Система автоматизации миграции прикладных программных систем с СУБД Oracle в СУБД PostgreSQL (CAM LUI4ORA2PG)

Методика миграции

Всего страниц 14

Оглавление

Введение

Документ предназначен для специалистов, перед которыми поставлена задача миграции программного комплекса, состоящего из базы данных на Oracle и интерфейса к ней (пользовательского и программного), а также выходных форм, в БД Postgres.

Средство учёта процесса миграции

Для учёта процесса миграции предлагается применение программного продукта LUI4ORA2PG, разработанного Форс-Телеком.

Участники процесса миграции и их задачи

Системный программист

Системный программист – подготовка стенда с БД Postgres, установка БПО и LUI4ORA2PG, организация доступа к Oracle и Postgres из LUI4ORA2PG, установка обновлений БПО и LUI4ORA2PG.

Администратор LUI4ORA2PG

Администратор LUI4ORA2PG – управление пользователями, группами пользователей, правами доступа к элементам интерфейса LUI4ORA2PG, возможностями участия в проектах миграции. Сам не является участником проектов миграции.

Руководитель проекта миграции

Руководитель проекта миграции – создание проекта миграции, добавление участников проекта, назначение и переназначение ответственных за миграцию отдельных объектов из числа участников проекта, контроль состояния процесса миграции, прогноз продолжительности миграции, выполнение автомиграции, формирование скриптов для применения в Postgres, исполнение скриптов. Одномоментно у проекта может быть только один руководитель. Но один и тот же пользователь LUI4ORA2PG может быть руководителем нескольких проектов.

Участник проекта миграции

Участник проекта миграции – модификация скриптов, полученных в результате автомиграции, управление состоянием объекта Postgres, сохранение промежуточных результатов модификации скриптов.

Порядок работы с проектом миграции

Руководитель проекта миграции:

1) Делает запрос Администратору LUI4ORA2PG на добавление в систему его (руководителя проекта) и участников проекта.
2) Создаёт проект и автоматически становится его руководителем. Он назначает уровень миграции, указывает дату начала над проектом, добавляет в проект миграции соединения с Oracle и Postgres.
3) Добавляет в проект объекты Oracle, подлежащие миграции. Он может добавлять их, используя соединение с Oracle или произвольный. Для объектов Oracle, полученных из БД, генерирует DDL-скрипты, выполняемые в Oracle.
4) Запускает автомиграцию в режиме получения оценочного отчёта.
5) Получив отчёт, прогнозирует и выставляет в проекте плановую дату завершения проекта миграции.
6) Запускает автомиграцию в режиме получения Postgres-скриптов для выбранных типов объектов Oracle. В процессе автомиграции объекты Oracle анализируются на наличие ограничений, из-за которых возникает необходимость ручной правки скриптов, полученных автоматически.
7) Исходя из своих знаний проекта на Oracle, распределяет выполнение полученных объектов Postgres по этапам. По мере работы над проектом, состав этапов миграции и порядок их выполнения руководитель проекта может изменять. Этапы миграции должны следовать в определённой последовательности, так, чтобы при последующем применении скриптов в Postgres не возникло ошибок выполнения из-за отсутствия объектов, которые должны быть созданы ранее.
8) назначает ответственных за объекты Postgres из числа участников проекта.

Далее ответственный участник проекта:

1) Анализирует скрипт для объекта в Oracle и скрипт, полученный при автомиграции.
2) Принимает решение о необходимости модификации скрипта Postgres.
3) Если модификация скрипта требуется, изменяет статус объекта Postgres, принимая его в работу.
4) Приступает к модификации объекта. Подстветка элементов кода в LUI4ORA2PG позволяет выявить синтаксические ошибки в скрипте.
5) Если не может справиться с ручной правкой скриптов, докладывает об этом руководителю проекта, и руководитель проекта назначает другого ответственного.
6) Когда скрипт, поправленный вручную, готов, ответственный указывает соответствующий статус объекта Postgres.

Далее руководитель проекта:

1) Контролирует готовность всех объектов Postgres для этапа миграции.
2) Назначает и переназначает ответственных за объекты Postgres.
3) Когда ручная работа над всеми объектами Postgres завершена, генерирует скрипт для применения в Postgres для всех объектов определённого этапа. В скрипте команды создания объектов перечисляются в том порядке, который указан в списке объектов Postgres.
4) Применяет скрипт для этапа в БД Postgres (соответственно, после выполнения скриптов для более ранних этапов).
5) При наличии ошибок, анализирует их и принимает решение об изменении состава и порядка следования этапов, доработке объектов Postgres, назначив новых ответственных за инвалидные объекты.
6) Перемещает объекты Postgres на другой этап при необходимости.
7) Добавляет объекты Postgres вручную, если они не сгенерировались при автомиграции.
8) Добавляет/удаляет этапы.

Процесс работы над проектом миграции повторяется итеративно до получения удовлетворительного результата.

Получив удовлетворительный результат применения скриптов для всех этапов в Postgres, руководитель проекта проставляет у проекта миграции состояние и дату фактического завершения проекта.

Особенности миграции объектов

Пользователи

Права на DBA-представления

Пользователь, выполняющий автомиграцию пользователей Oracle, должен обладать правами на DBA-представления:

- DBA_TAB_PRIVS,
- DBA_OBJECTS,
- DBA_COL_PRIVS,
- DBA_SYS_PRIVS,
- DBA_ROLE_PRIVS,
- DBA_USERS.

Предоставление прав выполняется командами:

grant select on dba_tab_privs to <user>;

grant select on dba_objects to <user>;

grant select on dba_col_privs to <user>;

grant select on dba_sys_privs to <user>;

grant select on dba_role_privs to <user>;

grant select on dba_users to <user>.

Необходимые права имеет также пользователь с ролью DBA. Предоставление роли выполняется командой:

grant dba to <user>;

Права пользователя на объекты мигрируемой схемы

Автомиграция пользователей выполняется только в том случае, если пользователь Oracle предоставил явные права другим пользователям или ролям на объекты мигрируемой схемы. Поэтому для выполнения успешной автомиграции необходимо выполнить одну из рекомендаций:

- предоставить мигрируемому пользователю явные права хотя бы на один объект мигрируемой схемы;
- добавить в проект мигрируемую схему, на объекты которой у мигрируемого пользователя есть явные права, и выполнять автомиграцию этой схемы;
- добавить объект Postgres типа USER вручную, написав для него требуемые команды создания и предоставления прав.

Пользователь с паролем

В Postgres-скрипте создания пользователя необходимо вручную указать пароль.

Выполнение скрипта миграции пользователя в Postgres

Скрипт создания пользователя в Postgres должен выполняться суперпользлователем.

Роли

Права на DBA-представления

Пользователь, выполняющий автомиграцию, должен обладать правами на представления:

- DBA_TAB_PRIVS,
- DBA_OBJECTS,
- DBA_COL_PRIVS,

- DBA_SYS_PRIVS,

- DBA_ROLE_PRIVS,

- DBA_USERS.

Предоставление прав выполняется командами:

grant select on dba_tab_privs to <user>;

grant select on dba_objects to <user>;

grant select on dba_col_privs to <user>;

grant select on dba_sys_privs to <user>;

grant select on dba_role_privs to <user>;

grant select on dba_users to <user>.

Необходимые права имеет также пользователь с ролью DBA. Предоставление роли выполняется командой:

grant dba to <user>;

Права роли на объекты мигрируемой схемы

Автомиграция ролей выполняется только в том случае, если пользователь Oracle предоставил явные права этим ролям на объекты мигрируемой схемы. Поэтому для выполнения успешной автомиграции необходимо выполнить одну из рекомендаций:

- предоставить мигрируемой роли явные права хотя бы на один объект мигрируемой схемы;
- добавить в проект мигрируемую схему, на объекты которой у роли есть права, и выполнять автомиграцию этой схемы;
- добавить объект Postgres типа ROLE вручную, написав для него требуемые команды создания и предоставления прав.

Роль с паролем

В Postgres-скрипте создания роли необходимо вручную указать пароль.

Выполнение скрипта миграции роли в Postgres

Скрипт создания роли в Postgres должен выполняться суперпользователем.

Табличные пространства

Автомиграция выполняется только для тех табличных пространств, в которых хранятся объекты мигрируемой схемы:

- таблицы;
- индексы;
- LOB-ы;
- секции секционированных таблиц;
- секции индексов секционированных таблиц.

Если требуется миграция табличного пространства, в котором не хранятся перечисленные объекты, то необходимо выполнить одно из перечисленных ниже действий:

- разместить в мигрируемом табличном пространстве таблицу, создать произвольную, добавить её в проект миграции, выполнить автомиграцию табличного пространства, таблицу удалить;
- переместить любую таблицу в мигрируемое табличное пространство, выполнить автомиграцию табличного пространства, переместить таблицу в прежнее табличное пространство, выполнить автомиграцию таблиц;
- добавить объект Postgres типа TABLESPACE вручную, написав для него требуемые команды создания и предоставления прав.

Такая миграция может потребоваться, если в результате миграции проекта планируется изменить место хранения какого-либо объекта.

Выполнение скрипта миграции табличного пространства в Postgres

Скрипт создания табличного пространства в Postgres должен выполняться суперпользователем. Перед выполнением скрипта следует убедиться, что указанное в скрипте расположение файла табличного пространства существует.

Таблицы

Временная таблица

Временные таблицы не автомигрируются. Несмотря на то, что в Postgres также, как в Oracle, существуют временные таблицы, их реализация и применение существенно различаются. Поэтому система не может автоматически принять решение о том, как преобразовывать данный объект.

Преобразование временной таблицы в обычную хранимую таблицу

Для выполнения автомиграции можно создать аналогичную хранимую таблицу, добавить её в проект, выполнить автомиграцию таблиц, а в полученном Postgres-скрипте изменить названия объектов на требуемые.

В программах, использующих временную таблицу, следует предусмотреть удаление добавленных данных, по завершении работы с таблицей.

Без автомиграции можно вручную добавить объект Postgres типа TABLE и написать скрипт для её создания.

Преобразование временной таблицы во временную таблицу

Автомиграция не выполняется. Требуется модификация хранимых программ и анонимных PL/SQL-блоков, использующих временную таблицу. В них следует предусмотреть команды создания временной таблицы.

Типы столбцов таблиц и переменных в хранимом коде

Типы столбцов и переменных в хранимом коде преобразуются следующим образом:

Тип в Oracle	Тип в Postgres
varchar2	varchar
number	numeric
date	timestamp
long	text
long raw	bytea
clob	text
nclob	text
blob	bytea
raw	bytea
urowid	oid
rowid	oid
float	double precision
dec	decimal
decimal	decimal
double precision	double precision
int	numeric

Тип в Oracle	Тип в Postgres
integer	numeric
real	real
smallint	smallint
binary_float	double 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

Типы данных rowtype в хранимом коде становятся просто табличным типом. Например:

v sys.all_users%rowtype;

будет преобразован в

v sys.all_users;

Типы

Аналога объектных типов в Postgres нет. Поэтому методы и тело объектных типов не автомигрируются, а их использование в мигрируемых программах требует существенной модификации.

Объектные типы следует вручную преобразовывать к созданию отдельной схемы, содержащей функции - методы объектного типа, как это делается для пакетов.

Стандартные функции

Некоторые стандартные функции Oracle имеют аналоги в Postgres, однако имеют другое наименование и/или способ вызова. Ниже в таблице перечислены функции, которые претерпевают изменения в результате автоматической миграции:

Функция в Oracle	Функция в Postgres	Примечания
nvl	coalesce	
sequence.nextval	nextval(‘sequence’)	
sequence.currval	currval(‘sequence’)	
decode	case	Требуется модификация вручную
ltrim	trim	
rtrim	trim	

Конструкции языка SQL

Команда в Oracle	Команда в Postgres	Примечания
bulk collect	Аналога нет. Автомиграция не выполняется.	Требуется переписывание кода вручную

Команда в Oracle	Команда в Postgres	Примечания
update … row=…	Аналога нет. Автомиграция не выполняется.	Вместо row требуется указать полный перечень столбцов.

Конструкции языка PL/SQL

Конструкция в Oracle	Аналог в Postgres	Примечания
… is/as Var vartype; begin …	…as $body$ declare Var vartype; begin …	Автомиграция выполняется. Но при ручной правке Posgres-кода следует помнить о том, что секция объявления переменных начинается с declare.
connect by	-	Аналога нет. Автомиграция не выполняется. Требуется переписывание кода вручную
into	strict into	Автомиграция выполняется. Но при ручной правке Posgres-кода этот момент следует учитывать.
for cursor_name in…	cursor_name record; … begin … for cursor_name in…	Автомиграция выполняется. Но при ручной правке Posgres-кода следует помнить о том, что курсор должен быть объявлен.

Стандартные пакеты Oracle

Рекомендуется установка и использование в Postgres пакета ORAFCE (Oracle's compatibility functions and packages), в состав которого входит программный код на языке PlPgSQL с попыткой реализации функциональности следующих пакетов Oracle:

- DBMS_OUTPUT,
- DBMS_PIPE,
- DBMS_ALERT,
- DBMS_UTILITY,
- DBMS_ASSERT,
- DBMS_RANDOM,
- UTL_FILE.

Если в исходном хранимом программном коде используются внешние пакеты DBMS/UTL, не реализованные в ORAFCE, или по каким-то причинам функциональность ORAFCE не удовлетворяет, рекомендуется реализовать вручную алгоритм работы всех используемых функций в схеме, соответствующей наименованию DBMS/UTL-пакета, скрипты для создания которых подготавливаются в процессе выполнения автомиграции.

По умолчанию, в результате выполнения автомиграции будет сформирован скрипт для создания в схеме DBMS_OUTPUT функции put_line с использованием "RAISE NOTICE '%', pstr", где pstr - параметр функции. Потребуется также вручную реализовать алгоритм работы с заданиями в функциях схем DBMS_JOB, DBMS_SCHEDULER.

Либо придумывать какую-то другую оригинальную реализацию работы планировщика заданий и взаимодействия с ним.

Триггеры

ON-logon

Аналогов триггеру on-logon в Postgres нет. Эти триггеры не добавляются в проект и не автомигрируются. Если такие триггеры используются в проекте на Oracle, решение для Postgres необходимо подбирать в зависимости от назначения такого триггера.

On-logon для переключения на другую схему

Вместо триггера следует использовать set search_path=schema.

On-logon для аудита подключений

Аудит подключений ведётся средствами Postgres. При миграции следует модифицировать программы и представления, обращающиеся к кастомизированной таблице аудита подключений таким образом, чтобы они обращались к стандартному представлению Postgres.

Составной триггер

Составной триггер не имеет аналогов в Postgres. Чтобы выполнить автомиграцию, его следует сначала разделить на несколько отдельных триггеров.

Локальные функции внутри тела триггера

Рекомендуется с помощью автомиграции триггеров выявить триггеры, имеющие данное ограничение, затем модифицировать в Oracle эти триггеры таким образом, чтобы внутри них не было определения локальных функций/процедур. Каждая локальная функция/процедура должна стать самостоятельной (либо автономной-standalone, либо внутри пакета-package). Затем повторно выполнить автомиграцию триггеров, а новые функции/процедуры/пакеты добавить в проект миграции.

Если предложенный выше вариант решения проблемы недопустим, то следует выполнить одну из рекомендаций ниже:

- вручную добавить объект Postgres типа FUNCTION, написав для него требуемые команды создания и предоставления прав, если локальная функция преобразуется в функцию;
- вручную добавить объекты Postgres типа SCHEMA и FUNCTION, написав для них требуемые команды создания и предоставления прав, если локальная функция преобразуется в функцию;

Модифицировать тело триггера, взяв за основу DDL для Oracle так, чтобы из тела триггера вызывались новые функции/пакет, учитывая при модификации другие ограничения миграции триггеров.

Процедуры

Функции внутри других функций

Рекомендуется с помощью автомиграции процедур выявить процедуры, имеющие данное ограничение, затем модифицировать хранимый программный код таким образом, чтобы внутри функций/процедур не было определения вложенных в них других функций/процедур. Каждая функция/процедура должна стать самостоятельной (либо автономной-standalone, либо внутри пакета-package). В ином случае, при автомиграции скрипты для создания программного кода на PlPgSQL будут содержать в себе следующий вариант реализации: Вместо кода Oracle, например, такого

```
... CREATE OR REPLACE FUNCTION test_outer() RETURNS varchar2
 AS
DECLARE
  s varchar2(4000);
  FUNCTION test_inner() RETURNS varchar2
  BEGIN
    RETURN 'inner';
```

```
  END;
BEGIN
  s := test_inner();
  return s;
END;
```

получится код Postgres, например, такой

```
 ... CREATE OR REPLACE FUNCTION test_outer() RETURNS text
AS $body$
DECLARE
  s text;
BEGIN
   CREATE OR REPLACE FUNCTION test_inner() RETURNS text
    AS $test_inner$
    BEGIN
      RETURN 'inner';
   END; $test_inner$ language plpgsql;
   SELECT test_inner() INTO s;
   DROP FUNCTION test_inner();
   return s;
END; $body$ language plsql;
```

Таким образом, функция не объявляется в блоке declare, а создаётся-выполняется-удаляется в блоке begin/end;

Хотя такая конструкция синтаксически верная, тем не менее, она вызывает блокировку словаря Postgres, что при наличии длинных транзакций или продолжительного выполнения функции не позволяет выполняться этой функции одновременно из двух различных сессий.

При отсутствии возможности создания в Oracle stand-along функций, которые были локальными, полученный в результате код функции следует вручную модифицировать таким образом, чтобы имя внутренней функции генерировалось, и в результате генерации получалось уникальное значение.

Функции

Функции внутри других функций

Рекомендуется с помощью автомиграции функций выявить функции, имеющие данное ограничение, затем модифицировать хранимый программный код таким образом, чтобы внутри функций/процедур не было определения вложенных в них других функций/процедур. Каждая функция/процедура должна стать самостоятельной (либо автономной-standalone, либо внутри пакета-package). В ином случае, при автомиграции скрипты для создания программного кода на PlPgSQL будут содержать в себе следующий вариант реализации: Вместо кода Oracle, например, такого

```
... CREATE OR REPLACE FUNCTION test_outer() RETURNS varchar2
 AS
DECLARE
  s varchar2(4000);
  FUNCTION test_inner() RETURNS varchar2
  BEGIN
```

```
    RETURN 'inner';
  END;
BEGIN
  s := test_inner();
  return s;
END;
```

получится код Postgres, например, такой

```
 ... CREATE OR REPLACE FUNCTION test_outer() RETURNS text
AS $body$
DECLARE
  s text;
BEGIN
   CREATE OR REPLACE FUNCTION test_inner() RETURNS text
    AS $test_inner$
    BEGIN
      RETURN 'inner';
   END; $test_inner$ language plpgsql;
   SELECT test_inner() INTO s;
   DROP FUNCTION test_inner();
   return s;
END; $body$ language plsql;
```

Таким образом, функция не объявляется в блоке declare, а создаётся-выполняется-удаляется в блоке begin/end;

Хотя такая конструкция синтаксически верная, тем не менее, она вызывает блокировку словаря Postgres, что при наличии длинных транзакций или продолжительного выполнения функции не позволяет выполняться этой функции одновременно из двух различных сессий.

При отсутствии возможности создания в Oracle stand-along функций, которые были локальными, полученный в результате код функции следует вручную модифицировать таким образом, чтобы имя внутренней функции генерировалось, и в результате генерации получалось уникальное значение.

Функции с out-параметрами

В результате автомиграции функции с out-параметрами вместо кода на Oracle

return function (p1,p2,p3);

мигрируются следующим образом:

return select * from function(p1) into strict p2, p3;

В Postgres аналога таким функциям нет. Получающийся код не будет работать так, как эта функция использовалась в Oracle. Возвращаемое значение функции игнорируется. Таким образом, все такие функции, а также их вызовы необходимо вручную приводить к виду, при котором будет возвращаемое значение будет out-параметром.

Пакеты

Глобальные переменные в теле пакета

Глобальные переменные в заголовке пакета

Функции внутри других функций

Рекомендуется с помощью автомиграции пакетов выявить пакеты, имеющие данное ограничение, затем модифицировать хранимый программный код таким образом, чтобы внутри функций/процедур не было определения вложенных в них других функций/процедур. Каждая функция/процедура должна стать самостоятельной (либо автономной-standalone, либо внутри пакета-package). В ином случае, при автомиграции скрипты для создания программного кода на PlPgSQL будут содержать в себе следующий вариант реализации: Вместо кода Oracle, например, такого

```
... CREATE OR REPLACE FUNCTION test_outer() RETURNS varchar2
 AS
DECLARE
  s varchar2(4000);
  FUNCTION test_inner() RETURNS varchar2
  BEGIN
    RETURN 'inner';
  END;
BEGIN
  s := test_inner();
  return s;
END;
```

получится код Postgres, например, такой

```
 ... CREATE OR REPLACE FUNCTION test_outer() RETURNS text
AS $body$
DECLARE
  s text;
BEGIN
   CREATE OR REPLACE FUNCTION test_inner() RETURNS text
    AS $test_inner$
    BEGIN
      RETURN 'inner';
   END; $test_inner$ language plpgsql;
   SELECT test_inner() INTO s;
   DROP FUNCTION test_inner();
   return s;
END; $body$ language plsql;
```

Таким образом, функция не объявляется в блоке declare, а создаётся-выполняется-удаляется в блоке begin/end;

Хотя такая конструкция синтаксически верная, тем не менее, она вызывает блокировку словаря Postgres, что при наличии длинных транзакций или продолжительного выполнения функции не позволяет выполняться этой функции одновременно из двух различных сессий.

При отсутствии возможности создания в Oracle stand-along функций, которые были локальными, полученный в результате код функции следует вручную модифицировать таким образом, чтобы имя внутренней функции генерировалось, и в результате генерации получалось уникальное значение.

Автономные транзакции

Автономные транзакции рекомендуется реализовывать через создание DB-link на ту же схему. Реализация этой функциональности требует программирования вручную.

Синонимы

Синоним ссылается на процедуру/функцию

Если синоним ссылается на процедуру или функцию, то мигрировать следует путём создания одноимённой процедуры или функции, в теле которых выполняется вызов оригинальной процедуры или функции.

При автомиграции создаётся прототип скрипта создания функции, в котором вызов оригинальной функции следует указывать вручную.

Синоним ссылается на пакет

Если синоним ссылается на пакет, то мигрировать следует путём создания одноимённой синониму схемы, содержащей все функции и процедуры спецификации оригинального пакета, в каждой из которых выполняется вызов соответствующей оригинальной функции/пакета.

При автомиграции создаётся прототип скрипта создания схемы, в котором создание функций, входящих в спецификацию пакета на Oracle, следует указывать вручную. При этом в каждой из функций новой схемы должен выполняться вызов соответствующей оригинальной функции/пакета.

Синоним ссылается на таблицу, представление или материализованное представление

Если тип объекта, на который ссылается синоним - таблица, представление или материализованное представление, то мигрировать его следует путём создания одноимённого представления Postgres. Если синоним ссылается на процедуру или функцию, то мигрировать следует путём создания одноимённой процедуры или функции, в теле которых выполняется вызов оригинальной процедуры или функции.

Неизвестный тип объекта, на который ссылается синоним

Такая ситуация возможна, когда синоним создаётся на объект из удалённой базы данных для того, чтобы скрыть ДБ-линк. В этом случае при автомиграции тип объекта Postgres не определяется. Объект Postgres необходимо создавать вручную с учётом вышеописанных особенностей миграции синонимов.