*E1.7*

Base de Datos de Históricos

*v1.0*

04 Febrero 2020

Para cualquier consulta o cuestión pendiente sobre la oferta la persona de contacto es:

|  |  |  |
| --- | --- | --- |
| Nombre | Teléfono | e-mail |
| José Vicente Higón Valero | +34 600 55 44 74 | [jvhigon@gvsig.com](mailto:jvhigon@gvsig.com) |

Información general:

Asociación gvSIG

CIF G98125362

Calle Ángel Guimerá 61, puerta 3

[http://www.gvsig.com](http://www.gvsig.com/)

Control de cambios

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Versión | Responsable | Organismo | Descripción del cambio | Fecha |
| v.1.0 | Nacho Brodin | Asoc gvSIG | Documento inicial | 04/02/20 |
| v1.1 | Nacho Brodin | Asoc gvSIG | Esquema, triggers y completar información del modelo | 24/02/20 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

**Índice**

[1. Actual Base de datos de históricos](#_gjdgxs) **4**

[2. Modelo propuesto](#_30j0zll) **6**

[2.1Descripción del modelo](#_k69laqc9szwx) 6

[2.2 Diagrama UML](#_1fob9te) 6

[2.3 Forma de inserción en histórico](#_58iuvr5fae3i) 8

[Actualización mediante triggers](#_cp8nfh9b3j65) 8

[Actualización a través del administrador de direcciones](#_pdr2j050qq3c) 8

[Histórico vs log](#_bae2x0xh6fqq) 8

[2.4 Inserciones masivas](#_d3cxotg9pf2h) 9

[2.5 Esquema “históricos”](#_fwv0boff5rr8) 9

[2.6 Procedimientos almacenados para históricos](#_cp2yra8rofau) 14

[2.7 Procedimiento para inserción en alias](#_q7z9gzpqbvhv) 19

[2.7 Triggers para la actualización e inserción](#_b5ng8qq0i5xp) 21

# 1. Actual Base de datos de históricos

La base de datos actual contiene tres tablas relacionadas con logs e históricos:

* log
* log\_base
* histórico

La primera de ellas no es usada por ninguno de los triggers.

Las siguientes tablas disponen de triggers que permiten volcar información en las tablas de log e histórico a través de los triggers que se indican.

|  |  |  |
| --- | --- | --- |
| **Tabla** | **trigger** | **Acción** |
| alias | log\_transac\_alias | Guarda en log\_base al insertar, borrar o actualizar |
| alias\_numero\_puerta | log\_transac\_alias\_nro | Guarda en log\_base al insertar, borrar o actualizar |
| calle | calle\_historico | Guarda en histórico antes de insertar, borrar o actualizar |
|  | log\_transac\_calle | Guarda en log\_base al insertar, borrar o actualizar |
| iso\_metadata\_reference | update\_imr\_timestamp | Actualiza columna timestamp |
| localidades\_no\_oficiales | log\_transac\_loc | Guarda en log\_base al insertar, borrar o actualizar |
| punto | log\_transac | Guarda en log\_base al insertar, borrar o actualizar |
|  | punto\_historico | Guarda en histórico antes de insertar, borrar o actualizar |
|  | trigger\_punto | Cuando cambia el punto actualiza el CP cruzando con la tabla codigos\_clasificacion |
|  | trigger\_punto\_notable | Inserta o actualiza en puntos\_notables\_a\_procesar cuando tiene nombre\_inmueble o se actualiza este campo |
|  | trigger\_punto\_zona | Inserta o actualiza el campo zona\_paquete cruzando con la tabla zonas\_paquetes |

Los campos contenidos en log\_base son los siguientes:

* id\_log
* tabla
* operación
* valor\_anterior
* nuevo\_valor
* fecha\_cambio
* usuario
* id
* geom

Los campos de histórico son los siguientes:

* id
* nombre\_tabla
* id\_entabla
* etiqueta
* desde
* hasta
* geom
* usu

# 2. Modelo propuesto

## 2.1Descripción del modelo

Los problemas del modelo existente son básicamente cuatro:

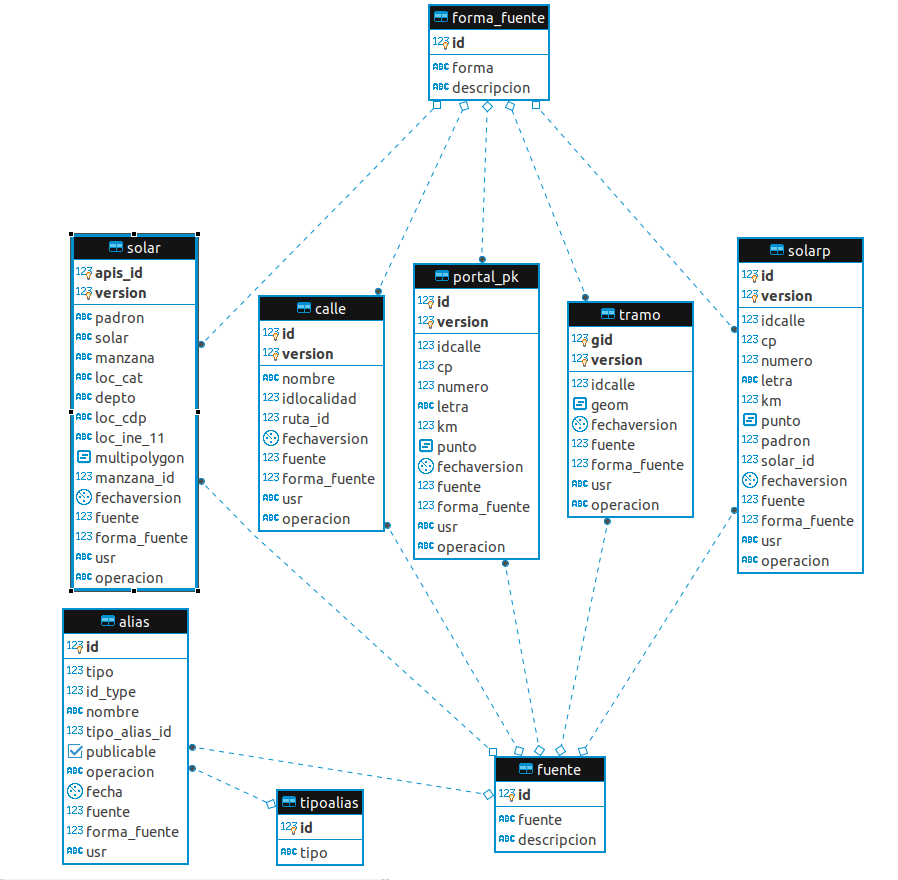
* Las tablas no contienen toda la información de los cambios
* La tabla de logs crece desmesuradamente ya que contiene toda la información y se hace pesada de manejar y consultar
* Es difícil extraer información si se quieren realizar estadísticas
* No se puede realizar la trazabilidad de un elemento de todos sus cambios. Por ejemplo: “para una calle ver cuando ha cambiado el campo nombre a partir de una fecha dada”

Se propone un modelo en el que cada tabla es un elemento de la base de datos al que se le quiere dar trazabilidad (calle, solar, portal, etc) e incorpora todos sus campos. En cada operación de inserción, actualización o borrado se guardan:

* los datos modificados
* un campo de versión que incrementa la versión anterior
* la fecha en la que se hizo el cambio
* el usuario que hizo el cambio
* la forma en la que se produjo el cambio

Con este modelo eliminamos las desventajas del modelo anterior

## 2.2 Diagrama UML



En este modelo los elementos del callejero (solar, calle, portal, etc) no sólo tienen un id como clave primaria sino que además tienen un id y una versión. Es decir para un elemento hay múltiples versiones del mismo a lo largo del tiempo.

En la tabla alias se vuelcan todos los datos de históricos de las seis tablas de alias que hay en el esquema budu. Estas vuelcan sus datos a través de un trigger.

Las tablas solar, solarp, calle, portal\_pk y tramo representan a las tablas con el mismo nombre del esquema budu. Tienen los mismos campos más los necesarios para el histórico.

* **fecha**: Fecha en que se realiza la inserción
* **fuente**: Perfil de usuario que ha realizado la operación definido en la tabla “fuente” del esquema budu
* **forma\_fuente**: Forma en la que se hace la operación definida en la tabla “forma\_fuente” del esquema budu
* **usr**: Usuario que realiza la operación

## 2.3 Forma de inserción en histórico

### Actualización mediante triggers

Este sistema consiste en la creación de triggers en los que cada vez que se inserte, actualice o borre en la tabla del esquema budu se realiza la operación de histórico de forma automática. Esta forma de inserción puede ser buena para alias y otras tablas que puedan incorporarse que no sean parte del modelo que intervienen en las formas canónicas (manzana, calle, etc). El problema de este sistema es que complicado saber quien realiza la operación y a través de qué medio. Sólo es posible almacenar el usuario de la base de datos.

### Actualización a través del administrador de direcciones

Al realizar la actualiza a través del administrador de direcciones es posible almacenar el usuario que lo realiza de gvSIGOnline. Estos usuarios pueden asignarse a grupos que coinciden con los tipos que aparecen en la tabla “fuente”. De esta forma puede realizarse la inserción en histórico de los datos de usuario, fuente, y forma\_fuente de forma precisa.

### Histórico vs log

Hay dos aproximaciones posibles para guardar información de copia en una base de datos:

**Sistema de logs**: cada registro que cambia en una tabla de budu se introduce a la vez en histórico. Esta forma es típica de un sistema de logs en el que cada dato que se introduce en la base de datos se guarda una copia. Esto hace que en el sistema de logs sea un reflejo de la base de datos de producción pero con alguna información adicional, quien hizo el asiento, en que fecha o desde que sistema lo hizo. De esta forma se puede llevar una trazabilidad de los cambios.

**En forma de histórico**: cada registro que se modifica en una tabla de budu antes de machacar el existente se copia este a una base de datos de histórico con una nueva versión. Esto hace que en la base de datos de producción se tenga la última versión de los elementos y en la base de datos de histórico sólo versiones anteriores.

En una idea inicial se iba a aplicar la segunda aproximación porque la idea era que para cada elemento del callejero poder recuperar dicho elemento en su versión actual (contenido en el esquema budu) y todas sus versiones anteriores (contenidas en el histórico). Finalmente se deshecha la idea y se aplica la primera aproximación por el siguiente motivo: el modelo histórico no guarda timestamp de cuando se hace la operación ni quien la hace a no ser que estos campos estén en la base de datos de producción, ya que cuando se hace la copia a histórico se hace no de lo que se inserta en ese momento sino de lo que ya estaba insertado. Como estos campos no existen porque no se considera necesario introducir más campos al modelo y dado que se quiere dejar registro de quien y cuando se realizan las operaciones se opta por el modelo de log.

## 2.4 Inserciones masivas

Se propone el uso de la funcionalidad de inserción masiva de puntos desarrollada para gvSIGOnline para la inserción. Esta forma permite la actualización de históricos de forma automática

## 2.5 Esquema “históricos”

|  |
| --- |
| CREATE SCHEMA IF NOT EXISTS historico;  CREATE TABLE historico.calle (  id bigint NOT NULL,  nombre text NOT NULL,  idlocalidad integer,  ruta\_id integer,  version integer NOT NULL,  fechaversion timestamp with time zone NOT NULL,  fuente integer,  forma\_fuente integer,  usr character varying(50),  operacion character varying(25),  CONSTRAINT calle\_historico\_pkey PRIMARY KEY (id, version),  CONSTRAINT historico\_calle\_forma\_fuente FOREIGN KEY (forma\_fuente)  REFERENCES budu.forma\_fuente (id),  CONSTRAINT historico\_calle\_fuente FOREIGN KEY (fuente)  REFERENCES budu.fuente (id) );  CREATE TABLE historico.tramo (  gid bigint NOT NULL,  idcalle bigint NOT NULL,  geom geometry(MultiLineString,32721),  version integer NOT NULL,  fechaversion timestamp with time zone NOT NULL,  fuente integer,  forma\_fuente integer,  usr character varying(50),  operacion character varying(25),  CONSTRAINT tramo\_historico\_pkey PRIMARY KEY (gid, version),  CONSTRAINT historico\_calle\_forma\_fuente FOREIGN KEY (forma\_fuente)  REFERENCES budu.forma\_fuente (id),  CONSTRAINT historico\_calle\_fuente FOREIGN KEY (fuente)  REFERENCES budu.fuente (id) );  CREATE TABLE historico.portal\_pk (  id bigint,  idcalle bigint,  cp integer,  numero integer,  letra character varying(5),  km double precision,  punto geometry,  version integer NOT NULL,  fechaversion timestamp with time zone NOT NULL,  fuente integer,  forma\_fuente integer,  usr character varying(50),  operacion character varying(25),  CONSTRAINT portal\_pk\_historico\_pkey PRIMARY KEY (id, version),  CONSTRAINT historico\_portal\_pk\_forma\_fuente FOREIGN KEY (forma\_fuente)  REFERENCES budu.forma\_fuente (id),  CONSTRAINT historico\_portal\_pk\_fuente FOREIGN KEY (fuente)  REFERENCES budu.fuente (id) );  CREATE TABLE historico.solar (  apis\_id bigint NOT NULL,  padron character varying(1024),  solar character varying(1024),  manzana character varying(1024),  loc\_cat character varying(1024),  depto character varying(1024),  loc\_cdp character varying(1024),  loc\_ine\_11 character varying(1024),  multipolygon geometry(MultiPolygon,32721),  manzana\_id integer,  version integer NOT NULL,  fechaversion timestamp with time zone NOT NULL,  fuente integer,  forma\_fuente integer,  usr character varying(50),  operacion character varying(25),  CONSTRAINT solar\_pkey PRIMARY KEY (apis\_id, version),  CONSTRAINT historico\_solar\_forma\_fuente FOREIGN KEY (forma\_fuente)  REFERENCES budu.forma\_fuente (id),  CONSTRAINT historico\_solar\_fuente FOREIGN KEY (fuente)  REFERENCES budu.fuente (id) );  CREATE TABLE historico.solarp (  id bigint,  idcalle integer,  cp integer,  numero integer,  letra character varying(5),  km double precision,  punto geometry,  padron integer,  solar\_id integer,  version integer NOT NULL,  fechaversion timestamp with time zone NOT NULL,  fuente integer,  forma\_fuente integer,  usr character varying(50),  operacion character varying(25),  CONSTRAINT solarp\_pkey PRIMARY KEY (id, version),  CONSTRAINT historico\_solarp\_forma\_fuente FOREIGN KEY (forma\_fuente)  REFERENCES budu.forma\_fuente (id),  CONSTRAINT historico\_solarp\_fuente FOREIGN KEY (fuente)  REFERENCES budu.fuente (id) );  CREATE TABLE historico.tipoalias (  id integer,  tipo character varying(25),  CONSTRAINT tipoalias\_pkey PRIMARY KEY (id) );  INSERT INTO historico.tipoalias VALUES (0, 'CALLE'); INSERT INTO historico.tipoalias VALUES (1, 'DEPARTAMENTO'); INSERT INTO historico.tipoalias VALUES (2, 'ENTIDAD COLECTIVA'); INSERT INTO historico.tipoalias VALUES (3, 'LOCALIDAD GEO'); INSERT INTO historico.tipoalias VALUES (4, 'MANZANA'); INSERT INTO historico.tipoalias VALUES (5, 'NUMERO PUERTA'); INSERT INTO historico.tipoalias VALUES (6, 'PUNTO NOTABLE'); INSERT INTO historico.tipoalias VALUES (7, 'SOLARP'); INSERT INTO historico.tipoalias VALUES (8, 'TRAMO RUTA');  CREATE TABLE historico.alias (  id serial NOT NULL ,  tipo integer,   id\_type integer,  nombre text,  tipo\_alias\_id integer,  publicable boolean,  operacion character varying(25),  fecha timestamp with time zone,  fuente integer,  forma\_fuente integer,  usr character varying(50),  CONSTRAINT typealias\_pkey PRIMARY KEY (id),  CONSTRAINT historico\_alias\_fuente FOREIGN KEY (fuente)  REFERENCES budu.fuente (id),  CONSTRAINT historico\_alias\_tipoalias FOREIGN KEY (tipo)  REFERENCES historico.tipoalias (id) );  CREATE TABLE historico.ruta (  id integer NOT NULL,  numero character varying,  nombre character varying,  longitud character varying(50),  origen character varying(50),  destino character varying(50),  tipo character varying,  tipo\_anexo character varying,  tipo\_obs character varying,  geom geometry(MultiLineString,32721),  operacion character varying(25),  version integer NOT NULL,  fecha timestamp with time zone NOT NULL,  fuente integer,  forma\_fuente integer,  usr character varying(50),  CONSTRAINT rutaid\_pkey PRIMARY KEY (id, version),  CONSTRAINT historico\_ruta\_forma\_fuente FOREIGN KEY (forma\_fuente)  REFERENCES budu.forma\_fuente (id),  CONSTRAINT historico\_ruta\_fuente FOREIGN KEY (fuente)  REFERENCES budu.fuente (id) ); |

## 2.6 Procedimientos almacenados para históricos

|  |
| --- |
| CREATE OR REPLACE FUNCTION historico.log\_portal\_pk(idportal integer, src integer, src\_type integer, usr varchar, op varchar)  RETURNS integer AS $BODY$ DECLARE  pkversion integer;  p budu.portal\_pk%ROWTYPE;  nrows integer; BEGIN  IF op = 'FIRST' THEN  SELECT count(\*) INTO nrows FROM historico.portal\_pk WHERE id = idportal;  IF nrows = 0 THEN  SELECT \* INTO p FROM budu.portal\_pk WHERE id = idportal;  INSERT INTO historico.portal\_pk VALUES(idportal, p.idcalle, p.cp, p.numero, p.letra, p.km, p.punto, 1, now(), src, src\_type, usr, op);  return 1;  END IF;  ELSE  SELECT MAX(version) INTO pkversion FROM historico.portal\_pk WHERE id = idportal;  IF pkversion IS NULL THEN  pkversion := 0;  END IF;  pkversion := pkversion + 1;  SELECT \* INTO p FROM budu.portal\_pk WHERE id = idportal;  INSERT INTO historico.portal\_pk VALUES(idportal, p.idcalle, p.cp, p.numero, p.letra, p.km, p.punto, pkversion, now(), src, src\_type, usr, op);  return pkversion;  END IF;  return 0; END; $BODY$  LANGUAGE plpgsql VOLATILE  COST 100;     CREATE OR REPLACE FUNCTION historico.log\_calle(idcalle integer, src integer, src\_type integer, usr varchar, op varchar)  RETURNS integer AS $BODY$ DECLARE  versioncalle integer;  p budu.calle%ROWTYPE;  nrows integer; BEGIN  IF op = 'FIRST' THEN  SELECT count(\*) INTO nrows FROM historico.calle WHERE id = idcalle;  IF nrows = 0 THEN  SELECT \* INTO p FROM budu.calle WHERE id = idcalle;  INSERT INTO historico.calle VALUES(idcalle, p.nombre, p.idlocalidad, p.ruta\_id, 1, now(), src, src\_type, usr, op);  return 1;  END IF;  ELSE  SELECT MAX(version) INTO versioncalle FROM historico.calle WHERE id = idcalle;  IF versioncalle IS NULL THEN  versioncalle := 0;  END IF;  versioncalle := versioncalle + 1;  SELECT \* INTO p FROM budu.calle WHERE id = idcalle;  INSERT INTO historico.calle VALUES(idcalle, p.nombre, p.idlocalidad, p.ruta\_id, versioncalle, now(), src, src\_type, usr, op);  return versioncalle;  END IF;  return 0; END; $BODY$  LANGUAGE plpgsql VOLATILE  COST 100;    CREATE OR REPLACE FUNCTION historico.log\_tramo(idc integer, src integer, src\_type integer, usr varchar, op varchar)  RETURNS integer AS $BODY$ DECLARE  versiontramo integer;  p budu.tramo%ROWTYPE;  nrows integer; BEGIN  IF op = 'FIRST' THEN  SELECT count(\*) INTO nrows FROM historico.tramo WHERE idcalle = idc;  IF nrows = 0 THEN  FOR p IN SELECT \* FROM budu.tramo WHERE idcalle = idc LOOP  INSERT INTO historico.tramo VALUES(p.gid, idc, p.geom, 1, now(), src, src\_type, usr, op);  END LOOP;  return 1;  END IF;  ELSE  SELECT MAX(version) INTO versiontramo FROM historico.tramo WHERE idcalle = idc;  IF versiontramo IS NULL THEN  versiontramo := 0;  END IF;  versiontramo := versiontramo + 1;  FOR p IN SELECT \* FROM budu.tramo WHERE idcalle = idc LOOP  INSERT INTO historico.tramo VALUES(p.gid, idc, p.geom, versiontramo, now(), src, src\_type, usr, op);  END LOOP;  return versiontramo;  END IF;  return 0; END; $BODY$  LANGUAGE plpgsql VOLATILE  COST 100;     CREATE OR REPLACE FUNCTION historico.log\_solar(idsolar integer, src integer, src\_type integer, usr varchar, op varchar)  RETURNS integer AS $BODY$ DECLARE  versionsolar integer;  p budu.solar%ROWTYPE;  nrows integer; BEGIN  IF op = 'FIRST' THEN  SELECT count(\*) INTO nrows FROM historico.solar WHERE apis\_id = idsolar;  IF nrows = 0 THEN  SELECT \* INTO p FROM budu.solar WHERE apis\_id = idsolar;  INSERT INTO historico.solar VALUES(idsolar, p.padron, p.solar, p.manzana, p.loc\_cat, p.depto, p.loc\_cdp, p.loc\_ine\_11, p.multipolygon, p.manzana\_id, 1, now(), NULL, NULL, NULL, 'FIRST');  return 1;  END IF;  ELSE  SELECT MAX(version) INTO versionsolar FROM historico.solar WHERE apis\_id = idsolar;  IF versionsolar IS NULL THEN  versionsolar := 0;  END IF;  versionsolar := versionsolar + 1;  SELECT \* INTO p FROM budu.solar WHERE apis\_id = idsolar;  INSERT INTO historico.solar VALUES(idsolar, p.padron, p.solar, p.manzana, p.loc\_cat, p.depto, p.loc\_cdp, p.loc\_ine\_11, p.multipolygon, p.manzana\_id, versionsolar, now(), src, src\_type, usr, op);  return versionsolar;  END IF;  return 0; END; $BODY$  LANGUAGE plpgsql VOLATILE  COST 100;       CREATE OR REPLACE FUNCTION historico.log\_solarp(idsolar integer, src integer, src\_type integer, usr varchar, op varchar)  RETURNS integer AS $BODY$ DECLARE  versionsolar integer;  p budu.solarp%ROWTYPE;  nrows integer; BEGIN  IF op = 'FIRST' THEN  SELECT count(\*) INTO nrows FROM historico.solarp WHERE solar\_id = idsolar;  IF nrows = 0 THEN  FOR p IN SELECT \* FROM budu.solarp WHERE solar\_id = idsolar LOOP  RAISE NOTICE 'ok %s', p.id;  INSERT INTO historico.solarp VALUES(p.id, p.numero, p.cp, p.numero, p.letra, p.km, p.punto, p.padron, idsolar, 1, now(), NULL, NULL, NULL, 'FIRST');  END LOOP;  return 1;  END IF;  ELSE  SELECT MAX(version) INTO versionsolar FROM historico.solarp WHERE solar\_id = idsolar;  IF versionsolar IS NULL THEN  versionsolar := 0;  END IF;  versionsolar := versionsolar + 1;  FOR p IN SELECT \* FROM budu.solarp WHERE solar\_id = idsolar LOOP  INSERT INTO historico.solarp VALUES(p.id, p.idcalle, p.cp, p.numero, p.letra, p.km, p.punto, p.padron, idsolar, versionsolar, now(), src, src\_type, usr, op);  END LOOP;  return versionsolar;  END IF;  return 0; END; $BODY$  LANGUAGE plpgsql VOLATILE  COST 100;         CREATE OR REPLACE FUNCTION historico.log\_ruta(idruta integer, src integer, src\_type integer, usr varchar, op varchar)  RETURNS integer AS $BODY$ DECLARE  versionruta integer;  p budu.ruta%ROWTYPE;  nrows integer; BEGIN  IF op = 'FIRST' THEN  SELECT count(\*) INTO nrows FROM historico.ruta WHERE id = idruta;  IF nrows = 0 THEN  SELECT \* INTO p FROM budu.ruta WHERE id = idruta;  INSERT INTO historico.ruta VALUES(idruta, p.numero, p.nombre, p.longitud, p.origen, p.destino, p.tipo, p.tipo\_anexo, p.tipo\_obs, p.geom, 'FIRST', 1, now(), NULL, NULL, NULL);  return 1;  END IF;  ELSE   SELECT MAX(version) INTO versionruta FROM historico.ruta WHERE id = idruta;  IF versionruta IS NULL THEN  versionruta := 0;  END IF;  versionruta := versionruta + 1;  SELECT \* INTO p FROM budu.ruta WHERE id = idruta;  INSERT INTO historico.ruta VALUES(idruta, p.numero, p.nombre, p.longitud, p.origen, p.destino, p.tipo, p.tipo\_anexo, p.tipo\_obs, p.geom, op, versionruta, now(), src, src\_type, usr);  return versionruta;  END IF;  return 0; END; $BODY$  LANGUAGE plpgsql VOLATILE  COST 100; 2.7 Procedimiento para inserción en alias CREATE OR REPLACE FUNCTION historico.log\_alias()  RETURNS trigger AS  $BODY$  DECLARE  forma\_fuente INTEGER;  BEGIN  IF (TG\_OP = 'INSERT') THEN  forma\_fuente = 1;  END IF;  IF (TG\_OP = 'UPDATE') THEN  forma\_fuente = 2;  END IF;      IF (TG\_OP = 'DELETE' OR TG\_OP = 'UPDATE') THEN  IF TG\_ARGV[0]::INTEGER = 0 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.idcalle, OLD.nombre, OLD.tipo\_alias\_id, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 1 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.departamento\_id, OLD.nombre, NULL, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 2 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.entidad\_colectiva\_id, OLD.nombre, NULL, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 3 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.localidad\_id, OLD.nombre, OLD.tipo\_alias\_id, OLD.publicable, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 4 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.id\_manzana, OLD.nombre, NULL, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 5 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.idpunto, OLD.numero::VARCHAR, NULL, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 6 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.id\_punto\_notable, OLD.nombre, NULL, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente1, USER);  ELSIF TG\_ARGV[0]::INTEGER = 7 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.id\_solarp, OLD.nombre, NULL, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 8 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, OLD.tramo\_ruta\_id, OLD.nombre, NULL, NULL, TG\_OP, now(), OLD.fuente\_id, forma\_fuente, USER);  END IF;  END IF;    IF (TG\_OP = 'INSERT' OR TG\_OP = 'UPDATE') THEN  IF TG\_ARGV[0]::INTEGER = 0 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.idcalle, NEW.nombre, NEW.tipo\_alias\_id, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 1 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.departamento\_id, NEW.nombre, NULL, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 2 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.entidad\_colectiva\_id, NEW.nombre, NULL, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 3 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.localidad\_id, NEW.nombre, NEW.tipo\_alias\_id, NEW.publicable, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 4 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.id\_manzana, NEW.nombre, NULL, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 5 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.idpunto, NEW.numero::VARCHAR, NULL, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 6 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.id\_punto\_notable, NEW.nombre, NULL, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente1, USER);  ELSIF TG\_ARGV[0]::INTEGER = 7 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.id\_solarp, NEW.nombre, NULL, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  ELSIF TG\_ARGV[0]::INTEGER = 8 THEN  INSERT INTO historico.alias VALUES(DEFAULT, TG\_ARGV[0]::INTEGER, NEW.tramo\_ruta\_id, NEW.nombre, NULL, NULL, TG\_OP, now(), NEW.fuente\_id, forma\_fuente, USER);  END IF;  --RAISE NOTICE 'INSERT % % %', TG\_ARGV[0], TG\_ARGV[1], TG\_TABLE\_NAME;  END IF;  RETURN NULL;  END;  $BODY$  LANGUAGE plpgsql VOLATILE  COST 100; |

## 2.7 Triggers para la actualización e inserción

|  |
| --- |
| CREATE TRIGGER historico\_alias\_calle  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_calle  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(0);    CREATE TRIGGER historico\_alias\_departamento  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_departamento  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(1);    CREATE TRIGGER historico\_alias\_entidad\_colectiva  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_entidad\_colectiva  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(2);    CREATE TRIGGER historico\_alias\_localidad\_geo  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_localidad\_geo  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(3);    CREATE TRIGGER historico\_alias\_manzana  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_manzana  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(4);    CREATE TRIGGER historico\_alias\_numero\_puerta  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_numero\_puerta  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(5);    CREATE TRIGGER historico\_alias\_punto\_notable  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_punto\_notable  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(6);    CREATE TRIGGER historico\_alias\_solarp  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_solarp  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(7);    CREATE TRIGGER historico\_alias\_tramo\_ruta  AFTER INSERT OR UPDATE OR DELETE  ON budu.alias\_tramo\_ruta  FOR EACH ROW  EXECUTE PROCEDURE historico.log\_alias(8); |