import java.io.FileInputStream;  
import java.io.PrintWriter;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.PreparedStatement;  
import java.sql.Statement;  
import java.util.Scanner;  
  
public class HistoryImporter {  
  
 private Connection connection;  
  
 private void initConnection() throws Exception {  
 String[] dbProps = getDBprops();  
 try {  
 connection = DriverManager.*getConnection*(dbProps[0], dbProps[1], dbProps[2]);  
 } catch (Exception e) {  
 System.*err*.println(e.toString());  
 }  
 }  
  
 private String[] getDBprops() throws Exception {  
 FileInputStream inputStream = new FileInputStream("./dbProps.txt");  
 Scanner sc = new Scanner(inputStream);  
 String url;  
 String user;  
 String password;  
 try {  
 url = sc.nextLine();  
 user = sc.nextLine();  
 password = sc.nextLine();  
 } finally {  
 inputStream.close();  
 sc.close();  
 }  
  
 return new String[]{url, user, password};  
 }  
  
 public void importFile(String fileDir, String fileName) throws Exception {  
 try {  
 initConnection();  
  
 long counter = 0L;  
  
 createTable(fileName);  
  
 FileInputStream inputStream = null;  
 PrintWriter writer = null;  
 Scanner sc = null;  
 try {  
 inputStream = new FileInputStream(fileDir + fileName);  
 writer = new PrintWriter("Report\_" + fileName + ".txt", "UTF-8");  
 sc = new Scanner(inputStream, "UTF-8");  
 while (sc.hasNextLine()) {  
 String line = sc.nextLine();  
 String[] historyItemDataArray = line.split("\\$#c\\$#");  
 historyItemDataArray[0] = historyItemDataArray[0].replace("$#r$#", "\r");  
 historyItemDataArray[0] = historyItemDataArray[0].replace("$#n$#", "\n");  
  
 if (historyItemDataArray.length < 6) {  
 String message = "String below (number is " + (++counter) + ") consists of " + historyItemDataArray.length +" fields. Process it by hand." + "\n"  
 + "$START$" + line + "$END$";  
  
 writer.println(message);  
 System.*out*.println(message);  
 continue;  
 }  
  
 for (int i = 0; i < historyItemDataArray.length; i++) {  
 historyItemDataArray[i] = historyItemDataArray[i].replace("'", "''");  
 }  
 loadHistoryItemDataToDB(historyItemDataArray, fileName);  
  
 if (counter % 10000 == 0) {  
 System.*out*.println("Processing data... " + counter);  
 }  
 counter++;  
 }  
 System.*out*.println("\nHistory import finished!");  
 if (sc.ioException() != null) {  
 throw sc.ioException();  
 }  
 } finally {  
 if (inputStream != null) {  
 inputStream.close();  
 }  
 if (sc != null) {  
 sc.close();  
 }  
 if (writer != null) {  
 writer.close();  
 }  
 }  
 } finally {  
 if (connection != null) {  
 connection.close();  
 }  
 }  
 }  
  
 private void createTable(String fileName) throws Exception {  
 Statement statement = connection.createStatement();  
 String query = "" +  
 "create table if not exists \"import\_" + fileName + "\"(\n" +  
 " \"fieldname1\" text,\n" +  
 " \"fieldname2\" text,\n" +  
 " \"fieldname3\" timestamp,\n" +  
 " \"fieldname4\" text,\n" +  
 " \"fieldname5\" text,\n" +  
 " \"fieldname6\" text,\n" +  
 " \"fieldname7\" bigint,\n" +  
 " \"fieldname8\" text,\n" +  
 " \"fieldname9\" text,\n" +  
 " \"fieldname10\" bigint,\n" +  
 " \"fieldname11\" text\n" +  
 ");";  
 System.*out*.println("Creating table \"fedor\_import\_" + fileName + "\"...");  
 System.*out*.println("Query:\n" + query);  
 statement.executeUpdate(query);  
 System.*out*.println("Table has been created");  
 }  
  
 private void loadHistoryItemDataToDB(String[] historyItemDataArray, String fileName) throws Exception {  
 PreparedStatement pst = connection.prepareStatement("" +  
 "INSERT INTO \"\_import\_" + fileName + "\"(\"fieldname1\",\"fieldname2\",\"fieldname3\",\"fieldname4\",\"fieldname5\",\"fieldname6\")\n" +  
 "VALUES ('" + historyItemDataArray[0] + "', '" + historyItemDataArray[1] + "', '" + historyItemDataArray[2] + "', '" + historyItemDataArray[3] + "', '" + historyItemDataArray[4] + "', '" + historyItemDataArray[5] + "');");  
 pst.executeUpdate();  
 }  
}

В PL/pgSQL можно создавать триггерные процедуры, которые будут вызываться при изменениях данных или событиях в базе данных. Триггерная процедура создаётся командой CREATE FUNCTION, при этом у функции не должно быть аргументов, а типом возвращаемого значения должен быть trigger (для триггеров, срабатывающих при изменениях данных) или event\_trigger (для триггеров, срабатывающих при событиях в базе). Для триггеров автоматически определяются специальные локальные переменные с именами вида TG\_*имя*, описывающие условие, повлёкшее вызов триггера.

Синтаксис:

CREATE [ CONSTRAINT ] TRIGGER ***name*** { BEFORE | AFTER | INSTEAD OF } { ***event*** [ OR ... ] }

ON ***table***

[ FROM ***referenced\_table\_name*** ]

[ NOT DEFERRABLE | [ DEFERRABLE ] { INITIALLY IMMEDIATE | INITIALLY DEFERRED } ]

[ FOR [ EACH ] { ROW | STATEMENT } ]

[ WHEN ( ***condition*** ) ]

EXECUTE PROCEDURE ***function\_name*** ( ***arguments*** )

where ***event*** can be one of:

INSERT

UPDATE [ OF ***column\_name*** [, ... ] ]

DELETE

TRUNCATE

Пример триггера по событию INSERT или UPDATE в таблицу users. Он вызывает функцию, которая выполняется перед выполнение вызываемой операции.

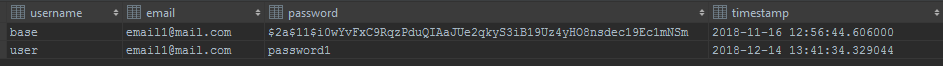
**DROP FUNCTION IF EXISTS** *emp\_stamp*();  
**CREATE FUNCTION** *emp\_stamp*() **RETURNS trigger AS**$$  
**BEGIN**  
 **IF NEW**.username <> 'user' **THEN  
 RAISE EXCEPTION** 'username should be "user"';  
 **END IF**;  
 **IF NEW**.email <> 'email1@mail.com' **THEN  
 RAISE EXCEPTION** 'email % is incorrect', **NEW**.email;  
 **END IF**;  
  
 **IF NEW**.password = 'password' **THEN  
 RAISE EXCEPTION** 'password is too obvious';  
 **END IF**;  
  
 **NEW**.timestamp := *current\_timestamp*;  
 **RETURN NEW**;  
**END**;  
$$ **LANGUAGE** plpgsql;  
  
**DROP TRIGGER IF EXISTS** my\_trigger **ON** users;  
**CREATE TRIGGER** my\_trigger **BEFORE INSERT OR UPDATE ON** users  
 **FOR EACH ROW EXECUTE PROCEDURE** *emp\_stamp*();

Данный триггер будет проверять значение, которое идет в таблицу users, и выдавать исключение (+ прерывать) операцию добавления данных, если username не равно ‘user’ или email не равен ‘email1@mail.com’ или password равен ‘password’.

Также триггер реализует автоматическое заполнение даты/времени добавления записи.

Выполним запросы для проверки:

**insert into** users(username, email, **password**) **values** ('user', 'email1@mail.com', 'password1');



Добавило юзера, прописало ему timestamp.

**insert into** users(username, email, **password**) **values** ('user1', 'email1@mail.com', 'password');



Получаем ошибку.

Когда функция на PL/pgSQL срабатывает как триггер, в блоке верхнего уровня автоматически создаются несколько специальных переменных, например переменная NEW, как в предыдущем примере.

**create table** article\_logs (  
 id **bigserial primary key**,  
 article\_id **bigint**,  
 operation\_type **text**,  
 operation\_timestamp **timestamp**);  
  
**CREATE OR REPLACE FUNCTION** *article\_logging*() **RETURNS trigger AS**$$  
**BEGIN  
 if** TG\_OP = 'INSERT' **or** TG\_OP = 'UPDATE' **then  
 INSERT INTO** article\_logs(article\_id, operation\_type, operation\_timestamp)  
 **SELECT NEW**.id, TG\_OP, *now*();  
 **elseif** TG\_OP = 'DELETE' **then  
 INSERT INTO** article\_logs(article\_id, operation\_type, operation\_timestamp)  
 **SELECT OLD**.id, TG\_OP, *now*();  
 **end if**;  
 **RETURN NULL**;  
**END**;  
$$ **LANGUAGE** plpgsql;  
  
**DROP TRIGGER IF EXISTS** article\_logger **ON** articles;  
**CREATE TRIGGER** article\_logger **AFTER INSERT OR UPDATE OR DELETE ON** articles  
 **FOR EACH ROW EXECUTE PROCEDURE** *article\_logging*();