# Программа переноса данных из СУБД Oracle в СУБД PostgreSQL (Ora2PgCopy)

# Руководство пользователя

Всего страниц 12

Оглавление

# 1. Назначение

Основной задачей программы Ora2PgCopy является перенос данных из СУБД Oracle в данные СУБД PostgreSQL с максимально возможной скоростью, разработана на Java для совместимости с различными операционными системами и использует PostgreSQL-команду COPY для обеспечения высокой скорости загрузки данных из внешних файлов.

Ora2PgCopy применяется в ходе процесса миграции в тот момент, когда в PostgreSQL уже созданы таблицы и перенесён программный код. Для обеспечения более быстрой миграции целесообразно индексы, ограничения целостности и активацию триггеров выполнять после переноса данных.

Ora2PgCopy может функционировать как модуль в составе системы автоматизации миграции (CAM) LUI4ORA2PG, так и независимо от неё.

# 2. Требования

Программа Ora2PgCopy  работает на платформах, поддерживающих работу с Java, СУБД PostgreSQL, СУБД Oracle.

Минимальный объём дисковой памяти для установки программы Ora2PgCopy составляет 10 МБ.

Используются языки программирования: Java, SQL

Для работы программы необходимы библиотеки jar:

- commons-compress-1.23.0.jar
- commons-io-2.4.jar
- gson-2.7.jar
- joda-time-2.11.0.jar
- lz4-java-1.8.0.jar
- ojdbc8-19.8.0.0.jar
- orai18n-21.11.0.0.jar
- postgresql-42.2.28.jar
- ru.fors.utils.lightsmtp.jar
- vasa.txt
- xdb6-18.3.0.0.jar
- xmlparserv2-18.3.0.0.jar

## 3. Установка программы

Установка программы выполняется распаковкой файла дистрибутива ora2pgcopy.zip, который можно получить на сайте http://ora2pgcopy.forstelecom.ru или по запросу на e-mail support@forstelecom.ru .

**Структура каталогов после распаковки:**

./ora2pgcopy/lib/
- commons-compress-1.23.0.jar
- commons-io-2.4.jar
- gson-2.7.jar
- joda-time-2.11.0.jar
- lz4-java-1.8.0.jar
- ojdbc8-19.8.0.0.jar
- orai18n-21.11.0.0.jar
- postgresql-42.2.28.jar
- ru.fors.utils.lightsmtp.jar
- vasa.txt
- xdb6-18.3.0.0.jar
- xmlparserv2-18.3.0.0.jar

./ora2pgcopy/
- start.sh
- start.bat
- start_agent.sh
- start_agent.bat
- ora2pgcopy.json
- ora2pgcopy.jar

./ora2pgcopy/doc/
- Ora2PgCopy.Users guide.pdf

**Пример файла конфигурации ora2pgcopy.json :**

```
{ /* пример JSON - структруры для утилиты ora2pgcopy.*/
 "sessions": 16, /*количество потоков - максимальное количество одновременно выгружаемых/загружаемых таблиц. необязательный параметр. default=7*/
 /* "outfile": "myoutfile",*/ /*имя sh-файла, в котором будут генерироваться вызовы для последовательных загрузок утилитой psql. необязательный параметр. Если не указан файл не будет создаваться.*/
 "truncate": true, /*необходимость выполнить в PG trucate table перед загрузкой. необязательный параметр. default=false*/
 "measure" : "R", /*Единица измерения метрик статистики R-строки таблиц, B-байты, K-килобайты, M-мегабайты*/
 "zip":false, /* следует ли сжимать выходные sql-файлы утилитой gzip. необязательный параметр. default=false*/
```

"oradb": "orahost:1521:orasid", /*jdbc-url для подключения к oracle. <хост>:<порт>:<sid> или //<хост>/<сервис> .*/

/*В некоторых операционных системах может увеличиться производительность при применении OCI-драйвера. Для его использования следует в параметре "oradb" указать полную строку соединения с сервером.

Примеры: "jdbc:oracle:oci:@orahost:1521:orasid"

"jdbc:oracle:oci8:@orahost:1521:orasid"

Для работы через (OCI-драйвер) на сервере, где работает Ora2PgCopy должен быть установлен Oracle Instant Client version 19 (https://www.oracle.com/database/technologies/instant-client/downloads.html)

Если у Вас нет возможности установить Oracle Instant Client version 19, но у Вас уже есть Oracle Instant Client версии 12 и выше, то можно выполнить следующее.

Скопируйте ojdbc8.jar из папки, в которой установлен Oracle Instant Client в папку lib утилиты Ora2PgCopy. При этом дайте файлу имя ojdbc8-19.8.0.0.jar. Т.е. следует заменить файл ojdbc8-19.8.0.0.jar.

Если данные из Oracle извлекаются с помощью агента утилиты Ora2PgCopy, то установку Oracle Instant Client следует выполнять на стороне агента.*/

"orausername": "orausr", /*имя пользователя oracle*/

"orapassword": "usrpwd", /*пароль пользователя oracle*/

"pghost": "localhost", /*имя или ip-адрес PG-сервера*/

"pgdb": "postgres", /*имя базы данных PG*/

"pgport":"5433", /*порт листенера PG. необязательный параметр. default=5432*/

"pgusername": "pgusr", /*имя пользователя PG*/

"pgpassword": "usrpwd", /*пароль пользователя PG*/

"set2unlogged":false, /*следует ли перед загрузкой в PG выполнять для таблицы alter table set unlogged. необязательный параметр. default=false*/

"set2logged":false, /*следует ли после загрузки в PG выполнять для таблицы alter table set logged. необязательный параметр. default=false*/

"unload":"yes", /*следует ли выполнять выгрузку из oracle. Возм. значения: "yes","no","if target empty". необязательный параметр. default="yes".

"if target empty" - выгрузка будет выполняться если в таблице PG нет строк.

Кроме того, возможно выполнить запрос в PG или Oracle, который должен вернуть одно из вышеописанных значений.

Примеры: "unload":"pg:select (case count(q.cnt)::text when '0' then 'yes' else 'no' end) from (select 1 cnt from sd_t_request where id>100 limit 1)q"

"unload":"ora:select my_function() from dual"

*/

"load":"direct", /*следует ли выполнять загрузку в PG. Возм. значения: "yes","no","if target empty","direct","directbinary". Необязательный параметр. default="no".

"if target empty" - загрузка будет выполняться если в таблице PG нет строк.

"direct" - выгружаемые данные загружаются напрямую без промежуточного sql-файла.

"directbinary" - выгружаемые данные загружаются напрямую без промежуточного sql-файла. При этом, команде COPY передаются данные в двоичном виде.

Это может увеличивать в 1.5-2 раза скорость переноса таблиц, в которых нет текстовых полей длиннее 1К.

"direct" и "directbinary" допустим, если unload="yes" или "if target empty"

Кроме того, возможно выполнить запрос в PG или Oracle, который должен вернуть одно из вышеописанных значений.

Примеры: См. выше в описании "unload" */

"index":"yes", /*следует ли выполнять операторы индексирования, если они указаны. Возм. значения: "yes","no","if not exist". необязательный параметр. default="yes".

"if not exists" подавляет ошибки типа "такое уже есть"

Кроме того, возможно выполнить запрос в PG или Oracle, который должен вернуть одно из вышеописанных значений.

Примеры: См. выше в описании "unload" */

"sessionwriters":5, /*количество подпроцессов параллельной выгрузки (в случае "direct" и загрузки в PG) таблицы из oracle. необязательный параметр. default=8

замечание: если выгрузка таблицы выполняется в несколько подпроцессов (т.е. sessionwriters >1), то в строки в PG могут оказаться загруженными не в том порядке,

в каком они выгружались*/

"parallel":false, /* Этот параметр действует если загрузка выполняется в режиме direct или directbinary. По умочанию = false. true указывает утилите на то, что данные в целевую таблицу

будут копироваться несколькими одновременно выполняемыми командами COPY, каждая из которых будет использовать отдельный сеанс в целевой БД. Количество таких параллельных COPY

определяется параметром sessionwriters. Устанавливать true для таблиц с CLOB или BLOB не рекомендуется. true эффективен тогда, когда данные из oracle извлекаются быстрее, чем

записываются в postgres. На длинных, но узких таблицах true может давать до 40% прироста скорости*/

"parallellimit":16, /* максимальное количество сеансов в postgres, передающих данные в режиме parallel и direct/directbinary.

Он позволяет ограничить избыточную нагрузку и конкуренцию в postgres и более равномерно распределить нагрузку на ресурсы по времени работы программы.

При старте задачи обработки таблицы (в режиме parallel=true), выполняется оценка количества уже инициированных сеансов в postgres.

И если это количество может быть превышено начинающимся процессом, то процесс ставится в режим ожидания освобождения нужного количества сеансов (sessionwriters).

По умолчанию этот параметр равен sessions, и если его не установить явно, то выигрыш от parallel может исчезнуть.

Равномерность распределения нагрузки на ресурсы можно попытаться достичь следующим й образом.

Пусть есть 100 таблиц и требуется перенести их данные, так чтобы количество сеансов в postgres не превышало 30.

Тогда sessions=100, parallel=true, sessionwriters=5, parallellimit=30. При этом, массив work должне быть сформирован в порядке убывания размеров данных в таблицах.

Т.е. самые объёмные таблицы должны быть указаны в начале массива work.

*/

"prefetchrows":4096, /*количество строк опережающего fetch из oracle для снижения частоты обращений к нему. необязательный параметр. default=256 */

"lobprefetchsize": 32768, /*количество байт опережающего fetch из oracle LOB для снижения частоты обращений к нему. необязательный параметр. default=32768 */

"lobstreambuffer": 32768, /*размер буфера для выкачивания данных из потока oracle LOB. необязательный параметр. default=16384 */

/* "agent":"agent_host:5555:lz4:0:1024",*/ /* Этот параметр указывает утилите, что данные надо брать не непосредственно из oracle, а из потоков данных передаваемых агентом. Количество потоков для каждой таблицы

определяется параметром sessionwriters. Желательно, чтобы агент работал на том же компьютере, где работает oracle. Агент может сжимать данные и передавать их по сети

основной программе. Основная программа будет их разжимать и передавать команде COPY или формировать файлы для последующего выполнения утилитой psql. Такая технология

экономит ресурсы сети и может увеличить скорость в несколько раз.

Этот параметр представляет собой коллекцию атрибутов разделённых символом ":".

1-й атрибут - имя или ip-адрес хоста, где запущен агент

2-й атрибут - порт, на котором агент слушает команды основной программы

3-й атрибут - алгоритм сжатия данных. Возможные значения: none-без сжатия, gzip - "gzip",lz4 - "lz4"

4-й атрибут - уровень сжатия. Для gzip - от 0 до 9 (по умолчанию 7), Для lz4 от 0 до 17 (по умолчанию 0)

5-й атрибут - размер буфера памяти (Кбайт) для выполнения сжатия (по умолчанию 64К)*/

"work": [ /*массив описаний таблиц для выгрузки/загрузки. В каждом элементе массива могут быть переопределены параметры описанные выше. (кроме sessions и sendmail, конечно)

замечание: если переопределить параметры подключения к oracle и pg, то можно выкачивать данные из разных баз и закачивать в разные базы

замечание: подключение к PG указанное на верхнем уровне используется для измерения скоростей загрузок и индексирования в PG,

и PG-подключения, заданные на уровне элемента массива выпадают из области измерений скоростей*/

{

"query": "select * from fwork.ad_t_audit", /*это запрос к исходным данным*/

"target": "lui.ad_t_audit", /*это имя целевых схемы и таблицы */

"indexstatements":"ALTER TABLE lui.ad_t_audit ADD CONSTRAINT ad_t_audit_pk PRIMARY KEY (id);CREATE INDEX ad_t_audit_usr ON lui.ad_t_audit USING btree (user_name);" /*это набор sql-операторв для создания индексов*/

},

{

"query": "select id,frmt,art,contr,date_begin,addq,latdt,decode(is_valid,'Y','true','N','false') is_valid,summa,date_valid,validation_result,new_txt

from prm.t_table where id <=1000000",

"target": "prm.t_table",

"indexstatements":"ALTER TABLE prm.t_table ADD CONSTRAINT t_table_pk PRIMARY KEY (id) USING INDEX TABLESPACE lui;",

/* Далее описаны "workid","sqlbefore","sqlafterdata","sqlafter". Они позволяют вести собственный журнал выполненных работ в PG или в Oracle.

Он особенно выгоден если одна целевая таблица указана в массиве несколько раз и загружается несколькими параллельными потоками.

А вместе с запросами в "unload", "load" и "index" эти параметры позволяют гибко управлять процессом миграции данных. Особенно если он выполняется итеративно - т.е. несколько раз,

например, в связи с возникающими ошибками. Эти параметры позволяют вновь выполнять процесс миграции без изменения файла конфигурации. При чём, каждый раз будет предприниматься попытка выполнить то,

что не было сделано в предшествующих итерациях.

Например, можно разбить самую большую таблицу на несколько частей по диапазонам первичного ключа. И выполнять её передачу в PG несколькими параллельными потоками.

Это повысит утилизацию ресурсов техники в конце работы утилиты. И сделает загрузку ресурсов более равномерной. Что делать если какая то часть не была перенесена из-за ошибок?

Каждая часть таблицы будет иметь свой "workid". В "sqlafterdata" Вы можете в своей собственной таблице сохранять факты успешных или неуспешных миграций частей.

А в параметрах "unload", "load" использовать запросы, которые будут возвращать "no" если эта часть данных перенесена. И "yes" и, например "direct" если - нет.

А в параметре "index" - запрос, который вернёт "yes" если все части таблицы перенесены.

Вычисление значения параметра "index" выполняется после выполнения "sqlafterdata".*/

"workid":"t_table1", /*это Ваш прикладной идентификатор выполняемой работы. Необязательный параметр. Он будет выдаваться в сообщениях вместе с именем таблицы. Он полезен, если для одной целевой таблицы

будет более одного элемента в массиве.*/

"sqlbefore":"pg:select my_function_before(?,?)", /*Это оператор SQL выполняемый в PG(pg:) или в Oracle(ora:) перед обработкой элемента массива.

Может быть задано не более 2-х текстовых параметров подстановки.

Они кодируются знаком вопроса (как принято в jdbc). 1-й - это значение атрибута "target". 2-й - это значение атрибута "workid".

Если workid не задан , то будет подставляться номер элемента массива..

Если знаков вопроса менее 2-х или нет совсем, то ошибки не будет. Если их более 2-х возможно возникновение ошибки.*/

"sqlafterdata":"pg:select my_function_after_data(?,?,?)", /*Это оператор SQL выполняемый в PG(pg:) или в Oracle(ora:) после передачи данных, но перед выполнением индексирования.

Он будет выполнен в любом случае, не зависимо от наличия ошибок, и не зависимо от необходимости индексирования.

Может быть задано не более 3-х текстовых параметров подстановки.

Они кодируются знаком вопроса (как принято в jdbc). 1-й - это значение атрибута "target". 2-й - это значение атрибута "workid".

Если workid не задан , то будет подставляться номер элемента массива.. 3-й параметр - это текст ошибки. Если обработка завершилась без ошибок - пустая строка.

Если знаков вопроса менее 3-х или нет совсем, то ошибки не будет. Если их более 3-х возможно возникновение ошибки.*/

"unload":"pg:select my_function_need_unload(?,?)", /*определение необходимости извлечения данных пользовательской функцией в PG.

Может быть задано не более 2-х текстовых параметров подстановки.

Они кодируются знаком вопроса (как принято в jdbc). 1-й - это значение атрибута "target". 2-й - это значение атрибута "workid".

Если workid не задан , то будет подставляться номер элемента массива.

Если знаков вопроса менее 2-х или нет совсем, то ошибки не будет. Если их более 2-х возможно возникновение ошибки.

Вычисление значения параметра "unload" выполняется после "sqlbefore"*/

"load":"pg:select my_function_need_load(?,?)", /*определение необходимости загрузки пользовательской функцией в PG.

Вычисление значения параметра "load" выполняется после "sqlbefore"*/

"index":"pg:select my_function_need_index(?,?)", /*определение необходимости индексирования пользовательской функцией в PG.

Вычисление значения параметра "index" выполняется после "sqlafterdata"*/

"sqlafter":"pg:select my_function_after(?,?,?)", /*Это оператор SQL выполняемый в PG(pg:) или в Oracle(ora:) после обработки элемента массива. В остальном описание совпадает с "sqlafterdata"*/

/* Поскольку вычисление выполняется с передачей параметров "target" и "workid", "sqlbefore","sqlafterdata","sqlafter","unload","load" и "index" целесообразно задавать не на уровне элемента массива,

а в общей части этого json */

},

{ /*это 2-я часть t_table*/

"query": "select id,frmt,art,contr,date_begin,addq,latdt,decode(is_valid,'Y','true','N','false') is_valid,summa,date_valid,validation_result,new_txt

from prm.t_table where id >1000000",

"target": "prm.t_table",

"indexstatements":"ALTER TABLE prm.t_table ADD CONSTRAINT t_table_pk PRIMARY KEY (id) USING INDEX TABLESPACE lui;",

"workid":"t_table2",

"sqlbefore":"pg:select my_function_before(?,?)",

"sqlafterdata":"pg:select my_function_after_data(?,?,?)",

"unload":"pg:select my_function_need_unload(?,?)",

"load":"pg:select my_function_need_load(?,?)",

"index":"pg:select my_function_need_index(?,?)",

"sqlafter":"pg:select my_function_after(?,?,?)",

},

{

"query": "select * from book.o2p_t_book_view_log ",

"target": "book.o2p_t_book_view_log",

"indexstatements":"ALTER TABLE book.o2p_t_book_view_log ADD CONSTRAINT o2p_t_book_view_log_pk PRIMARY KEY (id) USING INDEX TABLESPACE lui;",

"freeze": false,

"clobswithzerochar":"BOOK_ANNOTATION", /*перечень столбцов типа CLOB, в которых среди текста может встречаться chr(0). Этот параметр используется в directbinary-режиме. Для оптимизации производительности по умолчанию предполагается, что в CLOB таких символов нет*/

"load":"directbinary"

}

,

{

"query": "select * from mag.TBL_SYS_FILECONTENT01",

"target": "lui.TBL_SYS_FILECONTENT01",

"lobprefetchsize": 262144, /*количество байт опережающего fetch из oracle LOB для снижения частоты обращений к нему. необязательный параметр. default=32768 */

"lobstreambuffer": 262144, /*размер буфера для выкачивания данных из потока oracle LOB. необязательный параметр. default=16384 */

"locolumns":"content,content2" /* перечень столбцов для создания large object в postgres. Это нужно если данные в полях могут превышать 1Г. Перед загрузкой строки в таблицу будут создаваться объекты типа “lo” и их идентификаторы будут записаны в строку таблицы */

},

{"target":"lui.\"z#яtable\"", /*это пример кодирования таблиц в PG с не конвенциональными символами в имени */

"outfile":"lui.ztable",

"query":"select * from prm.z#яtable"

}

]

/* следить за ходом выгрузки/загрузки/индекстирования можно с помощью запроса:

```
select
case row_number() OVER()::text
        when '1' then 'TotalTables'
        when '2' then 'TotalUnloaded'
        when '3' then 'TotalLoaded'
        when '4' then 'TotalIndexed'
        when '5' then 'NowUnloading'
        when '6' then 'NowLoading'
        when '7' then 'NowIndexing'
        when '8' then 'UnloadingSpeed'
        when '9' then 'LoadingSpeed'
        when '10' then 'IndexSpeed'
        when '11' then 'Errors'
        end code,
        r::numeric value
from(
 select   regexp_split_to_table(replace(application_name,'ora2pgcopy ',''),';') r
 from pg_stat_activity
 where application_name like 'ora2pgcopy %')q
 where q.r>'';
```

следить за ходом создания больших объектов можно с помощью запроаса:

```
select
case row_number() OVER()::text
        when '1' then 'Object'
        when '2' then 'Bytes'
        end code,
r::numeric value
from (select regexp_split_to_table(REPLACE(application_name,'ora2pgcopy: loading large object ',''),' ') r
from pg_stat_activity where application_name like 'ora2pgcopy: loading large objects%'
)q;
```

```
*/
}
```

## 4. Старт и эксплуатация программы

Программа выполняется в пакетном режиме специально подготовленным скриптом, например ora2pgcopy-start.sh, где в приведённом примере выделяется 8 ГБ ОЗУ для JVM (это при необходимости работы с большим числом параллельных потоков, обычно не надо указывать этот параметр):

```
java -Xmx8192m -Dfile.encoding=UTF-8 -jar ora2pgcopy.jar ora2pgcopy.json
```

После старта утилита выводит строки диагностики выполнения в стандартный поток данных, также образуются файлы данных и возникающих ошибок.

Настройка работы программы выполняются в конфигурационном файле, см. пример файла ora2pgcopy.json в разделе п.3, где приведено описание параметров настроек.

Программа не создаёт таблицы, целевые таблицы должны уже существовать.

Имена целевых столбцов должны совпадать с именами столбцов запроса, порядок столбцов не имеет значения.

Типы данных каждого столбца должны быть совместимы.

## 5. Особенности программы

**Java.** Программа Ora2PgCopy разработана на Java для обеспечения независимости от типа операционной системы (ОС), при этом поддерживается версии JVM от 8 до 19. Поддержка устаревших версий Java объясняется их присутствием в дистрибутивах во многих отечественных ОС.

**COPY.** Для переноса данных программа Ora2PgCopy применяет команду COPY СУБД PostgreSQL, которая быстрее всех иных средств загружает данные в таблицы Postgres из внешних источников.

Ora2PgCopy поддерживает бинарный формат входных данных команды COPY. Бинарный формат в ряде случаев работает существенно быстрее текстового. Например, когда таблица состоит из небольшого количества числовых столбцов и/или столбцов типов дата/время. А также в том случае, когда в Postgres по сети передаются данные типа bytea (двоичная строка переменной длины).

**Многопоточность.** Ora2PgCopy использует многопоточные технологии Java. Одновременно выполняется обработка нескольких таблиц. При этом, обработка каждой таблицы выполняется также в несколько потоков.

**LOB/CLOB.** Особое внимание в ходе разработки было уделено переносу данных типа LOB и CLOB:

- учтены рекомендации Oracle Corp. по разработке программ для быстрого чтения таких данных
- реализована миграция в Postgres LOB и CLOB полей, объём которых превышает 1Гб. Известно, что Postgres не может в одном поле таблицы хранить данные длиннее одного гигабайта. Для таких данных в Postgres применяется специальная технология “large objects”.

**Сценарии**. Ora2PgCopy поддерживает следующие сценарии миграции данных:

**Сценарий 1 – Выгрузка.**

Выгрузка всех требуемых таблиц из Oracle в файлы, которые в дальнейшем обрабатываются утилитой psql (аналог Oracle SQL*Plus в Postgres). Эти файлы содержат SQL-команду COPY и массивы данных для неё. Полученные файлы в последующем могут обрабатываться уже отдельно вне зоны внимания системы и модуля миграции данных. Кроме того, полученные файлы могут обрабатываться модулем Ora2PgCopy. В этом случае программа автоматически запустит psql в несколько потоков. Данный сценарий может быть применён для тех случаев, когда физический перенос носителя с файлами из сервера Oracle в сервер Postgres выполняется быстрее, чем передача данных по сети.

**Сценарий 2 – Выгрузка/загрузка.**

Выгрузка всех требуемых таблиц из Oracle в файлы. Причём, как только таблица целиком выгружена в файл, сразу же начинается его загрузка в Postgres. После успешной загрузки файл удаляется.

**Сценарий 3 – Прямое копирование.**

Чтение данных из Oracle с их одновременной загрузкой в Postgres без создания промежуточных файлов. В этом режиме посредничество утилиты psql не требуется загрузка данных выполняется посредством Java-API команды COPY, который разработан и поставляется Postgre-сообществом.

Этот сценарий самый быстрый и удобен тогда, когда места для хранения промежуточных файлов практически нет. Однако, в случае возникновения ошибки при загрузке в Postgres какой-либо таблицы, хотя в журналах сохранится сообщение об ошибке и номер строки исходных данных, но, отсутствие промежуточного файла с данными вызовет некоторые трудности с поиском причины ошибки. Поэтому этот режим следует применять после итеративной отладки процесса миграции с помощью сценариев 1 и/или 2.

**Сценарий 4 – агент на стороне Oracle и основная программа на стороне Postgres**

Во многих случаях общая производительность работы трёх предыдущих сценариев будет зависеть от пропускной способности сети, по которой выполняется передача данных извлекаемых из Oracle. Для повышения производительности следует разместить ora2pgcopy на двух серверах – и на сервере с Oracle и на сервере

с Postgres. На сервере с Oracle утилита должна быть запущена в режиме агента. Пример:

```
java -Xmx8192m -Dfile.encoding=UTF-8 -jar ora2pgcopy.jar agent localhost:5555
```

На сервере с Postgres – в режиме основной программы. Пример:

```
java -Xmx8192m -Dfile.encoding=UTF-8 -jar ora2pgcopy.jar ora2pgcopy.json
```

При этом, json-файл должен содержать параметр “agent”. Пример:

"agent":"ora_host:5555:lz4:9:1024"

Здесь

ora_host – имя или ip-адрес сервера на котором работает агент,

5555 – номер порта,

lz4 – алгоритм сжатия,

9 – уровень сжатия,

1024 – размер буфера памяти для сжатия в килобайтах.

Параметр “agent” может быть указан на уровне элемента массива “work”.

**Дополнительные возможности Ora2PgCopy:**

- усечение таблиц перед загрузкой
- многопоточное индексирование после загрузки
- уведомление по e-mail о нормальном или аварийном завершении
- перевод требуемых таблиц в nologged - состояние перед загрузкой и logged после загрузки
- источником данных может быть не только таблица, но и любой sql-запрос в Oracle
- источниками данных могут быть несколько разных баз данных (экземпляров) Oracle
- целевые таблицы могут быть размещены в нескольких разных базах данных Postgres
- широкий набор настроек: количество одновременно обрабатываемых таблиц, количество потоков на таблицу, размеры буферной памяти и т.д.
- преобразование символьных и числовых данных к boolean
- за ходом процесса можно наблюдать не только на консоли, но и запросами в Postgres, извлекая следующие данные:
  - количество таблиц в процессе выгрузки
  - количество таблиц в процессе загрузки
  - количество таблиц в процессе индексирования
  - количество выгруженных таблиц
  - количество загруженных таблиц

- количество проиндексированных таблиц
- текущая скорость выгрузки, загрузки и индексирования.

---