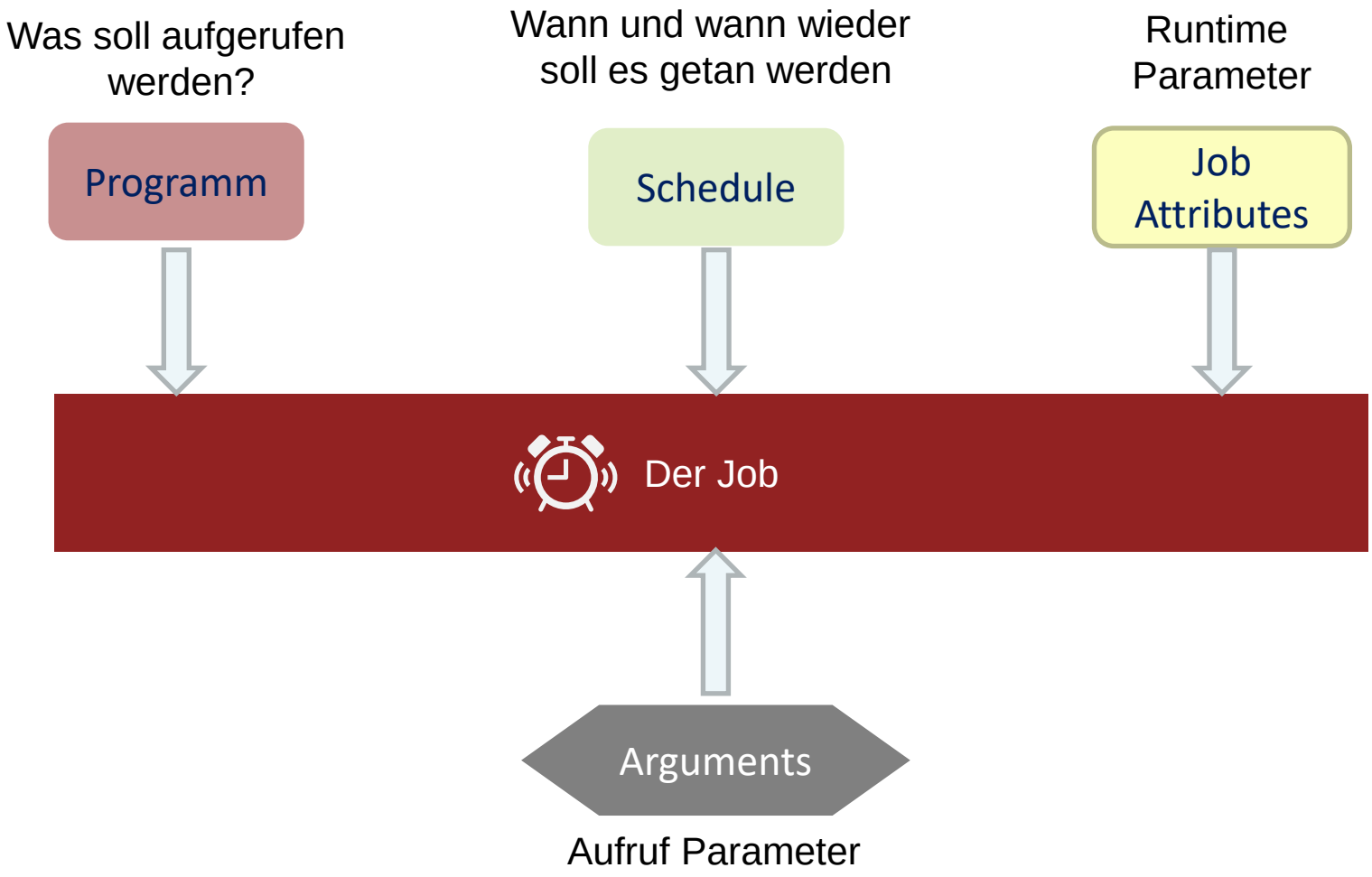
- In der APEX Welt integriert
- Es wird kein PL/SQL für das Anlegen von Jobs benötigt
- Nach einen Deployment sind die Jobs einfach “da” !
  - Jobs müssen nicht jedes Mal umständlich neu angelegt / angepasst werden

Datenbank Jobs im Detail

Architektur und Funktionalität von Datenbank Jobs

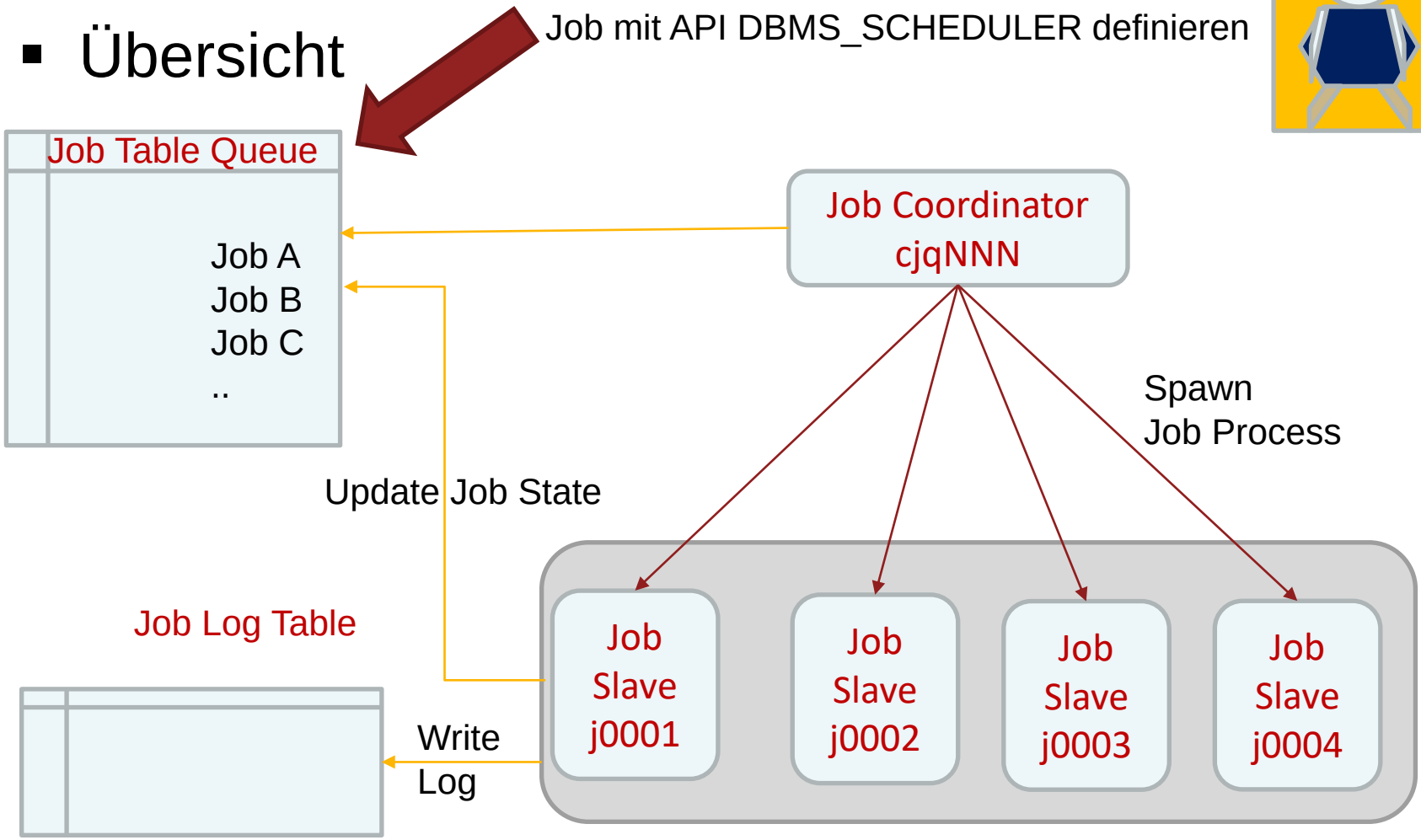


Aus was besteht ein Datenbank Job?



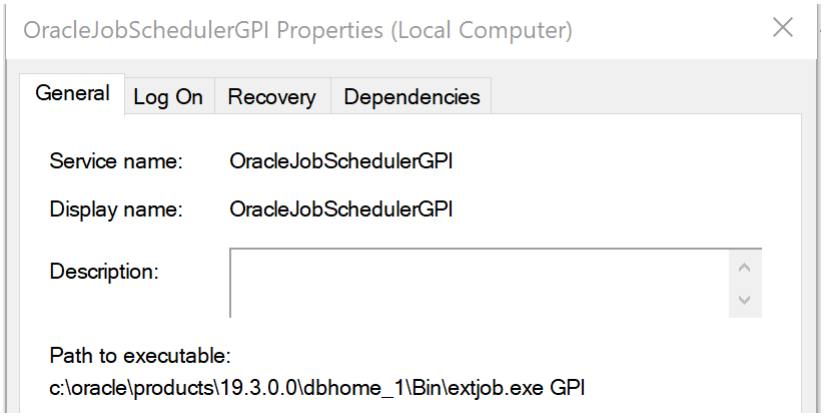
Architektur des Oracle Schedulers

- Übersicht



Oracle Scheduler Remote Agent

- Programm Aufruf kann auch auf einer anderen Maschine über den „Scheduler Remote Agent“ erfolgen
  - z.B. Windows:



  - Eine Oracle Datenbank muss auf dieser Maschine nicht installiert sein!

Siehe dazu:
https://oracle-base.com/articles/11g/scheduler-agent-installation-11gr2

Wichtige Begriffe zu DBMS Scheduler Jobs

- **Job** – Der eigentliche Job
  - Kann mit Parametern aufgerufen werden
- **Programm** – Abstrakte Definition / Logik für etwas zum Ausführen
  - Wie eine PL/SQL Procedure
  - Kann mit Parametern aufgerufen werden
- **Schedule** – Wann und wann wieder soll der Job starten

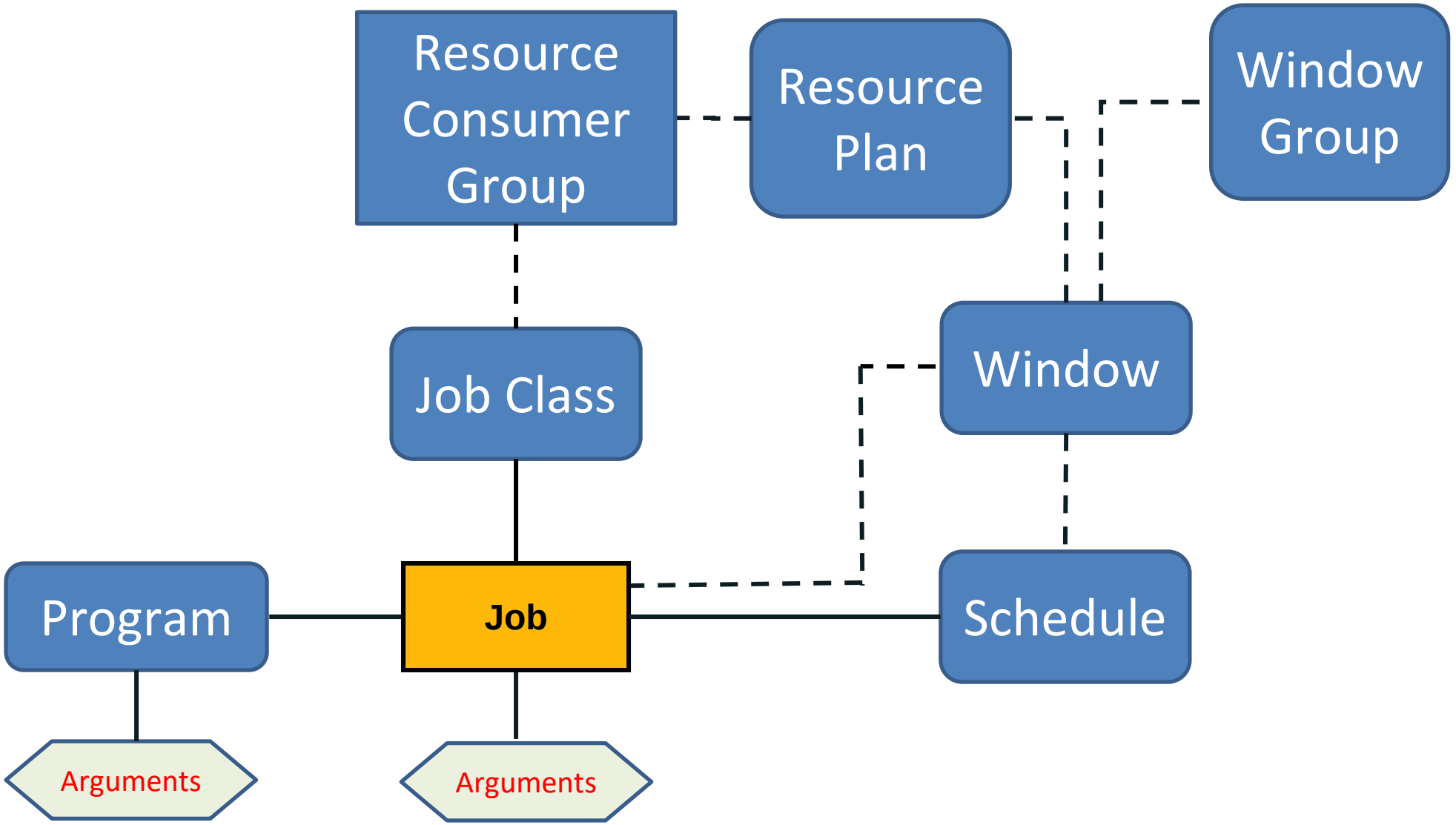
Wichtige Begriffe zu DBMS Scheduler Jobs

- **Job Class** – Mit welchen Eigenschaften soll der Job verknüpft werden
- **Window** – Zeitfenster in dem der Job laufen darf
- **Chain** – Kette von verknüpften Jobs
- **Resource Plan** – Was darf der Job an Ressourcen in der DB beanspruchen

Scheduler Job Typen

Normal	Lightweight Job	In Memory job
▪ Standard ▪ Full Feature Set ▪ job_style job attribute '**REGULAR**',	▪ Für “short-duration jobs”, die häufig starten ▪ Kein Schema Objekt ▪ Performanter und “platz sparender” in der Anlage ,da nicht so viele Daten im Data Dictionary hinterlegt werden müssen ▪ job_style job attribute '**LIGHTWEIGHT**‘.	▪ **In-memory runtime** Basiert auf den Ligthweight Jobs; Können ein „repeat interval“ haben und mehrfach starten ▪ Kein Logging da DEFAULT_IN_MEMORY_JOB_CLASS, auf logging level NONE. ▪ Existieren nur im Cache, nicht persistent. ▪ Nur auf der Instance auf der dieser Job angelegt wurde

Übersicht

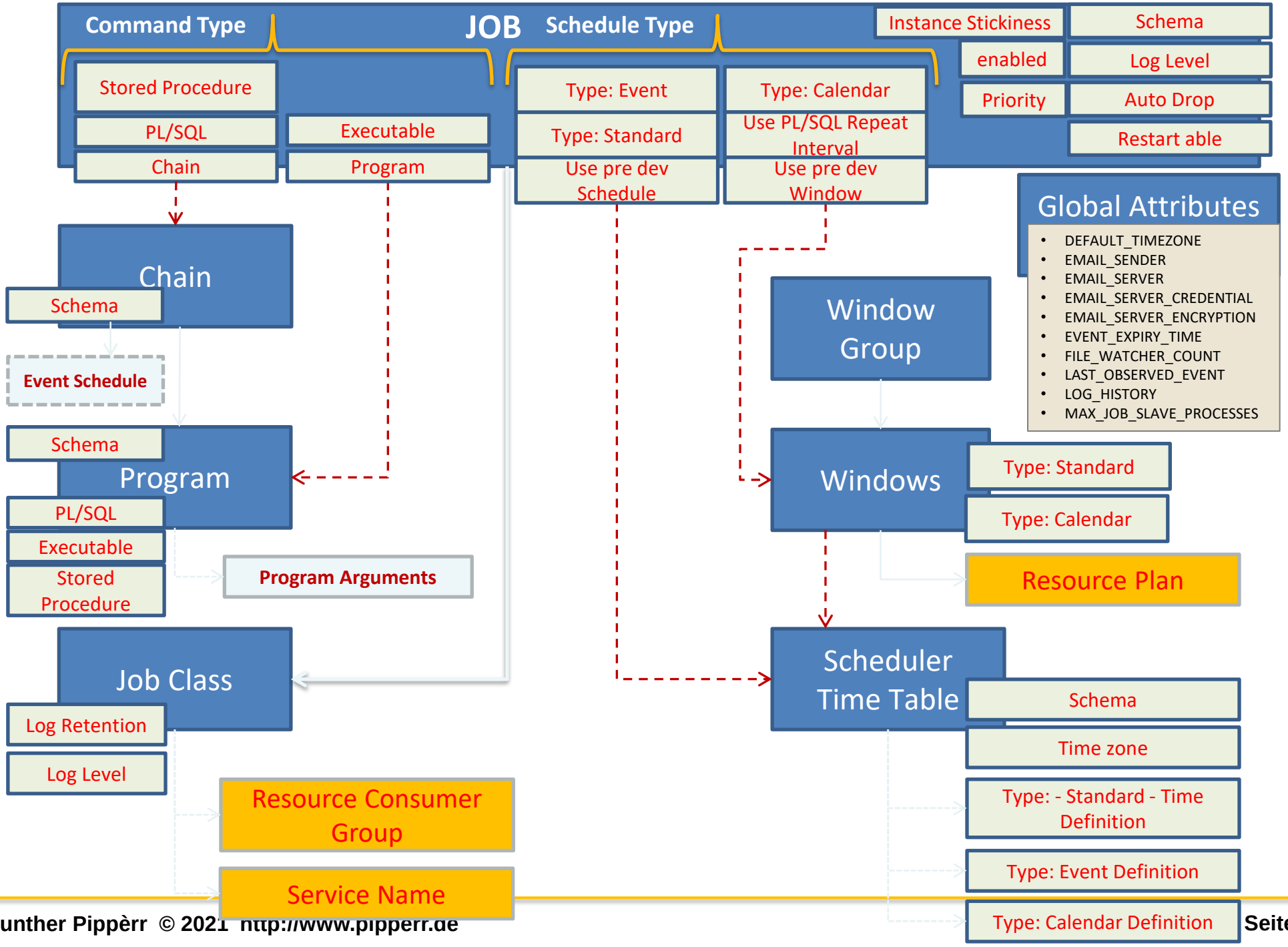


Aus https://download.oracle.com/owsf_2003/40615_goorah.ppt

Die API - DBMS_SCHEDULER (1)

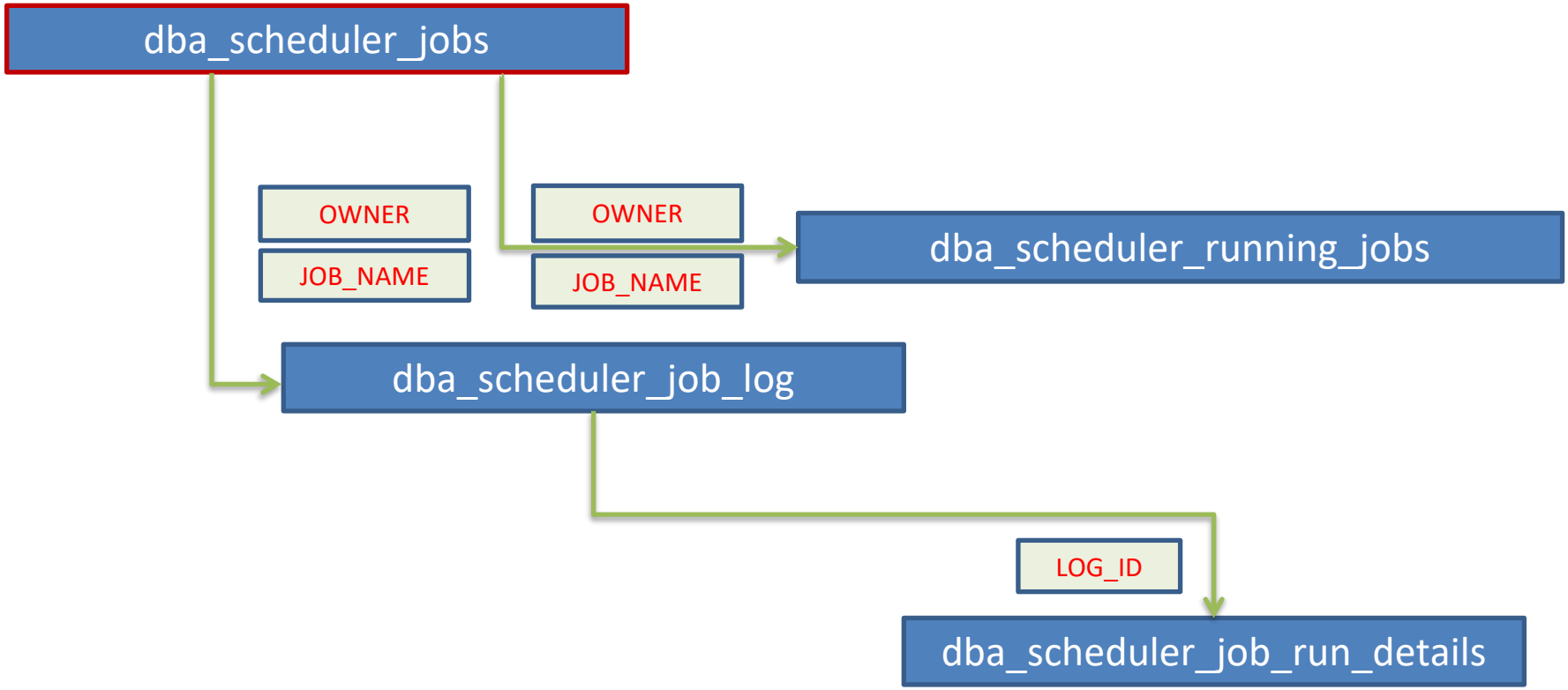
- Jobs erzeugen
  - Einfachster Aufruf

```
BEGIN
     DBMS_SCHEDULER.create_job (  job_name        => 'CLEAN_SQL_ERROR_LOG_TABLE‘
                            , job_type        => 'PLSQL_BLOCK‘
                            , job_action      => 'BEGIN system.deleteOraErrorTrigTab(15); END;‘
                            , start_date      => SYSTIMESTAMP
                            , repeat_interval => 'freq=daily; byhour=13; byminute=0‘
                            , end_date        => NULL
                            , enabled         => TRUE
                            , comments        => 'Job to clean all lean Error Log older then xx days‘);
END;
```

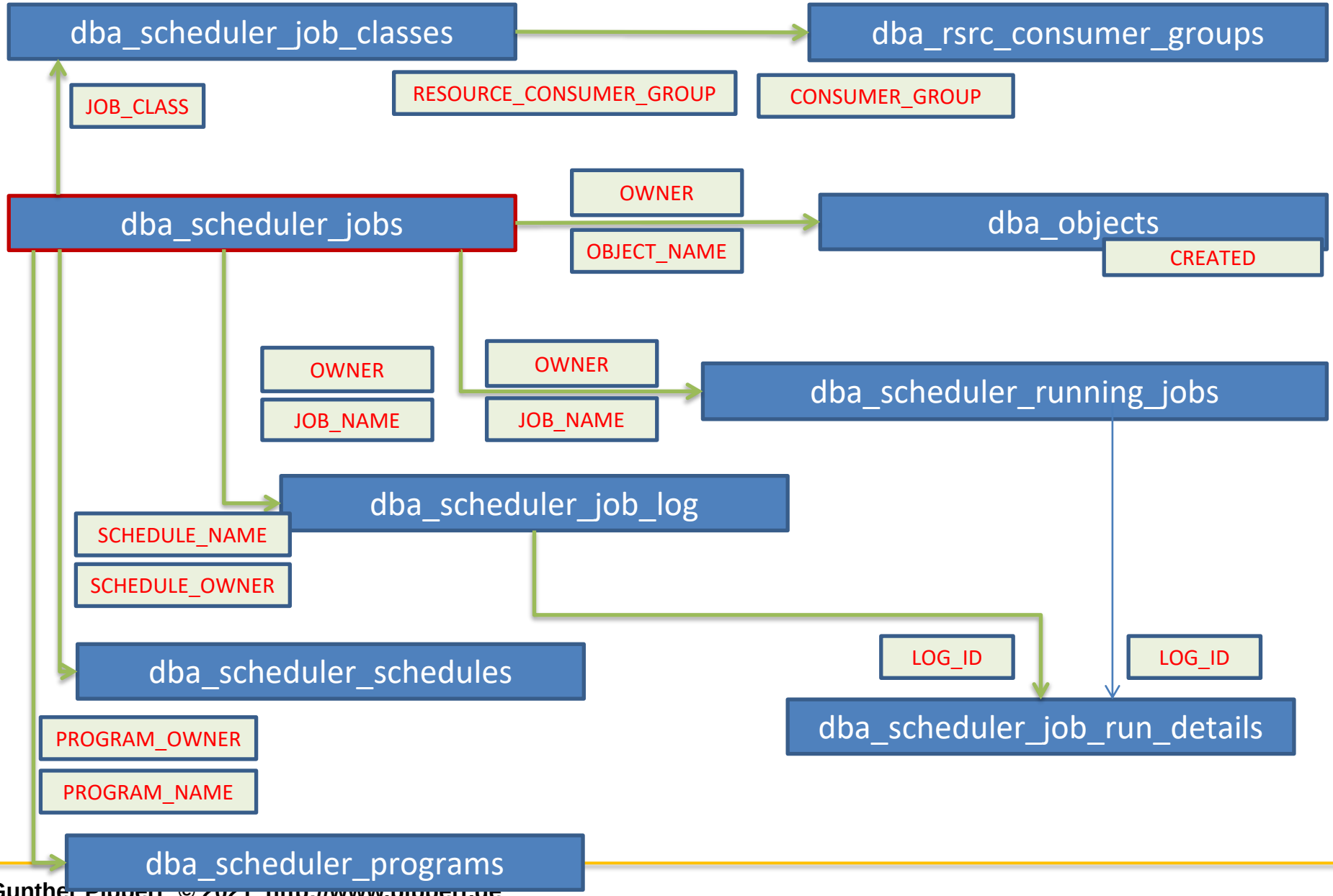

JOB
Command Type
Stored Procedure
PL/SQL
Chain
Executable
Program
Schedule Type
Type: Event
Type: Standard
Use pre dev Schedule
Type: Calendar
Use PL/SQL Repeat Interval
Use pre dev Window
Instance Stickiness
enabled
Priority
Schema
Log Level
Auto Drop
Restart able
Global Attributes
DEFAULT_TIMEZONE
EMAIL_SENDER
EMAIL_SERVER
EMAIL_SERVER_CREDENTIAL
EMAIL_SERVER_ENCRYPTION
EVENT_EXPIRY_TIME
FILE_WATCHER_COUNT
LAST_OBSERVED_EVENT
LOG_HISTORY
MAX_JOB_SLAVE_PROCESSES
Chain
Schema
Event Schedule
Program
Schema
PL/SQL
Executable
Stored Procedure
Program Arguments
Job Class
Log Retention
Log Level
Resource Consumer Group
Service Name
Window Group
Windows
Type: Standard
Type: Calendar
Resource Plan
Scheduler Time Table
Schema
Time zone
Type: - Standard - Time Definition
Type: Event Definition
Type: Calendar Definition

Scheduler Meta Data – Zentrale Views

- Was wurde wie definiert – die wichtigsten Views:



Scheduler Meta Data – Weitere Views



Scheduler Meta Data – Zentrale Views (1)

```
select js.owner
       ,  js.job_name
       ,  decode (js.state,  'SCHEDULED', 'SHUD',  'DISABLED', 'DIS',
'RUNNING', 'RUN',  js.state) as state
       ,  js.JOB_CLASS
       ,  js.program_name
       ,  js.job_action
       ,  to_char (o.CREATED, 'dd.mm.yyyy hh24:mi') as CREATED
       ,  JOB_STYLE
       ,  js.run_count
       ,  js.failure_count
       ,  to_char (js.last_start_date, 'dd.mm.yyyy hh24:mi') as
last_start_date
       ,  to_char (js.next_run_date, 'dd.mm.yyyy hh24:mi') as
next_run_date
    from dba_scheduler_jobs js, dba_objects o
   where      js.owner = o.owner(+)
         and js.job_name = o.OBJECT_NAME(+)
```

Scheduler Meta Data – Zentrale Views (2)

```
 select d.owner||'.'||d.job_name as job_name
     ,l.job_class
     ,d.log_date
     ,d.status
     ,d.error# error_number
                   ,d.errors
     ,d.actual_start_date
     ,extract (second from d.cpu_used)
                       + (  extract (minute from d.cpu_used)
                          * 60)
                       + (  extract (hour from d.cpu_used)
                          * 60
                          * 60)  as timeused
     ,l.additional_info
  from dba_scheduler_job_run_details d
   right join dba_scheduler_job_log l on (d.log_id = l.log_id)
;
```

Job Log File Pflege

- Wie voll ist das Log? Wie alt sind die Daten?

```
SELECT COUNT(*)       AS count_log
      , MIN(log_date) AS first_log
      , MAX(log_date) AS last_log
      , trunc(MAX(log_date)) -trunc(MIN(log_date))  AS  log_tage
   FROM dba_scheduler_job_log
/

   COUNT_LOG  FIRST_LOG   LAST_LOG    LOG_TAGE
------------  ---------   ----------  -----------
       67538  04-NOV-16   01-SEP-20   1397
```

- Einstellen mit:

```
SYS@GPI> EXEC DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE('log_history','10');
```

https://www.pipperr.de/dokuwiki/doku.php?id=dba:oracle_scheduler_log_pflege

Job Log File Pflege – Problem – Lösch Job prüfen

- Job “PURGE_LOG” prüfen!

```
  SELECT js.owner
       ,  js.job_name
       ,  decode (js.state,  'SCHEDULED', 'SHUD',  'DISABLED', 'DIS',  'RUNNING', 'RUN',  js.state) AS state
       ,  js.JOB_CLASS
       ,  js.program_name
       ,  js.job_action
       ,  to_char (o.CREATED, 'dd.mm.yyyy hh24:mi') AS CREATED
       ,  js.run_count
       ,  js.failure_count
       ,  to_char (js.last_start_date, 'dd.mm hh24:mi') AS last_start_date
       ,  to_char (js.next_run_date, 'dd.mm hh24:mi') AS next_run_date
    FROM dba_scheduler_jobs js, dba_objects o
   WHERE      js.owner = o.owner(+)
         AND js.job_name = o.OBJECT_NAME(+)
         AND js.job_name LIKE 'PURGE_LOG'
ORDER BY owner, job_name
/
```

Job Log File Pflege – Problem

- Globale Einstellung wird von Job Class überschrieben

```
SELECT owner
     , job_class_name
     , logging_level
     , log_history
     , comments
 FROM DBA_SCHEDULER_JOB_CLASSES
ORDER BY  log_history
/
```

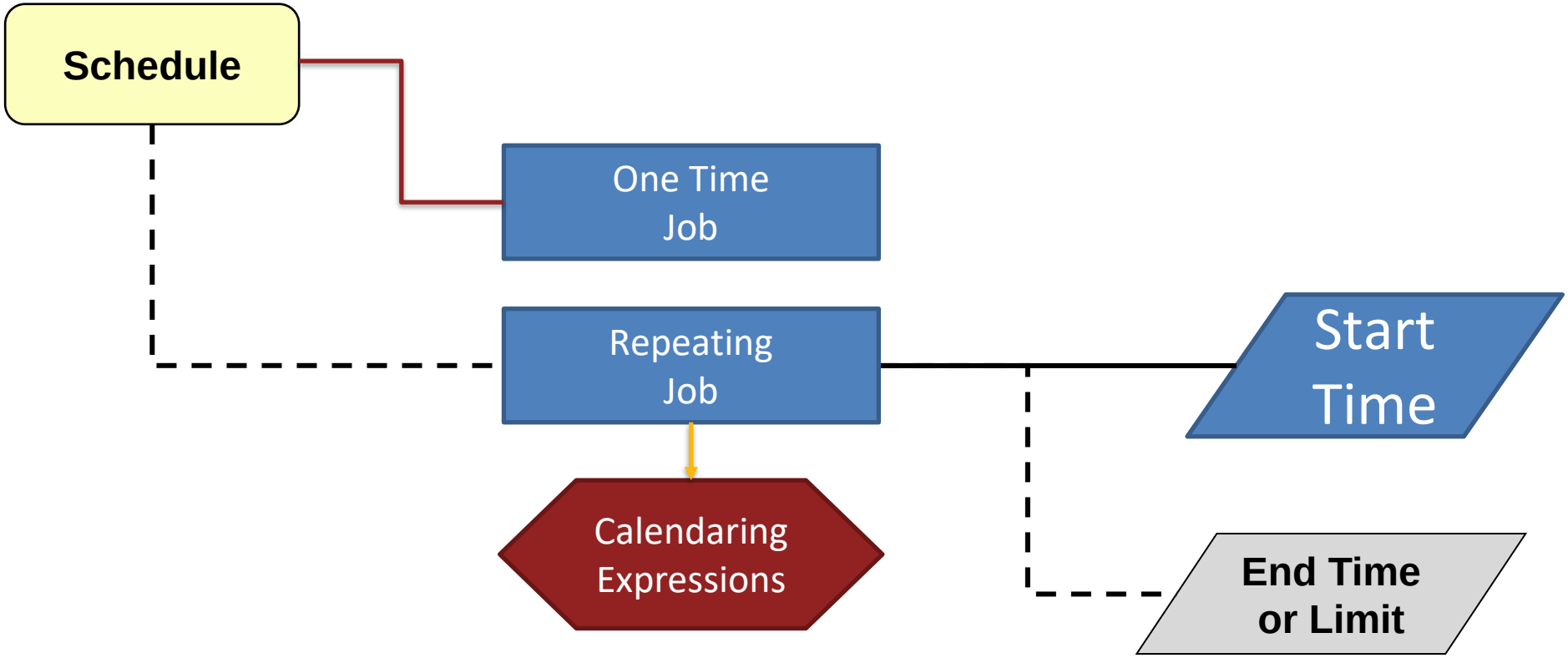
  - => Default Wert oft 10.000!
    - die Daten von heute werden dann am 30.07.4758 gelöscht; Wohl etwas spät …

- Einstellen mit

```
EXEC DBMS_SCHEDULER.SET_ATTRIBUTE('ORA$AT_JCNRM_OS','log_history','30');
```

Ein Schedule definieren (1)

- **Schedule** - Festlegen wann der Job laufen soll und wann er erneut starten muss

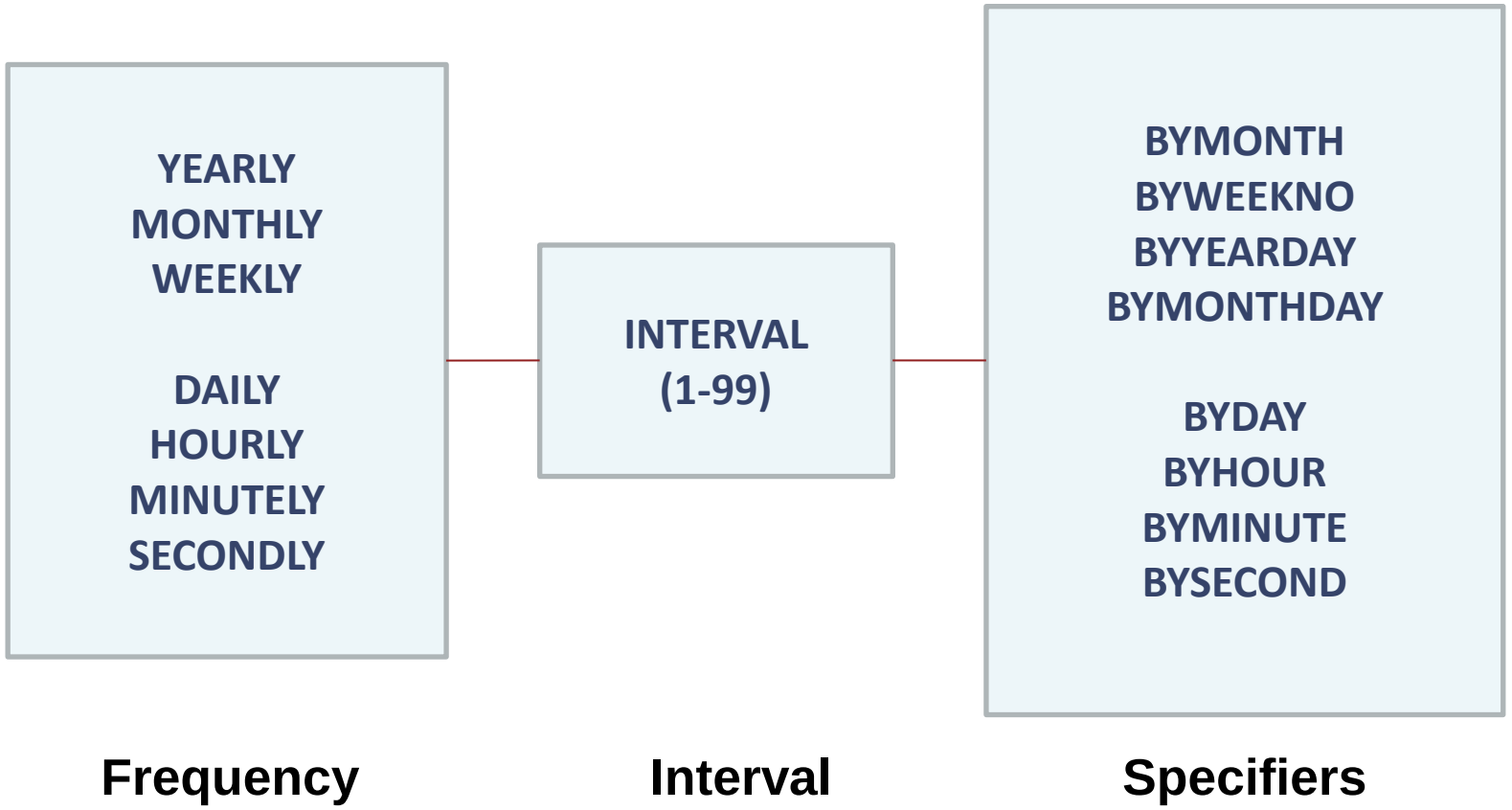


https://www.pipperr.de/dokuwiki/doku.php?id=dba:oracle_scheduler_repeat_interval

Ein Schedule definieren (2)

Calendaring Expressions

- Calendaring Expressions



Ein Schedule definieren (3)

- Calendaring Expressions – Beispiele

  - Every hour: FREQ=HOURLY; INTERVAL=1
  - Every two weeks: FREQ=WEEKLY; INTERVAL=2
  - Every 30 days: FREQ=DAILY; INTERVAL=30
  - Every 5 minutes: FREQ=MINUTELY; INTERVAL=5
  - Every second: FREQ=SECONDLY

  - Jeden 15th im Monat: FREQ=MONTHLY; BYMONTHDAY=15
  - Jede 4. Woche am Montag: FREQ=YEARLY; BYWEEKNO=4,8,12,16,20,24,28,32,36,40,44,48,52; BYDAY=MON
  - 14:30 jeden Dienstag: FREQ=WEEKLY; BYDAY=TUE; BYHOUR=14; BYMINUTE=30

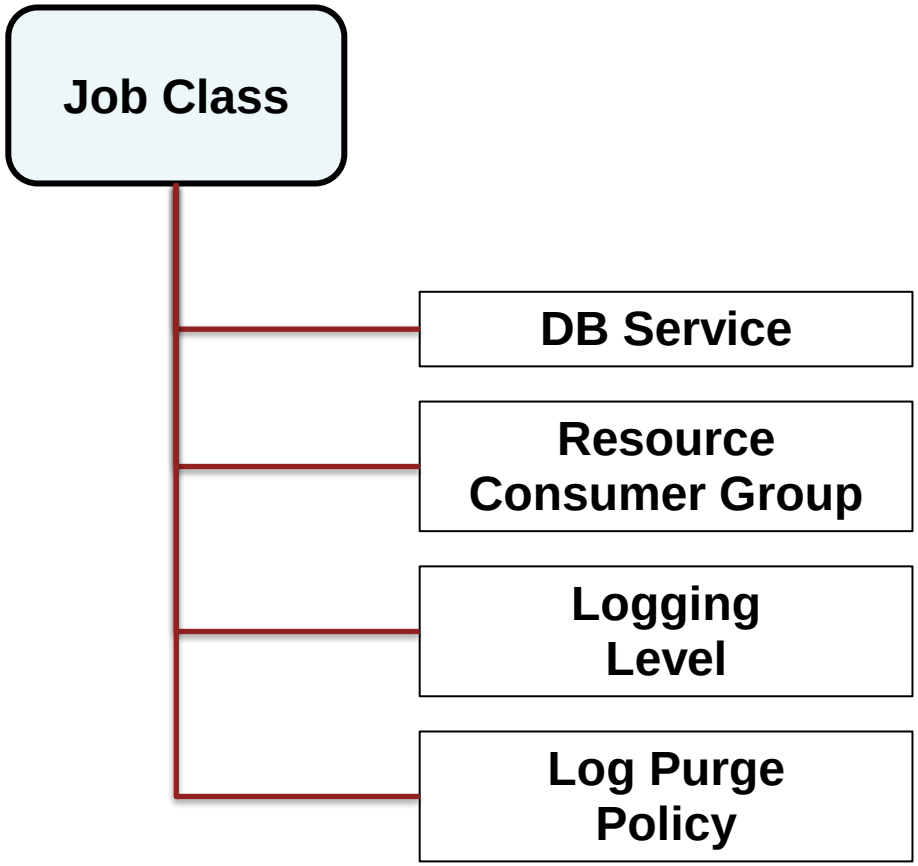
Eine "Calendaring Expressions" testen

- DBMS_SCHEDULER EVALUATE_CALENDAR_STRING

```
DECLARE
        v_next_run_date TIMESTAMP;
        v_start_date    TIMESTAMP:=systimestamp;
        v_return_date_after TIMESTAMP
BEGIN
  FOR i IN 1 ..10
  loop
  DBMS_SCHEDULER.EVALUATE_CALENDAR_STRING(
                 calendar_string   => 'FREQ=MINUTELY; INTERVAL=15'
               , start_date        => v_start_date
               , return_date_after => v_return_date_after
               , next_run_date     => v_next_run_date);

        DBMS_OUTPUT.PUT_LINE('-- Info actual date :: '||to_char(v_start_date,'dd.mm.yyyy hh24:mi')||
                             '   --> next_run_date:: '||to_char(v_next_run_date,'dd.mm.yyyy hh24:mi')
        );
         v_return_date_after := v_next_run_date;
  END loop;
END;
/
```

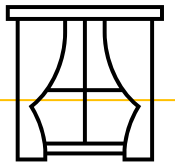
Job Class

- Die Elemente der Job Class

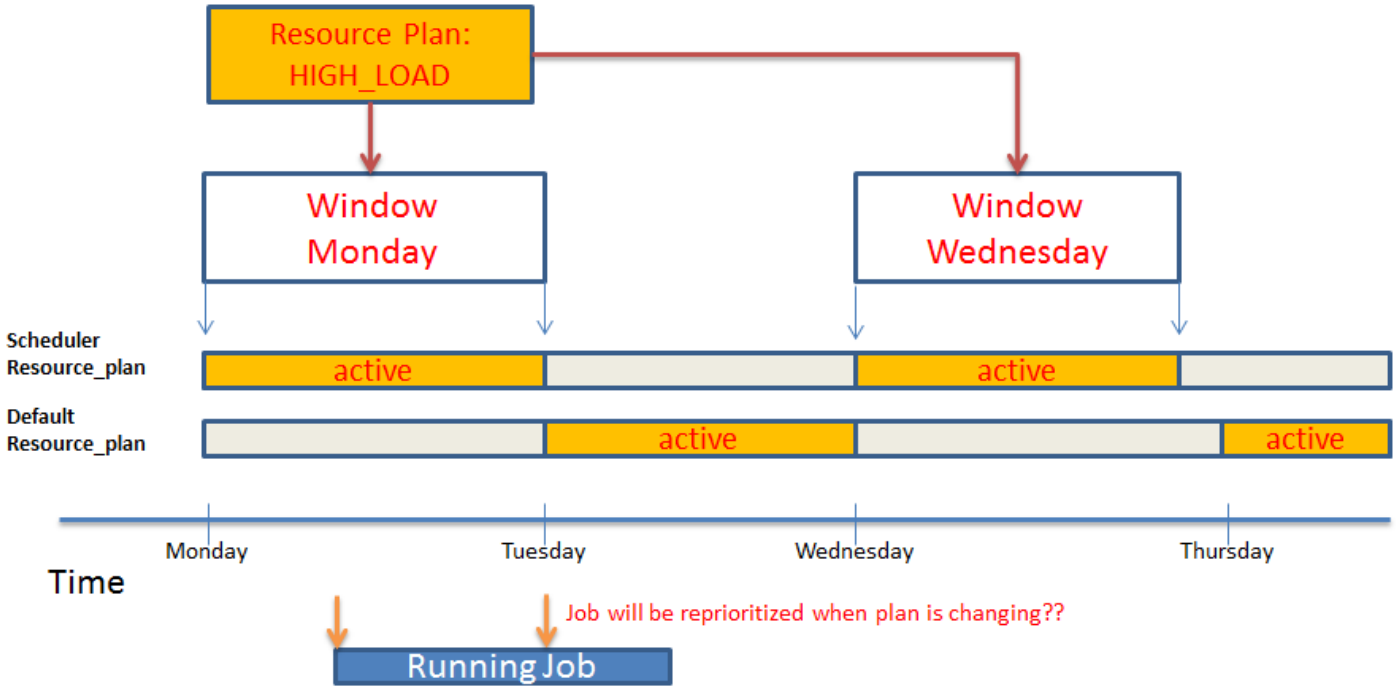



```
begin
DBMS_SCHEDULER.CREATE_JOB_CLASS(
      job_class_name          => 'ADMIN_JOBS'
    , resource_consumer_group => 'ADMIN_JOBS'
    , service                 => 'ADMIN_SRV'
);
End;
/
```

Die Window Funktionalität



- Job Lauf auf Zeiträume einschränken



Job priority/resource usage will be defined by:

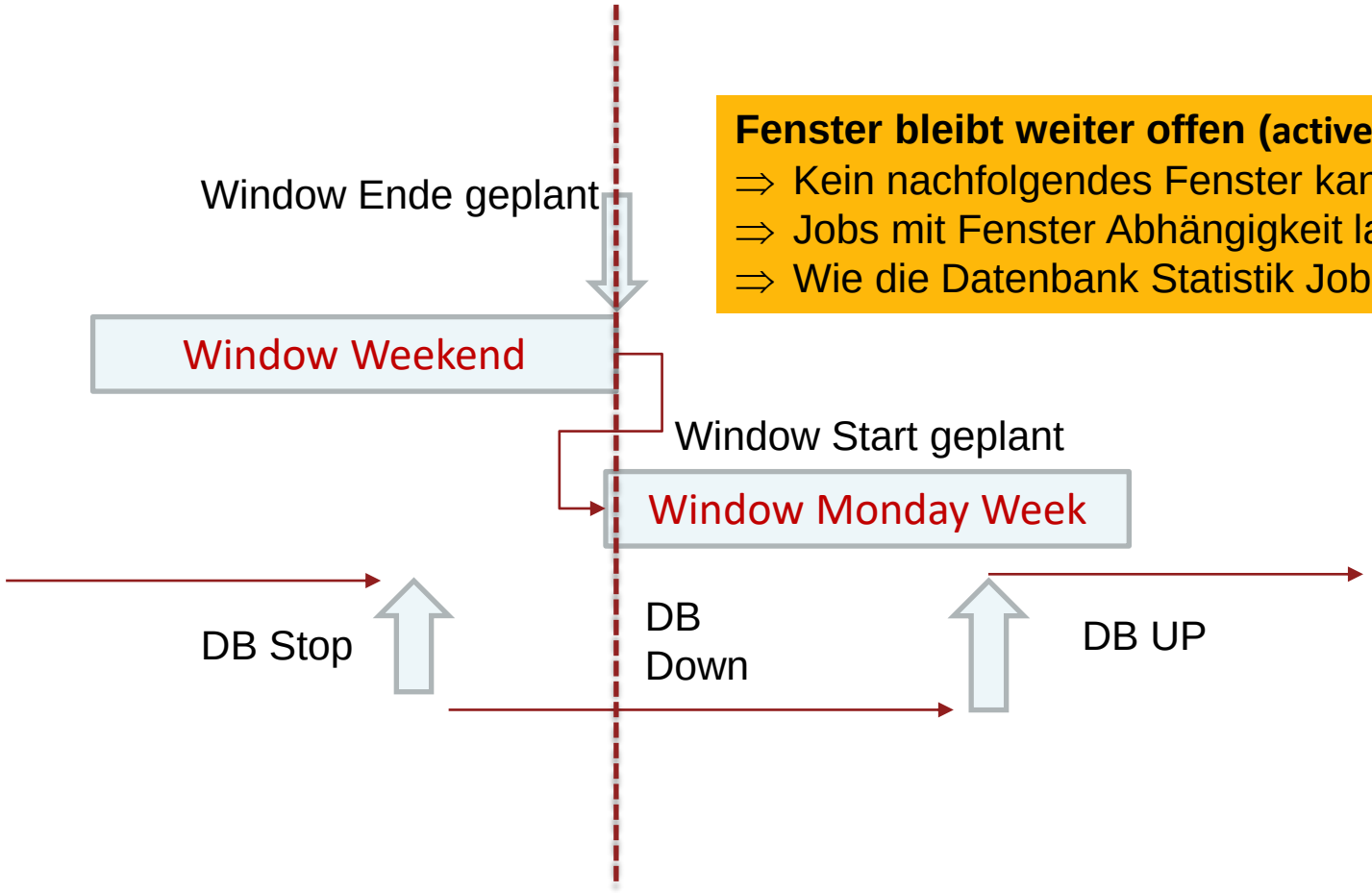
A) Over the Consumer Group Mappings Priority - only if NO Consumer Group is defined:

- **DB Service**
- **DB User to primary consumer group relation**

B) The settings of the Resource Consumer Group

Was passiert wenn ein Window offen bleibt?

- Szenario: Datenbank wird gestoppt und dann ein paar Stunden später wieder gestartet



Fenster bleibt weiter offen (active)
⇒ Kein nachfolgendes Fenster kann gestartet werden
⇒ Jobs mit Fenster Abhängigkeit laufen nun nicht mehr!
⇒ Wie die Datenbank Statistik Jobs!

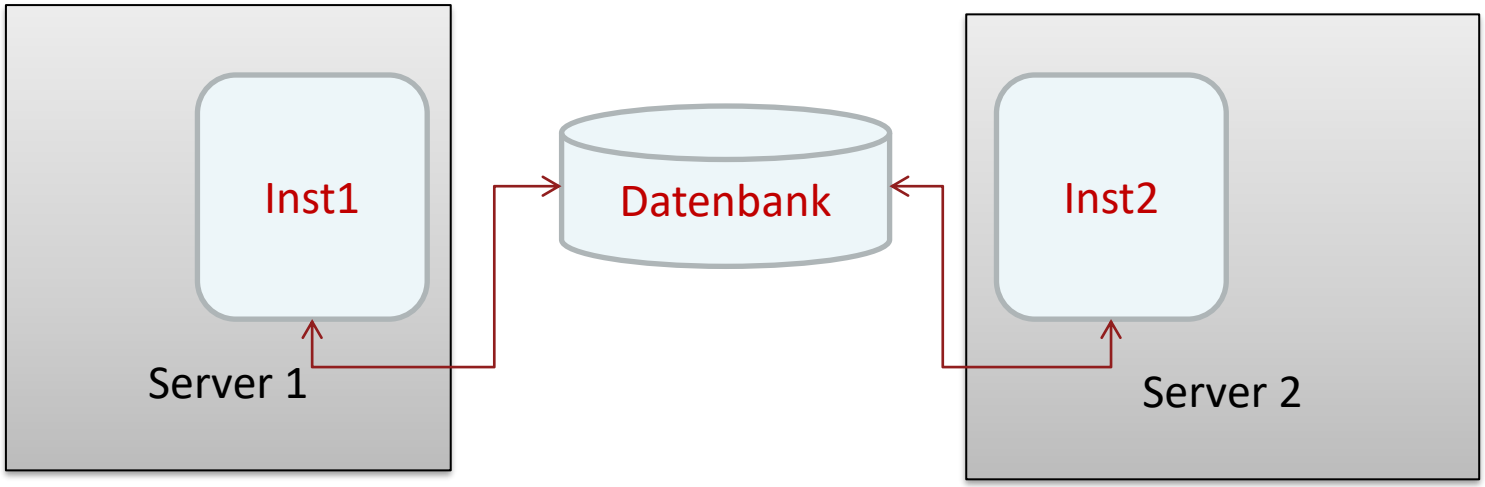
Prüfen mit:

- Nach längeren Downtimes prüfen:

```
column check_active format a10    heading 'Check|if ok’

select window_name
    ,  to_char (last_start_date, 'DD.MM.YYYY HH24:MI') as last_start_date
    ,  enabled
    ,  active
    ,  decode (active, 'TRUE', '<==CHECK IF POSSIBLE', '-') as check_active
 from dba_scheduler_windows
order by last_start_date
/
```

Real Applikation Cluster und Jobs

- Wo läuft der Job am Ende?



In einer Cluster Umgebung kann definiert werden auf welchem Knoten der Job laufen soll.

1. Fest auf eine Instance binden – Instance ID setzen
2. Letzte Instance verwenden auf dem der Job mal lief - Stickiness => True
3. Job Klasse und Service verwenden (am saubersten!)

RAC – Jobausführung per Service steuern

- **Job Klasse und Service**
  - Einen Service im Cluster anlegen der NUR auf der gewünschten Instance läuft
  - Eine Job Klasse anlegen, die diesen Service verwendet
  - Dem Job diese Job Klasse zuordnen

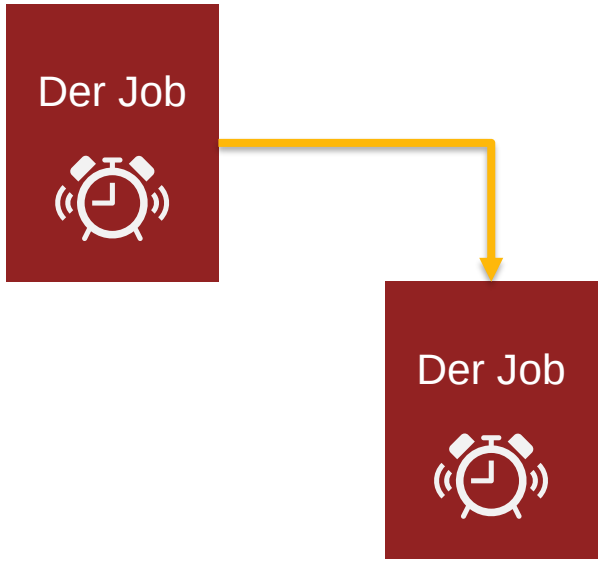
File Watcher Job - Übersicht

- Scheduler Credential anlegen
  - Execute granten
- File Watcher anlegen
  - Execute granten
- Scheduler Programm für den eigentlichen Task anlegen
- Event Based Job mit Referenz auf den File Watcher anlegen
- Alles aktiveren

Beispiel:
https://docs.oracle.com/cd/E18283_01/server.112/e17120/scheduse005.htm

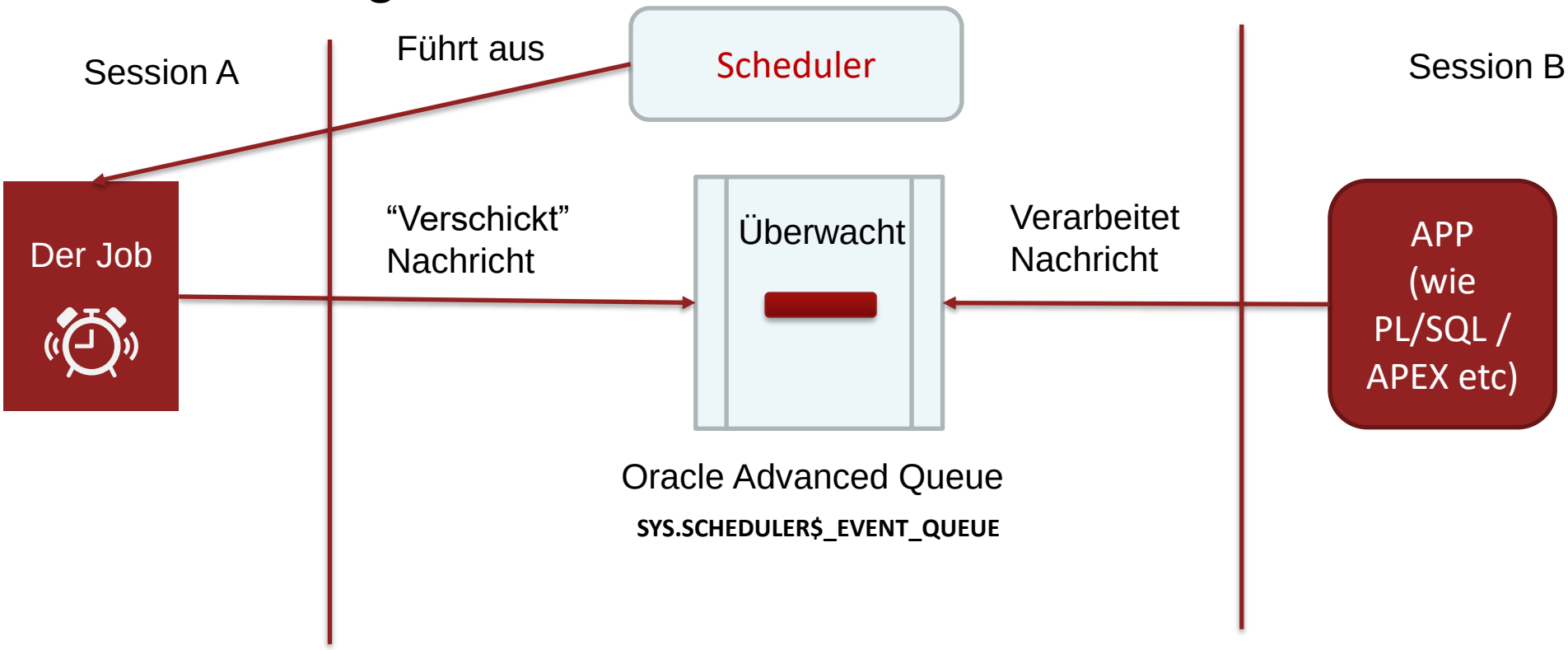
Job Chain

- Zwei Jobs miteinander verknüpfen



"Event Based Job" – Job sendet Nachricht

- Job erzeugt Events

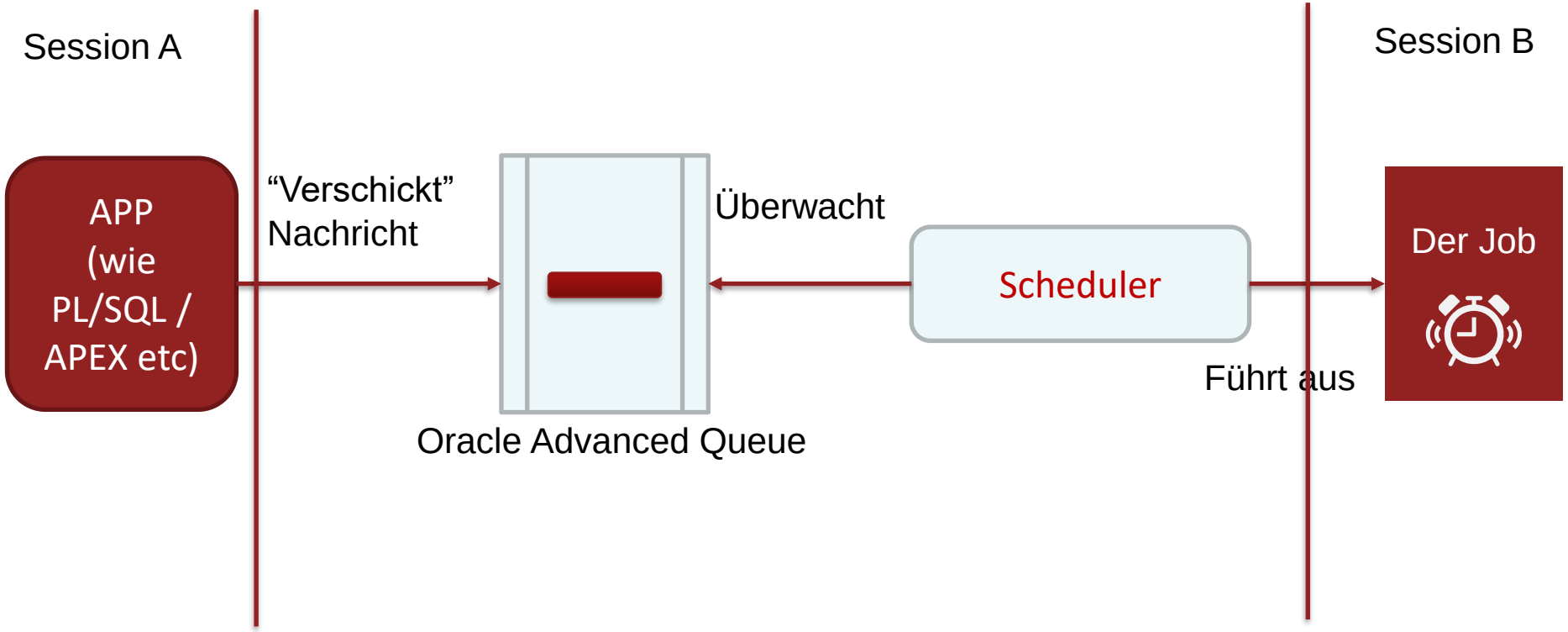


Szenario: Job benachrichtigt Application über Fehler

Seit Oracle 10g Database Release 2

"Event Based Job" – Job wird gestartet

- Aus der Applikation über “Events” einen Job triggern



Seit Oracle 10g Database Release 2

Szenario: Parallel zur "Applikations Session Thread" eine weitere Session starten

How to Build up an Event Based Job - An Example (Doc ID 1431664.1)

E-Mail Benachrichtigung aktiveren

- Eine Änderung des Job Status triggert die Versendung der Mail
  - Zuvor das Versenden von Mails mit ACL etc. aktivieren
  - Parameter definieren

```
BEGIN
  DBMS_SCHEDULER.set_scheduler_attribute('email_server', 'postfix.pipperr.local:25');
  DBMS_SCHEDULER.set_scheduler_attribute('email_sender', 'database@pipperr.de');
END;
/
```

  - An bestehenden Job “anhängen”

```
BEGIN
 DBMS_SCHEDULER.add_job_email_notification (
  job_name          =>  'DATA_IMPORT_RUN',
  recipients        =>  'gunther@pipperr.de',
  events            =>  'job_failed',
  filter_condition => ':event.error_code=600');
END;
/
```

Event wie.:job_started, job_succeeded , job_failed

Zusätzlich filtern

  - Überwachen über DBA_SCHEDULER_NOTIFICATIONS

Summary

Fazit „DB Jobs“

- APEX Automation
  - Jobs einfach in die eigene APP integrieren

- Der Oracle Scheduler
  - Etwas komplexer in der Verwendung als DBA_JOBS
  - Aber sehr mächtig!

Mehr

- Quellen
  - https://download.oracle.com/owsf_2003/40615_goorah.ppt

- Blog Gunther Pippèrr https://www.pipperr.de/dokuwiki/doku.php?id=dba:oracle_scheduler
- Wieder mal eine andere Skript Library
  - https://github.com/gpipperr/OraPowerShell

- Bildmaterial : https://pixabay.com

Source Code: https://github.com/gpipperr/APEX_CONNECT_2021_JOB_AUTOMATION

F&A

Fragen

APEX Job Steuerung

Diskussion

Fragen ?