```
####################  Ora2Pg Configuration file   #####################

# Support for including a common config file that may contain any
# of the following configuration directives.
#IMPORT common.conf

#------------------------------------------------------------------------------
# INPUT SECTION (Oracle connection or input file)
#------------------------------------------------------------------------------

# Set this directive to a file containing PL/SQL Oracle Code like function,
# procedure or a full package body to prevent Ora2Pg from connecting to an
# Oracle database end just apply his conversion tool to the content of the
# file. This can only be used with the following export type: PROCEDURE,
# FUNCTION or PACKAGE. If you don't know what you do don't use this directive.
#INPUT_FILE     ora_plsql_src.sql

# Set the Oracle home directory
# ORACLE_HOME   /usr/local/oracle/10g
ORACLE_HOME /u01/app/oracle/product/12.1/dbhome_1

# Set Oracle database connection (datasource, user, password)
ORACLE_DSN      dbi:Oracle:host=localhost;sid=xe;port=1521
ORACLE_USER     scott
ORACLE_PWD      tiger

# Set this to 1 if you connect as simple user and can not extract things
# from the DBA_... tables. It will use tables ALL_... This will not works
# with GRANT export, you should use an Oracle DBA username at ORACLE_USER
USER_GRANTS     0

# Trace all to stderr
DEBUG           0

# This directive can be used to send an initial command to Oracle, just after
# the connection. For example to unlock a policy before reading objects or
# to set some session parameters. This directive can be used multiple time.
#ORA_INITIAL_COMMAND


#------------------------------------------------------------------------------
# SCHEMA SECTION (Oracle schema to export and use of schema in PostgreSQL)
#------------------------------------------------------------------------------

# Export Oracle schema to PostgreSQL schema
EXPORT_SCHEMA   1

# Oracle schema/owner to use
SCHEMA          SCOTT
```

```
# Enable/disable the CREATE SCHEMA SQL order at starting of the output file.
# It is enable by default and concern on TABLE export type.
CREATE_SCHEMA   1

# Enable this directive to force Oracle to compile schema before exporting code.
# When this directive is enabled and SCHEMA is set to a specific schema name,
# only invalid objects in this schema will be recompiled. If SCHEMA is not set
# then all schema will be recompiled. To force recompile invalid object in a
# specific schema, set COMPILE_SCHEMA to the schema name you want to recompile.
# This will ask to Oracle to validate the PL/SQL that could have been invalidate
# after a export/import for example. The 'VALID' or 'INVALID' status applies to
# functions, procedures, packages and user defined types.
COMPILE_SCHEMA  0

# By default if you set EXPORT_SCHEMA to 1 the PostgreSQL search_path will be
# set to the schema name exported set as value of the SCHEMA directive. You can
# defined/force the PostgreSQL schema to use by using this directive.
#
# The value can be a comma delimited list of schema but not when using TABLE
# export type because in this case it will generate the CREATE SCHEMA statement
# and it doesn't support multiple schema name. For example, if you set PG_SCHEMA
# to something like "user_schema, public", the search path will be set like this
#        SET search_path = user_schema, public;
# forcing the use of an other schema (here user_schema) than the one from Oracle
# schema set in the SCHEMA directive. You can also set the default search_path
# for the PostgreSQL user you are using to connect to the destination database
# by using:
#        ALTER ROLE username SET search_path TO user_schema, public;
#in this case you don't have to set PG_SCHEMA.
#PG_SCHEMA

# Use this directive to add a specific schema to the search path to look
# for PostGis functions.
#POSTGIS_SCHEMA

# Allow to add a comma separated list of system user to exclude from
# Oracle extraction. Oracle have many of them following the modules
# installed. By default it will suppress all object owned by the following
# system users:
#        CTXSYS,DBSNMP,EXFSYS,LBACSYS,MDSYS,MGMT_VIEW,OLAPSYS,ORDDATA,OWBSYS,
#        ORDPLUGINS,ORDSYS,OUTLN,SI_INFORMTN_SCHEMA,SYS,SYSMAN,SYSTEM,WK_TEST,
#        WKSYS,WKPROXY,WMSYS,XDB,APEX_PUBLIC_USER,DIP,FLOWS_020100,FLOWS_030000,
#
FLOWS_040100,FLOWS_010600,FLOWS_FILES,MDDATA,ORACLE_OCM,SPATIAL_CSW_ADMIN_USR,
#        SPATIAL_WFS_ADMIN_USR,XS$NULL,PERFSTAT,SQLTXPLAIN,DMSYS,TSMSYS,WKSYS,
#        APEX_040200,DVSYS,OJVMSYS,GSMADMIN_INTERNAL,APPQOSSYS
# Other list of users set to this directive will be added to this list.
#SYSUSERS       OE,HR
```

```
# List of schema to get functions/procedures meta information that are used
# in the current schema export. When replacing call to function with OUT
# parameters, if a function is declared in an other package then the function
# call rewriting can not be done because Ora2Pg only know about functions
# declared in the current schema. By setting a comma separated list of schema
# as value of this directive, Ora2Pg will look forward in these packages for
# all functions/procedures/packages declaration before proceeding to current
# schema export.
#LOOK_FORWARD_FUNCTION  SCOTT,OE


#------------------------------------------------------------------------------
# ENCODING SECTION (Define client encoding at Oracle and PostgreSQL side)
#------------------------------------------------------------------------------

# Enforce default language setting following the Oracle database encoding. This
# may be used with multibyte characters like UTF8. Here are the default values
# used by Ora2Pg, you may not change them unless you have problem with this
# encoding. This will set $ENV{NLS_LANG} to the given value.
#NLS_LANG       AMERICAN_AMERICA.AL32UTF8
# This will set $ENV{NLS_NCHAR} to the given value.
#NLS_NCHAR      AL32UTF8

# By default PostgreSQL client encoding is automatically set to UTF8 to avoid
# encoding issue. If you have changed the value of NLS_LANG you might have to
# change  the encoding of the PostgreSQL client.
#CLIENT_ENCODING        UTF8


#------------------------------------------------------------------------------
# EXPORT SECTION (Export type and filters)
#------------------------------------------------------------------------------

# Type of export. Values can be the following keyword:
#       TABLE           Export tables, constraints, indexes, ...
#       PACKAGE         Export packages
#       INSERT          Export data from table as INSERT statement
#       COPY            Export data from table as COPY statement
#       VIEW            Export views
#       GRANT           Export grants
#       SEQUENCE        Export sequences
#       TRIGGER         Export triggers
#       FUNCTION        Export functions
#       PROCEDURE       Export procedures
#       TABLESPACE      Export tablespace (PostgreSQL >= 8 only)
#       TYPE            Export user defined Oracle types
#       PARTITION       Export range or list partition (PostgreSQL >= v8.4)
#       FDW             Export table as foreign data wrapper tables
#       MVIEW           Export materialized view as snapshot refresh view
#       QUERY           Convert Oracle SQL queries from a file.
#       KETTLE          Generate XML ktr template files to be used by Kettle.
#       DBLINK          Generate oracle foreign data wrapper server to use as 
dblink.
#       SYNONYM         Export Oracle's synonyms as views on other schema's objects.
#       DIRECTORY       Export Oracle's directories as external_file extension 
objects.
#       LOAD            Dispatch a list of queries over multiple PostgreSQl 
connections.
#       TEST            perform a diff between Oracle and PostgreSQL database.
#       TEST_VIEW       perform a count on both side of rows returned by views

TYPE   TABLE PACKAGE COPY VIEW GRANT SEQUENCE TRIGGER FUNCTION PROCEDURE TABLESPACE 
TYPE PARTITION

# Set this to 1 if you don't want to export comments associated to tables and
# column definitions. Default is enabled.
DISABLE_COMMENT         0

# Set which object to export from. By default Ora2Pg export all objects.
# Value must be a list of object name or regex separated by space. Note
# that regex will not works with 8i database, use % placeholder instead
# Ora2Pg will use the LIKE operator. There is also some extended use of
# this directive, see chapter "Limiting object to export" in documentation.
#ALLOW          TABLE_TEST

# Set which object to exclude from export process. By default none. Value
# must be a list of object name or regexp separated by space. Note that regex
# will not works with 8i database, use % placeholder instead Ora2Pg will use
# the NOT LIKE operator. There is also some extended use of this directive,
# see chapter "Limiting object to export" in documentation.
#EXCLUDE        OTHER_TABLES

# Set which view to export as table. By default none. Value must be a list of
# view name or regexp separated by space. If the object name is a view and the
# export type is TABLE, the view will be exported as a create table statement.
# If export type is COPY or INSERT, the corresponding data will be exported.
#VIEW_AS_TABLE  VIEW_NAME

# When exporting GRANTs you can specify a comma separated  list of objects
# for which privilege will be exported. Default is export for all objects.
# Here are the possibles values TABLE, VIEW, MATERIALIZED VIEW, SEQUENCE,
# PROCEDURE, FUNCTION, PACKAGE BODY, TYPE, SYNONYM, DIRECTORY. Only one object
# type is allowed at a time. For example set it to TABLE if you just want to
# export privilege on tables. You can use the -g option to overwrite it.
# When used this directive prevent the export of users unless it is set to
# USER. In this case only users definitions are exported.
#GRANT_OBJECT    TABLE

# By default Ora2Pg will export your external table as file_fdw tables. If
# you don't want to export those tables at all, set the directive to 0.
EXTERNAL_TO_FDW         0

# Add a TRUNCATE TABLE instruction before loading data on COPY and INSERT
# export. When activated, the instruction will be added only if there's no
# global DELETE clause or one specific to the current table (see bellow).
TRUNCATE_TABLE  0

# Support for include a DELETE FROM ... WHERE clause filter before importing
# data and perform a delete of some lines instead of truncatinf tables.
# Value is construct as follow: TABLE_NAME[DELETE_WHERE_CLAUSE], or
# if you have only one where clause for all tables just put the delete
# clause as single value. Both are possible too. Here are some examples:
#DELETE  1=1    # Apply to all tables and delete all tuples
#DELETE TABLE_TEST[ID1='001']   # Apply only on table TABLE_TEST
#DELETE TABLE_TEST[ID1='001' OR ID1='002] DATE_CREATE > '2001-01-01'
TABLE_INFO[NAME='test']
# The last applies two different delete where clause on tables TABLE_TEST and
# TABLE_INFO and a generic delete where clause on DATE_CREATE to all other tables.
# If TRUNCATE_TABLE is enabled it will be applied to all tables not covered by
# the DELETE definition.

# When enabled this directive forces ora2pg to export all tables, index
# constraints, and indexes using the tablespace name defined in Oracle database.
# This works only with tablespaces that are not TEMP, USERS and SYSTEM.
USE_TABLESPACE          0

# Enable this directive to reorder columns and minimized the footprint
# on disk, so that more rows fit on a data page, which is the most important
# factor for speed. Default is same order than in Oracle table definition,
# that should be enough for most usage.
REORDERING_COLUMNS      0

# Support for include a WHERE clause filter when dumping the contents
# of tables. Value is construct as follow: TABLE_NAME[WHERE_CLAUSE], or
# if you have only one where clause for each table just put the where
# clause as value. Both are possible too. Here are some examples:
#WHERE  1=1     # Apply to all tables
#WHERE  TABLE_TEST[ID1='001']   # Apply only on table TABLE_TEST
#WHERE  TABLE_TEST[ID1='001' OR ID1='002] DATE_CREATE > '2001-01-01'
TABLE_INFO[NAME='test']
# The last applies two different where clause on tables TABLE_TEST and
# TABLE_INFO and a generic where clause on DATE_CREATE to all other tables

# Sometime you may want to extract data from an Oracle table but you need a
# a custom query for that. Not just a "SELECT * FROM table" like Ora2Pg does
# but a more complex query. This directive allows you to override the query
# used by Ora2Pg to extract data. The format is TABLENAME[SQL_QUERY].
# If you have multiple tables to extract by replacing the Ora2Pg query, you can
# define multiple REPLACE_QUERY lines.
```

```
#REPLACE_QUERY  EMPLOYEES[SELECT e.id,e.fisrtname,lastname FROM EMPLOYEES e JOIN
EMP_UPDT u ON (e.id=u.id AND u.cdate>'2014-08-01 00:00:00')]

#------------------------------------------------------------------------------
# FULL TEXT SEARCH SECTION (Control full text search export behaviors)
#------------------------------------------------------------------------------

# Force Ora2Pg to translate Oracle Text indexes into PostgreSQL indexes using
# pg_trgm extension. Default is to translate CONTEXT indexes into FTS indexes
# and CTXCAT indexes using pg_trgm. Most of the time using pg_trgm is enough,
# this is why this directive stand for.
#
CONTEXT_AS_TRGM         0

# By default Ora2Pg creates a function-based index to translate Oracle Text
# indexes.
#    CREATE INDEX ON t_document
#                 USING gin(to_tsvector('french', title));
# You will have to rewrite the CONTAIN() clause using to_tsvector(), example:
#    SELECT id,title FROM t_document
#                    WHERE to_tsvector(title)) @@ to_tsquery('search_word');
#
# To force Ora2Pg to create an extra tsvector column with a dedicated triggers
# for FTS indexes, disable this directive. In this case, Ora2Pg will add the
# column as follow: ALTER TABLE t_document ADD COLUMN tsv_title tsvector;
# Then update the column to compute FTS vectors if data have been loaded before
#     UPDATE t_document SET tsv_title =
#                       to_tsvector('french', coalesce(title,''));
# To automatically update the column when a modification in the title column
# appears, Ora2Pg adds the following trigger:
#
# CREATE FUNCTION tsv_t_document_title() RETURNS trigger AS $$
# BEGIN
#        IF TG_OP = 'INSERT' OR new.title != old.title THEN
#                new.tsv_title :=
#                to_tsvector('french', coalesce(new.title,''));
#        END IF;
#        return new;
# END
# $$ LANGUAGE plpgsql;
# CREATE TRIGGER trig_tsv_t_document_title BEFORE INSERT OR UPDATE
#  ON t_document
#  FOR EACH ROW EXECUTE PROCEDURE tsv_t_document_title();
#
# When the Oracle text index is defined over multiple column, Ora2Pg will use
# setweight() to set a weight in the order of the column declaration.
#
FTS_INDEX_ONLY  1

# Use this directive to force text search configuration to use. When it is not
# set, Ora2Pg will autodetect the stemmer used by Oracle for each index and
# pg_catalog.english if nothing is found.
#
#FTS_CONFIG     pg_catalog.french

# If you want to perform your text search in an accent insensitive way, enable
# this directive. Ora2Pg will create an helper function over unaccent() and
# creates the pg_trgm indexes using this function. With FTS Ora2Pg will
# redefine your text search configuration, for example:
#
#       CREATE TEXT SEARCH CONFIGURATION fr (COPY = pg_catalog.french);
#       ALTER TEXT SEARCH CONFIGURATION fr
#               ALTER MAPPING FOR hword, hword_part, word WITH unaccent,
french_stem;
#
# When enabled, Ora2pg will create the wrapper function:
#
#       CREATE OR REPLACE FUNCTION unaccent_immutable(text)
#       RETURNS text AS
#       $$
#               SELECT public.unaccent('public.unaccent', )
#       $$  LANGUAGE sql IMMUTABLE
#           COST 1;
#
# indexes are exported as follow:
#
#       CREATE INDEX t_document_title_unaccent_trgm_idx ON t_document
#               USING gin (unaccent_immutable(title) gin_trgm_ops);
#
# In your queries you will need to use the same function in the search to
# be able to use the function-based index. Example:
#
#       SELECT * FROM t_document
#               WHERE unaccent_immutable(title) LIKE '%donnees%';
#
USE_UNACCENT            0

# Same as above but call lower() in the unaccent_immutable() function:
#
#      CREATE OR REPLACE FUNCTION unaccent_immutable(text)
#      RETURNS text AS
#      $$
#          SELECT lower(public.unaccent('public.unaccent', ));
#      $$ LANGUAGE sql IMMUTABLE;
#
USE_LOWER_UNACCENT      0


#------------------------------------------------------------------------------
# DATA DIFF SECTION (only delete and insert actually changed rows)
```

```
#------------------------------------------------------------------------------

# EXPERIMENTAL! Not yet working correctly with partitioned tables, parallelism,
# and direct Postgres connection! Test before using in production!
# This feature affects SQL output for data (INSERT or COPY).
# The deletion and (re-)importing of data is redirected to temporary tables
# (with configurable suffix) and matching entries (i.e. quasi-unchanged rows)
# eliminated before actual application of the DELETE, UPDATE and INSERT.
# Optional functions can be specified that are called before or after the
# actual DELETE, UPDATE and INSERT per table, or after all tables have been
# processed.
#
# Enable DATADIFF functionality
DATADIFF        0
# Use UPDATE where changed columns can be matched by the primary key
# (otherwise rows are DELETEd and re-INSERTed, which may interfere with
# inverse foreign keys relationships!)
DATADIFF_UPDATE_BY_PKEY 0
# Suffix for temporary tables holding rows to be deleted and to be inserted.
# Pay attention to your tables names:
# 1) There better be no two tables with names such that name1 + suffix = name2
# 2) length(suffix) + length(tablename) < NAMEDATALEN (usually 64)
DATADIFF_DEL_SUFFIX _del
DATADIFF_UPD_SUFFIX _upd
DATADIFF_INS_SUFFIX _ins
# Allow setting the work_mem and temp_buffers parameters
# to keep temp tables in memory and have efficient sorting, etc.
DATADIFF_WORK_MEM       256 MB
DATADIFF_TEMP_BUFFERS   512 MB

# The following are names of functions that will be called (via SELECT)
# after the temporary tables have been reduced (by removing matching rows)
# and right before or right after the actual DELETE and INSERT are performed.
# They must take four arguments, which should ideally be of type "regclass",
# representing the real table, the "deletions", the "updates", and the
# "insertions" temp table names, respectively. They are called before
# re-activation of triggers, indexes, etc. (if configured).
#DATADIFF_BEFORE   my_datadiff_handler_function
#DATADIFF_AFTER    my_datadiff_handler_function

# Another function can be called (via SELECT) right before the entire COMMIT
# (i.e., after re-activation of indexes, triggers, etc.), which will be
# passed in Postgres ARRAYs of the table names of the real tables, the
# "deletions", the "updates" and the "insertions" temp tables, respectively,
# with same array index positions belonging together. So this function should
# take four arguments of type regclass[]
#DATADIFF_AFTER_ALL    my_datadiff_bunch_handler_function
# If in doubt, use schema-qualified function names here.
# The search_path will have been set to PG_SCHEMA if EXPORT_SCHEMA == 1
# (as defined by you in those config parameters, see above),
# i.e., the "public" schema is not contained if EXPORT_SCHEMA == 1


#------------------------------------------------------------------------------
# CONSTRAINT SECTION (Control constraints export and import behaviors)
#------------------------------------------------------------------------------

# Support for turning off certain schema features in the postgres side
# during schema export. Values can be : fkeys, pkeys, ukeys, indexes, checks
# separated by a space character.
# fkeys         : turn off foreign key constraints
# pkeys         : turn off primary keys
# ukeys         : turn off unique column constraints
# indexes       : turn off all other index types
# checks        : turn off check constraints
#SKIP   fkeys pkeys ukeys indexes checks

# By default names of the primary and unique key in the source Oracle database
# are ignored and key names are autogenerated in the target PostgreSQL database
# with the PostgreSQL internal default naming rules. If you want to preserve
# Oracle primary and unique key names set this option to 1.
# Please note if value of USE_TABLESPACE is set to 1 the value of this option is
# enforced to 1 to preserve correct primary and uniqie key allocation to tablespace.
KEEP_PKEY_NAMES         0

# Enable this directive if you want to add primary key definitions inside the
# create table statements. If disabled (the default) primary key definition
# will be added with an alter table statement. Enable it if you are exporting
# to GreenPlum PostgreSQL database.
PKEY_IN_CREATE          0

# This directive allow you to add an ON UPDATE CASCADE option to a foreign
# key when a ON DELETE CASCADE is defined or always. Oracle do not support
# this feature, you have to use trigger to operate the ON UPDATE CASCADE.
# As PostgreSQL has this feature, you can choose how to add the foreign
# key option. There is three value to this directive: never, the default
# that mean that foreign keys will be declared exactly like in Oracle.
# The second value is delete, that mean that the ON UPDATE CASCADE option
# will be added only if the ON DELETE CASCADE is already defined on the
# foreign Keys. The last value, always, will force all foreign keys to be
# defined using the update option.
FKEY_ADD_UPDATE         never

# When exporting tables, Ora2Pg normally exports constraints as they are;
# if they are non-deferrable they are exported as non-deferrable.
# However, non-deferrable constraints will probably cause problems when
# attempting to import data to PostgreSQL. The following option set to 1
# will cause all foreign key constraints to be exported as deferrable
FKEY_DEFERRABLE 0
```

```
# In addition when exporting data the DEFER_FKEY option set to 1 will add
# a command to defer all foreign key constraints during data export and
# the import will be done in a single transaction. This will work only if
# foreign keys have been exported as deferrable and you are not using direct
# import to PostgreSQL (PG_DSN is not defined). Constraints will then be
# checked at the end of the transaction. This directive can also be enabled
# if you want to force all foreign keys to be created as deferrable and
# initially deferred during schema export (TABLE export type).
DEFER_FKEY      1

# If deferring foreign keys is not possible du to the amount of data in a
# single transaction, you've not exported foreign keys as deferrable or you
# are using direct import to PostgreSQL, you can use the DROP_FKEY directive.
# It will drop all foreign keys before all data import and recreate them at
# the end of the import.
DROP_FKEY       0


#------------------------------------------------------------------------------
# TRIGGERS AND SEQUENCES SECTION (Control triggers and sequences behaviors)
#------------------------------------------------------------------------------

# Disables alter of sequences on all tables in COPY or INSERT mode.
# Set to 1 if you want to disable update of sequence during data migration.
DISABLE_SEQUENCE        0

# Disables triggers on all tables in COPY or INSERT mode. Available modes
# are USER (user defined triggers) and ALL (includes RI system
# triggers). Default is 0 do not add SQL statement to disable trigger.
# If you want to disable triggers during data migration, set the value to
# USER if your are connected as non superuser and ALL if you are connected
# as PostgreSQL superuser. A value of 1 is equal to USER.
DISABLE_TRIGGERS 0


#------------------------------------------------------------------------------
# OBJECT MODIFICATION SECTION (Control objects structure or name modifications)
#------------------------------------------------------------------------------

# You may wish to just extract data from some fields, the following directives
# will help you to do that. Works only with export type INSERT or COPY
# Modify output from the following tables(fields separate by space or comma)
#MODIFY_STRUCT  TABLE_TEST(dico,dossier)

# You may wish to change table names during data extraction, especally for
# replication use. Give a list of tables separate by space as follow.
#REPLACE_TABLES ORIG_TB_NAME1:NEW_TB_NAME1 ORIG_TB_NAME2:NEW_TB_NAME2

# You may wish to change column names during export. Give a list of tables
# and columns separate by space as follow.
```

```
#REPLACE_COLS   TB_NAME(ORIG_COLNAME1:NEW_COLNAME1,ORIG_COLNAME2:NEW_COLNAME2)

# By default all object names are converted to lower case, if you
# want to preserve Oracle object name as-is set this to 1. Not recommended
# unless you always quote all tables and columns on all your scripts.
PRESERVE_CASE   0

# Add the given value as suffix to index names. Useful if you have indexes
# with same name as tables. Not so common but it can help.
#INDEXES_SUFFIX         _idx

# Enable this directive to rename all indexes using tablename_columns_names.
# Could be very useful for database that have multiple time the same index name
# or that use the same name than a table, which is not allowed by PostgreSQL
# Disabled by default.
INDEXES_RENAMING        0

# Operator classes text_pattern_ops, varchar_pattern_ops, and bpchar_pattern_ops
# support B-tree indexes on the corresponding types. The difference from the
# default operator classes is that the values are compared strictly character by
# character rather than according to the locale-specific collation rules. This
# makes these operator classes suitable for use by queries involving pattern
# matching expressions (LIKE or POSIX regular expressions) when the database
# does not use the standard "C" locale. If you enable, with value 1, this will
# force Ora2Pg to export all indexes defined on varchar2() and char() columns
# using those operators. If you set it to a value greater than 1 it will only
# change indexes on columns where the charactere limit is greater or equal than
# this value. For example, set it to 128 to create these kind of indexes on
# columns of type varchar2(N) where N >= 128.
USE_INDEX_OPCLASS       0

# Enable this directive if you want that your partition table name will be
# exported using the parent table name. Disabled by default. If you have
# multiple partitioned table, when exported to PostgreSQL some partitions
# could have the same name but different parent tables. This is not allowed,
# table name must be unique.
PREFIX_PARTITION        0

# Disable this directive if your subpartitions are dedicated to your partition
# (in case of your partition_name is a part of your subpartition_name)
PREFIX_SUB_PARTITION    1

# If you don't want to reproduce the partitioning like in Oracle and want to
# export all partitionned Oracle data into the main single table in PostgreSQL
# enable this directive. Ora2Pg will export all data into the main table name.
# Default is to use partitionning, Ora2Pg will export data from each partition
# and import them into the PostgreSQL dedicated partition table.
DISABLE_PARTITION       0

# Activating this directive will force Ora2Pg to add WITH (OIDS) when creating
# tables or views as tables. Default is same as PostgreSQL, disabled.
WITH_OID                0

# Allow escaping of column name using Oracle reserved words.
ORA_RESERVED_WORDS      audit,comment,references

# Enable this directive if you have tables or column names that are a reserved
# word for PostgreSQL. Ora2Pg will double quote the name of the object.
USE_RESERVED_WORDS      0

# By default Ora2Pg export Oracle tables with the NOLOGGING attribute as
# UNLOGGED tables. You may want to fully disable this feature because
# you will lost all data from unlogged table in case of PostgreSQL crash.
# Set it to 1 to export all tables as normal table.
DISABLE_UNLOGGED        0

#------------------------------------------------------------------------------
# OUTPUT SECTION (Control output to file or PostgreSQL database)
#------------------------------------------------------------------------------

# Define the following directive to send export directly to a PostgreSQL
# database. This will disable file output.
#PG_DSN         dbi:Pg:dbname=test_db;host=localhost;port=5432
#PG_USER        test
#PG_PWD         test

# By default all output is dump to STDOUT if not send directly to postgresql
# database (see above). Give a filename to save export to it. If you want
# a Gzip'd compressed file just add the extension .gz to the filename (you
# need perl module Compress::Zlib from CPAN). Add extension .bz2 to use Bzip2
# compression.
OUTPUT          output.sql

# Base directory where all dumped files must be written
#OUTPUT_DIR     /var/tmp

# Path to the bzip2 program. See OUTPUT directive above.
BZIP2

# Allow object constraints to be saved in a separate file during schema export.
# The file will be named CONSTRAINTS_OUTPUT. Where OUTPUT is the value of the
# corresponding configuration directive. You can use .gz xor .bz2 extension to
# enable compression. Default is to save all data in the OUTPUT file. This
# directive is usable only with TABLE export type.
FILE_PER_CONSTRAINT     0

# Allow indexes to be saved in a separate file during schema export. The file
# will be named INDEXES_OUTPUT. Where OUTPUT is the value of the corresponding
# configuration directive. You can use the .gz, .xor, or .bz2 file extension to
# enable compression. Default is to save all data in the OUTPUT file. This
# directive is usable only with TABLE or TABLESPACE export type.  With the
# TABLESPACE export, it is used to write "ALTER INDEX ... TABLESPACE ..." into
# a separate file named TBSP_INDEXES_OUTPUT that can be loaded at end of the
# migration after the indexes creation to move the indexes.
FILE_PER_INDEX          0

# Allow foreign key declaration to be saved in a separate file during
# schema export. By default foreign keys are exported into the main
# output file or in the CONSTRAINT_output.sql file. When enabled foreign
# keys will be exported into a file named FKEYS_output.sql
FILE_PER_FKEYS          0

# Allow data export to be saved in one file per table/view. The files
# will be named as tablename_OUTPUT. Where OUTPUT is the value of the
# corresponding configuration directive. You can use .gz xor .bz2
# extension to enable compression. Default is to save all data in one
# file. This is usable only during INSERT or COPY export type.
FILE_PER_TABLE  0

# Allow function export to be saved in one file per function/procedure.
# The files will be named as funcname_OUTPUT. Where OUTPUT is the value
# of the corresponding configuration directive. You can use .gz xor .bz2
# extension to enable compression. Default is to save all data in one
# file. It is usable during FUNCTION, PROCEDURE, TRIGGER and PACKAGE
# export type.
FILE_PER_FUNCTION       0

# By default Ora2Pg will force Perl to use utf8 I/O encoding. This is done through
# a call to the Perl pragma:
#
#    use open ':utf8';
#
# You can override this encoding by using the BINMODE directive, for example you
# can set it to :locale to use your locale or iso-8859-7, it will respectively use
#
#    use open ':locale';
#    use open ':encoding(iso-8859-7)';
#
# If you have change the NLS_LANG in non UTF8 encoding, you might want to set this
# directive. See http://perldoc.perl.org/5.14.2/open.html for more information.
# Most of the time, you might leave this directive commented.
#BINMODE                utf8

# Set it to 0 to not include the call to \set ON_ERROR_STOP ON in all SQL
# scripts. By default this order is always present.
STOP_ON_ERROR           1

# Enable this directive to use COPY FREEZE instead of a simple COPY to
# export data with rows already frozen. This is intended as a performance
# option for initial data loading. Rows will be frozen only if the table
# being loaded has been created or truncated in the current subtransaction.
# This will only works with export to file and when -J or ORACLE_COPIES is
# not set or default to 1. It can be used with direct import into PostgreSQL
# under the same condition but -j or JOBS must also be unset or default to 1.
COPY_FREEZE             0

# By default Ora2Pg use CREATE OR REPLACE in function DDL, if you need not
# to override existing functions disable this configuration directive,
# DDL will not include OR REPLACE.
CREATE_OR_REPLACE       1

# This directive can be used to send an initial command to PostgreSQL, just
# after the connection. For example to set some session parameters. This
# directive can be used multiple time.
#PG_INITIAL_COMMAND



#------------------------------------------------------------------------------
# TYPE SECTION (Control type behaviors and redefinitions)
#------------------------------------------------------------------------------

# If you're experiencing problems in data type export, the following directive
# will help you to redefine data type translation used in Ora2pg. The syntax is
# a comma separated list of "Oracle datatype:Postgresql data type". Here are the
# data type that can be redefined and their default value. If you want to
# replace a type with a precision and scale you need to escape the coma with
# a backslash. For example, if you want to replace all NUMBER(*,0) into bigint
# instead of numeric(38)add the following:
#       DATA_TYPE       NUMBER(*\,0):bigint
# Here is the default replacement for all Oracle's types. You don't have to
# recopy all type conversion but just the one you want to rewrite.
#DATA_TYPE      DATE:timestamp,LONG:text,LONG
RAW:bytea,CLOB:text,NCLOB:text,BLOB:bytea,BFILE:bytea,RAW:bytea,UROWID:oid,ROWID:oid
,FLOAT:double precision,DEC:decimal,DECIMAL:decimal,DOUBLE PRECISION:double
precision,INT:numeric,INTEGER:numeric,REAL:real,SMALLINT:smallint,BINARY_FLOAT:doubl
e precision,BINARY_DOUBLE:double
precision,TIMESTAMP:timestamp,XMLTYPE:xml,BINARY_INTEGER:integer,PLS_INTEGER:integer
,TIMESTAMP WITH TIME ZONE:timestamp with time zone,TIMESTAMP WITH LOCAL TIME
ZONE:timestamp with time zone

# If set to 1 replace portable numeric type into PostgreSQL internal type.
# Oracle data type NUMBER(p,s) is approximatively converted to real and
# float PostgreSQL data type. If you have monetary fields or don't want
# rounding issues with the extra decimals you should preserve the same
# numeric(p,s) PostgreSQL data type. Do that only if you need very good
# precision because using numeric(p,s) is slower than using real or double.
PG_NUMERIC_TYPE 1

# If set to 1 replace portable numeric type into PostgreSQL internal type.
# Oracle data type NUMBER(p) or NUMBER are converted to smallint, integer
# or bigint PostgreSQL data type following the length of the precision. If
# NUMBER without precision are set to DEFAULT_NUMERIC (see bellow).
PG_INTEGER_TYPE 1

# NUMBER() without precision are converted by default to bigint only if
# PG_INTEGER_TYPE is true. You can overwrite this value to any PG type,
# like integer or float.
DEFAULT_NUMERIC bigint

# Set it to 0 if you don't want to export milliseconds from Oracle timestamp
# columns. Timestamp will be formated with to_char(..., 'YYYY-MM-DD HH24:MI:SS')
# Enabling this directive, the default, format is 'YYYY-MM-DD HH24:MI:SS.FF'.
ENABLE_MICROSECOND      1

# If you want to replace some columns as PostgreSQL boolean define here a list
# of tables and column separated by space as follows. You can also give a type
# and a precision to automatically convert all fields of that type as a boolean.
# For example: NUMBER:1 or CHAR:1 will replace any field of type number(1) or
# char(1) as a boolean in all exported tables.
#REPLACE_AS_BOOLEAN     TB_NAME1:COL_NAME1 TB_NAME1:COL_NAME2 TB_NAME2:COL_NAME2

# Use this to add additional definitions of the possible boolean values in Oracle
# field. You must set a space separated list of TRUE:FALSE values. BY default:
#BOOLEAN_VALUES yes:no y:n 1:0 true:false enabled:disabled

# When Ora2Pg find a "zero" date: 0000-00-00 00:00:00 it is replaced by a NULL.
# This could be a problem if your column is defined with NOT NULL constraint.
# If you can not remove the constraint, use this directive to set an arbitral
# date that will be used instead. You can also use -INFINITY if you don't want
# to use a fake date.
#REPLACE_ZERO_DATE      1970-01-01 00:00:00

# Some time you need to force the destination type, for example a column
# exported as timestamp by Ora2Pg can be forced into type date. Value is
# a comma-separated list of TABLE:COLUMN:TYPE structure. If you need to use
# comma or space inside type definition you will have to backslash them.
#
#         MODIFY_TYPE     TABLE1:COL3:varchar,TABLE1:COL4:decimal(9,6)
#
# Type of table1.col3 will be replaced by a varchar and table1.col4 by
# a decimal with precision.
#
# If the column's type is a user defined type Ora2Pg will autodetect the
# composite type and will export its data using ROW(). Some Oracle user
# defined types are just array of a native type, in this case you may want
# to transform this column in simple array of a PostgreSQL native type.
# To do so, just redefine the destination type as wanted and Ora2Pg will
# also transform the data as an array. For example, with the following
# definition in Oracle:
```

```
#
#         CREATE OR REPLACE TYPE mem_type IS VARRAY(10) of VARCHAR2(15);
#         CREATE TABLE club (Name VARCHAR2(10),
#                 Address VARCHAR2(20),
#                 City VARCHAR2(20),
#                 Phone VARCHAR2(8),
#                 Members mem_type
#         );
#
# custom type "mem_type" is just a string array and can be translated into
# the following in PostgreSQL:
#
#         CREATE TABLE club (
#                 name varchar(10),
#                 address varchar(20),
#                 city varchar(20),
#                 phone varchar(8),
#                 members text[]
#         ) ;
#
# To do so, just use the directive as follow:
#
#         MODIFY_TYPE     CLUB:MEMBERS:text[]
#
# Ora2Pg will take care to transform all data of this column in the correct
# format. Only arrays of characters and numerics types are supported.
#MODIFY_TYPE

# By default Oracle call to function TO_NUMBER will be translated as a cast
# into numeric. For example, TO_NUMBER('10.1234') is converted into PostgreSQL
# call to_number('10.1234')::numeric. If you want you can cast the call to integer
# or bigint by changing the value of the configuration directive. If you need
# better control of the format, just set it as value, for example:
#       TO_NUMBER_CONVERSION    99999999999999999999.9999999999
# will convert the code above as:
#       TO_NUMBER('10.1234', '99999999999999999999.9999999999')
# Any value of the directive that it is not numeric, integer or bigint will
# be taken as a mask format. If set to none, no conversion will be done.
TO_NUMBER_CONVERSION    numeric

#------------------------------------------------------------------------------
# GRANT SECTION (Control priviledge and owner export)
#------------------------------------------------------------------------------

# Set this to 1 to replace default password for all extracted user
# during GRANT export
GEN_USER_PWD    0

# By default the owner of database objects is the one you're using to connect
# to PostgreSQL. If you use an other user (e.g. postgres) you can force
# Ora2Pg to set the object owner to be the one used in the Oracle database by
# setting the directive to 1, or to a completely different username by setting
# the directive value # to that username.
FORCE_OWNER     0

# Ora2Pg use the function's security privileges set in Oracle and it is often
# defined as SECURITY DEFINER. If you want to override those security privileges
# for all functions and use SECURITY DEFINER instead, enable this directive.
FORCE_SECURITY_INVOKER  0

#------------------------------------------------------------------------------
# DATA SECTION (Control data export behaviors)
#------------------------------------------------------------------------------

# Extract data by bulk of DATA_LIMIT tuples at once. Default 10000. If you set
# a high value be sure to have enough memory if you have million of rows.
DATA_LIMIT      2000

# When Ora2Pg detect a table with some BLOB it will automatically reduce the
# value of this directive by dividing it by 10 until his value is below 1000.
# You can control this value by setting BLOB_LIMIT. Exporting BLOB use lot of
# ressources, setting it to a too high value can produce OOM.
#BLOB_LIMIT     500

# By default all data that are not of type date or time are escaped. If you
# experience any problem with that you can set it to 1 to disable it. This
# directive is only used during a COPY export type.
# See STANDARD_CONFORMING_STRINGS for enabling/disabling escape with INSERT
# statements.
NOESCAPE        0

# This directive may be used if you want to change the default isolation
# level of the data export transaction. Default is now to set the level
# to a serializable transaction to ensure data consistency. Here are the
# allowed value of this directive: readonly, readwrite, serializable and
# committed (read committed).
TRANSACTION     serializable

# This controls whether ordinary string literals ('...') treat backslashes
# literally, as specified in SQL standard. This was the default before Ora2Pg
# v8.5 so that all strings was escaped first, now this is currently on, causing
# Ora2Pg to use the escape string syntax (E'...') if this parameter is not
# set to 0. This is the exact behavior of the same option in PostgreSQL.
# This directive is only used during INSERT export to build INSERT statements.
# See NOESCAPE for enabling/disabling escape in COPY statements.
STANDARD_CONFORMING_STRINGS     1

# Use this directive to set the database handle's 'LongReadLen' attribute to
# a value that will be the larger than the expected size of the LOB. The default
# is 1MB witch may not be enough to extract BLOB objects. If the size of the LOB
# exceeds the 'LongReadLen' DBD::Oracle will return a 'ORA-24345: A Truncation'
# error.  Default: 1023*1024 bytes. Take a look at this page to learn more:
#
http://search.cpan.org/~pythian/DBD-Oracle-1.22/Oracle.pm#Data_Interface_for_Persist
ent_LOBs
#
# Important note: If you increase the value of this directive take care that
# DATA_LIMIT will probably needs to be reduced. Even if you only have a 1MB blob
# trying to read 10000 of them (the default DATA_LIMIT) all at once will require
# 10GB of memory. You may extract data from those table separately and set a
# DATA_LIMIT to 500 or lower, otherwise you may experience some out of memory.
#LONGREADLEN    1047552

# If you want to bypass the 'ORA-24345: A Truncation' error, set this directive
# to 1, it will truncate the data extracted to the LongReadLen value.
#LONGTRUNCOK    0

# Disable this if you don't want to load full content of BLOB and CLOB and use
# LOB locators instead. This is usefull to not having to set LONGREADLEN. Note
# that this will not improve speed of BLOB export as most of the time is always
# consumed by the bytea escaping and in this case export is done line by line
# and not by chunk of DATA_LIMIT rows. For more information on how it works, see
#
http://search.cpan.org/~pythian/DBD-Oracle-1.74/lib/DBD/Oracle.pm#Data_Interface_for
_LOB_Locators
# Default is enabled, it will not use LOB locators for backward compatibility.
NO_LOB_LOCATOR  1

# Force the use of getStringVal() instead of getClobVal() for XML data export.
# Default is 1, enabled for backward compatibility. Set here to 0 to use extract
# method a la CLOB and export the XML code as it was stored.
XML_PRETTY      0

# Enable this directive if you want to continue direct data import on error.
# When Ora2Pg receives an error in the COPY or INSERT statement from PostgreSQL
# it will log the statement to a file called TABLENAME_error.log in the output
# directory and continue to next bulk of data. Like this you can try to fix the
# statement and manually reload the error log file. Default is disabled: abort
# import on error.
LOG_ON_ERROR            0

# If you want to convert CHAR(n) from Oracle into varchar(n) or text under
# PostgreSQL, you might want to do some triming on the data. By default
# Ora2Pg will auto-detect this conversion and remove any withspace at both
# leading and trailing position. If you just want to remove the leadings
# character, set the value to LEADING. If you just want to remove the trailing
# character, set the value to TRAILING. Default value is BOTH.
TRIM_TYPE               BOTH

# The default triming character is space, use the directive bellow if you need
# to change the character that will be removed. For example, set it to - if you
# have leading - in the char(n) field. To use space as triming charger, comment
# this directive, this is the default value.
#TRIM_CHAR              -

# Internal timestamp retrieves from custom type are extracted in the following
# format: 01-JAN-77 12.00.00.000000 AM. It is impossible to know the exact century
# that must be used, so by default any year below 49 will be added to 2000
# and others to 1900. You can use this directive to change this default value.
# this is only relevant if you have user defined type with a column timestamp.
INTERNAL_DATE_MAX       49

# Disable this directive if you want to disable check_function_bodies.
#
#        SET check_function_bodies = false;
#
# It disables validation of the function body string during CREATE FUNCTION.
# Default is to use de postgresql.conf setting that enable it by default.
FUNCTION_CHECK          1

# Exporting BLOB takes time, you may want to export all data except the BLOB
# columns. In this case enable this directive and the BLOB columns will not be
# included into data export. That mean that the BLOB column must not have a
# NOT NULL constraint.
NO_BLOB_EXPORT          0

# By default data export order will be done by sorting on table name. If you
# have huge tables at end of alphabetic order and you are using multiprocess
# it can be better to set the sort order on size so that multiple small tables
# can be processed before the largest tables finish. In this case set this
# directive to size. Possible values are name and size. Note that export type
# SHOW_TABLE and SHOW_COLUMN will use this sort order too, not only COPY or
# INSERT export type.
DATA_EXPORT_ORDER       name

#------------------------------------------------------------------------------
# PERFORMANCES SECTION (Control export/import performances)
#------------------------------------------------------------------------------

# This configuration directive adds multiprocess support to COPY, FUNCTION
# and PROCEDURE export type, the value is the number of process to use.
# Default is to not use multiprocess. This directive is used to set the number
# of cores to used to parallelize data import into PostgreSQL. During FUNCTION
# or PROCEDURE export type each function will be translated to plpgsql using a
# new process, the performances gain can be very important when you have tons
# of function to convert. There's no more limitation in parallel processing
# than the number of cores and the PostgreSQL I/O performance capabilities.
# Doesn't works under Windows Operating System, it is simply disabled.
JOBS            1
```

```
# Multiprocess support. This directive should defined the number of parallel
# connection to Oracle when extracting data. The limit is the number of cores
# on your machine. This is useful if Oracle is the bottleneck. Take care that
# this directive can only be used if there is a column defined in DEFINED_PK.
ORACLE_COPIES   1

# Multiprocess support. This directive should defined the number of tables
# in parallel data extraction. The limit is the number of cores on your machine.
# Ora2Pg will open one database connection for each parallel table extraction.
# This directive, when upper than 1, will invalidate ORACLE_COPIES but not JOBS.
# Note that this directive when set upper that 1 will also automatically enable
# the FILE_PER_TABLE directive if your are exporting to files.
PARALLEL_TABLES 1

# You can force Ora2Pg to use /*+ PARALLEL(tbname, degree) */ hint in each
# query used to export data from Oracle by setting a value upper than 1 to
# this directive. A value of 0 or 1 disable the use of parallel hint.
# Default is disabled.
DEFAULT_PARALLELISM_DEGREE      0

# Parallel mode will not be activated if the table have less rows than
# this directive. This prevent fork of Oracle process when it is not
# necessary. Default is 100K rows.
PARALLEL_MIN_ROWS               100000

# Multiprocess support. This directive is used to split the select queries
# between the different connections to Oracle if ORA_COPIES is used. Ora2Pg
# will extract data with the following prepare statement:
#       SELECT * FROM TABLE WHERE MOD(COLUMN, $ORA_COPIES) = ?
# Where $ORA_COPIES is the total number of cores used to extract data and set
# with ORA_COPIES directive, and ? is the current core used at execution time.
# This means that Ora2Pg needs to know the numeric column to use in this query.
# If this column is a real, float, numeric or decimal, you must add the ROUND()
# function with the column to round the value to the nearest integer.
#DEFINED_PK     TABLE:COLUMN TABLE:ROUND(COLUMN)

# Enabling this directive force Ora2Pg to drop all indexes on data import
# tables, except automatic index on primary key, and recreate them at end
# of data import. This may improve speed a lot during a fresh import.
DROP_INDEXES    0

# Specifies whether transaction commit will wait for WAL records to be written
# to disk before the command returns a "success" indication to the client. This
# is the equivalent to set synchronous_commit directive of postgresql.conf file.
# This is only used when you load data directly to PostgreSQL, the default is
# off to disable synchronous commit to gain speed at writing data. Some modified
# versions of PostgreSQL, like Greenplum, do not have this setting, so in this
# case set this directive to 1, ora2pg will not try to change the setting.
SYNCHRONOUS_COMMIT      0
```

```
#------------------------------------------------------------------------------
# PLSQL SECTION (Control SQL and PL/SQL to PLPGSQL rewriting behaviors)
#------------------------------------------------------------------------------

# If the above configuration directive is not enough to validate your PL/SQL code
# enable this configuration directive to allow export of all PL/SQL code even if
# it is marked as invalid. The 'VALID' or 'INVALID' status applies to functions,
# procedures, packages and user defined types.
EXPORT_INVALID  0

# Enable PLSQL to PLPSQL conversion. This is a work in progress, feel
# free modify/add you own code and send me patches. The code is under
# function plsql_toplpgsql in Ora2PG/PLSQL.pm. Default enabled.
PLSQL_PGSQL     1

# Ora2Pg can replace all conditions with a test on NULL by a call to the
# coalesce() function to mimic the Oracle behavior where empty field are
# considered equal to NULL. Ex: (field1 IS NULL) and (field2 IS NOT NULL) will
# be replaced by (coalesce(field1::text, '') = '') and (field2 IS NOT NULL AND
# field2::text <> ''). You might want this replacement to be sure that your
# application will have the same behavior but if you have control on you app
# a better way is to change it to transform empty string into NULL because
# PostgreSQL makes the difference.
NULL_EQUAL_EMPTY        0

# Force empty_clob() and empty_blob() to be exported as NULL instead as empty
# string for the first one and \\x for the second. If NULL is allowed in your
# column this might improve data export speed if you have lot of empty lob.
EMPTY_LOB_NULL          0

# If you don't want to export package as schema but as simple functions you
# might also want to replace all call to package_name.function_name. If you
# disable the PACKAGE_AS_SCHEMA directive then Ora2Pg will replace all call
# to package_name.function_name() by package_name_function_name(). Default
# is to use a schema to emulate package.
PACKAGE_AS_SCHEMA       1

# Enable this directive if the rewrite of Oracle native syntax (+) of
# OUTER JOIN is broken. This will force Ora2Pg to not rewrite such code,
# default is to try to rewrite simple form of rigth outer join for the
# moment.
REWRITE_OUTER_JOIN      1

# By default Oracle functions are marked as STABLE as they can not modify data
# unless when used in PL/SQL with variable assignment or as conditional
# expression. You can force Ora2Pg to create these function as VOLATILE by
# disabling the following configuration directive.
FUNCTION_STABLE         1

# By default call to COMMIT/ROLLBACK are kept untouched by Ora2Pg to force
# the user to review the logic of the function. Once it is fixed in Oracle
# source code or you want to comment this calls enable the following directive
COMMENT_COMMIT_ROLLBACK 0

# It is common to see SAVEPOINT call inside PL/SQL procedure together with
# a ROLLBACK TO savepoint_name. When COMMENT_COMMIT_ROLLBACK is enabled you
# may want to also comment SAVEPOINT calls, in this case enable it.
COMMENT_SAVEPOINT       0

# Ora2Pg replace all string constant during the pl/sql to plpgsql translation,
# string constant are all text include between single quote. If you have some
# string placeholder used in dynamic call to queries you can set a list of
# regexp to be temporary replaced to not break the parser.The list of regexp
# must use the semi colon as separator. For exemple:
#STRING_CONSTANT_REGEXP         <placeholder value=".*">

# If you want to use functions defined in the Orafce library and prevent
# Ora2Pg to translate call to these function, enable this directive.
# The Orafce library can be found here: https://github.com/orafce/orafce
# By default Ora2pg rewrite add_month(), add_year(), date_trunc() and
# to_char() functions, but you may prefer to use the orafce version of
# these function that do not need any code transformation.
USE_ORAFCE      1

# Enable translation of autonomous transactions into a wrapper function
# using dblink or pg_background extension. If you don't want to use this
# translation and just want the function to be exported as a normal one
# without the pragma call, disable this directive.
AUTONOMOUS_TRANSACTION  1

#------------------------------------------------------------------------------
# ASSESSMENT SECTION (Control migration assessment behaviors)
#------------------------------------------------------------------------------

# Activate the migration cost evaluation. Must only be used with SHOW_REPORT,
# FUNCTION, PROCEDURE, PACKAGE and QUERY export type. Default is disabled.
# Note that enabling this directive will force PLSQL_PGSQL activation.
ESTIMATE_COST           1

# Set the value in minutes of the migration cost evaluation unit. Default
# is five minutes per unit.
COST_UNIT_VALUE         5

# By default when using SHOW_REPORT the migration report is generated as
# simple text, enabling this directive will force ora2pg to create a report
# in HTML format.
DUMP_AS_HTML            0

# Set the total number of tables to display in the Top N per row and size
# list in the SHOW_TABLE and SHOW_REPORT output. Default 10.
```

```
TOP_MAX                 10

# Use this directive to redefined the number of human-days limit where the
# migration assessment level must switch from B to C. Default is set to 10
# human-days.
HUMAN_DAYS_LIMIT        5

# Set the comma separated list of username that must be used to filter
# queries from the DBA_AUDIT_TRAIL table. Default is to not scan this
# table and to never look for queries. This parameter is used only with
# SHOW_REPORT and QUERY export type with no input file for queries.
# Note that queries will be normalized before output unlike when a file
# is given at input using the -i option or INPUT directive.
#AUDIT_USER     USERNAME1,USERNAME2

# By default Ora2Pg will convert call to SYS_GUID() Oracle function
# with a call to uuid_generate_v4() from uuid-ossp extension. You can
# redefined it to use the gen_random_uuid() function from pgcrypto
# extension by changing the function name below.
#UUID_FUNCTION  uuid_generate_v4

#------------------------------------------------------------------------------
# POSTGRESQL FEATURE SECTION (Control which PostgreSQL features are available)
#------------------------------------------------------------------------------

# Set the PostgreSQL major version number of the target database. Ex: 9.6 or 10
# Default is current major version at time of a new release. This replace the
# old PG_SUPPORTS_* configuration directives.
PG_VERSION      11

# Use btree_gin extenstion to create bitmap like index with pg >= 9.4
# You will need to create the extension by yourself:
#       create extension btree_gin;
# Default is to create GIN index, when disabled, a btree index will be created
BITMAP_AS_GIN           1

# Use pg_background extension to create an autonomous transaction instead
# of using a dblink wrapper. With pg >= 9.5 only, default is to use dblink.
PG_BACKGROUND           0

# By default if you have an autonomous transaction translated using dblink
# extension instead of pg_background the connection is defined using the
# values set with PG_DSN, PG_USER and PG_PWD. If you want to fully override
# the connection string use this directive as follow to set the connection
# in the autonomous transaction wrapper function.
#DBLINK_CONN    port=5432 dbname=pgdb host=localhost user=pguser password=pgpass

# Some versions of PostgreSQL like Redshift doesn't support substr()
# and it need to be replaced by a call to substring(). In this case,
# disable it.
```

```
PG_SUPPORTS_SUBSTR      1

#------------------------------------------------------------------------------
# SPATIAL SECTION (Control spatial geometry export)
#------------------------------------------------------------------------------

# Enable this directive if you want Ora2Pg to detect the real spatial type and
# dimensions used in a spatial column. By default Ora2Pg will look at spatial
# indexes to see if the layer_gtype and sdo_indx_dims constraint parameters have
# been set, otherwise column will be created with the non-constrained "geometry"
# type. Enabling this feature will force Ora2Pg to scan a sample of 50000 lines
# to look at the GTYPE used. You can increase or reduce the sample by setting
# the value of AUTODETECT_SPATIAL_TYPE to the desired number of line.
AUTODETECT_SPATIAL_TYPE 1

# Disable this directive if you don't want to automatically convert SRID to
# EPSG using the sdo_cs.map_oracle_srid_to_epsg() function. Default: enabled
# If the SDO_SRID returned by Oracle is NULL, it will be replaced by the
# default value 8307 converted to its EPSG value: 4326 (see DEFAULT_SRID)
# If the value is upper than 1, all SRID will be forced to this value, in
# this case DEFAULT_SRID will not be used when Oracle returns a null value
# and the value will be forced to CONVERT_SRID.
# Note that it is also possible to set the EPSG value on Oracle side when
# sdo_cs.map_oracle_srid_to_epsg() return NULL if your want to force the value:
# Ex: system> UPDATE sdo_coord_ref_sys SET legacy_code=41014 WHERE srid = 27572;
CONVERT_SRID            1

# Use this directive to override the default EPSG SRID to used: 4326.
# Can be overwritten by CONVERT_SRID, see above.
DEFAULT_SRID            4326

# This directive can take three values: WKT (default), WKB and INTERNAL.
# When it is set to WKT, Ora2Pg will use SDO_UTIL.TO_WKTGEOMETRY() to
# extract the geometry data. When it is set to WKB, Ora2Pg will use the
# binary output using SDO_UTIL.TO_WKBGEOMETRY(). If those two extract type
# are called at Oracle side, they are slow and you can easily reach Out Of
# Memory when you have lot of rows. Also WKB is not able to export 3D geometry
# and some geometries like CURVEPOLYGON. In this case you may use the INTERNAL
# extraction type. It will use a pure Perl library to convert the SDO_GEOMETRY
# data into a WKT representation, the translation is done on Ora2Pg side.
# This is a work in progress, please validate your exported data geometries
# before use.
GEOMETRY_EXTRACT_TYPE   INTERNAL


#------------------------------------------------------------------------------
# FDW SECTION (Control Foreign Data Wrapper export)
#------------------------------------------------------------------------------

# This directive is used to set the name of the foreign data server that is used
# in the "CREATE SERVER name FOREIGN DATA WRAPPER oracle_fdw ..." command. This
# name will then be used in the "CREATE FOREIGN TABLE ..." SQL command. Default
# is arbitrary set to orcl. This only concerns export type FDW.
FDW_SERVER      orcl


#------------------------------------------------------------------------------
# MYSQL SECTION (Control MySQL export behavior)
#------------------------------------------------------------------------------

# Enable this if double pipe and double ampersand (|| and &&) should not be
# taken as equivalent to OR and AND. It depend of the variable @sql_mode,
# Use it only if Ora2Pg fail on auto detecting this behavior.
MYSQL_PIPES_AS_CONCAT           0

# Enable this directive if you want EXTRACT() replacement to use the internal
# format returned as an integer, for example DD HH24:MM:SS will be replaced
# with format; DDHH24MMSS::bigint, this depend of your apps usage.
MYSQL_INTERNAL_EXTRACT_FORMAT   0
```