Инкрементальная миграция данных из СУБД Oracle в СУБД Postgres

Ora2PgSync

Руководство пользователя

Всего страниц 16

Оглавление

1. Описание технологии и инструментов, используемых для выполнения миграции

1.1. Основные принципы

Инкрементальная миграция предполагается к выполнению после завершения основной миграции данных из СУБД Oracle в СУБД Postgres. Она представляет собой некое подобие синхронизации изменений данных в Oracle и Postgres. Если происходит добавление/изменение/удаление записи в таблице Oracle, аналогичное действие выполняется и в соответствующей таблице Postgres.

Инкрементальная миграция, как правило, требуется в период, когда после выполнения основной миграции данных пользователи всё ещё работают в БД Oracle, пока выполняется проверка и различные тесты в БД PostgreSQL корректности выполненной основной миграции, а также, пока подготавливается переход на клиентскую часть программного обеспечения (ПО), работающую с Postgres. В этот период следует постоянно актуализировать данные в БД PostgreSQL при их изменении в БД Oracle. И делать это надо до окончательного переключения пользователей на работу с БД PostgreSQL.

Исходными данными для выполнения инкрементальной миграции являются архивные redo-логи, создаваемые средствами Oracle (см. главу “Настройка логирования в Oracle”). В этих логах сохраняются сведения о добавлении/изменении/удалении каждой записи в каждой таблице, а также о транзакциях, в рамках которых выполнены изменения. В упомянутых логах, в том числе, есть сведения и о выполненных DDL-операциях. Но такие сведения не рассматриваются, т.к. при инкрементальной миграции должны быть запрещены какие-либо изменения в структуре объектов базы данных.

Для извлечения исходных данных из архивных логов используется встроенный в СУБД Oracle инструмент LogMiner. Он читает логи и формирует в БД Oracle представление данных (view), которое отображает интересующую информацию по изменению данных в БД в удобном виде.

Данные, представляемые LogMiner-ом, а именно, начало/завершение транзакции, а также выполненные в рамках неё DML-операции (insert/update/delete), анализируются программнвм обеспечением ora2pgsync. Сама программа преобразует текст оракловых DML-команд, к виду, который применим в Postgres. И выполняет аналогичные DML-команды в БД Postgres, синхронизируя данные в интересующих таблицах Oracle и Postgres.

1.2. Oracle LogMiner

1.2.1. Описание и основы использования

LogMiner, являющийся частью СУБД Oracle, позволяет выбирать данные из архивных redo лог-файлов через SQL-интерфейс, посредством формирования пользовательским приложением запросов (select) к подготавливаемому LogMiner-ом оракловому представлению (view) `v$logmnr_contents`.

Следует обратить внимание, что в данной реализации инкрементальной миграции анализируются только архивные лог-файлы. Обработка оперативных лог-файлы не выполняется.
Представление `v$logmnr_contents` содержит сведения о сессиях, транзакциях и выполненных в рамках них DML-операциях в БД Oracle. Этих сведений достаточно, чтобы выполнить аналогичные DML-операции в БД PostgreSQL.

Но до того, как данные попадут в представление v$logmnr_contents, LogMiner должен получить список анализируемых лог-файлов. Для добавления лог-файлов в LogMiner используется процедура `dbms_logmnr.add_logfile()`.

После этого необходимо как минимум запустить LogMiner процедурой `dbms_logmnr.start_logmnr()`, передав в неё ряд опций, определяющих режимы работы LogMiner. И только после этого становятся доступными запросы к представлению `v$logmnr_contents`.

Завершается работа LogMiner вызовом процедуры `dbms_logmnr.end_logmnr()`.

Все эти действия выполняются программой `ora2pgsync` в ходе выполнения инкрементальной миграции. В следующем пункте рассмотрено с какими опциями программа `ora2pgsync` запускает LogMiner и почему.

1.2.2. Режимы работы LogMiner, их анализ и обоснование выбора

При старте LogMiner, необходимо указать ряд опций, определяющих режим его работы.
В архивных лог-файлах вместо имён объектов присутствуют ссылки на них в виде чисел. Для сопоставления ссылок и объектов, и формирования корректных DML-команд при подготовке представления данных (view) `v$logmnr_contents` LogMiner должен обращаться к словарю. И при старте логмайнера указывается, какой словарь следует использовать.

- **DICT_FROM_ONLINE_CATALOG** – указание онлайн-каталога в качестве источника словаря при запуске LogMiner для использования словаря объектов, который в настоящее время используется для базы данных.

- **DICT_FROM_REDO_LOGS** – логмайнер ожидает найти словарь в самих архивных лог-файлах, добавляемых в логмайнер при помощи `dbms_logmnr.add_logfile()`. Это полезно, когда лог-файлы формируются одной базой данный, а анализируются в другой.
  Узнать, есть ли в архивных логах данные о словарях, можно выполнив следующие запросы к представлению `v$archived_log`:
  ```
  SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE DICTIONARY_BEGIN='YES';
  SQL> SELECT NAME FROM V$ARCHIVED_LOG WHERE DICTIONARY_END='YES';
  ```
  Если данные о словарях в архивных лог-файлах отсутствуют, то следует выполнить их добавление вызовом процедуры
  ```
  dbms_logmnr_d.build(options => dbms_logmnr_d.store_in_redo_logs).
  ```
  И заново вызвать `dbms_logmnr.add_logfile()`.

Есть ещё возможность выгружать словарь в отдельный файл при помощи `dbms_logmnr_d.build()`, а затем использовать его, указывая в качестве значения параметра

dictfilename при старте логмайнера, но такая возможность считается устаревшей и неоптимальной, поэтому не рассматривается.

Было принято решение для определения словаря для логмайнера использовать опцию dbms_logmnr.DICT_FROM_ONLINE_CATALOG.

Во-первых, размещение словаря внутри лог-файлов (dbms_logmnr.DICT_FROM_REDO_LOGS) увеличивает их размер, а также время на обработку логмайнером. Запрос данных из представления v$logmnr_contents сам по себе выполняется долго, а с логами внутри файлов он работает ещё медленнее.

Во-вторых, сам алгоритм добавления словарей внутрь логов является недостаточно прозрачным, и не очень надежным, если анализировать приходится каждый раз новые лог-файлы, забывая о старых. Получается, что для надёжности в каждом новом лог-файле должен быть полный словарь. Но Oracle при достаточно большом словаре размещает данные о нём в двух и более лог-файлах. В первом из них DICTIONARY_BEGIN='YES', в последнем DICTIONARY_END='YES'. Что не позволит обработать каждый из таких файлов по-отдельности. Возникнет ошибка "Complete LogMiner dictionary not found". Такое решение не подходит, т.к. по причине медленной работы с v$logmnr_contents логмайнеру выгодней обрабатывать как можно меньшее количество лог-файлов за раз, а в идеале один.

- **COMMITTED_DATA_ONLY** – в этом случае в представление v$logmnr_contents будут попадать только те изменения данных, которые принадлежат зафиксированным транзакциям.

  На первый взгляд, режим очень удобный. Автоматически отбрасываются все DML-операции, выполнение которых завершилось командой **rollback** или пока ещё не завершилось командой **commit**.

  Но есть у этого режима и весьма существенный недостаток. При длинных транзакциях при выполнении **commit** в представление v$logmnr_contents присутствует запись со значением start_scn, с которого началась транзакция и commit_scn, которым она завершилась. И если в логмайнер перед его стартом не добавлены все архивные лог-файлы, содержащие сведения обо всех изменениях с SCN (session change number) с момента start_scn, то все остальные записи об изменениях в рамках этой транзакции будут содержать **null** в поле start_scn, т.е. будут некорректными. Таких записей явно недостаточно, чтобы корректно внести все изменения в Postgres, выполненные в рамках транзакции в Oracle. И инкрементальная миграции будет неуспешна (с ошибками недовыполнения синхронизации).

  Для того, чтобы в v$logmnr_contents, в режиме dbms_logmnr.COMMITED_DATA_ONLY гарантированно присутствовали все необходимые данные, надо регистрировать в логмайнере все необходимые лог файлы.

  Для понимания, какие файлы являются необходимыми, надо:
  - сначала добавить в логмайнер все ещё не анализируемые новые лог-файлы;
  - стартовать логмайнер;
  - посмотреть запросом к представлению v$logmnr_contents все записи о commit, выбрать среди них минимальное значение start_scn;
  - запросом к представлению v$archived_log найти лог-файл, для которого найденное на предыдущем шаге значение start_scn попадает в интервал его значений first_change# и next_change#, а также все более поздние лог-файлы;
  - добавить все эти файлы в логмайнер;
  - и заново стартовать логмайнер, чтобы запрос к нему теперь уже выдавал корректные результаты.

  Помимо того, что сама по себе задача выглядит не очень красиво, так ещё и следует дважды делать запросы к представлению v$logmnr_contents , которое, как упоминалось, работает медленно. Причём, во второй раз, оно будет включать в себя данные из множества архивных файлов, что ещё больше замедлит его работу. И каждый раз, когда в следующем архивном файле будет **commit** длинной транзакции, вполне вероятно, что снова и снова придётся добавлять в логмайнер одни и те же файлы, из которых среди множества уже

обработанных данных будут выбираться сведения лишь по одной или нескольким завершившимся давно стартовавшим длинным транзакциям. Это очень неэффективно.

Принято решение не использовать опцию `dbms_logmnr.COMMITED_DATA_ONLY`. Каждый раз при обработке новых архивных лог-файлов один единственный раз следует анализировать только их содержимое и выполнять ровно те операции в Postgres, которые присутствуют в данных из обрабатываемых лог-файлов. В идеале будет обрабатываться каждый раз один новый архивный файл с redo-логами в случае его появления. Все незавершившиеся транзакции останутся таковыми и в Postgres, ожидая обработку нового архивного лог-файла (новых DML-операций в рамках тех же транзакций и возможных **`commit`** или **`rollback`**). Это гораздо более прозрачное и простое решение, которое является к тому же и более надежным. И приводит к значительному ускорению обработки и сокращению запаздывания синхронизации данных в Oracle и Postgres при инкрементальной миграции.

- **SKIP_CORRUPTION** – при указании этой опции любые повреждения в архивных redo-лог файлах пропускаются во время выполнения запросов к представлению `v$logmnr_contents`.

Такое решение неприемлемо, т.к существует вероятность рассинхронизации данных Oracle и Postgres, о которой умолчали. Поэтому данная опция не используется.

- **NO_SQL_DELIMITER** – в текст выполненных DML-операций не добавляется точка с запятой в конец предложения.

  Данная опция не используется.

- **PRINT_PRETTY_SQL** – пожелание получения текста DML-операции в отформатированном виде (несколько строк вместо одной).

  Совершенно ненужная опция. Не используется.

- **NO_ROWID_IN_STMT** – не указывать орakловое значение rowid в тексте DML-операции. Полезная опция, исключающая упоминание rowid в SQL-предложениях, текст которых запрашивается из `v$logmnr_contents.sql_redo`. Поскольку уникальные идентификаторы строк в Oracle и Postgres не совпадают, то нет надобности помещать rowid в Postgres и нет надобности сравнивать значения rowid. Наличие rowid в SQL-предложениях будет только вредить.

Принято решение использовать опцию `dbms_logmnr.NO_ROWID_IN_STMT` при запуске логманера средствами `ora2pgsync`.

- **DDL_DICT_TRACKING** - не допускать отсутствия архивных лог-файлов в списке зарегистрированных в логмайнере для запрошенного времени или диапазона SCN. Если вызов `dbms_logmnr.start_logmnr()` завершается неудачно, можно запросить столбец `STATUS` в представлении `v$logmnr_logs`, чтобы определить, какие архивные redo-лог файл в списке. Затем можно найти и вручную добавить эти файлы в LogMiner и попытаться снова вызвать `dbms_logmnr.start_logmnr()`.

Эта опция перекликается с недостатками, которые описаны в пункте про опцию COMMITED_DATA_ONLY. Всё равно для понимания границ диапазона SCN делать лишний запрос к `v$logmnr_contents`. Всё равно добавлять недостающие лог-файлы после чего стартовать логмайнер. И всё равно делать второй запрос к `v$logmnr_contents` за корректными результатами.

Решено не использовать данную опцию.

1.2.3. Команда для запуска Oracle LogMiner

Подводя итоги анализа в предыдущем пункте, запуск/старт логмайнера средствами `ora2pgsync` выполняется командой:

```
begin
  dbms_logmnr.start_logmnr(Options => dbms_logmnr.DICT_FROM_ONLINE_CATALOG +
                                      dbms_logmnr.NO_ROWID_IN_STMT
                           );
end;
```

2. Подготовка к миграции

2.1. Настройка логирования в Oracle

2.1.1. Режим архивирования логов

Этот режим нужен для получения целостной картины последовательности действий в базе данных. Результаты логирования DML-операций будут использоваться в процессе выполнения инкрементальной миграции данных.

Включён ли этот режим можно узнать, выполнив запрос в БД Oracle:

```
SQL> SELECT LOG_MODE FROM V$DATABASE;
```

Возможные значения:

- ARCHIVELOG – режим архивирования логов включён.
- NOARCHIVELOG – режим архивирования логов выключен.

Если включён режим ARCHIVELOG, то можно посмотреть список всех параметров, связанных с архивируемыми логами (места нахождения, интервалы и прочее), выполнив следующую команду в sqlplus:

```
SQL> SHOW PARAMETER LOG;
```

Ещё одна команда, показывающая детальную информацию (в том числе и об установленном месте хранения архивов), представлена ниже. Её следует выполнять от имени пользователя с правами DBA:

```
SQL> ARCHIVE LOG LIST;
```

2.1.2. Включение режима архивирования логов

Если режим архивирования логов не активен, то его следует включить.

а) Вход в sqlplus администратором БД на сервере Oracle.

```
sqlplus /nolog
SQL> connect / as sysdba
```

б) Указание места для хранения и формата именования лог-файлов, если ещё не указано. Например:

```
SQL> ALTER SYSTEM SET log_archive_dest_1=
'location=/opt/oracle/product/19.3/dbs/arch' SCOPE=spfile;
SQL> ALTER SYSTEM SET log_archive_format='%t_%s_%r.arc' SCOPE=spfile;
```

в) Выполнение последовательности команд:

```
SQL> SHUTDOWN IMMEDIATE;
SQL> STARTUP MOUNT;
```

```
SQL> ALTER DATABASE ARCHIVELOG;
```

г) Включение логирования на уровне базы данных. В логах нужны значения PK, чтобы узнать с какой строкой/записью в таблице производилась манипуляция.

```
SQL> ALTER DATABASE ADD SUPPLEMENTAL LOG DATA (PRIMARY KEY) COLUMNS;
```

д) Завершение настроек:

```
SQL> ALTER DATABASE OPEN;
SQL> EXIT
```

Проверить, включено ли логирование на уровне базы данных, можно так:

```
SQL> SELECT SUPPLEMENTAL_LOG_DATA_MIN FROM V$DATABASE;
```

2.1.3. Управление размерами архивных лог-файлов.

При необходимости изменения размера формируемых архивных лог-файлов следует выполнить ряд приведенных ниже команд. Выполнять следует от имени пользователя с правами DBA. В перечне команд приводится пример установки размера лог-файлов равным 20 Мбайт.

а) Временное добавление 3х новых групп логов (4,5,6):

```
SQL> alter database add logfile;
  group 4 ('/opt/oracle/oradata/DB/onlinelog/o1_mf_4_redo20mb.log') size 20m,
  group 5 ('/opt/oracle/oradata/DB/onlinelog/o1_mf_5_redo20mb.log') size 20m,
  group 6 ('/opt/oracle/oradata/DB/onlinelog/o1_mf_6_redo20mb.log') size 20m;
```

б) Несколько раз, пока 3 первых группы не станут INACTIVE (см. “запросы для проверки состояния логов”), следует выполнить:

```
SQL> alter system switch logfile;
```

Таким образом, одна из 3 новых групп (4,5,6) должна стать CURRENT.

в) Фиксация SCN.

```
SQL> alter system checkpoint;
```

г) Следует удалить те группы, которые были ранее (с другим размером файлов). После этого надо убедиться, что самих файлов не осталось (удалить или переместить старые файлы).

```
SQL> alter database drop logfile group 1, group 2, group 3;
```

д) Добавление 3х новых групп (1,2,3) с файлами нужных размеров вместо удалённых:

```
SQL> alter database add logfile
  group 1 ('/opt/oracle/oradata/DB/onlinelog/o1_mf_1_redo20mb.log') size 20m,
  group 2 ('/opt/oracle/oradata/DB/onlinelog/o1_mf_2_redo20mb.log') size 20m,
  group 3 ('/opt/oracle/oradata/DB/onlinelog/o1_mf_3_redo20mb.log') size 20m;
```

е) Несколько раз, пока 3 последних группы не станут INACTIVE (см. “запросы для проверки состояния логов”).

```
SQL> alter system switch logfile;
```

Таким образом, одна из 3 новых групп (1,2,3) должна стать CURRENT.

ж) Удаление групп 4,5,6. После этого надо убедиться, что самих файлов не осталось (удалить или переместить старые файлы).

```
SQL> alter database drop logfile group 4, group 5, group 6;
```

з) Проверка результатов выполненных команд.

Ниже приведены “запросы для проверки состояния логов”:

```
SQL> select group#, status, bytes/1024/1024 mb from v$log;
SQL> select * from v$log;
SQL> select * from v$logfile;
```

2.2. Подготовка к работе с Oracle LogMiner

2.2.1. Создание пользователя в Oracle и назначение ему необходимых прав для работы с Oracle LogMiner

Для создания пользователя следует выполнить следующую команду от имени пользователя с правами DBA. Имя пользователя можно указать любое незанятое. В приведённом примере создаётся пользователь с именем logmnr и таким же паролем.

```
SQL> CREATE USER logmnr IDENTIFIED BY logmnr;
```

Назначение необходимых прав выполняется командой:

```
SQL> GRANT CONNECT, SELECT_CATALOG_ROLE, EXECUTE_CATALOG_ROLE, LOGMINING TO logmnr;
```

3. Программа ora2pgsync

3.1. Общее описание принципов работы

Программа `ora2pgsync` реализована на языке Java. Протестирована работа в Java8 и Java17.
При запуске программы активируются 2 потока обработки.

В рамках одного из них выполняется:

- подключение к БД Oracle;
- периодический запрос и выявление новых необработанных архивных файлов с redo-логами;
- добавление этих файлов в список обрабатываемых логмайнером (Oracle LogMiner);
- запуск логмайнера;
- получение перечня выполненных в Oracle DML-команд, а также фактов начала транзакций и завершения транзакций (commit/rollback);
- помещение каждой такой команды в java-очередь на обработку с данными о номере оракловой транзакции и идентификаторами ораклового сеанса, в котором выполнена команда.

В рамках второго потока обработки выполняется:

- периодическое опрашивание java-очереди на предмет появления в ней новых команд;
- при обработке очередной DML-команды преобразование ораклового текста этой команды с заменой вызываемых функций на аналогичные для применения/выполнения в Postgres с учётом атрибутов таблицы и её столбцов, указанных в структуре json в конфигурационном файле;

- создание новой пользовательской сессии в Postgres (подключение к БД), помеченной в java идентификатором ораклового сеанса, если таковая отсутствует;
- создание и контроль в рамках этой сессии начала/завершения транзакции, помеченной номером транзакции, в которой выполнялась команда в Oracle;
- выполнение DML-команды или commit/rollback в БД PostgreSQL;
- периодическое удаление сессий в Postgres, созданных из java-приложения `ora2pgsync` с пометкой об идентификаторе соответствующей оракловой сессии, если такая сессия уже отсутствует в Oracle, и в рамках этой сессии в Postgres нет открытых транзакций;
- повтор выполнения приведенных выше пунктов, пока java-очередь не пуста, затем снова периодическое опрашивание очереди.

Команды в Postgres выполняются ровно в том же порядке, в каком их выполнение зафиксировано в архивных логах Oracle.

В процессе работы `ora2pgsync` формируется журнал выполненных действий. Насколько подробным будет этот журнал, указывается в конфигурационном файле в атрибуте “loglevel”.

3.2. Структура каталогов, описание файлов

- Программа `ora2pgsync` представлена в виде jar-файла (ora2pgsync.jar).
- К ней прилагается набор используемых java-библиотек. Библиотеки расположены в каталоге /lib рядом с ora2pgsync.jar.
- Также поставляется пример заполненного конфигурационного файла (ora2pgsync.json), содержащего необходимые атрибуты для работы `ora2pgsync`. Этот файл размещён в каталоге /config рядом с ora2pgsync.jar.
- Опционально в каталоге /config может находиться файл ora2pgsync.scn, содержащий сведения о максимальном значении SCN в последнем обработанном архивном файле с redo-логами Oracle. Если такой файл отсутствует, то он будет создан при первом запуске `ora2pgsync`.

Места расположения, а также имена, файлов *.json и *.scn указываются при запуске `ora2pgsync` в качестве атрибутов командной строки.

3.3. Конфигурационный файл. Описание и настройка

Содержимое файла представлено в формате JSON (JavaScript Object Notation). Указываются значения как основных параметров, необходимых для работы `ora2pgsync`, так и значения параметров/атрибутов, определённых для каждой таблицы и при необходимости даже для конкретных столбцов, задействованных в инкрементальной миграции данных.
Пример содержимого конфигурационного json-файла представлен ниже:

```
{
  "oradb": "192.168.1.1:1521/oradb",
  "orausername": "logmnr",
  "orapassword": "logmnr",
  "pgdb": "192.168.1.2:5432/postgres",
  "pgusername": "ora2pg_user",
  "pgpassword": "ora2pg_passwd",
  "loglevel": "info",
  "work": [
    {
      "schema": "book",
      "table": "o2p_t_author"
    },
    {
      "schema": "book",
      "table": "o2p_t_book"
    },
    {
      "schema": "book",
      "table": "o2p_t_book_view_log"
    },
    {
      "schema": "book",
      "table": "o2p_t_genre"
    },
    {
      "schema": "book",
      "table": "o2p_t_publisher"
    },
    {
      "schema": "book",
      "table": "o2p_t_abcd",
      "target_schema": "book",
      "target_table": "o2_p_t_abcde",
      "fields": [
        {
          "source": "decimal_fld",
          "target": "decimal_fld"
        },
        {
          "source": "raw_fld",
          "target": "raw_fld"
        },
        {
          "source": "flag_v",
          "target": "flag_b",
          "value_template": "case when $value = 'Y' then true else false end"
        }
      ]
    }
  ]
}
```

Описание основных параметров:

- “oradb” – строка подключения к БД Oracle, являющейся источником данных при выполнении миграции. Важно, чтобы в этой базе данных было включен режим архивирования redo-логов.
- “orausername” – имя пользователя БД Oracle, от имени которого будет выполняться подключение. Как правило, это пользователь, у которого есть необходимые права для использования Oracle LogMiner.
- “orapassword” – пароль пользователя БД Oracle.
- “pgdb” - строка подключения к БД PostgreSQL, являющейся приёмником данных при выполнении миграции.
- “pgusername” – имя пользователя БД PostgreSQL, который имеет доступ к перечисленным в массиве “work” таблицам.
- “pgpassword” – пароль пользователя БД PostgreSQL.
- “loglevel” – уровень логирования работы приложения ora2pgsync. Возможные значения от самого краткого к наиболее подробному логированию: “severe”, “warning”, “info”, “config”, “finest”. По умолчанию, если не указано значение для этого параметра, используется уровень “info”. В журнал будут попадать все сообщения об ошибках, предупреждения, информационные сообщения. Если указать уровень “finest”, то в журнал выполнения программы, в том числе, попадёт каждая DML-операция. Уровень “finest” рекомендуется

указывать при первоначальной тестовой настройке миграции для 1-2 таблиц, если есть необходимость понять, почему миграция не выполняется или выполняется некорректно.

- “work” – массив элементов структуры JSON, каждый из которых определяет атрибуты миграции для конкретной таблицы. Является обязательным. Другими словами, обязательно должен быть определён перечень таблиц, участвующих в инкрементальной миграции. Возможное содержимое отдельного элемента этого массива представлено ниже:
  - “schema” – имя схемы БД Oracle, в которой располагается таблица-источник данных. Обязательный атрибут.
  - “table” – имя таблицы БД Oracle, являющейся источником данных при выполнении миграции. Обязательный атрибут.
  - “target_schema” – имя схемы БД PostgreSQL, в которой располагается таблица-приёмник данных. Необязательный атрибут. Если отсутствует, то по умолчанию значение совпадает со значением атрибута “schema”.
  - “target_table” - имя таблицы БД PostgreSQL, являющейся приёмников данных при выполнении миграции. Необязательный атрибут. Если отсутствует, то по умолчанию значение совпадает со значением атрибута “table”.
  - “fields” - массив элементов структуры JSON, каждый из которых определяется атрибуты миграции для конкретного столбца. Этот массив не является обязательным для таблицы. По умолчанию изменение данных в любом столбце указанной таблицы Oracle приводит к аналогичному изменению данных в одноимённом столбце указанной таблицы Postgres. Но если задан список столбцов в массиве “fields”, то отслеживаются изменения только в перечисленных столбцах. Возможное содержимое отдельного элемента этого массива представлено ниже:
    - “source” – имя столбца в исходной таблице Oracle. Обязательный атрибут.
    - “target” – имя столбца в соответствующей таблице Postgres. Необязательный атрибут. Если отсутствует, то по умолчанию значение совпадает с значением атрибута “source”.
    - “value_template” – шаблон, используемый при необходимости установить определённое значение столбца таблицы Postgres в зависимости от значения соответствующего столбца таблицы Oracle. Необязательный атрибут. Если отсутствует, то значение указанного столбца таблицы Postgres по умолчанию совпадает со значением столбца таблицы Oracle.
      Этот атрибут выгодно использовать, если тип столбца в таблице Postgres существенно изменён по сравнению с аналогичным столбцом таблицы Oracle. Например, если в Oracle это был столбец типа `varchar2(1)` с возможными значениями Y/N, в Postgres этот столбец стал типа `boolean` с возможными значениями true/false. Тогда можно указать значение для “value_template” такое как на представленном выше рисунке с примером. При присвоении значения столбцу `flag_b` типа `boolean` в Postgres, орaкловое значение столбца `flag_v` типа `varchar2(1)` будет подставлено в выражение вместо `$value`, что позволит определить булевое значение для столбца в Postgres.

3.4. Файл SCN

Представляет собой обычный текстовый файл, содержащий одну строку с максимальным номером SCN операции, о которой есть данные внутри последнего обработанного программой

ora2pgsync архивного redo-лог файла. Файл используется в служебных целях для выявления ещё необработанных архивных файлов в случае их появления.

Если файл отсутствует или не содержит сведений о SCN, то ora2pgsync вместо обработки очередного архивного файла просто сохранит в “файл SCN” значение поля next_change# представления v$archived_log.

Если “значение SCN” в файле указано, то ora2pgsync сначала выполнит обработку всех новых архивных файлов (в информации о которых в представлении v$archived_log в поле first_change# значение больше либо равное “значению SCN” из scn-файла), а затем перезапишет значение последнего обработанного SCN в scn-файл.

3.5. Технология и алгоритм миграции. От и до

Как уже упоминалось ранее, программа ora2pgsync создана на языке Java.

В качестве исходных данных для инкрементальной миграции используются записи о выполненных изменениях данных (insert/update/delete/commit/rollback), так называемые redo-логи, создаваемые Oracle и размещаемые в архивных файлах.

Следует обратить внимание, что оперативные файлы с redo-логами не обрабатываются программой. И следует обратить внимание, что из redo-логов извлекаются данные о выполненных DML-операциях.

Все выполненные DDL-операции игнорируются, т.к. одним из условий инкрементальной миграции является полное отсутствие DDL-операций. Из DDL-операций допускается лишь создание партиций таблиц Oracle. Такие операции обязательно должны быть продублированы в Postgres без участия ora2pgsync в этом процессе.

Первыми действиями, которые после старта выполняет программа ora2pgsync, являются чтение конфигурационного json-файла и подключение к БД Oracle через драйвер jdbc. Сразу же выполняется проверка, включено ли логирование на уровне базы данных, и установлен ли режим архивирования логов. Если нет, на этом работа ora2pgsync завершается.

Активируются два потока обработки данных, так называемые, “поток-извлекатель” сведений о выполненных DML-командах, изменивших данные в таблицах Oracle, и “поток-исполнитель” аналогичных DML-команд, изменяющих данные в таблицах Postgres.

3.5.1. Извлечение сведений о DML-командах, выполненных в Oracle

Очередная итерация запускается периодически каждые 5 секунд.

Для извлечения сведений о выполненных в Oracle DML-командах используется Oracle LogMiner.

Перед его запуском, путём выполнения запроса к представлению v$archived_log определяется список необработанных архивных лог-файлов. Необработанными считаются файлы, в записи о которых значение поля first_change# больше либо равно значению, сохранённому в scn-файле. Если такого файла нет, или он пуст, то в этот файл записывается значение поля next_change# самого последнего на момент проверки архивного файла. И программа переходит к ожиданию появления нового архивного лог-файла.

Все новые архивные файлы добавляются в перечень файлов, обрабатываемых логмайнером, и выполняется запуск логмайнера.

После того, как логмайнер стартовал, становится доступным представление `v$logmnr_contents`, содержащее сведения о выполненной команде для изменения данных в таблице Oracle, а также сведения о транзакции и идентификаторе сессии Oracle, в рамках которых выполнена эта DML-команда.

Извлеченные данные о DML-командах следуют строго в той же последовательности, в которой они зафиксированы в архивных логах и выполнялись в Oracle. В этой же последовательности они добавляются в очередь команд, реализованную средствами java в приложении `ora2pgsync`. К этой очереди постоянно обращается другой поток, подготавливающий и исполняющий аналогичные команды в Postgres.

После завершения обработки всех новых файлов максимальное значение SCN обработанных записей записывается в scn-файл.

Для извлечения данных о выполненных в Oracle командах всегда используется одно-единственное соединение с БД Oracle. Это соединение устанавливается от имени пользователя, имеющего все необходимые права для работы с Oracle LogMiner. Атрибуты доступа к базе данных, а также имя и пароль пользователя, указаны в конфигурационном файле.

В процессе извлечения данных содержимое поля `sql_redo` представления `v$logmnr_contents` “разбирается” (анализируется) посимвольно и раскладывается в структуру таблица-столбец, которая также помещается в очередь команд в java в составе элемента очереди. В сведения о столбцах записываются значения “до” и “после”. Значения “до” в дальнейшем используются при конструировании фразы where в процессе подготовки команды update/delete для выполнения в Postgres. Значения “после” используются при конструировании фразы set в процессе подготовки команды update и фразы values при подготовке команды insert для выполнения в Postgres.

Следует обратить внимание, что во фразе where в данных, предоставляемых логмайнером, всегда перечисляются значения всех столбцов таблицы. При реконструировании sql-текста выполненной DML-операции, LogMiner использует все данные, которые есть в redo-логе. Это выгодно для определения конфликтов, когда в БД PostgreSQL строка с таким первичным ключом есть, но текущее значение столбца в изменяемой/удаляемой записи не совпадает со старым значением в БД Oracle.

3.5.2. Выполнение DML-команд в Postgres

Принцип таков. В Postgres выполняются все DML-операции в интересующих таблицах, которые были выполнены в Oracle, и о которых на текущих момент имеются сведения в оракловых архивных логах, независимо от того, прошёл ли в Oracle commit/rollback для них, или он ожидается в будущем.

Очередная итерация новой обработки очереди запускается периодически каждые 5 секунд после того, как очередь DML-команд на выполнение полностью очищена. Поток обращается за очередным элементом к очереди DML-команд, формирующейся другим потоком, извлекающим данные о выполненных командах из redo-лог файлов Oracle.

Каждый элемент очереди содержит сведения о типе выполненной операции, как они представлены в `v$logmnr_contents`. Перечень обрабатываемых операций содержит следующие команды: START, INSERT, UPDATE, DELETE, COMMIT, ROLLBACK. Операция типа START несёт в себе признак начала транзакции в Oracle. Помимо этого, в элементе очереди содержится структура таблица-столбец, упомянутая ранее, и сведения об идентификаторе орацловой сессии и транзакции, в рамках которых выполнена операция. С использованием этих данных подготавливается DML-команда и затем выполняется в Postgres.

Для выполнения DML-команды устанавливается соединение с Postgres. Атрибуты доступа к базе данных, а также имя и пароль пользователя, указаны в конфигурационном файле. Созданное соединение внутри `ora2pgsync` помечается идентификатором орацловой сессии. Если соединение с таким идентификатором уже есть, то используется существующее. Если нет, то создаётся новое. Можно сказать, что соединений с Postgres будет столько, сколько сессий в Oracle вносили изменения, известные на момент обработки очередного архивного файла. В идеале, в рамках сессии с Postgres операции выполняются в такой последовательности:

- открывается транзакция;
- выполняются DML-операции в рамках неё;
- транзакция завершается командой commit или rollback;
- затем следующая транзакция и т.д.

В рамках одной Postgres сессии в `ora2pgsync` не может начаться новая транзакция, если предыдущая не закрыта. Однако возникают ситуации, когда в Oracle в рамках одной сессии могут быть открыты одновременно более одной транзакции. Так бывает при создании автономных транзакций. В `ora2pgsync` для обработки таких случаев каждая главная Postgres-сессия изначально создаётся в некотором наборе сессий, помеченном идентификатором орацловой сессии. Как правило, каждый набор обычно содержит одну сессию (главную). Но если необходимо повторить автономную орацловую транзакцию со всеми командами внутри неё, то в наборе создаётся новая дополнительная Postgres-сессия, в ней полностью выполняются все команды такой транзакции, и эта дополнительная сессия с Postgres сразу же завершается. Если в Oracle в рамках сессии одновременно было несколько автономных транзакций, то в `ora2pgsync` будет создано несколько таких дополнительных сессий в рамках одного набора, и команды каждой транзакции будут выполнены в рамках своей дополнительной Postgres-сессии.

Каждый раз, когда обрабатывается новый архивный файл с redo-логами, `ora2pgsync` запрашивает в Oracle сведения о “живых” сессиях на момент обработки. Эти сведения используются для завершения сеансов в Postgres, входящих в наборы, помеченные идентификаторами уже несуществующих сессий Oracle. Таким образом, до тех пор, пока сессия в Oracle ещё существует (“жива”), соответствующая ей сессия будет сохраняться и Postgres, если она была создана для выполнения DML-команды. После того, как в Oracle сессия завершается, она будет завершена и в Postgres.

Вполне вероятно, что после обработки очередного архивного файла, в Postgres останутся сессии с открытыми транзакциями, созданные с использованием JDBC из `ora2pgsync`, т.к. сведения о завершении этих транзакций ещё не попали в орацловые логи. Или даже орацловая транзакция является настолько длинной, что она всё ещё не завершилась, но орацловые логи уже содержат сведения об изменениях данных, выполненных в рамках этой транзакции. В такой

ситуации нет ничего страшного. Эти транзакции будут завершены при обработке одного из следующих архивных файлов Oracle с redo-логами. А до тех пор, Postgres сессии будут находиться в ожидании новых DML-команд, которые должны быть выполнены в рамках незакрытой транзакции или команд завершения/отката транзакции.

Выполнение в Oracle массовых DML-операций update/delete отражается в redo-логах Oracle как выполнение update/delete для каждой записи таблицы.
Например, при выполнении в Oracle …

```
update table1 set notes = 'note '||id where id between 1 and 100000;
```

… в лог попадёт 100000 записей о том, что для каждого значения поля `id` был выполнен соответствующий update значения поля `notes`.
И если в Oracle потребовалась всего лишь одна общая команда update, то в Postgres понадобится 100000 отдельных команд для каждой записи. Это очень сильно замедляет выполнение инкрементальной миграции в таких случаях. Для ускорения выполнения в `ora2pgsync` одинаковых команд update/delete (и даже insert), но с разными значениями, указываемыми для полей таблицы, используется так называемая batch-технология в Java. Выполняемое sql-предложение (statement) подготавливается (prepare) и анализируется (parse) один раз. А затем партиями передаются наборы значений параметров, устанавливающих значения столбцов в DML-командах. И периодически (после обработки каждого 1000-го набора параметров или последнего набора параметров) выполняется statement. Получается нечто похожее на bulk collect в Oracle. Это существенно снижает нагрузку на сеть и повышает скорость выполнения команд при массовых DML-операциях.

3.6. Запуск

3.6.1. Параметры, указываемые при запуске

Пример команды запуска программы ora2pgsync представлен ниже:

java -Xmx4096m -Dfile.encoding=UTF-8 -jar ora2pgsync.jar ./config/ora2pgsync.json ./config/ora2pgsync.scn > ora2pgsync.log 2>&1

Основные параметры, указываемые при запуске:

- -Xmx4096m – выделяемая оперативная память;
- -Dfile.encoding=UTF-8 – используемая кодировка;
- -jar ora2pgsync.jar – имя выполняемого jar-файла;
- ./config/ora2pgsync.json – путь и имя конфигурационного json-файла
- ./config/ora2pgsync.scn- путь и имя scn-файла.

Выполняется перенаправление выходных потоков в файл ora2pgsync.log.

3.6.2. Рекомендации.

Перед стартом инкрементальной миграции, выполняемой средствами `ora2pgsync`, рекомендуется выполнить приведённые ниже действия:

- Выполнить подготовку Oracle к миграции, включающую в себя настройку логирования в Oracle, создание пользователя Oracle и назначение ему прав, необходимых для работы с Oracle

LogMiner. Рекомендуется задуматься над возможностью и необходимостью уменьшения размеров архивных файлов. Чем меньше размер файла, тем быстрее будут извлекаться из него данные при инкрементальной миграции. Процесс извлечения данных из архивных файлов занимает существенное время. Необходимость изменения размеров архивных файлов может быть определена по результатам выполнения тестовой инкрементальной миграции. Например в случае, когда синхронизация изменений в Postgres не поспевает за выполненными изменениями в Oracle.

- Настроить конфигурационный json-файл, указав в нём атрибуты подключения к базам данных Oracle и PostgreSQL, перечислив в нём все необходимые таблицы и столбцы (при надобности), указав уровень логирования.
- Если это первый запуск `ora2pgsync` после выполнения основной миграции данных:
  - Обязательно в правах пользователя DBA выполнить в Oracle команду:
    `SQL> alter system switch logfile`
  - После этого в представлении `v$archived_log` найти запись о самом последнем созданном архивном файле и значение поля `first_change#` этой записи поместить в единственную строку scn-файла. Таким образом, устанавливается исходное значение SCN, начиная с которого следует выполнять обработку оракловых архивных файлов с redo-логами.

4. Ограничения

Самое важное ограничение – это инкрементальная миграция в системах с большим количеством массовых DML-операций. Ввиду приведённых ранее особенностей выбранной технологии при выполнении массовых DML-операций следует отметить, что чем больше в системе таких массовых DML, тем сложнее программе `ora2pgsync` с ними справиться. Тем сложнее с небольшим запозданием выполнить синхронизацию данных в Postgres. Это займёт гораздо больше времени в Postgres (приблизительно в 2-3 раза больше при массовом изменении/удалении 100 000 записей), чем в Oracle.

Массовые DML – это стандартная проблема любой репликации, по крайней мере, это касается Oracle. И ни в одной системе репликации (в том числе, Advanced Replication, Streams, GoldenGate) она не решена должным образом.

Можно реализовать свою систему отслеживания массовых DML в Oracle и последующего их выполнения в Postgres, что потребует модификации кода в исходной БД Oracle. Но редкий Заказчик позволит это сделать. И это совсем другой подход, который не реализован в настоящий момент в `ora2pgsync` и не рассматривается в данном документе.