РАЗРАБОТКА АВТОМАТИЗИРОВАННОГО ПРОЦЕССА ПО КОРРЕЛЯЦИИ ИНФОРМАЦИИ МЕЖДУ ТАБЛИЦЕЙ БАЗЫ ДАННЫХ POSTGRESQL И ACTIVE DIRECTORY

Э.В. Бирих, *Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, be1982@mail.ru;*
К.П. Волокушина, *Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А. Бонч-Бруевича, ksylik@gmail.com.*

УДК 004.658:65.011.56

Аннотация. В статье представлен результат разработки автоматизированного решения для регулирования информации между таблицей базы данных и *Active Directory*. Решение позволяет устранить проблемы при переносе информации, сокращает время исполнения задачи и повышает производительность за счет экономии ресурсов, затрачиваемых человеком.

Ключевые слова: *Active Directory*; *Camunda*; *Python*; автоматизация; базы данных; код.

DEVELOPMENT OF AUTOMATED PROCESS FOR CORRELATION OF INFORMATION BETWEEN POSTGRESQL DATABASE TABLE AND ACTIVE DIRECTORY

Ernest Birikh, *St. Petersburg state university of telecommunications n/a prof. M.A. Bonch-Bruevich;*
Ksenia Volokushina, *St. Petersburg state university of telecommunications n/a prof. M.A. Bonch-Bruevich.*

Annotation. The article presents the result of developing an automated solution for regulating information between a database table and *Active Directory*. The solution allows you to eliminate problems during the transfer of information, reduces the time of task execution and increases productivity by saving human resources.

Keywords: *Active Directory*; *Camunda*; *Python*; automation; databases; code.

Введение

Автоматизация рутинных процессов при использовании сетевых технологий является одной из самых приоритетных задач сегодняшнего времени. В особенности, когда задача состоит в корреляции большого количества данных между разными системами, поскольку перенос данных вручную может сопровождаться как ошибками, так и затратой большого количества времени. Целью данной статьи является реализация интеграции информации из сторонней таблицы в *Active Directory* с использованием языка программирования *Python*, а также сервиса *Camunda* для создания саморегулирующихся процессов.

Прежде чем перейти непосредственно к описанию процесса разработки вышеописанной задачи, следует разобрать некоторые основные понятия, используемые в данной статье:

- *Active Directory* – службы каталогов корпорации *Microsoft* для операционных систем семейства *Windows Server.* С помощью данной службы можно объединить различные объекты сети, образовав систему, хранящую и обрабатывающую только необходимые данные [1]. Каталог представляет собой иерархическую структуру и может хранить сведения об учетных записях пользователей, именах, паролях и т.д. *Active Directory*

позволяет другим авторизованным пользователям в той же сети получать доступ к этой информации, а также использует структурированное хранилище данных в качестве основы для логической организации сведений [2].

- База данных *PostgreSQL* – это реляционная база данных с открытым кодом, позволяющая упорядочить данные и составить представление о связях между ними. Преимуществами именно данного программного обеспечения являются совместимость со всеми основными операционными системами, такими как *Linux* и *Windows*, а также поддержка широкого спектра типов данных [3].

Язык программирования *Python* и используемые библиотеки

Высокоуровневый язык программирования *Python* является одним из самых распространенных в своей категории благодаря не только простоте использования и скорости разработки проектов, но и широкому спектру доступных библиотек [4]. Среди последних, применяемых для работы с базами данных и *Active Directory*, наиболее активно используются *psycopg*2 и *pyad*:

- *psycopg* – адаптер базы данных *PostgreSQL* для языка программирования *Python*. В основном реализован на языке программирования С как обертка *libpq*, что приводит к большей эффективности и безопасности. Многие типы *Python* поддерживаются стандартными и адаптированы к соответствующим типам данных *PostgreSQL*; адаптация может быть расширена благодаря гибкой системе оптимизирования объектов [5].
- *pyad* – библиотека Python, предназначенная для обеспечения простого интерфейса с *Active Directory* через *ADSI* (*Active Directory Service Interfaces*) на платформе *Windows* [6].

Выбор был сделан в пользу этих библиотек, поскольку первая позволяет работать с базами данных по типу *PostgreSQL*, вторая имеет самый большой функционал для работы со службой каталогов. При помощи представленных модулей был реализован основной рабочий принцип конечной программы, а также вспомогательный скрипт, используемый на начальном этапе разработки с целью предварительной корреляции данных. Модуль позволяет собрать данные из таблицы и *Active Directory*, а затем проверяет наличие или отсутствие тех или иных составляющих. По полученным данным проводится регулирование данных вручную с целью контроля информации во избежание дальнейших возможных ошибок.

Триггерные процедуры *PostgreSQL*

PostgreSQL позволяет создавать триггерные процедуры, которые будут вызываться при изменениях данных или событиях в базе данных. Триггерная процедура создается командой *create function*, при этом у функции не должно быть аргументов, а типом возвращаемого значения должен быть *trigger* (для триггеров, срабатывающих при изменениях данных). Для триггеров автоматически определяются специальные локальные переменные с именами вида *tg*_имя, описывающие условие, повлекшее вызов триггера. Когда функция на *PostgreSQL* срабатывает как триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных:

- *new* – тип данных *record*. переменная содержит новую строку базы данных для команд *insert/update* в триггерах уровня строки; в триггерах уровня оператора и для команды *delete* этой переменной значение не присваивается;

- *old* – тип данных *record.* переменная содержит старую строку базы данных для команд *update/delete* в триггерах уровня строки; в триггерах уровня оператора и для команды *insert* этой переменной значение не присваивается;
- *tg_name* – тип данных *name.* переменная содержит имя сработавшего триггера;
- *tg_relname* – тип данных *name.* имя таблицы, для которой сработал триггер; эта переменная устарела и может стать недоступной в будущих релизах, вместо нее нужно использовать *tg_table_name;*
- *tg_table_name* – тип данных name. имя таблицы, для которой сработал триггер;
- *tg_nargs* – тип данных *integer.* число аргументов в команде *create trigger,* которые передаются в триггерную процедуру [7].

В программе, тематике которой посвящена данная статья, обрабатываются триггеры типов *update, insert* и *delete* при помощи созданных скриптом, написанном на ранее указанном языке программирования *Python*, функций. При срабатывании каждого из событий в базе данных та или иная функция начинает свою работу (рис. 1). Так на рис. 1 показан пример триггерной функции и обработка триггера типа *insert,* написанные синтаксисом *PostgreSQL* и помещенные в строковое значение *Python.* Первые три блока скрипта посвящены параметрам для обозначения и обработки функции. Первый создает или заменяет уже существующую функцию, срабатывающую после возникновения соответствующего события. Второй блок указывает значения, которые возвращает функция. В данном случае это имя произведенного действия и строковое значение добавленной в базу данных информации. Третий – в свою очередь указывает аргументы для внедрения функции в базу данных, а именно:

- язык, используемый для написания функции, конкретно для этого примера – *plpgsql (Procedural Language/PostgreSQL*);
- стоимость, от которой зависит, насколько часто будет вызываться функция (установлено значение по умолчанию, равное 100);
- конструкция *ALTER FUNCTION,* предназначенная для переопределения функции, в данном примере она указывает владельца, заданного в скрипте при помощи переменной *owner.*

Последний блок создает триггер, реагирующий на событие *insert* и выполняющий действия в зависимости от указанных функций. Так, в конкретном примере обработка события производится для каждой строки, где была добавлена новая информация, действия триггера производятся после возникновения события. Данный пункт важен, поскольку в случае удаления данных вызвать триггерную функцию необходимо до события, в ином случае, никакой информации получить невозможно. Еще двумя обязательными параметрами являются имя таблицы (*table_name*) базы данных, к которой привязан конкретный триггер, и имя исполняемой функции. Примечательным является то, что триггер обязательно должен быть привязан к определенной таблице, в то время как триггерные функции создаются для всей базы данных в целом. В зависимости от задач, одна и та же функция может использоваться в разных таблицах при различных событиях каждой из них, что позволяет разграничить обработку информации.

```
owner = 'OWNER'
table_name = 'SOME_TABLE'

for_insert = """
CREATE OR REPLACE FUNCTION new_record_notify()
RETURNS trigger AS
$BODY$

BEGIN
PERFORM pg_notify('new_record', row_to_json(NEW)::text);
RETURN NEW;
END;

$BODY$
LANGUAGE plpgsql VOLATILE
COST 100;
ALTER FUNCTION new_record_notify()
OWNER TO {owner};

CREATE TRIGGER add_new_data_event_trigger
AFTER INSERT
ON {table_name}
FOR EACH ROW
EXECUTE PROCEDURE new_record_notify();

""".format(owner=owner, table_name=table_name)
```

Рисунок 1

Проектирование схемы автоматизированного процесса при помощи сервиса *Camunda.*

Camunda – это платформа для автоматизации и моделирования рабочих процессов. Она представляет собой набор библиотек, которые позволяют выполнять описанные процессы и обрабатывать их в зависимости от указанных задач, адаптируя их под определенный тип исполнения (*user tasks, service tasks, events, gateways* и т.п.) [8]. Для интеграции *Camunda* в проект необходимо:

- создать модель с процессами в зависимости от типа осуществления каждой из подзадач;
- обозначить зависимости, последовательности, начальное и конечное события общего процесса (могут быть множественными);
- указать имена функций для автоматизированных процессов;
- использовать специальный *API* (*Application Programming Interface*) для используемого языка программирования.

В данном случае использовалась библиотека *pycamunda* – клиент *API Python REST* для механизма автоматизации рабочего процесса и принятия решений *Camunda* [9]. С помощью данного модуля были реализованы как функции для *service tasks* модели общего процесса задачи, так и *workers* – саморегулирующиеся объекты, реагирующие на привязанные к ним функции и события и посылающие сигналы на сервис *Camunda*, где идет обработка поступившей информации в соответствии с составленной схемой событий (рис. 2). На рис. 2 показана схема процессов программы от изменений в таблице базы данных до поступления обработанной информации в *Active Directory.* Началом схемы является срабатывание триггера в базе данных *PostgreSQL*, после чего идет его проверка на соответствие тому или иному событию по возвращаемым триггерной функцией значениям. Далее идут задачи, обрабатываемые человеком. Для каждой из таких задач была создана персональная форма, позволяющая скорректировать полученную из таблицы базы данных информацию и принять решение о добавлении данных или об отмене каких-либо действий. В случае подтверждения срабатывают автоматизированные процессы, нацеленные на работу с сервисом

Active Directory. Каждая из таких сервисных подзадач при помощи ранее описанного объекта *worker* привязана к определенному модулю/функции. Последние принимают значения, передаваемые через процессы *Camunda*, в данном случае – это данные о добавленном, удаленном или измененном пользователе из таблицы базы данных *PostgreSQL*.

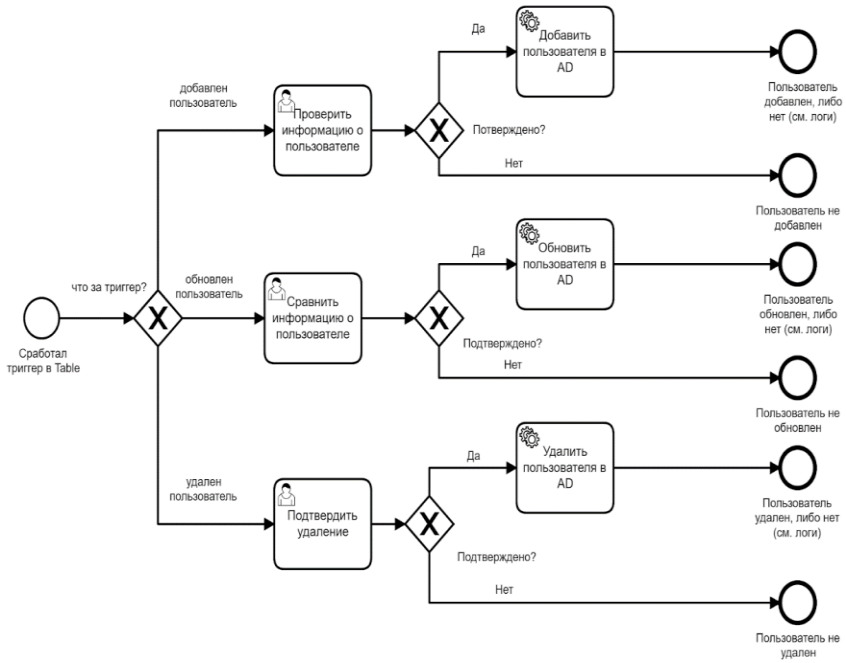

Рисунок 2

Работа со службой каталогов *Active Directory*

Модуль для работы с *Active Directory*, как было описано ранее, использует библиотеку *pyad*, являющуюся *API* к *Python* для грамотной работы со службой каталогов *Windows.* Главной особенностью является то, что в зависимости от текущего пользователя системы, библиотека автоматически определяет права доступа и при их наличии, подключается без дополнительной авторизации (рис. 3). В случае использования с аккаунта пользователя без необходимых прав, можно при помощи закомментированного кода на рис. 3 ввести данные учетной записи с необходимыми правами, которые скрипт будет использовать для доступа.

```
full_name = last_name + ' ' + first_name
base_dn = "CN=SOME_CN,DC=SOME_DC,DC=SOME_DC"

# if connection denied, set user with admin properties
# pyad.set_defaults(ldap_server="server/domain", username="user", password="pass")

try:
    ou = adcontainer.ADContainer.from_dn(distinguished_name=base_dn)
except pywintypes.com_error as err:

    msg = 'Ошибка. Нет доступа к Active Directory.\n' \
          'Пользователь ' + full_name + ' не будет добавлен в Active Directory.'
    make_log_record(msg=msg, err=err, level='error', full_name=full_name)
    return None
```

Рисунок 3

После подключения идет процесс добавления информации в соответствующий каталог *Active Directory*, к которому был получен доступ ранее (рис. 4). Как видно на рис. 4, сначала создаются переменные, принимающие значения из таблицы базы данных, переданные при помощи процессов *Camunda*, после чего вся информация становится атрибутами, используемыми для создания нового пользователя в *Active Directory* в конкретном случае.

```
last_name = str(last_name.value)
first_name = str(first_name.value)
login = str(login.value)
full_name = last_name + ' ' + first_name
email = login + "@domain"
full_name = last_name + ' ' + first_name

try:
    ou.create_user(full_name,
                   optional_attributes={
                       "name": "{name}".format(name=full_name),
                       "givenName": "{name}".format(name=first_name),
                       "distinguishedName": "CN={name},".format(name=full_name) + base_dn,
                       "sn": "{name}".format(name=last_name),
                       "displayName": "{name}".format(name=full_name),
                       "userPrincipalName": "{email}".format(email=email),
                       "sAMAccountName": "{login}".format(login=login)
                   })
    msg = 'Пользователь ' + full_name + ' добавлен в Active Directory.'
    make_log_record(msg=msg, level='info', full_name=full_name)
    return {}
```

Рисунок 4

Общий функционал программы

Итоговая программа представляет из себя сервис, состоящий из двух потоков: первый отслеживает и обрабатывает изменения в указанной таблице, и, если действие происходит – отправляет сигнал, на который реагирует второй основной модуль проекта. Последний создает рабочий процесс для *Camunda* в соответствии с составленной заранее схемой-обработчиком событий. В зависимости от полученного сигнала, появляется то или иное информационное сообщение, содержащее данные об изменениях таблицы из базы данных, после чего ожидается подтверждение человеком действий или же их отмену. В случае положительной проверки, данные будут либо добавлены в *Active Directory*, либо изменены в службе каталогов, либо же удалены из нее, в зависимости от контекста изначального сигнала. Конечный этап работы программы – логгирование произведенных действий.

Заключение

В соответствии с описанными ранее инструментами и модулями, итоговым результатом проделанной работы стал сервис, служащий в качестве вспомогательной утилиты для администрирования взаимодействия таблиц и службы каталогов *Active Directory.*

Данный проект позволяет сократить количество создаваемых ошибок по сравнению с переносом данных вручную, обеспечивает гибкую систему отслеживания действий, а также экономит затрачиваемые человеком ресурсы на рутинные задачи, что приводит к большей производительности труда.

Помимо вышеперечисленного, развитие функционала программы в будущем может быть направлено не только на обеспечения контроля процессов, но и на защиту информации, передаваемой из таблицы баз данных [10].

Литература

1. Структура хранилища Active Directory.
URL: https://1cloud.ru/help/windows/struktura-hranilischa-active-directory (дата последнего обращения: 11.11.2022).
2. Обзор доменных служб Active Directory. URL: https://learn.microsoft.com/ru-ru/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview (дата последнего обращения: 11.11.2022).
3. Что такое PostgreSQL. URL: https://azure.microsoft.com/ru-ru/resources/cloud-computing-dictionary/what-is-postgresql/ (дата последнего обращения: 11.11.2022)
4. Лутц М. Изучаем Python. – Изд-во Вильямс, 2019. – 832 с.
5. Psycopg2. Описание проекта. URL: https://pypi.org/project/psycopg2 (дата последнего обращения: 11.11.2022).
6. Pyad. Описание проекта. URL: https://pypi.org/project/pyad (дата последнего обращения: 11.11.2022).
7. Триггерные процедуры. Триггеры при изменении данных.
URL: https://postgrespro.ru/docs/postgresql/9.6/plpgsql-trigger (дата последнего обращения: 11.11.2022).
8. Camunda. The Universal Process Orchestrator. URL: https://camunda.com/platform (дата последнего обращения: 11.11.2022).
9. Pycamunda. URL: https://pypi.org/project/pycamunda (дата последнего обращения: 11.11.2022).
10. Бирих Э.В., Виткова Л.А., Гореленко В.В., Казаков Д.Б. Защита информации в базах данных [Электронный ресурс] // В сборнике: Актуальные проблемы инфотелекоммуникаций в науке и образовании. Материалы конференции АПИНО 2017, 2017. – С. 89-92.