Table of Contents

[1 Base Objects 3](#_Toc39794264)

[1.1 DATABASE TABLES 3](#_Toc39794265)

[1.2 DATABASE TRIGGERS 3](#_Toc39794266)

[1.3 DATABASE SEQUENCES 3](#_Toc39794267)

[1.4 DATABASE PACKAGES 3](#_Toc39794268)

[1.5 UNIX FILES 3](#_Toc39794269)

[1.6 SQL FILES IN UNIX 3](#_Toc39794270)

[2 Functionality of each object 4](#_Toc39794271)

[2.1 ETL\_JOB\_CONFIG table 4](#_Toc39794272)

[2.2 ETL\_JOB\_CONFIG\_AUDIT table 4](#_Toc39794273)

[2.3 ETL\_JOB\_STATUS table 5](#_Toc39794274)

[2.4 ETL\_JOB\_RUN table 5](#_Toc39794275)

[2.5 ETL\_JOB\_LIST table 6](#_Toc39794276)

[2.6 ETL\_JOB\_GLOBAL\_CONFIG table 6](#_Toc39794277)

[2.7 ETL\_JOB\_LOG table 6](#_Toc39794278)

[2.8 AUD\_TR\_ETL\_JOB\_CONFIG trigger 7](#_Toc39794279)

[2.9 SEQ\_ETL\_JOB\_CONFIG\_ID sequence 7](#_Toc39794280)

[2.10 SEQ\_ETL\_JOB\_RUN\_ID sequence 7](#_Toc39794281)

[2.11 ETL\_JOB Package 7](#_Toc39794282)

[2.11.1 SET\_ETL\_JOBS procedure 7](#_Toc39794283)

[2.11.2 ADD\_ETL\_JOB procedure 8](#_Toc39794284)

[2.11.3 UPD\_ETL\_JOB procedure 8](#_Toc39794285)

[2.11.4 GET\_NEXT\_JOB\_SCHEDULE function 9](#_Toc39794286)

[2.12 etl\_job\_control.sh 10](#_Toc39794287)

[2.13 etl\_job\_exec.sh 10](#_Toc39794288)

[2.14 .set\_env.sh 11](#_Toc39794289)

[2.15 kettle.properties file 11](#_Toc39794290)

[2.16 testscript.sh 11](#_Toc39794291)

[2.17 set\_etl\_jobs.sql 11](#_Toc39794292)

[2.18 get\_etl\_jobs.sql 12](#_Toc39794293)

[2.19 get\_etl\_stuck\_jobs.sql 12](#_Toc39794294)

[2.20 add\_etl\_job.sql 12](#_Toc39794295)

[2.21 upd\_etl\_job.sql 13](#_Toc39794296)

[3 ETL Job Control - Process Flow 14](#_Toc39794297)

[4 Process Flow – Verification (NOT UPDATED) 15](#_Toc39794298)

[4.1 Create all DB Objects 15](#_Toc39794299)

[4.2 Load a job into BPM\_JOB\_CONFIG table 15](#_Toc39794300)

[4.3 Execute etl\_job\_control.sh 15](#_Toc39794301)

[4.4. Verify the job list extracted 15](#_Toc39794302)

[4.5 Check new run instance of the job is created 15](#_Toc39794303)

[4.6 Check the status of Job and run instance 15](#_Toc39794304)

[4.7 Make sure JOB\_NEXT\_EXEC\_DTM is calculated and updated 15](#_Toc39794305)

[4.8 Verify the job is successfully executed 15](#_Toc39794306)

[4.9 Check the status of Job and run instance after job execution 15](#_Toc39794307)

[4.10 Make some changes to ETL\_JOB\_CONFIG and verify audit table 15](#_Toc39794308)

[4.11 Verify log files as well as ETL\_JOB\_LOG table 15](#_Toc39794309)

# 1 Base Objects

## 1.1 DATABASE TABLES

* ETL\_JOB\_CONFIG : Configuration table for Jobs
* ETL\_JOB\_CONFIG\_AUDIT : Audit table that tracks all changes to Configuration table
* ETL\_JOB\_STATUS : Current status for each job, next execution time, etc.,
* ETL\_JOB\_RUN : History table for all job runs
* ETL\_JOB\_LIST : Staging table that gets refreshed by ETL control job
* ETL\_JOB\_GLOBAL\_CONFIG : Global parameters for ETL job control process
* ETL\_JOB\_LOG : Hold all necessary logs and error messages

## 1.2 DATABASE TRIGGERS

* AUD\_TR\_ETL\_JOB\_CONFIG : Tracks all Update/Delete changes to

ETL\_JOB\_CONFIG table in ETL\_JOB\_CONFIG\_AUDIT table

## 1.3 DATABASE SEQUENCES

* SEQ\_ETL\_JOB\_CONFIG\_ID : Sequence generator for ETL\_JOB\_CONFIG.JOB\_ID
* SEQ\_ETL\_JOB\_RUN\_ID : Sequence generator for ETL\_JOB\_RUN.RUN\_ID

## 1.4 DATABASE PACKAGES

* ETL\_JOB
  + Procedure : SET\_ETL\_JOBS : Identifies list of jobs to execute/ got stuck
  + Procedure : ADD\_ETL\_JOB : starts a new run instance of the job
  + Procedure : UPD\_ETL\_JOB : update the status for the run instance
  + Function : GET\_NEXT\_JOB\_SCHEDULE : get the next job schedule date/time
  + Function : IS\_NUMBER\_VALID : sub function for GET\_NEXT\_JOB\_SCHEDULE
  + Function : PARSE\_NUMBER\_RANGE : sub function for GET\_NEXT\_JOB\_SCHEDULE
  + Function : GET\_NEXT\_NUMBER : sub function for GET\_NEXT\_JOB\_SCHEDULE
  + Function : GET\_MIN\_NUMBER : sub function for GET\_NEXT\_JOB\_SCHEDULE

## 1.5 UNIX FILES

* etl\_job\_control.sh : main ETL job control shell script
* etl\_job\_exec.sh : shell script that gets executed for each job
* .set\_env.sh : environment variables, kettle path, etc.,
* kettle.properties : database connection details
* testscript.sh : only for prototype testing

## 1.6 SQL FILES IN UNIX

* set\_etl\_jobs.sql : executes SET\_ETL\_JOBS procedure of ETL\_JOB package
* get\_etl\_jobs.sql : SQL to extract list of jobs to execute from ETL\_JOB\_LIST table
* get\_etl\_stuck\_jobs : SQL to extract list of stuck jobs from ETL\_JOB\_LIST table
* add\_etl\_job.sql : executes ADD\_ETL\_JOB procedure of ETL\_JOB package
* upd\_etl\_job.sql : executes UPD\_ETL\_JOB procedure of ETL\_JOB package

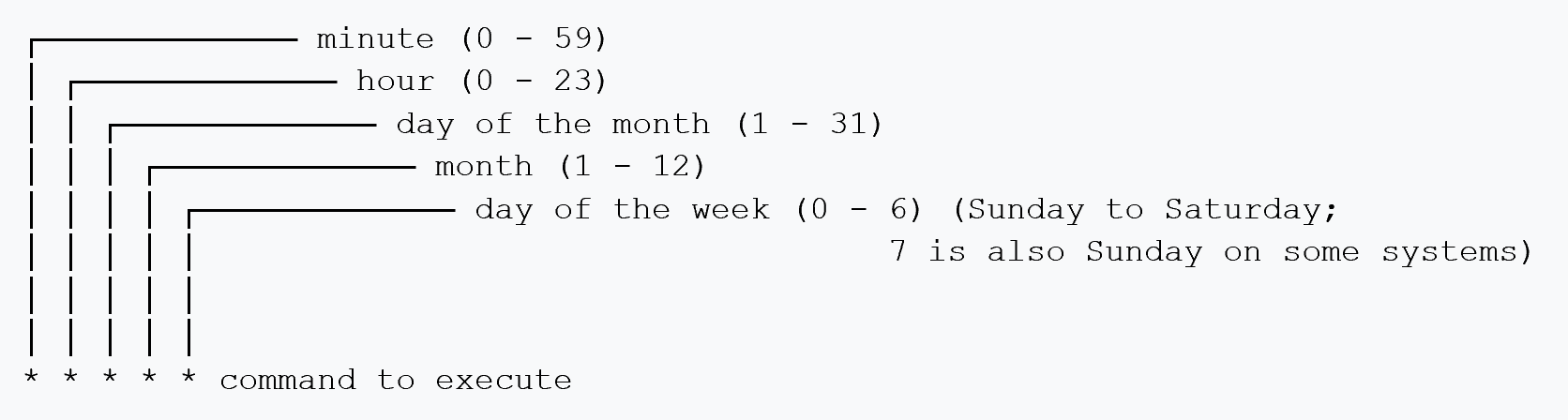
# 2 Functionality of each object

## 2.1 ETL\_JOB\_CONFIG table

This table holds all the jobs to be scheduled. List of fields in this table are:

|  |  |
| --- | --- |
| **FIELD** | **SIGNIFICANCE** |
| JOB\_ID | Unique ID for each job - auto generated using SEQ\_ETL\_JOB\_CONFIG\_ID sequence |
| PROJECT\_NAME | Foreign key to valid project names in BPM\_PROJECT\_LOOKUP.NAME |
| JOB\_NAME | Job Name - This help identify the job like: pa\_run\_bpm |
| JOB\_TYPE | Valid Values: PARENT, CHILD, INDIVIDUAL, ADHOC |
| PARENT\_JOB\_ID | In case of child Job, this gives the parent job id |
| JOB\_SCHEDULE | Defines the schedule for the job using CRONTAB format |
| JOB\_ENABLED | Valid Values: Y, N |
| JOB\_SCRIPT\_NAME | Actual script name WITHOUT script path, like : "pa\_run\_bpm.sh" |
| JOB\_TIMEOUT\_SEC | Maximum execution time, after which a job is considered stuck - in seconds. |
| LAST\_UPD\_DT | Last Updated Datetime |
| LAST\_UPD\_USER | Last Updated User |

* JOB\_SCHEDULE allows the same format as the CRON as given below. Custom code will be developed to accept all possible cron formats, but it may limit to fewer formats if the logic becomes more complex



* JOB\_TYPE allows the below valid values based on the type of the job:

|  |  |
| --- | --- |
| INDIVIDUAL | No Parent or Child jobs. It should have its own JOB\_SCHEDULE |
| PARENT | It is expected to have some dependent/child jobs. It should have its own JOB\_SCHEDULE. PARENT job run is considered as complete only when all the child jobs are completed. |
| CHILD | Must have PARENT job and cannot have JOB\_SCHEDULE of its own. This gets triggered once PARENT job is complete. CHILD job can also have CHILD jobs further down the hierarchy. |
| ADHOC | Get disabled after each run. For additional runs, it should be enabled. If JOB\_SCHEDULE is provided, ADHOC job will be executed based on the next execution date/time based on the JOB\_SCHEDULE. If JOB\_SCHEDULE is not provided, it will picked up by the next ETL job control run. |

* JOB\_SCRIPT\_NAME should be the actual shell script name without any path. The path will be read from the .set\_env.sh. ETL Job control process will check for this script in one of the script paths from the .set\_env.sh and executes the script if it exists in one of those paths.

## 2.2 ETL\_JOB\_CONFIG\_AUDIT table

This table captures DELETE / UPDATE changes to ETL\_JOB\_CONFIG table thru AUD\_TR\_ETL\_JOB\_CONFIG trigger on this table.

|  |  |
| --- | --- |
| **FIELD** | **SIGNIFICANCE** |
| JOB\_ID | source - ETL\_JOB\_CONFIG table |
| AUDIT\_TYPE | Valid Values: DELETE, UPDATE |
| AUDIT\_UPD\_DT | Datetime audit was logged |
| PROJECT\_NAME | source - ETL\_JOB\_CONFIG table |
| JOB\_NAME | source - ETL\_JOB\_CONFIG table |
| JOB\_TYPE | source - ETL\_JOB\_CONFIG table |
| PARENT\_JOB\_ID | source - ETL\_JOB\_CONFIG table |
| JOB\_SCHEDULE | source - ETL\_JOB\_CONFIG table |
| JOB\_ENABLE | source - ETL\_JOB\_CONFIG table |
| JOB\_TIMEOUT\_SEC | source - ETL\_JOB\_CONFIG table |
| JOB\_SCRIPT\_NAME | source - ETL\_JOB\_CONFIG table |
| LAST\_UPD\_DT | source - ETL\_JOB\_CONFIG table |
| LAST\_UPD\_USER | source - ETL\_JOB\_CONFIG table |

## 2.3 ETL\_JOB\_STATUS table

This table stores the latest status of each job, whether job is runnable or not, and the next execution date/time

|  |  |
| --- | --- |
| **FIELD** | **SIGNIFICANCE** |
| JOB\_ID | Foreign key to ETL\_JOB\_CONFIG.JOB\_ID |
| JOB\_NEXT\_EXEC\_DT | This gets populated after each run based on ETL\_JOB\_CONFIG.JOB\_SCHEDULE |
| JOB\_RUNNABLE | Valid Values: Y, N |
| JOB\_STATUS | Job current status (of last run).. Foreign key to ETL\_JOB\_RUN.RUN\_STATUS |
| JOB\_LAST\_RUN\_ID | will store the last RUN\_ID of the job from ETL\_JOB\_RUN table. |
| LAST\_UPD\_DT | Last Updated Datetime |
| LAST\_UPD\_USER | Last Updated User |

* JOB\_RUNNABLE: This field will replace the check-file logic. When any job is running, it is set to NO and will be updated with YES only when the job is COMPLETED, FAILED, RESET, or CANCELLED
* JOB\_NEXT\_EXEC\_DT: This gets populated when a new run instance of the job is STARTED by calling GET\_NEXT\_JOB\_SCHEDULE function of the ETL\_JOB package, based on current system date/time and ETL\_JOB\_CONFIG.JOB\_SCHEDULE field
* JOB\_RUN\_ID: This was added to store the last RUN\_ID of the job to simplify some queries. Without this field, the queries have to hit ETL\_JOB\_RUN table multiple times to get the latest RUN\_ID of the job

## 2.4 ETL\_JOB\_RUN table

This table holds each run details of the jobs:

|  |  |
| --- | --- |
| **FIELD** | **SIGNIFICANCE** |
| RUN\_ID | Unique ID for each job - auto generated using SEQ\_ETL\_JOB\_RUN\_ID sequence |
| JOB\_ID | Foreign key to ETL\_JOB\_CONFIG.JOB\_ID |
| RUN\_STATUS | Valid Values: STARTED, FAILED, COMPLETED, CANCELLED, RESET, WAITING etc., |
| RUN\_START\_DT | Start Datetime for the instance |
| RUN\_END\_DT | End Datetime for the instance |
| LAST\_UPD\_DT | Last Updated Datetime |
| LAST\_UPD\_USER | Last Updated User |

* RUN\_STATUS: WAITING will be status when the parent job is complete and it has at least one active dependent job to get executed. When all the active dependent jobs are complete, the parent job status will get updated as COMPLETED.
* When the job is failed, ETL\_JOB\_LOG table will be updated with the corresponding error details, so this table will not store the error details.

## 2.5 ETL\_JOB\_LIST table

This is a staging table used to store the list of jobs to be executed and list of jobs that got stuck. This table is refreshed every time SET\_ETL\_JOBS procedure of the ETL\_JOB package is executed.

|  |  |
| --- | --- |
| **FIELD** | **SIGNIFICANCE** |
| JOB\_ID | source - ETL\_JOB\_CONFIG table |
| PROJECT\_NAME | source - ETL\_JOB\_CONFIG table |
| JOB\_NAME | source - ETL\_JOB\_CONFIG table |
| JOB\_TYPE | source - ETL\_JOB\_CONFIG table |
| PARENT\_JOB\_ID | source - ETL\_JOB\_CONFIG table |
| JOB\_SCRIPT\_NAME | source - ETL\_JOB\_CONFIG table |
| JOB\_NEXT\_EXEC\_DT | source - ETL\_JOB\_STATUS table |
| JOB\_STUCK | Valid Values: Y, N |

* JOB\_STUCK: This field will indicate whether the job got stuck (when it is Y) or the job is ready for run (when it is N). Stuck jobs will be reported only one a day.

## 2.6 ETL\_JOB\_GLOBAL\_CONFIG table

This table will be used to store the global constants at project level.

For example: this table will store,

* how many jobs can be kicked off in parallel with each etl job control process run
* what is the runout time for any job
* how much history should be retained in the ETL\_JOB\_RUN or ETL\_JOB\_LOG or ETL\_JOB\_CONFIG\_AUDIT tables etc.,

|  |  |
| --- | --- |
| **FIELD** | **SIGNIFICANCE** |
| PROJECT\_NAME | Foreign key to ETL\_JOB\_CONFIG.PROJECT\_NAME field |
| PARAM\_KEY | Parameter Key |
| PARAM\_VALUE | Parameter Value |

## 2.7 ETL\_JOB\_LOG table

This table will store all the debug and non-error logs for each job run.

|  |  |
| --- | --- |
| **FIELD** | **SIGNIFICANCE** |
| JOB\_ID | Foreign key to ETL\_JOB\_CONFIG.JOB\_ID |
| RUN\_ID | Foreign key to ETL\_JOB\_RUN.RUN\_ID |
| STEP\_ID | Step number within the job logic |
| STEP\_START\_DT | Start Datetime for the step |
| STEP\_END\_DT | End Datetime for the step |
| STEP\_STATUS | Valid Values: STARTED, COMPLETED, FAILED |
| LOG\_SEVERITY\_LEVEL | Valid Values: FATAL, ERROR, WARN, INFO, DEBUG |
| LOG\_CODE | User defined code |
| LOG\_DESC | User defined message for the error or log |
| LAST\_UPD\_DT | Last Updated Datetime |
| LAST\_UPD\_USER | Last Updated User |

## 2.8 AUD\_TR\_ETL\_JOB\_CONFIG trigger

* This trigger on ETL\_JOB\_CONFIG table captures all the changes after UPDATE & DELETE actions to this table and logs them into ETL\_JOB\_CONFIG\_AUDIT table.
* For UPDATE, it will log the pre-update (from :OLD) record
* For DELETE, it will log the deleted record

## 2.9 SEQ\_ETL\_JOB\_CONFIG\_ID sequence

This sequence will be used for ETL\_JOB\_CONFIG.JOB\_ID field to auto generate Job IDs.

## 2.10 SEQ\_ETL\_JOB\_RUN\_ID sequence

This sequence will be used for ETL\_JOB\_RUN.RUN\_ID field to auto generate Job Run IDs.

## 2.11 ETL\_JOB Package

This package will have the below procedures and functions.

## 2.11.1 SET\_ETL\_JOBS procedure

This procedure will truncate and load ETL\_JOB\_LIST table with list of jobs to be executed and list of jobs that got stuck for any reason. Stuck jobs logged into ETL\_JOB\_LOG table and are logged/reported only once a day. The criteria to pick up the jobs are:

Criteria for list of jobs to be executed:

* ADHOC jobs:
  + JOB\_ENABLED = YES
  + JOB\_NEXT\_EXEC\_DT <= SYSDATE() if JOB\_SCHEDULE is provided
* CHILD jobs:
  + JOB\_ENABLED = YES
  + Parent job’s
    - JOB\_ENABLED = YES
    - JOB\_STATUS = WAITING
* PARENT/INDIVIDUAL jobs
  + JOB\_ENABLED = YES
  + JOB\_RUNNABLE = YES
  + JOB\_NEXT\_EXEC\_DT <= SYSDATE()

Criteria for list of jobs that got stuck:

* Current run instance execution time is more than JOB\_TIMEOUT\_SEC (if this field is populated)
* Current run instance execution time is more than the maximum job execution time parameter in ETL\_JOB\_GLOBAL\_CONFIG table. This will be used only when JOB\_TIMEOUT\_SEC is not provided for the job
* INDIVIDUAL/PARENT Jobs:
  + JOB\_ENABLED = YES
  + JOB\_RUNNABLE = NO
  + JOB\_NEXT\_EXEC\_DT <= SYSDATE()

## 2.11.2 ADD\_ETL\_JOB procedure

This procedure accepts Job ID as the parameter and will perform the below steps:

|  |  |
| --- | --- |
| **Step/Action** | **Updates Performed** |
| Insert a new record in ETL\_JOB\_RUN table | ETL\_JOB\_RUN.RUN\_STATUS = STARTED ETL\_JOB\_RUN.RUN\_START\_DT = SYSDATE |
| Picks up the latest RUN\_ID created |  |
| If Job has no records in ETL\_JOB\_STATUS table, it inserts a new record for the job | ETL\_JOB\_STATUS.JOB\_STATUS = STARTED ETL\_JOB\_STATUS.JOB\_RUNNABLE = N ETL\_JOB\_STATUS.JOB\_RUN\_ID = (latest RUN\_ID for the job) ETL\_JOB\_STATUS.JOB\_NEXT\_EXEC\_DT = (next exec date/time by calling GET\_NEXT\_JOB\_SCHEDULE function based on ETL\_JOB\_CONFIG.JOB\_SCHEDULE) |
| ADHOC job will be disabled after the run | ETL\_JOB\_CONFIG.JOB\_ENABLED = N |

## 2.11.3 UPD\_ETL\_JOB procedure

This procedure accepts Job ID and Status as the parameters, and will make necessary updates based on the status.

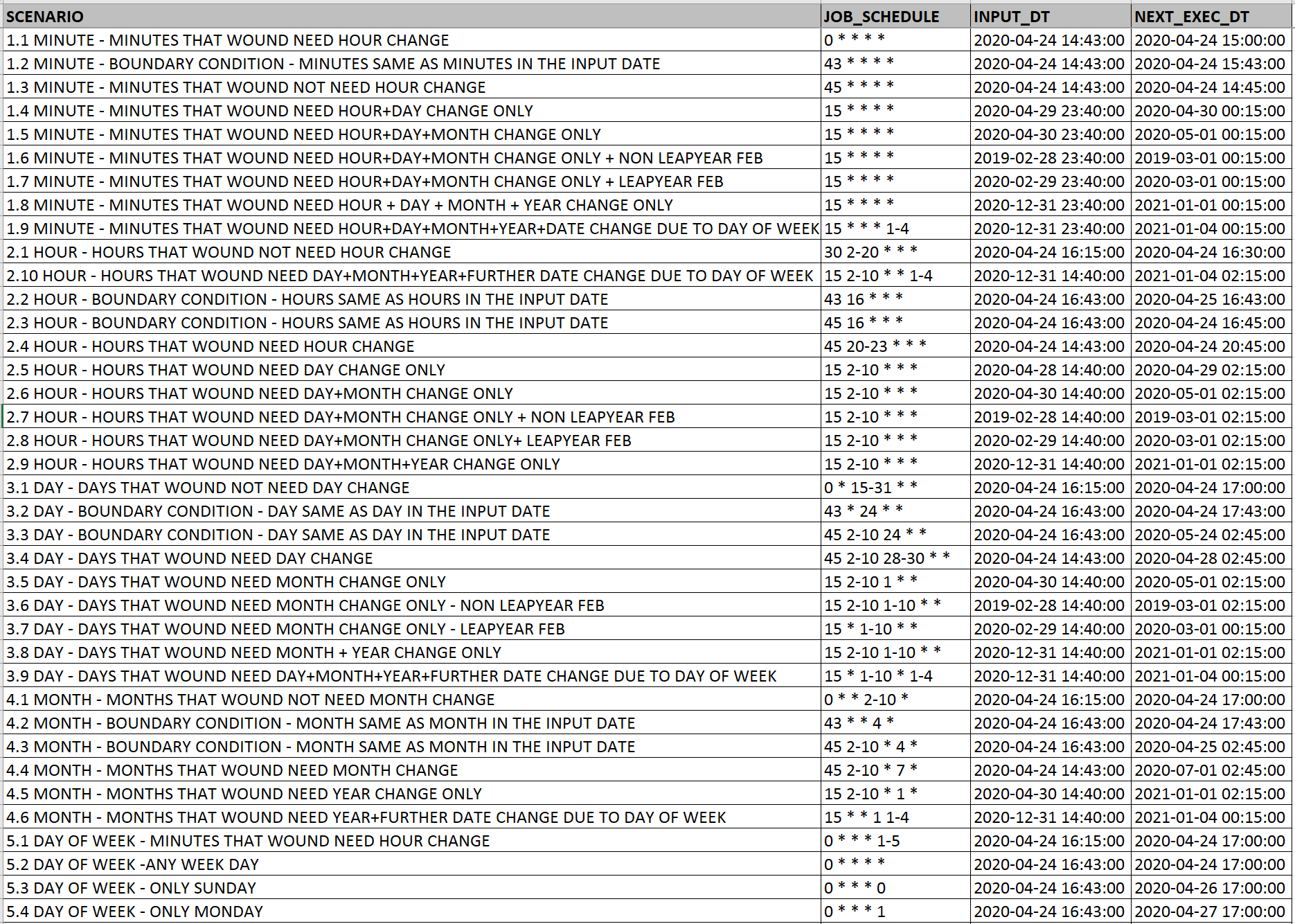
|  |  |  |
| --- | --- | --- |
| **Status Parameter** | **Scenario for the input job** | **Actions performed by procedure** |
| ENABLE |  | ETL\_JOB\_CONFIG.JOB\_ENABLED = Y |
| DISABLE |  | ETL\_JOB\_CONFIG.JOB\_ENABLED = N |
| COMPLETED | 1) ADHOC 2) INDIVIDUAL 3) PARENT with NO active CHILD jobs | ETL\_JOB\_RUN.RUN\_STATUS = COMPLETED (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = COMPLETED ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y |
| 1) PARENT and has active CHILD jobs 2) CHILD and has active CHILD jobs | ETL\_JOB\_RUN.RUN\_STATUS = WAITING (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = WAITING ETL\_JOB\_STATUS.JOB\_RUNNABLE = N |
| CHILD with NO active CHILD jobs | ETL\_JOB\_RUN.RUN\_STATUS = COMPLETED (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = COMPLETED ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y  Calls UPD\_ETL\_JOB procedure for the job's PARENT\_JOB\_ID with status as COMPLETED |
| RESET | Only INDIVIDUAL AND PARENT jobs can be RESET. It will set the job to RUNNABLE with JOB\_NEXT\_EXEC\_DT = SYSDATE, so that it will be run in the next ETL Job Control Run | For the job as well as all the child jobs that are running currently, ETL\_JOB\_RUN.RUN\_STATUS = RESET (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = RESET ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y  For the job, ETL\_JOB\_STATUS.JOB\_NEXT\_EXEC\_DT = SYSDATE |
| CANCELLED | Only INDIVIDUAL AND PARENT jobs can be CANCELLED. Unlike RESET, process will not update the JOB\_NEXT\_EXEC\_DT and it will let the job executed next based on the JOB\_NEXT\_EXEC\_DT already calculated | For the job as well as all the child jobs that are running currently, ETL\_JOB\_RUN.RUN\_STATUS = CANCELLED (for the last RUN\_ID) ETL\_JOB\_RUN.RUN\_END\_DT = SYSDATE (for the last RUN\_ID) ETL\_JOB\_STATUS.JOB\_STATUS = CANCELLED ETL\_JOB\_STATUS.JOB\_RUNNABLE = Y |

## 2.11.4 GET\_NEXT\_JOB\_SCHEDULE function

This function takes an input date and job schedule (in crontab format) and returns the next execution date/time with respect to the input date. This function will internally call the below sub-functions:

* IS\_NUMBER\_VALID
* PARSE\_NUMBER\_RANGE
* GET\_NEXT\_NUMBER
* GET\_MIN\_NUMBER

Here are some examples on how this function will return the next execution date/time based on the input date and job schedule:



## 2.12 etl\_job\_control.sh

This shell script controls all the jobs. This script will be scheduled to run every 15 minutes in CRONTAB.

When this script gets executed it performs the following steps:

* Sets the necessary environmental variables by executing .set\_env.sh
* Extracts the database connection details from kettle.properties file. The path for this file will be again read from .set\_env.sh
* Executes set\_etl\_jobs.sql, which populates list of jobs to be executed and list of jobs that got stuck, into ETL\_JOB\_LIST table
* Executes get\_etl\_jobs.sql to extracts the list of jobs to be started, from ETL\_JOB\_LIST table
* Runs etl\_job\_exec.sh script for each job in the background, by passing JOB\_ID and JOB\_SCRIPT\_NAME for each job
* Executes get\_etl\_stuck\_jobs.sql to extracts the list of jobs stuck, from ETL\_JOB\_LIST table
* Writes the list of stuck jobs, instruction on how to reset them, as well as any other error messages into log file

## 2.13 etl\_job\_exec.sh

This script will be executed in the background for each Job that need to be run by etl\_job\_control.sh. This script will receive the JOB\_ID and JOB\_SCRIPT\_NAME as the parameters, and performs the below steps:

* a new run instance of the job with status as STARTED will be created by executing add\_etl\_job.sql
* will check if JOB\_SCRIPT\_NAME is found in one of the SCRIPT\_PATHs (up to 3 SCRIPT\_PATHs are being planned to be added to .set\_env.sh) and executes the script if found.
* run and job status will be updated as COMPLETED by executing upd\_etl\_job.sql
* writes any errors into log file

## 2.14 .set\_env.sh

This will set all the necessary environment variables including,

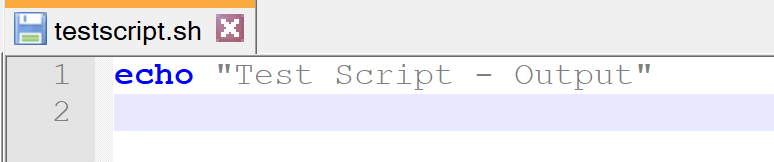
* path for the log file
* path for the kettle.properties file
* Up to 3 paths for the script files. This will help executing scripts for more than once project within the same environment. Process will try to locate JOB\_SCRIPT\_NAME in one the 3 paths and will execute wherever it finds it first. If it is unable to locate in all 3 paths, job will get error out,

## 2.15 kettle.properties file

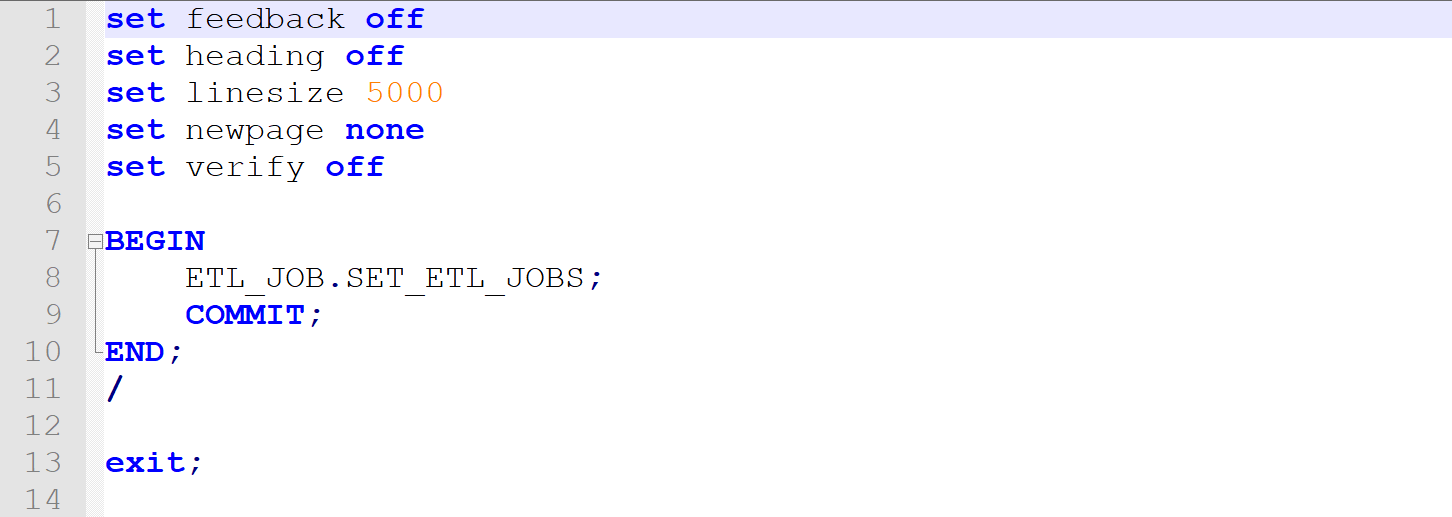
This is standard kettle.properties file that the process will read to extract the database connection details. Path for this file will be configured in the .set\_env.sh

## 2.16 testscript.sh

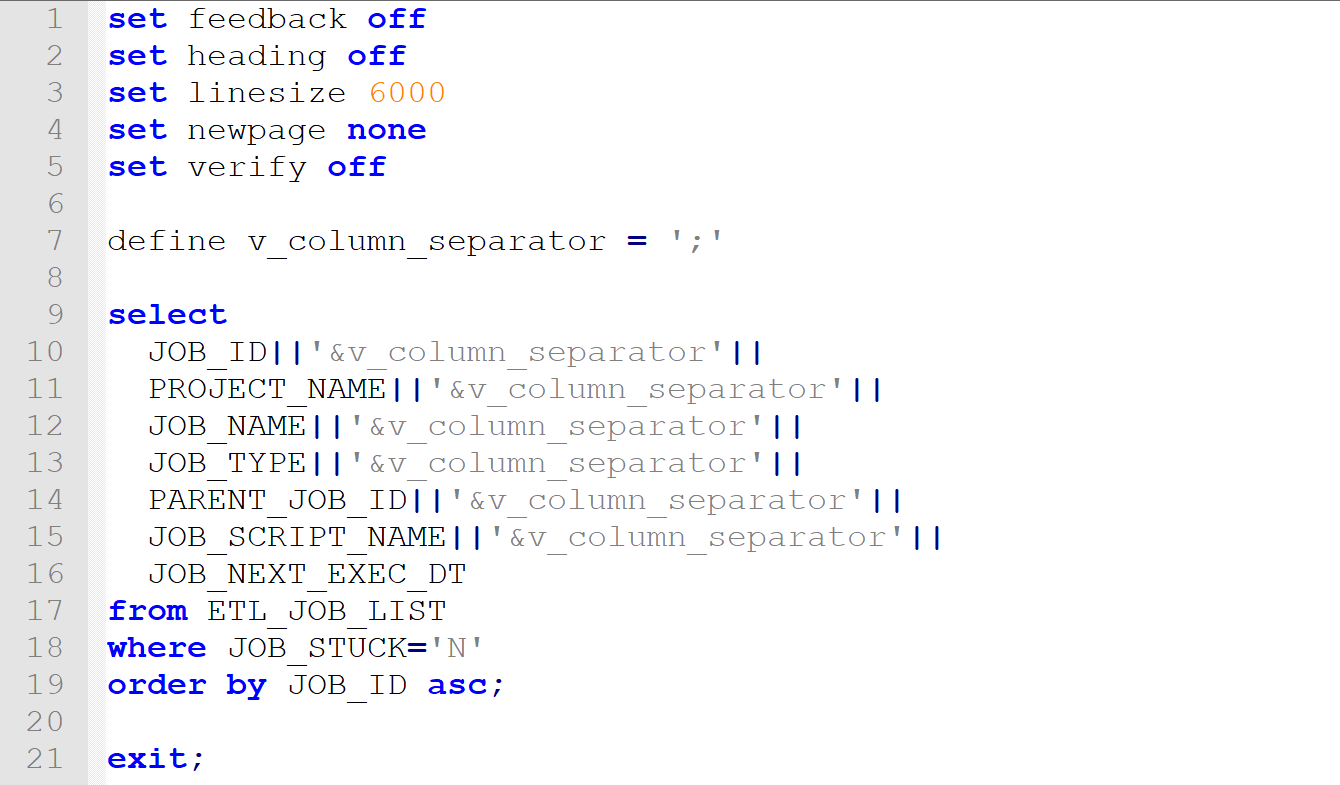
This script is only for prototype testing.



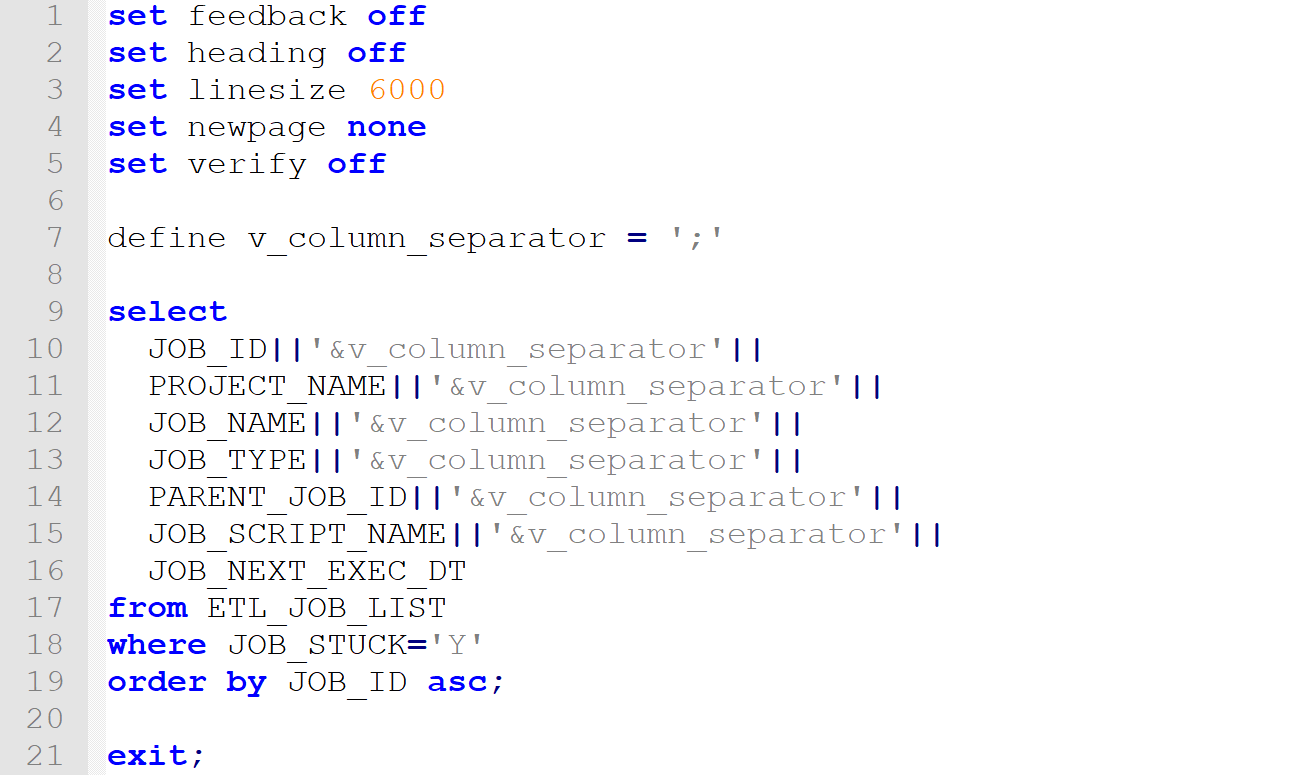
## 2.17 set\_etl\_jobs.sql



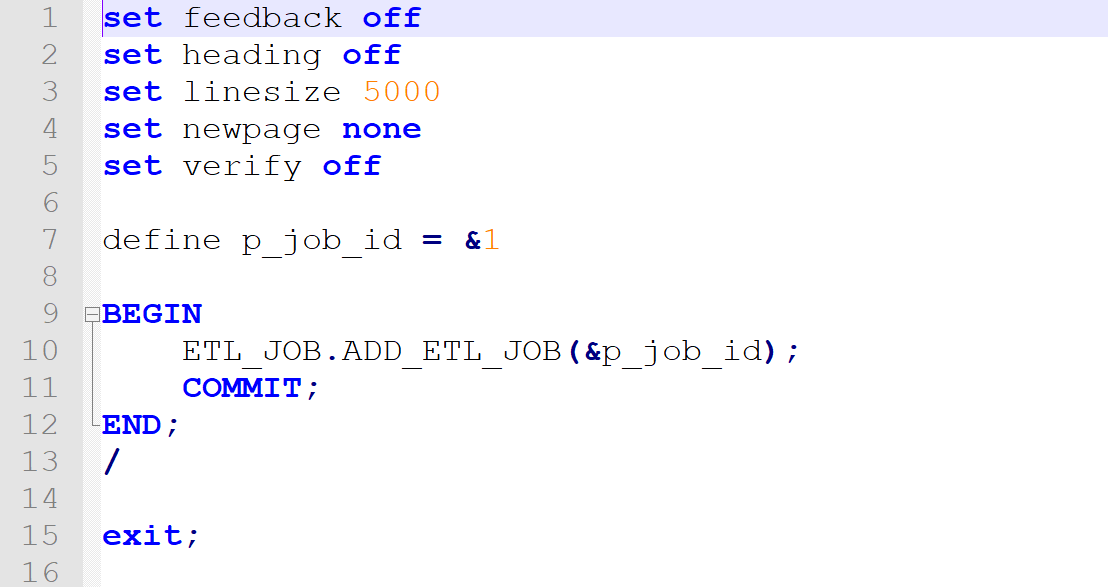
## 2.18 get\_etl\_jobs.sql



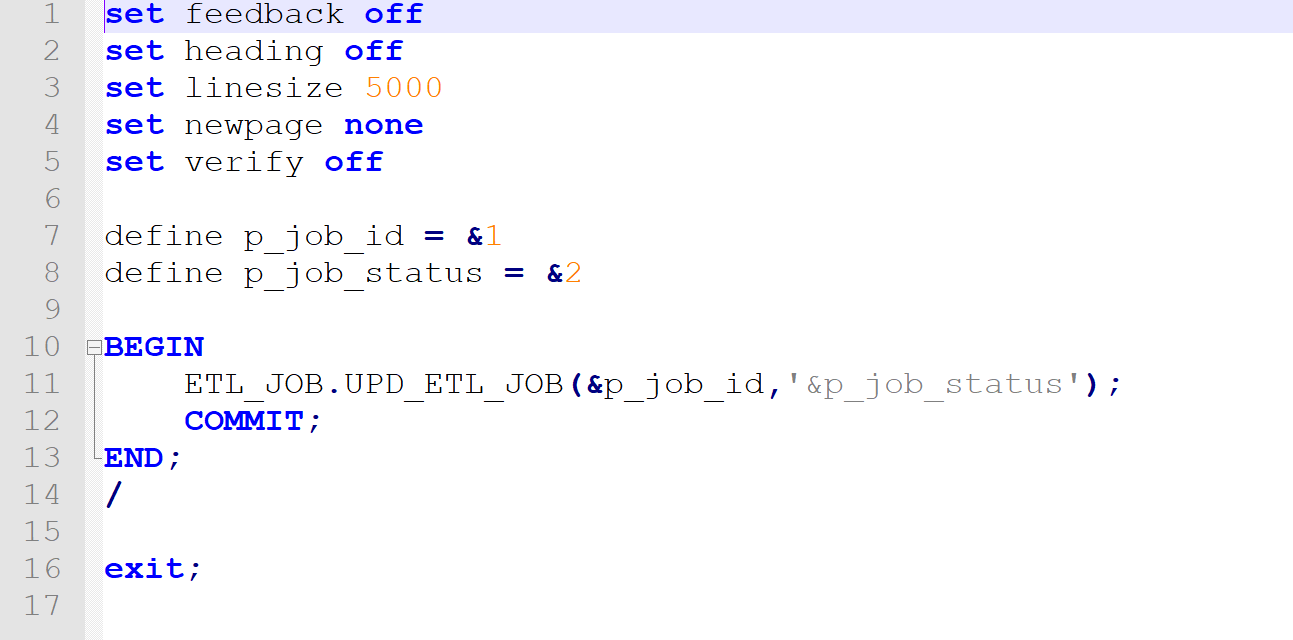
## 2.19 get\_etl\_stuck\_jobs.sql



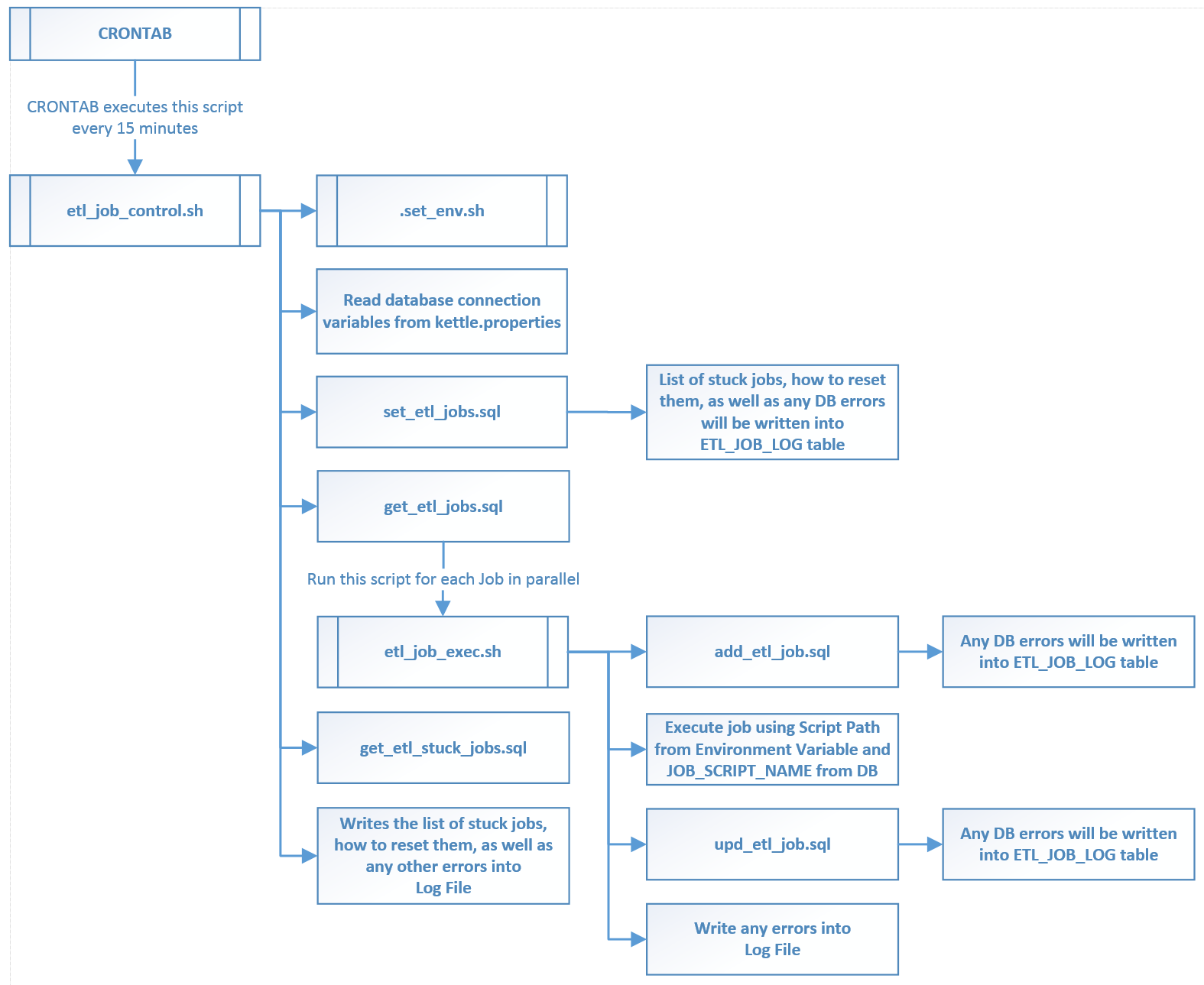
## 2.20 add\_etl\_job.sql



## 2.21 upd\_etl\_job.sql



# 3 ETL Job Control - Process Flow



# 4 Process Flow – Verification (NOT UPDATED)

## 4.1 Create all DB Objects

## 4.2 Load a job into BPM\_JOB\_CONFIG table

## 4.3 Execute etl\_job\_control.sh

## 4.4. Verify the job list extracted

## 4.5 Check new run instance of the job is created

## 4.6 Check the status of Job and run instance

## 4.7 Make sure JOB\_NEXT\_EXEC\_DTM is calculated and updated

## 4.8 Verify the job is successfully executed

## 4.9 Check the status of Job and run instance after job execution

## 4.10 Make some changes to ETL\_JOB\_CONFIG and verify audit table

## 4.11 Verify log files as well as ETL\_JOB\_LOG table