TOPTAL TEST PROJECT

Solution Description

Pablo Mombrú

Content

Introduction

This document has been created to describe the solutions to the requirements included in Toptal's test project.

The project description and requirements can be found in Toptal Test Project Requirements.

The implementation of the test project will be reviewed in a meeting on September 23rd 2019 with Ivan Ilijasic from Toptal.

The intention of this document is to support the implementation tasks.

TaxesCollection Database

The RDBMS used for implementing this project is PostgreSQL v.11, running under Windows 10 OS.

A PostgreSQL server has been installed called ***PostgreSQL 11***.

On PostgreSQL 11 server, a database called ***TaxesCollection*** was created in order to fulfill the Toptal requirements.

Based on the description of the project requirements, following entities were identified:

- **Taxpayer**: Who pays taxes. A taxpayer can be an Individuals or a Company.
- **Individual**: Person who pays taxes. Individuals are also owners of one or more companies.
- **Company**: Enterprise that pays taxes. Companies are owned by individuals.
- **Own_Rel**: Ownership relationships between individuals and companies. The relationship Individual/Company is unique.
- **Agency**: Office where the taxpayers pay taxes.
- **Tax Payment**: Payments done by a taxpayer (individual or company) to an agency. Payments are unique by the combination of Agency, Taxpayer, Payment Date and Tax Type
- **Tax Type**: Type of tax. For example: stamp duty, real estate, automotive patent, gross income.
- **Phone Type**: Types of phone. For example: landline, mobile, fax.
- **Taxpayer Phone Numbers:** Phone numbers associated to taxpayers for contacting them.
- **Agency Phone Numbers:** Phone numbers associated to agencies for contacting them.

These entities were created as tables into TaxesCollection DB, along with respective constraints and indexes.

The detailed information on tables, attributes and constraints is specified in a data dictionary Excel spreadsheet.

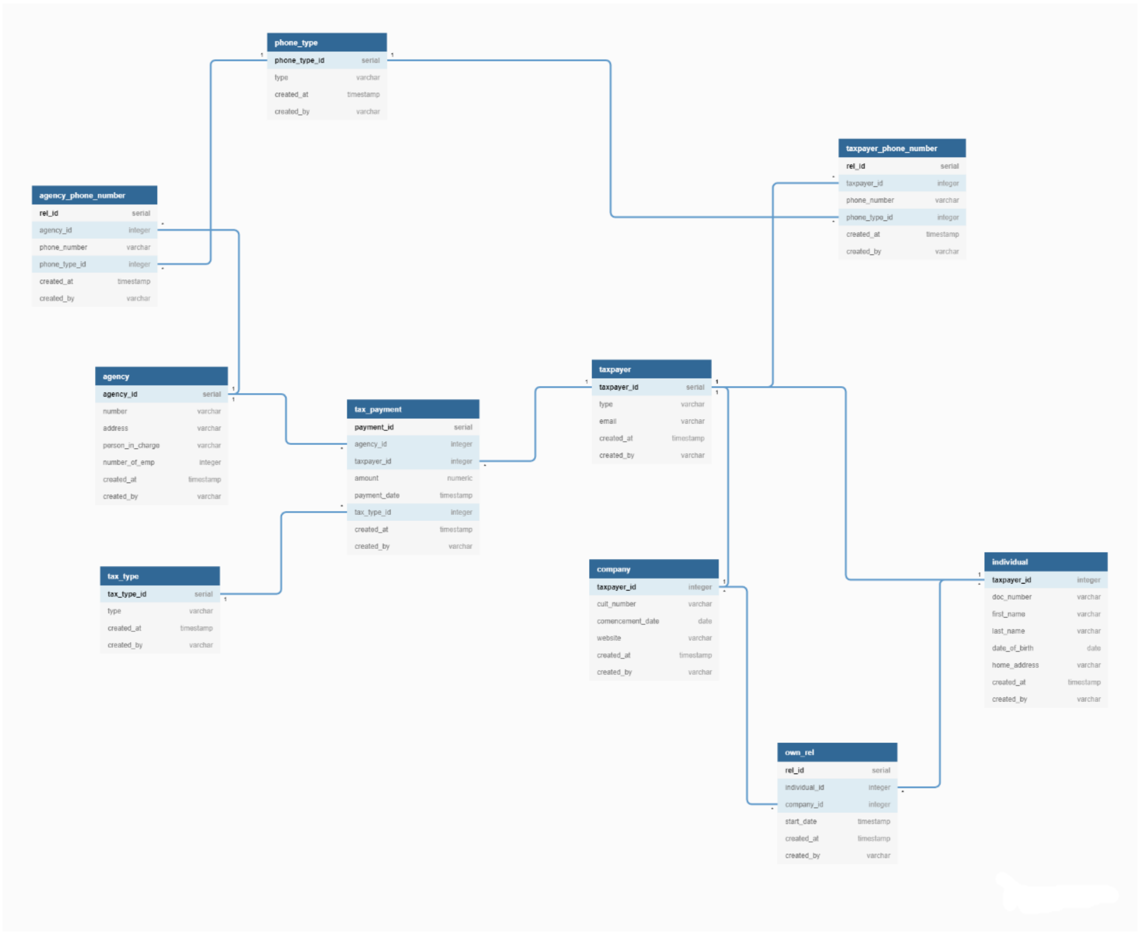
TaxesCollection_DB
_Data_Dictionary.xls

The script to create the database with all it's objects is in following SQL file.

Taxes_Collection_D
B.sql

Entity Relationship Diagram (ERD)

The figure below shows the ERD for the DataCollection Database



Database Interactions

Functions have been created in TaxesCollection DB that can be used to obtain following information:

How much tax did certain taxpayer pay per month.

The function ***totaltaxpermonth*** has been created in order to return the information

Input Parameters

- **Taxpayer Id**: The id of the taxpayer whose total amount per month has to be calculated. This is a **required** parameter.
- **Year**: The year for which the total amount wants to be calculated. This is an **optional** parameter.
- **Month**: The month for which the total amount wants to be calculated. This is an **optional** parameter. **Important:** The month has to be always informed along with the year parameter.

Function Output

A table that includes following columns:

- **Tax Amount**: Total amount payed by a taxpayer whose id has been received as input parameter. This amount is totalized by year and month.
- **Year**: The year for which the total amount has been calculated.
- **Month**: The month of the year, for which the total amount has been calculated.

The resulting table is ordered in descendent way by Year and Month.

If the input taxpayer ID does not exist in DB, the function will return ***no data***.

Call Statement

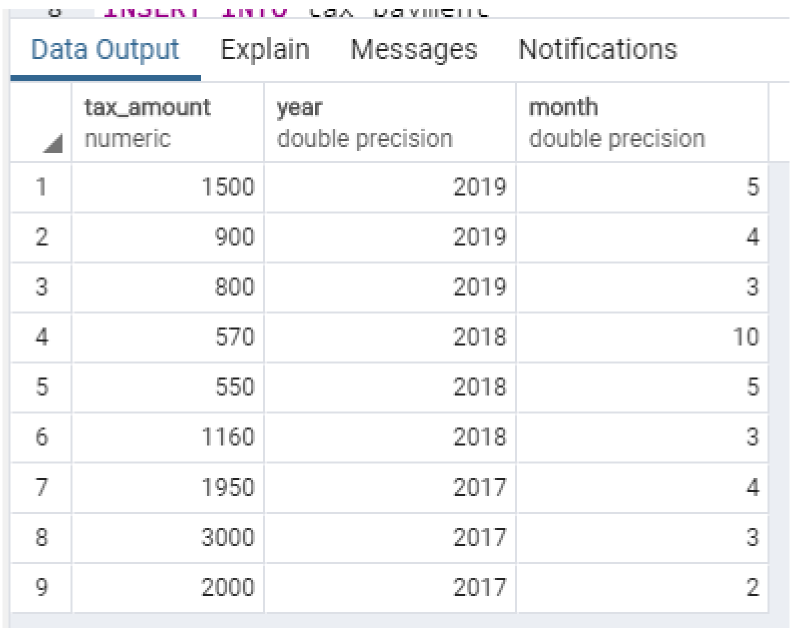
Following call statements are valid:

Total amount without considering particular year and month

For taxpayer id '42', the call statement would be:

*SELECT * FROM TOTALTAXPERMONTH(42);*

And the output will include the total amount payed by taxpayer id '42' in all the years and months:



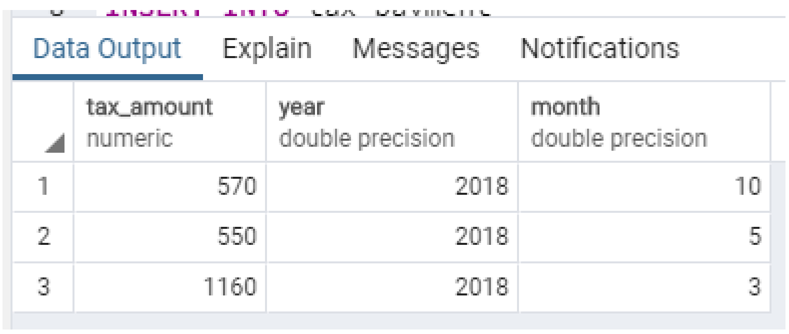
	tax_amount numeric	year double precision	month double precision
1	1500	2019	5
2	900	2019	4
3	800	2019	3
4	570	2018	10
5	550	2018	5
6	1160	2018	3
7	1950	2017	4
8	3000	2017	3
9	2000	2017	2

Total amount for a particular year

For taxpayer id ‘42’ and year ‘2018’, the call statement would be:

*SELECT * FROM TOTALTAXPERMONTH(42,2018);*

And the output will include the total amount payed by taxpayer id ‘42’ in the months for year ‘2018’:



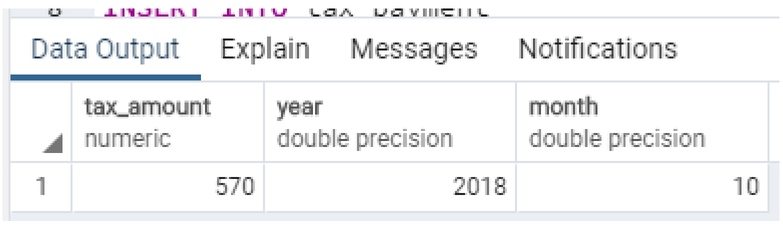
	tax_amount numeric	year double precision	month double precision
1	570	2018	10
2	550	2018	5
3	1160	2018	3

Total amount for a particular year and month

For taxpayer id ‘42’ and month 10 of year ‘2018’, the call statement would be:

*SELECT * FROM TOTALTAXPERMONTH(42,2018,10);*

And the output will include the total amount payed by taxpayer id ‘42’ in month ‘10’ for year ‘2018’:



	tax_amount numeric	year double precision	month double precision
1	570	2018	10

Total tax paid for all companies person is owner or co-owner of.

The function ***ownedcompaniestaxpayed*** has been created in order to return the information.

Input Parameters

- **Individual Id**: The id of the individual that owns the companies whose total amount payed has to be calculated.

Function Output

A table that includes following columns:

- **Amount**: Total amount payed by all the companies owned by the individual whose id has been received as input parameter.

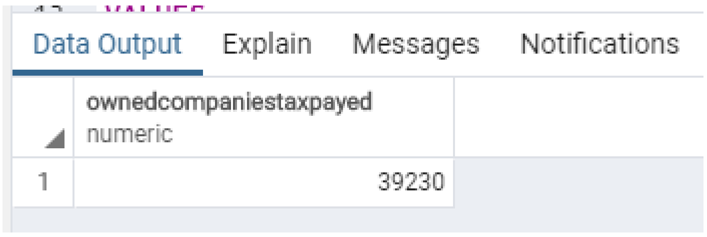
If the input ID does not correspond to an individual or does not exist in DB, the function will return ***NULL***.

Call Statement

A valid call statement for individual id '5' would be:

*SELECT * FROM OWNEDCOMPANIESTAXPAYED(5);*

Output Display



```
   ownedcompaniestaxpayed
   numeric
1                   39230
```

Inserting a new company along with defining its owner(s).

The function ***insertcompany*** has been created in order to insert a new company along with its related information.

The function can be called passing different input structure:

- Input data as separate values .
- Input data as XML structure.

Insertcompany function with input data as separate values

Input Parameters

- **Email**: Email address of the company to insert.
- **Cuit Number:** CUIT number of the new company.
- **Comencement Date:** Starting date of operations for new company.
- **Website:** New company's website address.
- **Individuals:** A list of individual ids that will be related to the new company as the owners. **Important:** All the individual IDs included in this list **must** exist in table "Individuals" before executing this function.
- **Phone Numbers:** A list of phone numbers of the new company. These numbers will be related to the company by the function.

- **Phone Types:** A list of pone type IDs that will be considered when relating the pone numbers to the new company. **<u>Important:</u>** All the pone type IDs included in this list **must** exist in table "Phone_Type" before executing this function.

Insertcompany function with input data as XML structure

Input XML

The same input data mentioned above can be passed to the function using a XML format that **must** respect the XSD structure in file

company_data.xsd

Function Output

This function does not return any value. It does following tasks:

- Creates a new taxpayer in Taxpayer table, with input information.
- Creates a new company in Company table, with input information.
- Creates as many Individual/Company ownership relationships as individuals are informed in the Individuals input list.
    - The individuals must exist in table *Individuals* before executing this function. ***This function does not créate individuals***.
- Creates as many Phone Number/Company relationships as pone numbers are informed in the Phone Numbers input list.
    - For each relation a phone type id will be used at the moment of relationship creation.
    - The phone type ids must exist in table *Phone_Type* before executing this function. ***This function does not créate pone types***.
    - Each phone type informed as input will need a correspondent pone_type_id. Base don this, both input lists, Phone Numbers and Phone Type Ids, will need a correspondence between each elements, i.e.:
        - If the phone list has 2 phone numbers, the phone type ids list will need to have 2 phone type ids, where the 1st phone type id correspond to the first phone number and the 2nd phone type id to the 2nd pone number.

Error Management

Following errors can be returned by this function in case of any failure with the input information:

- If the list of individuals ids is not informed, following error is shown: 'At least 1 individual that owns the company must be informed'.
- If the list of phone numbers is not informed, following error is shown: 'At least 1 phone number for the company must be informed'.
- If the list of phone type ids is not informed, following error is shown: 'Phone type ids must be informed for phone numbers'.
- If quantity of elements in phone numbers list is different than the quantity of phone type ids list, following error is shown: 'Quantity of phone type ids does not match the informed quantity of phone numbers'.

- Consistency error messages generated by DB at the moment of data insertion, based on defined
  - Constraints
    - Company’s comencement date must not be after current date time.
    - Email must be in the right standard email’s format.
  - Unique Indexes
    - Company’s unique CUIT number is validated.
  - Foreign Keys
    - Individual Ids existence in DB is validated.
    - Phone Type Ids existence in DB is validated.
- Transaction management has been provided in order to ensure that if any error occurs between the multiple insertions in this function, complete transaction is rolled back and no insertion is completed for any involved table.

Call Statement

For function that receives separated values as input

For example, if a new company has to be created into TaxesCollection DB with following information:

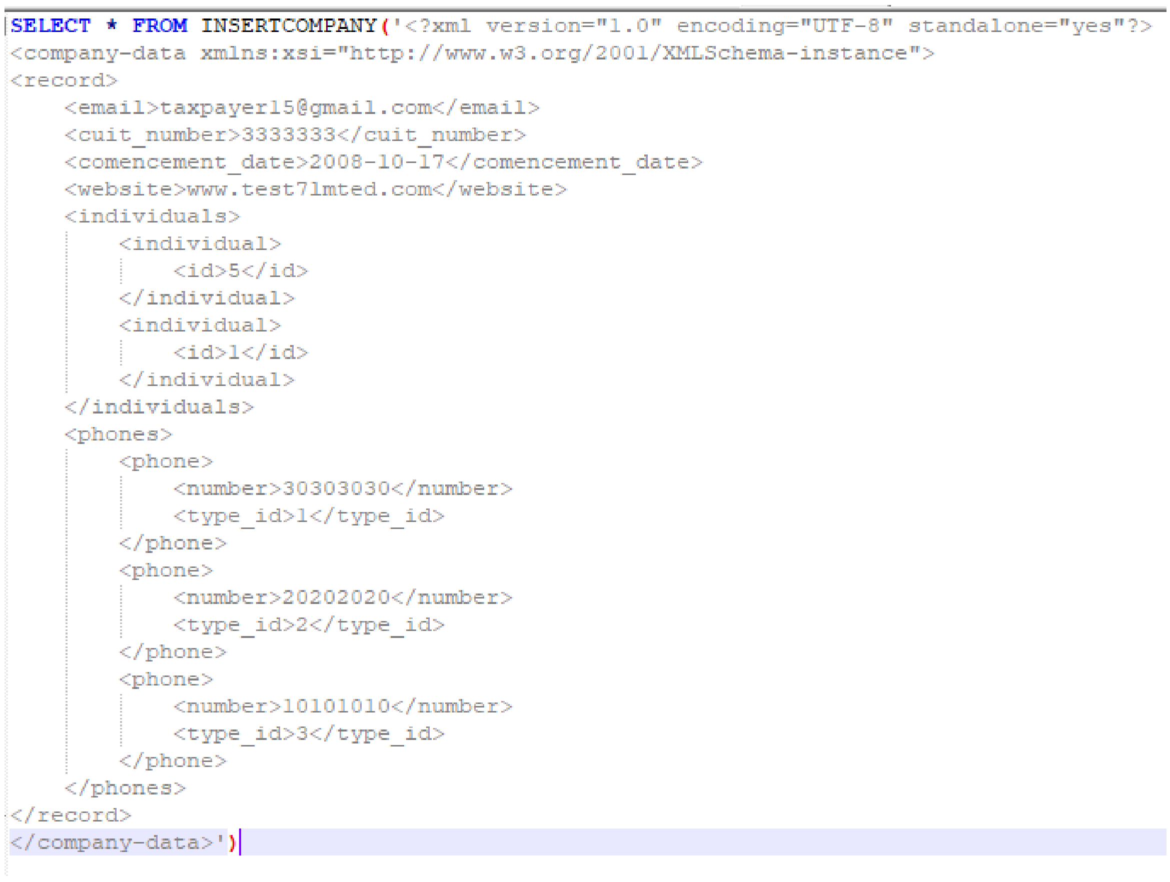
- Email: taxpayer15@gmail.com
- Cuit Number: 3333333
- Comencement Date: 2008-10-17
- Website: www.test7lmted.com
- Owner Individual Ids: ‘1’ and ‘5’
- Phone Numbers: ‘10101010’, ‘20202020’ and ‘30303030’
- Phone Type Ids: ‘3’, ‘2’ and ‘1’
  - Type Id ‘3’ (Fax) is for phone number ‘10101010’
  - Type Id ‘2’ (Mobile) is for phone number ‘20202020’
  - Type Id ‘1’ (Landline) is for phone number ‘30303030’

A valid call statement would be:

*SELECT * FROM INSERTCOMPANY('taxpayer15@gmail.com','3333333','2008-10-17','www.test7lmted.com','{1,5}','{10101010,20202020,30303030}','{3,1,2}');*

For function that receives an XML structure as input

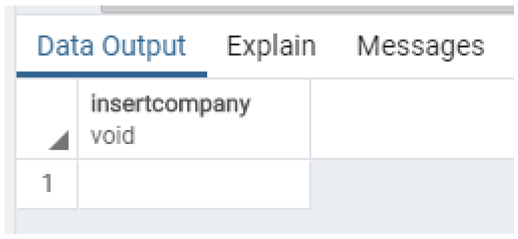
Using the same example as mentioned above, a valid call structure would be:



```
SELECT * FROM INSERTCOMPANY('<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<company-data xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<record>
    <email>taxpayer15@gmail.com</email>
    <cuit_number>3333333</cuit_number>
    <comencement_date>2008-10-17</comencement_date>
    <website>www.test7lmted.com</website>
    <individuals>
        <individual>
            <id>5</id>
        </individual>
        <individual>
            <id>1</id>
        </individual>
    </individuals>
    <phones>
        <phone>
            <number>30303030</number>
            <type_id>1</type_id>
        </phone>
        <phone>
            <number>20202020</number>
            <type_id>2</type_id>
        </phone>
        <phone>
            <number>10101010</number>
            <type_id>3</type_id>
        </phone>
    </phones>
</record>
</company-data>')
```

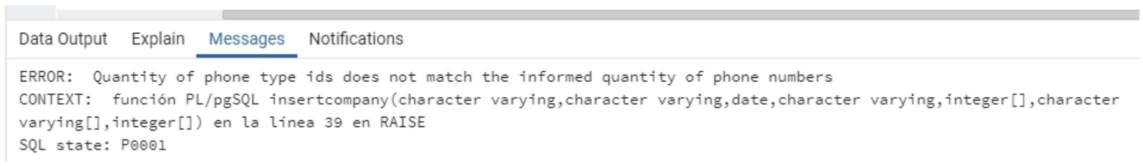
Output Display

If the company insertion is succesfull, the output received is:



	insertcompany void
1	

If there’s any error during the insertion process, one example output will be shown as follows:



```
ERROR:  Quantity of phone type ids does not match the informed quantity of phone numbers
CONTEXT:  función PL/pgSQL insertcompany(character varying,character varying,date,character varying,integer[],character
varying[],integer[]) en la línea 39 en RAISE
SQL state: P0001
```

Inserting new tax payment along with necessary validation.

The function ***InsertTaxPayment*** has been created in order to execute the new tax payment creation in DB.

Input Parameters

- **Agency Id**: Unique Id of an agency. This Id must exist in Agency table.

- **Taxpayer Id:** Unique Id of a taxpayer. This Id must exist in Taxpayer table.
- **Amount:** Amount of the tax payment.
- **Payment Date:** Tax payment date.
- **Tax Type Id:** A tax type id for the tax payment. This Id must exist in Tax_Type table.

Function Output

This function does not return any value. It does following tasks:

- Creates a new tax payment in Tax_Payment table, with input information.
- This function ***does not creates*** agencies, taxpayers nor tax types.

Error Management

Following errors can be returned by this function in case of any failure with the input information:

- Consistency error messages generated by DB at the moment of data insertion, based on defined
  - Constraints
    - Amount must be a positive value.
    - Payment Date must not be after current date time.
  - Unique Indexes
    - Unique combination of agency id, taxpayer id, payment date and tax type id is validated.
  - Foreign Keys
    - Agency Id existence in DB is validated.
    - Taxpayer Id existence in DB is validated.
    - Tax Type Id existence in DB is validated.

Call Statement

For example, if a tax payment with following information has to be created in TaxesCollection DB:

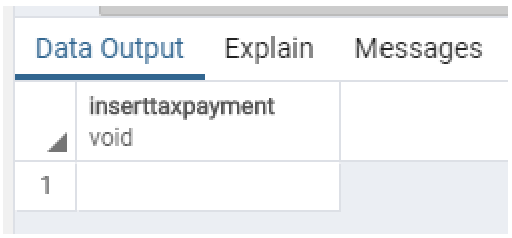
- Agency Id = '2'
- Taxpayer Id = '38'
- Amount = 8000
- Payment Date = 2019-09-13
- Tax Type Id = 4 ('Gross Income')

A valid call statement would be

*SELECT * FROM INSERTTAXPAYMENT('2','38',8000,'2019-09-13',4);*

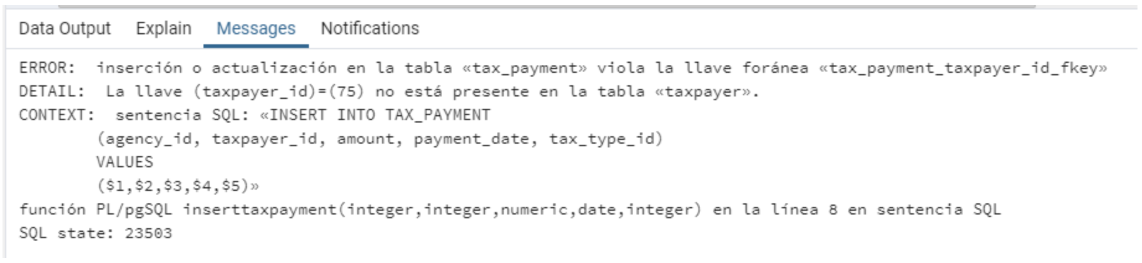
Output Display

If the tax payment insertion is succesfull, the output received is:



```
   inserttaxpayment
   void
1
```

If there's any error during the insertion process, one example output will be shown as follows:



```
ERROR:  inserción o actualización en la tabla «tax_payment» viola la llave foránea «tax_payment_taxpayer_id_fkey»
DETAIL:  La llave (taxpayer_id)=(75) no está presente en la tabla «taxpayer».
CONTEXT:  sentencia SQL: «INSERT INTO TAX_PAYMENT
        (agency_id, taxpayer_id, amount, payment_date, tax_type_id)
        VALUES
        ($1,$2,$3,$4,$5)»
función PL/pgSQL inserttaxpayment(integer,integer,numeric,date,integer) en la línea 8 en sentencia SQL
SQL state: 23503
```

Database Replication

The replication method used is ***PostgreSQL Logical Replication***. This replication method allows to do a real-time replication between two databases (master DB and replica DB) for inserts, updates and deletes operations at table level.

This method is based on

- **A publish creation** in master DB. This publish has to include each table to be replicated from master DB.
- **A subscription creation** in replica DB, that points to the corresponding publish created in master DB. Based on this, every modification done in master taxesCollection DB will be replicated in corresponding tables on replica TaxesCollection DB.

In order to ensure a real replication process, a 2nd PostgreSQL server (***PostgreSQL 11 StandBy***) has been installed on the same machine as where the original server is running (PostgreSQL 11).

Same TaxesCollection database structure has been created in ***PostgreSQL 11 Standby*** server. This DB will be the destination of the replications from master DB.

The replica TaxesCollection DB will be access by default with a readonly user called "**readonly**". This will ensure two things:

- The connected user can only query information from the DB.
- The replication process can be done as it was created with a superuser (postgres user in replica DB)

Environments Information

Master DB

- Host: localhost
- Port: 5432
- Server Name: PostgreSQL 11
- Server Type: PostgreSQL
- Version: PostgreSQL 11.4, compiled by Visual C++ build 1914, 64-bit
- Super User: postgres/billboard
- Replication User: rep/billboard

Replica DB (Readonly)

- Host: localhost
- Port: 5433
- Server Name: PostgreSQL 11 StandBy
- Server Type: PostgreSQL
- Version: PostgreSQL 11.5, compiled by Visual C++ build 1914, 64-bit
- User: readonly/readonly
- Super User: postgres/billboard

Replication tasks in master DB.

Following tasks have been executed in order to prepare master DB for replication:

1. Create a user for replication execution. In this case a user "rep" has been created.

```
-- Creates role for replication
CREATE ROLE rep WITH REPLICATION LOGIN PASSWORD '<password>';
```

2. Assign privileges to the replication user in order to have all control of the tables to be replicated

```
-- Grants to replication role on tables to be replicated
GRANT ALL ON AGENCY TO rep;
GRANT ALL ON TAXPAYER TO rep;
GRANT ALL ON TAX_TYPE TO rep;
GRANT ALL ON AGENCY_PHONE_NUMBER TO rep;
GRANT ALL ON COMPANY TO rep;
GRANT ALL ON INDIVIDUAL TO rep;
GRANT ALL ON OWN_REL TO rep;
GRANT ALL ON PHONE_TYPE TO rep;
GRANT ALL ON TAX_PAYMENT TO rep;
GRANT ALL ON TAXPAYER_PHONE_NUMBER TO rep;
```

3. Create a publication

```
-- Creates publish entity
CREATE PUBLICATION PUB_TAXES_COLLECTION;
```

4. Add the tables to be replicated to the publication

```
-- Adds tables to publish entity to be automatically replicated
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE AGENCY;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE TAXPAYER;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE TAX_TYPE;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE AGENCY_PHONE_NUMBER;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE COMPANY;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE INDIVIDUAL;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE OWN_REL;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE PHONE_TYPE;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE TAX_PAYMENT;
ALTER PUBLICATION PUB_TAXES_COLLECTION ADD TABLE TAXPAYER_PHONE_NUMBER;
```

By implementing the mentioned tasks, the master DB was configured to do replication of the defined tables.

Replication tasks in replica DB.

In replica DB the only task executed was creating the subscription to the master DB publication.

```
--Create subscription from standby DB (hostname.5433) to master db (hostname.5432)
CREATE SUBSCRIPTION SUBS_TAXES_COLLECTION CONNECTION 'host=127.0.0.1 user=rep password=<password> port=5432
    dbname=TaxesCollection' PUBLICATION PUB_TAXES_COLLECTION;
```

This ensures that replica DB will be synchronized with master DB at real-time after each insert, update, delete operations executed in master DB.

Automatic Backups

A batch file was created to authomatize backup execution for TaxesCollection DB. The tasks executed by this file are:

1. Define a file name for the each backup, based on the current date and time. The file will have a name like: ***taxescollection<yyyymmddhhmm>.backup***
2. Invoke the backup PostgreSQL command ("pg_dump") to execute the backup and generate the mentioned file in task 1.

The batch file is called ***taxescollection_bkp.bat***. It has been uploaded into the Gitlab repository. The file is:

taxescollection_bkp
.bat

A windows task has been created in order to automatically execute the batch file to generate a TaxesCollection DB every **2** hours.